Zapoznanie się z Xcode

Mam nadzieję, że uznasz to za przydatne wprowadzenie do tworzenia i publikowania aplikacji na iOS 16 w App Store. Tu dowiesz się, jak pobrać i zainstalować Xcode na komputerze Mac. Zapoznasz się z różnymi częściami interfejsu użytkownika Xcode, utworzysz swoją pierwszą aplikację na iOS i uruchomisz ją w symulatorze iOS. Następnie dowiesz się, jak podłączyć urządzenie iOS do Xcode przez USB, aby móc na nim uruchomić aplikację, jak dodać Apple ID do Xcode, aby można było utworzyć i zainstalować niezbędne certyfikaty cyfrowe na swoim urządzeniu oraz jak ufać certyfikat na swoim urządzeniu. Na koniec dowiesz się, jak połączyć się z urządzeniem przez Wi-Fi, dzięki czemu nie musisz już podłączać urządzenia za każdym razem, gdy chcesz uruchomić aplikację. Pod koniec tego rozdziału będziesz wiedział, jak tworzyć i uruchamiać aplikacje na symulatorze lub urządzeniu iOS, co będzie potrzebne podczas tworzenia własnych aplikacji. W tej części zostaną poruszone następujące tematy:

• Pobieranie i instalowanie Xcode z App Store

• Zrozumienie interfejsu użytkownika Xcode

• Uruchomienie aplikacji w symulatorze iOS

• Używanie urządzenia iOS do programowania

Wymagania techniczne

Do wykonania ćwiczeń potrzebne będą:

• Komputer Apple Mac (Apple Silicon lub Intel) z systemem macOS 12 Monterey lub macOS 13 Ventura

• Apple ID

• Opcjonalnie urządzenie iOS z systemem iOS 16

Pobieranie i instalowanie Xcode ze sklepu App Store

Xcode to zintegrowane środowisko programistyczne (IDE) firmy Apple do tworzenia aplikacji na iOS. Zanim zaczniesz pisać swoją pierwszą aplikację na iOS, musisz pobrać i zainstalować Xcode ze sklepu App Store na komputerze Mac. Aby to zrobić, wykonaj następujące kroki:

1. Wybierz App Store z menu Apple.

2. W polu wyszukiwania w prawym górnym rogu wpisz Xcode i naciśnij klawisz Return.

3. Powinieneś zobaczyć Xcode w wynikach wyszukiwania. Kliknij Pobierz, a następnie Zainstaluj.

4. Jeśli masz Apple ID, wpisz go w polu tekstowym Apple ID. Jeśli go nie masz, kliknij przycisk Utwórz Apple ID i postępuj zgodnie ze szczegółowymi instrukcjami, aby go utworzyć

5. Po zainstalowaniu Xcode uruchom go. Powinieneś zobaczyć ekran Umowy licencyjnej. Kliknij Zgadzam się

6. Zostaniesz poproszony o podanie nazwy użytkownika i hasła administratora komputera Mac. Gdy już to zrobisz, kliknij OK

7. Zobaczysz ekran pokazujący dostępne platformy programistyczne. Na razie potrzebujesz tylko komputera Mac i systemu iOS. Pozostaw ustawienia na wartości domyślne i kliknij Zainstaluj

8. Jeśli używasz komputera Apple Silicon Mac i nie zainstalowałeś jeszcze aplikacji Rosetta, która umożliwia uruchamianie aplikacji Mac z procesorem Intel na komputerach Apple Silicon Mac, zostaniesz poproszony o jej teraz zainstalowanie. Kliknij Zainstaluj

9. Powinieneś zobaczyć następujący ekran Witamy w Xcode. Kliknij opcję Utwórz nowy projekt Xcode w lewym panelu

10. Zobaczysz ekran nowego projektu w następujący sposób. W sekcji Wybierz szablon dla swojego nowego projektu: wybierz iOS. Następnie wybierz Aplikacja i kliknij Dalej

11. Zobaczysz ekran Wybierz opcje dla swojego nowego projektu:

Skonfiguruj opcje pokazane na poprzednim zrzucie ekranu w następujący sposób:

• Nazwa produktu: nazwa Twojej aplikacji. Wpisz ExploringXcode w polu tekstowym.

• Identyfikator organizacji: używany do tworzenia unikalnego identyfikatora aplikacji w App Store. Na razie wpisz com.yourname. Nazywa się to formatem odwrotnej notacji nazwy domeny i jest powszechnie używany przez programistów iOS.

• Interfejs: Metoda używana do tworzenia interfejsu użytkownika aplikacji. Ustaw to na Storyboard.

• Język: używany język programowania. Ustaw to na Swift.

Pozostaw pozostałe ustawienia jako wartości domyślne i upewnij się, że wszystkie pola wyboru są odznaczone. Po zakończeniu kliknij Dalej.

12. Zobaczysz okno dialogowe Zapisz. Wybierz lokalizację, w której chcesz zapisać projekt, na przykład folder Pulpit lub Dokumenty, i kliknij Utwórz

13. Zobaczysz okno dialogowe z informacją, że utworzenie repozytorium Git nie powiodło się, kliknij Napraw.

14. Zobaczysz ekran ustawień kontroli źródła

Wprowadź następujące informacje:

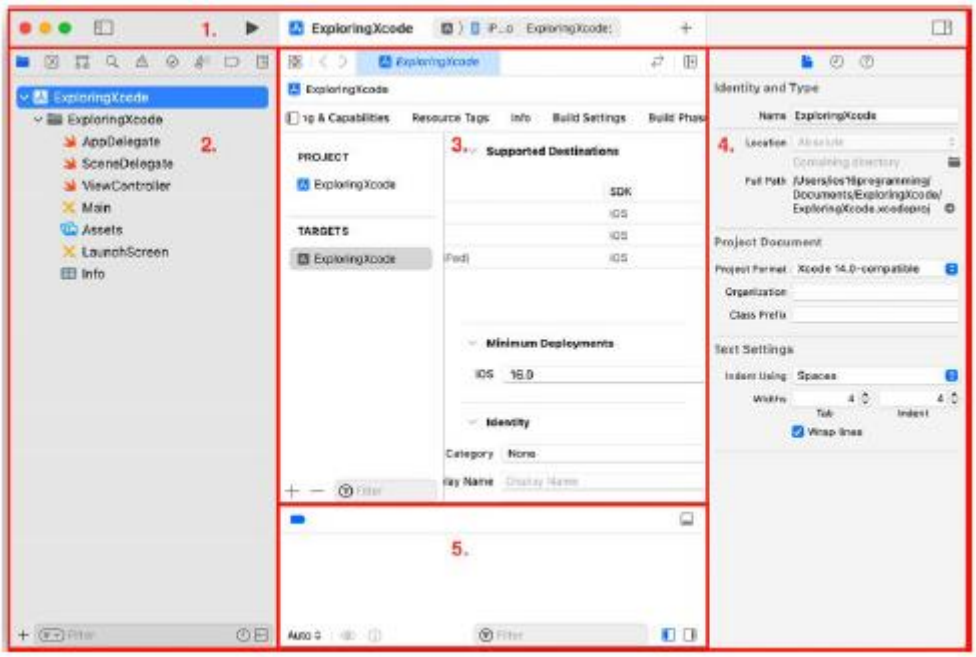
• Imię autora: Twoje własne imię i nazwisko

• E-mail autora: Twój adres e-mail

Po zakończeniu zamknij ekran ustawień Kontroli źródła, klikając czerwony przycisk Zamknij w lewym górnym rogu. Pojawi się główne okno Xcode. Fantastyczny! Pomyślnie pobrałeś i zainstalowałeś Xcode oraz utworzyłeś swój pierwszy projekt. W następnej sekcji poznasz interfejs użytkownika Xcode.

Zrozumienie interfejsu użytkownika Xcode

Właśnie stworzyłeś swój pierwszy projekt Xcode! Jak widać, interfejs użytkownika Xcode jest podzielony na kilka odrębnych części, jak pokazano tutaj:



Przyjrzyjmy się każdej części bardziej szczegółowo. Poniższy opis odpowiada numerom pokazanym na powyższym zrzucie ekranu:

• Pasek narzędzi (1) — służy do tworzenia i uruchamiania aplikacji oraz przeglądania postępu uruchomionych zadań.

• Obszar nawigatora (2) – zapewnia szybki dostęp do różnych części projektu. Domyślnie wyświetlany jest nawigator projektu.

• Obszar edytora (3) — umożliwia edycję kodu źródłowego, interfejsów użytkownika i innych zasobów.

• Obszar inspektora (4) – umożliwia przeglądanie i edycję informacji o elementach wybranych w obszarze nawigatora lub obszarze edytora.

• Obszar debugowania (5) — zawiera pasek debugowania, widok zmiennych i konsolę. Obszar debugowania można przełączać, wpisując Shift + Command + Y.

Następnie przyjrzyjmy się bliżej Paskowi narzędzi. Lewa strona paska narzędzi jest pokazana tutaj:

Przyjrzyjmy się każdej części bardziej szczegółowo. Poniższy opis odpowiada numerom pokazanym na powyższym zrzucie ekranu:

• Przycisk Nawigatora (1) – włącza i wyłącza obszar Nawigatora.

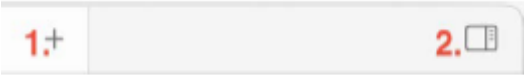
• Przycisk Stop (2) – pojawia się obok przycisku Odtwórz tylko wtedy, gdy aplikacja jest uruchomiona. Zatrzymuje aktualnie uruchomioną aplikację.

• Przycisk Odtwórz (3) — służy do tworzenia i uruchamiania aplikacji.

• Menu Schemat (4) — pokazuje konkretny schemat tworzenia projektu (ExploringXcode) i miejsce docelowe, na którym można uruchomić aplikację (iPhone SE (3. generacji)).

Schematy i miejsca docelowe są różne. Schematy określają ustawienia dotyczące budowania i uruchamiania projektu. Miejsca docelowe określają lokalizacje instalacji aplikacji i istnieją dla urządzeń fizycznych i symulatorów.

• Widok aktywności (5) – Wyświetla postęp uruchomionych zadań. Prawa strona paska narzędzi jest pokazana tutaj:



Przyjrzyjmy się każdej części bardziej szczegółowo. Poniższy opis odpowiada numerom pokazanym na powyższym zrzucie ekranu:

• Przycisk Biblioteka (1) — wyświetla elementy interfejsu użytkownika, fragmenty kodu i inne zasoby.

• Przycisk Inspektora (2) — włącza i wyłącza obszar Inspektora.

Nie daj się przytłoczyć różnymi częściami, ponieważ dowiesz się o nich bardziej szczegółowo w późniejszych częściach. Teraz, gdy znasz już interfejs Xcode, uruchomisz właśnie utworzoną aplikację w symulatorze iOS, który wyświetli reprezentację Twojego urządzenia iOS.

Uruchamianie aplikacji w symulatorze iOS

Symulator systemu iOS jest instalowany podczas instalacji Xcode. Zapewnia symulowane urządzenie z systemem iOS, dzięki czemu możesz zobaczyć, jak wygląda Twoja aplikacja i jak się zachowuje, bez konieczności posiadania fizycznego urządzenia z systemem iOS. Może modelować wszystkie rozmiary i rozdzielczości ekranów zarówno dla iPada, jak i iPhone'a, dzięki czemu możesz łatwo testować swoją aplikację na wielu urządzeniach. Aby uruchomić aplikację w symulatorze, wykonaj następujące kroki:

1. Kliknij menu Schemat na pasku narzędzi, a zobaczysz listę symulatorów. Z tego menu wybierz iPhone′a SE (3. generacji).

2. Kliknij przycisk Odtwórz, aby zainstalować i uruchomić aplikację na aktualnie wybranym symulatorze. Możesz także użyć skrótu klawiaturowego Command + R.

3. Symulator uruchomi się i wyświetli reprezentację iPhone′a SE (3. generacji). Twoja aplikacja wyświetla biały ekran, ponieważ nie dodałeś jeszcze niczego do swojego projektu

4. Wróć do Xcode i kliknij przycisk Stop (lub naciśnij Command + .), aby zatrzymać aktualnie uruchomiony projekt.

Właśnie utworzyłeś i uruchomiłeś w symulatorze swoją pierwszą aplikację na iOS! Dobra robota! Jeśli spojrzysz na menu Schemat, możesz się zastanawiać, do czego służą sekcje Urządzenia i Kompilacja. Przyjrzyjmy się im w następnej sekcji.

Zrozumienie sekcji Urządzenia i Budowa

W poprzedniej sekcji dowiedziałeś się, jak wybrać symulator w menu Schemat, aby uruchomić aplikację. Umożliwia uruchamianie aplikacji na rzeczywistych urządzeniach z systemem iOS i przygotowywanie aplikacji do przesłania do App Store. Kliknij menu Schemat na pasku narzędzi, aby wyświetlić sekcje Urządzenia i Kompilacja u góry menu. Jeśli masz komputer Mac Apple Silicon, w sekcji Urządzenia zostanie wyświetlony tekst Mój Mac (zaprojektowany dla iPada), ponieważ na komputerach Apple Silicon Mac można uruchamiać aplikacje dla systemu iOS. W przeciwnym razie zostanie wyświetlony komunikat Brak urządzeń. Jeśli podłączysz urządzenie z systemem iOS, pojawi się

ono w tej sekcji i będziesz mógł uruchamiać opracowane na nim aplikacje w celu przetestowania. Zaleca się uruchamianie aplikacji na rzeczywistym urządzeniu, ponieważ symulator nie będzie dokładnie odzwierciedlał charakterystyki wydajności rzeczywistego urządzenia z systemem iOS i nie ma niektórych funkcji sprzętowych ani interfejsów API oprogramowania, które mają rzeczywiste urządzenia. Sekcja Kompilacja zawiera tylko jedną pozycję menu: Dowolne urządzenie iOS. Jest to używane, gdy musisz zarchiwizować aplikację przed przesłaniem jej do App Store. Dowiesz się, jak to zrobić w rozdziale 27, Testowanie i przesyłanie aplikacji do App Store. Zobaczmy teraz, jak zbudować i uruchomić aplikację na prawdziwym urządzeniu z systemem iOS. Zdecydowana większość instrukcji zawartych w tej książce nie wymaga jednak posiadania urządzenia z systemem iOS, więc jeśli go nie masz, możesz pominąć następną sekcję i przejść od razu do Rozdziału 2, Proste wartości i typy.

Używanie urządzenia iOS do programowania

Chociaż większość ćwiczeń będzie można wykonać za pomocą symulatora, zaleca się tworzenie i testowanie aplikacji na rzeczywistym urządzeniu z systemem iOS, ponieważ symulator nie będzie w stanie symulować niektórych komponentów sprzętowych ani interfejsów API oprogramowania . Oprócz urządzenia będziesz potrzebować Apple ID lub płatnego konta Apple Developer, aby zbudować i uruchomić aplikację na swoim urządzeniu. Na razie będziesz używać tego samego Apple ID, którego użyłeś do pobrania Xcode z App Store. Wykonaj następujące kroki:

1. Użyj kabla dostarczonego z urządzeniem iOS, aby podłączyć urządzenie do komputera Mac i upewnij się, że urządzenie iOS jest odblokowane.

2. Twoje urządzenie iOS wyświetli alert Zaufaj temu komputerowi. Kliknij Zaufaj i po wyświetleniu monitu wprowadź hasło urządzenia.

3. Twój Mac wyświetli powiadomienie Zezwól akcesorium na połączenie. Kliknij Zezwalaj. Twoje urządzenie iOS powinno być teraz podłączone do komputera Mac i pojawi się w menu Schemat Xcode.

Możesz przeglądać podłączone urządzenia wybierając Okno | Urządzenia i symulatory na pasku menu Xcode

4. Zwróć uwagę, że w menu Schemat Twoje urządzenie z systemem iOS nie ma włączonego trybu programisty:

Tryb programisty został wprowadzony przez firmę Apple podczas konferencji programistów World Wide Developer w 2022 r. (WWDC 2022) i jest wymagany do instalowania, uruchamiania i debugowania aplikacji na urządzeniach z systemem iOS 16.

5. Jeśli spróbujesz zbudować i uruchomić aplikację na urządzeniu z systemem iOS, zobaczysz następujący alert:

6. Aby włączyć tryb programisty na swoim urządzeniu iOS, przejdź do Ustawienia | Prywatność i bezpieczeństwo, przewiń w dół do elementu Tryb programisty i dotknij go.

7. Włącz przełącznik Tryb programisty:

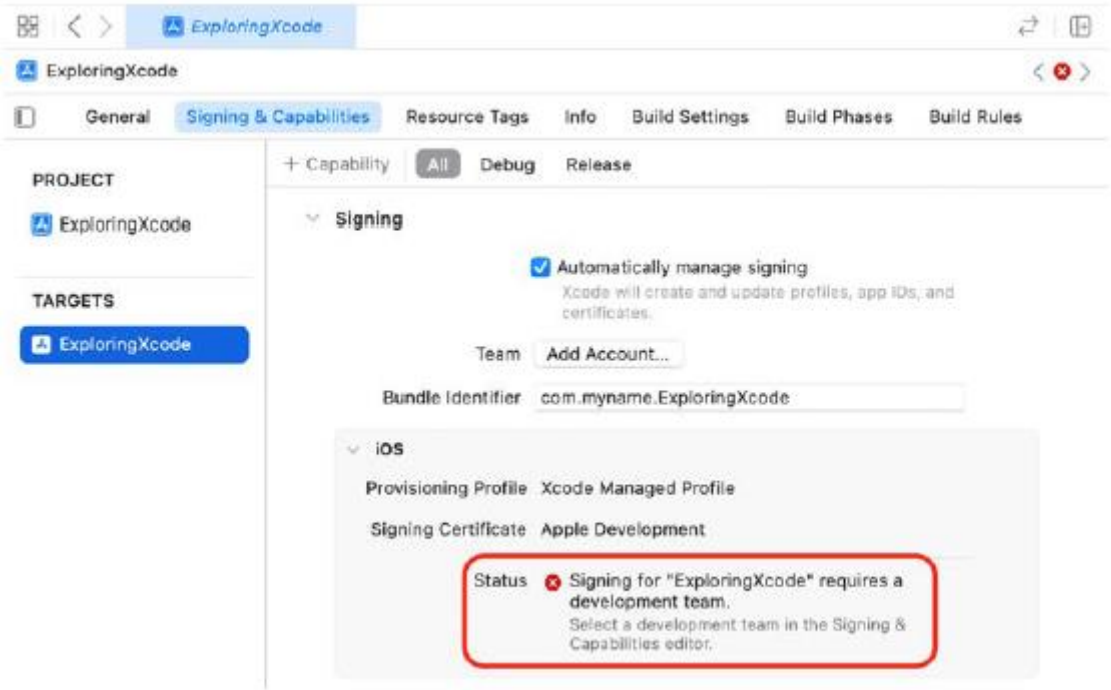
8. Pojawi się alert ostrzegający, że tryb programisty zmniejsza bezpieczeństwo Twojego urządzenia iOS. Stuknij przycisk Uruchom ponownie alertu.

9. Po ponownym uruchomieniu urządzenia iOS i jego odblokowaniu potwierdź, że chcesz włączyć tryb programisty, dotykając opcji Włącz i wprowadzając hasło urządzenia iOS.

10. Sprawdź, czy (Tryb programisty wyłączony) nie jest już wyświetlany obok urządzenia iOS w menu Schemat. Twoje urządzenie z systemem iOS jest teraz gotowe do instalowania i uruchamiania aplikacji z Xcode:

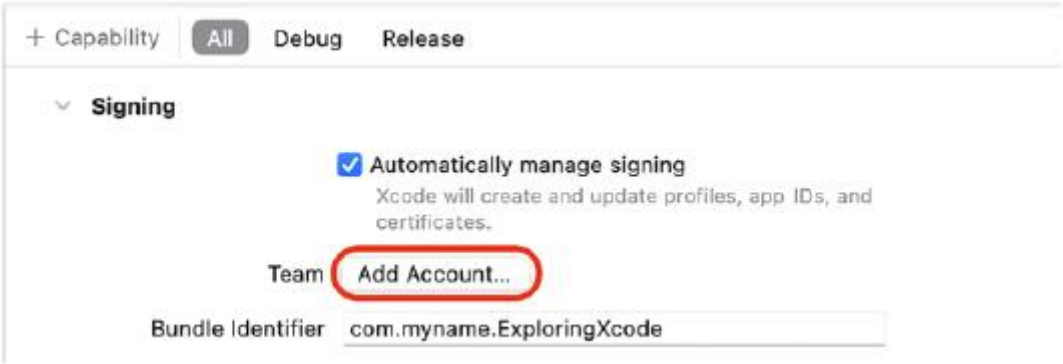
11. Poczekaj, aż Xcode zakończy indeksowanie i przetwarzanie, co zajmie trochę czasu. Po zakończeniu tej czynności w oknie stanu zostanie wyświetlony komunikat Gotowy

12. Uruchom projekt klikając przycisk Odtwórz (lub użyj Command + R). Pojawi się następujący błąd: Signing for “ExploringXcode” requires a development team:



Dzieje się tak, ponieważ do uruchomienia aplikacji na urządzeniu z systemem iOS wymagany jest certyfikat cyfrowy i musisz dodać identyfikator Apple ID lub płatne konto Apple Developer do Xcode, aby można było wygenerować certyfikat cyfrowy. Korzystanie z Apple ID umożliwi testowanie aplikacji na urządzeniu z systemem iOS, ale do rozpowszechniania aplikacji w App Store potrzebne będzie płatne konto Apple Developer. Certyfikaty zapewniają, że na Twoim urządzeniu działają tylko te aplikacje, które autoryzowałeś. Pomaga to chronić przed złośliwym oprogramowaniem

13. Kliknij przycisk Dodaj konto:

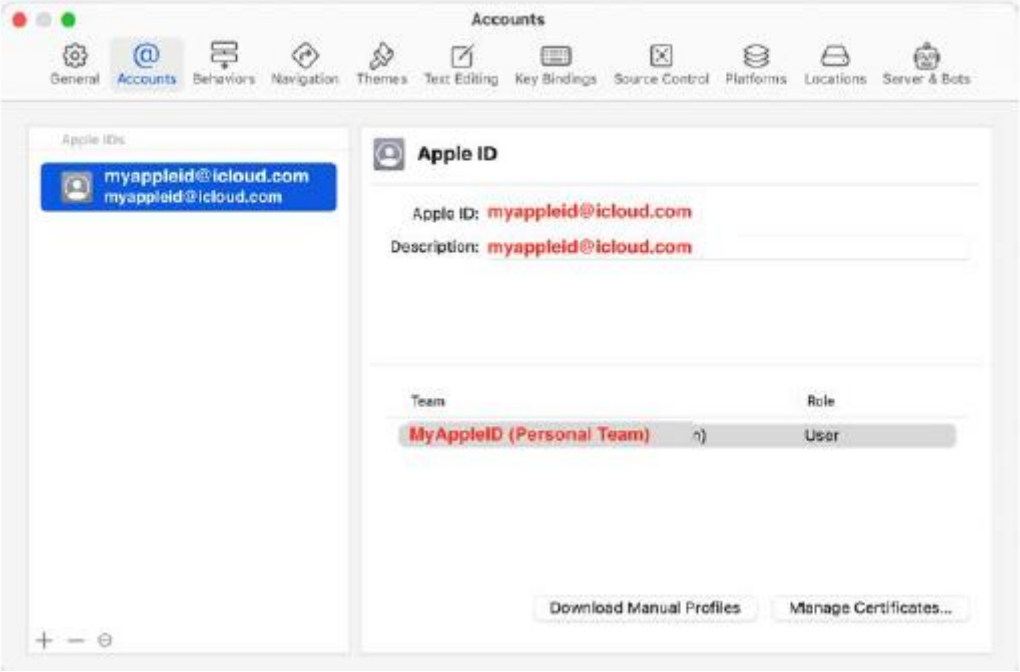


14. Pojawi się okno Ustawienia Xcode z wybranym panelem Konta. Wprowadź swój Apple ID i kliknij Dalej:

Pamiętaj, że możesz utworzyć inny identyfikator Apple ID, jeśli chcesz, korzystając z przycisku Utwórz Apple ID. Dostęp do ustawień Xcode można także uzyskać, wybierając opcję Ustawienia w menu Xcode.

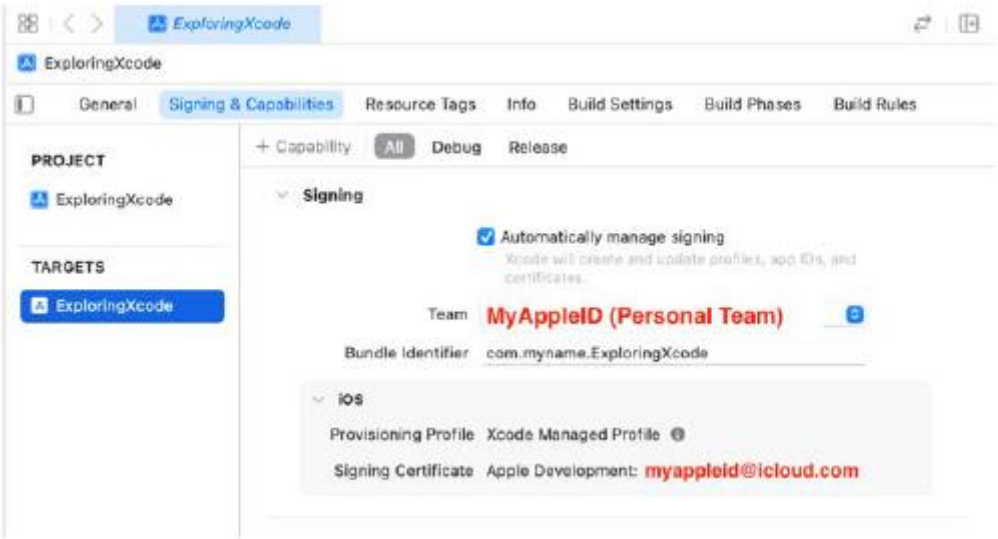
15. Po wyświetleniu monitu wprowadź hasło. Po kilku minutach w panelu Konta zostaną wyświetlone ustawienia Twojego konta:

<img src="images/ios10.png">



16. Po zakończeniu zamknij okno Ustawienia, klikając czerwony przycisk Zamknij w lewym górnym rogu.

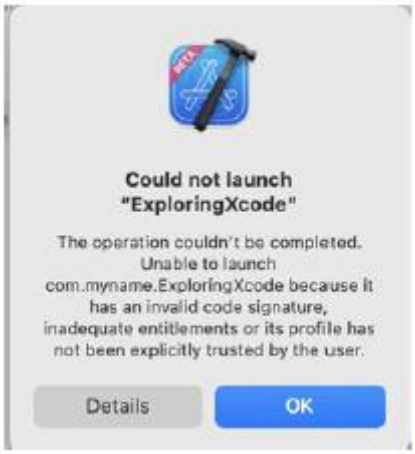
17. W obszarze Edytora Xcode kliknij Podpisywanie i możliwości. Upewnij się, że jest zaznaczona opcja Automatycznie zarządzaj podpisywaniem i wybierz opcję Zespół osobisty z wyskakującego menu Zespół:



18. Jeśli nadal widzisz błędy na tym ekranie, spróbuj zmienić identyfikator pakietu, wpisując w niego kilka losowych znaków, na przykład com.myname4352.ExploringXcode.

19. Po zbudowaniu i uruchomieniu wszystko powinno działać, a Twoja aplikacja zostanie zainstalowana na Twoim urządzeniu z systemem iOS. Jednak nie uruchomi się i zobaczysz następujący komunikat:

<img src="images/ios12.png">



Oznacza to, że musisz zaufać certyfikatowi zainstalowanemu na Twoim urządzeniu. Dowiesz się, jak to zrobić w następnej sekcji.

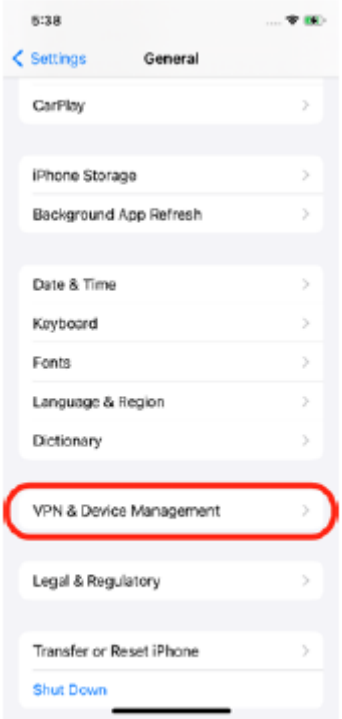
Zaufanie certyfikatowi aplikacji deweloperskiej na urządzeniu z systemem iOS

Certyfikat aplikacji dla programistów to specjalny plik instalowany na urządzeniu z systemem iOS wraz z aplikacją.

Zanim Twoja aplikacja będzie mogła działać, musisz jej zaufać. Wykonaj następujące kroki:

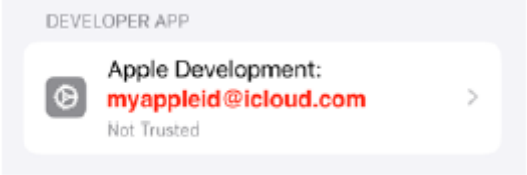
1. Na urządzeniu z systemem iOS dotknij Ustawienia| Ogólne | Zarządzanie VPN i urządzeniami:

<img src="images/ios13.png">

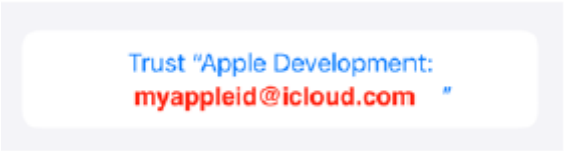


2. Stuknij opcję Apple Development:

<img src="images/ios14.png">



3. Stuknij Trust "Apple Development":



4. Stuknij Trust:

<img src="images/ios16.png">

5. Powinieneś zobaczyć następujący tekst, który oznacza, że aplikacja jest teraz zaufana:

<img src="images/ios17.png">

6. Kliknij przycisk Odtwórz w Xcode, aby skompilować i uruchomić ponownie. Zobaczysz, jak aplikacja uruchamia się i działa na urządzeniu z systemem iOS.

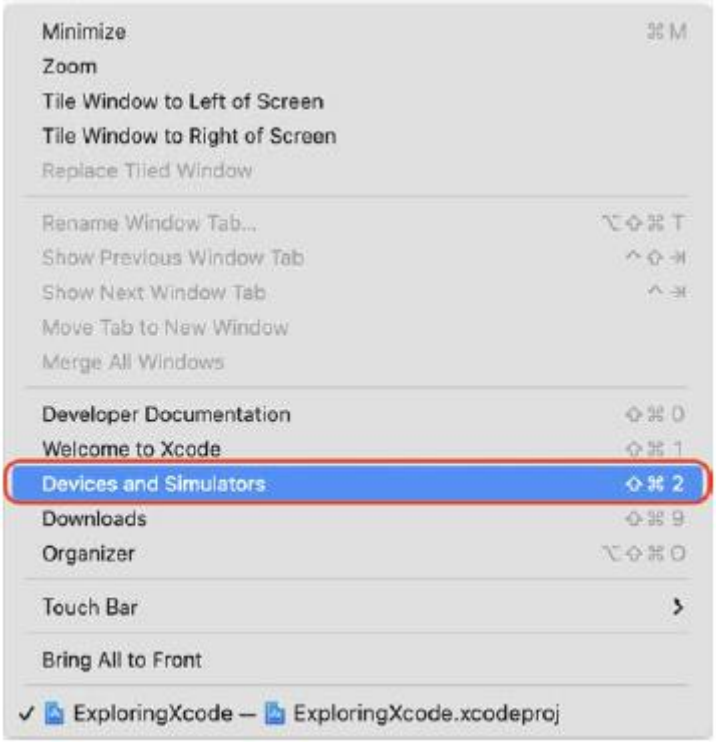
Gratulacje! Pamiętaj, że aby skompilować i uruchomić aplikację, musisz podłączyć urządzenie iOS do komputera Mac za pomocą kabla. W następnej sekcji dowiesz się, jak połączyć się z urządzeniem przez Wi-Fi.

Podłączanie urządzenia iOS bezprzewodowo

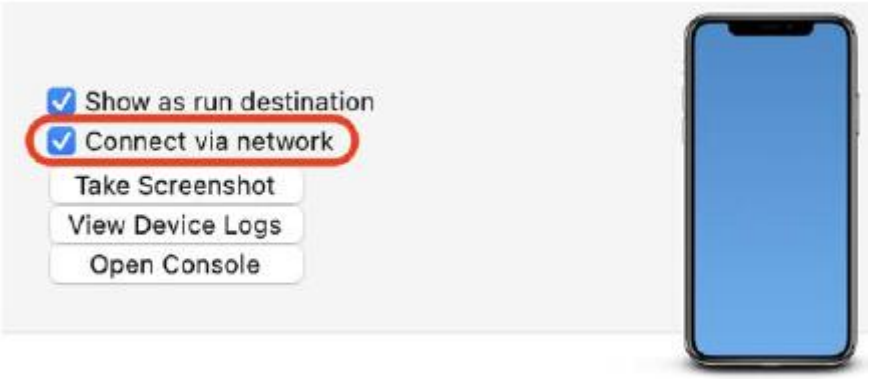
Odłączanie i ponowne podłączanie urządzenia iOS do komputera Mac może po pewnym czasie stać się dość kłopotliwe, dlatego teraz skonfigurujesz Xcode tak, aby łączył się z urządzeniem iOS przez Wi-Fi. Wykonaj następujące kroki:

1. Upewnij się, że urządzenie iOS jest podłączone do komputera Mac oraz że zarówno Mac, jak i urządzenie iOS znajdują się w tej samej sieci bezprzewodowej.

2. Wybierz Okno | Urządzenia i symulatory z paska menu Xcode:



3. Kliknij pole wyboru Connect via network:



Wspaniały! Twoje urządzenie iOS jest teraz połączone bezprzewodowo z Xcode i nie potrzebujesz już kabla USB, aby go podłączyć.

Streszczenie

Nauczyłeś się, jak pobrać i zainstalować Xcode na komputerze Mac. Zapoznałeś się z różnymi częściami interfejsu użytkownika Xcode. Utworzyłeś swoją pierwszą aplikację na iOS, wybrałeś symulator, zbudowałeś i uruchomiłeś aplikację. Dowiedziałeś się, do czego służą pozycje menu Urządzenia i Kompilacja. Możesz teraz tworzyć i uruchamiać aplikacje na iOS na komputerze Mac bez konieczności posiadania urządzenia z systemem iOS. Nauczyłeś się, jak podłączyć urządzenie iOS do Xcode przez USB, aby móc uruchomić na nim aplikację. Dodałeś identyfikator Apple ID do Xcode, aby można było utworzyć i zainstalować niezbędne certyfikaty cyfrowe na swoim urządzeniu oraz zaufać certyfikatowi na swoim urządzeniu. Dzięki temu możesz uruchamiać aplikacje na rzeczywistym urządzeniu, dzięki

czemu możesz dokładniej określić ich wydajność i skorzystać z funkcji niedostępnych w symulatorze iOS. Wreszcie nauczyłeś się, jak łączyć się z urządzeniem przez Wi-Fi, więc nie musisz już podłączać urządzenia za każdym razem, gdy chcesz uruchomić aplikację. Dzięki temu tworzenie i testowanie aplikacji na urządzeniu z systemem iOS jest znacznie wygodniejsze, ponieważ wszelkie nowe kompilacje można natychmiast przesyłać bezprzewodowo.

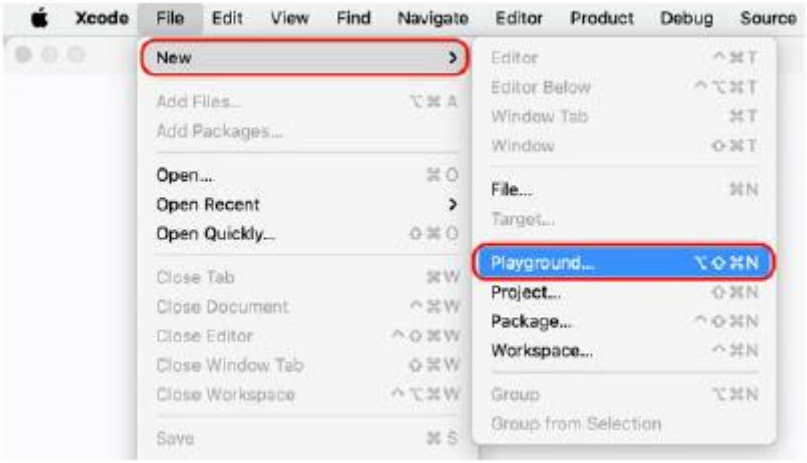
Proste wartości i typy

Teraz, gdy odbyłeś już krótką wycieczkę po Xcode, przyjrzyjmy się językowi programowania Swift. Najpierw poznasz place zabaw Swift, interaktywne środowisko, w którym możesz wpisać kod Swift i natychmiast wyświetlić wyniki. Następnie dowiesz się, jak Swift reprezentuje i przechowuje różne typy danych. Następnie zapoznasz się z kilkoma fajnymi funkcjami Swifta, takimi jak wnioskowanie o typie i bezpieczeństwo typów, które pomogą Ci pisać kod w sposób bardziej zwięzły i unikać typowych błędów. Na koniec dowiesz się, jak wykonywać typowe operacje na danych i jak drukować komunikaty w obszarze debugowania, aby pomóc w rozwiązywaniu problemów. Pod koniec tej części powinieneś umieć pisać proste programy przechowujące i przetwarzające litery i cyfry.

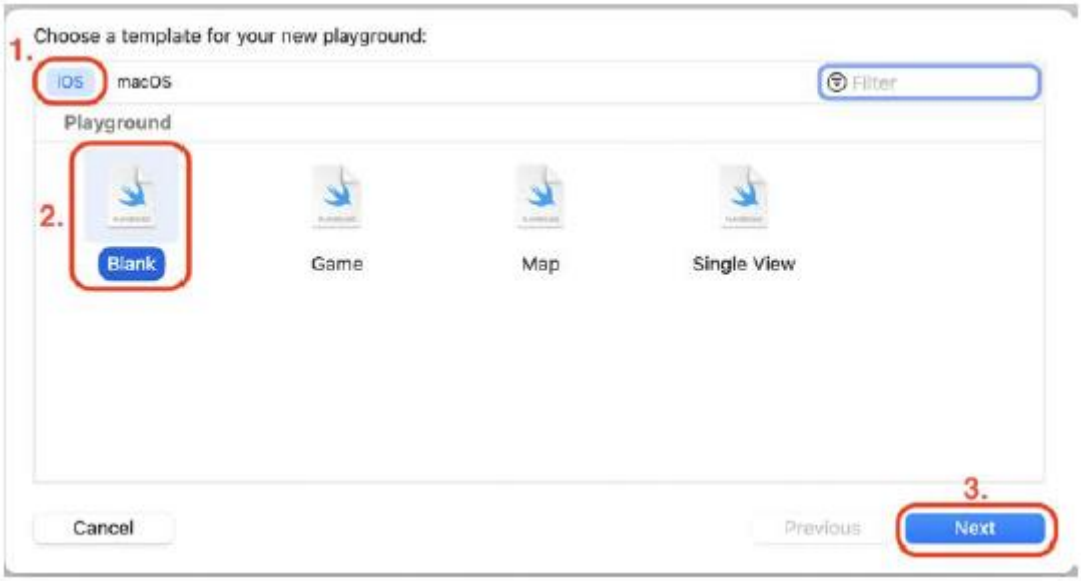
Zrozumienie placów zabaw Swift

Place zabaw to interaktywne środowiska kodowania. Wpisujesz kod w lewym panelu, a wyniki są natychmiast wyświetlane w prawym panelu. To świetny sposób na eksperymentowanie z kodem i odkrywanie systemowych interfejsów API. Zacznijmy od stworzenia nowego placu zabaw i sprawdzenia jego interfejsu użytkownika. Wykonaj następujące kroki:

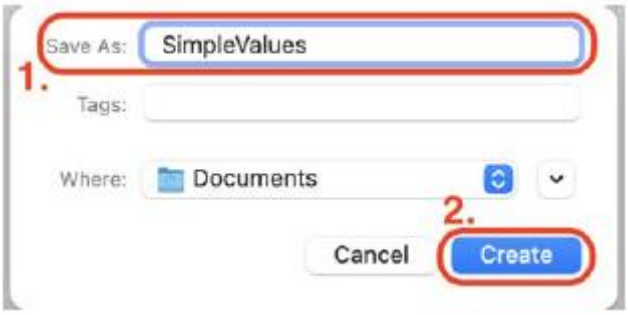
1. Aby utworzyć plac zabaw, uruchom Xcode i wybierz Plik | Nowy | Plac zabaw... z paska menu Xcode:



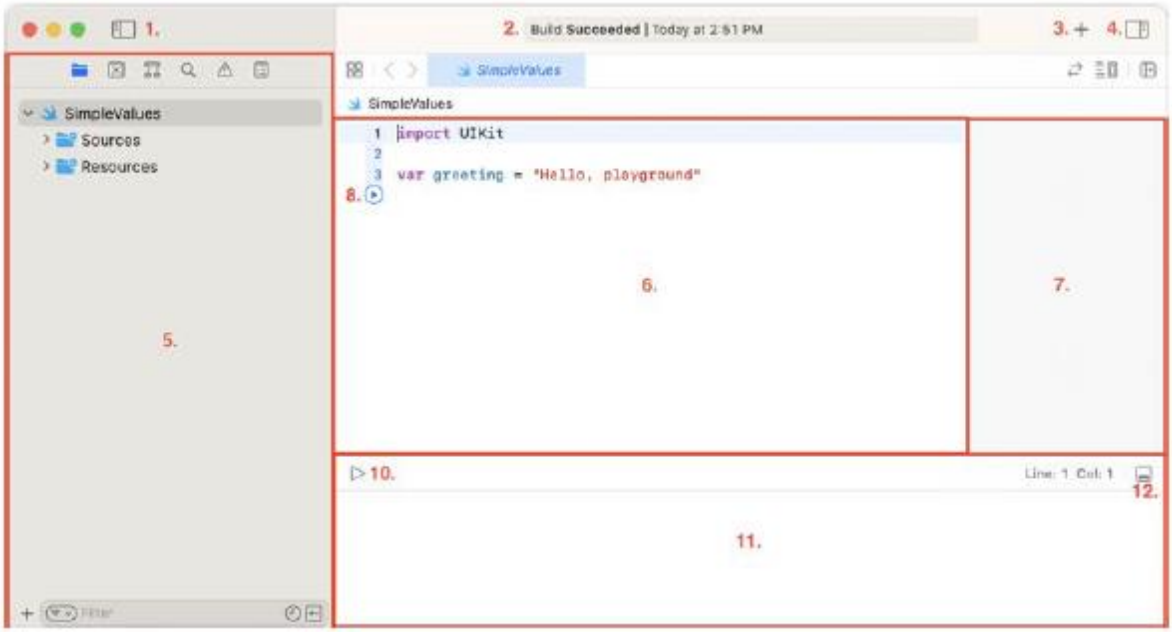
2. Pojawi się ekran szablonu. iOS powinien być już wybrany. Wybierz opcję Puste i kliknij Dalej:



3. Nazwij swój plac zabaw SimpleValues i zapisz go w dowolnym miejscu. Po zakończeniu kliknij Utwórz:



4. Na ekranie powinieneś zobaczyć plac zabaw:



Jak widać, jest to znacznie prostsze niż projekt Xcode. Przyjrzyjmy się interfejsowi bardziej szczegółowo:

• Przycisk Nawigatora (1) – Pokazuje lub ukrywa obszar Nawigatora

• Widok aktywności (2) – Pokazuje bieżącą operację lub stan

• Przycisk Biblioteka (3) – wyświetla fragmenty kodu i inne zasoby

• Przycisk Inspektora (4) – Pokazuje lub ukrywa obszar Inspektora

• Obszar nawigatora (5) – zapewnia szybki dostęp do różnych części projektu. Domyślnie wyświetlany jest nawigator projektu

• Obszar edytora (6) – tutaj piszesz kod

• Obszar wyników (7) — zapewnia natychmiastową informację zwrotną na temat napisanego kodu

• Przycisk Play (8) – Wykonuje kod z wybranej linii

• Obramowanie (9) — Ta ramka oddziela obszary Edytora i Wyniki. Jeśli stwierdzisz, że wyniki wyświetlane w obszarze Wyniki są obcięte, przeciągnij ramkę w lewo, aby zwiększyć jej rozmiar

• Przycisk Odtwórz/Zatrzymaj (10) – Wykonuje lub zatrzymuje wykonywanie całego kodu na placu zabaw

• Obszar debugowania (11) – Wyświetla wyniki polecenia print().

• Przycisk Debug (12) – Pokazuje i ukrywa obszar Debug

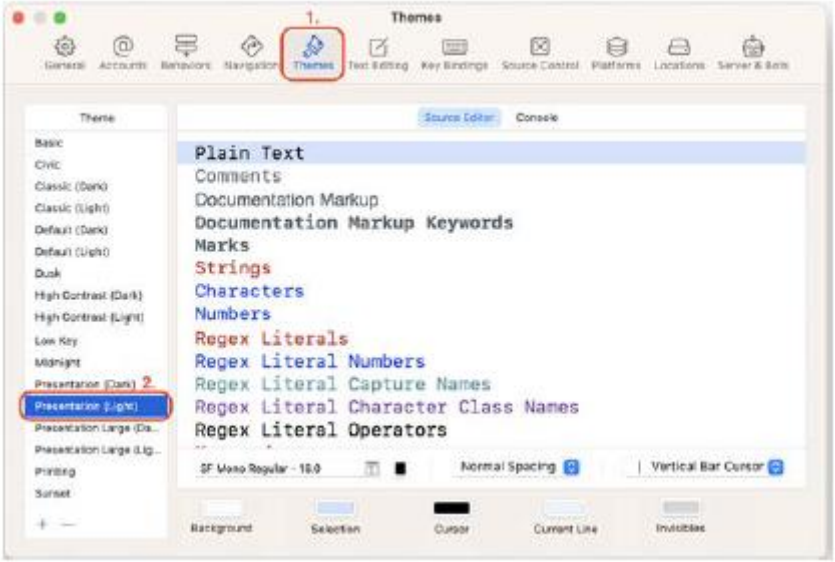
Kod na placu zabaw może okazać się zbyt mały i trudny do odczytania.

Dostosowywanie czcionek i kolorów

Xcode oferuje rozbudowane opcje dostosowywania. Dostęp do nich można uzyskać w menu Ustawienia Jeśli okaże się, że tekst jest mały i słabo widoczny, wykonaj następujące kroki:

1. Wybierz Ustawieniaz menu Xcode, aby wyświetlić okno preferencji.

2. W oknie ustawień kliknij Motywy i wybierz Prezentacja (Lekka), aby powiększyć kod i uczynić go łatwiejszym do odczytania:



3. Zamknij okno ustawień, aby powrócić do placu zabaw. Zwróć uwagę, że tekst na placu zabaw jest większy niż wcześniej. Jeśli chcesz, możesz także wypróbować inne motywy. Teraz, gdy już dostosowałeś czcionki i kolory do swoich upodobań, zobaczmy, jak uruchomić kod na placu zabaw w następnej sekcji.

Uruchamianie kodu placu zabaw

Twój plac zabaw zawiera już instrukcję. Aby wykonać instrukcję, wykonaj następujące kroki:

1. Kliknij przycisk Odtwórz/Zatrzymaj w lewym dolnym rogu placu zabaw. W obszarze wyników wyświetli się komunikat „Witaj, placu zabaw”:

Możesz użyć skrótu klawiaturowego Command + Shift + Return, aby uruchomić kod na swoim placu zabaw.

Aby przygotować plac zabaw do użycia w pozostałej części , usuń z placu zabaw instrukcję var pozdrowienia = "Witam, plac zabaw". W trakcie gry wpisz kod pokazany w tej części na placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić. W następnej sekcji zajmiemy się prostymi typami danych używanymi w Swift.

Eksploracja typów danych

Wszystkie języki programowania mogą przechowywać liczby, stany logiczne i słowa, a Swift nie jest wyjątkiem. Nawet jeśli jesteś doświadczonym programistą, może się okazać, że Swift reprezentuje te obiekty inaczej niż inne języki, które możesz znać. W następnych sekcjach omówimy wersje Swift liczb całkowitych, liczb zmiennoprzecinkowych, wartości logicznych i ciągów znaków.

Reprezentowanie liczb całkowitych

Załóżmy, że chcesz przechowywać następujące informacje:

• Liczba restauracji w mieście

• Pasażerowie samolotu

• Pokoje w hotelu

Można by użyć liczb całkowitych, czyli liczb bez elementu ułamkowego (w tym liczb ujemnych). Liczby całkowite w języku Swift są reprezentowane przez typ Int.

Reprezentowanie liczb zmiennoprzecinkowych

Załóżmy, że chcesz przechowywać następujące informacje:

• Pi (3.14159)

• Zero absolutne (-273,15°°C)

Można użyć liczb zmiennoprzecinkowych, które są liczbami ze składnikiem ułamkowym. Domyślnym typem liczb zmiennoprzecinkowych w języku Swift jest Double, który wykorzystuje 64 bity, w tym liczby ujemne. Możesz także użyć Float, który używa 32 bitów, ale Double jest reprezentacją domyślną.

Reprezentowanie wartości logicznych

Załóżmy, że chcesz przechowywać odpowiedzi na proste pytania typu „tak/nie”, takie jak następujące:

• Czy pada deszcz?

• Czy w restauracji są wolne miejsca?

W tym celu używasz wartości logicznych. Swift udostępnia typ Bool, który można przypisać jako prawdziwy lub fałszywy.

Reprezentowanie ciągów

Załóżmy, że chcesz przechowywać następujące informacje:

• Nazwa restauracji, np. „Bombay Palace”

• Opis stanowiska, np. „Księgowy” lub „Programista”

• Rodzaj owocu, np. „banan”

Można użyć typu String Swifta, który reprezentuje sekwencję znaków i jest w pełni zgodny z Unicode. Ułatwia to reprezentowanie różnych czcionek i języków. Teraz, gdy już wiesz, jak Swift reprezentuje te popularne typy danych, wypróbujmy je na placu zabaw, który utworzyłeś wcześniej w następnej sekcji.

Używanie popularnych typów danych na placu zabaw

Wszystko, co wpiszesz na placu zabaw, zostanie wykonane, a wyniki pojawią się w obszarze Wyniki. Zobaczmy, co się stanie, gdy wpiszesz liczby, wartości logiczne i ciągi znaków na swoim obszarze i wykonaj to. Wykonaj następujące kroki:

1. Wpisz następujący kod w obszarze edytora swojego placu zabaw:

// Proste wartości

42

-23

3.14159

0,1

-273,15

PRAWDA

FAŁSZ

"Witaj świecie"

"albatros"

Należy pamiętać, że komentarze nie są wyświetlane w obszarze Wyniki.

Fajnie! Właśnie stworzyłeś i uruchomiłeś swój pierwszy plac zabaw. Przyjrzyjmy się, jak przechowywać różne typy danych w następnej sekcji.

Badanie stałych i zmiennych

Teraz, gdy wiesz już o prostych typach danych obsługiwanych przez Swift, przyjrzyjmy się, jak je przechowywać, tzw. możesz później wykonywać na nich operacje. Do przechowywania danych można używać stałych lub zmiennych. Obydwa są kontenerami, które mają nazwę, ale stałą wartość można ustawić tylko raz i nie można jej zmienić po ich ustawieniu, podczas gdy wartość zmiennej można zostać zmieniona w dowolnym momencie. Musisz zadeklarować stałe i zmienne przed ich użyciem. Stałe deklaruje się za pomocą słowa kluczowego let , podczas gdy zmienne są deklarowane za pomocą słowa kluczowego var. Przyjrzyjmy się, jak działają stałe i zmienne, wdrażając je na swoim placu zabaw. Podążać za tymi krokami:

1. Dodaj następujący kod do swojego placu zabaw, aby zadeklarować trzy stałe:

let theAnswerToTheUltimateQuestion = 42

let pi = 3.14159

let myName = "Ahmad Sahar"

2. Kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić. W każdym przypadku tworzony jest kontener, nazwany i zapisana przypisana mu wartość.

Być może zauważyłeś, że nazwy stałych i zmiennych zaczynają się od małej litery, a jeśli w nazwie znajduje się więcej niż jedno słowo, każde kolejne słowo zaczyna się od dużej litery. Nazywa się to etui na wielbłąda. Nie musisz tego robić, ale jest to zalecane, ponieważ większość doświadczonych programistów Swift przestrzega tej konwencji. Należy zauważyć, że do przypisania wartości myName używana jest sekwencja znaków ujęta w podwójny cudzysłów „Ahmad Sahar”. Są one znane jako literały łańcuchowe.

3. Dodaj następujący kod po deklaracjach stałych, aby zadeklarować trzy zmienne:

var currentTemperatureInCelsius = 27

var myAge = 50

var myLocation = "home"

Podobnie jak w przypadku stałych, w każdym przypadku tworzony jest kontener i nazwany, a przypisana wartość jest przechowywana. Zapisane wartości zostaną wyświetlone w obszarze Wyniki.

4. Po ustawieniu wartości stałej nie można zmienić. Aby to przetestować, dodaj następujący kod po deklaracjach zmiennych:

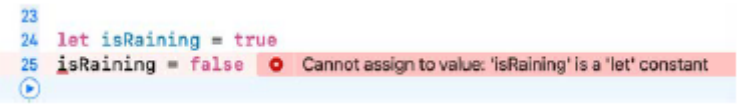
let isRaining = true

isRaining = false

Podczas wpisywania drugiej linii kodu pojawi się wyskakujące menu z sugestiami:

Użyj klawiszy strzałek w górę i w dół, aby wybrać stałą isRaining i naciśnij klawisz Tab, aby ją wybrać. Ta funkcja nazywa się autouzupełnianiem i pomaga zapobiegać błędom podczas wprowadzania kodu.

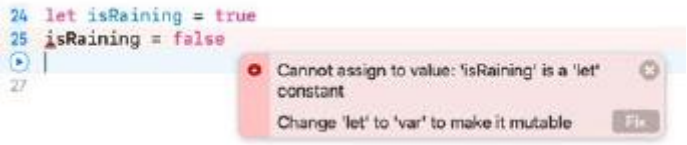
5. Po zakończeniu pisania poczekaj kilka sekund. W drugiej linii powinieneś zobaczyć czerwone kółko z białą kropką pośrodku:



Oznacza to, że w Twoim programie wystąpił błąd i Xcode uważa, że można go naprawić. Błąd pojawia się, ponieważ próbujesz przypisać nową wartość do stałej po ustawieniu jej wartości początkowej.

6. Kliknij czerwone kółko, aby rozwinąć komunikat o błędzie. Powinieneś zobaczyć następujące pole z przyciskiem Napraw:

<img src-=”images/ios28.png”>

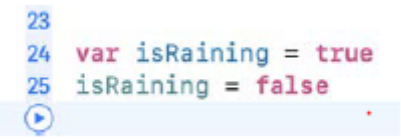


Xcode informuje Cię, na czym polega problem (nie można przypisać do wartości: „isRaining” jest stałą „let”) i sugeruje korektę (zmień „let” na „var”, aby można było go modyfikować).

7. Kliknij przycisk Napraw.

8. Powinieneś zobaczyć, że deklaracja stałej isRaining została zmieniona na deklarację zmiennej:

<img src-=”images/ios29.png”>



Ponieważ do zmiennej można przypisać nową wartość po jej utworzeniu, błąd został rozwiązany. Pamiętaj jednak, że sugerowana korekta może nie być najlepszym rozwiązaniem. Gdy zdobędziesz więcej doświadczenia w programowaniu na iOS, będziesz w stanie określić najlepszy sposób działania. Jeśli spojrzysz na wpisany kod, możesz się zastanawiać, skąd Xcode zna typ danych przechowywanych w zmiennej lub stałej. Jak to się robi, dowiesz się w następnej sekcji.

Zrozumienie wnioskowania o typie i bezpieczeństwa typu

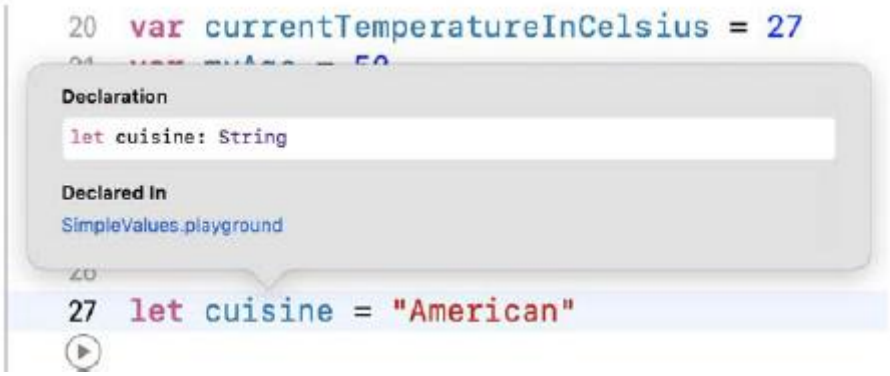
W poprzedniej sekcji zadeklarowałeś stałe i zmienne oraz przypisałeś im wartości. Swift automatycznie określa typ stałej lub zmiennej na podstawie podanej wartości. Nazywa się to wnioskowaniem o typie. Typ stałej lub zmiennej można zobaczyć, przytrzymując klawisz Opcja i klikając jej nazwę. Aby zobaczyć to w akcji, wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw, aby zadeklarować ciąg znaków:

let cuisine = "American"

2. Kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić.

3. Przytrzymaj klawisz Opcja i kliknij kuchnia, aby wyświetlić stały typ. Powinieneś zobaczyć następujący komunikat:



Jak widać, typ kuchni to String.

Co się stanie, jeśli chcesz ustawić konkretny typ zmiennej lub stałej? Zobaczysz, jak to zrobić w następnej sekcji.

Użycie adnotacji typu do określenia typu

Widziałeś, że Xcode próbuje automatycznie określić typ danych zmiennej lub stałej na podstawie podanej wartości. Czasami jednak możesz chcieć określić typ, zamiast pozwolić Xcode zrobić to za Ciebie. Aby to zrobić, wpisz dwukropek (:) po nazwie stałej lub zmiennej, a następnie żądany typ. Nazywa się to adnotacją typu.

Dodaj następujący kod do swojego placu zabaw, aby zadeklarować zmienną określonego typu, a następnie kliknij przycisk Odtwórz/Zatrzymaj, aby ją uruchomić:

var restaurantRating: Double = 3

Tutaj określono, że ocena restauracji ma określony typ, Double. Nawet jeśli do restauracjiRating przypisano liczbę całkowitą, będzie ona przechowywana jako liczba zmiennoprzecinkowa. W następnej

sekcji dowiesz się, jak Xcode pomaga zmniejszyć liczbę błędów w programie poprzez egzekwowanie bezpieczeństwa typów.

Korzystanie z zabezpieczeń typu do sprawdzania wartości

Swift jest językiem bezpiecznym dla typów. Sprawdza, czy przypisujesz wartości prawidłowego typu do zmiennych i oznacza niedopasowane typy jako błędy. Zobaczmy, jak to działa, wykonując następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw, aby przypisać ciąg do restauracjiRating:

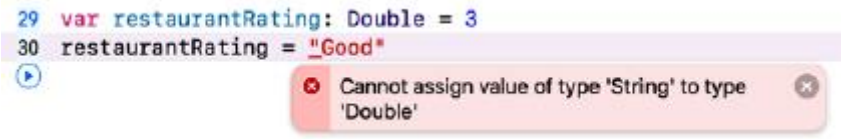
restaurantRating = "Good"

2. Kliknij przycisk Odtwórz/Zatrzymaj, aby uruchomić kod.

3. Powinieneś zobaczyć czerwone kółko z x w środku. X oznacza, że Xcode nie może zasugerować rozwiązania tego problemu. Kliknij czerwone kółko.

4. Ponieważ próbujesz przypisać ciąg znaków do zmiennej typu Double, wyświetlany jest następujący komunikat o błędzie:

<img src=”images/ios31.png”>



5. Skomentuj linię, wpisując // przed nią, jak pokazano:

// restaurantRating = "Good"

Czerwone kółko zniknie, ponieważ w programie nie ma już błędów. Teraz, gdy wiesz, jak przechowywać dane w stałych i zmiennych, przyjrzyjmy się, jak wykonywać na nich operacje w następnej sekcji.

Eksploracja operatorów

W Swift możesz wykonywać operacje arytmetyczne, porównawcze i logiczne. Operatory arytmetyczne służą do typowych operacji matematycznych. Operatory porównania i operatory logiczne sprawdzają wartość wyrażenia i zwracają wartość prawda lub fałsz. Przyjrzyjmy się bardziej szczegółowo każdemu typowi operatora. W następnej sekcji zaczniesz od operatorów arytmetycznych (dodawanie, odejmowanie, mnożenie i dzielenie).

Stosowanie operatorów arytmetycznych

Możesz wykonywać operacje matematyczne na liczbach całkowitych i zmiennoprzecinkowych, używając pokazanych tutaj standardowych operatorów arytmetycznych:

+ : dodawanie

- odejmowanie

* mnożenie

/ : dzielenie

Zobaczmy, jak używane są te operatory. Wykonaj następujące kroki:

1. Dodaj następujący kod, aby dodać operacje arytmetyczne do swojego placu zabaw:

let suma = 23 + 20

let wynik = 32 - suma

let suma = wynik * 5

let dzieli = suma / 10

2. Kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić. Wyniki wyświetlone w obszarze Wyniki będą wynosić odpowiednio 43, -11, -55 i -5. Należy zauważyć, że 55 podzielone przez 10 zwraca 5 zamiast 5,5, ponieważ obie liczby są liczbami całkowitymi. 3. Operatory mogą pracować tylko z operandami tego samego typu. Wpisz następujący kod i uruchom go, aby zobaczyć, co się stanie, jeśli operandy będą różnych typów:

let a = 12

let b = 12,0

let c = a + b

Otrzymasz komunikat o błędzie (operatora binarnego „+” nie można zastosować do operandów typu „Int” i „Double”). Dzieje się tak, ponieważ a i b to różne typy. Pamiętaj, że Xcode nie może tego naprawić automatycznie, więc nie wyświetla żadnych sugestii dotyczących naprawy.

4. Aby naprawić błąd, zmodyfikuj program w następujący sposób:

niech c = Double(a) + b Double(a) pobiera wartość zapisaną w a i tworzy z niej liczbę zmiennoprzecinkową. Oba operandy są teraz tego samego typu i teraz możesz dodać do nich wartość z b. Wartość zapisana w c wynosi 24,0, a w obszarze Wyniki zostanie wyświetlona liczba 24. Teraz, gdy wiesz, jak używać operatorów arytmetycznych, w następnej sekcji przyjrzysz się złożonym operatorom przypisania (+=, -=, *= i /=).

Używanie złożonych operatorów przypisania

Możesz wykonać operację na wartości i przypisać wynik do zmiennej, używając przedstawionych tutaj złożonych operatorów przypisania:

+= Dodaje wartość i przypisuje wynik do zmiennej

-= Odejmuje wartość i przypisuje wynik do zmiennej

*= Mnoży wartość i przypisuje wynik do zmiennej

/= Dzieli wartość i przypisuje wynik do zmiennej

Zobaczmy, jak używane są te operatory. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

var d = 1

d += 2

d -= 1

Wyrażenie d += 2 jest skrótem dla d = d + 2, więc wartość w d wynosi teraz 1 + 2, a 3 zostanie przypisane do d. W ten sam sposób d -= 1 jest skrótem od d = d - 1, więc wartość w d wynosi teraz 3 - 1, a 2 zostanie

przypisane do d. Teraz, gdy znasz już złożone operatory przypisania, przyjrzyjmy się operatorom porównania (==, /=, >, <, >= i <=) w następnej sekcji.

Korzystanie z operatorów porównania

Możesz porównać jedną wartość z drugą za pomocą operatorów porównania, a wynik będzie prawdziwy lub fałszywy. Możesz użyć następujących operatorów porównania:

== Równe

!= Nierówne

> Większe niż

< Mniejsze niż

>= Większe niż lub równe

<= Mniejsze niż lub równe

Zobaczmy, jak używane są te operatory. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Przycisk Stop, aby go uruchomić:

1 == 1

2!= 1

2 > 1

1 < 2

1 >= 1

2 <= 1

Zobaczmy jak to działa:

• 1 == 1 zwraca wartość true, ponieważ 1 jest równe 1

• 2 != 1 zwraca wartość true, ponieważ 2 nie jest równe 1

• 2 > 1 zwraca prawdę, ponieważ 2 jest większe niż 1

• 1 < 2 zwraca prawdę, ponieważ 1 jest mniejsze niż 2

• 1 >= 1 zwraca wartość true, ponieważ 1 jest większe lub równe 1

• 2 <= 1 zwraca wartość false, ponieważ 2 jest nie mniejsze lub równe 1

Zwrócone wartości logiczne zostaną wyświetlone w obszarze Wyniki.

Co się stanie, jeśli chcesz sprawdzić więcej niż jeden warunek? Tutaj właśnie pojawiają się operatory logiczne (AND, OR i NOT). Przeanalizujesz je w następnej sekcji.

Używanie operatorów logicznych

Operatory logiczne są przydatne, gdy mamy do czynienia z dwoma lub większą liczbą warunków. Na przykład, jeśli jesteś w sklepie ogólnospożywczym, możesz zapłacić za produkty, jeśli masz gotówkę lub kartę kredytową. OR jest w tym przypadku operatorem logicznym. Można używać następujących operatorów logicznych:

&& Logiczne AND – zwraca true jeśli wszystkie warunki są true

|| Logiczne OR – zwraca true jeśli dowolny warunek jest true

! Logiczne NOT – zwraca przeciwieństwo wartości boolowskiej

Aby zobaczyć, jak korzystają z tych operatorów, dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

(1 == 1) && (2 == 2)

(1 == 1) && (2 != 2)

(1 == 1) || (2 == 2)

(1 == 1) || (2!= 2)

(1 != 1) || (2!= 2)

!(1 == 1)

Zobaczmy jak to działa:

• (1 == 1) && (2 == 2) zwraca prawdę, ponieważ oba operandy są prawdziwe, więc prawda ORAZ prawda zwraca prawdę

• (1 == 1) && (2 != 2) zwraca fałsz, ponieważ jeden operand jest fałszywy, więc prawda ORAZ fałsz zwraca fałsz

• (1 == 1) || (2 == 2) zwraca prawdę, ponieważ oba operandy są prawdziwe, więc prawda LUB prawda zwraca prawdę

• (1 == 1) || (2 != 2) zwraca prawdę, jeśli jeden z operandów jest prawdziwy, więc prawda LUB fałsz zwraca prawdę

• (1 != 1) || (2 != 2) zwraca wartość false, ponieważ oba operandy są fałszywe, więc wartość false OR false zwraca wartość false

• !(1 == 1) zwraca fałsz, ponieważ 1==1 jest prawdą, zatem NIE prawda zwraca fałsz

Zwrócone wartości logiczne zostaną wyświetlone w obszarze Wyniki. Do tej pory pracowałeś tylko z liczbami. W następnej sekcji zobaczysz, jak możesz wykonywać operacje na słowach i zdaniach, które są przechowywane jako ciągi znaków przy użyciu typu String Swifta.

Wykonywanie operacji na ciągach

Jak widzieliśmy wcześniej, ciąg znaków to ciąg znaków. Są one reprezentowane przez typ String i są w pełni zgodne z Unicode. Poznajmy niektóre typowe operacje na ciągach znaków. Wykonaj następujące kroki:

1. Możesz połączyć dwa ciągi znaków za pomocą operatora +. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

let greeting = "Good" + " Morning"

Wartości literałów łańcuchowych „Dobry” i „Poranny” są łączone, a w obszarze Wyniki wyświetlana jest wartość „Dzień dobry”.

2. Można łączyć ciągi znaków ze stałymi i zmiennymi innych typów, rzutując je na ciągi znaków. Aby rzutować stałą ocenę na ciąg znaków, wprowadź następujący kod i uruchom go:

let rating = 3.5

var ratingResult = "The restaurant rating is " +

String(rating)

Stała oceny zawiera 3,5, wartość typu Double. Umieszczenie oceny w nawiasach funkcji String() powoduje pobranie wartości zapisanej w ocenie i utworzenie na jej podstawie nowego ciągu „3,5”, który jest łączony z ciągiem znaków w zmiennej ratingResult, zwracając ciąg „Ocena restauracji to 3,5”

3. Istnieje prostszy sposób łączenia ciągów, zwany interpolacją ciągów. Interpolację ciągów wykonuje się poprzez wpisanie nazwy stałej lub zmiennej pomiędzy „\(” i „)” w ciągu. Wpisz następujący kod i uruchom go:

ratingResult = "The restaurant rating is \(rating)"

Podobnie jak w poprzednim przykładzie, wartość oceny jest używana do utworzenia nowego ciągu „3,5”, zwracając ciąg „Ocena restauracji to 3,5”. Dotychczasowe wyniki swoich instrukcji możesz zobaczyć w obszarze Wyniki. Jeśli jednak napiszesz aplikację przy użyciu Xcode, nie będziesz mieć dostępu do obszaru Wyniki widocznego na placu zabaw. Aby wyświetlić zawartość zmiennych i stałych podczas działania programu, w następnej sekcji dowiesz się, jak wydrukować je w obszarze Debugowanie.

Korzystanie z instrukcji print().

Jak widziałeś w Rozdziale 1, Zapoznanie się z Xcode, projekt Xcode nie ma obszaru wyników, który ma plac zabaw, ale zarówno projekt, jak i plac zabaw mają obszar debugowania. Użycie instrukcji print() spowoduje wydrukowanie zawartości zawartej w nawiasach w obszarze debugowania. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

print(ratingResult)

Wartość ratingResult pojawi się w obszarze Debugowanie:

```
64
65  print(ratingResult)

The restaurant rating is 3.5
```

Jeśli dopiero zaczynasz, możesz używać dowolnej liczby instrukcji print(). To naprawdę dobry sposób, aby zrozumieć, co dzieje się w programie.

Streszczenie

W tej lekcji nauczyłeś się tworzyć i używać plików zabaw, które pozwalają odkrywać i eksperymentować ze Swiftem. Widziałeś, jak Swift reprezentuje różne typy danych oraz jak używać stałych i zmiennych. Umożliwia to przechowywanie w programie liczb, wartości logicznych i ciągów znaków. Poznałeś także wnioskowanie o typie, adnotację typu i bezpieczeństwo typów, które pomogą Ci pisać kod zwięźle i z mniejszą liczbą błędów. Przyjrzałeś się sposobowi wykonywania operacji na liczbach i ciągach znaków, co umożliwia wykonywanie prostych zadań przetwarzania danych. Nauczyłeś się, jak naprawiać błędy i jak drukować w obszarze Debugowanie, co jest przydatne, gdy

próbujesz znaleźć i naprawić błędy w pisanych programach. W następnym rozdziale przyjrzysz się warunkom i opcjom. Warunki warunkowe dotyczą dokonywania logicznych wyborów w programie, a opcje opcjonalne dotyczą przypadków, w których zmienna może mieć wartość lub nie.

Warunki i opcje

Przyjrzałeś się typom danych, stałym, zmiennym i operacjom. W tym momencie możesz już pisać proste programy przetwarzające litery i cyfry. Jednak programy nie zawsze działają po kolei. Często będziesz musiał wykonać różne instrukcje w zależności od warunku. Swift pozwala to zrobić za pomocą warunków warunkowych, a w tej części dowiesz się, jak z nich korzystać. Kolejną rzeczą, którą mogłeś zauważyć, jest to, że w ostatniej części każdej zmiennej lub stałej została natychmiast przypisana wartość. Co się stanie, jeśli potrzebujesz zmiennej, której wartość może początkowo nie być obecna? Będziesz potrzebował sposobu na utworzenie zmiennej, która może mieć wartość lub nie. Swift pozwala to zrobić za pomocą opcji, o których również dowiesz się w tym rozdziale. Pod koniec tego rozdziału powinieneś potrafić pisać programy wykonujące różne czynności w oparciu o różne warunki oraz obsługiwać zmienne, które mogą mieć wartość lub nie.

Omówione zostaną następujące tematy:

• Wprowadzenie warunków

• Wprowadzenie opcji i opcjonalnego wiązania

Poświęć trochę czasu na zrozumienie opcji. Mogą być zniechęcające dla początkującego programisty.

Przedstawiamy warunki

Czasami będziesz chciał wykonać różne bloki kodu w oparciu o określony warunek, na przykład w następujących scenariuszach:

• Wybór pomiędzy różnymi rodzajami pokoi w hotelu. Cena za większe pokoje byłaby wyższa.

• Przełączanie pomiędzy różnymi metodami płatności w sklepie internetowym. Różne metody płatności wymagają różnych procedur.

• Podejmowanie decyzji, co zamówić w restauracji typu fast-food. Procedury przygotowania każdego artykułu spożywczego będą inne.

Aby to zrobić, użyj warunków. W Swift jest to realizowane za pomocą instrukcji if (dla pojedynczego warunku) i instrukcji switch (dla wielu warunków). Zobaczmy, jak instrukcje if są używane do wykonywania różnych zadań w zależności od wartości warunku w następnej sekcji.

Używanie instrukcji if

Instrukcja if wykonuje blok kodu, jeśli warunek jest prawdziwy, i opcjonalnie inny blok kodu, jeśli warunek jest fałszywy. Instrukcja if wygląda następująco:

```
if condition {

code1

} else {

code2

}
```

Zaimplementujmy teraz instrukcję if, aby zobaczyć to w działaniu. Wyobraź sobie, że programujesz aplikację dla restauracji. Aplikacja pozwoli Ci sprawdzić, czy restauracja jest otwarta, wyszukać ją i

sprawdzić, czy klient nie przekroczył wieku uprawniającego do spożywania alkoholu. Wykonaj następujące kroki:

1. Aby sprawdzić, czy restauracja jest otwarta, dodaj następujący kod do swojego placu zabaw, aby utworzyć stałą i wykonać instrukcję, jeśli wartość stałej jest prawdziwa. Kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

```
let isRestaurantOpen = true

if isRestaurantOpen {

print("Restaurant is open.")

}
```

Najpierw utworzyłeś stałą isRestaurantOpen i przypisałeś jej wartość true. Następnie masz instrukcję if, która sprawdza wartość przechowywaną w isRestaurantOpen. Ponieważ wartość jest prawdziwa, wykonywana jest instrukcja print(), a w obszarze Debugowanie wyświetlana jest informacja Restauracja jest otwarta.

2. Spróbuj zmienić wartość isRestaurantOpen na false i ponownie uruchom kod. Ponieważ warunek jest teraz fałszywy, w obszarze debugowania nic nie zostanie wydrukowane.

3. Możesz także wykonywać instrukcje, jeśli wartość jest fałszywa. Załóżmy, że klient szukał konkretnej restauracji, której nie ma w bazie aplikacji, więc aplikacja powinna wyświetlić komunikat, że restauracji nie znaleziono. Wpisz następujący kod, aby utworzyć stałą i wykonać instrukcję, jeśli wartość stałej jest fałszywa:

```
let isRestaurantFound = false

if isRestaurantFound == false {

print("Restaurant was not found")

}
```

Stała isRestaurantFound ma wartość false. Następnie sprawdzana jest instrukcja if. Warunek isRestaurantFound == false zwraca wartość true, a w obszarze debugowania wyświetlana jest informacja: Nie znaleziono restauracji.

4. Spróbuj zmienić wartość isRestaurantFound na true. Ponieważ warunek jest teraz fałszywy, w obszarze debugowania nic nie zostanie wydrukowane.

5. Aby wykonać jeden zestaw instrukcji, jeśli warunek jest prawdziwy, i inny zestaw instrukcji, jeśli warunek jest fałszywy, użyj słowa kluczowego else. Wpisz poniższy kod, który sprawdzi, czy klient baru nie przekroczył wieku uprawniającego do spożycia alkoholu:

```
let drinkingAgeLimit = 21

let customerAge = 23

if customerAge < drinkingAgeLimit {

print("Under age limit")

} else {
```

```
print("Over age limit")

}
```

Tutaj DrinkAgeLimit ma przypisaną wartość 21, a klientAge ma przypisaną wartość 23. W instrukcji if sprawdzany jest klientAge < DrinkAgeLimit. Ponieważ 23 < 21 zwraca wartość false, wykonywana jest instrukcja else, a w obszarze Debugowanie wyświetlany jest limit wieku Over. Jeśli zmienisz wartość CustomerAge na 19, CustomerAge < DrinkAgeLimit zwróci wartość true, więc w obszarze Debug zostanie wyświetlony limit poniżej wieku.

Do tej pory miałeś do czynienia tylko z pojedynczymi warunkami. A co jeśli warunków jest wiele? W tym miejscu pojawiają się instrukcje switch, o których dowiesz się w następnej sekcji.

Korzystanie z instrukcji switch

Aby zrozumieć instrukcje switch, zacznijmy od zaimplementowania instrukcji if z wieloma warunkami. Wyobraź sobie, że programujesz sygnalizację świetlną. Istnieją trzy możliwe kolory sygnalizacji świetlnej — czerwony, żółty lub zielony — i chcesz, aby w zależności od koloru światła wydarzyło się coś innego. Aby to zrobić, możesz połączyć ze sobą wiele instrukcji if. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw, aby zaimplementować sygnalizację świetlną za pomocą wielu instrukcji if i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

```
var trafficLightColor = "Yellow"

if trafficLightColor == "Red" {

print("Stop")

} else if trafficLightColor == "Yellow" {

print("Caution")

} else if trafficLightColor == "Green" {

print("Go")

} else {

print("Invalid color")

}
```

Pierwszy warunek if, TrafficLightColor == "Red", zwraca wartość false, więc wykonywana jest instrukcja else. Drugi warunek if, TrafficLightColor == "Yellow", zwraca wartość true, więc w obszarze debugowania wyświetlana jest informacja Uwaga, a nie więcej, jeśli oceniane są warunki. Spróbuj zmienić wartość TrafficLightColor, aby zobaczyć inne wyniki. Zastosowany tutaj kod działa, ale jest trochę trudny do odczytania. W tym przypadku instrukcja switch byłaby bardziej zwięzła i łatwiejsza do zrozumienia. Instrukcja switch wygląda następująco:

```
switch value {

case firstValue:

code1

case secondValue:
```

```
code2

default:

code3

}
```

Wartość jest sprawdzana i dopasowywana do przypadku, po czym wykonywany jest kod dla tego przypadku. Jeśli żaden z przypadków nie pasuje, wykonywany jest kod w przypadku domyślnym.

2. Oto jak zapisać pokazaną wcześniej instrukcję if jako instrukcję switch. Wpisz następujący kod:

```
trafficLightColor = "Yellow"

switch trafficLightColor {

case "Red":

print("Stop")

case "Yellow":

print("Caution")

case "Green":

print("Go")

default:

print("Invalid color")

}
```

Kod tutaj jest znacznie łatwiejszy do odczytania i zrozumienia w porównaniu do poprzedniej wersji. Wartością w TrafficLightColor jest „Yellow”, więc wielkość liter „Yellow” jest dopasowana, a w obszarze Debugowanie zostanie wydrukowana uwaga. Spróbuj zmienić wartość TrafficLightColor, aby zobaczyć inne wyniki. Są dwie rzeczy do zapamiętania na temat instrukcji switch:

• Instrukcje switch w Swift domyślnie nie przechodzą na dół każdego przypadku i przechodzą do następnego. W przykładzie pokazanym wcześniej, po dopasowaniu przypadku „Czerwony”: przypadek „Żółty”: przypadek „Zielony” i wartość domyślna: nie zostanie wykonana.

• Instrukcje switch muszą obejmować wszystkie możliwe przypadki. W powyższym przykładzie dowolna wartość TrafficLightColor inna niż „Red”, „Yellow” lub „Green” zostanie dopasowana do wartości domyślnej, a w obszarze debugowania zostanie wydrukowany nieprawidłowy kolor.

Na tym kończy się sekcja dotycząca instrukcji if i switch. W następnej sekcji dowiesz się o opcjach, które pozwalają tworzyć zmienne bez wartości początkowych, oraz opcjonalnym wiązaniu, które pozwala na wykonanie instrukcji, jeśli opcja ma wartość.

Przedstawiamy opcje i opcjonalne wiązanie

Do tej pory za każdym razem, gdy deklarowałeś zmienną lub stałą, natychmiast przypisywałeś jej wartość. Ale co, jeśli chcesz najpierw zadeklarować zmienną, a później przypisać jej wartość? W tym przypadku użyłbyś opcji. Nauczmy się, jak tworzyć i używać opcji oraz zobaczmy, jak są one wykorzystywane w programie. Wyobraź sobie, że piszesz program, w którym użytkownik musi wpisać

imię i nazwisko współmałżonka. Oczywiście, jeśli użytkownik nie jest żonaty, nie będzie to miało żadnej wartości. W takim przypadku możesz użyć opcji, aby przedstawić imię i nazwisko współmałżonka. Opcja opcjonalna może mieć jeden z dwóch możliwych stanów. Może zawierać wartość lub nie zawierać wartości. Jeśli opcja zawiera wartość, możesz uzyskać dostęp do wartości znajdującej się w niej. Proces uzyskiwania dostępu do wartości opcji nazywany jest rozpakowywaniem opcji. Zobaczmy, jak to działa. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw, aby utworzyć zmienną i wydrukować jej zawartość:

var spouseName: String

print(spouseName)

2. Kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić. Ponieważ Swift jest bezpieczny pod względem typu, wyświetli błąd (zmienna „nazwa współmałżonka” użyta przed inicjalizacją).

3. Aby rozwiązać ten problem, możesz przypisać pusty ciąg do współmałżonkaName. Zmodyfikuj swój kod, jak pokazano:

var spouseName: String = ""

To sprawia, że błąd znika, ale pusty ciąg znaków nadal jest wartością, a małżonkaName nie powinna mieć wartości.

4. Ponieważ nazwa małżonka początkowo nie powinna mieć wartości, uczyńmy ją opcjonalną. Aby to zrobić, wpisz znak zapytania po adnotacji typu i usuń przypisanie pustego ciągu:

var spouseName: String?

Zobaczysz ostrzeżenie, ponieważ współmałżonekName jest teraz opcjonalną zmienną łańcuchową zamiast zwykłej zmiennej łańcuchowej, a instrukcja print() oczekuje zwykłej zmiennej łańcuchowej.

```
44 var spouseName: String?
45 print(spouseName)        ⚠ Expression implicitly coerced from 'String?' to 'Any'
```

Nawet jeśli pojawi się ostrzeżenie, program zostanie wykonany. Na razie zignoruj ostrzeżenie. Wartość małżonkaName jest wyświetlana jako „nil\n” w obszarze Wyniki, a zero jest drukowane w obszarze Debugowanie. nil to specjalne słowo kluczowe, które oznacza, że opcjonalna zmienna małżonka imienia nie ma wartości.

5. Pojawia się ostrzeżenie, ponieważ instrukcja print traktuje małżonkaName jako typ Any zamiast String?. Kliknij żółty trójkąt, aby wyświetlić możliwe poprawki i wybierz pierwszą poprawkę:

```
44 var spouseName: String?
45 print(spouseName)
47
   ⚠ Expression implicitly coerced from 'String?' to 'Any'
     Provide a default value to avoid this warning        [Fix]
     Force-unwrap the value to avoid this warning         [Fix]
     Explicitly cast to 'Any' with 'as Any' to silence this warning  [Fix]
```

Instrukcja zmieni się na print(nazwa współmałżonka? wartość domyślna). Zwróć uwagę na użycie ?? operator. Oznacza to, że jeśli małżonkaName nie zawiera wartości, w instrukcji print zostanie użyta podana przez Ciebie wartość domyślna.

6. Zamień symbol zastępczy wartości domyślnej na „Brak wartości w współmałżonku”, jak pokazano. Ostrzeżenie zniknie. Uruchom program ponownie, a w obszarze Wyniki pojawi się informacja „Brak wartości w współmałżonku”:

```
44 var spouseName: String?                                   nil
45 print(spouseName ?? "No value in spouseName")             "No value in sp...
```

7. Przypiszmy wartość małżonkowiName. Zmodyfikuj kod, jak pokazano:

var spouseName: String?

spouseName = "Nia"

print(spouseName ?? "No value in spouseName")

Po uruchomieniu programu Nia pojawi się w obszarze debugowania.

8. Dodaj jeszcze jedną linię kodu, aby połączyć małżonkaName z innym ciągiem, jak pokazano:

print(spouseName ?? "No value in spouseName")

let greeting = "Hello, " + spouseName

Pojawi się błąd, a w obszarze Debugowanie zostaną wyświetlone informacje o błędzie i miejsce jego wystąpienia. Stało się tak, ponieważ nie można połączyć zwykłej zmiennej typu String z opcjonalną za pomocą operatora +. Najpierw musisz rozpakować opcję opcjonalną.

9. Kliknij czerwone kółko, aby wyświetlić możliwe poprawki, a zobaczysz następujące informacje:

```
46 print(spouseName ?? "No value in spouseName")
47 let greeting = "Hello, " + spouseName
49
50
```

Value of optional type 'String?' must be unwrapped to a value of type 'String'

Coalesce using '??' to provide a default when the optional value contains 'nil' — Fix

Force-unwrap using '!' to abort execution if the optional value contains 'nil' — Fix

Druga poprawka zaleca wymuszenie rozpakowania, aby rozwiązać ten problem. Wymuszone rozpakowywanie powoduje rozpakowanie opcji niezależnie od tego, czy zawiera ona wartość, czy nie. Działa dobrze, jeśli współmałżonekName ma wartość, ale jeśli małżonekName ma wartość zero, program ulegnie awarii.

10. Kliknij drugą poprawkę, a po nazwie małżonka w ostatnim wierszu kodu pojawi się wykrzyknik, co oznacza, że opcja jest wymuszona:

let greeting = "Hello, " + spouseName!

11. Po uruchomieniu programu powitanie zostanie przypisane do powitania, jak pokazano w obszarze Wyniki. Oznacza to, że małżonkaName została pomyślnie rozpakowana.

12. Aby zobaczyć efekt wymuszonego rozpakowania zmiennej zawierającej zero, ustaw współmałżonkaName na nil:

spouseName = nil

Twój program ulega awarii i możesz zobaczyć, co było przyczyną awarii w obszarze Debugowanie:

```
43
44 var spouseName: String?
45 spouseName = nil
46 print(spouseName ?? "No value in
       spouseName")
47 let greeting = "Hello, " +
       spouseName!
```

```
No value in spouseName
__lldb_expr_40/ConditionalsAndOptionals.playground:47: Fatal
error: Unexpectedly found nil while unwrapping an Optional
value
```

Ponieważ współmałżonekName ma teraz wartość zero, program uległ awarii podczas próby wymuszenia rozpakowania małżonkaName. Lepszym sposobem poradzenia sobie z tym jest użycie opcjonalnego powiązania. W opcjonalnym wiązaniu próbujesz przypisać wartość opcjonalną do zmiennej tymczasowej (możesz nadać jej dowolną nazwę). Jeśli przypisanie się powiedzie, wykonywany jest blok kodu.

13. Aby zobaczyć efekt opcjonalnego powiązania, zmodyfikuj swój kod w następujący sposób:

spouseName = "Nia"

print(spouseName ?? "No value in spouseName")

if let spouseTempVar = spouseName {

let greeting = "Hello, " + spouseTempVar

print(greeting)

}

Witaj, Nia pojawi się w obszarze debugowania. Oto jak to działa. Jeśli małżonkName ma wartość, zostanie ona rozpakowana i przypisana do tymczasowej stałej, małżonkTempVar, a instrukcja if zwróci wartość true. Instrukcje zawarte w nawiasach klamrowych zostaną wykonane, a stałemu powitaniu zostanie przypisana wartość Witaj, Nia. Następnie w obszarze debugowania zostanie wydrukowana wiadomość Witaj, Nia. Należy pamiętać, że zmienna tymczasowa małżonkaTempVar nie jest opcjonalna. Jeśli małżonkName nie ma wartości, nie można przypisać żadnej wartości do współmałżonkaTempVar, a instrukcja if zwróci wartość fałszywą. W takim przypadku instrukcje w nawiasach klamrowych w ogóle nie zostaną wykonane.

14. Aby zobaczyć efekt opcjonalnego powiązania, gdy opcja zawiera zero, przypisz nil jeszcze raz do małżonka:

spouseName = nil

Zauważysz, że w obszarze Debugowanie nic się nie pojawia, a Twój program nie ulega już awariom, mimo że nazwa małżonka ma wartość zero. Na tym kończy się sekcja poświęcona opcjom i opcjonalnym powiązaniom. Możesz teraz tworzyć i używać zmiennych opcjonalnych. Wspaniale!

Streszczenie

Świetnie sobie radzisz! Nauczyłeś się używać instrukcji if i switch, co oznacza, że możesz teraz pisać własne programy, które wykonują różne czynności w oparciu o różne warunki. Dowiedziałeś się także o opcjach i opcjonalnym wiązaniu. Oznacza to, że możesz teraz reprezentować zmienne, które mogą mieć wartość lub nie, i wykonywać instrukcje tylko wtedy, gdy wartość zmiennej jest obecna. W następnym rozdziale dowiesz się, jak używać zakresu wartości zamiast pojedynczych wartości i jak powtarzać instrukcje programu za pomocą pętli.

Operatory zakresu i pętle

W poprzedniej części zapoznałeś się z warunkami warunkowymi, które pozwalają wykonywać różne czynności w oparciu o różne warunki, oraz opcjami, które umożliwiają tworzenie zmiennych, które mogą mieć wartość lub nie. W tej części dowiesz się o operatorach zakresu i pętlach. Operatory zakresu pozwalają reprezentować zakres wartości poprzez określenie wartości początkowej i końcowej zakresu. Dowiesz się także o różnych typach operatorów zakresu. Pętle umożliwiają wielokrotne powtarzanie instrukcji lub sekwencji instrukcji. Możesz powtórzyć sekwencję określoną liczbę razy lub powtarzać sekwencję, aż spełniony zostanie warunek. Dowiesz się o różnych typach pętli używanych do osiągnięcia tego celu. Pod koniec tego rozdziału dowiesz się, jak korzystać z zakresów oraz jak tworzyć i używać różnych typów pętli (for-in, while i powtarzaj-while).

Odkrywanie operatorów zakresu

Operatory zakresu pozwalają reprezentować zakres wartości. Załóżmy, że chcesz przedstawić sekwencję liczb zaczynającą się od pierwszego numeru i kończącą się na ostatnim numerze. Nie musisz podawać każdej wartości; możesz po prostu określić zakres w ten sposób:

firstNumber ... lastNumber

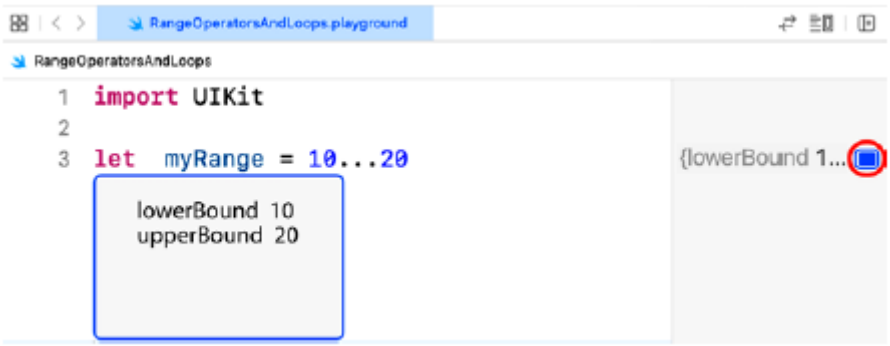
Wyobraź sobie, że musisz napisać program dla domu towarowego, który automatycznie wysyła kupon rabatowy klientom w wieku od 18 do 30 lat. Byłoby bardzo kłopotliwe, gdybyś musiał skonfigurować instrukcję if lub switch dla każdego wieku. W tym przypadku o wiele wygodniej jest użyć operatora zakresu. Wypróbujmy to na placu zabaw. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

let myRange = 10...20

Spowoduje to przypisanie sekwencji liczb zaczynającej się od 10 i kończącej się na 20, włączając obie liczby, do stałej myRange. Nazywa się to operatorem zamkniętego zasięgu. Wartości LowerBound i UpperBound dla myRange zostaną wyświetlone w obszarze Wyniki.

2. Wynik wyświetlany w obszarze Wyniki może zostać obcięty. Kliknij kwadratową ikonę po prawej stronie wyniku. Zostanie wyświetlony w obszarze Edytora:



Możesz teraz zobaczyć pełny wynik w polu pod linią kodu. Jeśli chcesz, możesz przeciągnąć prawą krawędź, aby powiększyć pole.

Pamiętaj, że możesz przeciągnąć granicę pomiędzy obszarami Wyniki i Edytor, aby zwiększyć rozmiar obszaru Wyniki.

3. Jeśli nie chcesz uwzględniać ostatniej cyfry ciągu w zakresie, użyj ..< zamiast .... Wpisz i uruchom następujący kod:

let myRange2 = 10..<20

4. Spowoduje to zapisanie sekwencji zaczynającej się od 10 i kończącej się na 19 w stałej myRange2 i jest to znane jako operator zakresu półotwartego.

Istnieje jeszcze jeden rodzaj operatora zakresu, jednostronny operator zakresu, o którym dowiesz się w następnej części.

Teraz, gdy już wiesz, jak tworzyć zakresy i z nich korzystać, w następnej sekcji dowiesz się o pętlach, różnych typach pętli i sposobach ich używania.

Odkrywanie pętli

W programowaniu często trzeba robić to samo w kółko. Na przykład co miesiąc firma będzie musiała generować paski płac dla każdego pracownika. Jeśli firma zatrudnia 10 000 pracowników, nieefektywne byłoby pisanie 10 000 instrukcji tworzenia pasków płac. Lepsze byłoby powtórzenie pojedynczej instrukcji 10 000 razy i używa się do tego pętli. Istnieją trzy typy pętli: pętla for-in, pętla while i pętla powtarzania-while. Pętla for-in będzie powtarzana znaną liczbę razy, natomiast pętle while i Repeat-while będą się powtarzać tak długo, jak warunek pętli będzie spełniony. Przyjrzyjmy się kolejno każdemu typowi, zaczynając od pętli for-in, której używamy, gdy wiemy, ile razy pętla powinna zostać powtórzona.

Korzystanie z pętli for-in

Pętla for-in przechodzi przez każdą wartość w sekwencji i za każdym razem wykonywany jest zestaw instrukcji w nawiasach klamrowych, zwany treścią pętli. Każda wartość jest kolejno przypisana do zmiennej tymczasowej, a zmienna tymczasowa może zostać użyta w treści pętli. Oto jak to wygląda:

for item in sequence { code

}

Liczba powtórzeń pętli zależy od liczby elementów w sekwencji. Zacznijmy od utworzenia pętli for-in, która wyświetli wszystkie liczby w myRange. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

for number in myRange {

print(number)

}

Powinieneś zobaczyć każdy numer w sekwencji wyświetlony w obszarze Debugowanie. Należy zauważyć, że instrukcje wewnątrz pętli są wykonywane 11 razy, ponieważ myRange zawiera ostatnią liczbę z zakresu.

2. Spróbujmy tego samego programu, ale tym razem z myRange2. Zmodyfikuj kod w następujący sposób i uruchom go:

```
for number in myRange2 {

print(number)

}
```

Instrukcje wewnątrz pętli są wykonywane 10 razy, a ostatnia wartość wydrukowana w obszarze Debug to 19.

3. Możesz nawet użyć operatora zakresu bezpośrednio po słowie kluczowym in. Wpisz i uruchom następujący kod:

```
for number in 0...5 {

print(number)

}
```

Każda liczba od 0 do 5 jest wyświetlana w obszarze debugowania.

4. Jeśli chcesz, aby sekwencja została odwrócona, użyj funkcji Reversed(). Zmodyfikuj kod w następujący sposób i uruchom go:

```
for number in (0...5).reversed() {

print(number)

}
```

Każda liczba od 5 do 0 jest wyświetlana w obszarze debugowania.

Dobra robota! W następnej sekcji przyjrzyjmy się pętlom while, które są używane, gdy sekwencja pętli powinna być powtarzana, dopóki warunek jest spełniony.

Korzystanie z pętli while

Pętla while zawiera warunek i zestaw instrukcji w nawiasach klamrowych, zwanych treścią pętli. Najpierw sprawdzany jest warunek; jeśli prawda, wykonywany jest korpus pętli i pętla jest powtarzana, aż warunek stanie się fałszywy. Oto przykład, jak wygląda pętla while:

```
while condition == true {

code

}
```

Dodaj następujący kod, aby utworzyć zmienną, zwiększ ją o 5 i rób to tak długo, aż wartość zmiennej będzie mniejsza niż 50. Kliknij przycisk Odtwórz/Zatrzymaj, aby ją uruchomić:

```
var y = 0

while y < 50 {

y += 5

print("y is \(y)")

}
```

Przejdźmy przez kod. Początkowo y jest ustawione na 0. Warunek y < 50 jest sprawdzany i zwraca wartość true, zatem wykonywane jest ciało pętli. Wartość y jest zwiększana o 5, a wartość y wynosi 5 jest drukowana w obszarze Debugowanie. Pętla się powtarza i ponownie sprawdzane jest y < 50. Ponieważ y wynosi teraz 5, a 5 < 50 nadal zwraca wartość true, ciało pętli jest wykonywane ponownie. Powtarza się to, aż wartość y osiągnie 50, w którym to momencie y < 50 zwróci wartość false i pętla się zatrzyma. Jeśli na początku warunek pętli while jest fałszywy, treść pętli nigdy nie zostanie wykonana. Aby to zobaczyć, spróbuj zmienić wartość y na 100. W następnej części zapoznasz się z pętlami powtarzania-while. Spowodują one wykonanie najpierw instrukcji z treści pętli przed sprawdzeniem warunku pętli.

Pętla repeat- while

Podobnie jak pętla while, pętla powtarzania while również zawiera warunek i treść pętli, ale treść pętli jest wykonywana najpierw przed sprawdzeniem warunku. Jeśli warunek jest prawdziwy, pętla jest powtarzana, aż warunek zwróci wartość false. Oto przykład, jak wygląda pętla powtarzania podczas:

```
repeat {

code

} while condition == true
```

Dodaj następujący kod, aby utworzyć zmienną, zwiększ ją o 5 i rób to tak długo, aż wartość zmiennej będzie mniejsza niż 50. Kliknij przycisk Odtwórz/Zatrzymaj, aby ją uruchomić:

```
var x = 0

repeat {

x += 5

print("x is \(x)")

} while x < 50
```

Przejdźmy przez kod. Początkowo x jest ustawione na 0. Wykonywany jest korpus pętli. Wartość x jest zwiększana o 5, więc teraz x zawiera 5, a x wynosi 5 jest wypisywane w obszarze debugowania. Sprawdzany jest warunek x < 50, a ponieważ zwraca wartość true, pętla się powtarza. Wartość x jest zwiększana o 5, więc teraz x zawiera 10, a x wynosi 10 i jest wypisywane w obszarze debugowania. Pętla jest powtarzana, aż x będzie zawierało 50, w którym to momencie x < 50 zwróci wartość false i pętla się zatrzyma. Treść pętli zostanie wykonana co najmniej raz, nawet jeśli na początku warunek jest fałszywy. Aby to zobaczyć, spróbuj zmienić wartość x na 100. Teraz wiesz, jak tworzyć i używać różnych typów pętli. Wspaniale!

Dobra robota!

W następnym rozdziale zapoznasz się z typami kolekcji, które umożliwiają przechowywanie kolekcji danych, do których odwołuje się indeks, kolekcji par klucz-wartość oraz kolekcji danych bez struktury.

Streszczenie

Przyjrzałeś się operatorom zakresu zamkniętego i półotwartego, które pozwalają określić zakres liczb zamiast dyskretnego określania każdej pojedynczej liczby. Poznałeś także trzy różne typy pętli: pętlę for-in, pętlę while i pętlę powtarzania. Pętla for-in umożliwia powtarzanie zestawu instrukcji określoną

liczbę razy, natomiast pętle while i repeat-while umożliwiają powtarzanie zestawu instrukcji, dopóki warunek jest spełniony.

Typy kolekcji

W tym momencie nauczyłeś się całkiem sporo! Możesz teraz utworzyć program przechowujący dane w stałych lub zmiennych i wykonujący na nich operacje, a także możesz kontrolować przepływ za pomocą warunków i pętli. Jednak do tej pory przechowywałeś głównie pojedyncze wartości. W tym rozdziale dowiesz się, jak przechowywać kolekcje wartości. Swift ma trzy typy kolekcji: tablice, które przechowują uporządkowaną listę wartości; słowniki przechowujące nieuporządkowaną listę par klucz-wartość; i zestawy, które przechowują nieuporządkowaną listę wartości. Pod koniec tego rozdziału dowiesz się, jak tworzyć tablice, słowniki i zbiory oraz jak wykonywać na nich operacje. Pierwszym typem kolekcji, o którym się dowiesz, są tablice, które umożliwiają przechowywanie informacji w postaci uporządkowanej listy.

Zrozumienie tablic

Załóżmy, że chcesz przechowywać następujące informacje:

• Lista artykułów do kupienia w sklepie ogólnospożywczym

• Obowiązki, które musisz wykonywać co miesiąc

Do tego odpowiednie będą tablice. Tablica przechowuje wartości na uporządkowanej liście. Oto jak to wygląda:

Index : Wartość

0 : wartość 1

1 : wartość 2

2 : wartość 3

Wartości muszą być tego samego typu. Dostęp do dowolnej wartości w tablicy można uzyskać za pomocą indeksu tablicy, który zaczyna się od 0. Jeśli utworzysz tablicę za pomocą słowa kluczowego let, jej zawartości nie będzie można zmienić po jej utworzeniu. Jeśli chcesz zmienić zawartość tablicy po utworzeniu, użyj słowa kluczowego var. Zobaczmy, jak pracować z tablicami. W następnej sekcji utworzysz tablicę, przypisując jej wartość.

Tworzenie tablicy

W poprzednich częściach tworzyłeś stałą lub zmienną, deklarując ją i przypisując jej wartość początkową. W ten sam sposób możesz utworzyć tablicę. Wyobraź sobie, że twój współmałżonek poprosił cię o przyniesienie kilku artykułów ze sklepu ogólnospożywczego. Zaimplementujmy listę zakupów za pomocą tablicy. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

```
var shoppingList = ["Jajka", "Mleko"]
```

Ta instrukcja tworzy zmienną tablicową o nazwie shoppingList. Przypisana wartość ["Jajka", "Mleko"] jest literałem tablicowym. Reprezentuje tablicę składającą się z dwóch elementów typu String, z „Eggs” w indeksie 0 i „Milk” w indeksie 1. Użycie tutaj słowa kluczowego var oznacza, że zawartość tablicy można modyfikować. Ponieważ Swift używa wnioskowania typu, elementy tej tablicy będą typu String. Wyobraź sobie, że musisz sprawdzić, ile artykułów potrzebujesz, aby dostać się do sklepu. W następnej sekcji dowiesz się, jak określić liczbę elementów w tablicy.

Dodanie nowego elementu do tablicy

Możesz dodać nowy element na końcu tablicy, używając funkcji append(_:). Wpisz i uruchom następujący kod:

shoppingList.append("Olej spożywczy")

Na końcu tablicy shoppingList dodano „Cooking Oil”, która zawiera teraz trzy elementy – „Eggs”, „Mleko” i „Cooking Oil”. Można to zobaczyć w obszarze Wyniki. Możesz dodać nowy element o określonym indeksie, używając funkcji wstaw(_:at:). Wpisz i uruchom następujący kod:

shoppingList.insert("Kurczak", w: 1)

Spowoduje to wstawienie „Kurczaka” pod indeksem 1, więc teraz tablica shoppingList zawiera „Jajka”, „Kurczak”, „Mleko” i „Olej kuchenny”. Zauważ, że „Kurczak” jest drugim elementem tablicy, w którym znajduje się pierwszy element indeks 0. Można to zobaczyć w obszarze Wyniki. Można również sprawdzić, czy tablica jest pusta, używając shoppingList.count == 0, ale użycie shoppingList.isEmpty zapewnia lepszą wydajność. Możesz także dodać nowy element do tablicy za pomocą operatora +, używając następującego kodu:

shoppingList = shoppingList + ["Olej kuchenny"].

Wyobraź sobie, że masz pierwszą pozycję na liście zakupów i teraz musisz wiedzieć, jaka będzie następna pozycja na liście. W następnej sekcji zobaczysz, jak uzyskać dostęp do określonego elementu tablicy za pomocą indeksu tablicy.

Dostęp do elementu tablicy

Można określić indeks tablicy, aby uzyskać dostęp do określonego elementu. Wpisz i uruchom następujący kod:

shoppingList[2]

Spowoduje to zwrócenie elementu tablicy przechowywanego pod indeksem 2, a w obszarze Wyniki zostanie wyświetlona wartość „Mleko”. Wyobraź sobie, że zadzwonił do Ciebie współmałżonek i poprosił Cię o mleko sojowe zamiast mleka. Ponieważ tablica ta została zadeklarowana przy użyciu słowa kluczowego var, możesz modyfikować przechowywane w niej wartości. Dowiesz się, jak to zrobić w następnej sekcji.

Przypisanie nowej wartości do konkretnego indeksu

Istniejący element tablicy można zastąpić, określając indeks i przypisując mu nową wartość. Wpisz i uruchom następujący kod:

shoppingList[2] = "Mleko sojowe"

shoppingList

Zastępuje to wartość zapisaną w indeksie 2 „Mleko” wartością „Mleko sojowe”. Tablica shoppingList zawiera teraz „Jajka”, „Kurczak”, „Mleko sojowe” i „Olej kuchenny”, jak pokazano w obszarze Wyniki. Należy pamiętać, że zastosowany indeks musi być ważny. Na przykład nie możesz użyć indeksu 4, ponieważ jedynymi prawidłowymi indeksami są tutaj 0, 1, 2 i 3. Spowodowałoby to awarię programu. Wyobraź sobie, że zadzwonił do Ciebie współmałżonek i powiedział, że w lodówce jest kurczak, więc nie musisz już go kupować. W następnej sekcji zobaczysz dwa sposoby usuwania elementów z tablicy. Usuwanie elementu z tablicy Element z tablicy można usunąć za pomocą polecenia Remove(at:). Wpisz i uruchom następujący kod: shoppingList.remove(at: 1)

shoppingList

Spowoduje to usunięcie elementu o indeksie 1, „Kurczak”, z tablicy shoppingList, więc teraz zawiera on „Jajka”, „Mleko sojowe” i „Olej kuchenny”. Możesz to zobaczyć w obszarze Wyniki. Jeśli usuwasz ostatni element tablicy, możesz zamiast tego użyć funkcji usuwaniaLast().Wyobraź sobie, że masz każdy element na liście i chcesz sprawdzić przyrost listy, aby się upewnić. Będziesz musiał uzyskać dostęp do każdego elementu tablicy po kolei i wykonaj operacje na każdym elemencie. Zobaczysz, jak to zrobić w następnej sekcji. Iteracja po tablicy Czy pamiętasz pętlę for-in, którą studiowałeś w poprzednim rozdziale? Możesz jej użyć do iteracji po każdym elemencie tablicy array.Wpisz i uruchom następujący kod:

for shoppingListItem na shoppingList {

print (shoppingList)

}

Spowoduje to wyświetlenie każdego elementu tablicy w obszarze debugowania. Można także używać jednostronnych operatorów zakresu. Są to operatory zakresów mające tylko wartość początkową, na przykład 1. Wpisz i uruchom następujący kod:

for shoppingListItem w shoppingList[1...] {

print (shoppingList)

}

Spowoduje to wyświetlenie elementów tablicy, zaczynając od elementu o indeksie 1 do obszaru debugowania. Teraz wiesz, jak używać tablicy do tworzenia uporządkowanej listy, takiej jak lista zakupów, oraz jak wykonywać operacje na tablicy, takie jak uzyskiwanie dostępu, dodawanie i usuwanie elementów. W następnej sekcji przyjrzyjmy się, jak przechowywać nieuporządkowaną listę par klucz-wartość za pomocą słownika.v

Zrozumienie słowników

Załóżmy, że piszesz aplikację książki adresowej. Będziesz musiał przechowywać listę nazwisk i odpowiadających im numerów kontaktowych. Słownik byłby do tego idealny. Słownik przechowuje pary klucz-wartość na nieuporządkowanej liście. Oto jak to wygląda:

Klucz : Wartość

Key1 : Value1

Key2 : Value2

Key3 : Value3

Wszystkie klucze muszą być tego samego typu i muszą być unikalne. Wszystkie wartości muszą być tego samego typu, ale niekoniecznie są unikalne. Klucze i wartości nie muszą być tego samego typu. Używasz klucza, aby uzyskać odpowiednią wartość. Jeśli utworzysz słownik za pomocą słowa kluczowego let, jego zawartości nie będzie można zmienić po jego utworzeniu. Jeśli chcesz zmienić zawartość po utworzeniu, użyj słowa kluczowego var. Przyjrzyjmy się, jak pracować ze słownikami. W następnej sekcji utworzysz słownik, przypisując mu wartość.

Tworzenie słownika

Wyobraź sobie, że tworzysz aplikację książki adresowej. W tej aplikacji będziesz używać słownika do przechowywania kontaktów. Podobnie jak tablicę, możesz utworzyć nowy słownik, deklarując go i przypisując mu wartość początkową. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

var contactList = ["Shah": "+60123456789", "Aamir": "+0223456789"]

Ta instrukcja tworzy zmienną słownikową o nazwie contactList. Przypisana wartość ["Shah": "+60123456789", "Aamir": "+0223456789"] jest literałem słownikowym. Reprezentuje słownik z dwoma elementami. Każdy element to para klucz-wartość, której kluczem jest nazwa kontaktu, a wartością numer kontaktu. Pamiętaj, że ponieważ nazwa kontaktu jest polem kluczowym, powinna być unikalna. Ponieważ słownik contactList jest zmienną, możesz zmienić zawartość słownika po jego utworzeniu. Zarówno klucz, jak i wartość są typu String ze względu na wnioskowanie o typie. Wyobraź sobie, że Twoja aplikacja musi wyświetlać całkowitą liczbę kontaktów. W następnej sekcji dowiesz się, jak określić liczbę elementów w słowniku.

Sprawdzanie liczby elementów w słowniku

Aby dowiedzieć się, ile elementów znajduje się w słowniku, użyj funkcji count. Wpisz i uruchom następujący kod:

contactList.count

Ponieważ w słowniku contactList znajdują się dwa elementy, w obszarze Wyniki zostanie wyświetlona liczba 2. Możesz sprawdzić, czy słownik jest pusty, używając isEmpty. Wpisz i uruchom następujący kod:

contactList.isEmpty

Ponieważ słownik contactList zawiera dwa elementy, w obszarze Wyniki wyświetlana jest wartość false. Można także sprawdzić, czy słownik jest pusty, używając contactlist.count == 0, ale użycie contactList.isEmpty zapewnia lepszą wydajność.

Wyobraź sobie, że właśnie zakończyłeś spotkanie i chcesz dodać nowy kontakt do swojej aplikacji. Ponieważ słownik ten został zadeklarowany przy użyciu słowa kluczowego var, można do niego dodać pary klucz-wartość.

Dodanie nowego elementu do słownika

Aby dodać nowy element do słownika należy podać klucz i przypisać mu wartość. Wpisz i uruchom następujący kod:

contactList["Jane"] = "+0229876543"

contactList

Spowoduje to dodanie nowej pary klucz-wartość z kluczem „Jane” i wartością „+0229876543” do słownika contactList. Obecnie składa się z „Shah”: „+60126789345”, „Aamir”: „+0223456789” i „Jane”: „+0229876543”. Możesz to zobaczyć w obszarze Wyniki. Wyobraź sobie, że chcesz zadzwonić do jednego ze swoich kontaktów i potrzebujesz numeru telefonu tego kontaktu. W następnej sekcji zobaczysz, jak uzyskać dostęp do elementów słownika, określając klucz dla żądanej wartości.

Dostęp do elementu słownika

Można określić klucz słownika, aby uzyskać dostęp do odpowiadającej mu wartości. Wpisz i uruchom następujący kod:

```
contactList["Shah"]
```

Spowoduje to zwrócenie wartości klucza „Shah”, a w obszarze Wyniki zostanie wyświetlony numer +60123456789. Wyobraź sobie, że jeden z Twoich kontaktów ma nowy telefon, więc musisz zaktualizować numer telefonu tego kontaktu. Możesz modyfikować pary klucz-wartość przechowywane w słowniku. Dowiesz się, jak to zrobić w następnej sekcji.

Przypisanie nowej wartości do istniejącego klucza

Możesz przypisać nową wartość do istniejącego klucza. Wpisz i uruchom następujący kod:

```
contactList["Shah"] = "+60126789345"

contactList
```

Spowoduje to przypisanie nowej wartości do klucza „Shah”. Słownik contactList zawiera teraz słowa „Shah”: „+60126789345”, „Aamir”: „+0223456789” i „Jane”: „+0229876543”. Możesz to zobaczyć w obszarze Wyniki.

Wyobraź sobie, że musisz usunąć kontakt ze swojej aplikacji. Zobaczmy, jak usunąć elementy ze słownika w następnej sekcji.

Usuwanie elementu ze słownika

Aby usunąć element ze słownika, przypisz nil do istniejącego klucza. Wpisz i uruchom następujące polecenie:

```
contactList["Jane"] = nil

contactList
```

Spowoduje to usunięcie elementu z kluczem „Jane” ze słownika contactList i zawiera on teraz „Shah”: „+60126789345” i „Aamir”: „+0223456789”. Możesz to zobaczyć w obszarze Wyniki. Jeśli chcesz zachować usuwaną wartość, użyj zamiast tego polecenia RemoveValue(for:Key). Wpisz i uruchom następujący kod:

```
var oldDictValue = contactList.removeValue(forKey: "Aamir")

oldDictValue

contactList
```

Spowoduje to usunięcie elementu z kluczem „Aamir” ze słownika contactList i przypisanie jego wartości do oldDictValue. oldDictValue zawiera teraz „+0223456789”, a słownik contactList zawiera „Shah”: „+60126789345”. Możesz to zobaczyć w obszarze Wyniki. Wyobraź sobie, że chcesz zadzwonić do każdego kontaktu i życzyć mu Szczęśliwego Nowego Roku. Będziesz musiał uzyskać dostęp do każdego elementu słownika po kolei i wykonać operacje na każdym elemencie. Zobaczysz, jak to zrobić w następnej sekcji.

Iterowanie po słowniku

Podobnie jak w przypadku tablic, możesz użyć pętli for-in do iteracji po każdym elemencie słownika. Wpisz i uruchom następujący kod:

```
for (name, contactNumber) in contactList {

print("\(name) : \(contactNumber)")

}
```

Spowoduje to wydrukowanie każdego elementu słownika w obszarze debugowania. Ponieważ słowniki są nieuporządkowane, po ponownym uruchomieniu tego kodu możesz otrzymać wyniki w innej kolejności. Teraz wiesz, jak używać słownika do tworzenia nieuporządkowanej listy par klucz-wartość, takiej jak lista kontaktów, i jak wykonywać operacje słownikowe. W następnej sekcji zobaczmy, jak przechowywać nieuporządkowaną listę wartości w zestawie.

Zrozumienie zbiorów

Załóżmy, że piszesz aplikację Filmy i chcesz przechowywać listę gatunków filmowych. Można to zrobić za pomocą zbioru. Zbiór przechowuje wartości na nieuporządkowanej liście. Oto jak to wygląda:

Wartość

Wartość1

Wartość2

Wartość3

Wszystkie wartości są tego samego typu. Jeśli utworzysz zestaw za pomocą słowa kluczowego let, jego zawartości nie będzie można zmienić po jego utworzeniu. Jeśli chcesz zmienić zawartość po utworzeniu, użyj słowa kluczowego var. Przyjrzyjmy się, jak pracować z zestawami. Utworzysz zbiór, przypisując mu wartość w następnej sekcji.

Tworzenie zbioru

Wyobraź sobie, że tworzysz aplikację Filmy i chcesz przechowywać w niej gatunki filmów. Jak widziałeś w przypadku tablic i słowników, możesz utworzyć zbiór, deklarując go i przypisując mu nową wartość. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

```
var movieGenres: Set = ["Horror", "Action", "Romantic Comedy" ]
```

Ta instrukcja tworzy zmienną zestawu o nazwie movieGenres. Należy zauważyć, że przypisany do niego literał zbioru ["Horror", "Akcja", "Komedia romantyczna"] ma taki sam format jak literał tablicowy, dlatego też za pomocą adnotacji typu można ustawić typ movieGenres na Set. W przeciwnym razie wnioskowanie o typie Swifta utworzy zmienną tablicową, a nie zmienną ustawioną. Użycie tutaj słowa kluczowego var oznacza, że zawartość znioru może być modyfikowana. Elementy tego zestawu będą typu String ze względu na wnioskowanie o typie. Wyobraź sobie, że musisz wyświetlić całkowitą liczbę gatunków w swojej aplikacji. Zobaczmy, jak znaleźć liczbę elementów znajdujących się w zbiorze w następnej sekcji.

Sprawdzanie liczby elementów w zbiorze

Aby dowiedzieć się, ile elementów znajduje się w zbiorze, użyj funkcji count. Wpisz i uruchom następujący kod:

```
movieGenres.count
```

Ponieważ zbiór movieGenres zawiera trzy elementy, w obszarze Wyniki zostanie wyświetlona liczba 3. Możesz sprawdzić, czy zestaw jest pusty, używając isEmpty. Wpisz i uruchom następujący kod:

movieGenres.isEmpty

Ponieważ movieGenres zawiera trzy elementy, w obszarze Wyniki wyświetlana jest wartość false. Wyobraź sobie, że użytkownicy Twojej aplikacji mogą dodawać do niej więcej gatunków. Ponieważ ten zbiór został zadeklarowany przy użyciu słowa kluczowego var, możesz dodawać do niego elementy. Dowiesz się, jak to zrobić w następnej sekcji.

Dodanie nowego elementu do zbioru

Możesz dodać nowy element do zestawu, używając wstawki (_:). Wpisz i uruchom następujący kod:

movieGenres.insert("War")

movieGenres

Spowoduje to dodanie nowego elementu „Wojna” do zestawu gatunków filmowych, który zawiera teraz „Horror”, „Komedia romantyczna”, „Wojna” i „Akcja”. Jest to wyświetlane w obszarze Wyniki. Wyobraź sobie, że użytkownik chciałby wiedzieć, czy w Twojej aplikacji dostępny jest określony gatunek. W następnej sekcji dowiesz się, jak sprawdzić, czy element znajduje się w zbiorze.

Sprawdzanie, czy zbiór zawiera element

Aby sprawdzić, czy zbiór zawiera element, użyj funkcji zawiera(_:). Wpisz i uruchom następujący kod:

movieGenres.contains("War")

Ponieważ „Wojna” jest jednym z elementów zestawu movieGenres, prawda jest wyświetlana w obszarze Wyniki. Wyobraź sobie, że użytkownik chce usunąć gatunek ze swojej listy gatunków. Zobaczmy, jak usunąć elementy z zestawu, które nie są już potrzebne, w następnej sekcji.

Usuwanie elementu ze zbioru

Aby usunąć element z zestawu, użyj polecenia usuń(_:). Usuwaną wartość można przypisać do zmiennej lub stałej. Jeśli wartość nie istnieje w zestawie, zwrócone zostanie nil. Wpisz i uruchom następujący kod:

var oldSetValue = movieGenres.remove("Action")

oldSetValue

movieGenres

Pozycja „Akcja” została usunięta z zestawu movieGenres i przypisana do oldSetValue, a zestaw movieGenres zawiera teraz „Horror”, „Komedia romantyczna” i „Wojna”. Zobaczysz to w obszarze Wyniki. Aby usunąć wszystkie elementy ze zbioru, użyj funkcji RemoveAll(). Wyobraź sobie, że chcesz wyświetlić wszystkie gatunki dostępne w Twojej aplikacji jako rekomendacje dla jej użytkowników. Możesz iterować i wykonywać operacje na każdym elemencie zestawu. Zobaczmy, jak to zrobić w następnej sekcji.

Iterowanie po zbiorze

Podobnie jak w przypadku tablic i słowników, możesz użyć pętli for-in do iteracji po każdym elemencie w zestawie. Wpisz i uruchom następujący kod:

for genre in movieGenres {

print(genre)

}

Powinieneś zobaczyć każdy ustawiony element w obszarze Debugowanie. Ponieważ zestawy są nieposortowane, po ponownym uruchomieniu tego kodu możesz otrzymać wyniki w innej kolejności. Wyobraź sobie, że chcesz, aby Twoja aplikacja wykonywała operacje na gatunkach, które lubisz, z gatunkami, które lubi inna osoba. W następnej sekcji dowiesz się o różnych operacjach, które możesz wykonać na zbiorach w Swift.

Odkrywanie operacji na zbiorach

Łatwo jest wykonywać operacje na zbiorach, takie jak suma, przecięcie, odejmowanie i różnica symetryczna. Wpisz i uruchom następujący kod:

```
let movieGenres2: Set = ["Science Fiction", "War", "Fantasy"]

movieGenres.union(movieGenres2)

movieGenres.intersection(movieGenres2)

movieGenres.subtracting(movieGenres2)

movieGenres.symmetricDifference(movieGenres2)
```

Tutaj wykonujesz operacje na zestawach na dwóch zestawach: movieGenres i movieGenres2. Zobaczmy wyniki każdej operacji na zestawie:

• union(_:) zwraca nowy zestaw zawierający wszystkie wartości z obu zestawów, więc w obszarze Wyniki zostanie wyświetlony {"Horror", "Komedia romantyczna", "Wojna", "Science Fiction", "Fantasy"} .

• skrzyżowanie(_:) zwraca nowy zestaw zawierający tylko wartości wspólne dla obu zbiorów, więc w obszarze Wyniki zostanie wyświetlona opcja {"War"}.

• odejmowanie(_:) zwraca nowy zestaw bez wartości z określonego zestawu, więc w obszarze Wyniki wyświetli się {"Horror", "Komedia romantyczna"}.

• symmetricDifference(_:) zwraca nowy zestaw bez wartości wspólnych dla obu zbiorów, więc w obszarze Wyniki zostanie wyświetlony element {"Horror", "Komedia romantyczna", "Science Fiction", "Fantasy"}.

Wyobraź sobie, że chcesz, aby Twoja aplikacja porównywała gatunki, które lubisz, z gatunkami, które lubi inna osoba. W następnej sekcji dowiesz się, jak sprawdzić, czy zbiór jest równy innemu zbiorowi, jest częścią innego zbioru lub nie ma nic wspólnego z innym zbiorem.

Badanie członkostwa w zbiorze i równości

Łatwo jest sprawdzić, czy zbiór jest równy podzbiorowi, nadzbiorowi lub rozłącznemu z innego zbioru. Wpisz i uruchom następujący kod:let movieGenresSubset: Set = ["Horror", "Romantic Comedy"]

```
let movieGenresSuperset: Set = ["Horror", "Romantic Comedy", "War", "Science

Fiction", "Fantasy"]

let movieGenresDisjoint: Set = ["Bollywood"]

movieGenres == movieGenres2
```

movieGenresSubset.isSubset(of: movieGenres)

movieGenresSuperset.isSuperset(of: movieGenres)

movieGenresDisjoint.isDisjoint(with: movieGenres)

Zobaczmy jak działa ten kod:

• Operator isEqual (==) sprawdza, czy wszystkie elementy jednego zbioru są takie same jak elementy innego zbioru. Ponieważ nie wszystkie elementy zestawu movieGenres są takie same jak te w zestawie movieGenres2, w obszarze Wyniki zostanie wyświetlona wartość false.

• isSubset(of:) sprawdza, czy zbiór jest podzbiorem innego zbioru. Ponieważ wszyscy członkowie zestawu movieGenresSubset znajdują się w zestawie movieGenres, w obszarze Wyniki zostanie wyświetlona wartość true.

• isSuperset(of:) sprawdza, czy zbiór jest nadzbiorem innego zbioru. Ponieważ wszyscy członkowie zestawu movieGenres znajdują się w zestawie movieGenresSuperset, w obszarze Wyniki zostanie wyświetlona wartość true.

• isDisjoint(with:) sprawdza, czy zbiór nie ma wspólnych wartości z innym zbiorem. Ponieważ zestaw movieGenresDisjoint nie ma wspólnych elementów z zestawem movieGenres, w obszarze Wyniki zostanie wyświetlona wartość true.

Wiesz już, jak używać zestawu do tworzenia nieuporządkowanej listy wartości, takiej jak lista gatunków filmowych, i jak wykonywać operacje na zbiorach. Na tym kończy się rozdział dotyczący typów kolekcji. Dobrze zrobiony!

Streszczenie

W tej części przyjrzeliśmy się typom kolekcji w Swift. Najpierw dowiedziałeś się o tablicach. Umożliwiają one użycie uporządkowanej listy wartości do reprezentowania elementu takiego jak lista zakupów i wykonywanie na nim operacji. Następnie dowiedziałeś się o słownikach. Dzięki temu można używać nieuporządkowanej listy par klucz-wartość do reprezentowania elementu takiego jak lista kontaktów i wykonywać na nim operacje. W końcu dowiedziałeś się o zestawach. Dzięki temu można używać nieuporządkowanej listy wartości do reprezentowania elementu takiego jak lista gatunków filmowych i wykonywać na nim operacje. W następnym rozdziale dowiesz się, jak grupować zestaw instrukcji za pomocą funkcji. Jest to przydatne, gdy chcesz wielokrotnie wykonać zestaw instrukcji w swoim programie.

Funkcje i zamknięcia

W tym momencie możesz napisać dość złożone programy, które będą mogły podejmować decyzje i powtarzać sekwencje instrukcji. Możesz także przechowywać dane dla swoich programów, korzystając z typów kolekcji. W miarę jak programy, które piszesz, stają się coraz większe i bardziej złożone, coraz trudniej jest zrozumieć, co robią. Aby ułatwić zrozumienie dużych programów, Swift umożliwia tworzenie funkcji, które pozwalają łączyć ze sobą wiele instrukcji i wykonywać je poprzez wywołanie jednej nazwy. Można także tworzyć domknięcia, co pozwala połączyć ze sobą kilka instrukcji bez nazwy i przypisać je do stałej lub zmiennej. Pod koniec tego rozdziału dowiesz się o funkcjach, funkcjach zagnieżdżonych, funkcjach jako typach zwracanych, funkcjach jako argumentach i instrukcji Guard. Dowiesz się także, jak tworzyć i używać zamknięć.

Zrozumienie funkcji

Funkcje są przydatne do hermetyzacji szeregu instrukcji, które łącznie wykonują określone zadanie, na przykład:

• Naliczenie 10% opłaty za obsługę posiłku w restauracji.

• Obliczanie miesięcznej płatności za samochód, który chcesz kupić.

Oto jak wygląda funkcja:

```
func nazwa funkcji (parametr 1: Typ parametru, ...) -> Typ zwrotu {

kod

}
```

Każda funkcja ma opisową nazwę. Można zdefiniować jedną lub więcej wartości, które funkcja przyjmuje jako dane wejściowe, zwanych parametrami. Możesz także zdefiniować, co funkcja wyświetli po zakończeniu, co jest znane jako typ zwracany. Zarówno parametry, jak i typy zwracane są opcjonalne. „Wywołujesz” nazwę funkcji, aby ją wykonać. Tak wygląda wywołanie funkcji:

```
nazwafunkcji(parametr1: argument1, )
```

Podajesz wartości wejściowe (tzw. argumenty), które odpowiadają typowi parametrów funkcji.

Tworzenie funkcji

W najprostszej formie funkcja po prostu wykonuje pewne instrukcje i nie ma żadnych parametrów ani typów zwracanych wartości. Jak to działa, przekonasz się pisząc funkcję obliczającą opłatę za usługę za posiłek. Opłata za obsługę powinna wynosić 10% kosztu posiłku. Dodaj następujący kod do swojego placu zabaw, aby utworzyć i wywołać tę funkcję, a następnie kliknij przycisk Odtwórz/Zatrzymaj, aby ją uruchomić:

```
func serviceCharge() {

let mealCost = 50

let serviceCharge = mealCost / 10

print("Service charge is \(serviceCharge)")

}

serviceCharge()
```

Właśnie utworzyłeś bardzo prostą funkcję o nazwie serviceCharge(). Jedyne, co robi, to oblicza 10% opłaty za obsługę za posiłek kosztujący 50 dolarów, czyli 50/10, zwracając 5. Następnie wywołujesz tę funkcję, używając jej nazwy. W obszarze debugowania zobaczysz, że opłata za usługę wynosi 5. Ta funkcja nie jest zbyt użyteczna, ponieważ koszt posiłku wynosi zawsze 50 przy każdym wywołaniu tej funkcji, a wynik 5 jest drukowany tylko w obszarze Debugowanie i nie można go użyć w innym miejscu programu. Dodajmy do tej funkcji kilka parametrów i typ zwracany, aby była bardziej użyteczna. Zmodyfikuj swój kod, jak pokazano:

```
func serviceCharge(mealCost: Int) -> Int {

return mealCost / 10

}

let serviceChargeAmount = serviceCharge(mealCost: 50)

print(serviceChargeAmount)
```

To jest dużo lepsze. Teraz możesz ustawić koszt posiłku wywołując funkcję serviceCharge(mealCost:) a wynik można przypisać do zmiennej lub stałej. Jednak wygląda to trochę niezręcznie. Powinieneś spróbować sprawić, aby podpisy funkcji w Swift były czytane jak zdanie w języku angielskim, ponieważ jest to uważane za najlepszą praktykę. Zobaczmy, jak to zrobić w następnej sekcji, w której użyjesz niestandardowych etykiet, aby Twoja funkcja była bardziej angielska i łatwiejsza do zrozumienia.

Używanie niestandardowych etykiet argumentów

Należy pamiętać, że funkcja serviceCharge(mealCost:) nie jest zbyt angielska. Aby ułatwić zrozumienie funkcji, można dodać do parametru niestandardową etykietę. Zmodyfikuj swój kod, jak pokazano:

```
func serviceCharge(forMealPrice mealCost: Int) -> Int {

return mealCost / 10>

}

let serviceChargeAmount = serviceCharge(forMealPrice: 50)

print(serviceChargeAmount)
```

Funkcja działa dokładnie tak samo jak poprzednio, z tą różnicą, że do jej wywołania służy serviceCharge(forMealPrice:). Brzmi to bardziej jak angielski i ułatwia zrozumienie, do czego służy dana funkcja. W następnej sekcji dowiesz się, jak używać kilku mniejszych funkcji w treściach innych funkcji i są one nazywane funkcjami zagnieżdżonymi.

Korzystanie z funkcji zagnieżdżonych

Możliwe jest posiadanie funkcji w ciele innej funkcji i nazywa się to funkcjami zagnieżdżonymi. Funkcja zagnieżdżona może używać zmiennych funkcji otaczającej. Zobaczmy jak działają funkcje zagnieżdżone pisząc funkcję obliczającą miesięczne raty kredytu. Wpisz i uruchom następujący kod:

```
func calculateMonthlyPayments(carPrice: Double, downPayment: Double,

interestRate: Double, paymentTerm: Double) -> Double {

func loanAmount() -> Double {

return carPrice - downPayment
```

```
}

func totalInterest() -> Double {

return interestRate * paymentTerm

}

func numberOfMonths() -> Double {

return paymentTerm * 12

}

return ((loanAmount() + ( loanAmount() *

totalInterest() / 100 )) / numberOfMonths())

}

calculateMonthlyPayments(carPrice: 50000, downPayment: 5000, interestRate: 3.5,

paymentTerm: 7.0)
```

Tutaj istnieją trzy funkcje w ramach CalcMonthlyPayments(carPrice:downPayment:interestRate:paymentTerm:). Przyjrzyjmy się im:

• Pierwsza zagnieżdżona funkcja LoanAmount() oblicza całkowitą kwotę pożyczki odejmując downPayment od carPrice. Zwraca 50000 - 5000 = 45000.

• Druga zagnieżdżona funkcja, totalInterest(), oblicza całkowitą kwotę odsetek naliczonych za okres płatności poprzez pomnożenie stopy procentowej przez termin płatności. Zwraca 3,5 * 7 = 24,5.

• Trzecia zagnieżdżona funkcja, numberOfMonths(), oblicza całkowitą liczbę miesięcy w okresie płatności, mnożąc PaymentTerm przez 12. Zwraca 7 * 12 = 84.

Należy zauważyć, że wszystkie trzy funkcje zagnieżdżone korzystają ze zmiennych funkcji otaczającej. Zwracana wartość to (45000 + (45000 * 24,5 / 100)) / 84 = 666,96, czyli kwota, którą musisz płacić co miesiąc przez 7 lat na zakup tego samochodu. Jak widzieliście, funkcje w Swift są podobne do funkcji w innych językach, ale mają fajną funkcję. Funkcje są typami pierwszej klasy w języku Swift, dlatego można ich używać jako parametrów i typów zwracanych. Zobaczmy, jak to się robi w następnej sekcji.

Używanie funkcji jako typów zwracanych

Funkcja może zwracać inną funkcję jako typ zwracany. Wpisz i uruchom następujący kod, aby utworzyć funkcję generującą wartość Pi:

```
func makePi() -> (() -> Double) {

func generatePi() -> Double {

return 22.0 / 7.0

}
```



```
return generatePi
```

```
}

let pi = makePi()

print(pi())
```

Typ zwracany przez funkcję makePi() to funkcja, która nie ma parametrów, a typem zwracanym jest Double. generatePi() to funkcja, która nie ma parametrów, a zwracany typ to Double i będzie to zwracana funkcja. Zatem do pi zostanie przypisany generatePi() i po wywołaniu zwróci 22,0/7,0. Numer 3.142857142857143 zostanie wydrukowany w obszarze debugowania. Zobaczmy, jak funkcja może zostać użyta jako parametr innej funkcji w następnej sekcji.

Używanie funkcji jako parametrów

Funkcja może przyjmować funkcję jako parametr. Wpisz i uruchom następujący kod, aby utworzyć funkcję sprawdzającą, czy na liście liczb istnieje liczba spełniająca określony warunek:

```
func isThereAMatch(listOfNumbers: [Int], warunek: (Int) -> Bool) -> Bool {

dla elementu na liścieOfNumbers {

func isThereAMatch(listOfNumbers: [Int], condition: (Int) -> Bool) -> Bool {

for item in listOfNumbers {

if condition(item) {

return true

}

}

return false

}

func oddNumber(number: Int) -> Bool {

return (number % 2) > 0

}

let numbersList = [2, 4, 6, 7]

isThereAMatch(listOfNumbers: numbersList, condition: oddNumber)
```

isThereAMatch(listOfNumbers:condition:) ma dwa parametry: tablicę liczb całkowitych i funkcję. Funkcja podana jako argument musi przyjmować wartość całkowitą i zwracać wartość logiczną. oddNumber(liczba:) przyjmuje liczbę całkowitą i zwraca wartość true, jeśli liczba jest liczbą nieparzystą, co oznacza, że może być argumentem dla drugiego parametru. NumberList, tablica zawierająca liczbę nieparzystą, jest używana jako argument pierwszego parametru. Ponieważ numberList zawiera liczbę nieparzystą, isTher eAMatch(listOfNumbers:condition:) po wywołaniu zwróci wartość true. W następnej sekcji zobaczysz, jak możesz wcześniej zakończyć funkcję, jeśli użyte argumenty nie są odpowiednie.

Zrozumienie zamknięć

Zamknięcie, podobnie jak funkcja, zawiera sekwencję instrukcji i może przyjmować argumenty i zwracać wartości. Zamknięcia nie mają jednak nazw. Sekwencja instrukcji w zamknięciu jest otoczona nawiasami klamrowymi ({ }), a słowo kluczowe in oddziela argumenty i typ zwracany od treści zamknięcia. Zamknięcia można przypisać do stałej lub zmiennej, są więc przydatne, jeśli trzeba je przekazać w programie. Załóżmy na przykład, że masz aplikację, która pobiera plik z Internetu i musisz coś zrobić z plikiem po zakończeniu pobierania. Możesz umieścić listę instrukcji przetwarzania pliku w zamknięciu i pozwolić programowi wykonać je po zakończeniu pobierania pliku.

Napiszesz teraz zamknięcie, które zastosuje obliczenia do każdego elementu tablicy liczb. Dodaj następujący kod do swojego placu zabaw i kliknij przycisk Odtwórz/Zatrzymaj, aby go uruchomić:

```
var numbersArray = [2, 4, 6, 7]

let myClosure = { (number: Int) -> Int in

let result = number * number

return result

}

let mappedNumbers = numbersArray.map(myClosure)

niech mappedNumbers = NumberArray.map(myClosure)
```

Spowoduje to przypisanie domknięcia, które oblicza potęgę dwójki liczby, do myClosure. Następnie funkcja map() stosuje to zamknięcie do każdego elementu w NumberArray. Każdy element jest mnożony przez siebie, a w obszarze Wyniki pojawia się [4, 16, 36, 49]. Zamknięcia można pisać w bardziej zwięzły sposób. W następnej sekcji zobaczysz, jak to zrobić.

Upraszczanie zamknięć

Jedną z rzeczy, z którymi nowi programiści mają problemy, jest bardzo zwięzła metoda, której doświadczeni programiści Swift używają do pisania domknięć. Rozważmy kod pokazany w poniższym przykładzie:

```
var testNumbers = [2, 4, 6, 7]

let mappedTestNumbers = testNumbers.map({ (number: Int)

-> Int in

let result = number * number

return result

})

print(mappedTestNumbers)
```

Tutaj masz testNumbers, tablicę liczb i używasz funkcji map(_:) do mapowania zamknięcia na każdy element tablicy po kolei. Kod w zamknięciu mnoży liczbę przez siebie, generując kwadrat tej liczby. Wynik [4, 16, 36, 49] jest następnie drukowany w obszarze debugowania. Jak zobaczysz, kod zamknięcia można zapisać bardziej zwięźle. Gdy typ zamknięcia jest już znany, możesz usunąć typ parametru, typ zwracany lub oba. Zamknięcia pojedynczymi instrukcjami domyślnie zwracają wartość

swojej jedynej instrukcji, co oznacza, że możesz również usunąć instrukcję return. Można więc zapisać zamknięcie w następujący sposób:

```
let mappedTestNumbers = testNumbers.map({ number in

number * number

})
```

Jeśli zamknięcie jest jedynym argumentem funkcji, można pominąć nawiasy otaczające zamknięcie w następujący sposób:

```
let mappedTestNumbers = testNumbers.map { number in

number * number

}
```

Do parametrów można odwoływać się za pomocą liczby wyrażającej ich względną pozycję na liście argumentów, zamiast według nazwy, w następujący sposób:

```
let mappedTestNumbers = testNumbers.map { $0 * $0 }
```

Zatem zamknięcie jest obecnie bardzo zwięzłe, ale dla nowych programistów będzie trudne do zrozumienia. Możesz pisać zamknięcia w sposób, który Ci odpowiada. Teraz wiesz, jak tworzyć i używać domknięć oraz jak pisać je bardziej zwięźle. Świetnie

Streszczenie

W tej części nauczyłeś się grupować instrukcje w funkcje. Nauczyłeś się, jak używać niestandardowych etykiet argumentów, funkcji wewnątrz innych funkcji, funkcji jako typów zwracanych i funkcji jako parametrów. Przyda się to później, gdy będziesz musiał wykonać to samo zadanie w różnych punktach programu. Nauczyłeś się także, jak tworzyć zamknięcia. Będzie to przydatne, gdy będziesz musiał przekazywać bloki kodu w swoim programie. W następnym rozdziale zajmiemy się klasami, strukturami i wyliczeniami. Klasy i struktury pozwalają na tworzenie złożonych obiektów, które mogą przechowywać stan i zachowanie, a wyliczenia mogą służyć do ograniczania wartości, które można przypisać do zmiennej lub stałej, zmniejszając ryzyko błędu.

Klasy, struktury i wyliczenia

W poprzedniej części nauczyłeś się grupować sekwencje instrukcji za pomocą funkcji i domknięć. Czas pomyśleć o tym, jak reprezentować złożone obiekty w kodzie. Pomyśl na przykład o samochodzie. Możesz użyć stałej String do przechowywania nazwy samochodu i zmiennej Double do przechowywania ceny samochodu, ale nie są one ze sobą powiązane. Widziałeś, że możesz grupować instrukcje, aby tworzyć funkcje i domknięcia. W tej części dowiesz się, jak grupować stałe i zmienne w jedną całość przy użyciu klas i struktur oraz jak nimi manipulować. Dowiesz się także, jak używać wyliczeń do grupowania zestawu powiązanych wartości. Pod koniec tego rozdziału dowiesz się, jak tworzyć i inicjować klasę, tworzyć podklasę z istniejącej klasy, tworzyć i inicjować strukturę, rozróżniać klasy od struktur oraz tworzyć wyliczenia.

Zrozumienie klas

Klasy są przydatne do reprezentowania złożonych obiektów, na przykład:

• Indywidualne informacje o pracownikach firmy

• Przedmioty na sprzedaż w witrynie handlu elektronicznego

• Przedmioty, które masz w domu dla celów ubezpieczenia

Oto jak wygląda deklaracja i definicja klasy:

```
class ClassName {

property1

property2

property3

method1() {

code

}

method2() {

code

}

}

}
```

Każda klasa ma opisową nazwę i zawiera zmienne lub stałe używane do reprezentowania obiektu. Zmienne lub stałe powiązane z klasą nazywane są właściwościami. Klasa może również zawierać funkcje wykonujące określone zadania. Funkcje powiązane z klasą nazywane są metodami. Po zadeklarowaniu i zdefiniowaniu klasy można tworzyć instancje tej klasy. Wyobraź sobie, że tworzysz aplikację dla zoo. Jeśli masz klasę Animal, możesz używać instancji tej klasy do reprezentowania różnych typów zwierząt w zoo. Każde z tych wystąpień będzie miało inne wartości swoich właściwości. przyjrzyjmy się, jak pracować z klasami. Dowiesz się, jak deklarować i definiować klasy, tworzyć instancje na podstawie deklaracji klasy oraz manipulować tymi instancjami. W następnej sekcji zaczniesz od utworzenia deklaracji klasy reprezentującej zwierzęta.

Tworzenie deklaracji klasy

Zadeklarujmy i zdefiniujmy klasę, która może przechowywać szczegółowe informacje o zwierzętach. Dodaj następujący kod do swojego placu zabaw:

```
class Animal {

var name: String = ""

var sound: String = ""

var numberOfLegs: Int = 0

var breathesOxygen: Bool = true

func makeSound() {

print(self.sound)

}

}
```

Właśnie zadeklarowałeś bardzo prostą klasę o nazwie Animal. Konwencja stanowi, że nazwy klas zaczynają się wielką literą. Ta klasa ma właściwości przechowujące nazwę zwierzęcia, wydawany przez niego dźwięk, liczbę nóg i to, czy oddycha tlenem, czy nie. Klasa ta posiada również metodę makeSound(), która wypisuje wytwarzany przez siebie szum w obszarze debugowania. Teraz, gdy masz klasę Animal, użyjmy jej do utworzenia instancji zwierzęcia w następnej sekcji.

Tworzenie instancji klasy

Po zadeklarowaniu i zdefiniowaniu klasy można tworzyć instancje tej klasy. Utworzysz teraz instancję klasy Animal reprezentującą kota. Wykonaj następujące kroki:

1. Aby utworzyć instancję klasy Animal, wypisz wszystkie jej właściwości i wywołaj metodę makeSound(), po deklaracji klasy wpisz następujące polecenie i uruchom ją:

```
let cat = Animal()

print(cat.name)

print(cat.sound)

print(cat.numberOfLegs)

print(cat.breathesOxygen)

cat.makeSound()
```

Dostęp do właściwości i metod instancji można uzyskać, wpisując kropkę po nazwie instancji, a następnie żądaną właściwość lub metodę. Zobaczysz wartości właściwości instancji i wywołań metod wymienionych w obszarze Debugowanie. Ponieważ wartości są wartościami domyślnymi przypisanymi podczas tworzenia klasy, nazwa i dźwięk zawierają puste ciągi znaków, liczbaOfLegs zawiera 0, BreathsOxygen zawiera wartość true, a metoda makeSound() wyświetla pusty ciąg.

2. Przypiszmy pewne wartości właściwościom tej instancji. Zmodyfikuj swój kod, jak pokazano:

```
let cat = Animal()
```

```
cat.name = "Cat"

cat.sound = "Mew"

cat.numberOfLegs = 4

cat.breathesOxygen = true

print(cat.name)
```

Teraz po uruchomieniu programu w obszarze Debugowanie zostanie wyświetlony następujący komunikat:

```
Cat

Mew

4

true

Mew
```

Wartości wszystkich właściwości instancji oraz wynik metody makeSound() są drukowane w obszarze Debugowanie. Pamiętaj, że tutaj najpierw tworzysz instancję, a następnie przypisujesz wartości do tej instancji. Możliwe jest również przypisanie wartości podczas tworzenia instancji, a robisz to poprzez implementację inicjatora w deklaracji klasy.

3. Inicjator jest odpowiedzialny za zapewnienie, że wszystkie właściwości instancji mają prawidłowe wartości podczas tworzenia klasy. Dodajmy inicjator dla klasy Animal. Zmodyfikuj definicję klasy, jak pokazano:

```
class Animal {

var name: String

var sound: String

var numberOfLegs: Int

var breathesOxygen: Bool

init(name: String, sound: String, numberOfLegs:

Int, breathesOxygen: Bool) {

self.name = name

self.sound = sound

self.numberOfLegs = numberOfLegs

self.breathesOxygen = breathesOxygen

}

func makeSound() {

print(self.sound)
```

}

}

Jak widać, inicjator używa słowa kluczowego init i zawiera listę parametrów, które zostaną użyte do ustawienia wartości właściwości. Należy zauważyć, że słowo kluczowe self odróżnia nazwy właściwości od parametrów. Na przykład self.name odnosi się do właściwości, a nazwa odnosi się do parametru. Na końcu procesu inicjalizacji każda właściwość w klasie powinna mieć prawidłową wartość.

4. W tym momencie zobaczysz kilka błędów w swoim kodzie. Aby rozwiązać ten problem, musisz zaktualizować wywołanie funkcji. Zmodyfikuj swój kod, jak pokazano, i uruchom go:

func makeSound() {

print(self.sound)

}

}

let cat = Animal(name: "Cat", sound: "Mew",

numberOfLegs: 4, breathesOxygen: true)

print(cat.name)

Wyniki są takie same jak w kroku 2, ale utworzyłeś instancję i ustawiłeś jej właściwości w jednej instrukcji. Doskonały! Obecnie istnieją różne rodzaje zwierząt, takie jak ssaki, ptaki, gady i ryby. Można utworzyć klasę dla każdego typu, ale można także utworzyć podklasę w oparciu o istniejącą klasę. Zobaczmy, jak to zrobić w następnej sekcji.

Tworzenie podklasy

Podklasa klasy dziedziczy wszystkie metody i właściwości istniejącej klasy. Jeśli chcesz, możesz także dodać do niego dodatkowe właściwości i metody. Utworzysz teraz Mammal, podklasę klasy Animal. Wykonaj następujące kroki:

1. Aby zadeklarować klasę Mammal, wpisz następujący kod zaraz po deklaracji klasy Animal:

class Mammal: Animal {

let hasFurOrHair: Bool = true

}

Wpisanie : Animal po nazwie klasy powoduje, że klasa Mammal staje się podklasą klasy Animal. Posiada wszystkie właściwości i metody zadeklarowane w klasie Animal oraz jedną dodatkową właściwość hasFurOrHair. Ponieważ klasa Animal jest rodzicem klasy Mammal, można ją nazwać nadklasą klasy Mammal.

2. Zmodyfikuj kod tworzący instancję klasy, jak pokazano, i uruchom go:

let cat = Mammal(name: "Cat", sound: "Mew",

numberOfLegs: 4, breathesOxygen: true)

cat jest teraz instancją klasy Mammal zamiast klasy Animal. Jak widać wyniki wyświetlane w obszarze Debugowanie są takie same jak poprzednio i nie ma żadnych błędów. Jednak wartość hasFurOrHair nie została wyświetlona. Naprawmy to.

3. Wpisz następujący kod po całym innym kodzie na swoim placu zabaw, aby wyświetlić zawartość właściwości hasFurOrHair i uruchomić ją:

print(cat.hasFurOrHair)

Ponieważ inicjator klasy Animal nie ma parametru umożliwiającego przypisanie wartości do hasFurOrHair, używana jest wartość domyślna, a w obszarze Debugowanie zostanie wyświetlona wartość true. Widziałeś, że podklasa może mieć dodatkowe właściwości. Podklasa może również posiadać dodatkowe metody, a implementacja metod w podklasie może różnić się od implementacji nadklasy. Zobaczmy, jak to zrobić w następnej sekcji.

Zastępowanie metody nadklasy

Do tej pory używałeś wielu instrukcji print() do wyświetlania wartości instancji klasy. Zaimplementujesz metodę opisu(), aby wyświetlić wszystkie właściwości instancji w obszarze debugowania, dzięki czemu wiele instrukcji print() nie będzie już wymaganych. Wykonaj następujące kroki:

1. Zmodyfikuj deklarację klasy Animal, aby zaimplementować metodę description(), jak pokazano:

Do tej pory używałeś wielu instrukcji print() do wyświetlania wartości instancji klasy. Zaimplementujesz metodę opisu(), aby wyświetlić wszystkie właściwości instancji w obszarze debugowania, dzięki czemu wiele instrukcji print() nie będzie już wymaganych. Wykonaj następujące kroki:

1. Zmodyfikuj deklarację klasy Animal, aby zaimplementować metodę opisu(), jak pokazano:

```
class Animal {

var name: String

var sound: String

var numberOfLegs: Int

var breathesOxygen: Bool = true

init(name: String, sound: String, numberOfLegs:

Int, breathesOxygen: Bool) {

self.name = name

self.sound = sound

self.numberOfLegs = numberOfLegs

self.breathesOxygen = breathesOxygen

}

func makeSound() {

print(self.sound)

}
```

```
func description() -> String {
return "name: \(self.name)
sound: \(self.sound)
numberOfLegs: \(self.numberOfLegs)
breathesOxygen: \(self.breathesOxygen)"
}
}
```

2. Zmodyfikuj kod zgodnie z ilustracją, aby użyć metody opis() zamiast wielu instrukcji print() i uruchom program:

```
let cat = Mammal(name: "Cat", sound: "Mew",
numberOfLegs: 4, breathesOxygen: true)
print(cat.description())
cat.makeSound()
```

W obszarze Debugowanie zobaczysz następujące informacje:

name: Cat sound: Mew numberOfLegs: 4 breathesOxygen: true

Mew

Jak widać, mimo że metoda opis() nie jest zaimplementowana w klasie Mammal, to jest ona zaimplementowana w klasie Animal. Oznacza to, że zostanie odziedziczona przez klasę Mammal, a właściwości instancji zostaną wydrukowane w obszarze Debug. Należy zwrócić uwagę, że brakuje wartości właściwości hasFurOrHair i nie można jej umieścić w metodzie opis(), ponieważ właściwość hasFurOrHair nie istnieje dla klasy Animal.

3. Możesz zmienić implementację metody opis() w klasie Mammal, aby wyświetlała wartość właściwości hasFurOrHair. Dodaj następujący kod do definicji klasy Mammal i uruchom go:

```
Mammal: Animal {
let hasFurOrHair: Bool = true
override func description() -> String {
return super.description() + " hasFurOrHair:
\(self.hasFurOrHair)"
}
}
```

Słowo kluczowe override służy tutaj do określenia, że zaimplementowana tutaj metoda opisu () ma być używana zamiast implementacji nadklasy. Słowo kluczowe super służy do wywołania implementacji

opisu() nadklasy. Wartość hasFurOrHair jest następnie dodawana do ciągu zwracanego przez super.description(). W obszarze Debugowanie zobaczysz następujące informacje:

name: Cat sound: Mew numberOfLegs: 4 breathesOxygen: true hasFurOrHair:

true

Mew

Wartość właściwości hasFurOrHair jest wyświetlana w obszarze Debugowanie, co oznacza, że korzystasz z implementacji podklasy Mammal metody opisu(). Utworzyłeś deklaracje klas i podklas oraz utworzyłeś instancje obu. Do obu dodałeś także inicjatory i metody. Fajny! Przyjrzyjmy się, jak deklarować i używać struktur w następnej sekcji.

Zrozumienie struktur

Podobnie jak klasy, struktury również grupują właściwości i metody używane do reprezentowania obiektu i wykonywania określonych zadań. Pamiętasz klasę Animal, którą stworzyłeś? Możesz także użyć struktury, aby osiągnąć to samo. Istnieją jednak różnice pomiędzy klasami i strukturami, o których dowiesz się więcej w dalszej części tej części. Oto jak wygląda deklaracja i definicja struktury:

struct StructName {

property1

property2

property3

method1() {

code

}

method2(){

code

}

}

Jak widać, struktura jest bardzo podobna do klasy. Ma również opisową nazwę, może zawierać właściwości i metody oraz umożliwia tworzenie instancji. Przyjrzyjmy się, jak pracować ze strukturami. Dowiesz się, jak deklarować i definiować struktury, tworzyć instancje w oparciu o strukturę i manipulować nimi. W następnej sekcji zaczniesz od utworzenia struktury reprezentującej gady.

Tworzenie deklaracji struktury

Kontynuując temat zwierząt, zadeklarujmy i zdefiniujmy strukturę, która może przechowywać szczegóły gadów. Dodaj następujący kod po wszystkich innych kodach na swoim placu zabaw:

struct Reptile {

var name: String

var sound: String

```
var numberOfLegs: Int

var breathesOxygen: Bool

let hasFurOrHair: Bool = false

func makeSound() {

print(sound)

}

func description() -> String {

return "Structure: Reptile name: \(self.name)

sound: \(self.sound)

numberOfLegs: \(self.numberOfLegs)

breathesOxygen: \(self.breathesOxygen)

hasFurOrHair: \(self.hasFurOrHair)"

}

}
```

Jak widać, jest to prawie to samo, co wcześniejsza deklaracja klasy Animal. Nazwy struktur również zaczynają się od dużej litery, a struktura ta ma właściwości przechowujące nazwę zwierzęcia, dźwięk, jaki wydaje, liczbę nóg, to, czy oddycha tlenem i czy ma futro czy sierść. Struktura ta posiada również metodę makeSound(), która wypisuje wydawany przez siebie dźwięk w obszarze debugowania. Teraz, gdy masz już deklarację struktury Reptile, użyjmy jej do utworzenia instancji reprezentującej węża w następnej sekcji.

Tworzenie instancji konstrukcji

Podobnie jak w przypadku klas, możesz tworzyć instancje na podstawie deklaracji struktury. Utworzysz teraz instancję struktury Reptile reprezentującą węża, wydrukujesz wartości właściwości tej instancji i wywołasz metodę makeSound(). Wpisz następujące polecenie po deklaracji struktury i uruchom ją:

```
var snake = Reptile(name: "Snake", sound: "Hiss",

numberOfLegs: 0, breathesOxygen: true)

print(snake.description())

snake.makeSound()
```

Zauważ, że nie było potrzeby implementowania inicjatora; Struktury automatycznie otrzymują inicjator dla wszystkich swoich właściwości, zwany inicjatorem członkowskim. Schludny! Poniższe informacje zostaną wyświetlone w obszarze debugowania:

Struktura: Nazwa gada: Dźwięk węża: Liczba syknięć: 0 oddychaTlen: prawda

hasFurOrHair: fałsz

Syk

Mimo że deklaracja struktury jest bardzo podobna do deklaracji klasy, istnieją dwie różnice między klasą a strukturą:

• Struktury nie mogą dziedziczyć z innej struktury

• Klasy są typami referencyjnymi, natomiast struktury są typami wartościowymi

Przyjrzyjmy się różnicy między typami wartościowymi a typami referencyjnymi w następnej sekcji.

Porównywanie typów wartości i typów referencyjnych

Klasy są typami referencyjnymi. Oznacza to, że kiedy przypiszesz instancję klasy do zmiennej, w rzeczywistości przechowujesz lokalizację pamięci oryginalnej instancji w zmiennej, a nie samą instancję. Struktury są typami wartości. Oznacza to, że kiedy przypiszesz instancję struktury do zmiennej, instancja ta zostanie skopiowana i wszelkie zmiany wprowadzone w oryginalnej instancji nie będą miały wpływu na kopię. Teraz utworzysz instancję klasy i struktury i zaobserwujesz różnice między nimi. Wykonaj następujące kroki:

1. Zaczniesz od utworzenia zmiennej zawierającej instancję struktury i przypisania jej do drugiej zmiennej, a następnie zmienisz wartość właściwości w drugiej zmiennej. Wpisz następujący kod i uruchom go:

```
struct SampleValueType {

var sampleProperty = 10

}

var a = SampleValueType()

var b = a

b.sampleProperty = 20

print(a.sampleProperty)

print(b.sampleProperty)
```

W tym przykładzie zadeklarowano strukturę SampleValueType, która zawiera jedną właściwość sampleProperty. Następnie utworzyłeś instancję tej struktury i przypisałeś ją do zmiennej, a. Następnie przypisałeś a do nowej zmiennej, b. Następnie zmieniłeś wartość sampleProperty b na 20. Po wydrukowaniu wartości sampleProperty a, w obszarze Debug zostanie wydrukowana liczba 10, co oznacza, że wszelkie zmiany wprowadzone w wartości sampleProperty b nie mają wpływu na wartość sampleProperty a. Dzieje się tak, ponieważ kiedy przypisałeś a do b, kopia a została przypisana do b, więc są to całkowicie oddzielne instancje, które nie wpływają na siebie nawzajem.

2. Następnie utworzysz zmienną zawierającą instancję klasy i przypiszesz ją do drugiej zmiennej, a następnie zmienisz wartość właściwości w drugiej zmiennej. Wpisz następujący kod i uruchom go:

```
class SampleReferenceType {

var sampleProperty = 10

}

var c = SampleReferenceType()
```

```
var d = c

c.sampleProperty = 20

print(c.sampleProperty)

print(d.sampleProperty)
```

W tym przykładzie zadeklarowano klasę SampleReferenceType, która zawiera jedną właściwość sampleProperty. Następnie utworzyłeś instancję tej klasy i przypisałeś ją do zmiennej c. Następnie przypisałeś c do nowej zmiennej, d. Następnie zmieniono wartość sampleProperty d na 20. Po wydrukowaniu wartości sampleProperty c, w obszarze Debug zostanie wydrukowana liczba 20, co oznacza, że wszelkie zmiany wprowadzone w c lub d mają wpływ na tę samą instancję SampleReferenceType. Teraz pytanie brzmi: jakich klas czy struktur należy użyć? Przyjrzyjmy się temu w następnej sekcji.

Wybieranie pomiędzy klasami i strukturami

Widziałeś, że możesz użyć klasy lub struktury do reprezentowania złożonego obiektu. Którego więc użyć? Zaleca się używanie struktur, chyba że potrzebujesz czegoś wymagającego klas, np. podklas. Pomaga to w zapobieganiu pewnym subtelnym błędom, które mogą wystąpić, gdy klasy są typami referencyjnymi. Fantastyczny! Skoro już wiesz o klasach i strukturach, w następnej sekcji przyjrzyjmy się wyliczeniom, które pozwalają grupować powiązane wartości.

Zrozumienie wyliczeń

Wyliczenia umożliwiają grupowanie powiązanych ze sobą wartości, na przykład:

- Kierunki kompasu (E, W, N i S)
- Kolory sygnalizacji świetlnej
- Kolory tęczy

Aby zrozumieć, dlaczego wyliczenia byłyby idealne do tego celu, rozważmy następujący przykład. Wyobraź sobie, że programujesz sygnalizację świetlną. Możesz użyć zmiennej całkowitej do przedstawienia różnych kolorów sygnalizacji świetlnej, gdzie 0 to czerwony, 1 to żółty, a 2 to zielony, w następujący sposób:

```
var TrafficLightColor = 2
```

Chociaż jest to możliwy sposób przedstawienia sygnalizacji świetlnej, co się stanie, jeśli przypiszesz wartość 3 do koloru TrafficLightColor? Będzie to powodować problemy, ponieważ 3 nie reprezentuje prawidłowego koloru sygnalizacji świetlnej. Więc byłoby lepiej, gdybyśmy mogli ograniczyć możliwe wartości TrafficLightColor do kolorów, które może wyświetlać. Oto jak wygląda deklaracja wyliczenia:

```
enum EnumName {

case value1

case value2

case value3

}
```

Każde wyliczenie ma opisową nazwę, a treść zawiera skojarzone z nim wartości. Przyjrzyjmy się, jak pracować z wyliczeniami. Dowiesz się, jak je tworzyć i nimi manipulować. W następnej sekcji zaczniesz od utworzenia koloru reprezentującego kolor sygnalizacji świetlnej.

Tworzenie wyliczenia

Utwórzmy wyliczenie reprezentujące sygnalizację świetlną. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw i uruchom go:

```
enum TrafficLightColor {

case red

case yellow

case green

}

var trafficLightColor = TrafficLightColor.red
```

Spowoduje to utworzenie wyliczenia o nazwie TrafficLightColor, które grupuje wartości czerwony, żółty i zielony. Jak widać, wartość zmiennej TrafficLightColor jest ograniczona do koloru czerwonego, żółtego i zielonego; ustawienie jakiejkolwiek innej wartości spowoduje wygenerowanie błędu.

2. Podobnie jak klasy i struktury, wyliczenia mogą zawierać metody. Dodajmy metodę do TrafficLightColor. Zmodyfikuj kod, jak pokazano, aby TrafficLightColor zwrócił ciąg znaków reprezentujący kolor sygnalizacji świetlnej i uruchom go:

```
enum TrafficLightColor {

case red

case yellow

case green

func description() -> String {

switch self {

case .red:

return "red"

case .yellow:

return "yellow"

default:

return "green"

}

}

}
```

```
var trafficLightColor = TrafficLightColor.red

print(trafficLightColor.description())
```

Metoda opis() zwraca ciąg znaków w zależności od wartości TrafficLightColor. Ponieważ wartość TrafficLightColor to TrafficLightColor.red, w obszarze debugowania pojawi się kolor czerwony. Nauczyłeś się, jak tworzyć wyliczenia i używać ich do przechowywania zgrupowanych wartości oraz jak dodawać do nich metody. Na tym kończy się ten rozdział. Dobra robota!

Streszczenie

W tej części nauczyłeś się, jak deklarować złożone obiekty za pomocą klasy, tworzyć instancje klasy, tworzyć podklasę i zastępować metodę klasy. Nauczyłeś się także, jak deklarować strukturę, tworzyć instancje struktury i rozumieć różnicę między typami referencyjnymi i wartościowymi. Na koniec nauczyłeś się, jak używać wyliczeń do reprezentowania określonego zestawu wartości. Teraz wiesz, jak używać klas i struktur do reprezentowania złożonych obiektów oraz jak używać wyliczeń do grupowania powiązanych wartości we własnych programach. W następnym rozdziale dowiesz się, jak określać wspólne cechy klas i struktur za pomocą protokołów, rozszerzać możliwości klas wbudowanych za pomocą rozszerzeń i obsługiwać błędy w programach.

Protokoły, rozszerzenia i obsługa błędów

W poprzedniej części nauczyłeś się reprezentować złożone obiekty za pomocą klas lub struktur oraz jak używać wyliczeń do grupowania powiązanych wartości. W tej części dowiesz się o protokołach, rozszerzeniach i obsłudze błędów. Protokoły definiują schemat metod, właściwości i innych wymagań, które mogą zostać przyjęte przez klasę, strukturę lub wyliczenie. Rozszerzenia umożliwiają udostępnienie nowej funkcjonalności istniejącej klasie, strukturze lub wyliczeniu. Obsługa błędów obejmuje sposób reagowania na błędy w programie i naprawiania ich. Pod koniec tego rozdziału będziesz w stanie pisać własne protokoły, aby spełnić wymagania swoich aplikacji, używać rozszerzeń, aby dodawać nowe możliwości do istniejących typów i obsługiwać warunki błędów w swoich aplikacjach bez awarii.

Zrozumienie protokołów

Protokoły są jak plany, które określają, jakie właściwości lub metody powinien mieć obiekt. Po zadeklarowaniu protokołu klasy, struktury i wyliczenia mogą przyjąć ten protokół i zapewnić własną implementację wymaganych właściwości i metod. Oto jak wygląda deklaracja protokołu:

```
protocol ProtocolName {

var readWriteProperty1 {get set}

var readOnlyProperty2 {get}

methodName1()

methodName2()

}
```

Podobnie jak klasy i struktury, nazwy protokołów zaczynają się od dużej litery. Właściwości należy zadeklarować za pomocą słowa kluczowego var. Używasz {get set}, jeśli chcesz mieć właściwość, z której można czytać lub do której można zapisywać, i {get}, jeśli chcesz mieć właściwość tylko do odczytu. Pamiętaj, że po prostu określasz nazwy właściwości i metod; implementacja odbywa się w obrębie klasy, struktury lub wyliczenia. Aby pomóc Ci zrozumieć protokoły, wyobraź sobie aplikację używaną przez restaurację typu fast food. Kierownictwo zdecydowało się pokazywać kalorie serwowanych posiłków. Aplikacja ma obecnie następującą klasę, strukturę i wyliczenie i żadna z nich nie ma zaimplementowanego liczenia kalorii:

- Zajęcia z burgerami
- Struktura frytek
- Wyliczenie sosów

Dodaj następujący kod do swojego placu zabaw, aby zadeklarować klasę Burger, strukturę Fries i wyliczenie Sauce:

```
class Burger {

}

struct Fries {

}
```

```
enum Sauce {

case chili

case tomato

}
```

Reprezentują one istniejącą klasę, strukturę i wyliczenie w aplikacji. Nie przejmuj się pustymi definicjami, ponieważ nie są one wymagane w tej lekcji. Jak widać, żaden z nich obecnie nie liczy kalorii. Przyjrzyjmy się, jak pracować z protokołami, aby określić właściwości i metody potrzebne do wdrożenia liczenia kalorii. W następnej sekcji zaczniesz od zadeklarowania protokołu, który określa wymagane właściwości i metody.

Tworzenie deklaracji protokołu

Stwórzmy protokół określający wymaganą właściwość, kalorie i metodę opis(). Wpisz następujące polecenie na swoim placu zabaw nad deklaracjami klasy, struktury i wyliczenia:

```
protocol CalorieCount {

var calories: Int { get }

func description() -> String

}
```

Protokół ten nosi nazwę CalorieCount. Określa, że każdy obiekt, który go przyjmuje, musi mieć właściwość kalorie przechowującą liczbę kalorii oraz metodę opis() zwracającą ciąg znaków. { get } oznacza, że wystarczy, że będziesz w stanie odczytać wartość zapisaną w kaloriach i nie musisz jej zapisywać. Należy zauważyć, że definicja metody opis() nie jest określona, ponieważ zostanie to zrobione w klasie, strukturze lub wyliczeniu. Wszystko, co musisz zrobić, aby przyjąć protokół, to wpisać dwukropek po nazwie klasy, po której następuje nazwa protokołu, a następnie zaimplementować wymagane właściwości i metody. Aby klasa Burger była zgodna z tym protokołem, zmodyfikuj swój kod w następujący sposób:

```
class Burger: CalorieCount {

let calories = 800

func description() -> String {

return "This burger has \(calories) calories"

}

}
```

Jak widać, do klasy Burger dodano właściwość kalorie oraz metodę opis(). Mimo że protokół określa zmienną, można tutaj użyć stałej, ponieważ protokół wymaga jedynie, aby można było uzyskać wartość kalorii, a nie ją ustawić. Sprawmy, aby struktura Fries również przyjęła ten protokół. Zmodyfikuj kod struktury Fries w następujący sposób:

```
struct Fries: CalorieCount {

let calories = 500
```

```
func description() -> String {
```



```
return "These fries have \(calories) calories"
}
}
```

Ten sam proces, który został zastosowany w przypadku klasy Burger, został zastosowany w strukturze Fries i teraz jest on również zgodny z protokołem CalorieCount. Możesz zmodyfikować wyliczenie Sauce w ten sam sposób, ale zamiast tego zróbmy to za pomocą rozszerzeń. Rozszerzenia rozszerzają możliwości istniejących klas. W następnej sekcji dodasz protokół CalorieCount do wyliczenia sosu, używając rozszerzenia.

Zrozumienie rozszerzeń

Rozszerzenia umożliwiają nadanie obiektowi dodatkowych możliwości bez modyfikowania oryginalnej definicji obiektu. Możesz ich używać na obiektach dostarczonych przez Apple (gdzie nie masz dostępu do definicji obiektu) lub gdy chcesz posegregować kod w celu zapewnienia czytelności i łatwości konserwacji. Oto jak wygląda rozszerzenie:

```
class ExistingType {
property1
method1()
}
extension ExistingType : ProtocolName {
property2
method2()
}
```

W tym przypadku rozszerzenie służy do zapewnienia dodatkowej właściwości i metody istniejącej klasie. Przyjrzyjmy się, jak używać rozszerzeń. Zaczniesz od dostosowania wyliczenia sosu do protokołu CalorieCount, korzystając z rozszerzenia opisanego w następnej sekcji.

Przyjęcie protokołu poprzez rozszerzenie

Obecnie wyliczenie sosu nie jest zgodne z protokołem CalorieCount. Użyjesz rozszerzenia, aby dodać właściwości i metody wymagane do zapewnienia zgodności. Wpisz poniższy kod po deklaracji wyliczenia sosu:

```
enum Sauce {
case chili
case tomato
}
```

```
extension Sauce: CalorieCount {

var calories: Int {

switch self {

case .chili:

return 20

case .tomato:

return 15

}

}

func description() -> String {

return "This sauce has \(calories) calories"

}

}
```

Jak widać, w pierwotnej definicji wyliczenia Sauce nie wprowadzono żadnych zmian. Jest to również bardzo przydatne, jeśli chcesz rozszerzyć możliwości istniejących standardowych typów Swift, takich jak String i Int. Wyliczenia nie mogą mieć przechowywanych właściwości, dlatego do zwrócenia liczby kalorii na podstawie wartości wyliczenia za pomocą słowa kluczowego self używana jest instrukcja switch. Metoda opis() jest taka sama jak w klasie Burger i strukturze Fries. W tym momencie wszystkie trzy obiekty mają właściwość kalorii i metodę opisu(). Świetnie! Zobaczmy, jak umieścić je w tablicy i wykonać operację, aby uzyskać całkowitą liczbę kalorii w posiłku.

Tworzenie tablicy różnych typów obiektów

Zwykle elementy tablicy muszą być tego samego typu. Ponieważ jednak klasa Burger, struktura Fries i wyliczenie Sauce są zgodne z protokołem CalorieCount, można utworzyć tablicę zawierającą elementy zgodne z tym protokołem. Wykonaj następujące kroki:

1. Aby dodać instancje klasy Burger, struktury Fries i wyliczenia Sauce do tablicy, wpisz następujący kod po wszystkich deklaracjach protokołu i obiektu:

```
let burger = Burger()

let fries = Fries()

let sauce = Sauce.tomato

let foodArray: [CalorieCount] = [burger, fries, sauce]
```

2. Aby uzyskać całkowitą liczbę kalorii, dodaj następujący kod po linii, w której utworzyłeś stałą foodArray:

```
var totalCalories = 0

for food in foodArray {

totalCalories += food.calories
```

}

print(totalCalories)

Pętla for iteruje po każdym elemencie tablicy foodArray. Dla każdej iteracji wartość właściwości kalorii każdego artykułu spożywczego zostanie dodana do sumy kalorii, a całkowita ilość (1315) zostanie wyświetlona w obszarze debugowania. Nauczyłeś się, jak utworzyć protokół i dostosować do niego klasę, strukturę lub wyliczenie, albo w ramach definicji klasy, albo poprzez rozszerzenia. Przyjrzyjmy się następnie obsłudze błędów, która opisuje, jak reagować na błędy w programie lub je naprawiać.

Badanie obsługi błędów

Pisząc aplikacje, pamiętaj, że mogą wystąpić błędy, a obsługa błędów to sposób, w jaki aplikacja będzie reagować na takie warunki i naprawiać je. Najpierw tworzysz typ zgodny z protokołem błędów Swift, który pozwala na użycie tego typu do obsługi błędów. Zwykle stosuje się wyliczenia, ponieważ można określić powiązane wartości dla różnych rodzajów błędów. Gdy wydarzy się coś nieoczekiwanego, możesz zatrzymać wykonywanie programu, zgłaszając błąd. Używasz do tego instrukcji rzutu i dostarczasz instancję typu zgodnego z protokołem Error z odpowiednią wartością. Dzięki temu możesz zobaczyć, co poszło nie tak. Oczywiście byłoby lepiej, gdybyś mógł odpowiedzieć na błąd bez zatrzymywania programu. Dla tego, możesz użyć bloku do-catch, który wygląda tak:

do {

try expression1

statement1

} catch {

statement2

}

Tutaj próbujesz wykonać kod w bloku do za pomocą słowa kluczowego try. Jeżeli zostanie zgłoszony błąd, wykonywane są instrukcje z bloku catch. Możesz mieć wiele bloków catch do obsługi różnych typów błędów. Załóżmy na przykład, że masz aplikację, która musi uzyskać dostęp do strony internetowej. Jeśli jednak serwer, na którym znajduje się ta strona internetowa, nie działa, do Ciebie należy napisanie kodu obsługującego błąd, na przykład wypróbowanie alternatywnego serwera WWW lub poinformowanie użytkownika, że serwer nie działa. Utwórzmy wyliczenie zgodne z protokołem Error, użyjmy instrukcji rzut, aby zatrzymać wykonywanie programu w przypadku wystąpienia błędu, i użyjmy bloku do-catch do obsługi błędu. Wykonaj następujące kroki:

1. Wpisz następujący kod na swoim placu zabaw:

enum WebsiteError: Error {

case noInternetConnection

case siteDown

case wrongURL

}

Deklaruje to wyliczenie WebsiteError, które przyjmuje protokół Error. Obejmuje trzy możliwe warunki błędu; brak połączenia z Internetem, witryna nie działa lub nie można ustalić adresu URL.

2. Wpisz poniższy kod, aby zadeklarować funkcję sprawdzającą, czy strona internetowa działa po deklaracji WebpageError:

```
func checkWebsite(siteUp: Bool) throws -> String {

if siteUp == false {

throw WebsiteError.siteDown

}

return "Site is up"

}
```

Jeśli siteUp ma wartość true, zwracany jest komunikat „Witryna działa”. Jeśli siteUp ma wartość false, program przestanie działać i zgłosi błąd.

3. Wpisz następujący kod po deklaracji funkcji, aby wywołać funkcję i uruchomić program:

```
let siteStatus = true

try checkWebsite(siteUp: siteStatus)
```

Ponieważ status witryny ma wartość true, w obszarze Wyniki pojawi się informacja Witryna działa.

4. Zmień wartość siteStatus na false i uruchom program. Program ulega awarii i w obszarze debugowania wyświetlany jest następujący komunikat o błędzie:

```
Playground execution terminated: An error was thrown and was not caught:

__lldb_expr_5.WebsiteError.siteDown
```

5. Oczywiście zawsze lepiej jest, jeśli potrafisz poradzić sobie z błędami bez powodowania awarii programu. Można to zrobić za pomocą bloku do-catch. Zmodyfikuj swój kod, jak pokazano, i uruchom go:

```
let siteStatus = false

do {

print(try checkWebsite(siteUp: siteStatus))

} catch {

print(error)

}
```

Blok do próbuje wykonać funkcję checkWebsite(siteUp:) i wypisuje status, jeśli się powiedzie. Jeśli wystąpi błąd, zamiast awarii zostaną wykonane instrukcje z bloku catch, a w obszarze Debugowanie pojawi się komunikat o błędzie siteDown. Nauczyłeś się, jak radzić sobie z błędami w aplikacji, nie powodując jej awarii. Świetnie!

Streszczenie

W tej części nauczyłeś się pisać protokoły i dostosowywać do nich klasy, struktury i wyliczenia. Nauczyłeś się także, jak rozszerzać możliwości klasy za pomocą rozszerzenia. Wreszcie nauczyłeś się obsługiwać błędy za pomocą bloku do-catch. Może się to teraz wydawać dość abstrakcyjne i trudne do

zrozumienia, ale jak zobaczysz, będziesz używać protokołów do implementowania wspólnych funkcjonalności w różnych częściach programu, zamiast pisać w kółko ten sam program. Zobaczysz, jak przydatne są rozszerzenia w organizowaniu kodu, co ułatwia jego utrzymanie. Na koniec zobaczysz, jak dobra obsługa błędów ułatwia wskazanie błędów popełnionych podczas kodowania aplikacji. W kolejnej części poznasz współbieżność Swifta, czyli nowy sposób na obsługę operacji asynchronicznych w Swift.

Szybka współbieżność

Firma Apple wprowadziła Swift Concurrency podczas WWDC 2021, która dodaje obsługę strukturalnego programowania asynchronicznego i równoległego do Swift 5.5. Pozwala to na pisanie współbieżnego kodu, który jest bardziej czytelny i łatwiejszy do zrozumienia. W tej części poznasz podstawowe pojęcia dotyczące współbieżności Swift. Następnie sprawdzisz aplikację bez współbieżności i przeanalizujesz jej problemy. Następnie użyjesz async/await, aby zaimplementować współbieżność w aplikacji. Wreszcie, zwiększysz wydajność swojej aplikacji, używając asynchronizacji. Pod koniec tego rozdziału poznasz podstawy działania Swift Concurrency i sposobów aktualizowania własnych aplikacji, aby z niego korzystać.

Zrozumienie szybkiej współbieżności

W Swift 5.5 Apple dodał obsługę pisania kodu asynchronicznego i równoległego w ustrukturyzowany sposób. Kod asynchroniczny umożliwia aplikacji zawieszanie i wznawianie kodu. Dzięki temu aplikacja może na przykład aktualizować interfejs użytkownika, a jednocześnie wykonywać takie operacje, jak pobieranie danych z Internetu. Kod równoległy umożliwia aplikacji jednoczesne uruchamianie wielu fragmentów kodu. Aby dać ci wyobrażenie o tym, jak działa Swift Concurrency, wyobraź sobie, że robisz na śniadanie kanapkę z jajkiem w koszulce. Oto jeden ze sposobów:

1. Włóż dwie kromki chleba do tostera.

2. Odczekaj dwie minuty, aż chleb się zarumieni.

3. Włóż jajko do miski z wodą i włóż miskę do kuchenki mikrofalowej.

4. Poczekaj sześć minut, aż jajko się ugotuje.

5. Zrób kanapkę.

Całość zajmuje osiem minut. Teraz pomyśl o tej sekwencji wydarzeń. Czy spędzasz ten czas wpatrując się w toster i kuchenkę mikrofalową? Prawdopodobnie będziesz korzystać z telefonu, gdy chleb będzie w tosterze, a jajko w kuchence mikrofalowej. Innymi słowy, możesz robić inne rzeczy podczas przygotowywania chleba i jajka. Zatem sekwencję wydarzeń można dokładniej opisać w następujący sposób:

1. Włóż dwie kromki chleba do tostera.

2. Używaj telefonu przez dwie minuty, aż chleb się upiecze.

3. Włóż jajko do miski z wodą i włóż miskę do kuchenki mikrofalowej.

4. Używaj telefonu przez sześć minut, aż jajko będzie ugotowane.

5. Zrób kanapkę.

Tutaj widać, że interakcję z tosterem i kuchenką mikrofalową można zawiesić, a następnie wznowić, co oznacza, że te operacje są asynchroniczne. Operacja nadal trwa osiem minut, ale w tym czasie mogła Pani wykonywać inne czynności. Należy wziąć pod uwagę jeszcze jeden czynnik. Nie musisz czekać, aż chleb się upiecze, zanim włożysz jajko do kuchenki mikrofalowej. Oznacza to, że możesz zmodyfikować sekwencję kroków w następujący sposób:

1. Włóż dwie kromki chleba do tostera.

2. Podczas opiekania chleba włóż jajko do miski z odrobiną wody i włóż miskę do kuchenki mikrofalowej.

3. Używaj telefonu przez sześć minut, aż jajko będzie ugotowane.

4. Zrób kanapkę.

Opiekanie chleba i gotowanie jajek odbywa się teraz równolegle, co pozwala zaoszczędzić dwie minuty. Świetnie! Pamiętaj jednak, że masz więcej rzeczy do śledzenia. Teraz, gdy rozumiesz koncepcje operacji asynchronicznych i równoległych, w następnej sekcji przeanalizujmy problemy związane z aplikacją bez współbieżności.

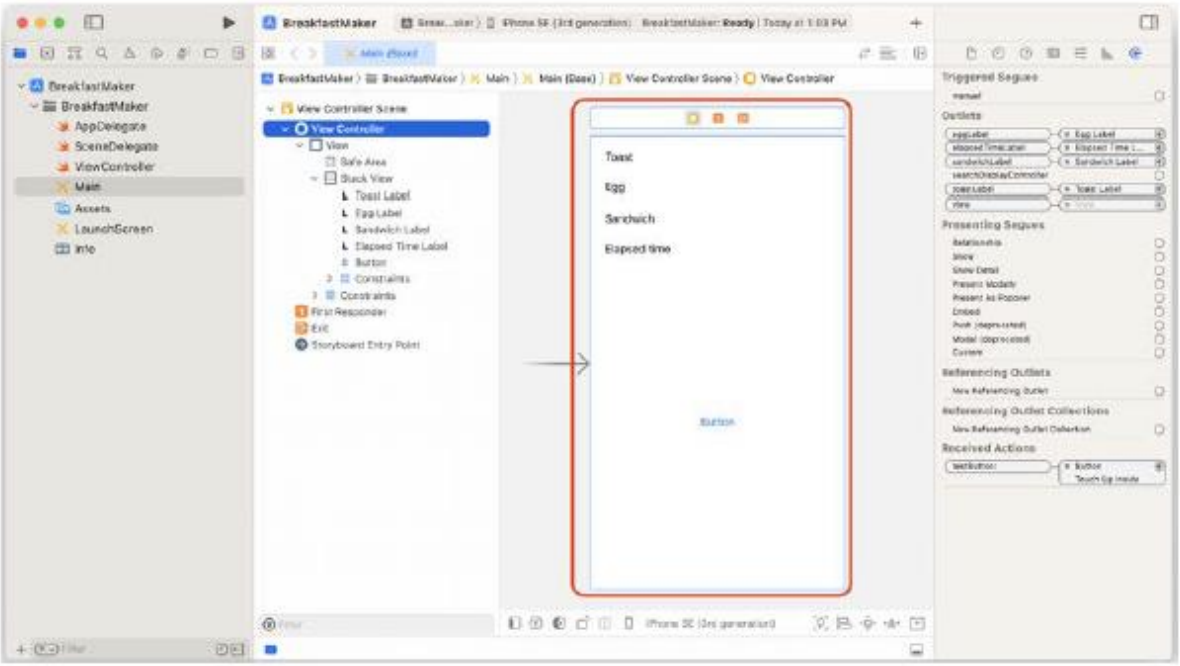
Sprawdzanie aplikacji bez współbieżności

Widziałeś, jak operacje asynchroniczne i równoległe mogą pomóc Ci szybciej przygotować śniadanie i umożliwić korzystanie z telefonu podczas jego wykonywania. Przyjrzyjmy się teraz przykładowej aplikacji symulującej proces przygotowywania śniadania. Początkowo ta aplikacja nie ma zaimplementowanej współbieżności, więc możesz zobaczyć, jak to wpływa na aplikację. Wykonaj następujące kroki:

1. Jeśli jeszcze tego nie zrobiłeś, pobierz folder Chapter09 pakietu kodu pod tym linkiem: https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Seventh-Edition

2. Otwórz folder Chapter09, a zobaczysz dwa foldery, BreakfastMaker-start i BreakfastMaker-complete. Pierwszy folder zawiera aplikację, którą będziesz modyfikować w tym rozdziale, a drugi zawiera ukończoną aplikację.

3. Otwórz folder BreakfastMaker-start i otwórz projekt BreakfastMaker Xcode. Kliknij plik głównego scenorysu w nawigatorze projektu. Powinieneś zobaczyć cztery etykiety i przycisk w scenie kontrolera widoku, jak pokazano:



Aplikacja wyświetli ekran pokazujący status tostu, jajka i kanapki oraz czas potrzebny na przygotowanie kanapki. W aplikacji wyświetli się także przycisk, za pomocą którego możesz przetestować responsywność interfejsu użytkownika.

4. Kliknij plik ViewController w nawigatorze projektu. Powinieneś zobaczyć następujący kod w obszarze Edytora:

```
import UIKit

class ViewController: UIViewController {

@IBOutlet var toastLabel: UILabel!

@IBOutlet var eggLabel: UILabel!

@IBOutlet var sandwichLabel: UILabel!

@IBOutlet var elapsedTimeLabel: UILabel!

override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

let startTime = Date().timeIntervalSince1970

toastLabel.text = "Making toast..."

toastLabel.text = makeToast()

eggLabel.text = "Poaching egg..."

eggLabel.text = poachEgg()

sandwichLabel.text = makeSandwich()

let endTime = Date().timeIntervalSince1970

elapsedTimeLabel.text = "Elapsed time is

\(((endTime - startTime) * 100).rounded()

/ 100) seconds"

}

func makeToast() -> String {

sleep(2)

return "Toast done"

}

func poachEgg() -> String {

sleep(6)

return "Egg done"

}

func makeSandwich() -> String {

return "Sandwich done"
```

}

@IBAction func testButton(_ sender: UIButton) {

print("Button tapped")

}

}

Jak widać, kod ten symuluje proces przygotowywania śniadania opisany w poprzedniej sekcji. Rozbijmy to:

@IBOutlet var toastLabel: UILabel!

@IBOutlet var eggLabel: UILabel!

@IBOutlet var sandwichLabel: UILabel!

@IBOutlet var elapsedTimeLabel: UILabel!

Te punkty sprzedaży są powiązane z czterema etykietami w głównym pliku scenorysu. Po uruchomieniu aplikacji etykiety te będą wyświetlać status tostu, jajka i kanapki, a także czas potrzebny na ukończenie procesu.

override func viewDidAppear(_ animated: Bool) {

Ta metoda jest wywoływana, gdy na ekranie pojawia się widok kontrolera widoku.

let startTime = Date().timeIntervalSince1970

Spowoduje to ustawienie czasu startTime na bieżący czas, dzięki czemu aplikacja będzie mogła później obliczyć, ile czasu zajmie przygotowanie kanapki.

toastLabel.text = "Making toast..."

To sprawia, że toastLabel wyświetla tekst Robienie tostów....

toastLabel.text = makeToast()

Wywołuje to metodę makeToast(), która czeka dwie sekundy na symulację czasu potrzebnego na wykonanie tostu, a następnie zwraca tekst Toast sporządzono, który zostanie wyświetlony przez toastLabel.

eggLabel.text = "Poaching egg..."

To sprawia, że EggLabel wyświetla tekst Kłusowanie jaj....

eggLabel.text = poachEgg()

Wywołuje to metodę poachEgg(), która czeka przez sześć sekund na symulację czasu potrzebnego na ugotowanie jajka w koszulce, a następnie zwraca tekst Egg made, który zostanie wyświetlony przez EggLabel.

sandwichLabel.text = makeSandwich()

Wywołuje to metodę makeSandwich(), która zwraca tekst Sandwich made, który będzie wyświetlany przez SandwichLabel.

let endTime = Date().timeIntervalSince1970

This sets endTime to the current time.

elapsedTimeLabel.text = "Elapsed time is

\(((endTime - startTime) * 100).rounded()

/ 100) seconds"

Spowoduje to obliczenie czasu, który upłynął (około ośmiu sekund), który będzie wyświetlany jako elapsedTimeLabel.

@IBAction func testButton(_ sender: UIButton) {

print("Button tapped")

}

Spowoduje to wyświetlenie przycisku naciśniętego w obszarze debugowania za każdym razem, gdy zostanie naciśnięty przycisk na ekranie. Zbuduj i uruchom aplikację, a następnie naciśnij przycisk w momencie pojawienia się interfejsu użytkownika:

Powinieneś zauważyć następujące problemy:
• Naciśnięcie przycisku początkowo nie daje żadnego efektu, a po około ośmiu sekundach w obszarze Debugowanie zobaczysz przycisk naciśnięty.
• Robienie tostów... i Gotowanie jajek... nigdy nie są wyświetlane, natomiast Tosty gotowe i Jajka gotowe pojawiają się dopiero po około ośmiu sekundach.

Dzieje się tak dlatego, że kod aplikacji nie zaktualizował interfejsu użytkownika, gdy działają metody makeToast() i poachEgg(). Twoja aplikacja zarejestrowała dotknięcia przycisków, ale tak się stało jest w stanie je przetworzyć i zaktualizować etykiety dopiero po zakończeniu wykonywania funkcji makeToast() i poachEgg(). Te problemy nie zapewniają użytkownikowi dobrego doświadczenia z Twoją aplikacją. Wystąpiły problemy związane z aplikacją, która nie ma zaimplementowanej współbieżności. W następnej sekcji zmodyfikujesz aplikację za pomocą funkcji async/await, aby tak się stało

Aktualizowanie aplikacji przy użyciu async/await

Jak widzieliśmy wcześniej, aplikacja nie odpowiada, gdy działają metody makeToast() i poachEgg(). Aby rozwiązać ten problem, użyjesz asynchronizacji/await w aplikacji. Zapisanie słowa kluczowego async w deklaracji metody wskazuje, że metoda jest asynchroniczna. Oto jak to wygląda:

```
func nazwa_metody() async -> returnType {
```

Zapisanie słowa kluczowego oczekującego przed wywołaniem metody oznacza moment, w którym wykonanie może zostać wstrzymane, umożliwiając w ten sposób wykonanie innych operacji. Oto jak to wygląda:

```
await methodName()
```

Zmodyfikujesz swoją aplikację tak, aby korzystała z funkcji async/await. Umożliwi to zawieszenie metod makeToast() i poachEgg() w celu przetwarzania naciśnięć przycisków i aktualizacji interfejsu użytkownika, a następnie wznowienie wykonywania obie metody później. Wykonaj następujące kroki:

1. Zmodyfikuj metody makeToast() i poachEgg(), jak pokazano, aby kod w ich treściach był asynchroniczny:

```
func makeToast() -> String {

try? await Task.sleep(nanoseconds: 2 * 1_000_000_000)

return "Toast done"

}

func poachEgg() -> String {

try? await Task.sleep(nanoseconds: 6 *

1_000_000_000)

return "Egg done"

}
```

Zadanie reprezentuje jednostkę pracy asynchronicznej. Zadanie ma metodę statyczną, Sleep(nanosekundy:), która wstrzymuje wykonanie na określony czas mierzony w nanosekundach. Mnożenie przez 1 000 000 000 powoduje przeliczenie czasu trwania na sekundy. Ponieważ ta metoda jest metodą rzucania, użyjesz metody try? słowo kluczowe, aby je wywołać bez konieczności implementowania bloku do-catch. Słowo kluczowe Wait wskazuje, że ten kod można zawiesić, aby umożliwić uruchomienie innego kodu.

2. Pojawią się błędy zarówno dla makeToast(), jak i poachEgg(). Kliknij dowolną ikonę błędu, aby wyświetlić komunikat o błędzie:

```
29      func makeToast() -> String {
30          try? await
                Task.sleep(nanoseconds: 2 *
                1_000_000_000)
31          return "Toast done"
32      }
33
34      func poachEgg() -> String {
35          try? await
                Task.sleep(nanoseconds: 6 *
                1_000_000_000)
36          return "Egg done"
37      }
```

Błąd jest wyświetlany, ponieważ wywołujesz metodę asynchroniczną wewnątrz metody, która nie obsługuje współbieżności. Będziesz musiał dodać słowo kluczowe async do deklaracji metody, aby wskazać, że jest ona asynchroniczna.

3. Dla każdej metody kliknij przycisk Napraw, aby dodać słowo kluczowe async do deklaracji metody.

4. Po zakończeniu sprawdź, czy Twój kod wygląda tak:

func makeToast() async -> String {

try? await Task.sleep(nanoseconds: 2 * 1_000_000_000)

return "Toast done"

}

func poachEgg() async -> String {

try? await Task.sleep(nanoseconds: 6 * 1_000_000_000)

return "Egg done"

}

5. Błędy w metodach makeToast() i poachEgg() powinny zniknąć, ale pojawią się nowe błędy w metodzie viewDidAppear(). Kliknij jedną z ikon błędów, aby wyświetlić komunikat o błędzie, który będzie taki sam, jak komunikat, który widziałeś wcześniej w kroku 2. Dzieje się tak, ponieważ wywołujesz metodę asynchroniczną wewnątrz metody, która nie obsługuje współbieżności.

6. Kliknij przycisk Napraw, a pojawi się więcej błędów.

7. Na razie zignoruj tę w deklaracji metody i kliknij tę obok wywołania metody makeToast(), aby zobaczyć komunikat o błędzie:

```
override func viewDidAppear(_ animated: Bool)
    async {
    super.viewDidAppear(animated)
    let startTime = Date().timeIntervalSince1970
    toastLabel.text = "Making toast..."
    toastLabel.text = makeToast()   Expression is 'as...
    eggLabel.text = "Poaching eggs..."
    eggLabel.text = poachEgg()   Expression is 'async'...
    sandwichLabel.text = makeSandwich()
    let endTime = Date().timeIntervalSince1970
```

Ten komunikat o błędzie jest wyświetlany, ponieważ nie użyto funkcji Wait podczas wywoływania funkcji asynchronicznej.

8. Kliknij przycisk Napraw, aby wstawić słowo kluczowe Wait przed wywołaniem metody.

9. Powtórz kroki 7 i 8 w przypadku błędu obok wywołania metody poachEgg(). Słowo kluczowe Wait zostanie wstawione również dla wywołania metody poachEgg().

10. Kliknij ikonę błędu w deklaracji metody viewDidLoad(), aby zobaczyć komunikat o błędzie:

```
16
17      override func viewDidAppear(_ animated: Bool)
            async {
18          super.viewDidAppear(animated)
19          let startTime = Date().timeIntervalSince1970
```

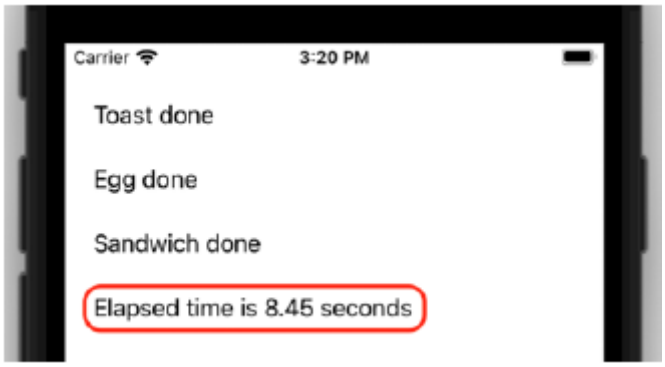
Ten błąd jest wyświetlany, ponieważ nie można użyć słowa kluczowego async, aby uczynić metodę viewDidAppear() asynchroniczną, ponieważ taka możliwość nie jest dostępna w nadklasie.

11. Aby rozwiązać ten problem, usuniesz słowo kluczowe async i umieścisz cały kod po super. viewDidAppear() w bloku Task, co umożliwi mu wykonanie asynchroniczne w metodzie synchronicznej. Zmodyfikuj swój kod w następujący sposób:

```
override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

Task {

let startTime = Date().timeIntervalSince1970

toastLabel.text = "Making toast..."

toastLabel.text = makeToast()

eggLabel.text = "Poaching egg..."

eggLabel.text = poachEgg()

sandwichLabel.text = makeSandwich()

let endTime = Date().timeIntervalSince1970

elapsedTimeLabel.text = "Elapsed time is

\(((endTime - startTime) * 100).rounded()

/ 100) seconds"

}

}
```

Zbuduj i uruchom aplikację, a gdy tylko zobaczysz interfejs użytkownika, dotknij przycisku. Zwróć uwagę, że naciśnięcie przycisku pojawia się teraz natychmiast w obszarze debugowania, a etykiety są aktualizowane tak, jak powinny. Dzieje się tak, ponieważ aplikacja może teraz zawieszać metody

makeToast() i poachEgg(), aby reagować na kliknięcia i aktualizować interfejs użytkownika, a następnie wznawiać je później. Wspaniały! Jeśli jednak spojrzysz na czas, jaki upłynął, zobaczysz, że przygotowanie śniadania aplikacji zajmuje nieco więcej czasu niż wcześniej:



Jest to częściowo spowodowane dodatkowym przetwarzaniem wymaganym do zawieszania i wznawiania metod, ale wiąże się to z innym czynnikiem. Mimo że metody makeToast() i poachEgg() są teraz asynchroniczne, wykonywanie metody poachEgg() rozpoczyna się dopiero po zakończeniu wykonywania metody makeToast(). W następnej sekcji zobaczysz, jak użyć funkcji async-let do równoległego uruchomienia metod makeToast() i poachEgg().

Poprawa wydajności przy użyciu asynchronizacji

Mimo że Twoja aplikacja reaguje teraz na naciśnięcia przycisków i może aktualizować interfejs użytkownika, gdy działają metody makeToast() i poachEgg(), obie metody nadal działają sekwencyjnie. Rozwiązaniem jest tutaj użycie async-let. Zapisanie słowa async przed instrukcją le podczas definiowania stałej, a następnie zapisanie Wait podczas uzyskiwania dostępu do stałej umożliwia równoległe wykonywanie operacji asynchronicznych metody:

```
async let temporaryConstant1 = methodName1()

async let temporaryConstant2 = methodName2()

await variable1 = temporaryConstant1

await variable2 = temporaryConstant1
```

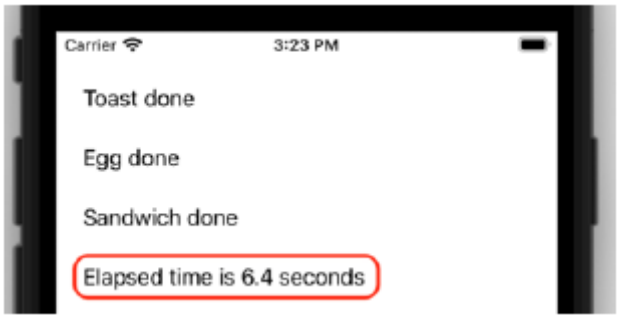
Tutaj methodName1() i methodName2() będą działać równolegle.

Zmodyfikujesz swoją aplikację tak, aby korzystała z funkcji async-let, aby umożliwić równoległe działanie metod makeToast() i poachEgg(). W pliku ViewController zmodyfikuj kod w bloku Task w następujący sposób:

```
Task {

let startTime = Date().timeIntervalSince1970

toastLabel.text = "Making toast..."

async let tempToast = makeToast()

eggLabel.text = "Poaching egg..."
```

```
async let tempEgg = poachEgg()

await toastLabel.text = tempToast

await eggLabel.text = tempEgg

sandwichLabel.text = makeSandwich()

let endTime = Date().timeIntervalSince1970

elapsedTimeLabel.text = "Elapsed time is

\(((endTime - startTime) * 100).rounded()

/ 100) seconds"

}
```

Kompiluj i uruchamiaj aplikację. Zobaczysz, że czas, który upłynął, jest teraz krótszy niż wcześniej:



Dzieje się tak, ponieważ użycie funkcji async-let umożliwia równoległe działanie metod makeToast() i poachEgg(), a metoda poachEgg() nie czeka już na zakończenie metody makeToast() przed rozpoczęciem wykonywania. Fajny!

Pomyślnie zaimplementowałeś kod asynchroniczny w swojej aplikacji. Fantastyczny! Jest jeszcze wiele rzeczy do nauczenia się na temat Swift Concurrency, takich jak współbieżność strukturalna i aktorzy, ale to jest poza naszym . Poklep się po plecach; ukończyłeś pierwszą część tej książki!

Streszczenie

W tej części dowiedziałeś się o Swift Concurrency i o tym, jak wdrożyć ją w aplikacji BreakfastMaker. Zacząłeś od poznania podstawowych pojęć dotyczących współbieżności Swift. Następnie sprawdziłeś aplikację bez współbieżności i zbadałeś jej problemy. Następnie zaimplementowałeś współbieżność w aplikacji za pomocą async/await. Wreszcie zwiększyłeś wydajność swojej aplikacji, używając async-let. Rozumiesz teraz podstawy Swift Concurrency i będziesz mógł używać async/await i async-let we własnych aplikacjach. W następnym rozdziale zaczniesz pisać swoją pierwszą aplikację na iOS, tworząc dla niej ekrany za pomocą scenorysów, które pozwalają szybko prototypować aplikację bez konieczności wpisywania dużej ilości kodu.

Konfigurowanie interfejsu użytkownika

W części 1 uczyłeś się o języku Swift i jego działaniu. Teraz, gdy masz już dobrą praktyczną znajomość języka, możesz nauczyć się tworzyć aplikację na iOS. W tej części zbudujesz interfejs użytkownika (UI) aplikacji do rezerwacji restauracji Let's Eat. Użyjesz do tego narzędzia Interface Builder Xcode, a kodowanie zostanie ograniczone do minimum. Rozpoczniesz ten rozdział od poznania przydatnych terminów używanych przy tworzeniu aplikacji na iOS, które są szeroko używane. Następnie obejrzysz ekrany używane w aplikacji Let's Eat i dowiesz się, jak użytkownik będzie korzystał z aplikacji. Na koniec zaczniesz odtwarzać interfejs aplikacji za pomocą narzędzia Interface Builder, zaczynając od paska kart, który umożliwia użytkownikowi wybór pomiędzy ekranami Eksploruj i Mapa. Dodasz paski nawigacyjne u góry obu ekranów. Dowiesz się także, jak skonfigurować ekran uruchamiania wyświetlany po uruchomieniu aplikacji oraz jak używać niestandardowych ikon na ekranie uruchamiania i przyciskach paska kart. Pod koniec tego rozdziału poznasz popularne terminy używane przy tworzeniu aplikacji na iOS, poznasz, jak będzie wyglądać przepływ aplikacji, jak dodawać zasoby do aplikacji i jak używać Konstruktora interfejsów do dodawania, konfigurowania i położenie elementów interfejsu użytkownika.

Nauka przydatnych terminów w programowaniu iOS

Rozpoczynając swoją przygodę z tworzeniem aplikacji na iOS, natkniesz się na specjalne terminy i definicje. Oto niektóre z najczęściej używanych terminów i definicji. Na razie je tylko przeczytaj. Nawet jeśli nie wszystko jeszcze rozumiesz, z biegiem czasu stanie się to jaśniejsze:

• Widok: Widok jest instancją klasy UIView lub jednej z jej podklas. Wszystko, co widzisz na ekranie (przyciski, pola tekstowe, etykiety itd.), jest widokiem. Będziesz używać widoków do tworzenia interfejsu użytkownika.

• Widok stosu: Widok stosu jest instancją klasy UIStackView, która jest podklasą UIView. Służy do grupowania widoków w stos poziomy lub pionowy. Dzięki temu łatwiej jest je ustawić na ekranie za pomocą funkcji automatycznego układu, co opisano w dalszej części tej sekcji.

• Kontroler widoku: Kontroler widoku jest instancją klasy UIViewController. Określa, co widok wyświetla użytkownikowi i co się dzieje, gdy użytkownik wchodzi w interakcję z widokiem. Każdy kontroler widoku ma właściwość view, która zawiera odwołanie do widoku.

• Kontroler widoku tabeli: Kontroler widoku tabeli jest instancją klasy UITableViewController, która jest podklasą klasy UIViewController. Jej właściwość view zawiera odwołanie do instancji UITableView (widoku tabeli), która wyświetla pojedynczą kolumnę instancji UITableViewCell (komórek widoku tabeli).

Aplikacja Ustawienia wyświetla ustawienia urządzenia w widoku tabeli:

Jak widać, wszystkie ustawienia (Ogólne, Dostępność, Prywatność itd.) są wyświetlane w komórkach widoku tabeli wewnątrz widoku tabeli.

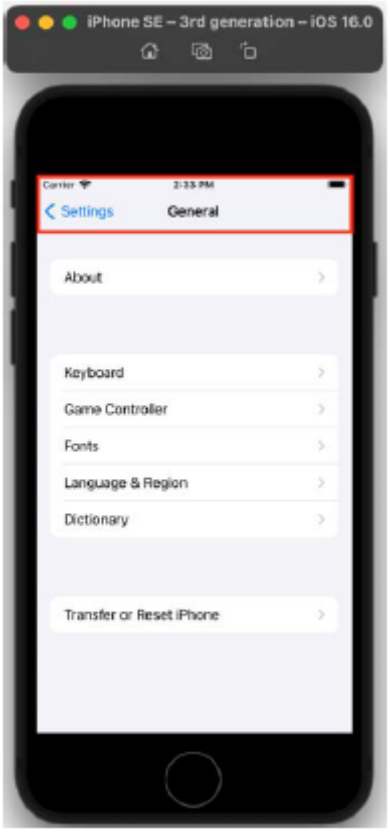
• Kontroler widoku kolekcji: Kontroler widoku kolekcji jest instancją klasy UICollectionViewController, która jest podklasą klasy UIViewController. Jej właściwość view zawiera odwołanie do instancji UICollectionView (widok kolekcji), która wyświetla siatkę instancji UICollectionViewCell (komórki widoku kolekcji).

Aplikacja Zdjęcia wyświetla zdjęcia w widoku kolekcji:

Jak widać, miniatury obrazów są wyświetlane w komórkach widoku kolekcji.

• Kontroler nawigacji: Kontroler nawigacji jest instancją klasy UINavigationController, która jest podklasą klasy UIViewController. Posiada właściwość viewControllers, która przechowuje tablicę kontrolerów widoku. Na ekranie pojawi się widok ostatniego kontrolera widoku w tablicy wraz z paskiem nawigacyjnym u góry ekranu.

Kontroler widoku tabeli w aplikacji Ustawienia jest osadzony w kontrolerze nawigacji, a pasek nawigacyjny widać nad widokiem tabeli:



Po dotknięciu ustawienia kontroler widoku dla tego ustawienia zostanie dodany do tablicy kontrolerów widoku przypisanych do właściwości viewControllers. Użytkownik widzi widok tego kontrolera widoku przesuwany z prawej strony. Zwróć uwagę na pasek nawigacyjny u góry ekranu, na którym może znajdować się tytuł i przyciski. W lewym górnym rogu paska nawigacyjnego pojawi się przycisk < Ustawienia. Dotknięcie tego przycisku powoduje powrót do poprzedniego ekranu.

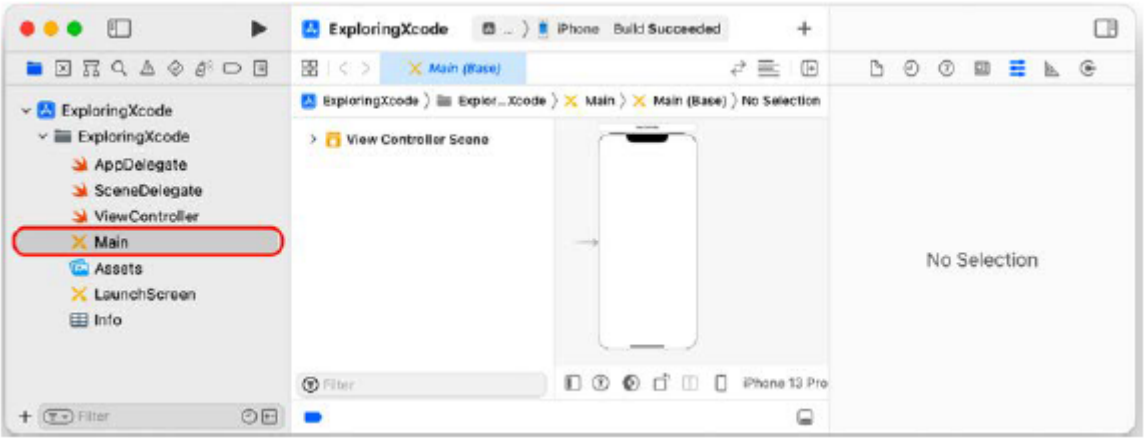
• Kontroler paska kart: Kontroler paska kart jest instancją klasy UITabBarController, która jest podklasą klasy UIViewController. Posiada właściwość viewControllers, która przechowuje tablicę kontrolerów widoku. Na ekranie pojawi się widok pierwszego kontrolera widoku w tablicy wraz z paskiem kart z przyciskami na dole. Przycisk po lewej stronie odpowiada pierwszemu kontrolerowi widoku w tablicy i zostanie już wybrany. Po dotknięciu innego przycisku zostanie załadowany odpowiedni kontroler widoku, a jego widok pojawi się na ekranie.

Aplikacja Zdjęcia korzysta z paska zakładek, aby wyświetlić rząd przycisków u dołu ekranu:

Dotknięcie każdego przycisku na pasku kart spowoduje wyświetlenie innego ekranu.

• Kontroler widoku modelu (MVC): Jest to bardzo powszechny wzorzec projektowy używany przy tworzeniu aplikacji na iOS. Użytkownik wchodzi w interakcję z widokami na ekranie. Dane aplikacji są przechowywane w obiektach modelu danych. Kontrolery zarządzają przepływem informacji pomiędzy widokami i obiektami modelu danych.

• Storyboard: plik scenorysu zawiera wizualną reprezentację tego, co widzi użytkownik. Każdy ekran informacyjny jest reprezentowany przez scenę scenorysową. Otwórz projekt ExploringXcode utworzony wcześniej i kliknij główny plik scenorysu.



Zobaczysz w nim jedną scenę, a kiedy uruchomisz aplikację w symulatorze iOS, zawartość tej sceny zostanie wyświetlona na ekranie. W pliku scenorysu możesz mieć więcej niż jedną scenę.

• Przejście: Jeśli w aplikacji znajduje się więcej niż jedna scena, przejścia z jednej sceny do drugiej służą do przejścia. Projekt ExploringXcode nie ma żadnych przejść, ponieważ w jego scenorysie znajduje się tylko jedna scena, ale zobaczysz je w dalszej części .

• Układ automatyczny: Jako programista musisz upewnić się, że Twoja aplikacja wygląda dobrze na urządzeniach o różnych rozmiarach ekranów. Układ automatyczny pomaga rozplanować interfejs użytkownika w oparciu o określone ograniczenia. Można na przykład ustawić ograniczenie, aby przycisk był wyśrodkowany na ekranie niezależnie od jego rozmiaru, lub aby pole tekstowe rozszerzało się po obróceniu urządzenia z pozycji pionowej do poziomej.

Teraz, gdy znasz już terminy używane przy tworzeniu aplikacji na iOS, przyjrzyjmy się aplikacji, którą utworzysz.

Zwiedzanie aplikacji Let's Eat

Przyjrzyjmy się szybko aplikacji, którą utworzysz. Aplikacja Let's Eat to aplikacja restauracyjna, która umożliwia użytkownikom przeglądanie listy restauracji podzielonych na kategorie według kuchni lub przeglądanie mapy pokazującej wszystkie restauracje w danym obszarze. W kolejnych sekcjach zobaczysz wszystkie ekrany używane w aplikacji i jej ogólny przebieg.

Korzystanie z ekranu Eksploruj

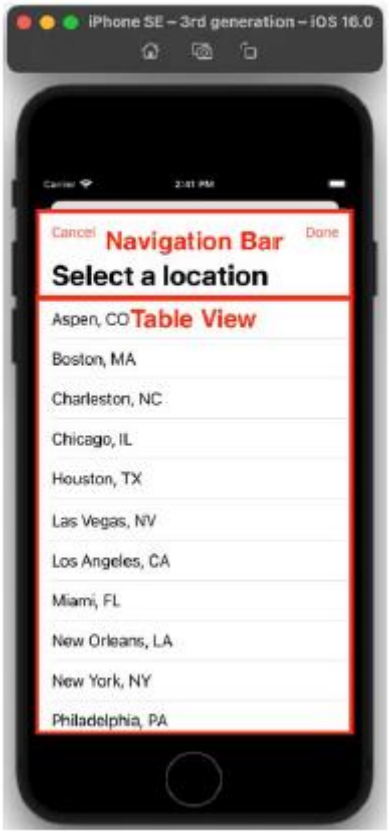
Po uruchomieniu aplikacji zobaczysz ekran Eksploruj:

Przyjrzyjmy się różnym częściom tego ekranu. Instancja UITabBar (pasek kart) u dołu ekranu wyświetla przyciski Eksploruj i Mapa. Zostanie wybrany przycisk Eksploruj i zobaczysz widok kolekcji zawierający listę kuchni w komórkach widoku kolekcji. Instancja UICollectionReusableView (nagłówek sekcji)

zawierająca przycisk LOKALIZACJA znajduje się w górnej części ekranu. Zanim będziesz mógł wybrać kuchnię, musisz wybrać lokalizację, dotykając przycisku LOKALIZACJA.

Korzystanie z ekranu Lokalizacje

Po dotknięciu przycisku LOKALIZACJA wyświetli się ekran Lokalizacje:



Przyjrzyjmy się różnym częściom tego ekranu. Pasek nawigacyjny u góry ekranu zawiera przyciski Anuluj i Gotowe. Widok tabeli wyświetla listę lokalizacji w komórkach widoku tabeli. Musisz dotknąć wiersza, aby wybrać lokalizację, a następnie dotknąć przycisku Gotowe, aby potwierdzić. Po dotknięciu Gotowe wrócisz do ekranu Eksploruj, na którym będziesz mógł wybrać kuchnię. Możesz także dotknąć opcji Anuluj, aby powrócić do ekranu Eksploruj bez wybierania lokalizacji.

Korzystanie z ekranu Lista restauracji

Po ustawieniu lokalizacji (w tym przypadku ASPEN, Kolorado) możesz wybrać kuchnię. Spowoduje to wyświetlenie ekranu Lista restauracji:

Przyjrzyjmy się różnym częściom tego ekranu. Pasek nawigacyjny u góry ekranu zawiera przycisk Wstecz. Widok kolekcji wyświetla listę restauracji w danej lokalizacji oferujących wybraną kuchnię, w komórkach widoku kolekcji. Aby zobaczyć szczegóły, musisz dotknąć restauracji. Możesz także dotknąć przycisku Wstecz, aby powrócić do ekranu Eksploruj bez wybierania restauracji.

Korzystanie z ekranu szczegółów restauracji

Dotknięcie restauracji na ekranie Lista restauracji powoduje wyświetlenie szczegółów tej restauracji na ekranie Szczegóły restauracji:

Przyjrzyjmy się różnym częściom tego ekranu. Pasek nawigacyjny u góry ekranu zawiera przycisk pokazujący lokalizację (w tym przypadku ASPEN, CO). Widok tabeli wyświetla lokalizację restauracji, ocenę, recenzje klientów, recenzje zdjęć i mapę lokalizacji w komórkach widoku tabeli. Możesz dotknąć przycisku ASPEN, CO, aby powrócić do ekranu Lista restauracji lub dotknąć przycisku Dodaj recenzję lub Dodaj zdjęcie, aby wyświetlić ekrany Formularz recenzji lub Filtr zdjęć.

Korzystanie z ekranu formularza recenzji

Kliknięcie przycisku Dodaj recenzję powoduje wyświetlenie ekranu formularza recenzji:

Przyjrzyjmy się różnym częściom tego ekranu. Pasek nawigacyjny u góry ekranu zawiera przyciski Anuluj i Zapisz. Widok tabeli wyświetla ocenę i pola tekstowe w komórkach widoku tabeli. Na tym ekranie możesz ustawić ocenę i napisać recenzję restauracji. Następnie możesz dotknąć przycisku Zapisz, aby zapisać swoją ocenę i recenzję, lub przycisku Anuluj, aby powrócić do ekranu Szczegóły restauracji bez zapisywania.

Korzystanie z ekranu Filtr zdjęć

Kliknięcie przycisku Dodaj zdjęcie powoduje wyświetlenie ekranu Filtr zdjęć:

<img src=”images/ios57.png”>

Przyjrzyjmy się różnym częściom tego ekranu. Pasek nawigacyjny u góry ekranu zawiera przyciski Anuluj, Aparat i Zapisz. Widok obrazu wyświetla obraz, a widok kolekcji wyświetla filtry fotograficzne w komórkach widoku kolekcji. Na tym ekranie możesz wybrać zdjęcie i zastosować do niego filtr. Następnie możesz dotknąć przycisku Zapisz, aby zapisać zdjęcie, lub przycisku Anuluj, aby powrócić do ekranu Szczegóły restauracji bez zapisywania

Korzystanie z ekranu Mapa

Dotknięcie przycisku Mapa na pasku zakładek powoduje wyświetlenie ekranu Mapa:

Przyjrzyjmy się różnym częściom tego ekranu. Na pasku zakładek u dołu ekranu znajdują się przyciski Eksploruj i Mapa. Przycisk Mapa jest zaznaczony i widzisz instancję MKMapView (widok mapy) wyświetlającą na ekranie mapę ze znacznikami wskazującymi lokalizacje restauracji. Kliknięcie pinezki wyświetli adnotację, a dotknięcie przycisku w adnotacji wyświetli Ekran szczegółów restauracji dla tej restauracji. Na tym kończy się prezentacja aplikacji. Teraz czas zacząć tworzyć interfejs użytkownika dla swojej aplikacji!

Tworzenie nowego projektu Xcode

Teraz, gdy wiesz, jak będą wyglądać ekrany aplikacji, możesz rozpocząć tworzenie aplikacji. Zacznijmy od stworzenia nowego projektu. Jest to ten sam proces, którego użyłeś do utworzenia projektu ExploringXcode. Wykonaj następujące kroki:

1. Uruchom Xcode i kliknij Utwórz nowy projekt Xcode.

2. iOS powinien być już wybrany. Wybierz aplikację i kliknij Dalej.

3. Wyświetlony zostanie ekran Wybierz opcje dla nowego projektu:

<img src=”images/ios59.png”>

Wprowadź informacje, jak pokazano:

Nazwa produktu: LetsEat

Nazwa organizacji: Twoje własne imię i nazwisko.

Identyfikator organizacji: com. po którym następuje Twoje imię i nazwisko.

Interfejs użytkownika: Storyboard

Pozostałe ustawienia pozostaw na wartościach domyślnych. Kliknij Następny.

4. Wybierz lokalizację, w której chcesz zapisać projekt i kliknij Utwórz.

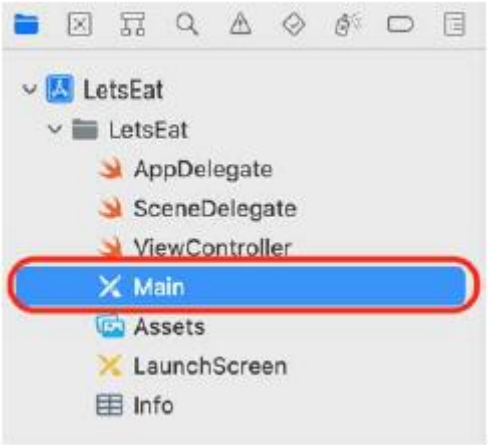
5. Jako urządzenia testowego będziesz używać symulatora iOS iPhone SE (3. generacji). W menu Schemat wybierz symulator iPhone'a SE (3. generacji).

Kompiluj i uruchamiaj swoją aplikację. Zobaczysz pusty biały ekran. Jeśli klikniesz plik głównego scenorysu w nawigatorze projektu, zobaczysz, że zawiera on pojedynczą scenę zawierającą pusty widok. Dlatego po uruchomieniu aplikacji widzisz tylko pusty biały ekran. Aby skonfigurować interfejs użytkownika, zmodyfikuj główny plik scenorysu za pomocą Konstruktora interfejsów. Konstruktor interfejsów umożliwia dodawanie i konfigurowanie scen. Każda scena reprezentuje ekran, który zobaczy użytkownik. Możesz dodać do sceny obiekty interfejsu użytkownika, takie jak widoki i przyciski, i skonfigurować je zgodnie z wymaganiami, korzystając z Inspektora atrybutów. Po utworzeniu projektu dodasz do niego scenę kontrolera paska kart. Ta scena wyświetla pasek kart z dwiema zakładkami na dole ekranu. Dotknięcie karty spowoduje wyświetlenie powiązanego z nią ekranu. Ekrany te odpowiadają ekranom Eksploruj i Mapa pokazanym w przewodniku po aplikacji. Zobaczmy, jak to zrobić w następnej sekcji.

Konfigurowanie sceny kontrolera paska zakładek

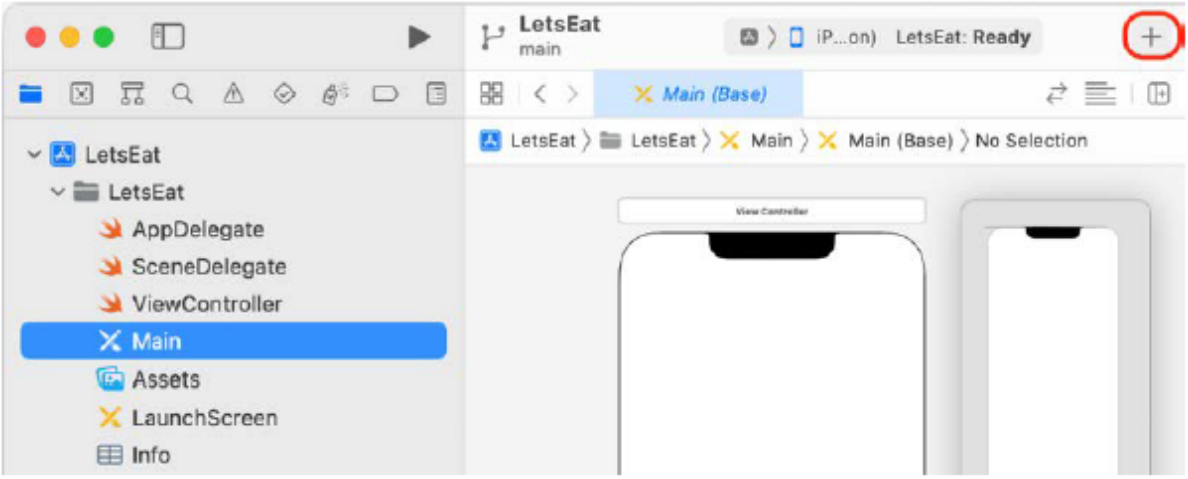
Jak widzieliście podczas prezentacji aplikacji, aplikacja Let’s Eat ma pasek kart z dwoma przyciskami u dołu ekranu, które służą do wyświetlania ekranów Eksploruj i Mapa. Usuniesz istniejącą scenę kontrolera widoku i plik ViewController Swift i dodasz do projektu scenę kontrolera paska kart z dwoma przyciskami. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu w nawigatorze projektu:



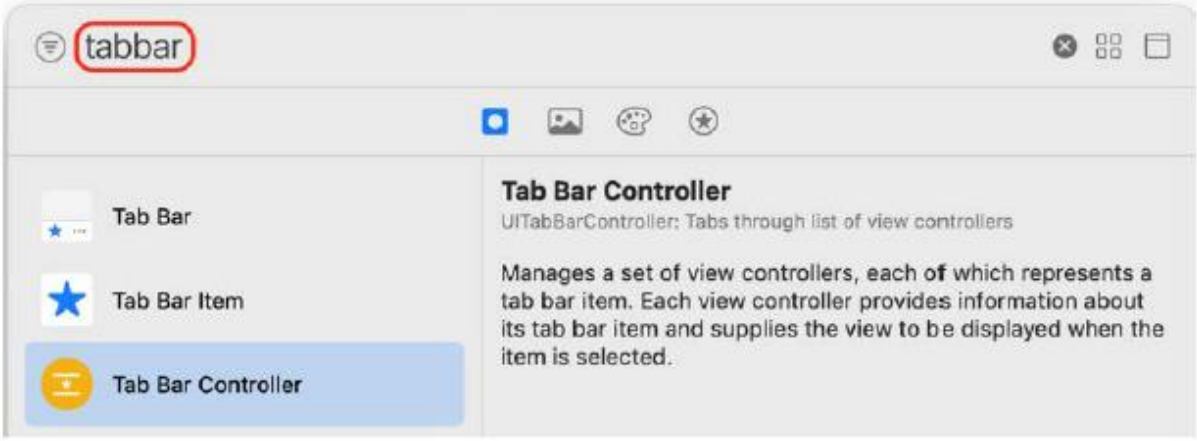
2. Zawartość głównego pliku scenorysu pojawi się w obszarze Edytora. Kliknij przycisk Konspekt dokumentu, aby zwinąć konspekt dokumentu, jeśli jest obecny. Daje to więcej miejsca do pracy z:

3. Kliknij przycisk +, aby otworzyć bibliotekę:

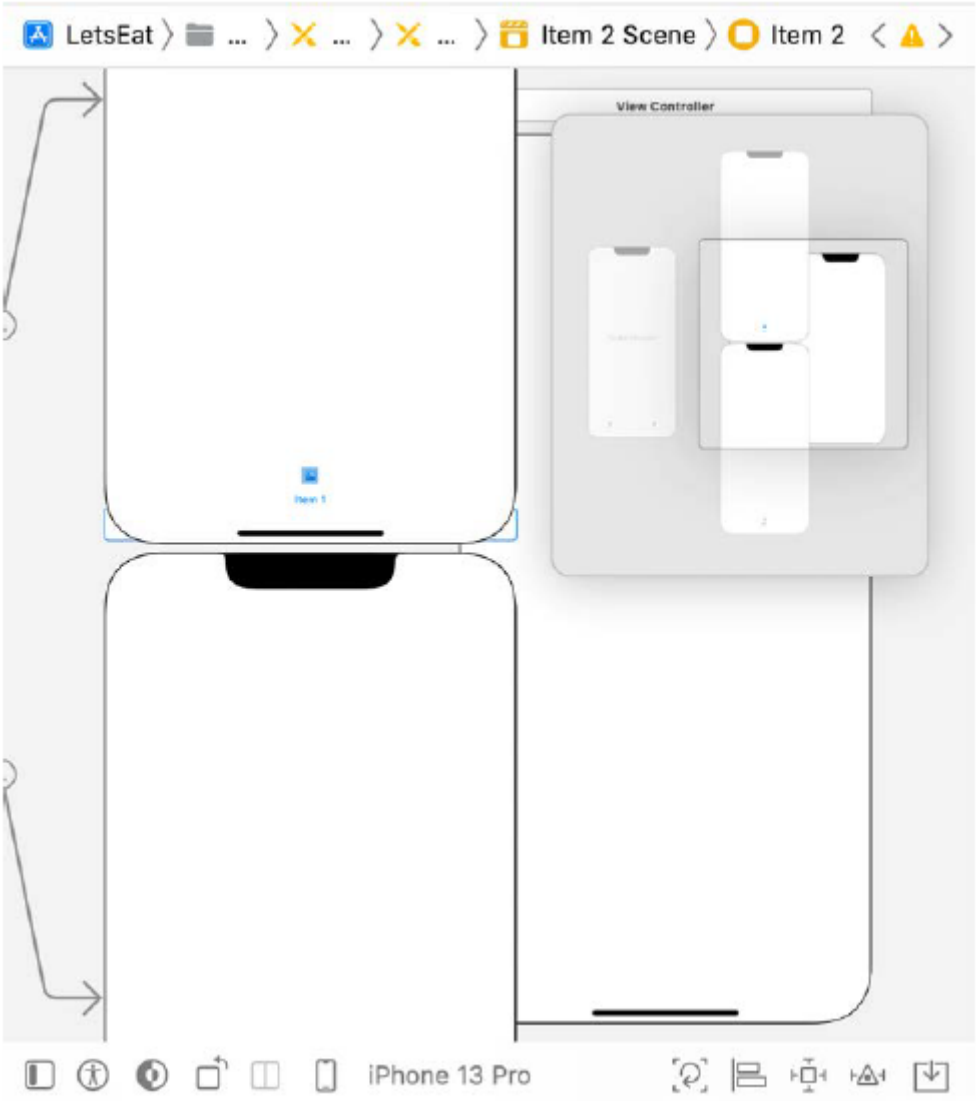


Biblioteka umożliwia wybranie obiektów interfejsu użytkownika, które mają zostać dodane do sceny.

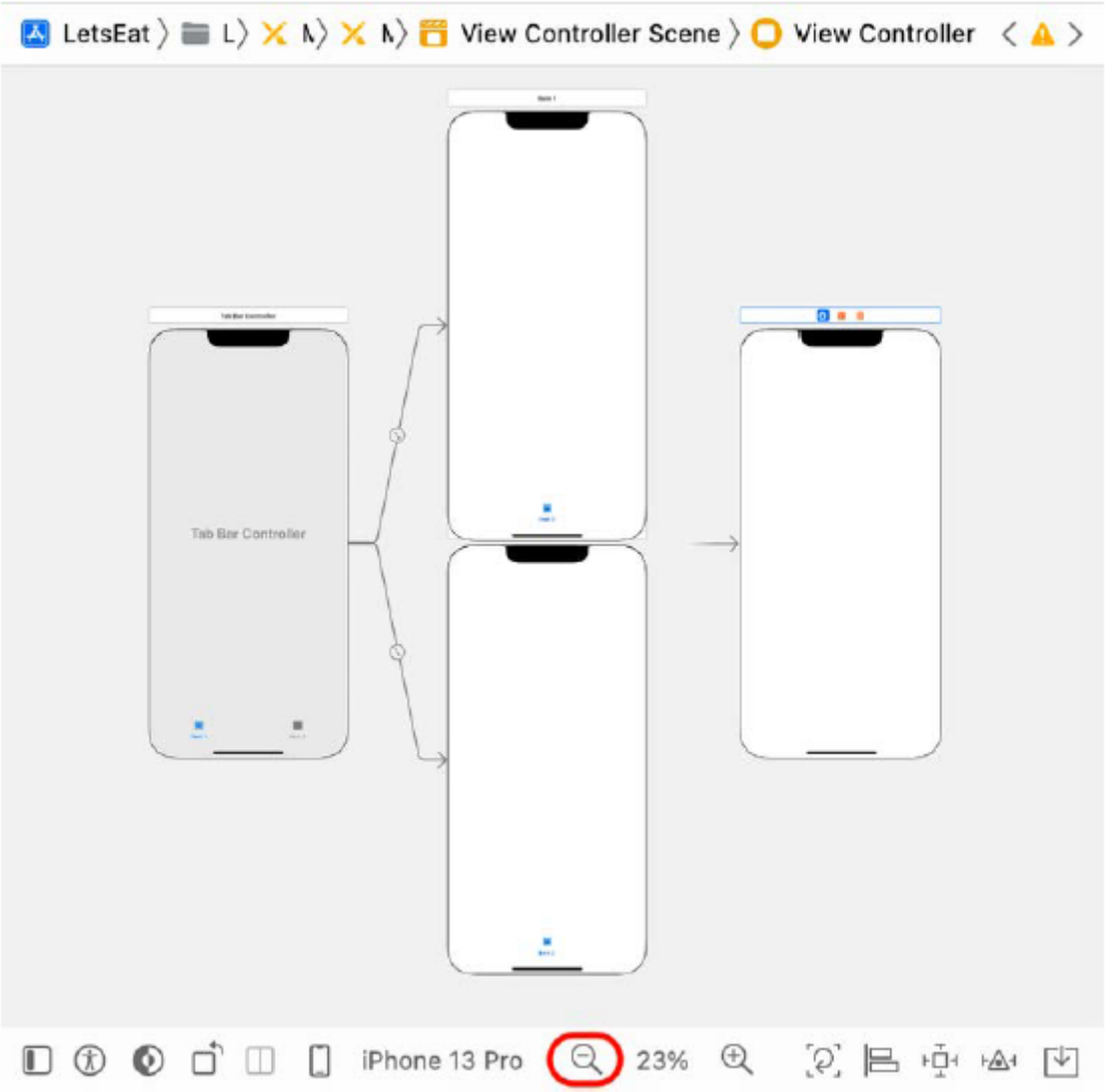
4. Wpisz tabbar w polu filtra. Na liście wyników pojawi się obiekt Tab Bar Controller:



5. Przeciągnij obiekt Kontroler paska kart do scenorysu, aby dodać nową scenę kontrolera paska kart. Nie ma problemu, jeśli obejmuje istniejącą scenę kontrolera widoku. Jak widać, składa się ze sceny z dwiema strzałkami przedstawiającymi przejścia prowadzące do dwóch kolejnych scen:



6. Kliknij przycisk -, aby pomniejszyć i zmienić układ scen w scenorysie tak, aby widoczna była zarówno scena kontrolera paska zakładek, jak i scena kontrolera widoku:



7. Wybierz strzałkę wskazującą scenę kontrolera widoku, jak pokazano. Ta strzałka określa początkową scenę kontrolera widoku Twojej aplikacji, dzięki czemu widok pojawia się po uruchomieniu aplikacji:

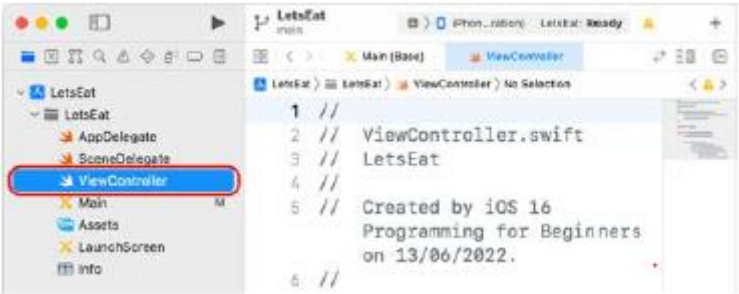
8. Przeciągnij strzałkę ze sceny kontrolera widoku do sceny kontrolera paska kart, jak pokazano. To sprawia, że scena kontrolera paska kart jest sceną początkową, a pasek kart pojawi się po uruchomieniu aplikacji:

Możesz także ustawić scenę kontrolera paska zakładek jako scenę kontrolera widoku początkowego, zaznaczając ją, klikając przycisk Inspektora atrybutów i zaznaczając pole wyboru Kontroler widoku początkowego. Więcej o Inspektorze atrybutów dowiesz się w następnej sekcji, Ustawianie tytułów przycisków paska zakładek.

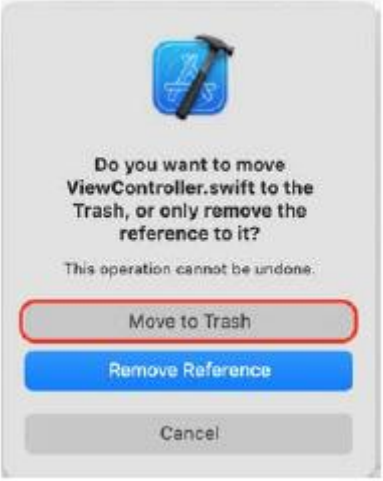
9. Wybierz istniejącą scenę kontrolera widoku w scenorysie i naciśnij klawisz Delete na klawiaturze, aby ją usunąć, ponieważ nie będziesz jej używać w tym projekcie.

10. Wybierz plik ViewController Swift w nawigatorze projektu i naciśnij klawisz Delete na klawiaturze, aby go usunąć, ponieważ nie będziesz go używać w tym projekcie:

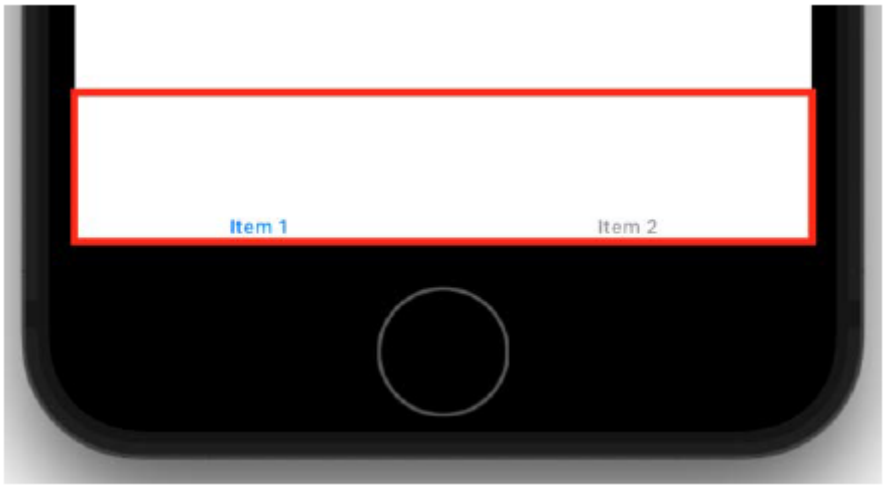
<img src="images/ios68.png">



11. Kliknij opcję Przenieś do kosza w wyskakującym oknie dialogowym:



Zbuduj i uruchom swoją aplikację w symulatorze iOS, a na dole ekranu zobaczysz pasek kart z dwoma przyciskami:

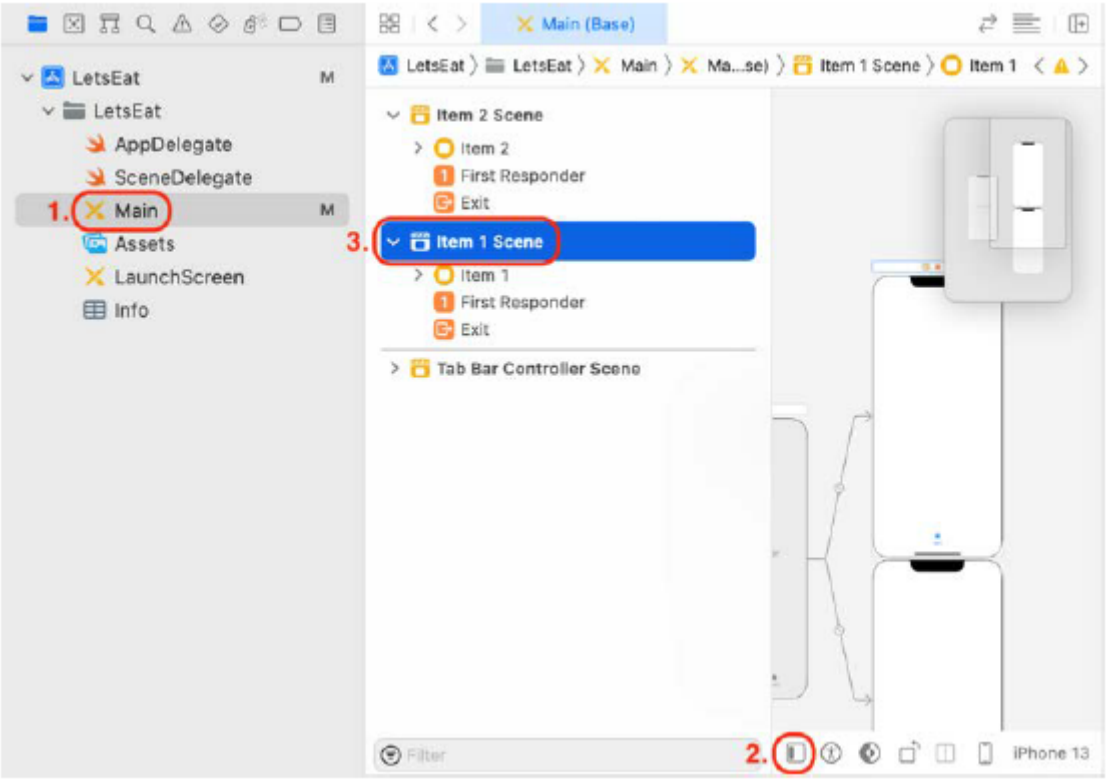


Pomyślnie dodałeś pasek kart do swojego projektu, ale jak widzisz, tytuły przycisków to obecnie Element 1 i Element 2. W następnej sekcji zmienisz je na Eksploruj i Mapa.

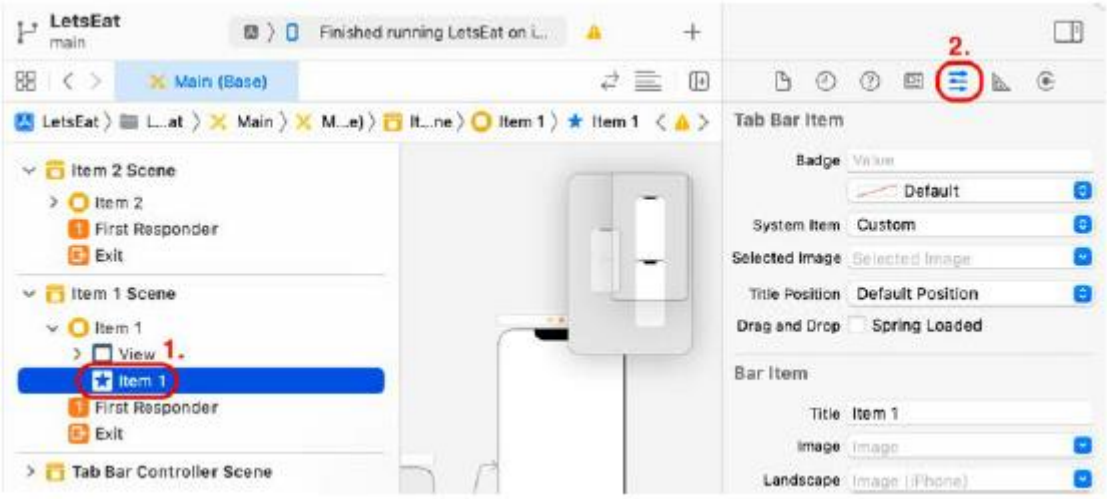
Ustawianie tytułów przycisków paska zakładek

Twoja aplikacja wyświetla teraz pasek kart na dole ekranu, ale tytuły przycisków nie odpowiadają tytułom pokazanym w przewodniku po aplikacji. Aby je dopasować, w Inspektorze atrybutów skonfigurujesz tytuły przycisków tak, aby brzmiały Eksploruj i Mapuj. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu w nawigatorze projektu. Kliknij przycisk Konspekt dokumentu, aby wyświetlić konspekt dokumentu. Kliknij Element 1 Scena w konspekcie dokumentu:

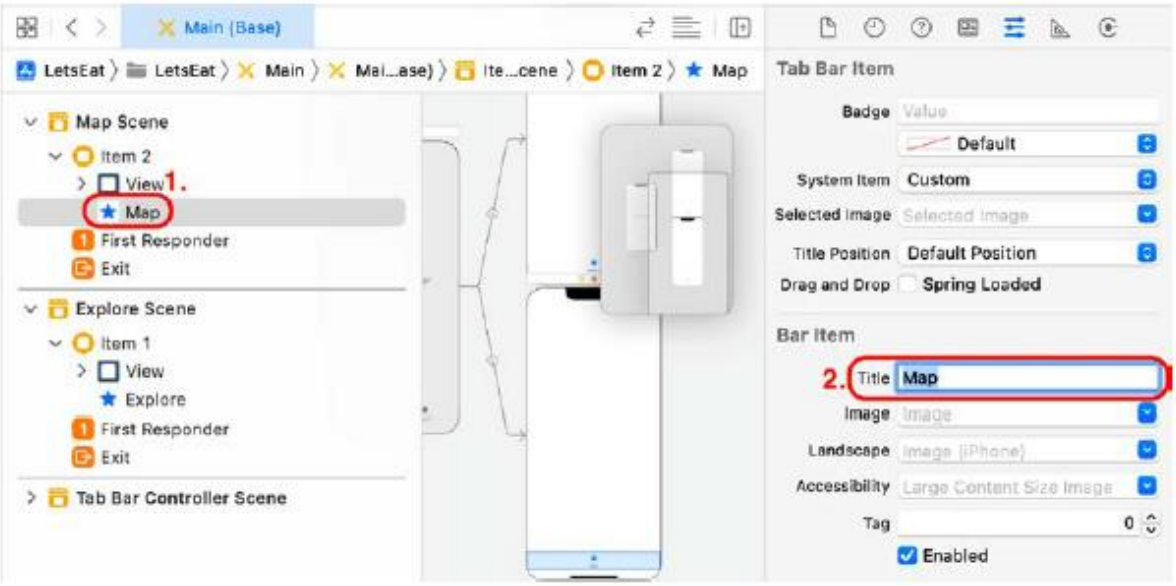


2. Kliknij przycisk Element 1 pod sceną Element 1. Kliknij przycisk Inspektora atrybutów:



3. W obszarze Element paska ustaw Tytuł na Eksploruj:

4. Kliknij przycisk Element 2 w scenie Element 2 i w obszarze Element paska ustaw Tytuł na Mapa:



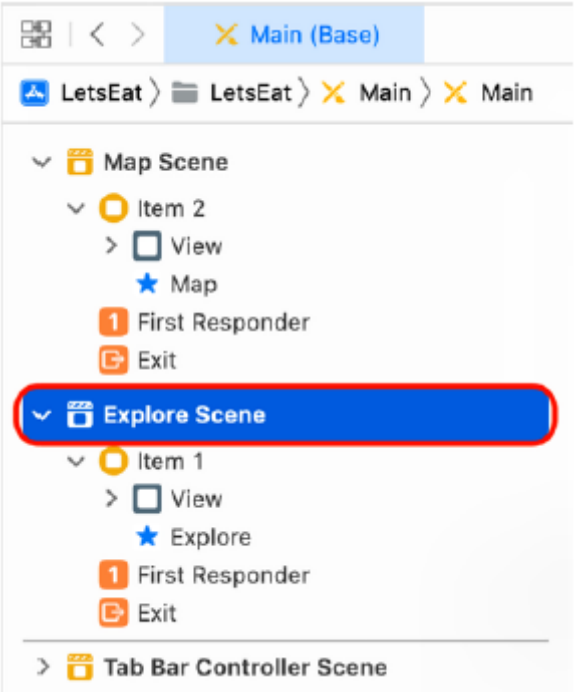
Zbuduj i uruchom swoją aplikację w symulatorze. Zobaczysz, że tytuły przycisków zmieniły się odpowiednio na Eksploruj i Mapa. Świetnie! Dotknięcie przycisków Eksploruj i Mapa spowoduje wyświetlenie scen ekranów Eksploracja i Mapa. Jak pokazano w przewodniku po aplikacji, jeśli dotkniesz przycisku LOKALIZACJA na ekranie Eksploruj, u góry ekranu Lokalizacje pojawi się pasek nawigacyjny zawierający przyciski Anuluj i Gotowe. Zobaczysz także pusty pasek nawigacyjny u góry ekranu mapy. Jak widziałeś podczas prezentacji aplikacji, niektóre ekrany mają tytuły i przyciski na pasku nawigacyjnym. W następnej sekcji dowiesz się, jak dodać paski nawigacyjne do swoich ekranów, aby móc później dodawać do nich przyciski i tytuły, jeśli zajdzie taka potrzeba.

Osadzanie kontrolerów widoku w kontrolerach nawigacji

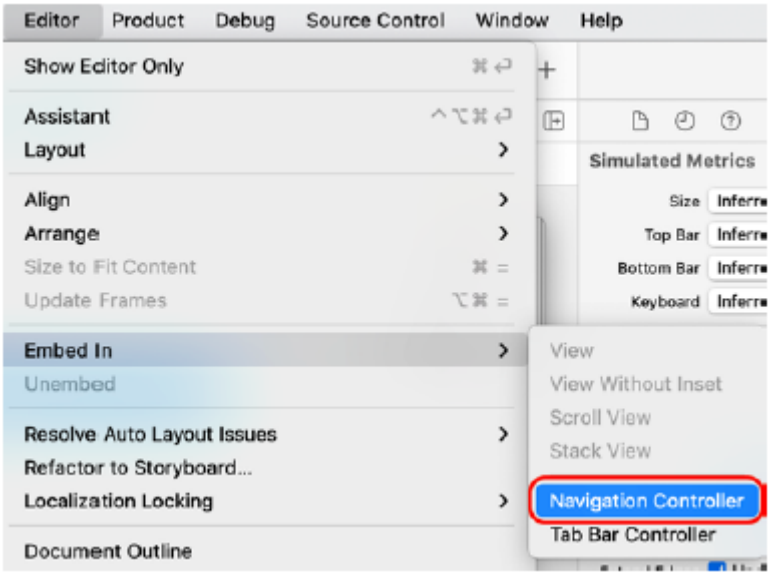
Jak widziałeś podczas prezentacji aplikacji, ekrany Eksploracja i Mapa mają pasek nawigacyjny u góry ekranu. Aby dodać paski nawigacyjne dla obu ekranów, osadź kontrolery widoku scen Eksploruj i Mapa w kontrolerze nawigacyjnym. Spowoduje to wyświetlenie pasków nawigacyjnych u góry ekranu, gdy wyświetlane są ekrany Eksploruj i Mapa.

Wykonaj następujące kroki:

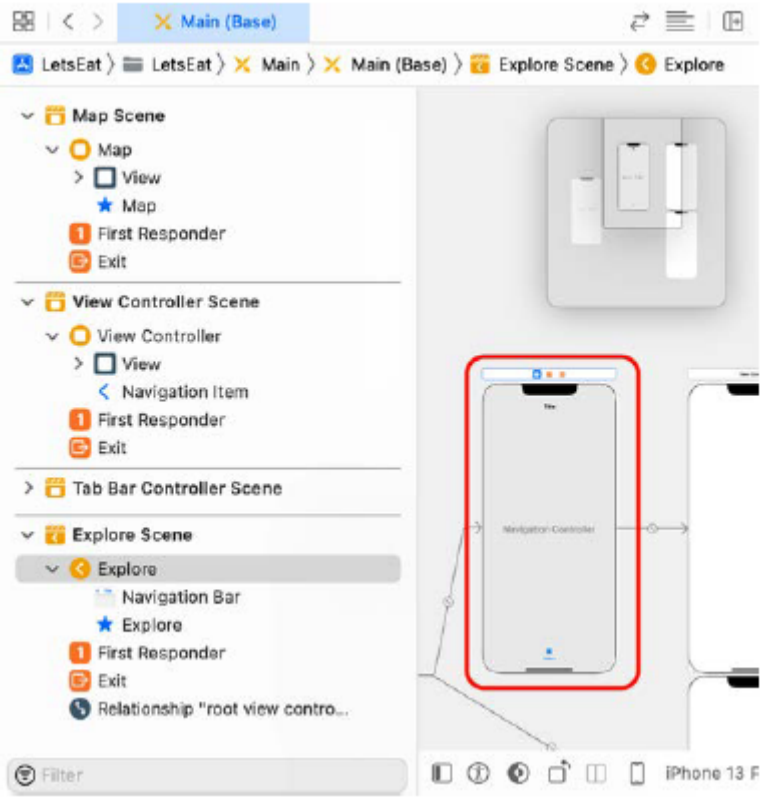
1. Kliknij Eksploruj scenę w konspekcie dokumentu:



2. Wybierz Edytor | Osadź w | Kontroler nawigacji:

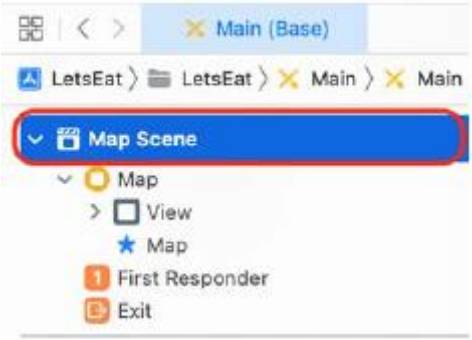


3. Pomiędzy sceną kontrolera paska zakładek a sceną eksploracji pojawia się scena kontrolera nawigacji:



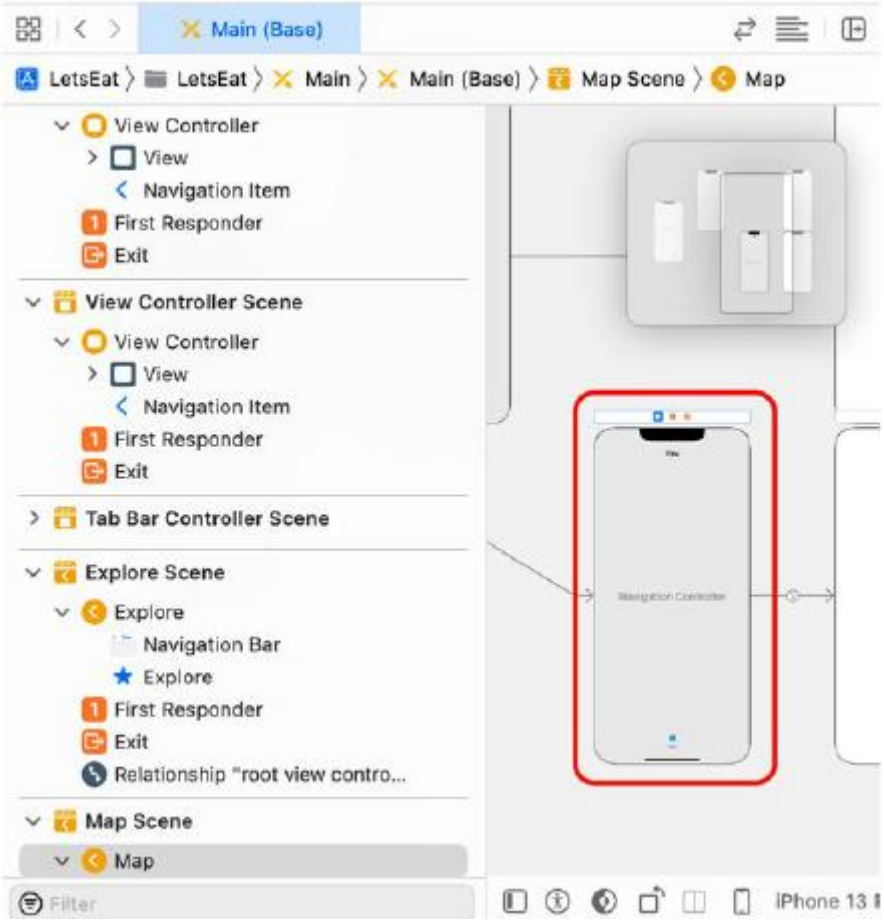
Kompiluj i uruchamiaj swoją aplikację. Ekran Eksploruj ma teraz pasek nawigacyjny, ale ponieważ ma ten sam kolor co tło, nie jest on widoczny na ekranie. Osadzanie kontrolera widoku w kontrolerze nawigacji powoduje dodanie tego kontrolera widoku do tablicy viewControllers kontrolera nawigacji. Następnie kontroler nawigacyjny wyświetla na ekranie widok kontrolera widoku. Kontroler nawigacyjny wyświetla także pasek nawigacji u góry ekranu. Ekran Mapa nie posiada jeszcze paska nawigacyjnego. Dodajmy teraz jedno. Wykonaj następujące kroki:

1. Kliknij Scenę mapy w konspekcie dokumentu:



2. Wybierz Edytor | Osadź w | Kontroler nawigacji.

3. Scena kontrolera nawigacji pojawia się pomiędzy sceną kontrolera paska zakładek a sceną mapy:

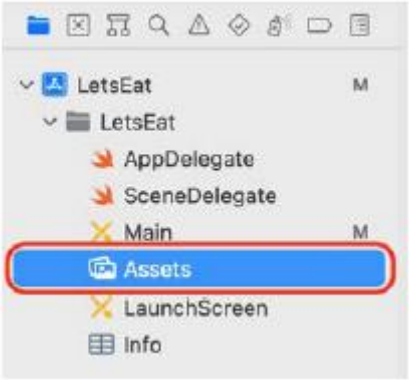


Kompiluj i uruchamiaj swoją aplikację. Zarówno ekrany Eksploracji, jak i Mapa mają teraz pasek nawigacyjny, chociaż w tej chwili ich nie widzisz. Przyciski paska kart umożliwiają przełączanie między ekranami Eksploracja i Mapa, a każdy ekran ma teraz pasek nawigacyjny. Tytuły przycisków są poprawne, ale same przyciski nie mają ikon. Aby uzyskać ikony przycisków, w następnej sekcji dodasz plik zawierający wszystkie zasoby graficzne wymagane dla Twojego projektu.

Dodanie pliku Assets.xcassetsM

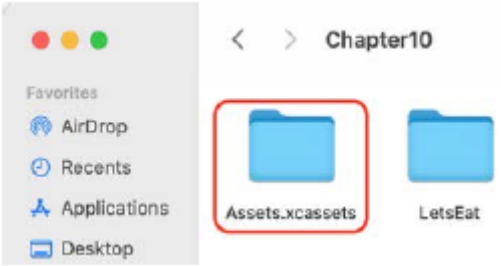
Plik Assets.xcassets zawiera zasoby Twojego projektu, takie jak ikony aplikacji i niestandardowe obrazy. Obecnie jest pusty, ponieważ właśnie utworzyłeś ten projekt. Jeśli jeszcze tego nie zrobiłeś, musisz pobrać plik Assets.xcassets .Po pobraniu pliku możesz dodać go do swojego projektu, wykonując następujące kroki:

1. Musisz usunąć istniejący plik Assets.xcassets (pokazany jako Assets w nawigatorze projektu) ze swojego projektu. Zaznacz go w nawigatorze projektu i naciśnij klawisz Delete, aby go usunąć:

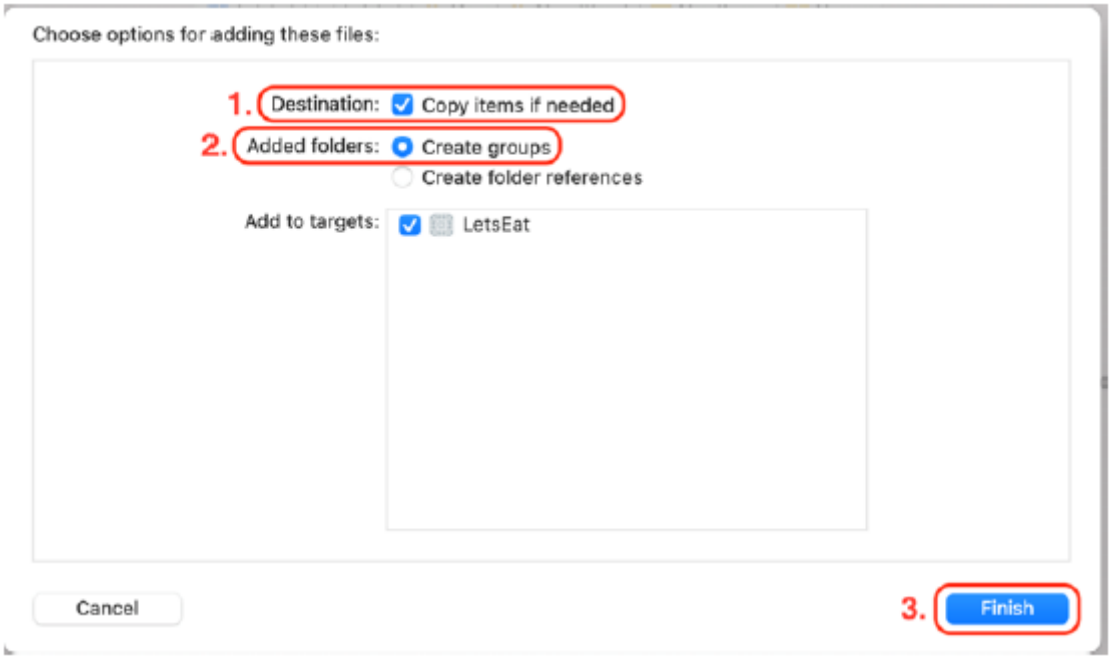


2. Kliknij opcję Przenieś do kosza w wyskakującym oknie dialogowym.

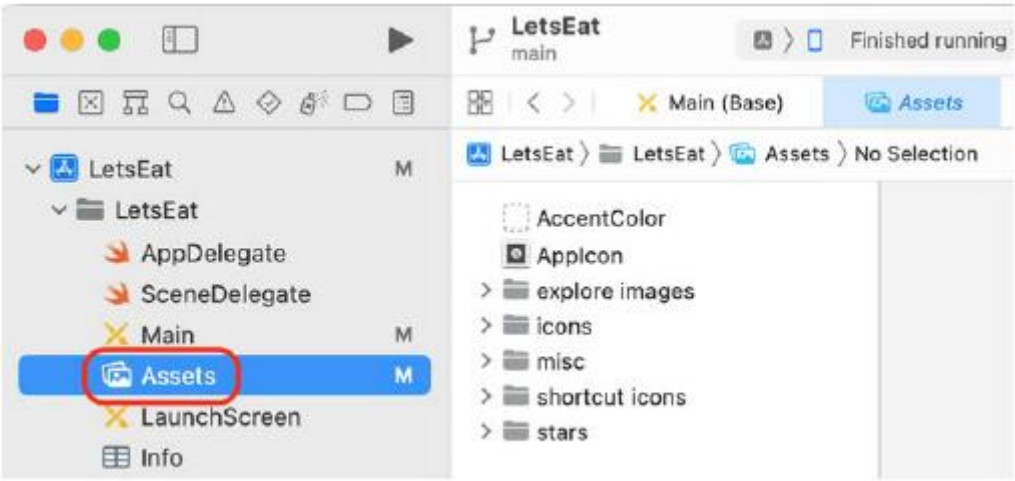
3. Otwórz folder Chapter10 wewnątrz pobranych plików pakietu kodu. Wewnątrz zobaczysz Assets.xcassets:



4. Przeciągnij nowy plik Assets.xcassets do obszaru nawigatora projektu. Pojawi się okno dialogowe Wybierz opcje dodawania tych plików. Zaznacz pole wyboru Kopiuj elementy w razie potrzeby. Zaznacz przycisk opcji Utwórz grupy. Pozostałe ustawienia pozostaw na wartościach domyślnych. Kliknij Zakończ:



5. Do Twojego projektu dodano plik Assets.xcassets. Należy pamiętać, że jest on wyświetlany jako Zasoby w nawigatorze projektu. Kliknij go, aby zobaczyć jego zawartość:

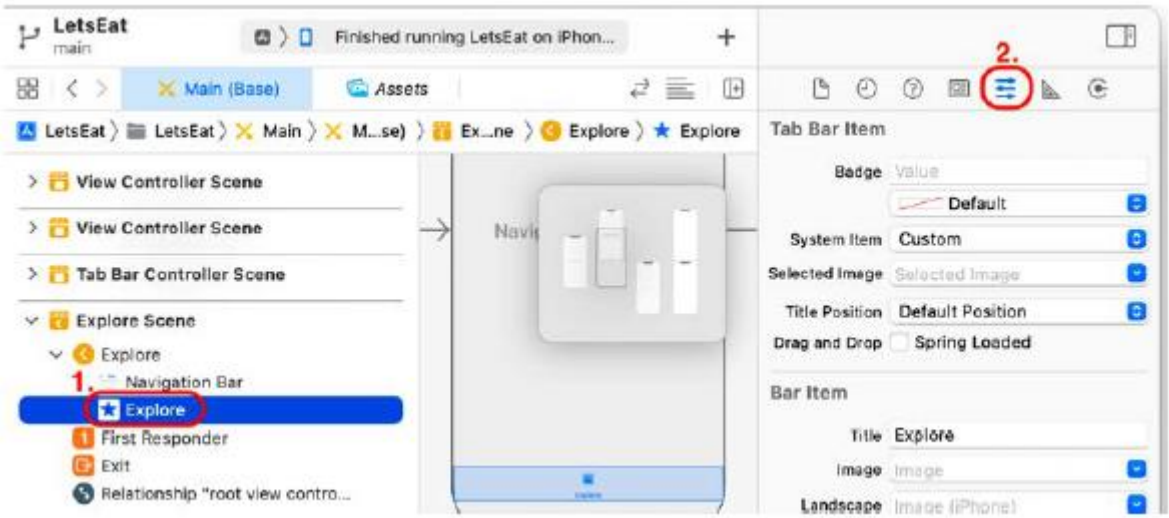


Wśród dołączonych zasobów graficznych znajdują się ikony przycisków paska zakładek. W następnej sekcji dodasz ikony przycisków Eksploruj i Mapa.

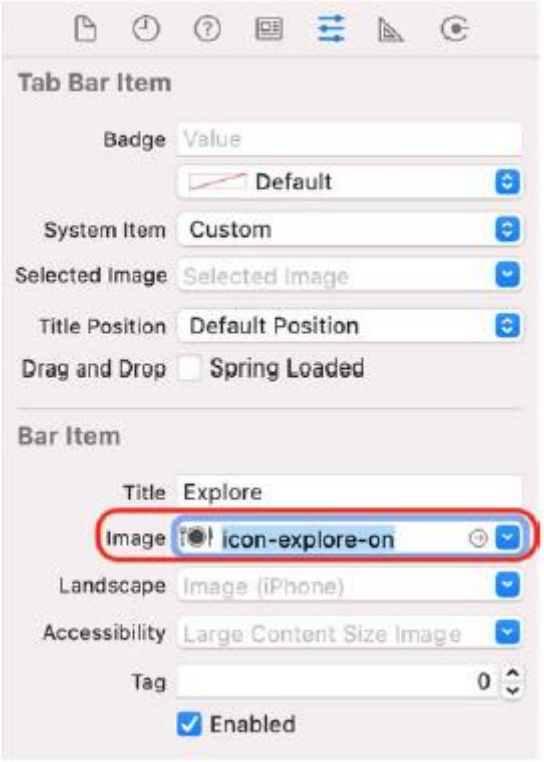
Dodanie ikon przycisków Eksploruj i Mapa

Przyciski Eksploruj i Mapa nie mają obecnie ikon. Ikony te znajdują się w folderze Assets.xcassets. Dodaj je teraz do przycisków, wykonując następujące kroki:

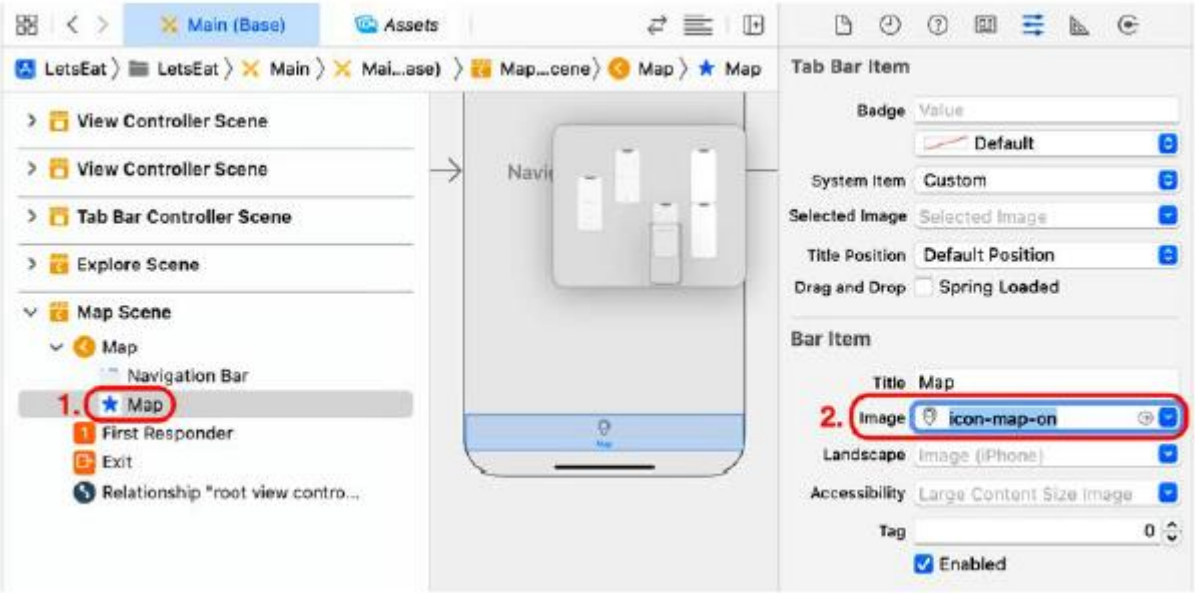
1. Kliknij plik głównego scenorysu. Kliknij przycisk Eksploruj w obszarze Eksploruj scenę w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów:



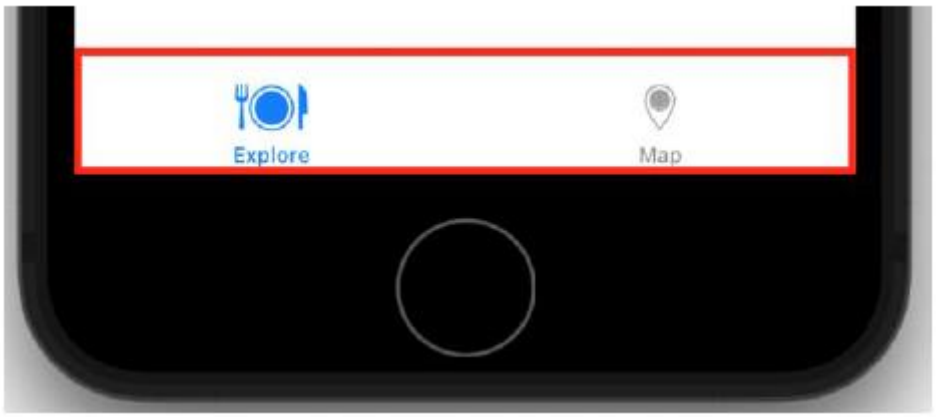
2. W obszarze Element paska ustaw Obraz na opcję Eksploruj ikony:



3. Kliknij przycisk Mapa w obszarze Scena mapy. W obszarze Element paska ustaw Obraz na ikonę-mapę:



Kompiluj i uruchamiaj swoją aplikację. Możesz zobaczyć, że przyciski Eksploruj i Mapa mają teraz ikony:



Gratulacje! Właśnie skonfigurowałeś pasek kart dla swojej aplikacji! Po uruchomieniu aplikacji może przez chwilę pojawić się biały ekran, a następnie pasek kart. Ten ekran nazywa się ekranem uruchamiania i jest wyświetlany przez chwilę podczas uruchamiania aplikacji. W następnej sekcji dowiesz się, jak skonfigurować ten ekran, aby wyświetlał niestandardowy kolor i logo aplikacji.

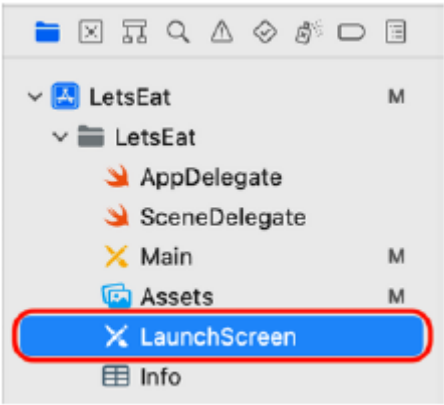
Konfigurowanie ekranu uruchamiania

Ekran uruchamiania oznacza ekran wyświetlany na chwilę podczas uruchamiania aplikacji. Możesz to skonfigurować, modyfikując plik scenorysu LaunchScreen w swoim projekcie. Ten plik jest tworzony automatycznie podczas tworzenia projektu Xcode. Utworzysz nowy, niestandardowy kolor dla tego ekranu, dodasz ikonę z folderu Assets.xcassets i ustawisz lokalizację ikony, korzystając z ograniczeń automatycznego układu. W następnej sekcji zaczniesz od utworzenia nowego niestandardowego koloru.

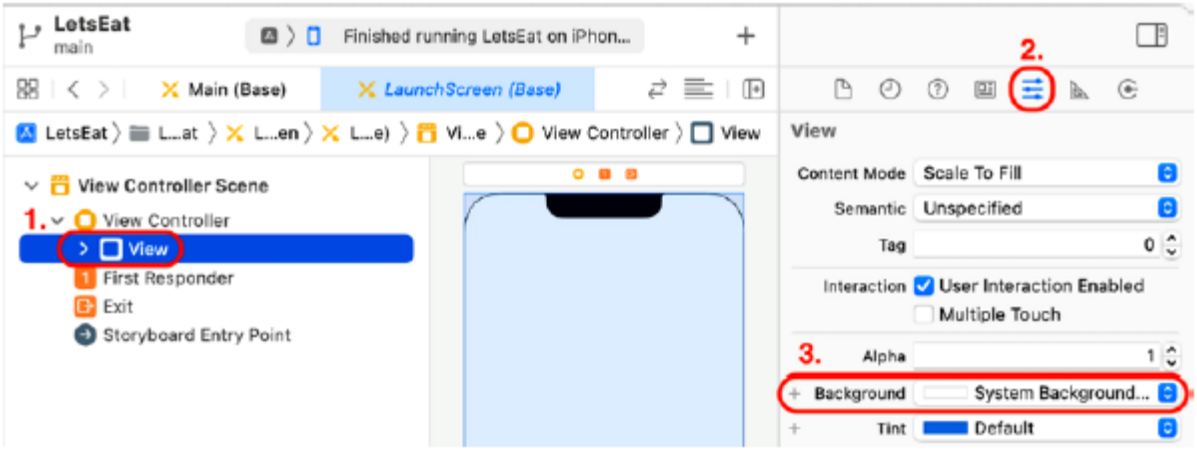
Konfigurowanie koloru tła ekranu uruchamiania

Inspektora atrybutów można używać do modyfikowania kolorów elementów interfejsu użytkownika na ekranie. Możesz określić dokładny kolor, używając niestandardowych wartości koloru czerwonego, zielonego i niebieskiego. Aby ustawić niestandardowy kolor ekranu uruchamiania, wykonaj następujące kroki:

1. Kliknij plik scenorysu LaunchScreen w nawigatorze projektu:

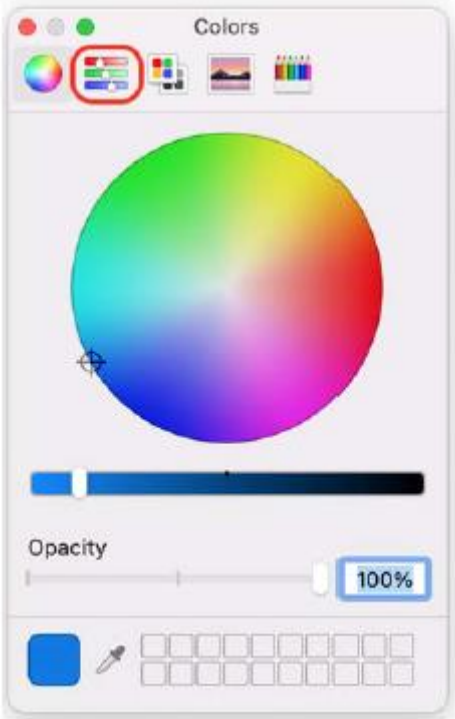


2. Wybierz opcję Widok w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. W obszarze Widok kliknij menu podręczne Tło:

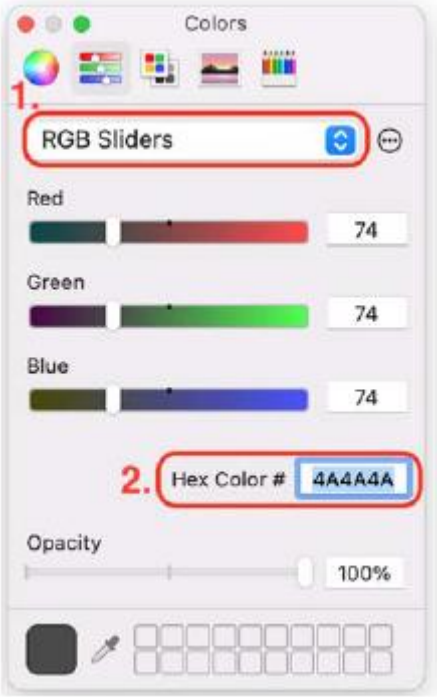


3. Wybierz opcję Niestandardowe... z wyskakującego menu:

4. W próbniku kolorów wybierz drugą zakładkę (tę z trzema suwakami):



5. Z wyskakującego menu wybierz opcję Suwaki RGB i wpisz 4A4A4A w polu Kolor szesnastkowy #:

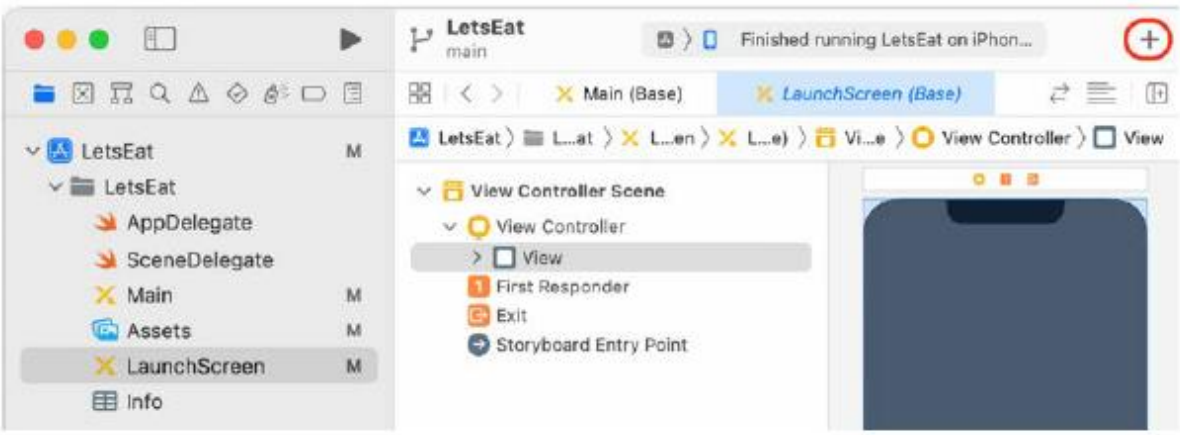


Kolor składa się z czerwieni, zieleni i błękitu. Zakres wartości każdego koloru wynosi od 0 do 255. Ta wartość szesnastkowa ustawia wartości czerwonego, zielonego i niebieskiego na 74, co daje przyjemny ciemnoszary kolor. Kompiluj i uruchamiaj swoją aplikację. Powinieneś przez chwilę zobaczyć ciemnoszary ekran, zanim pojawi się pasek kart. Fajny! W następnej sekcji dodasz logo do tego ekranu i użyjesz ograniczeń układu automatycznego, aby umieścić je dokładnie na środku ekranu, niezależnie od urządzenia i orientacji. Dodanie logo informuje użytkownika, że aplikacja się uruchamia, i jest jednym ze sposobów na poprawę wyglądu aplikacji.

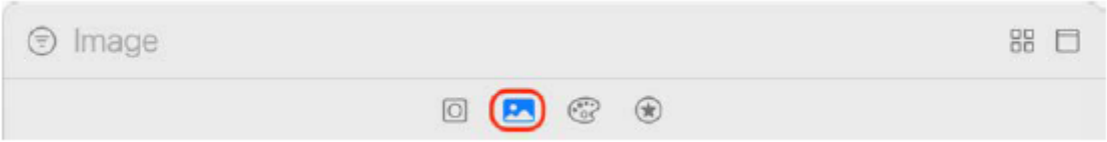
Dodanie logo i ograniczeń do ekranu uruchamiania

Twój ekran startowy jest raczej prosty, więc dodajmy do niego logo. Logo aplikacji Let's Eat znajdziesz w pliku Assets.xcassets, który wcześniej dodałeś do swojego projektu. Użyjesz także ograniczeń układu automatycznego, aby umieścić logo na środku ekranu. Wykonaj następujące kroki:

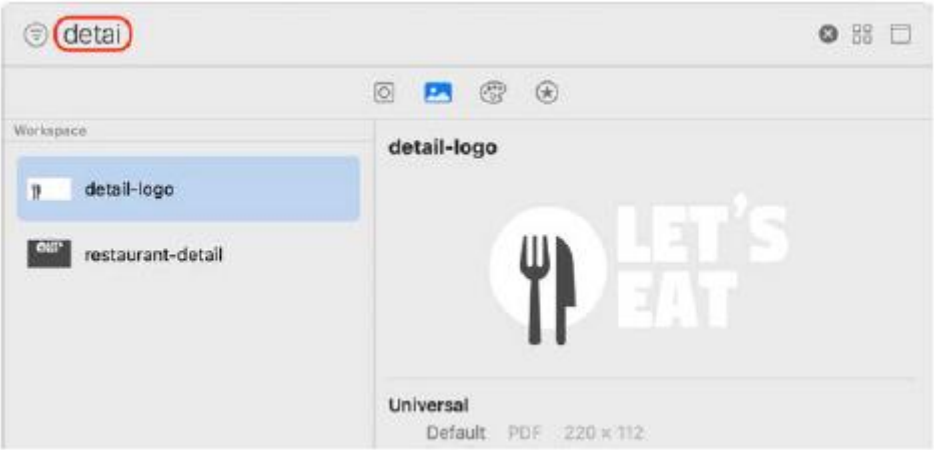
1. W nawigatorze projektu powinien być nadal zaznaczony plik scenorysu LaunchScreen. Kliknij przycisk +, aby wyświetlić bibliotekę:



2. Kliknij przycisk Media, aby wyświetlić wszystkie pliki graficzne w projekcie:



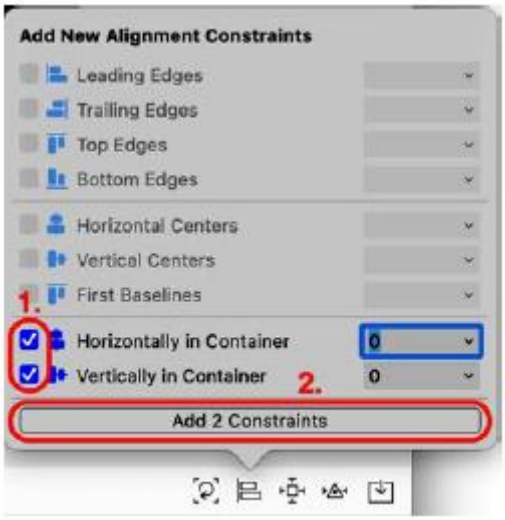
3. Wpisz szczegóły w polu filtra. W wynikach zobaczysz szczegółowe logo:



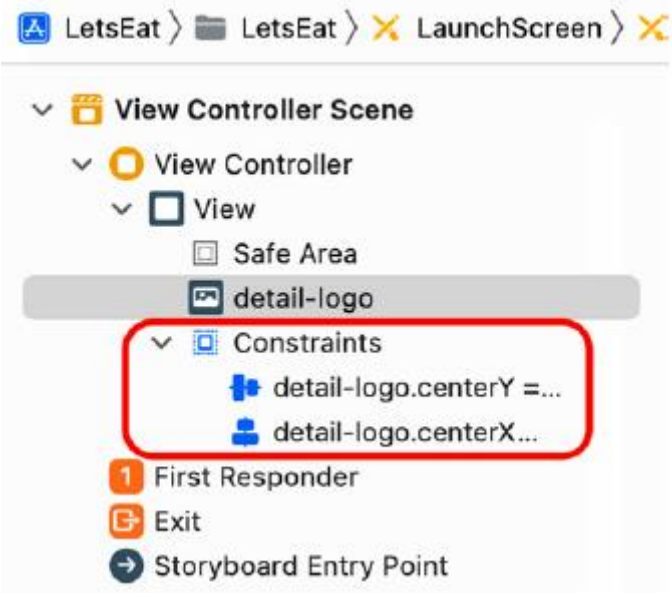
4. Przeciągnij logo szczegółów do widoku sceny kontrolera widoku i wyśrodkuj je w pionie i poziomie. Zobaczysz niebieskie wskazówki, które Ci pomogą. Gdy skończysz, kliknij przycisk Automatyczne wyrównanie układu:

Jeśli nie widzisz przycisku Automatyczne wyrównywanie układu, kliknij przycisk Inspektor na pasku narzędzi, aby ukryć obszar Inspektora.

5. Zaznacz opcję Poziomo w kontenerze i Pionowo w kontenerze. Kliknij opcję Dodaj 2 wiązania:



6. Ograniczenia zostały dodane do logo detalu i są widoczne w konspekcie dokumentu:



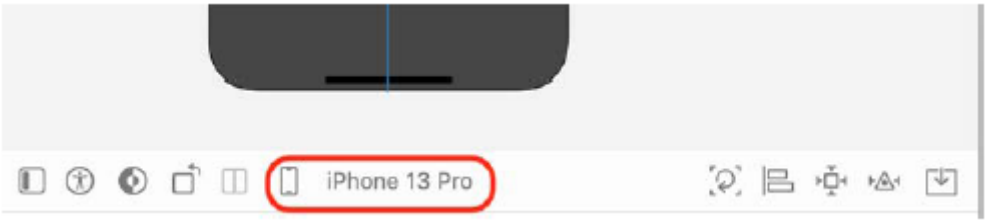
Ograniczenia określają położenie logo w stosunku do widoku kontrolera widoku. Widok kontrolera widoku jest w tym przypadku kontenerem. Opcja Horizontly in Container oblicza poziomą pozycję logo względem lewej i prawej strony kontenera, a Vertically in Container oblicza pionową pozycję logo względem górnej i dolnej części kontenera. Kompiluj i uruchamiaj swoją aplikację. Zobaczysz logo na środku ekranu. Nawet jeśli spróbujesz uruchomić aplikację w symulatorze iOS na ekranie o innym rozmiarze, logo nadal będzie znajdować się dokładnie na środku ekranu. Gratulacje! Pomyślnie skonfigurowałeś ekran uruchamiania swojej aplikacji! Być może zauważyłeś, że ekrany przedstawione w Konstruktorze interfejsów nie odpowiadają modelowi iPhone'a wybranemu w symulatorze iOS, a wyświetlanie minimapy może przeszkadzać w rozmieszczaniu ekranów w aplikacji. Wykonajmy dodatkową konfigurację dla Konstruktora interfejsów, aby to naprawić.

Konfigurowanie Konstruktora interfejsów

Nawet jeśli skonfigurowałeś iPhone'a SE (3. generacji) jako symulator iOS dla swojej aplikacji, może się okazać, że sceny pokazane w Konstruktorze interfejsów dotyczą innego modelu iPhone'a. Możesz także

chcieć , aby ukryć wyświetlanie minimapy. Skonfigurujmy Interface Builder tak, aby korzystał z ekranów z iPhone'a SE (3. generacji) i ukrywał wyświetlanie minimapy. Wykonaj następujące kroki:

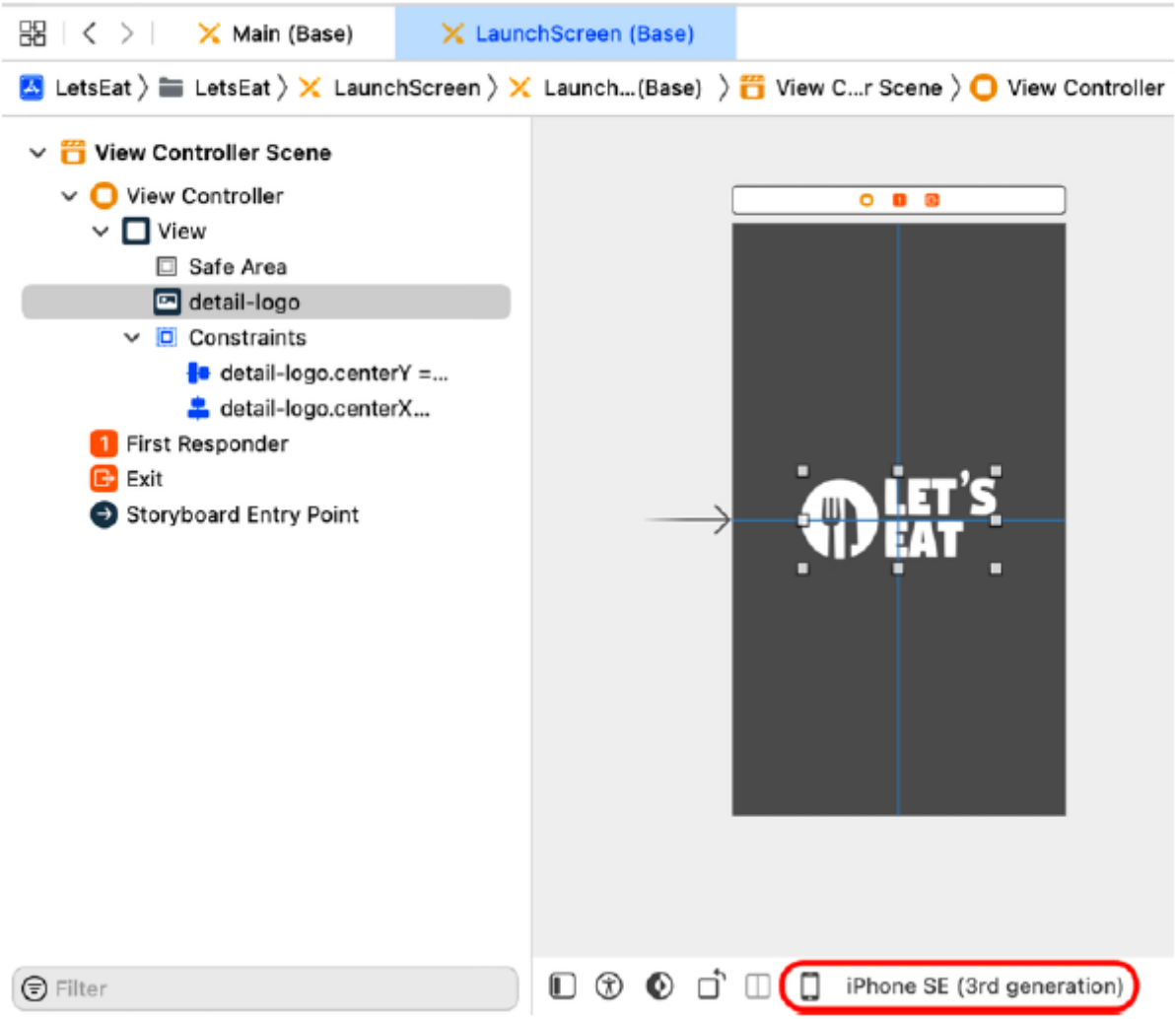
1. Plik scenorysu LaunchScreen powinien być nadal wybrany. Aby skonfigurować Konstruktora interfejsów, kliknij przycisk konfiguracji urządzenia:



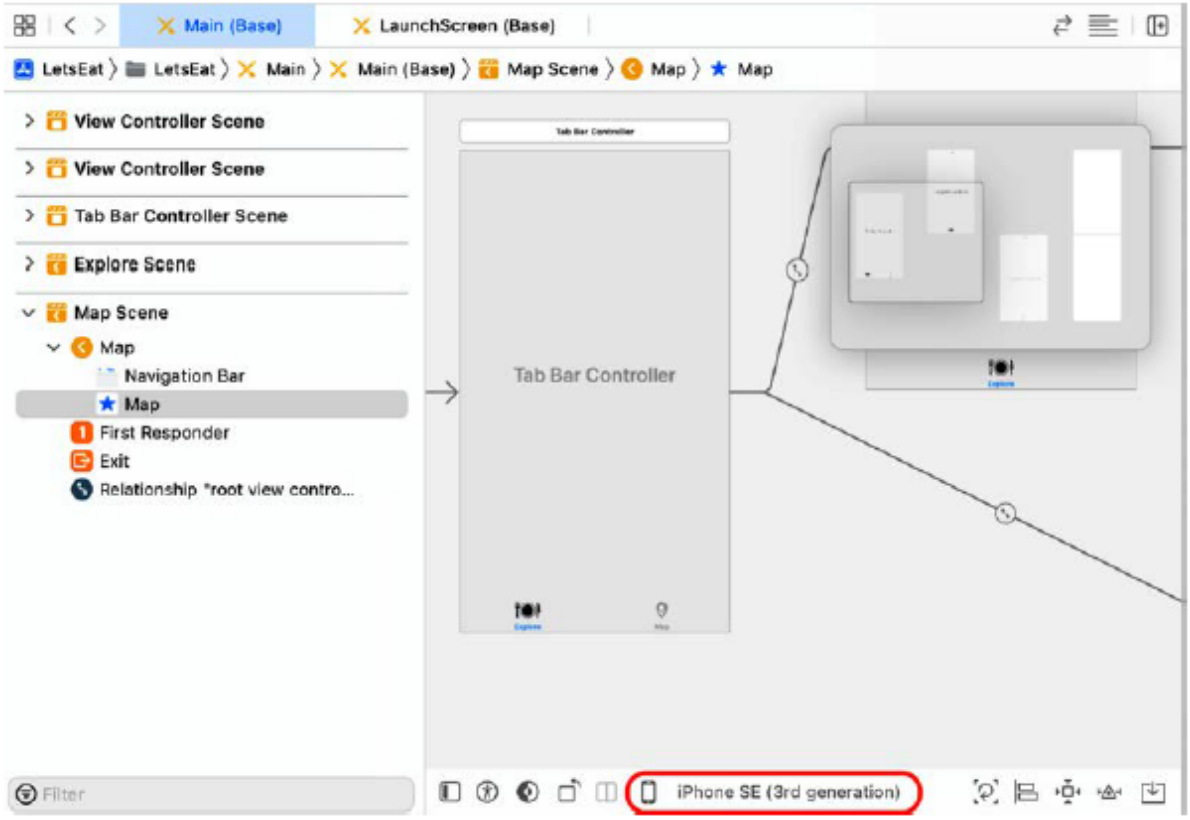
2. Pojawi się wyskakujące okienko wyświetlające różne ekrany urządzeń:

3. Z wyskakującego okna wybierz iPhone'a SE (3. generacji):

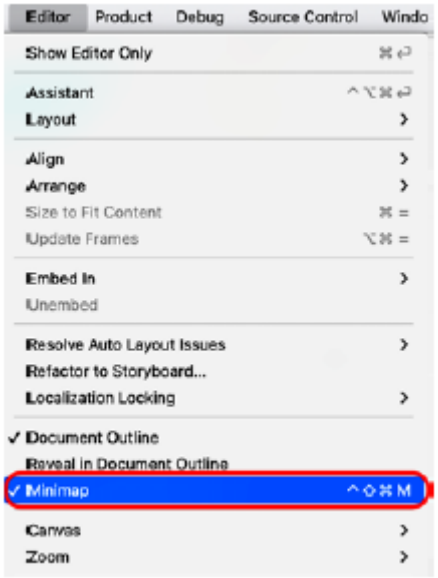
4. Po ustawieniu iPhone'a SE (3. generacji) w wyskakującym oknie urządzenia zwróć uwagę, że scena zmieniła się, aby odzwierciedlać ekran iPhone'a SE (3. generacji). Logo nadal znajduje się dokładnie na środku ekranu uruchamiania:



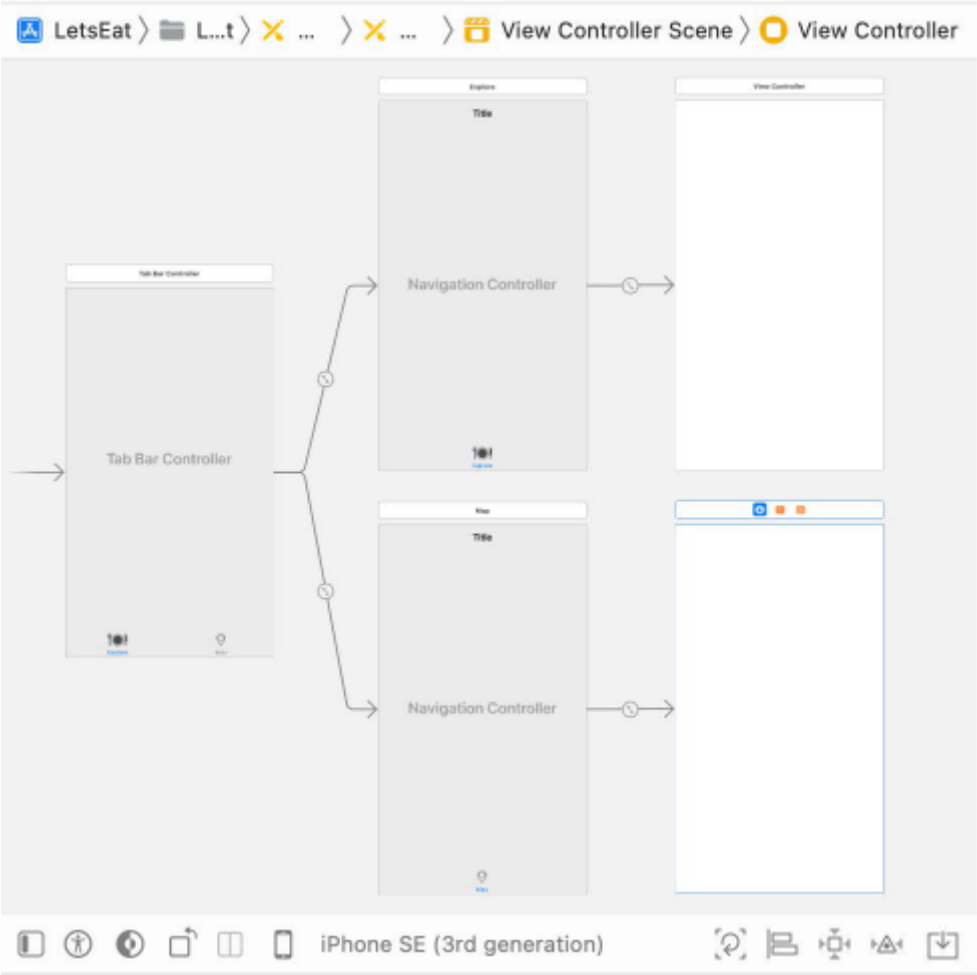
5. Kliknij plik głównego scenorysu w nawigatorze projektu. Skonfiguruj tutaj scenorysy, aby używać również iPhone'a SE (3. generacji):



6. Jeśli chcesz ukryć minimapę, wybierz Edytor | Minimapa z paska menu Xcode, aby ją odznaczyć.



7. Sprawdź, czy w głównym pliku scenorysu znajdują się następujące sceny:



Kompiluj i uruchamiaj swoją aplikację. Powinno działać tak samo jak wcześniej. Utworzyłeś ekrany Eksploruj i Mapa dla swojej aplikacji! Dobrze zrobione!

Streszczenie

W tej części poznałeś kilka przydatnych terminów używanych przy tworzeniu aplikacji na iOS. Ułatwi Ci to zrozumienie pozostałej części tej książki, a także innych książek i zasobów internetowych na ten temat. Następnie dowiedziałeś się również o różnych ekranach używanych w aplikacji Let's Eat i o tym, jak użytkownik będzie korzystał z aplikacji. Odtwarzając interfejs użytkownika aplikacji od zera, będziesz mógł porównać to, co robisz, z tym, jak wygląda rzeczywista aplikacja. Na koniec nauczyłeś się, jak używać Konstruktora interfejsów i scenorysów, aby dodać scenę kontrolera paska kart do swojej aplikacji, skonfigurować tytuły przycisków i skonfigurować pasek nawigacyjny dla ekranów Eksploruj i Mapa. Dodano także plik Assets.xcassets zawierający wszystkie pliki graficzne wymagane dla projektu, skonfigurowano niestandardowe ikony przycisków na pasku kart oraz skonfigurowano ekran uruchamiania aplikacji z niestandardowym kolorem i ikoną. Dzięki temu zapoznasz się z dodawaniem elementów interfejsu użytkownika, konfigurowaniem ich i ustawianiem ograniczeń dla własnych aplikacji. W następnym rozdziale będziesz kontynuować konfigurowanie interfejsu użytkownika aplikacji i zapoznawać się z większą liczbą elementów interfejsu użytkownika. Skonfigurujesz ekran Eksploruj tak, aby wyświetlał widok kolekcji zawierający komórki widoku kolekcji oraz nagłówek sekcji widoku kolekcji zawierający przycisk wyświetlający inny widok po dotknięciu.

Budowanie interfejsu użytkownika

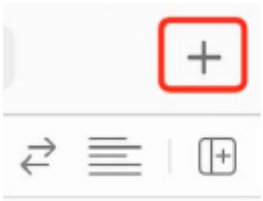
Utworzyłeś nowy projekt Xcode, dodałeś pasek kart do swojej aplikacji, który umożliwiał użytkownikowi wybór pomiędzy ekranami Eksploracja i Mapa, dodałeś plik Assets.xcassets zawierający zasoby dla Twojej aplikacji i zmodyfikowałeś ekran uruchamiania aplikacji za pomocą niestandardowego koloru i ikony. Po uruchomieniu aplikacji powinien zostać przez chwilę wyświetlony ekran uruchamiania. Następnie zostanie wyświetlony ekran Eksploruj, ale obecnie jest pusty. Jak widzieliśmy podczas prezentacji aplikacji , ekran Eksploruj powinien wyświetlać widok kolekcji przedstawiający listę kuchni w komórkach widoku kolekcji oraz nagłówek sekcji widoku kolekcji zawierający przycisk LOKALIZACJA. Naciśnięcie przycisku LOKALIZACJA powinno wyświetlić ekran Lokalizacje zawierający listę lokalizacji. Tu sprawisz, że ekran Eksploracji będzie wyświetlał widok kolekcji zawierający 20 pustych komórek widoku kolekcji, a także nagłówek sekcji widoku kolekcji zawierający przycisk, który po dotknięciu wyświetli widok reprezentujący ekran Lokalizacje. Skonfigurujesz także przycisk Anuluj, aby odrzucić ten widok i powrócić do ekranu Eksploruj. Będziesz dodawał niewielką ilość kodu do swojej aplikacji, ale nie przejmuj się tym zbytnio . Pod koniecdowiesz się, jak dodawać kontrolery widoku do sceny scenorysu, łączyć wyjścia w kontrolerach widoku ze scenami, konfigurować komórki widoku kolekcji i nagłówki sekcji widoku kolekcji oraz modalnie prezentować kontroler widoku.

Dodanie widoku kolekcji do ekranu Eksploruj

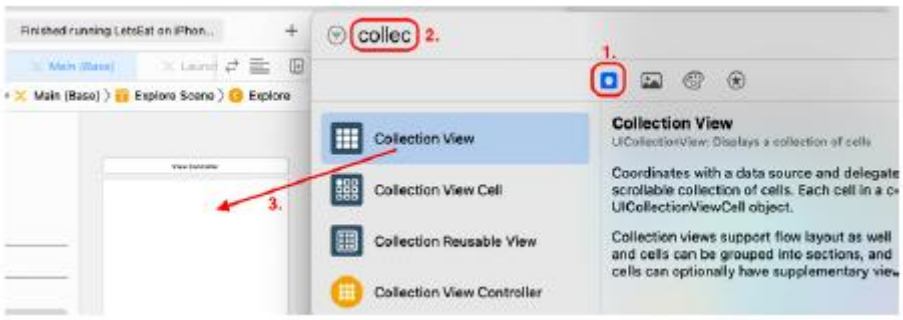
Widok kolekcji jest instancją klasy UICollectionView. Podobnie jak program arkusza kalkulacyjnego, wyświetla siatkę komórek. Każda komórka w widoku kolekcji jest komórką widoku kolekcji, która jest instancją klasy UICollectionViewCell. Zaczniesz od dodania widoku kolekcji do sceny kontrolera widoku dla ekranu Eksploruj w głównym pliku scenorysu, a następnie dodasz ograniczenia automatycznego układu, aby wypełnił ekran.

Otwórz projekt LetsEat utworzony w poprzednim rozdziale i uruchom aplikację, aby upewnić się, że wszystko nadal działa tak, jak powinno, a następnie wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu w nawigatorze projektu i kliknij przycisk Library:

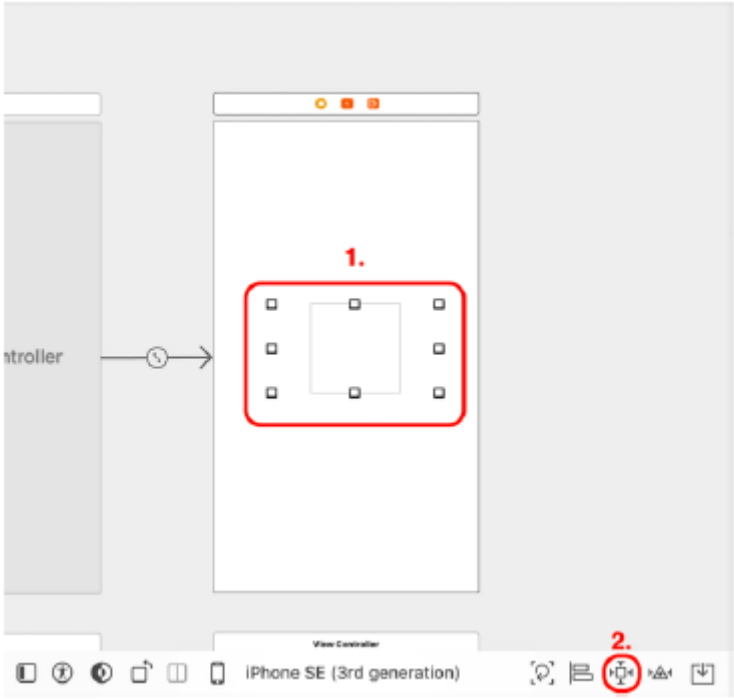


2. Pojawi się biblioteka. Upewnij się, że przycisk Obiekty jest zaznaczony, a następnie wpisz collec w polu filtru. Obiekt widoku kolekcji pojawi się jako jeden z wyników. Przeciągnij go na środek widoku sceny kontrolera widoku na ekranie Eksploruj:

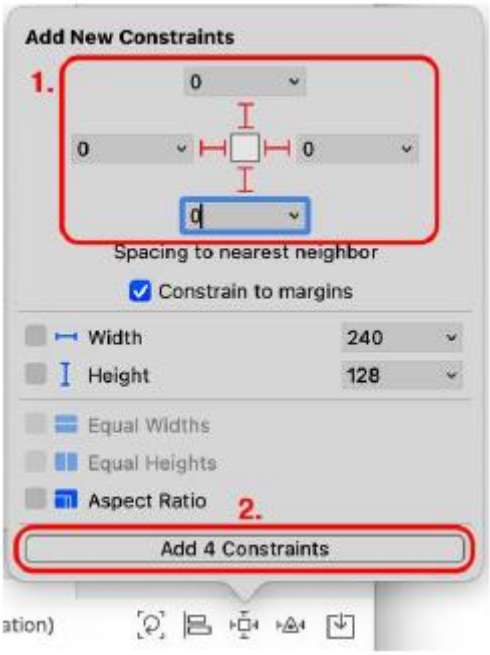


Dodano widok kolekcji (zawierający pojedynczą komórkę prototypową), ale zajmuje on tylko niewielką część ekranu. Jak pokazano w prezentacji aplikacji wcześniej, powinna ona wypełnić ekran.

3. Użyjesz przycisku Auto Layout Dodaj nowe wiązania, aby powiązać krawędzie widoku kolekcji z krawędziami otaczającego go widoku. Upewnij się, że wybrany jest widok kolekcji. Kliknij przycisk Układ automatyczny, dodaj nowe wiązania:

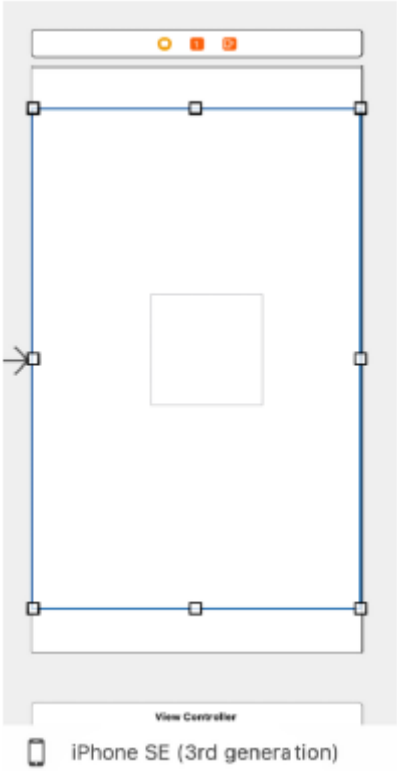


4. Wpisz 0 w polach ograniczających górną, lewą, prawą i dolną krawędź i kliknij wszystkie jasnoczerwone rozpórki. Upewnij się, że wszystkie rozpórki zmieniły kolor na jaskrawoczerwony. Kliknij przycisk Dodaj 4 ograniczenia:



Ustawia to odstęp między krawędziami widoku kolekcji a krawędziami widoku otaczającego na 0, wiążąc krawędzie widoku kolekcji z krawędziami widoku otaczającego. Teraz widok kolekcji wypełni ekran, niezależnie od urządzenia i orientacji.

5. Sprawdź, czy wszystkie cztery boki widoku kolekcji są teraz powiązane z krawędziami ekranu, jak pokazano:



Do widoku sceny kontrolera widoku na ekranie Eksploruj dodano widok kolekcji i użyto ograniczeń układu automatycznego, aby wypełnić ekran, ale ekran Eksploruj będzie nadal pusty po uruchomieniu aplikacji. W następnej sekcji dodasz do swojego projektu plik Cocoa Touch Class, dzięki czemu będziesz

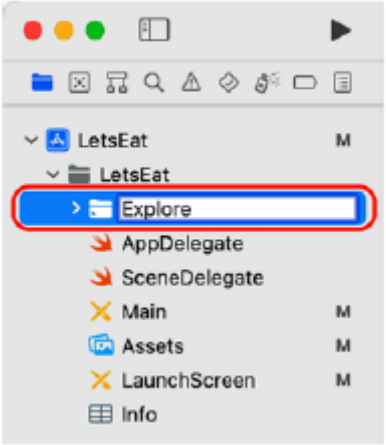
mógł zaimplementować kod dla klasy ExploreViewController, a także połączysz wyjścia w tej klasie z elementami interfejsu użytkownika na ekranie Eksploruj. Dzięki temu instancja klasy ExploreViewController będzie mogła kontrolować to, co jest wyświetlane na ekranie Eksploruj.

Łączenie elementów scenorysu z gniazdami w kontrolerze widoku

Dodałeś widok kolekcji do ekranu Eksploruj, ale nie będzie on jeszcze w stanie niczego wyświetlić. Aby zarządzać widokiem na ekranie Eksploruj, musisz zaimplementować kontroler widoku. Aby to zrobić, dodasz do projektu plik Cocoa Touch Class, zadeklarujesz i zdefiniujesz podklasę UIViewController w tym pliku, a następnie połączysz elementy interfejsu użytkownika na ekranie Eksploruj z kodem w podklasie UIViewController. Zacznijmy od dodania pliku Cocoa Touch Class do Twojego projektu, abyś mógł zadeklarować i zdefiniować podklasę UIViewController w następnej sekcji. Dodawanie pliku klasy Cocoa Touch do projektu Cocoa Touch to środowisko programistyczne umożliwiające tworzenie aplikacji dla systemów iOS, iPadOS, watchOS i tvOS. Plik klasy Cocoa Touch ułatwia wdrożenie dowolnej klasy lub podklasy Cocoa Touch. Zawiera standardowy kod oparty na nadklasie określonej podczas jej tworzenia. W tej sekcji dodasz plik zajęć Cocoa Touch do swojego projektu. Najpierw utworzysz w swoim projekcie nową grupę Eksploruj, aby zachować porządek. Następnie utworzysz i dodasz do tej grupy plik klasy Cocoa Touch o nazwie ExploreViewController. W tym pliku zadeklarujesz i zdefiniujesz podklasę klasy UIViewController o nazwie ExploreViewController i uczynisz instancję tej klasy kontrolerem widoku dla ekranu Eksploruj. Dodasz do tej klasy właściwości i metody, aby zarządzać widokiem kolekcji dodanym w poprzedniej sekcji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy grupę LetsEat w nawigatorze projektu i wybierz opcję Nowa grupa.

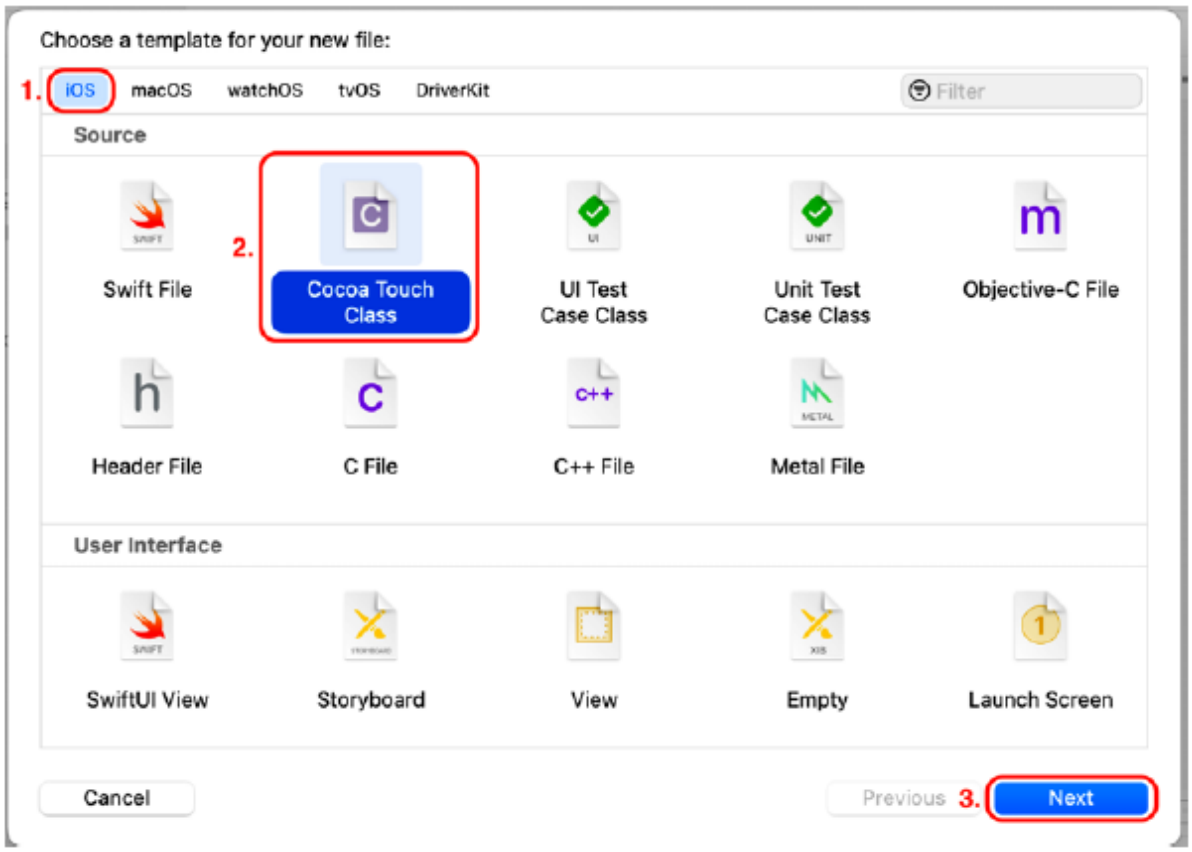
2. Nazwa grupy zostanie podświetlona. Zmień go na Eksploruj i gdy skończysz, naciśnij Return na klawiaturze:



Jeśli się pomylisz, naciśnij jeszcze raz Return. Dzięki temu pole będzie edytowalne, co umożliwi wprowadzanie zmian w nazwie.

3. Kliknij prawym przyciskiem myszy grupę Eksploruj w nawigatorze projektu i wybierz Nowy plik... .

4. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej:



5. Pojawi się ekran Wybierz opcje nowego pliku:

6. W polach Klasa i Podklasa wpisz:

Class: ExploreViewController

Subclass of: UIViewController

Kliknij Next.

7. Kliknij Utwórz, aby dodać plik do projektu. Zobaczysz, że plik ExploreViewController został dodany do projektu w folderze Explore w nawigatorze projektu. Przejrzyj kod w obszarze Edytor. Należy zauważyć, że ExploreViewController jest podklasą IViewController, co oznacza, że dziedziczy właściwości i metody z klasy UIViewController. W definicji klasy znajduje się jedna metoda, viewDidLoad(), ale nie będzie ona teraz używana.

8. Usuń skomentowany kod po klasie viewDidLoad() z pliku ExploreViewController, tak aby wyglądał następująco:

```
LetsEat > LetsEat > Explore > ExploreViewController > ExploreViewController
3  //  LetsEat
4  //
5  //  Created by iOS 16 Programming for Beginners
       on 14/06/2022.
6  //
7
8  import UIKit
9
10 class ExploreViewController: UIViewController {
11
12     override func viewDidLoad() {
13         super.viewDidLoad()
14
15         // Do any additional setup after loading
               the view.
16     }
17
18 }
19
```

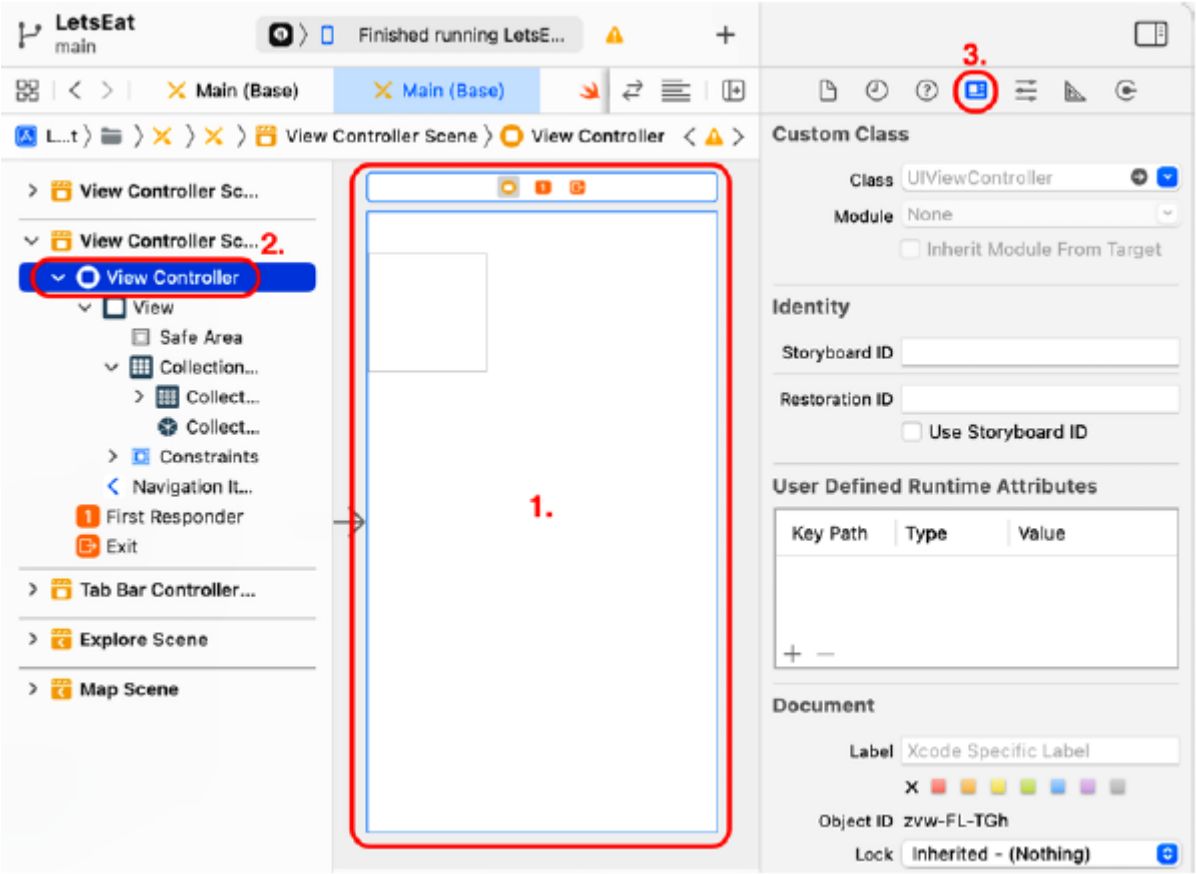
Właśnie dodałeś do swojej aplikacji plik ExploreViewController zawierający deklarację i definicję klasy ExploreViewController. Następnym krokiem jest przypisanie klasy ExploreViewController jako tożsamości kontrolera widoku dla ekranu Eksploruj i przypisanie punktu wyjścia dla widoku kolekcji, który został wcześniej dodany do sceny kontrolera widoku. Zobaczysz, jak to się robi w następnej sekcji.

Podłączanie elementów scenorysu do kontrolera widoku

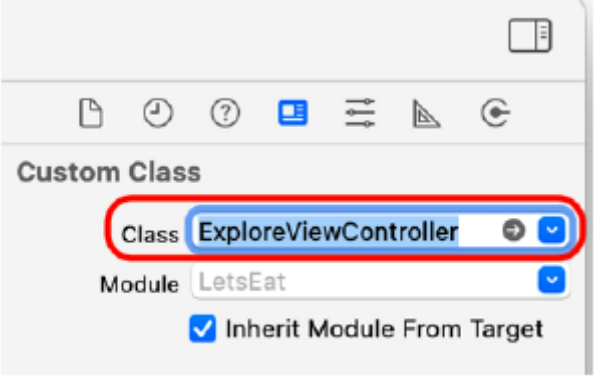
Sprawdźmy, gdzie teraz jesteś. W głównym pliku scenorysu znajduje się scena kontrolera widoku dla ekranu Eksploruj. Wewnątrz znajduje się widok zawierający widok kolekcji. W pliku ExploreViewController znajduje się kod, który deklaruje i definiuje klasę ExploreViewController. Musisz przypisać klasę ExploreViewController jako tożsamość kontrolera widoku ekranu Eksploruj i podłączyć widok kolekcji do gniazdka w tej klasie. Punkty sprzedaży można traktować jako połączenie między interfejsem użytkownika a kodem. Spowoduje to, że instancja klasy ExploreViewController stanie się kontrolerem widoku dla ekranu Eksploruj po uruchomieniu aplikacji i umożliwi zarządzanie tym, co jest wyświetlane w widoku kolekcji.

Wykonaj następujące kroki:

1. Kliknij plik Main scenorysu w nawigatorze projektu. Upewnij się, że wybrano opcję Wyświetl scenę kontrolera na ekranie Eksploruj. Kliknij ikonę Kontrolera widoku w konspekcie dokumentu i kliknij przycisk Inspektora tożsamości:

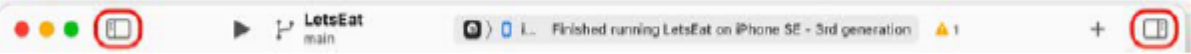


2. W klasie niestandardowej ustaw klasę na ExploreViewController:

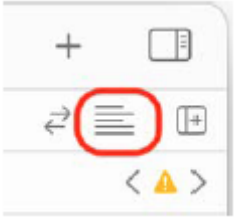


Spowoduje to utworzenie instancji ExploreViewController jako kontrolera widoku dla tej sceny po uruchomieniu aplikacji. Należy zauważyć, że nazwa sceny została zmieniona z View Controller Scene na Explore View Controller Scene. Stwórzmy teraz wylot dla widoku kolekcji.

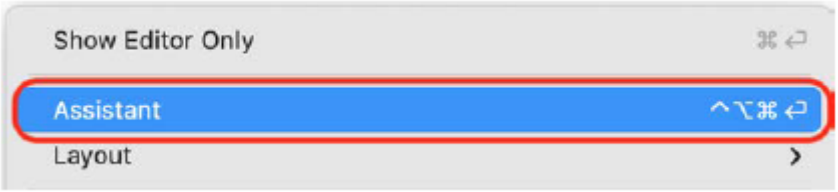
3. Kliknij przyciski Nawigator i Inspektor, aby ukryć obszary Nawigatora i Inspektora, zapewniając więcej miejsca do pracy:



4. Kliknij przycisk Dostosuj opcje edytora:



5. Z wyskakującego menu wybierz opcję Asystent:



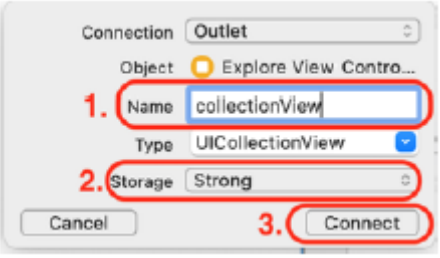
Spowoduje to wyświetlenie wszystkich plików Swift powiązanych z tą sceną w asystentze edytora.

6. Jak widać, zawartość głównego pliku scenorysu pojawia się po lewej stronie, a definicja klasy ExploreViewController pojawia się po prawej stronie obszaru Edytor. Spójrz na pasek tuż nad kodem. Sprawdź, czy wybrany jest plik ExploreViewController.swift:

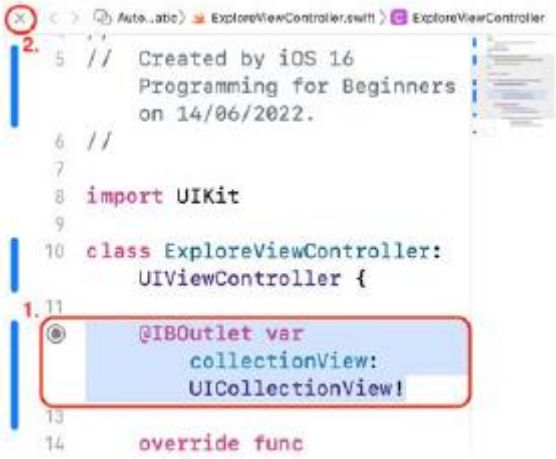
7. Jeśli nie widzisz zaznaczonego pliku ExploreViewController.swift, kliknij pasek i wybierz ExploreViewController.swift z wyskakującego menu.

8. Aby połączyć widok kolekcji w scenie Explore View Controller z gniazdem w klasie ExploreViewController, naciśnij klawisz Ctrl i przeciągnij z widoku kolekcji do pliku ExploreViewController, tuż pod deklaracją nazwy klasy:

9. Pojawi się małe wyskakujące okno dialogowe. Wpisz nazwę punktu sprzedaży, kolekcjaView, w polu tekstowym Nazwa i kliknij Połącz:



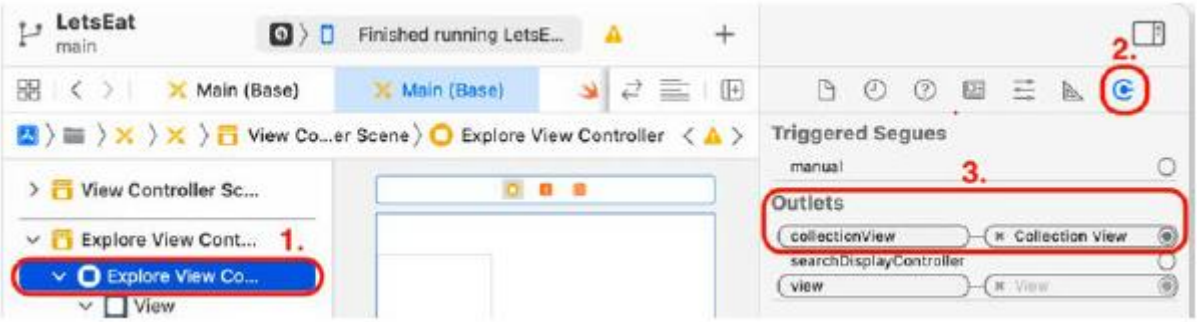
10. Sprawdź, czy kod tworzący outlet kolekcjiView został automatycznie dodany do pliku ExploreViewController. Zwróć uwagę na słowo kluczowe IBOutlet, które wskazuje, że kolekcjaView jest gniazdem. Po wykonaniu tej czynności kliknij x, aby zamknąć okno asystenta edytora:



Klasa ExploreViewController ma teraz punkt wyjściowy, CollectionView, do widoku kolekcji na ekranie Eksploruj. Oznacza to, że instancja ExploreViewController może zarządzać kolekcją . Często popełniane są błędy podczas używania Ctrl + Przeciągnij do przeciągania z elementu w scenie scenorysu do pliku klasy Cocoa Touch. Jeśli popełnisz przy tym błąd, może to spowodować awarię podczas uruchamiania aplikacji. Aby sprawdzić, czy w połączeniu widoku kolekcji z klasą ExploreViewController nie występują błędy, wykonaj następujące kroki:

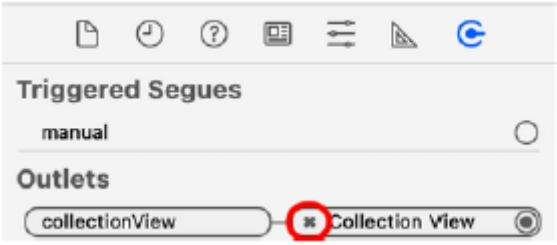
1. Kliknij przyciski Nawigator i Inspektor, aby wyświetlić obszary Nawigator i Inspektor.

2. Po wybraniu opcji Eksploruj kontroler widoku w scenie Eksploruj kontroler widoku kliknij przycisk Inspektora połączeń:

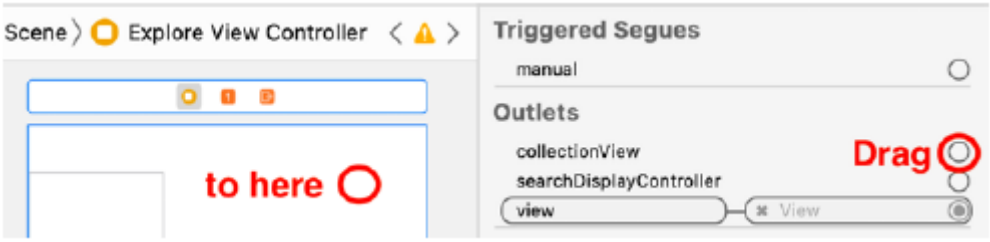


Inspektor połączeń wyświetla łącza między obiektami interfejsu użytkownika a kodem. Powinieneś widzieć outlet kolekcjiView połączony z widokiem kolekcji w sekcji Outlety.

3. Jeśli zobaczysz małą ikonę błędu, kliknij x, aby przerwać połączenie:



4. W obszarze Outlets przeciągnij z outletu CollectionView z powrotem do widoku kolekcji, aby ponownie nawiązać połączenie:



W tym momencie przypisałeś klasę ExploreViewController jako tożsamość kontrolera widoku ekranu Eksploruj i utworzono w klasie ExploreViewController punkt wyjścia dla widoku kolekcji. Aby wyświetlić komórki widoku kolekcji na ekranie, należy zaimplementować metody źródła danych dla widoku kolekcji, dodając kod do klasy ExploreViewController. Zrobisz to w następnej sekcji.

Konfigurowanie metod źródła danych dla widoku kolekcji

Gdy aplikacja jest uruchomiona, instancja klasy ExploreViewController działa jako kontroler widoku na ekranie Eksploruj. Odpowiada za załadowanie i wyświetlenie wszystkich widoków na tym ekranie, łącznie z dodanym wcześniej widokiem kolekcji. Widok kolekcji musi wiedzieć, ile komórek widoku kolekcji ma zostać wyświetlonych i co wyświetlić w każdej komórce. Zwykle za dostarczenie tych informacji odpowiedzialny jest kontroler widoku. W tym celu Apple stworzył już protokół UICollectionViewDataSource. Wszystko, co musisz zrobić, to połączyć wylot dataSource widoku kolekcji z klasą ExploreViewController i zaimplementować wymagane metody tego protokołu. Widok kolekcji musi także wiedzieć, co zrobić, gdy użytkownik dotknie komórki widoku kolekcji. Ponownie odpowiedzialny jest kontroler widoku dla widoku kolekcji i Apple stworzył w tym celu protokół UICollectionViewDelegate. Połączysz punkt delegowania widoku kolekcji z klasą ExploreViewController, ale nie będziesz jeszcze implementować żadnych metod z tego protokołu. W tej części będziesz musiał wpisać niewielką ilość kodu. W następnej sekcji użyjesz Inspektora połączeń, aby przypisać źródło danych widoku kolekcji i delegować wyjścia do klasy ExploreViewController.

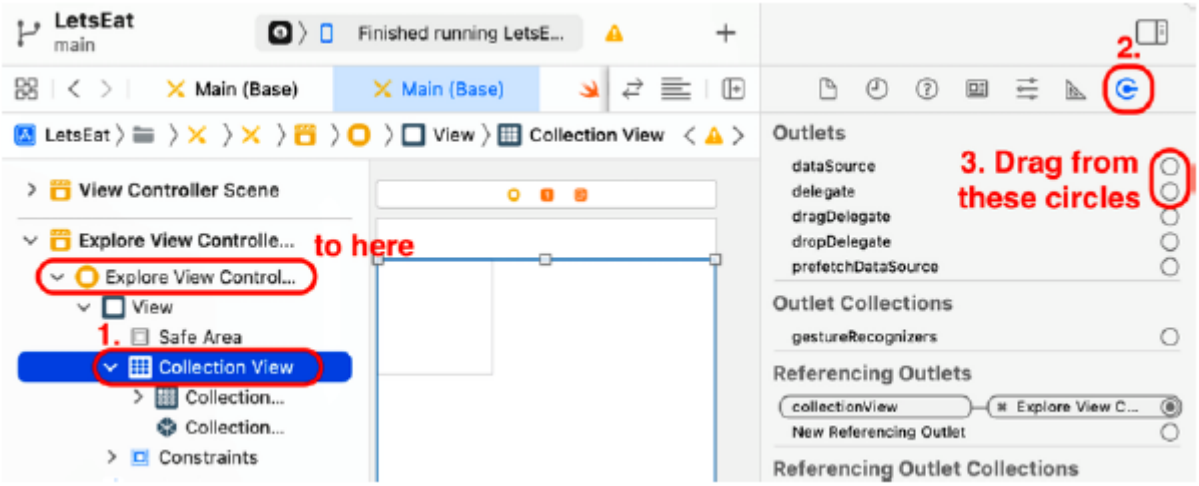
Ustawianie właściwości delegata i źródła danych widoku kolekcji

Instancja klasy ExploreViewController udostępnia dane wyświetlane w widoku kolekcji, a także metody, które zostaną wykonane, gdy użytkownik wejdzie w interakcję z widokiem kolekcji. Aby to zadziałało, musisz połączyć źródło danych widoku kolekcji i delegować właściwości do gniazd w klasie ExploreViewController. Wykonaj następujące kroki:

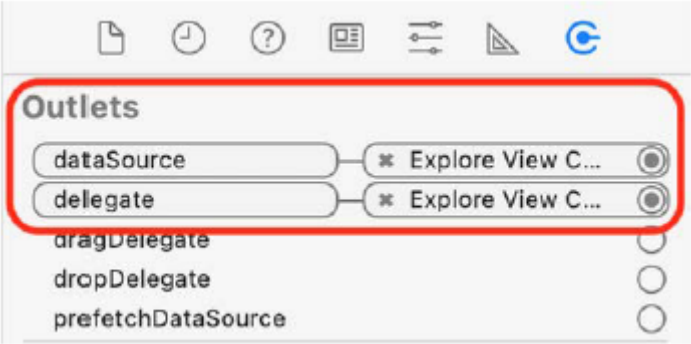
1. Kliknij przyciski Nawigator i Inspektor, aby ponownie wyświetlić obszary Nawigatora i Inspektora, jeśli jeszcze tego nie zrobiłeś.

2. Główny plik scenorysu powinien być nadal wybrany. Kliknij widok kolekcji dla sceny kontrolera widoku Eksploruj w konspekcie dokumentu, aby ją wybrać. Kliknij przycisk Inspektora połączeń.

Spójrz na sekcję Outlety. Zwróć uwagę, że obok źródła danych i gniazd delegatów znajdują się dwa puste kółka. Przeciągnij z każdego pustego okręgu na ikonę ExploreViewController w konspekcie dokumentu:



3. Sprawdź, czy właściwości dataSource i delegowanie widoku kolekcji zostały połączone z gniazdami w klasie ExploreViewController:

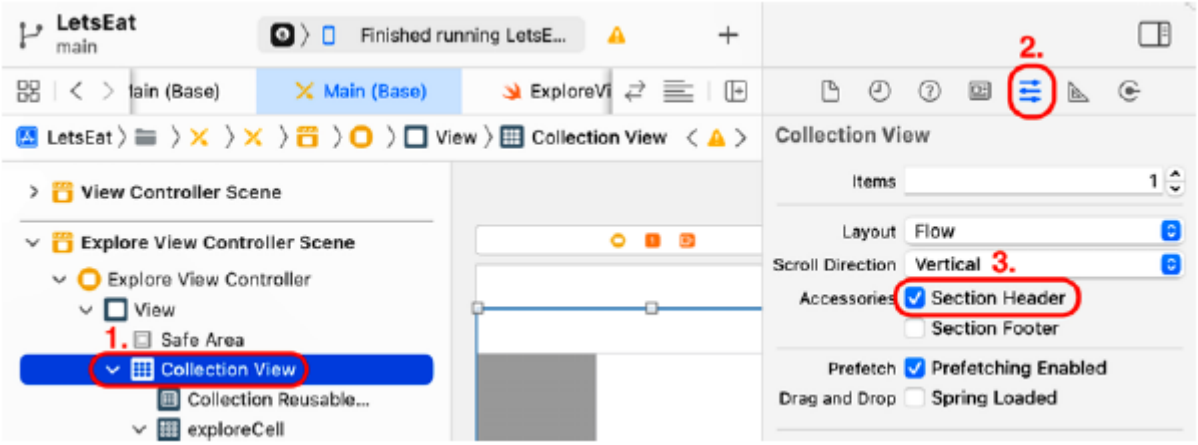


W następnej sekcji dodasz kod, aby klasa ExploreViewController była zgodna z protokołem UICollectionViewDataSource i skonfigurujesz widok kolekcji tak, aby po uruchomieniu aplikacji wyświetlał 20 komórek widoku kolekcji.

Dodanie nagłówka sekcji do widoku kolekcji

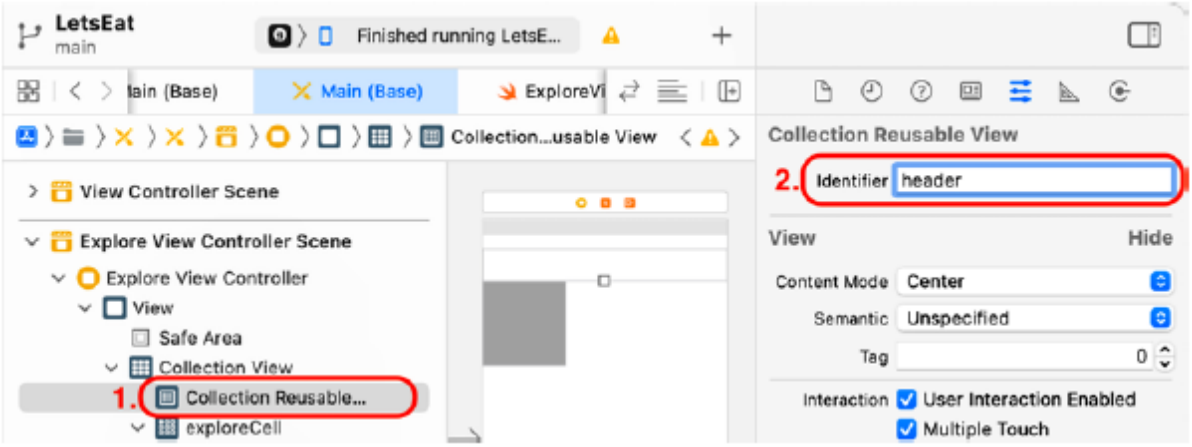
Widok kolekcji można skonfigurować za pomocą nagłówka i stopki sekcji. Obydwa są instancjami klasy UICollectionReusableView. Włączysz nagłówek sekcji dla widoku kolekcji na ekranie Eksploruj, dzięki czemu będziesz mieć miejsce na umieszczenie przycisku LOKALIZACJA. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu w nawigatorze projektu i kliknij Widok kolekcji w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. W obszarze Widok kolekcji zaznacz pole wyboru Nagłówek sekcji:



Włącza to nagłówek sekcji dla widoku kolekcji.

2. Zwróć uwagę, że w konspekcie dokumentu pojawia się widok kolekcji do ponownego wykorzystania. Reprezentuje nagłówek sekcji widoku kolekcji. Kliknij opcję Widok kolekcji do ponownego wykorzystania w konspekcie dokumentu. W Inspektorze atrybutów, w obszarze Widok kolekcji do ponownego użycia, ustaw Identyfikator na nagłówek, a gdy skończysz, naciśnij Return:



3.Kliknij plik ExploreViewController w nawigatorze projektu. Tuż przed metodami źródła danych wpisz następujący kod:

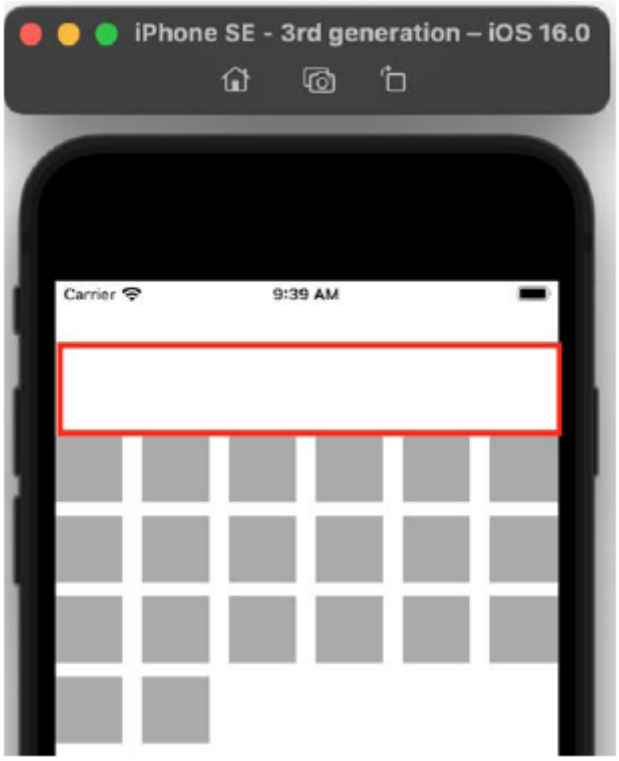
```
func collectionView(_ collectionView: UICollectionView,

viewForSupplementaryElementOfKind kind: String, at indexPath: IndexPath)

-> UICollectionReusableView {

let headerView =

collectionView.dequeueReusableSupplementaryView(

ofKind: kind, withReuseIdentifier: "header",
```

for: indexPath)

return headerView

}

Ta metoda zwraca instancję UICollectionReusableView z właśnie skonfigurowanym nagłówkiem identyfikatora, który zostanie wyświetlony na ekranie.

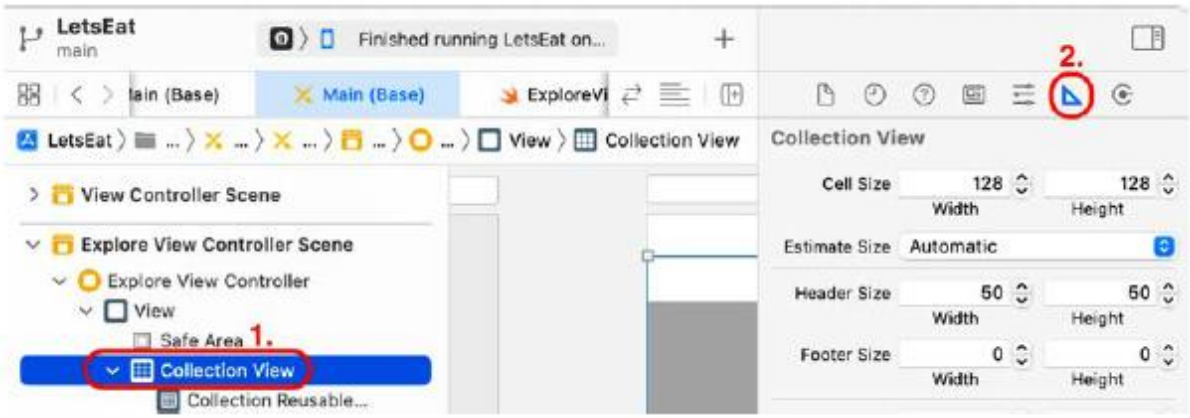


Zanim dodasz przycisk LOKALIZACJA, musisz zwiększyć wysokość nagłówka sekcji widoku kolekcji i rozmiar komórki widoku kolekcji, aby dopasować je do ekranu Eksploruj pokazanego w przewodniku po aplikacji. Rozmiar komórki i wysokość nagłówka ustawisz za pomocą Inspektor rozmiaru w następnej sekcji.

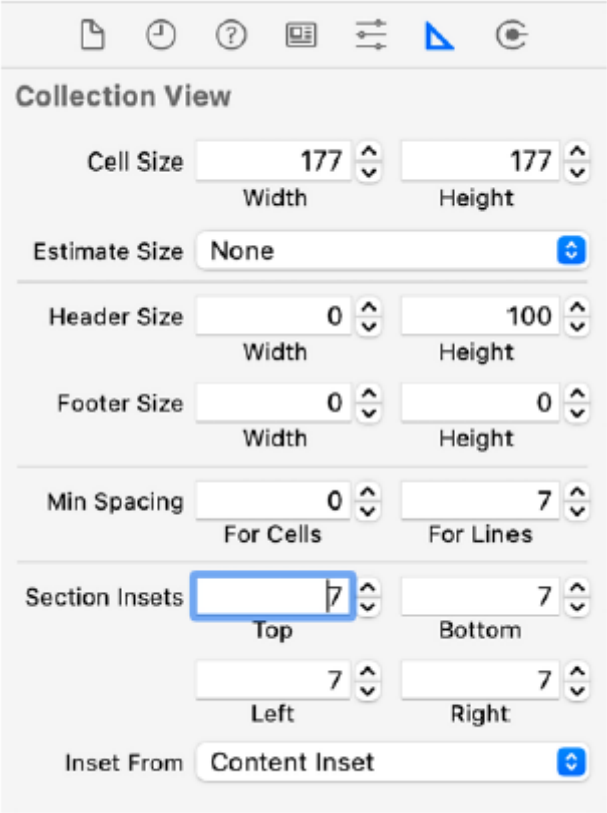
Konfigurowanie rozmiarów elementów scenorysu

Inspektor rozmiaru służy do zmiany rozmiaru elementów scenorysu. Użyjesz go do zmiany rozmiaru komórki widoku kolekcji i nagłówka sekcji widoku kolekcji, aby dopasować je do ekranu Eksploruj, który został pokazany podczas prezentacji aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu w nawigatorze projektu i kliknij Widok kolekcji w konspekcie dokumentu. Kliknij przycisk Inspektora rozmiaru:



2. Ustawienia rozmiaru widoku kolekcji zostaną wyświetlone w Inspektorze rozmiaru, jak pokazano poniżej:



3. Skonfiguruj ustawienia rozmiaru widoku kolekcji w następujący sposób:

Rozmiar komórki: Ustaw szerokość na 177, a wysokość na 177.

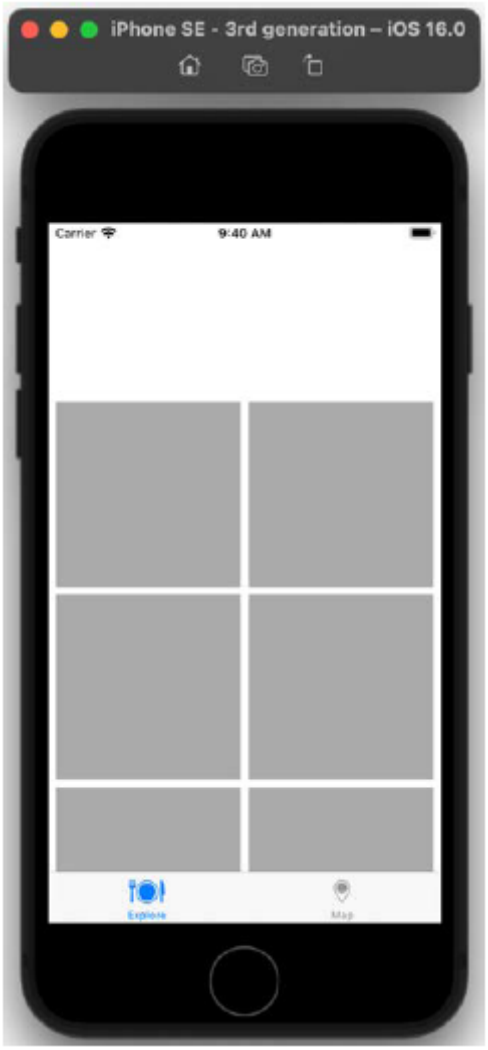
Szacowany rozmiar: brak.

Rozmiar nagłówka: Ustaw szerokość na 0 i wysokość na 100.

Minimalny odstęp: Ustaw dla komórek na 0 i dla linii na 7.

Wstawki sekcji: Ustaw Górę, Dół, Lewo i Prawo na 7.

Pamiętaj, aby po zmianie każdej wartości nacisnąć klawisz Return. Jednostką używaną w Inspektorze rozmiaru są punkty. Każdy punkt może odnosić się do jednego lub większej liczby pikseli na ekranie urządzenia. W przypadku iPhone'a SE (3. generacji) ekran ma 375 punktów szerokości i 667 punktów wysokości, chociaż faktyczna rozdzielczość ekranu wynosi 750 x 1334 pikseli. Rozmiar komórki określa rozmiar komórki widoku kolekcji. Rozmiar nagłówka określa rozmiar nagłówka sekcji widoku kolekcji. Minimalny odstęp określa odstęp między komórkami. Odsunięcia sekcji określają odstęp pomiędzy sekcją zawierającą komórki po bokach otaczającego widoku. Te ustawienia są specyficzne dla iPhone'a SE (3. generacji). W rozdziale 23, Pierwsze kroki z Mac Catalyst, obliczysz optymalny rozmiar komórki na podstawie wymiarów ekranu urządzenia. Zbuduj i uruchom aplikację. Na ekranie Eksploruj powinieneś zobaczyć 20 komórek widoku kolekcji i nagłówek sekcji widoku kolekcji:



Zwróć uwagę, że chociaż w komórkach nie ma żadnych danych, a nagłówek nie ma przycisku, wygląda on podobnie do ekranu Eksploruj, który został pokazany podczas prezentacji aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika. W dalszej części tej książki skonfigurujesz komórki tak, aby wyświetlały dane. Na razie dodajmy przycisk do nagłówka sekcji widoku kolekcji, który później posłuży do wyświetlenia ekranu Lokalizacje.

Prezentowanie widoku modalnie

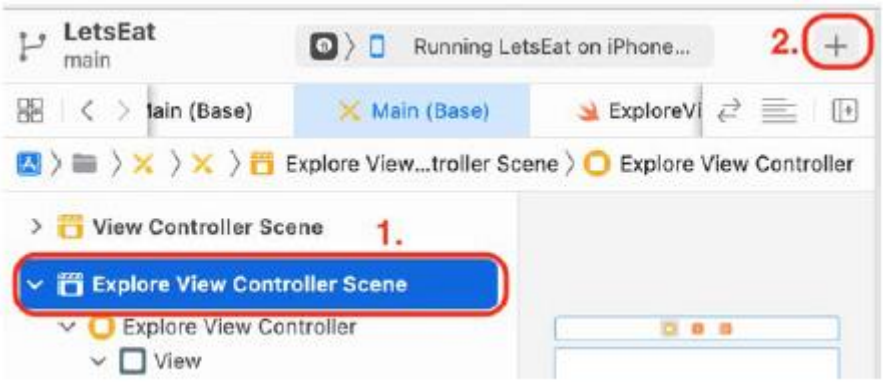
W tej sekcji dodasz przycisk do nagłówka sekcji widoku kolekcji. Po dotknięciu tego przycisku wyświetli się widok przedstawiający ekran Lokalizacji. Ten widok będzie pochodził z nowej sceny kontrolera

widoku osadzonej w kontrolerze nawigacji, którą dodasz do projektu. Widok zostanie zaprezentowany modalnie, co oznacza, że dopóki nie zostanie odrzucony, nie będziesz mógł zrobić nic więcej. Aby go odrzucić, do paska nawigacyjnego widoku dodasz przycisk Anuluj. Dodasz także przycisk Gotowe, ale jego funkcjonalność zaimplementujesz dopiero później. Zacznijmy od dodania przycisku z biblioteki do nagłówka sekcji widoku kolekcji.

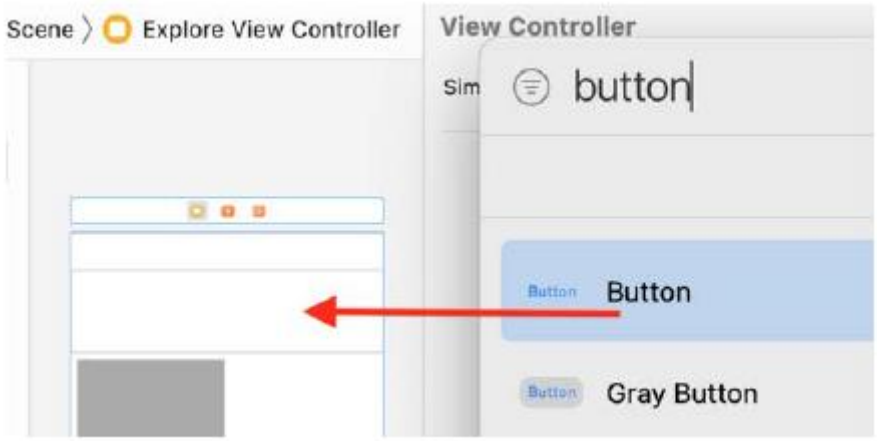
Dodanie przycisku do nagłówka widoku kolekcji

Jak pokazano w przewodniku po aplikacji, w prawym górnym rogu ekranu znajduje się przycisk LOKALIZACJA. Dodasz przycisk reprezentujący przycisk LOKALIZACJA w nagłówku sekcji widoku kolekcji. Wykonaj następujące kroki:

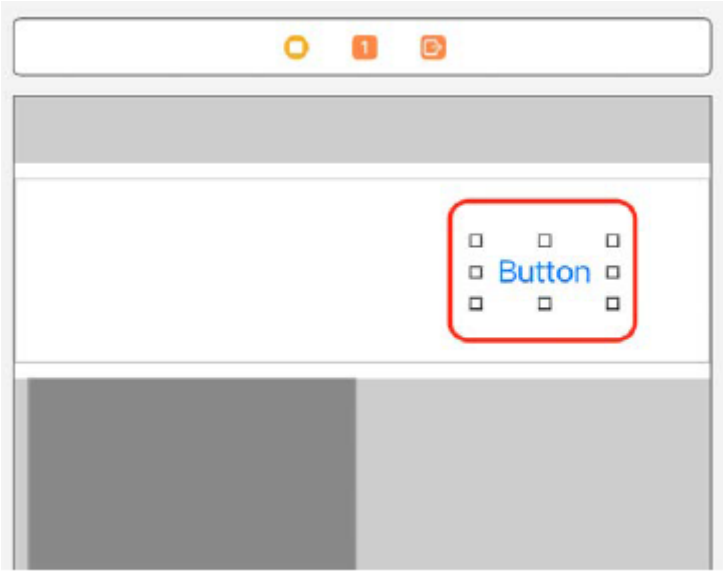
1. Kliknij plik głównego scenorysu w nawigatorze projektu. Upewnij się, że wybrano opcję Eksploruj scenę kontrolera widoku. Kliknij przycisk Biblioteka, aby wyświetlić bibliotekę:



2. Wpisz przycisk w polu filtra. W wynikach pojawi się obiekt Button. Przeciągnij przycisk do nagłówka sekcji widoku kolekcji:



3. Umieść przycisk po prawej stronie nagłówka sekcji widoku kolekcji:



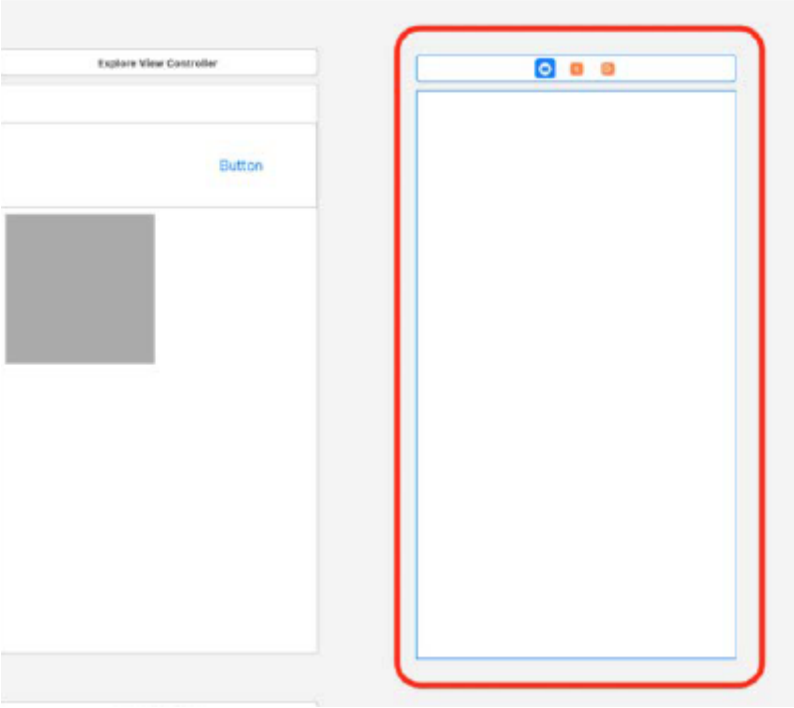
Masz teraz przycisk w nagłówku sekcji widoku kolekcji. Następnie dodasz scenę kontrolera widoku, która będzie reprezentować ekran Lokalizacje, który pojawi się po dotknięciu przycisku.

Dodanie nowej sceny kontrolera widoku

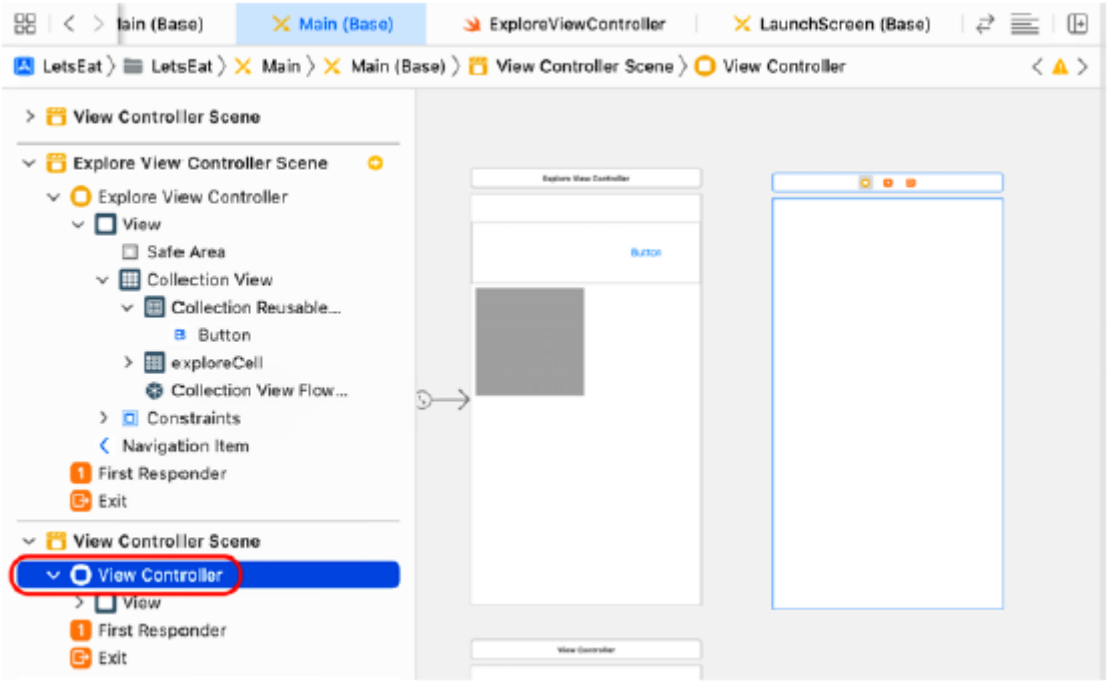
Jak pokazano w przewodniku po aplikacji, po dotknięciu przycisku LOKALIZACJA na ekranie Lokalizacje pojawi się lista lokalizacji. Dodasz do projektu nową scenę kontrolera widoku, która będzie reprezentować ten ekran. Wykonaj następujące kroki:

1. Kliknij przycisk Biblioteka, aby wyświetlić bibliotekę i wpisz view con w polu filtru. Obiekt View Controller będzie wśród wyników wyszukiwania. Przeciągnij obiekt View Controller na scenorys:

2. Ustaw go po prawej stronie sceny kontrolera widoku Eksploruj:

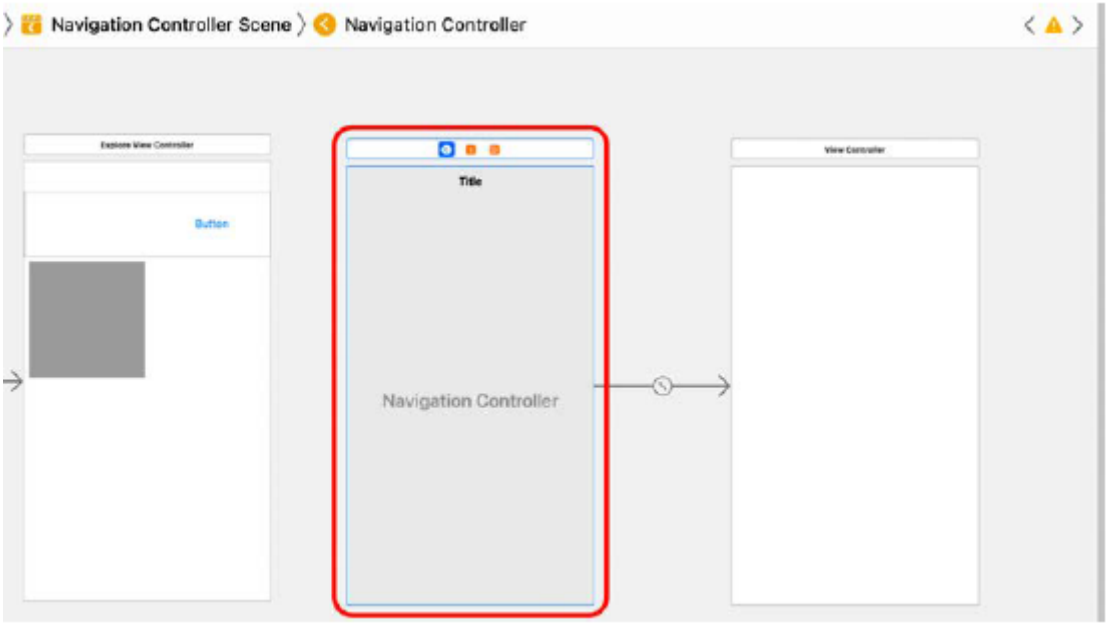


3. Nowo dodana scena kontrolera widoku powinna być już wybrana. W konspekcie dokumentu kliknij ikonę kontrolera widoku dla tej sceny:

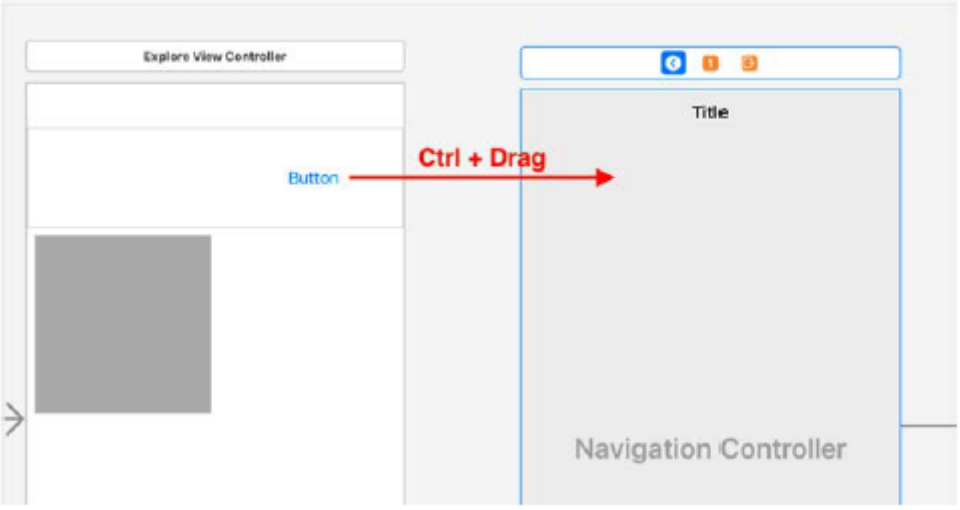


4. Będziesz potrzebować miejsca na przyciski Anuluj i Gotowe, dlatego osadzisz tę scenę kontrolera widoku w kontrolerze nawigacji, aby zapewnić pasek nawigacji, na którym można umieścić przyciski. Wybierz opcję Osadź w | Kontroler nawigacji z menu Edytor.

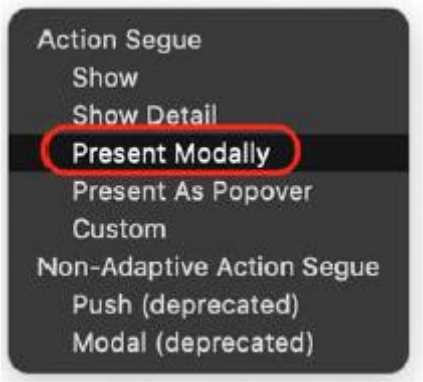
5. Po lewej stronie sceny kontrolera widoku pojawi się scena kontrolera nawigacji:



6. Ctrl + Przeciągnij z przycisku na scenę kontrolera nawigacji:

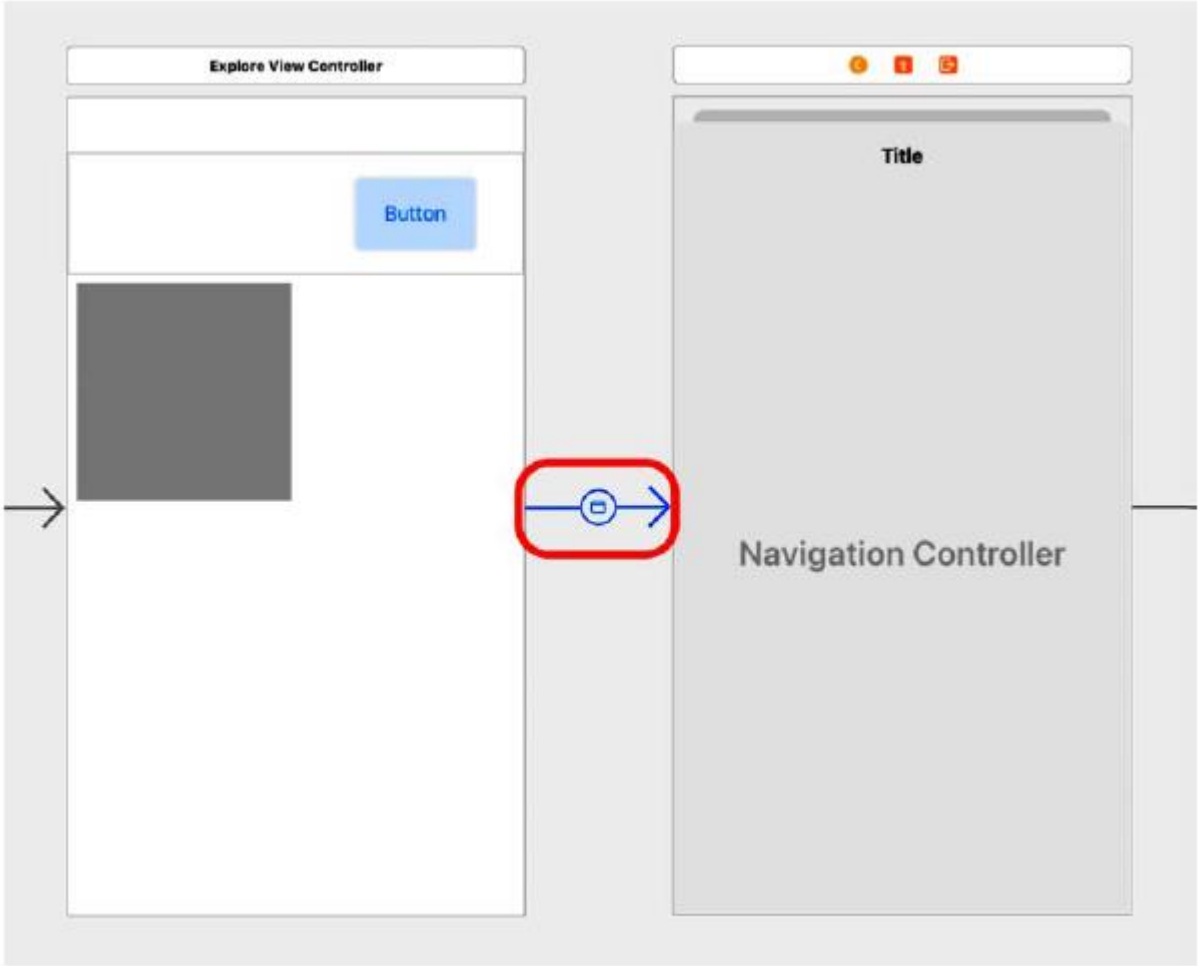


7. Pojawi się wyskakujące menu Przejdź. Wybierz opcję Prezentuj modalnie:

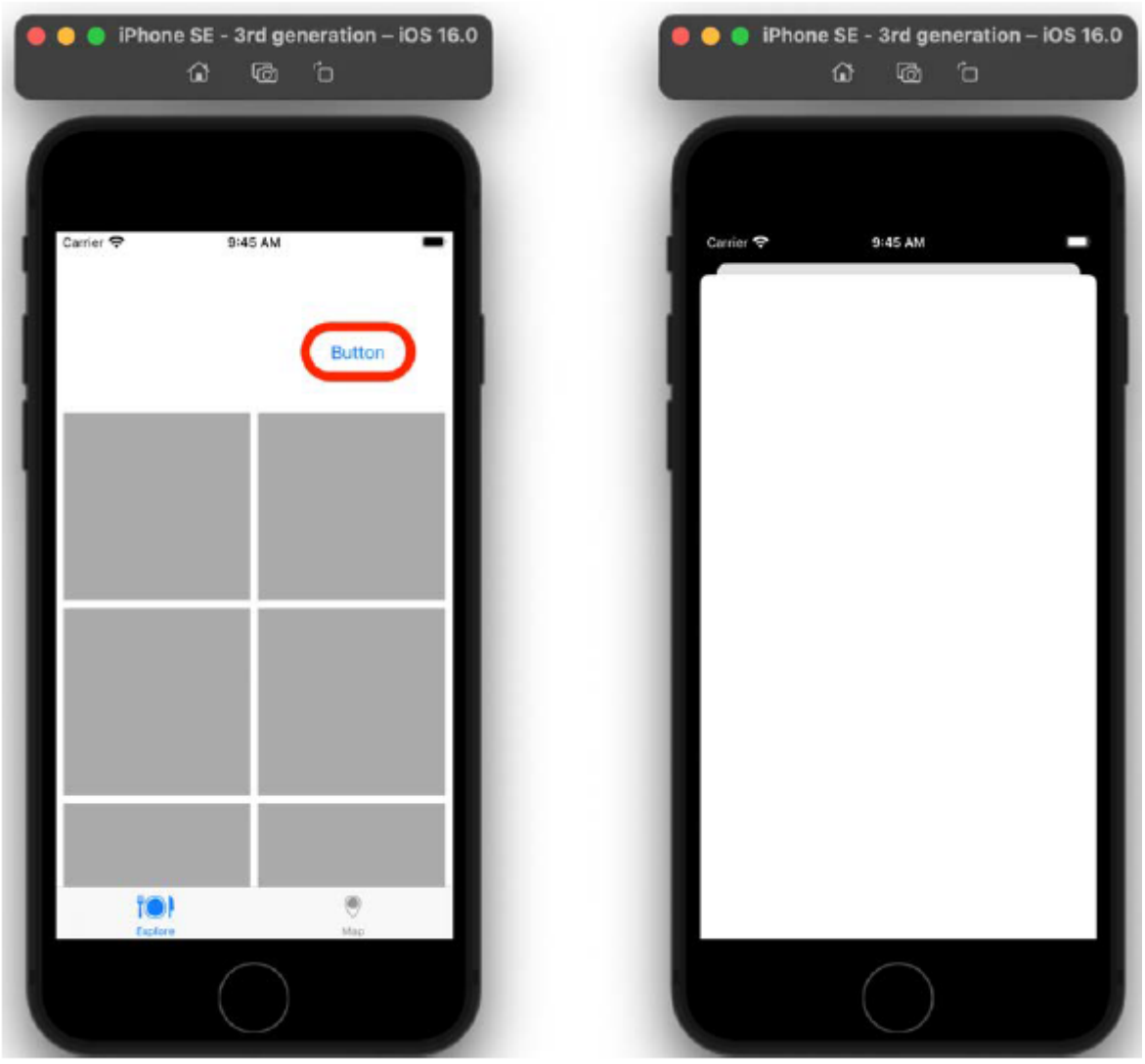


To sprawia, że widok kontrolera widoku przesuwa się w górę od dołu ekranu po dotknięciu przycisku. Nie będziesz mógł wchodzić w interakcję z żadnym innym widokiem, dopóki ten widok nie zostanie odrzucony.

8. Sprawdź, czy sekwencja połączyła razem scenę Eksploruj widok kontrolera i scenę kontrolera nawigacji:



Kompiluj i uruchamiaj swoją aplikację. Jeśli klikniesz przycisk, widok nowego kontrolera widoku powinien przesunąć się w górę od dołu ekranu:



W tej chwili nie można odrzucić tego poglądu. W następnej sekcji dodasz przycisk Anuluj do paska nawigacyjnego i zaprogramujesz go tak, aby odrzucał widok. Dodasz także przycisk Gotowe, ale nie będziesz go jeszcze programować.

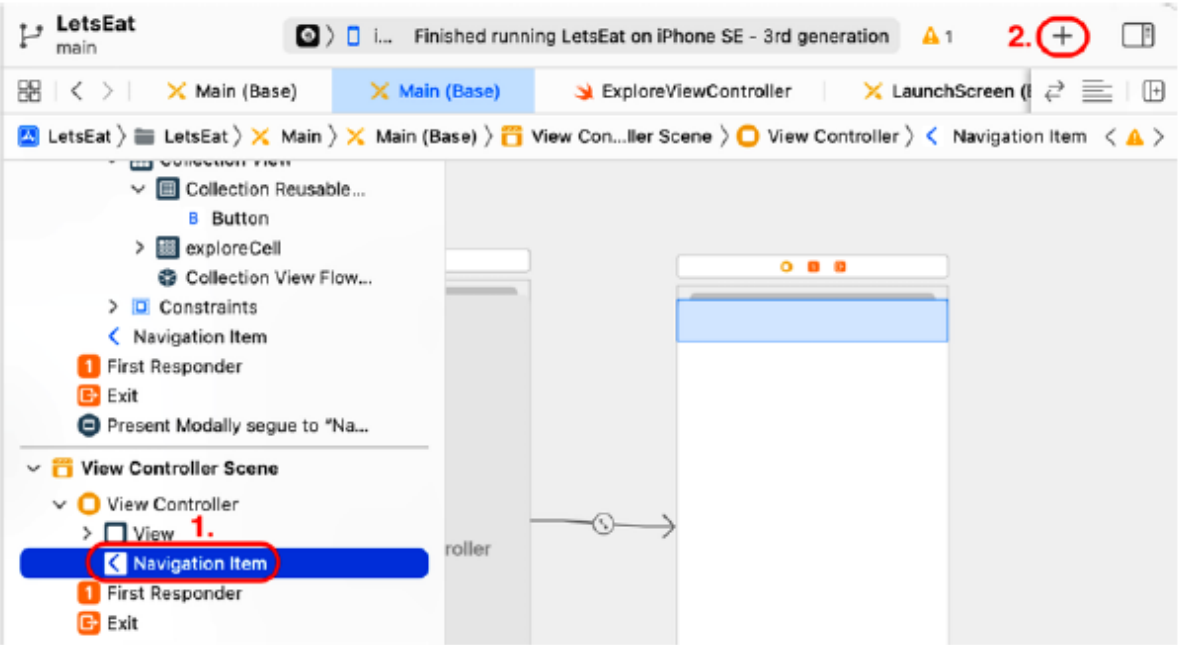
str. 30

Dodanie przycisków Anuluj i Gotowe do paska nawigacyjnego

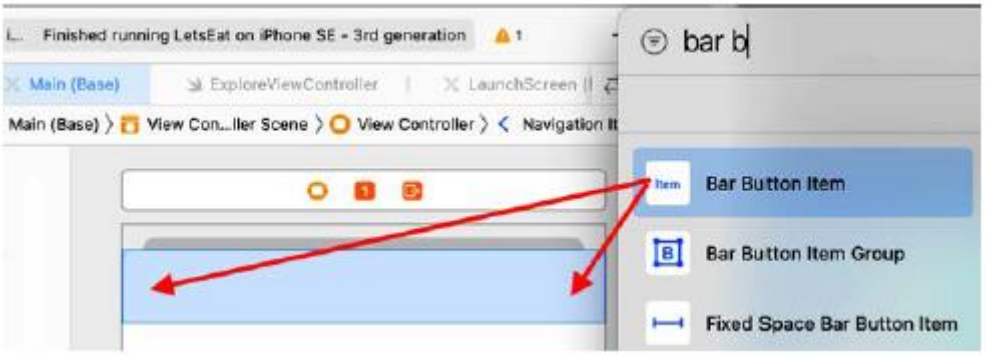
Jedną z zalet osadzania kontrolera widoku w kontrolerze nawigacji jest pasek nawigacji u góry ekranu. Przyciski można umieścić po jego lewej i prawej stronie. Wykonaj poniższe kroki, aby dodać

przyciski Anuluj i Gotowe na pasku nawigacyjnym:

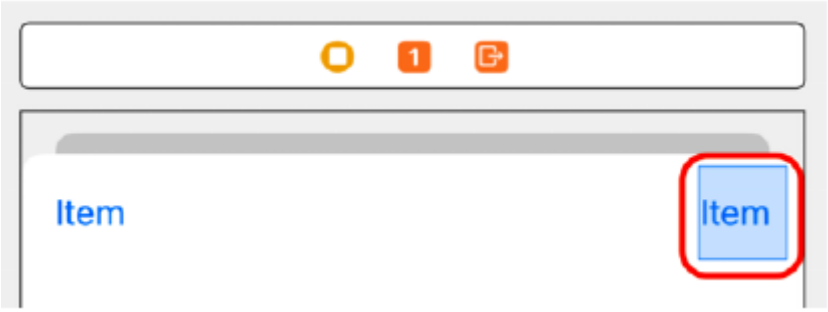
1. Kliknij element nawigacyjny dla sceny kontrolera widoku w konspekcie dokumentu. Kliknij przycisk Biblioteka:



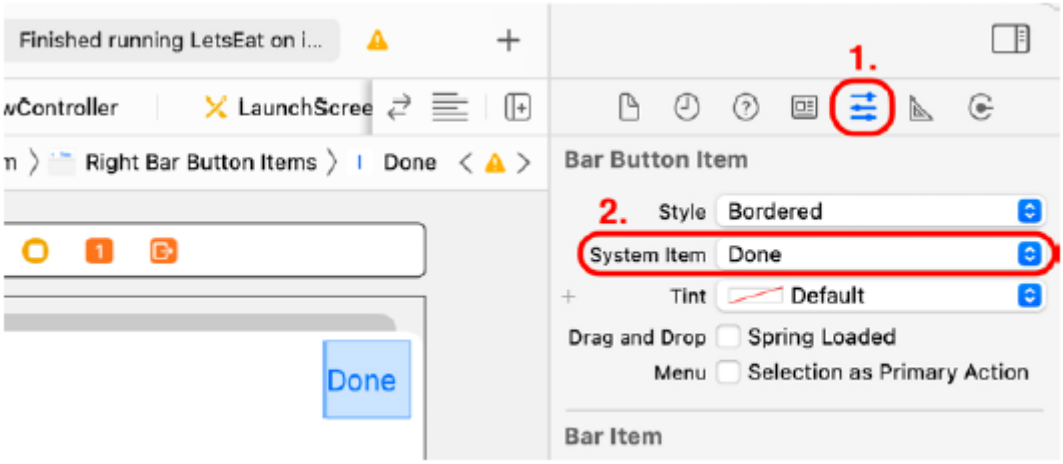
2. Wpisz bar b w polu filtru i przeciągnij dwa obiekty Bar Button Item po obu stronach paska nawigacyjnego:



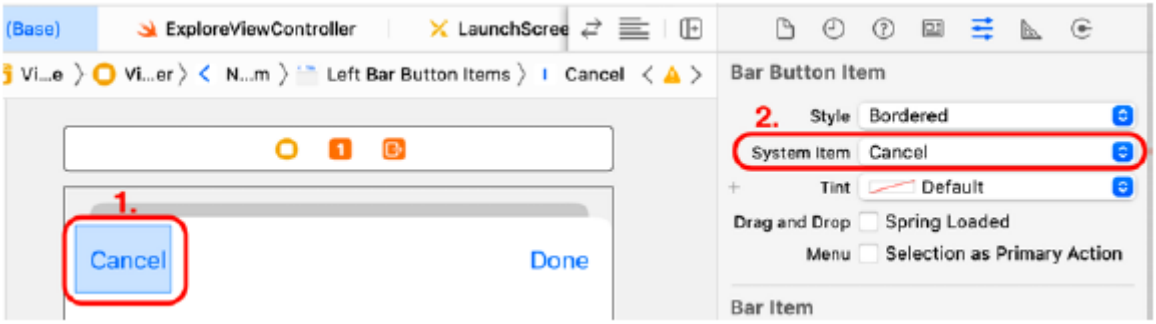
3. Kliknij przycisk Element po prawej stronie:



4. Kliknij przycisk Inspektora atrybutów. W obszarze Element paska wybierz opcję Gotowe z menu Element systemowy:



5. Kliknij przycisk Element po lewej stronie i wybierz opcję Anuluj z menu Element systemowy:

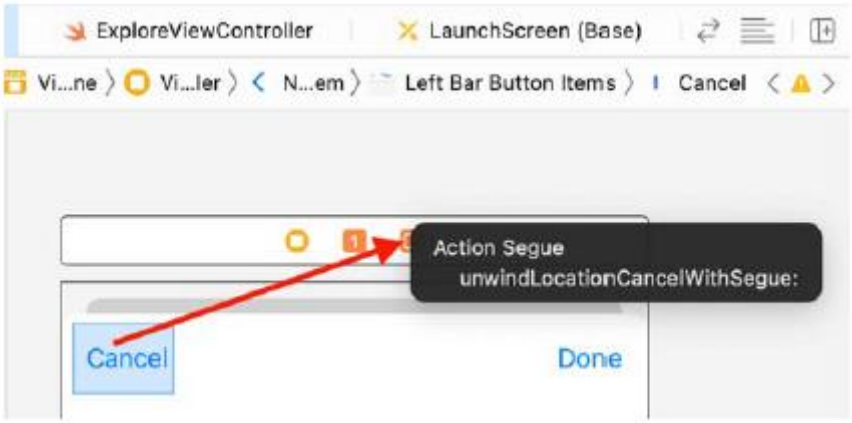


Pamiętaj, że kontroler nawigacji ma właściwość viewControllers, która przechowuje tablicę kontrolerów widoku. Po kliknięciu przycisku na ekranie Eksploruj nowy kontroler widoku zostanie dodany do tablicy viewControllers, a jego widok pojawi się u dołu ekranu, zakrywając ekran Eksploruj.

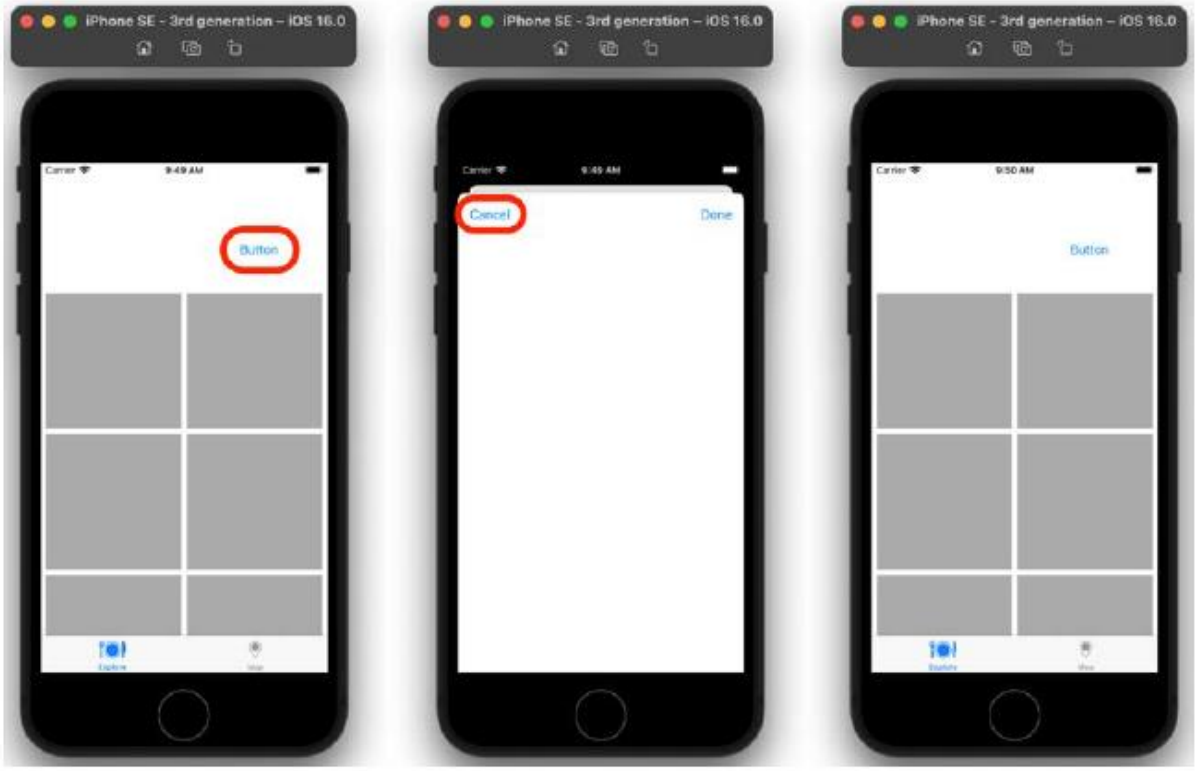
6. Aby zamknąć widok, połącz przycisk Anuluj z wyjściem ze sceny i zaimplementuj metodę w klasie ExploreViewController, która zostanie wykonana, gdy ponownie pojawi się ekran Eksploruj. W nawigatorze projektu kliknij plik ExploreViewController i dodaj następującą metodę na dole pliku, tuż przed ostatnim nawiasem klamrowym:

```
@IBAction func unwindLocationCancel(segue:

UIStoryboardSegue){

}
```

7. Kliknij plik Main scenorysu w nawigatorze projektu. Ctrl + Przeciągnij przycisk Anuluj na ikonę wyjścia ze sceny (trzecia ikona) i wybierz opcję unwindLocationCancelWithSegue: z wyskakującego menu:



Gdy aplikacja jest uruchomiona, kliknięcie przycisku Anuluj spowoduje usunięcie kontrolera widoku z tablicy viewControllers kontrolera nawigacyjnego, co spowoduje zniknięcie widoku prezentowanego modalnie i wykonanie metody unwindLocationCancel(segue:). Pamiętaj, że ta metoda nic nie daje. Zbuduj i uruchom aplikację, a następnie kliknij przycisk w nagłówku sekcji ekranu Eksploruj. Nowy widok pojawi się na ekranie. Po kliknięciu przycisku Anuluj nowy widok zniknie:



Gratulacje! Ukończyłeś podstawową strukturę ekranu Eksploruj.

Streszczenie

W tej części dodałeś widok kolekcji do ekranu Eksploruj w głównym pliku scenorysu i dodałeś nowy plik ExploreViewController, który zawiera implementację klasy ExploreViewController. Uczyniłeś klasę ExploreViewController kontrolerem widoku dla sceny zawierającej widok kolekcji. Następnie zmodyfikowałeś klasę ExploreViewController tak, aby miała miejsce dla widoku kolekcji w scenorysie, i uczyniłaś ją źródłem danych i delegatem dla widoku kolekcji. Dodałeś nagłówek sekcji widoku kolekcji do widoku kolekcji i ustawiłeś rozmiar komórek widoku kolekcji oraz nagłówka sekcji widoku kolekcji.

Na koniec dodałeś przycisk umożliwiający wyświetlenie drugiego widoku i skonfigurowałeś przycisk Anuluj, aby go odrzucić. Na tym etapie powinieneś już dość biegle posługiwać się Konstruktorem interfejsów w celu dodawania widoków i kontrolerów widoków do sceny scenorysu, łączenia wyjść kontrolerów widoku z elementami interfejsu użytkownika w scenorysach, konfigurowania komórek widoków kolekcji i nagłówków sekcji oraz modalnego prezentowania widoków. Będzie to bardzo przydatne podczas projektowania interfejsu użytkownika dla własnych aplikacji. W następnym rozdziale skonfigurujesz nowy kontroler widoku do wyświetlania widoku tabeli, zaimplementujesz pozostałe ekrany aplikacji i zaimplementujesz widok mapy na ekranie Mapa.

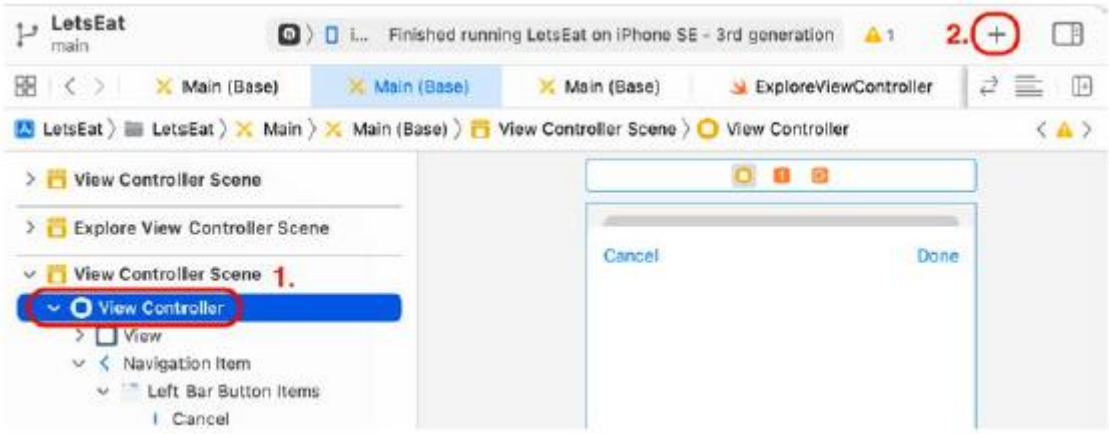
Kończenie interfejsu użytkownika

W poprzedniej części skonfigurowałeś ekran Eksploruj tak, aby wyświetlał 20 pustych komórek widoku kolekcji w widoku kolekcji, dodałeś przycisk do nagłówka sekcji widoku kolekcji, aby modalnie przedstawić widok reprezentujący ekran Lokalizacje, i dodałeś przycisk Anuluj, aby go zamknąć. Tu zaimplementujesz pozostałe ekrany, które zostały pokazane w przewodniku po aplikacji pokazanym wcześniej. Najpierw dodasz pusty widok tabeli do ekranu Lokalizacje. Następnie dodasz ekran Lista restauracji, który wyświetli się po dotknięciu komórki na ekranie Eksploruj. Skonfigurujesz ten ekran tak, aby wyświetlał widok kolekcji zawierający pojedynczą komórkę widoku kolekcji. Następnie dodasz ekran szczegółów restauracji, który zostanie wyświetlony po dotknięciu komórki na ekranie listy restauracji. Skonfigurujesz ten ekran tak, aby wyświetlał widok tabeli ze statycznymi komórkami widoku tabeli. Do jednej z komórek dodasz także przycisk, który po dotknięciu wyświetla widok reprezentujący ekran formularza recenzji. Na koniec sprawisz, że ekran Mapa wyświetli mapę. Pod koniec tego rozdziału dowiesz się, jak dodawać i konfigurować widok tabeli do sceny scenorysu, jak dodawać przejścia między scenami i jak dodawać do sceny widok mapy. Podstawowy interfejs użytkownika Twojej aplikacji będzie kompletny i będziesz mógł przejść przez wszystkie ekrany symulatora.

Dodanie widoku tabeli do ekranu Lokalizacje

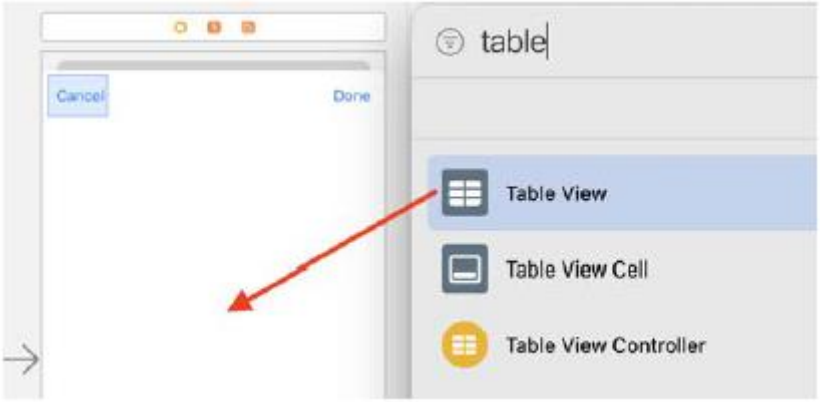
Po dotknięciu przycisku w nagłówku sekcji widoku kolekcji na ekranie Eksploruj inny widok reprezentujący ekran Lokalizacje zostanie wyświetlony modalnie, ale obecnie jest pusty. Dodajmy do tego widoku widok tabeli. Aby to zrobić, wykonaj następujące kroki:

1. Zbuduj i uruchom aplikację LetsEat, aby mieć pewność, że wszystko nadal działa tak, jak powinno. Kliknij plik głównego scenorysu w nawigatorze projektu. W konspekcie dokumentu wybierz ikonę kontrolera widoku w scenie kontrolera widoku prezentowanej modalnie za pomocą przycisku w scenie eksploruj kontrolera widoku. Kliknij przycisk Biblioteka:

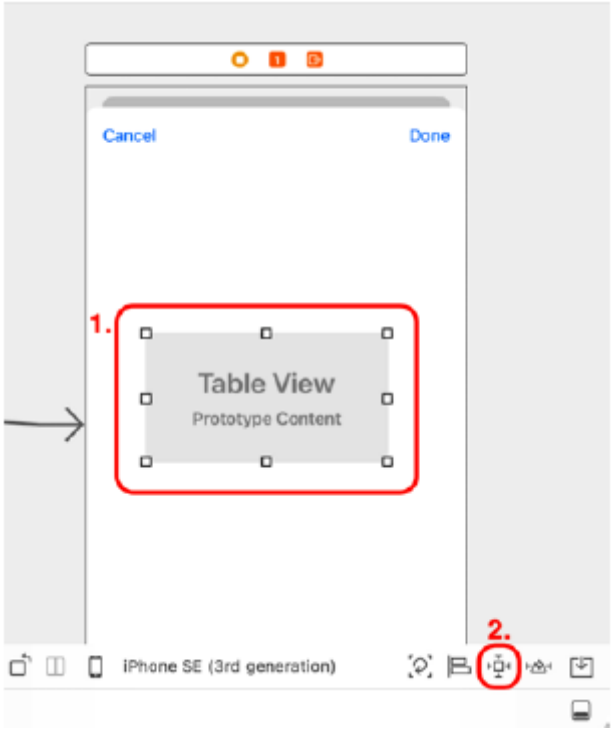


2. Pojawi się biblioteka. Wpisz tabela w polu filtru. W wynikach pojawi się obiekt Widok tabeli.

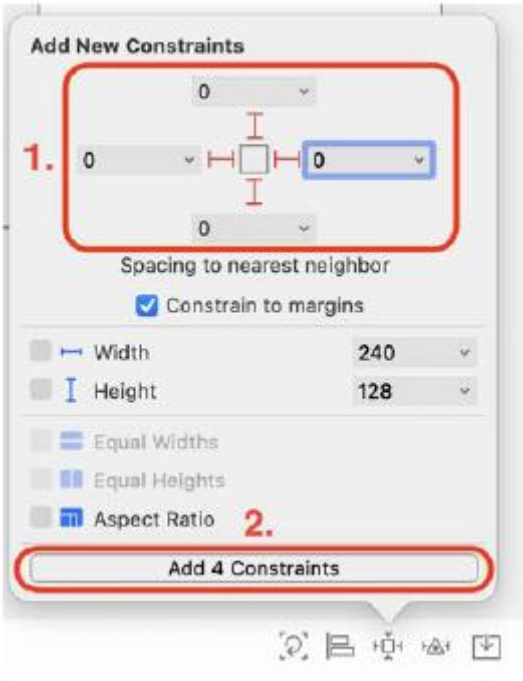
3. Przeciągnij obiekt Widok tabeli do widoku w scenie kontrolera widoku:



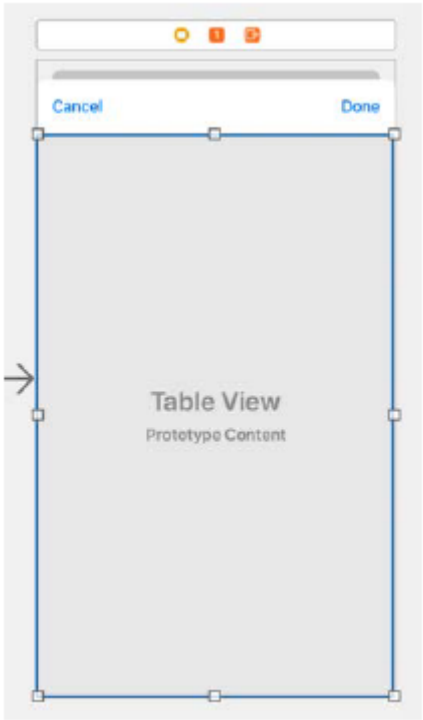
4. Dodasz ograniczenia, aby widok tabeli wypełniał cały ekran. Po wybraniu widoku tabeli kliknij przycisk Dodaj nowe ograniczenia:



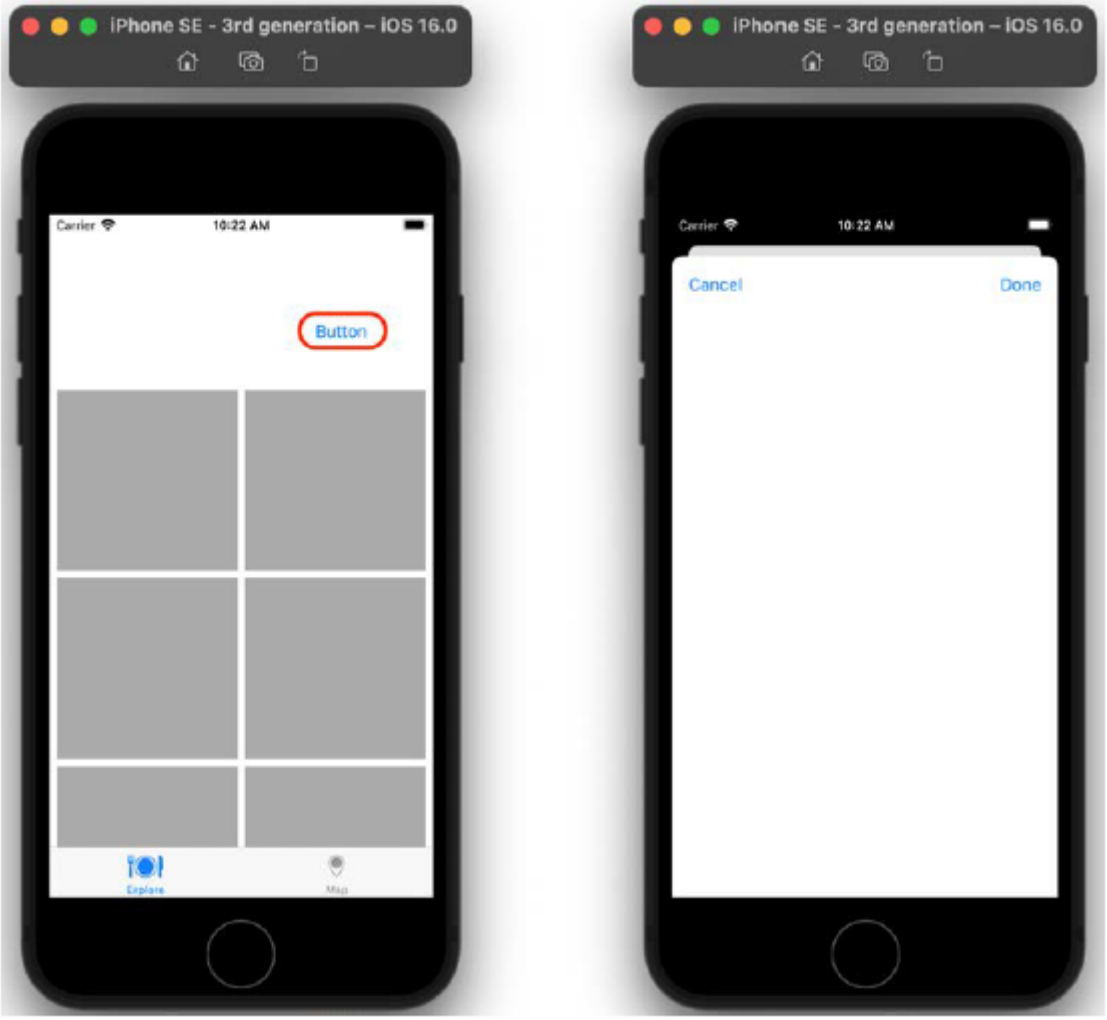
5. Wpisz 0 we wszystkich polach Odstępy do najbliższego sąsiada i upewnij się, że wszystkie bladoczerwone pręty są zaznaczone (staną się jaskrawoczerwone). Kliknij przycisk Dodaj 4 ograniczenia:



6. Sprawdź, czy krawędzie widoku tabeli są teraz zrównane z krawędziami widoku w scenie kontrolera widoku:



Zbuduj i uruchom aplikację, a następnie dotknij przycisku w nagłówku sekcji. Chociaż tego nie widzisz, na ekranie Lokalizacje znajduje się pusty widok tabeli:

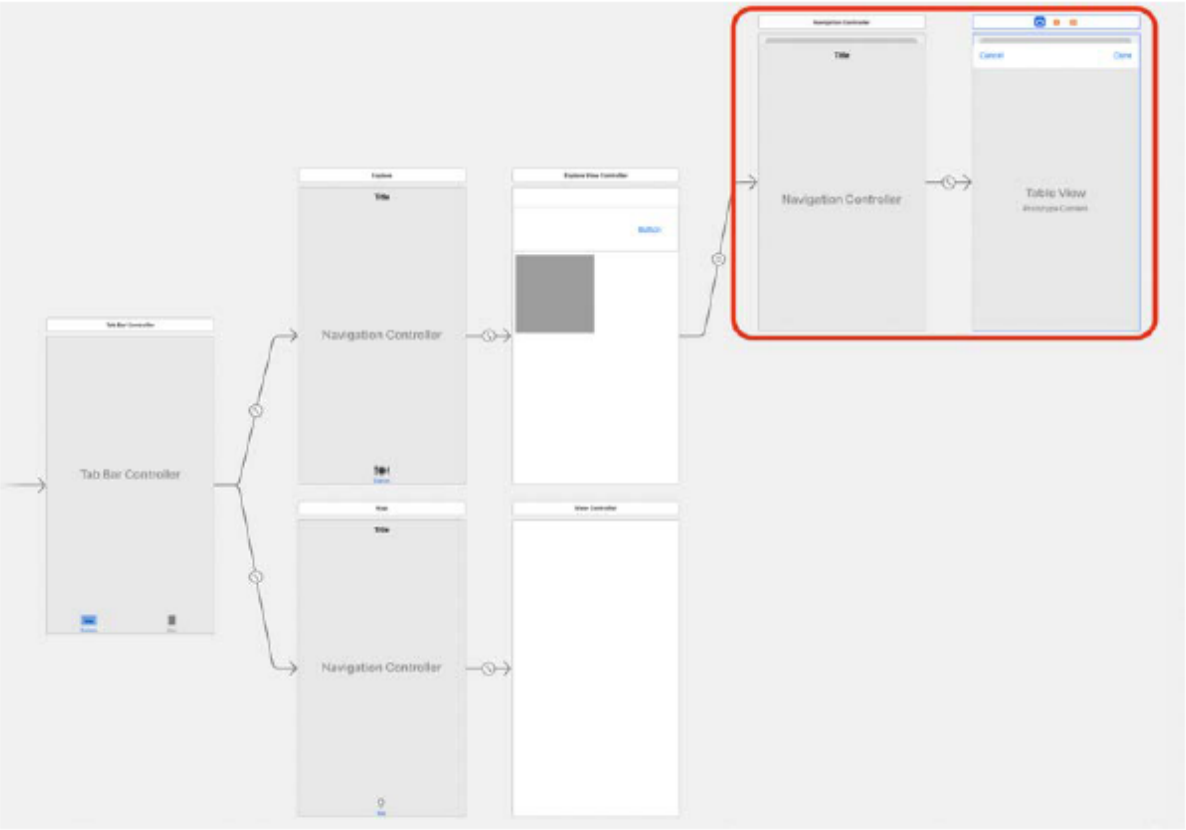


Zaimplementujesz kontroler widoku dla ekranu Lokalizacje w Rozdziale 16, Pierwsze kroki z widokami tabel. Ostatecznie w tym widoku tabeli zostanie wyświetlona lista lokalizacji restauracji, jak pokazano w przewodniku po aplikacji. Jak widać, proces ten przypomina dodanie widoku kolekcji do ekranu Eksploruj, co zrobiłeś w poprzednim rozdziale. W następnej sekcji dodasz do scenorysu scenę kontrolera widoku, która będzie reprezentować ekran Lista restauracji.

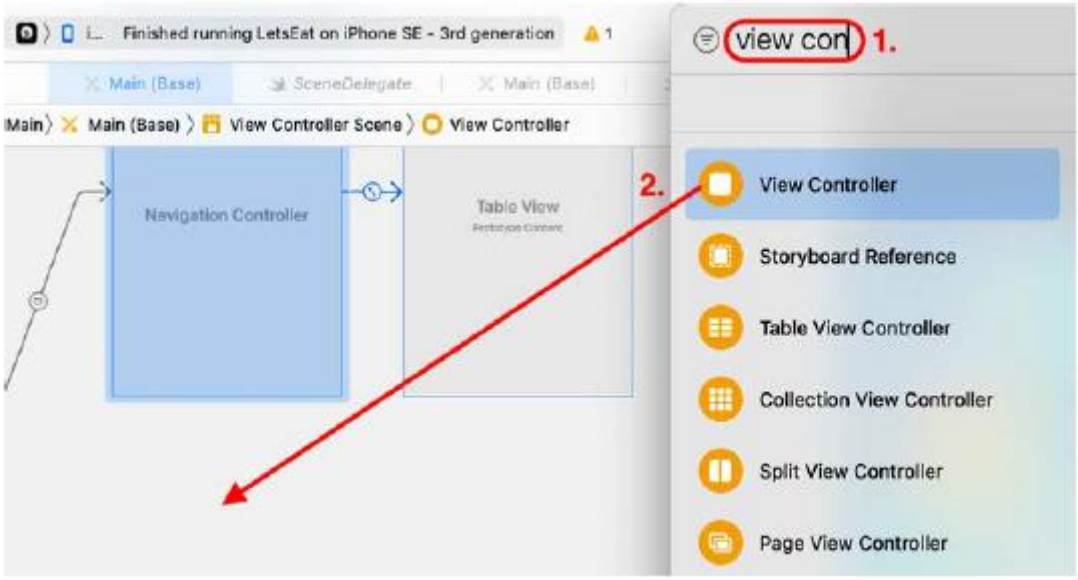
Implementacja ekranu Lista Restauracji

Jak można zobaczyć podczas prezentacji aplikacji w rozdziale 10, Konfigurowanie interfejsu użytkownika, po ustaleniu lokalizacji i dotknięciu kuchni na ekranie Eksploruj pojawi się ekran Lista restauracji, zawierający listę restauracji. Aby zaimplementować ekran Lista restauracji, dodasz nową scenę kontrolera widoku do scenorysu i dodasz widok kolekcji do widoku w tej scenie. Dodasz także nowy plik klasy Cocoa Touch, zadeklarujesz i zdefiniujesz klasę RestaurantListViewController, ustawisz ją jako kontroler widoku dla widoku sceny kontrolera widoku i połączysz wyjścia widoku kolekcji z tą klasą. Kroki są bardzo podobne do tych, które wykonałeś w poprzednim rozdziale dotyczącym klasy ExploreViewController. Zacznijmy od dodania nowej sceny kontrolera widoku. Wykonaj następujące kroki:

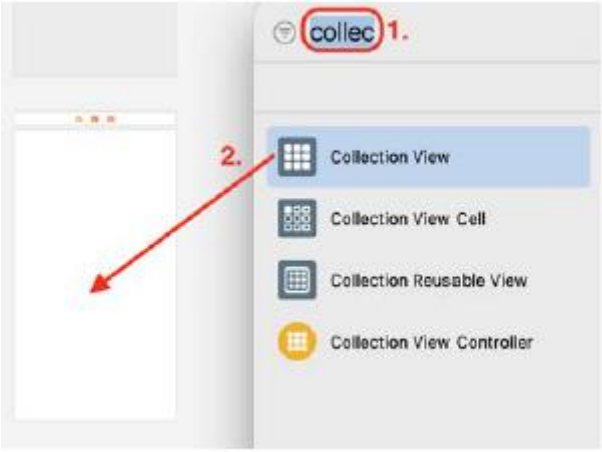
1. W głównym pliku scenorysu przesuń w górę scenę kontrolera nawigacji i kontrolera widoku dodaną w poprzednim rozdziale, aby zrobić miejsce na nową dodaną scenę kontrolera widoku:



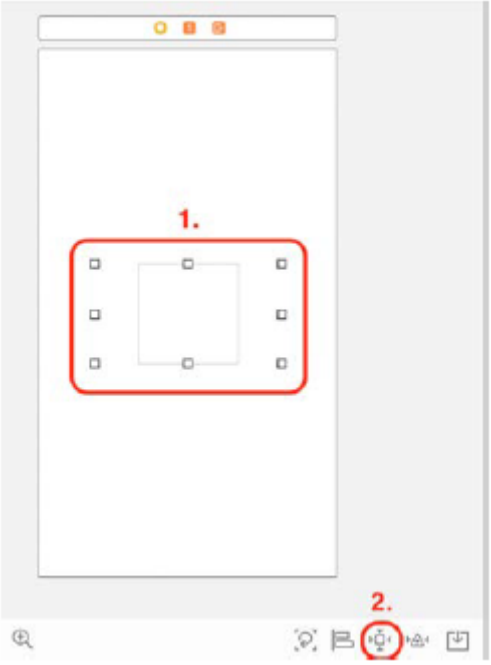
2. Kliknij przycisk Biblioteka i wpisz view con w polu filtru. Obiekt View Controller będzie widoczny w wynikach. Przeciągnij obiekt View Controller do scenorysu, aby reprezentować ekran Lista restauracji:



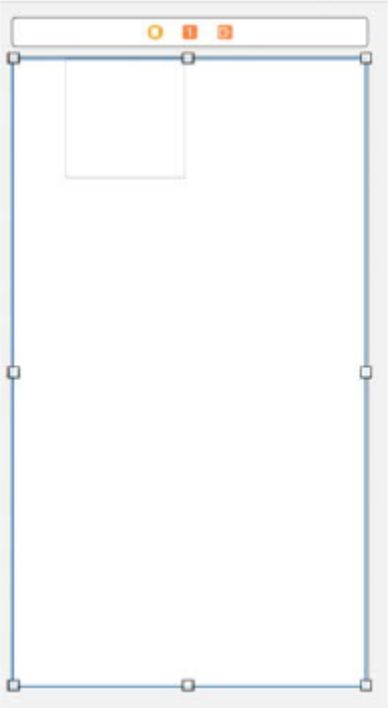
3. Kliknij przycisk Biblioteka i wpisz collec w polu filtru. Obiekt widoku kolekcji będzie widoczny w wynikach. Przeciągnij obiekt widoku kolekcji do widoku w scenie kontrolera widoku:



4. Dodasz ograniczenia, aby widok kolekcji wypełniał cały ekran. Po wybraniu widoku kolekcji kliknij przycisk Dodaj nowe ograniczenia:



5. Wpisz 0 we wszystkich polach Odstępy do najbliższego sąsiada i upewnij się, że wszystkie bladoczerwone pręty są zaznaczone (staną się jaskrawoczerwone). Kliknij przycisk Dodaj 4 wiązania. Sprawdź, czy krawędzie widoku kolekcji są teraz zrównane z krawędziami widoku w scenie kontrolera widoku:



Dodano scenę kontrolera widoku dla ekranu Lista restauracji, ale nie ma ona jeszcze kontrolera widoku. Będziesz go potrzebował, aby wyświetlić komórkę widoku kolekcji. W następnej sekcji dodasz do swojej aplikacji nowy plik klasy Cocoa Touch, dzięki czemu będziesz mógł zadeklarować i zdefiniować nową klasę kontrolera widoku dla tego ekranu.

Deklarowanie klasy RestaurantListViewController

Podobnie jak w poprzednim rozdziale, dodasz do swojego projektu nowy plik Cocoa Touch Class, ale tym razem zaimplementujesz klasę RestaurantListViewController. Instancję tej klasy wykorzystasz jako kontroler widoku na ekranie Lista restauracji. Wykonaj następujące kroki:

1. Kliknij przyciski Nawigator i Inspektor, aby włączyć obszary Nawigator i Inspektor.

2. Kliknij prawym przyciskiem myszy grupę LetsEat i wybierz Nowa grupa z wyskakującego menu.

3. Nadaj tej nowej grupie nazwę Restauracje. Jeśli się pomylisz, kliknij nazwę i naciśnij klawisz Return na klawiaturze, aby ponownie umożliwić jej edycję.

4. Kliknij prawym przyciskiem myszy grupę Restauracje i wybierz Nowy plik... .

5. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

6. Pojawi się ekran Wybierz opcje nowego pliku. Wpisz następujące informacje w polach Klasa i Podklasa:

Klasa: RestaurantListViewController

Podklasa: UIViewController

Kliknij Dalej, kiedy skończysz.

7. Na następnym ekranie kliknij Utwórz.

8. Do projektu został dodany plik RestaurantListViewController, w którym zobaczysz standardowy kod klasy RestaurantListViewController. Klasa RestaurantListViewController jest podklasą klasy UIViewController i zawiera jedną metodę viewDidLoad(). Podobnie jak to zrobiłeś wcześniej w poprzednim rozdziale, usuń skomentowany kod po klasie viewDidLoad() w klasie RestaurantListViewController, aż pozostanie tylko kod pokazany na poniższym zrzucie ekranu:

```
4  //
5  //  Created by iOS 16 Programming for Beginners
       on 15/06/2022.
6  //
7
8  import UIKit
9
10 class RestaurantListViewController:
       UIViewController {
11
12     override func viewDidLoad() {
13         super.viewDidLoad()
14
15         // Do any additional setup after
               loading the view.
16     }
17
18 }
19
```

Podobnie jak poprzednio w przypadku ekranu Eksploruj, uczynisz klasę RestaurantListViewController kontrolerem widoku dla widoku w scenie kontrolera widoku i przyjmiesz źródło danych widoku kolekcji oraz protokoły delegowania. Dodasz także ręcznie wylot widoku kolekcji w definicji klasy i użyj Inspektora połączeń, aby połączyć wylot z widokiem kolekcji w scenorysie. Zrobisz to w następnej sekcji.

Przyjęcie protokołów delegatów i źródeł danych

Zmodyfikujesz klasę RestaurantListViewController, aby była zgodna z protokołami UICollectionViewDataSource i UICollectionViewDelegate oraz dodasz wszelkie wymagane metody protokołów. Dodasz także punkt wyjściowy dla widoku kolekcji i uczynisz instancję klasy RestaurantListViewController kontrolerem widoku dla tego widoku. Wykonaj następujące kroki:

1. Zmodyfikuj deklarację klasy RestaurantListViewController, jak pokazano, aby przyjęła protokoły UICollectionViewDataSource i UICollectionViewDelegate:

class RestaurantListViewController: UIViewController,

UICollectionViewDataSource, UICollectionViewDelegate {

2. Gdy pojawi się ikona błędu, kliknij ją.

3. Zobaczysz ten błąd, ponieważ metody wymagane do zgodności z dodanymi protokołami nie są obecne w definicji klasy. Kliknij przycisk Napraw, aby dodać fragmenty wymaganych metod do definicji klasy.

4. Sprawdź, czy do pliku dodano kody pośredniczące metod. Zmień wszystko tak, aby kody pośredniczące znajdowały się po metodzie viewDidLoad():

```
10 class RestaurantListViewController:
12     override func viewDidLoad() {
15         // Do any additional setup after
               loading the view.
16     }
17
18     func collectionView(_ collectionView:
           UICollectionView,
           numberOfItemsInSection section: Int) ->
           Int {
19         code
20     }
21
22     func collectionView(_ collectionView:
           UICollectionView, cellForItemAt
           indexPath: IndexPath) ->
           UICollectionViewCell {
23         code
24     }
```

5. Zmodyfikuj wycinki metod zgodnie z ilustracją, aby po uruchomieniu aplikacji widok kolekcji wyświetlał na ekranie pojedynczą komórkę widoku kolekcji:

func collectionView(_ collectionView: UICollectionView,

numberOfItemsInSection section: Int) -> Int {

1

}

func collectionView(_ collectionView: UICollectionView, cellForItemAt

indexPath: IndexPath) -> UICollectionViewCell

{

collectionView.dequeueReusableCell(

withReuseIdentifier: "restaurantCell",

for: indexPath)

}

6. Sprawdź, czy Twój kod wygląda następująco:

```
10 class RestaurantListViewController:
17
18     func collectionView(_ collectionView:
           UICollectionView,
           numberOfItemsInSection section: Int) ->
           Int {
19         1
20     }
21
22     func collectionView(_ collectionView:
           UICollectionView, cellForItemAt
           indexPath: IndexPath) ->
           UICollectionViewCell {
23         collectionView
               .dequeueReusableCell
               (withReuseIdentifier:
               "restaurantCell", for: indexPath)
24     }
25
```

7. Dodaj outlet,kolekcjaView, zaraz po deklaracji klasy:

@IBOutlet var collectionView: UICollectionView!

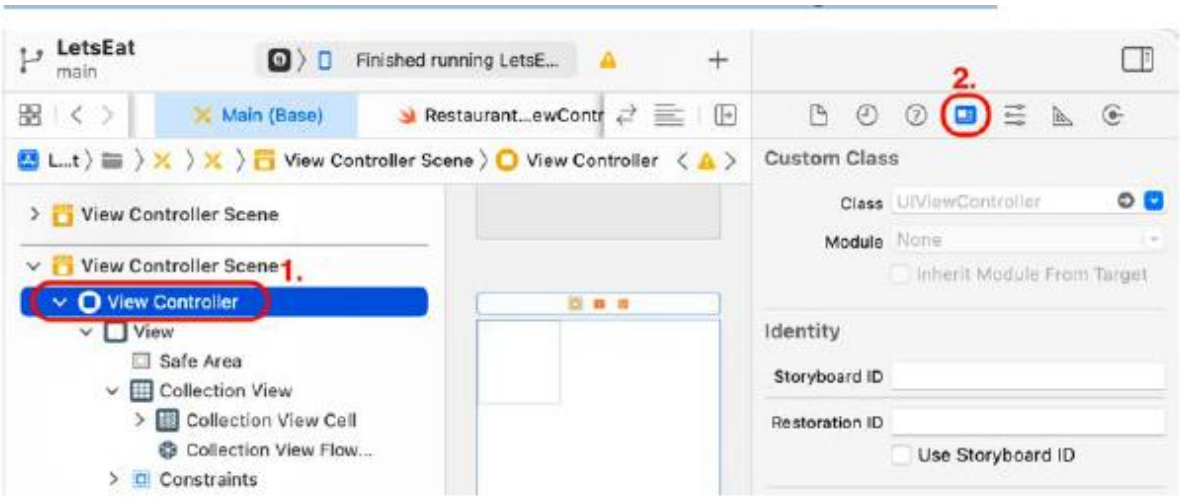
Później połączysz to z widokiem kolekcji w scenorysie.

8. Sprawdź, czy Twój kod wygląda następująco:

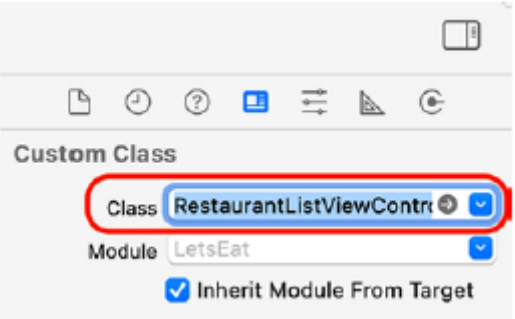
```
10 class RestaurantListViewController:
       UIViewController,
       UICollectionViewDataSource,
       UICollectionViewDelegate {
11
       @IBOutlet var collectionView:
           UICollectionView!
```

Nie będziesz używać asystenta edytora do łączenia wylotu z widokiem kolekcji, tak jak to robiłeś w poprzednim rozdziale. Jest to kwestia osobistych preferencji – możesz wybrać, która metoda najbardziej Ci odpowiada.

9. Kliknij plik głównego scenorysu i kliknij ikonę kontrolera widoku nowo dodanej sceny kontrolera widoku w konspekcie dokumentu. Kliknij przycisk Inspektora tożsamości:

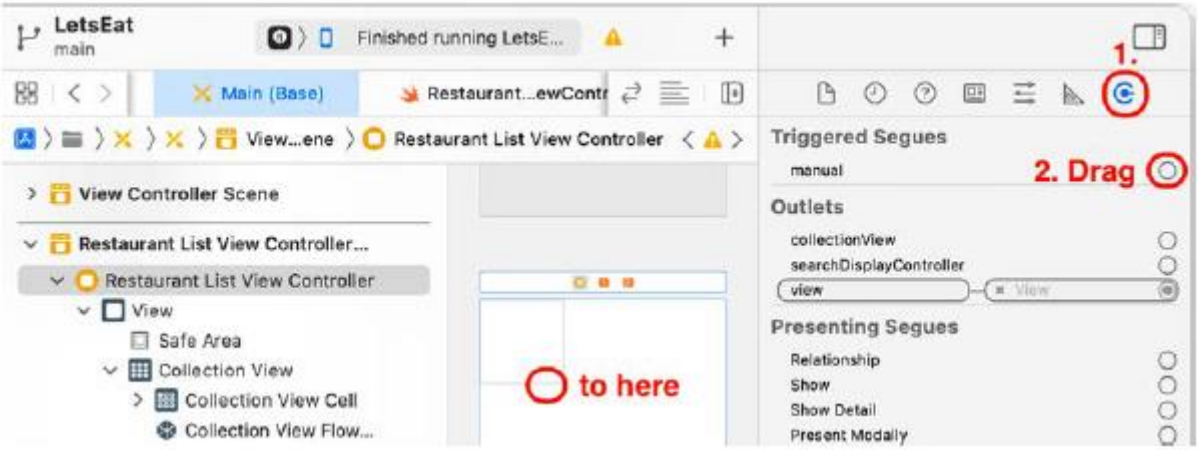


10. Aby instancja klasy RestaurantListViewController stała się kontrolerem widoku dla tej sceny, w polu Klasa wybierz RestaurantListViewController:

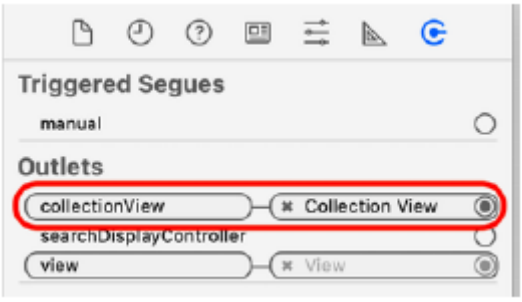


Należy pamiętać, że nazwa sceny kontrolera widoku została zmieniona na Scena kontrolera widoku listy restauracji.

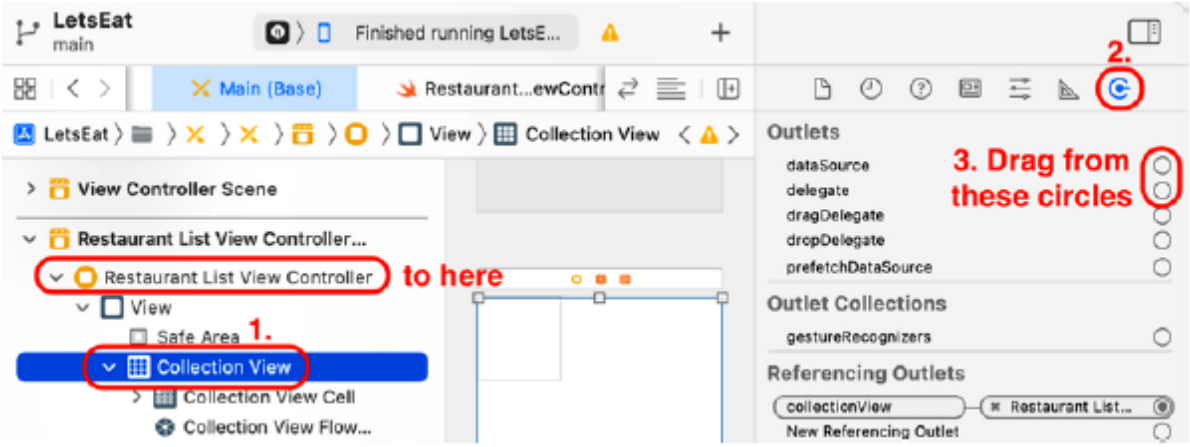
11. Kliknij przycisk Inspektora połączeń. Aby przypisać widok kolekcji w scenie kontrolera widoku listy restauracji do gniazdka w definicji klasy RestaurantListViewController, przeciągnij z okręgu obok gniazdka CollectionView do widoku kolekcji w scenie kontrolera widoku listy restauracji:



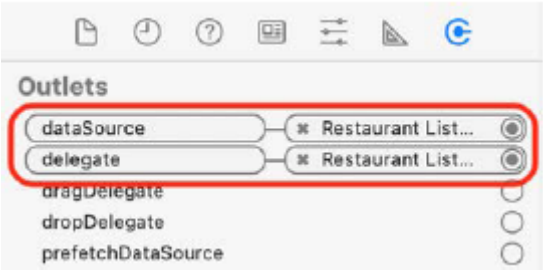
12. Sprawdź, czy widok kolekcji w scenie kontrolera widoku listy restauracji i wylot kolekcjiView w definicji klasy RestaurantListViewController są teraz połączone:



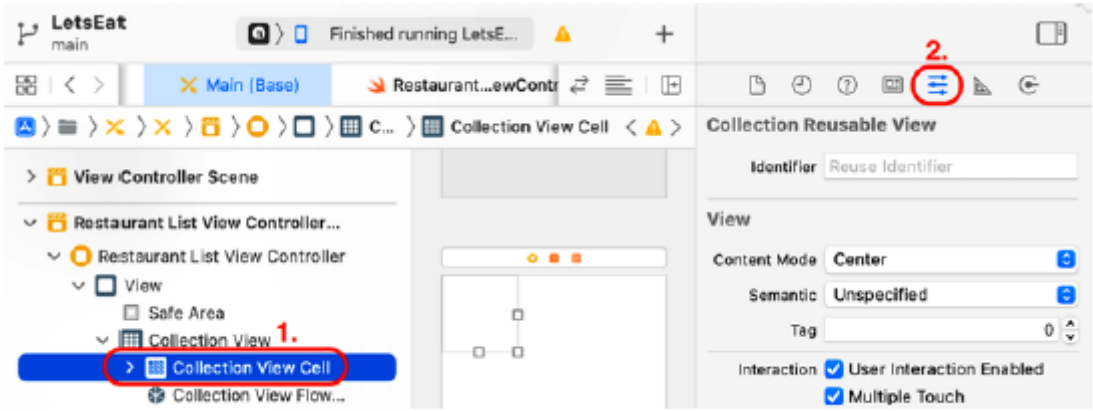
13. Kliknij Widok kolekcji w konspekcie dokumentu, a następnie kliknij Inspektor połączeń. Aby uczynić instancję klasy RestaurantListViewController źródłem danych i obiektem delegowanym dla widoku kolekcji, przeciągnij z okręgów obok źródła danych i deleguj punkty sprzedaży do ikony kontrolera widoku listy restauracji w konspekcie dokumentu:



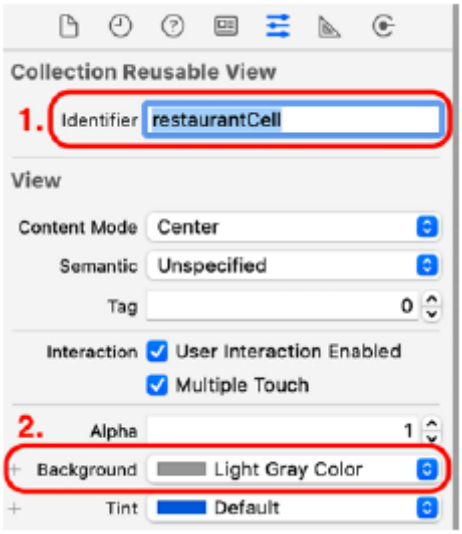
14. Sprawdź, czy źródła danych i gniazda delegatów są teraz połączone:



15. Kliknij komórkę Widok kolekcji w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów, aby ustawić identyfikator i kolor komórki widoku kolekcji



16. Ustaw Identyfikator komórki widoku kolekcji na RestaurantCell i ustaw Kolor tła na Jasnoszary Kolor:

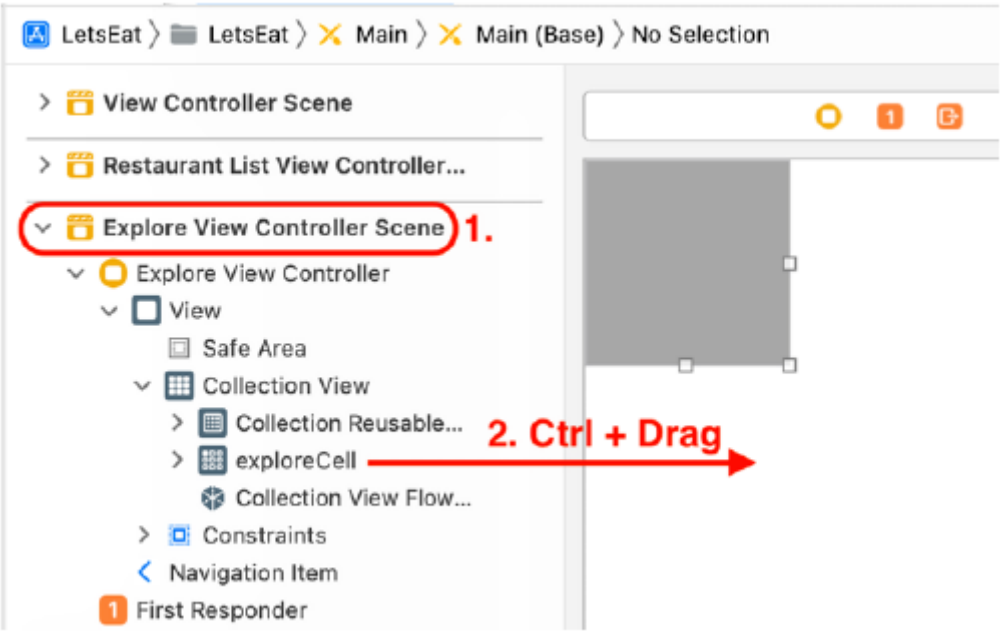


Konfiguracja sceny kontrolera widoku listy restauracji została zakończona. Teraz musisz wyświetlić ten ekran po dotknięciu komórki na ekranie Eksploruj. Aby to zrobić, w następnej sekcji dodasz przejście pomiędzy ekranem Eksploruj a ekranem Lista restauracji.

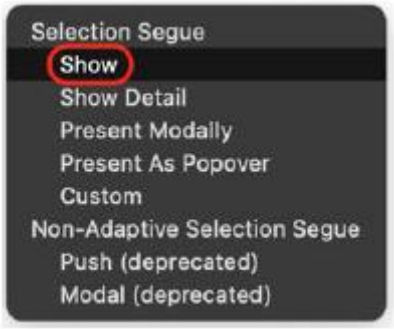
Wyświetlanie ekranu listy restauracji

W poprzednim rozdziale dodałeś przejście, które powoduje wyświetlenie ekranu Lokalizacje po dotknięciu przycisku na ekranie Eksploruj. Aby wyświetlić ekran Lista restauracji po dotknięciu komórki na ekranie Eksploruj, użyj również przejścia. Wykonaj następujące kroki:

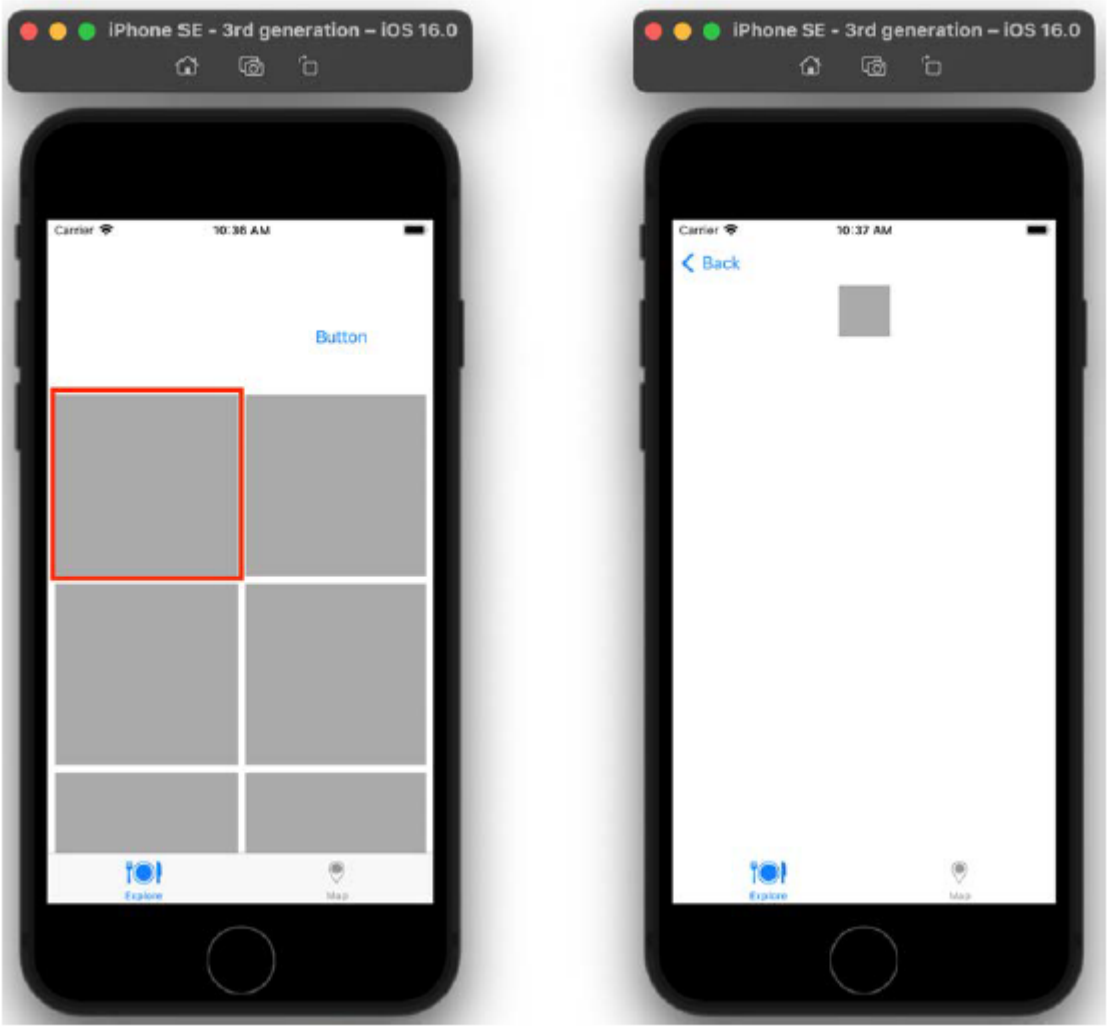
1. Kliknij opcję Przeglądaj scenę kontrolera widoku w konspekcie dokumentu. Ctrl + Przeciągnij z exploreCell w konspekcie dokumentu do sceny kontrolera widoku listy restauracji, aby dodać przejście między nimi:



2. Pojawi się menu Przejdź. Z menu wybierz opcję Pokaż:



Dzięki temu ekran Lista restauracji przesuwa się z prawej strony po dotknięciu komórki na ekranie Eksploruj. Na pasku nawigacyjnym pojawi się przycisk <Wstecz. Kompiluj i uruchamiaj swoją aplikację. Na ekranie Eksploruj dotknij komórki. Powinieneś zobaczyć ekran Lista restauracji z widokiem kolekcji zawierającym pojedynczą komórkę. Dotknięcie przycisku <Wstecz na pasku nawigacyjnym spowoduje zamknięcie ekranu Lista restauracji:

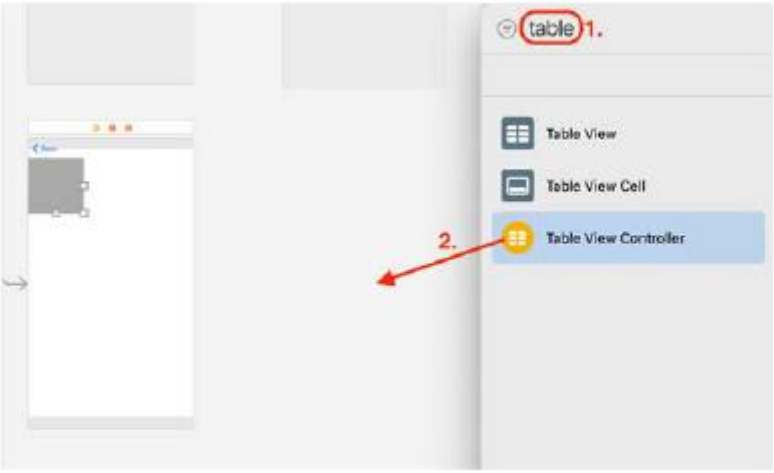


Implementacja ekranu Lista restauracji została zakończona i możesz przechodzić z ekranu Eksploruj do ekranu Lista restauracji i z powrotem. Ostatecznie widok kolekcji na tym ekranie wyświetli listę restauracji w określonej lokalizacji, jak pokazano w przewodniku po aplikacji. Świetnie! Następną rzeczą, którą zrobisz, będzie dodanie sceny kontrolera widoku, która będzie reprezentować ekran szczegółów restauracji. Ten ekran zostanie wyświetlony po dotknięciu komórki na ekranie Lista restauracji. Zrobisz to w następnej sekcji.

Implementacja ekranu szczegółów restauracji

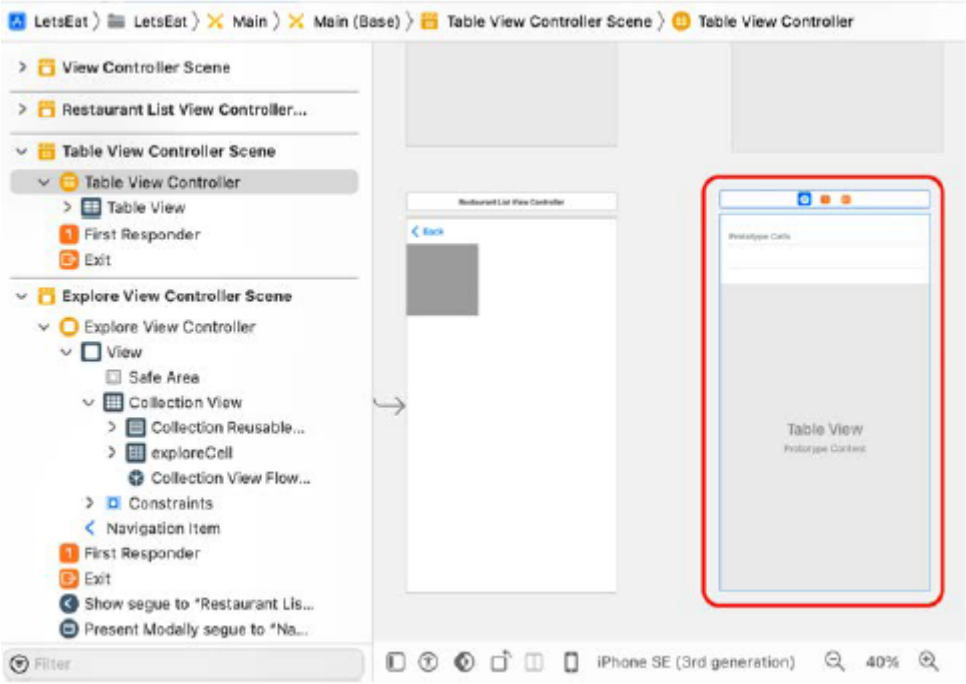
Jak pokazano w prezentacji aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika, po dotknięciu restauracji na ekranie Lista restauracji pojawi się ekran Szczegóły restauracji zawierający szczegółowe informacje o tej restauracji. Kliknięcie przycisku Dodaj recenzję spowoduje wyświetlenie ekranu Formularz recenzji, na którym możesz dodać recenzje, a dotknięcie przycisku Dodaj zdjęcie spowoduje wyświetlenie ekranu Filtr zdjęć, na którym możesz dodawać zdjęcia i stosować do nich filtry. W tej sekcji dodasz do scenorysu nową scenę kontrolera widoku tabeli, która będzie reprezentować ekran szczegółów restauracji, oraz dodasz drugą scenę kontrolera widoku, która będzie reprezentować ekran formularza recenzji. Umieścisz przycisk w jednej z komórek widoku tabeli, aby wyświetlić ekran formularza recenzji. Zacznijmy od dodania nowej sceny kontrolera widoku. Wykonaj następujące kroki:

1. Kliknij przycisk Biblioteka, wpisz tabela w polu filtru i przeciągnij obiekt Kontrolera widoku tabeli do scenorysu obok sceny kontrolera widoku listy restauracji:



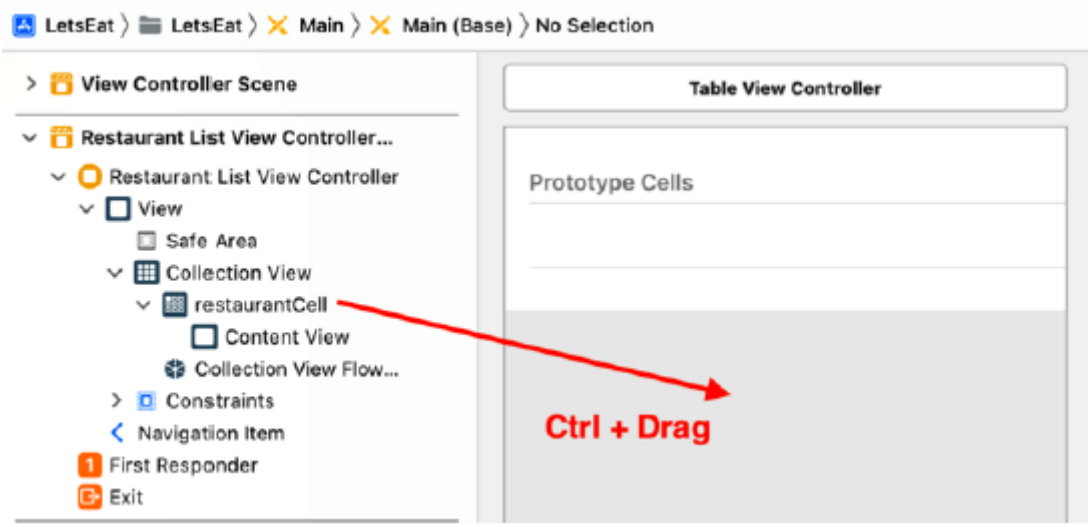
Będzie to reprezentować ekran szczegółów restauracji.

2. Sprawdź, czy dodano scenę kontrolera widoku tabeli:



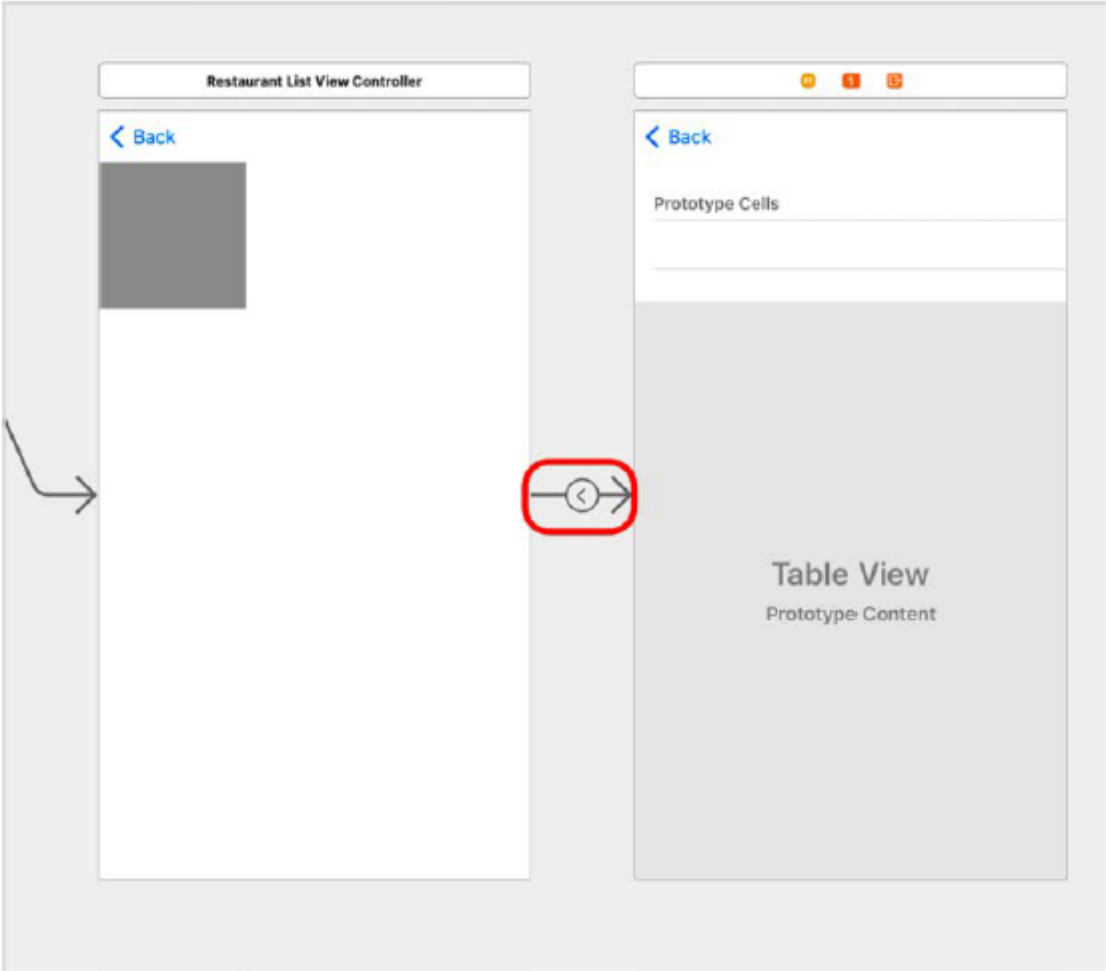
Zauważ, że zawiera już widok tabeli, więc nie musisz dodawać widoku tabeli do sceny, tak jak to zrobiłeś w poprzedniej sekcji.

3. Aby wyświetlić ekran szczegółów restauracji po dotknięciu komórki na ekranie Lista restauracji, naciśnij klawisz Ctrl i przeciągnij z komórki restauracji (w konspekcie dokumentu pod sceną kontrolera widoku listy restauracji) do widoku tabeli, aby dodać przejście między nimi:

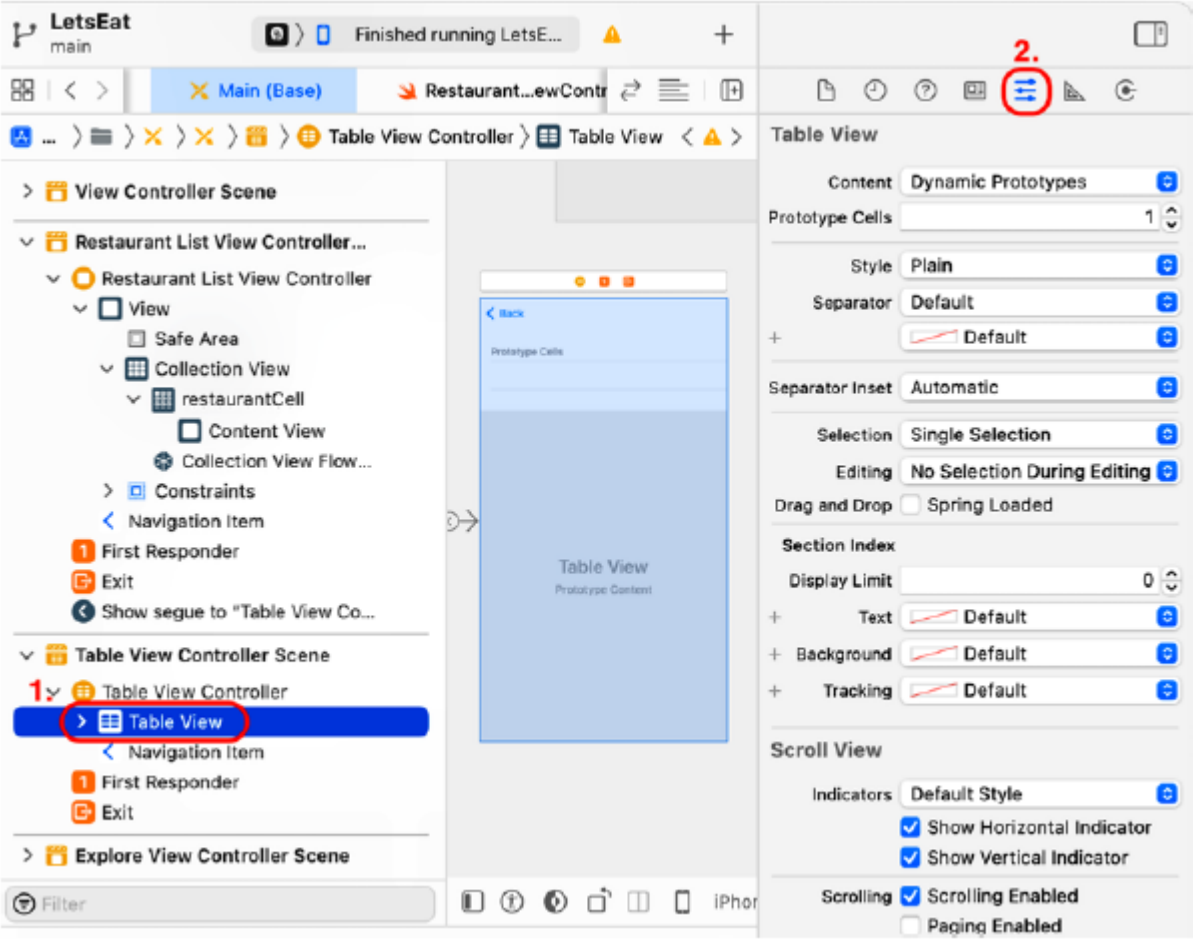


4. Wybierz opcję Pokaż z menu Przejście. Dzięki temu ekran szczegółów restauracji przesuwa się z prawej strony po dotknięciu komórki na ekranie listy restauracji. Na pasku nawigacyjnym pojawi się przycisk <Wstecz.

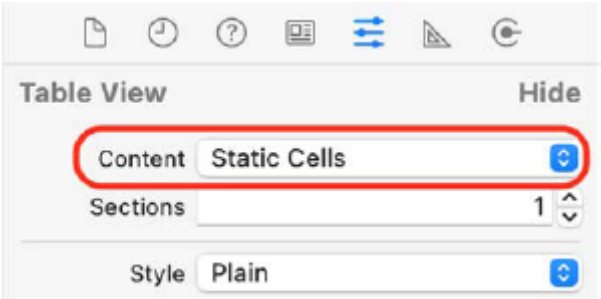
5. Sprawdź, czy pomiędzy dwiema scenami pojawiła się przerwa:



6. Ekran szczegółów restauracji zawsze wyświetla stałą liczbę komórek. W konspekcie dokumentu kliknij opcję Widok tabeli w obszarze Scena kontrolera widoku tabeli i kliknij przycisk Inspektora atrybutów



7. Ustaw Treść na Komórki statyczne, aby na ekranie Szczegóły restauracji wyświetlana była stała liczba komórek.



Robisz to, ponieważ ekran szczegółów restauracji zawsze wykorzystuje tę samą liczbę komórek do wyświetlania szczegółów restauracji. Kompiluj i uruchamiaj swoją aplikację. Kliknij komórkę na ekranie Eksploruj, aby wyświetlić ekran Lista restauracji. Następnie kliknij komórkę na ekranie Lista restauracji, aby wyświetlić ekran Szczegóły restauracji:

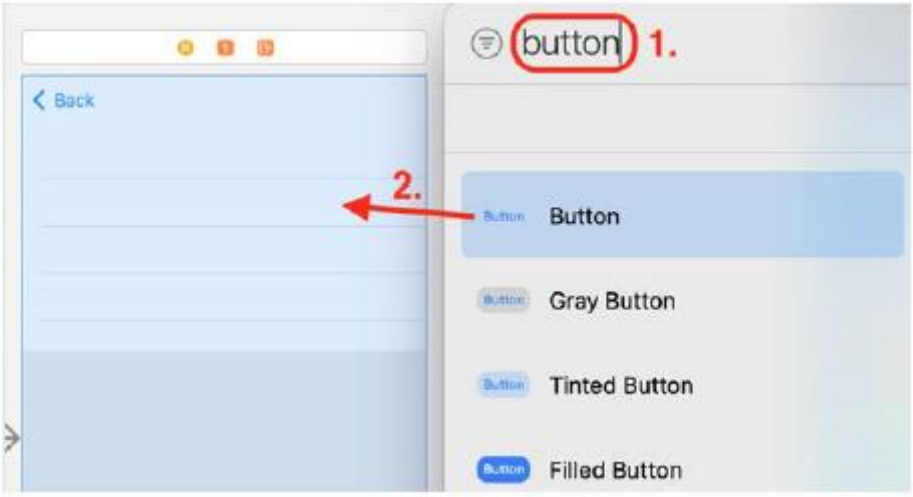
Kliknij przycisk <Wstecz, aby wrócić do ekranu Lista restauracji.

W następnej sekcji zaimplementujesz przycisk wewnątrz jednej z komórek widoku tabeli, aby wyświetlić ekran reprezentujący ekran formularza recenzji.

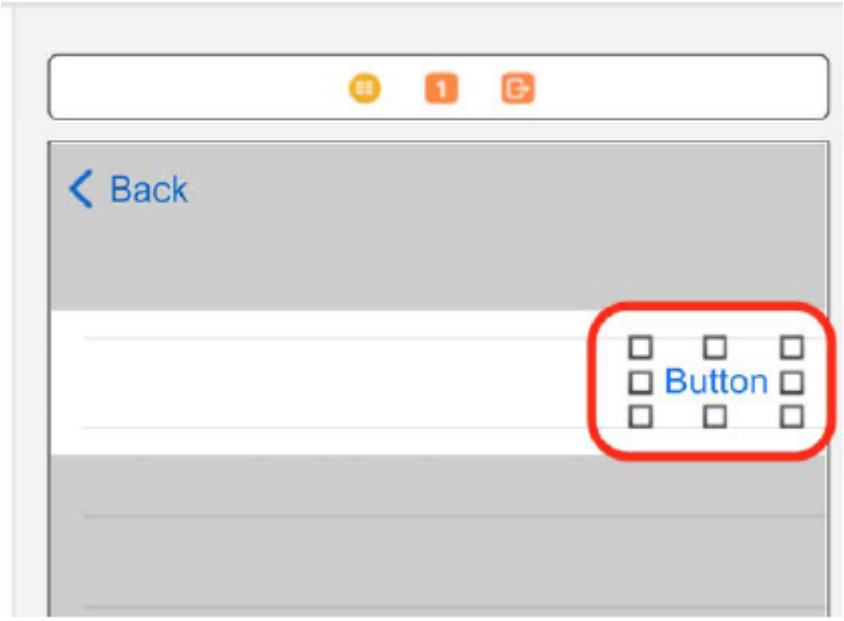
Implementacja ekranu formularza recenzji

W tej sekcji zaimplementujesz nową scenę kontrolera widoku, która będzie reprezentować ekran formularza recenzji, i skonfigurujesz przycisk na ekranie szczegółów restauracji, aby go wyświetlić. Wykonaj następujące kroki:

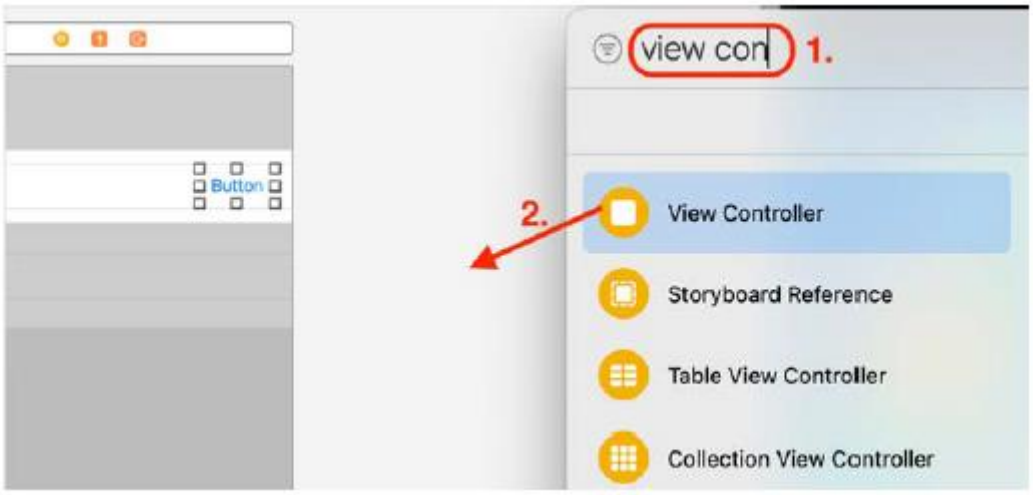
1. Potrzebujesz przycisku na ekranie szczegółów restauracji, aby po dotknięciu wyświetlić ekran formularza recenzji. Kliknij przycisk Biblioteka i wpisz przycisk w polu filtru. Obiekt Button pojawi się jako jeden z wyników. Przeciągnij go do górnej komórki statycznej w scenie kontrolera widoku tabeli reprezentującej ekran szczegółów restauracji:



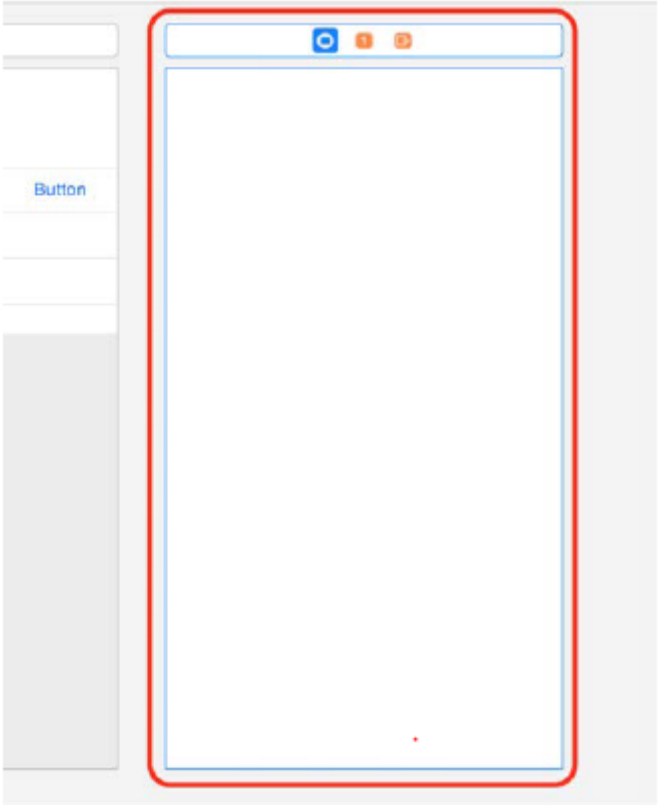
2. Umieść go po prawej stronie komórki:



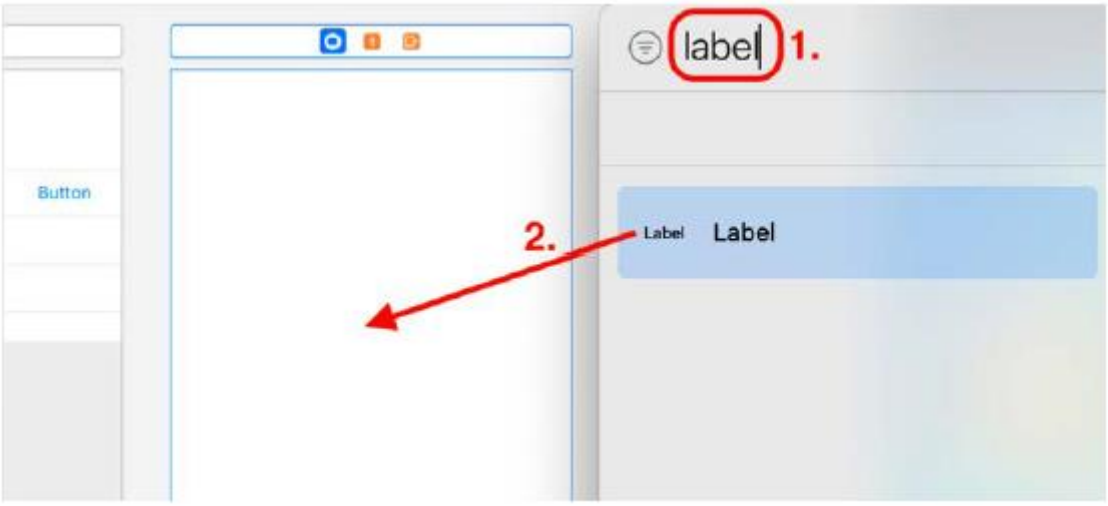
3. Kliknij przycisk Biblioteka i wpisz view con w polu filtru. Obiekt View Controller pojawi się jako jeden z wyników. Przeciągnij go obok sceny kontrolera widoku tabeli, aby wyświetlić ekran formularza recenzji:



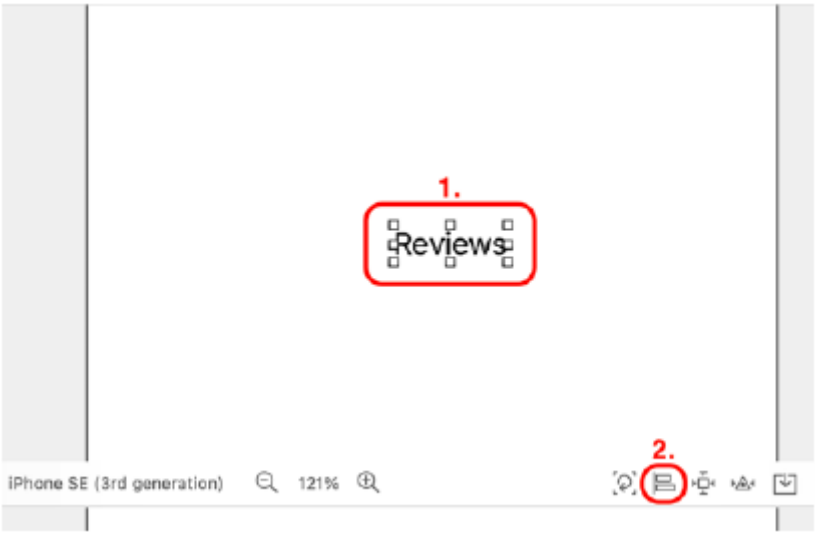
4. Sprawdź, czy została dodana nowa scena kontrolera widoku:



5. Kliknij przycisk Biblioteka i wpisz etykietę w polu filtru. Obiekt Label pojawi się jako jeden z wyników. Przeciągnij go na środek nowej sceny kontrolera widoku, aby przedstawić recenzję:



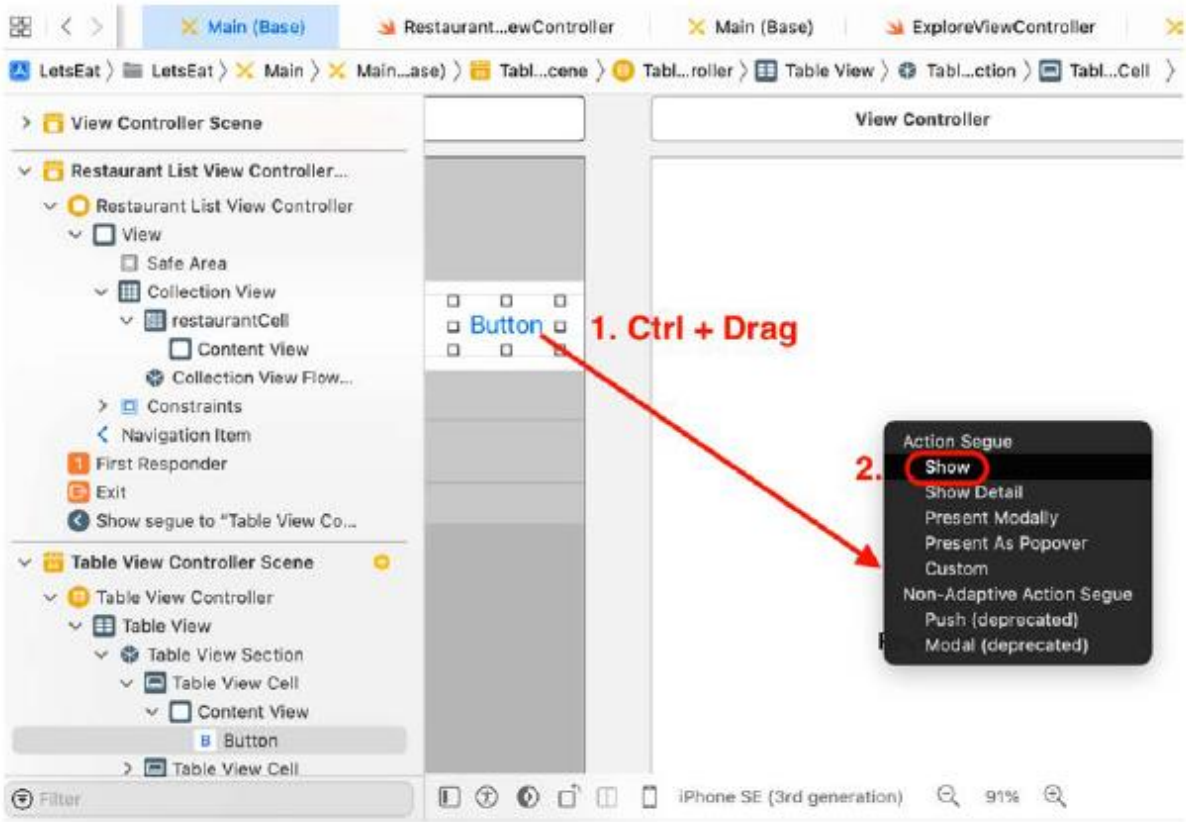
6. Zmień tekst etykiety na Recenzje. Kliknij przycisk Wyrównaj, aby dodać do niego wiązania poziome i pionowe:



7. Zaznacz pola wyboru Poziomo w kontenerze i Pionowo w kontenerze. Kliknij przycisk Dodaj 2 wiązania. Sprawdź, czy ograniczenia zostały dodane:

Te ograniczenia zapewniają, że etykieta Recenzje będzie zawsze znajdować się na środku ekranu po uruchomieniu aplikacji, niezależnie od orientacji i rozmiaru ekranu.

8. Ctrl + Przeciągnij z przycisku w komórce widoku tabeli do nowo dodanej sceny kontrolera widoku i wybierz Pokaż z wyskakującego menu. Spowoduje to wyświetlenie ekranu formularza recenzji po dotknięciu przycisku:



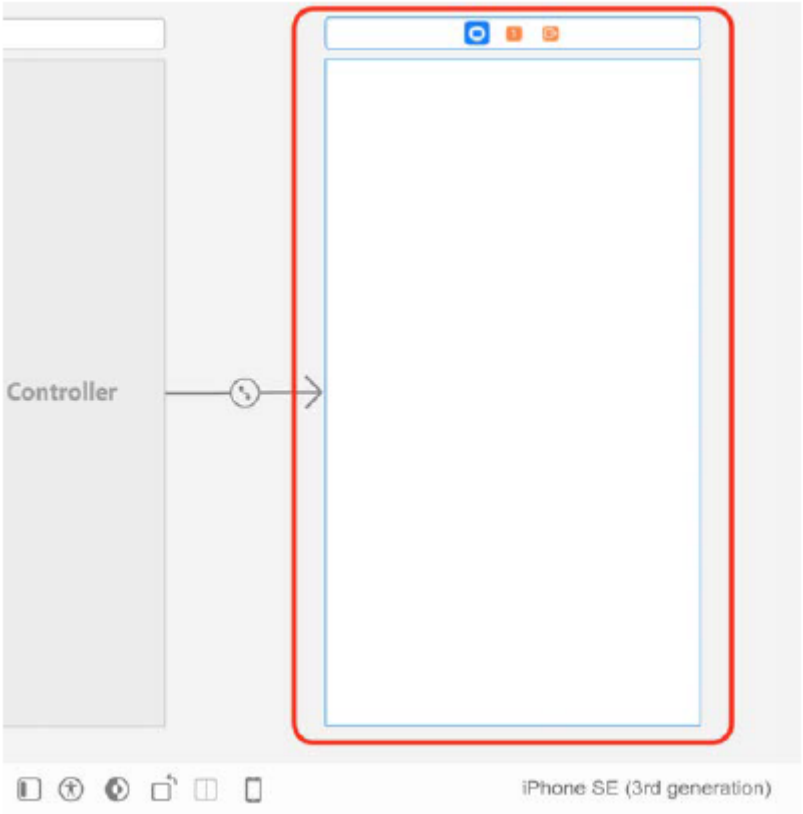
Kompiluj i uruchamiaj swoją aplikację. Kliknij komórkę na ekranie Eksploruj, a następnie kliknij komórkę na ekranie Lista restauracji. Kliknij przycisk na ekranie szczegółów restauracji, aby wyświetlić ekran formularza recenzji:

Fantastyczny! Wszystkie ekrany dostępne z karty Eksploruj, z wyjątkiem ekranu Filtru zdjęć, zostały teraz zaimplementowane i nie wymagają prawie żadnego kodowania! Jeśli chcesz, możesz powtórzyć kroki opisane w tej sekcji, aby dodać ekran Filtr zdjęć. Ostatnią rzeczą do zrobienia jest wyświetlenie mapy na ekranie Mapa. Zrobisz to w następnej sekcji.

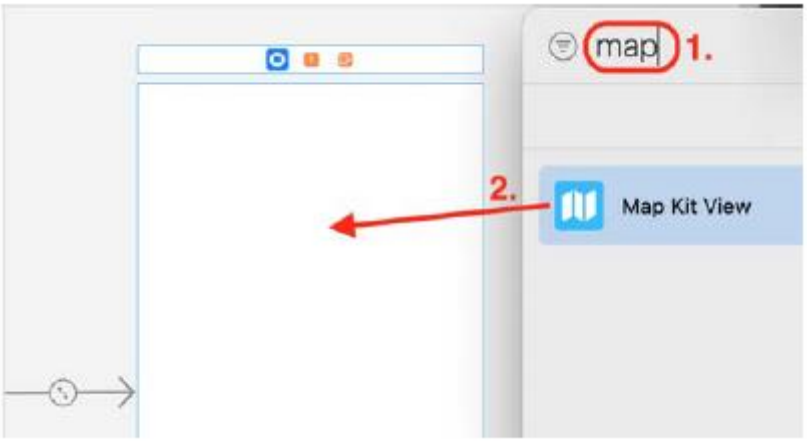
Implementacja ekranu Mapa

Po uruchomieniu aplikacji zostanie wyświetlony ekran Eksploruj. Dotknięcie przycisku Mapa na pasku kart powoduje wyświetlenie ekranu Mapa, ale jest on pusty. Aby ekran Mapa wyświetlał mapę, dodasz widok mapy do widoku w scenie kontrolera widoku dla ekranu Mapa. Wykonaj następujące kroki:

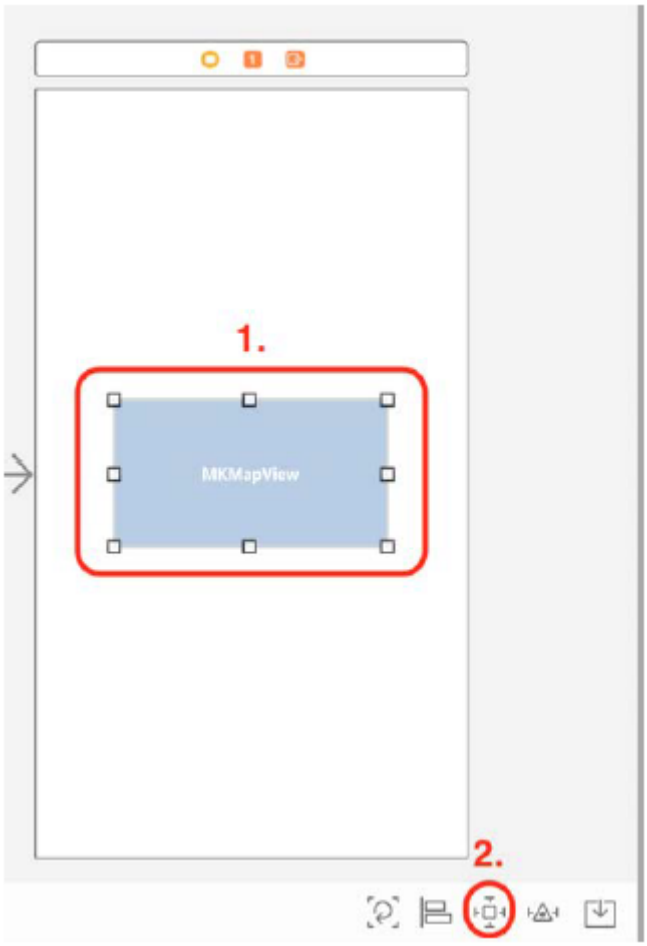
1. Wybierz scenę kontrolera widoku dla ekranu Mapa:



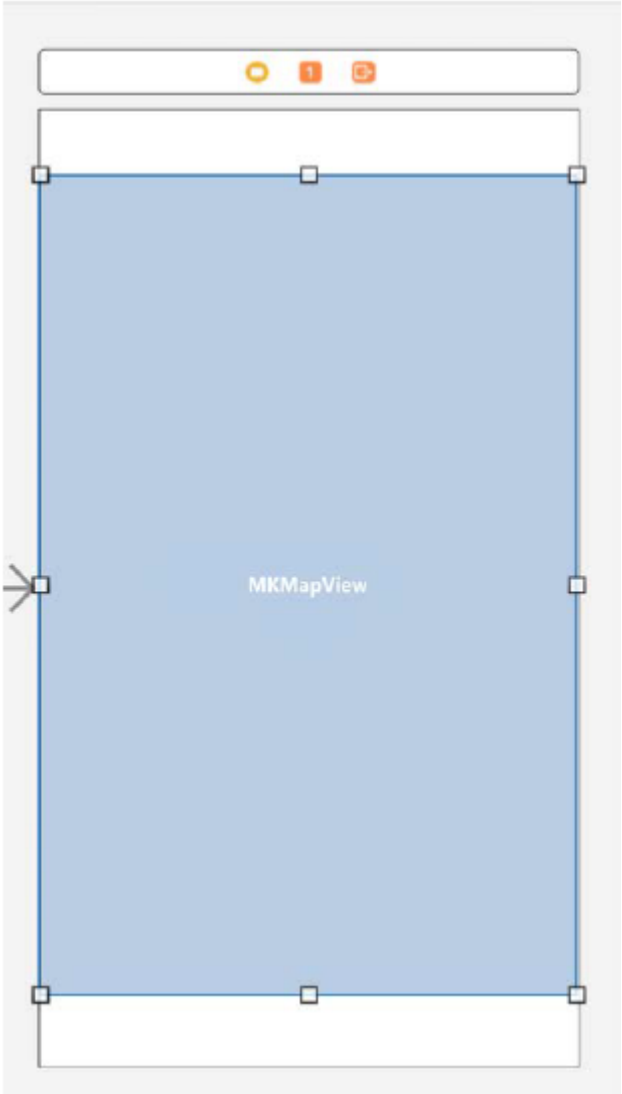
2. Aby w tej scenie była wyświetlana mapa, kliknij przycisk Biblioteka i wpisz mapa w polu filtru. Obiekt widoku zestawu map pojawi się jako jeden z wyników. Przeciągnij go do widoku w scenie kontrolera widoku:



3. Mapa powinna wypełniać cały ekran. Po wybraniu widoku mapy kliknij przycisk Dodaj nowe ograniczenia:



4. Wpisz 0 we wszystkich polach Odstępy do najbliższego sąsiada i upewnij się, że zaznaczone są bladoczerwone rozpórki (staną się jaskrawoczerwone). Kliknij przycisk Dodaj 4 wiązania. Sprawdź, czy widok mapy wypełnia cały ekran:



Kompiluj i uruchamiaj swoją aplikację. Kliknij przycisk Mapa. Powinieneś zobaczyć mapę podobną do pokazanej na rysunku:

Jeśli spojrzysz w tym momencie na główny plik scenorysu, powinieneś zobaczyć coś takiego:

Sprawdź, czy masz wszystkie sceny pokazane na poprzednim zrzucie ekranu, a także możesz uruchomić aplikację w symulatorze, aby sprawdzić, czy wszystkie ekrany działają poprawnie. Wspaniały! Ukończyłeś podstawowy interfejs użytkownika swojej aplikacji!

Streszczenie

Uukończyłeś podstawową strukturę swojej aplikacji. Najpierw dodano pusty widok tabeli do ekranu Lokalizacje. Dodałeś także nową scenę kontrolera widoku do scenorysu, która będzie reprezentować ekran Lista restauracji, dodałeś i skonfigurowałeś widok kolekcji dla tego ekranu oraz zaimplementowałeś przejście, które wyświetla ją po dotknięciu komórki na ekranie Eksploruj. Dodano nową scenę kontrolera widoku tabeli reprezentującą ekran szczegółów restauracji, skonfigurowano widok tabeli ze statycznymi komórkami dla tego ekranu i zaimplementowano przejście, które wyświetli ten ekran po dotknięciu komórki na ekranie Lista restauracji. Dodałeś także przycisk do jednego z wierszy na ekranie Szczegóły restauracji, dodałeś scenę kontrolera widoku tabeli reprezentującą ekran formularza recenzji i skonfigurowałeś dodany przycisk tak, aby go wyświetlał. Na koniec dodano widok mapy do sceny kontrolera widoku na ekranie Mapa i teraz wyświetla ona mapę po dotknięciu przycisku Mapa. Pomyślnie zaimplementowałeś wszystkie ekrany wymagane dla Twojej aplikacji i będziesz mógł przetestować przepływ aplikacji po uruchomieniu jej w symulatorze. Powinieneś także lepiej posługiwać się Konstruktorem interfejsów. Teraz wiesz, jak dodać i skonfigurować widok tabeli do sceny scenorysu, jak dodać przejścia między scenami i jak dodać widok mapy do sceny. Będzie to przydatne podczas wdrażania własnych aplikacji zawierających widoki tabel, używania przejść do nawigacji między różnymi ekranami i wyświetlania map. Świetnie! W następnej części zmodyfikujesz komórki na ekranie Eksploruj, ekranie Lista restauracji i ekranie Lokalizacje, tak aby odpowiadały projektom pokazanym w przewodniku po aplikacji.

Modyfikowanie i konfigurowanie komórek

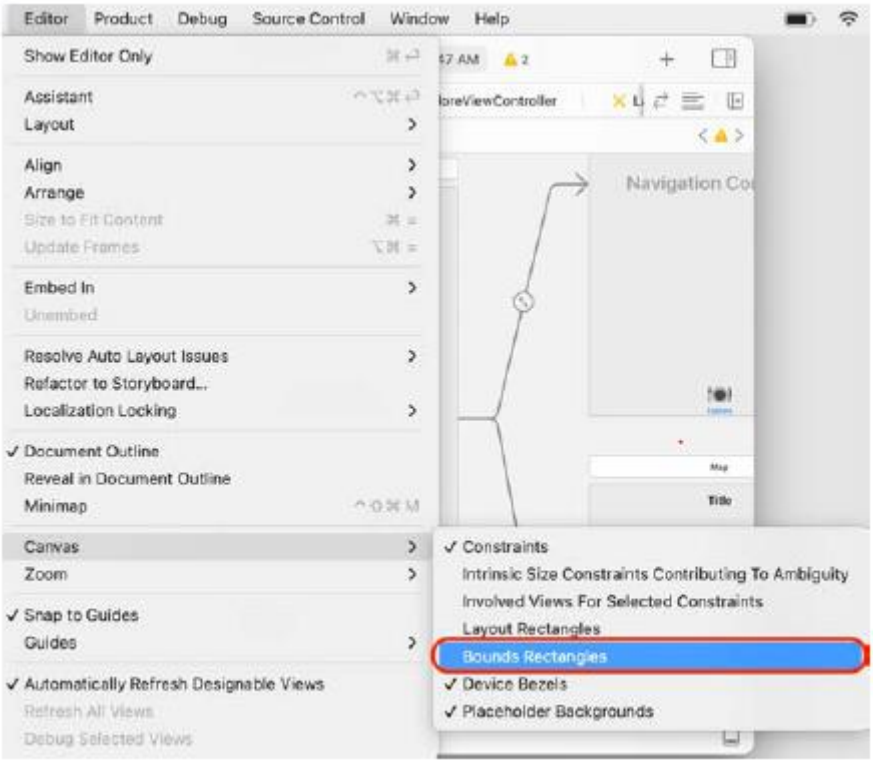
Zaimplementowałeś wszystkie ekrany wymagane dla Twojej aplikacji, ale komórki na ekranach Eksploruj, Lista restauracji i Lokalizacje nadal wymagają pracy. Na przykład nagłówek sekcji widoku kolekcji i komórki widoku kolekcji na ekranie Eksploruj nie odpowiadają wyglądowi pokazanemu w przewodniku po aplikacji . Tu zmodyfikujesz i skonfigurujesz ekrany Eksploruj, Lista restauracji i Lokalizacje, aby dopasować je do wyglądu pokazanego w przewodniku po aplikacji. Na ekranie Eksploruj dodasz etykiety i widoki do nagłówka sekcji widoku kolekcji oraz skonfigurujesz wygląd przycisku. Zmodyfikujesz także komórkę widoku kolekcji exploreCell, dodając do niej widok obrazu i etykietę. Na ekranie Lista restauracji zmodyfikujesz komórkę widoku kolekcji RestaurantCell, dodając do niej etykiety, przyciski i widok obrazu. Skonfigurujesz także widok obrazu, aby wyświetlał obraz domyślny. Na ekranie Lokalizacje skonfigurujesz prototypową komórkę dla widoku tabeli i ustawisz identyfikator LocationCell dla komórek widoku tabeli. Pod koniec tego rozdziału będziesz biegły w dodawaniu i ustawianiu elementów interfejsu użytkownika oraz będziesz wiedział, jak używać ograniczeń do określenia ich wzajemnego położenia.

Modyfikowanie nagłówka sekcji ekranu Eksploruj

Zobaczmy, jak wygląda nagłówek sekcji widoku kolekcji na ekranie Eksploruj podczas prezentacji aplikacji:

W nagłówku sekcji widoku kolekcji znajdują się cztery elementy: dwie etykiety (tytuł i podtytuł), przycisk i widok (szara linia pod tytułem i przyciskiem). Dodałeś już przycisk do nagłówka sekcji widoku kolekcji w widoku kolekcji na ekranie Eksploruj. Teraz dodasz etykiety i widoki, a następnie zmodyfikujesz wszystkie elementy, aby dopasować je do nagłówka sekcji widoku kolekcji pokazanego w prezentacji aplikacji. Wykonaj następujące kroki:

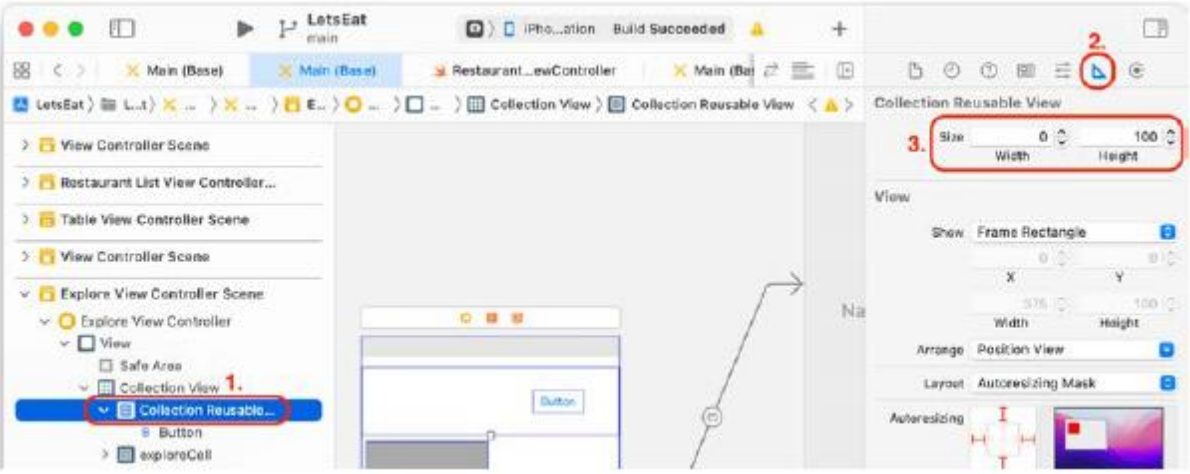
1. Najpierw włącz prostokąty ograniczające w obszarze Edytor. Spowoduje to podświetlenie granic elementów interfejsu użytkownika na niebiesko i ułatwienie ich zobaczenia. Wybierz Płótno | Granice prostokątów z menu Edytor, aby je włączyć:



2. Sprawdź, czy rozmiar nagłówka sekcji widoku kolekcji został ustawiony prawidłowo. W głównym pliku scenorysu znajdź Eksploruj scenę kontrolera widoku. Wybierz opcję Widok kolekcji do ponownego wykorzystania w konspekcie dokumentu (pamiętaj, że jest to nagłówek sekcji widoku kolekcji). Kliknij przycisk Inspektora rozmiaru. W razie potrzeby zaktualizuj wartości w sekcji Rozmiar:

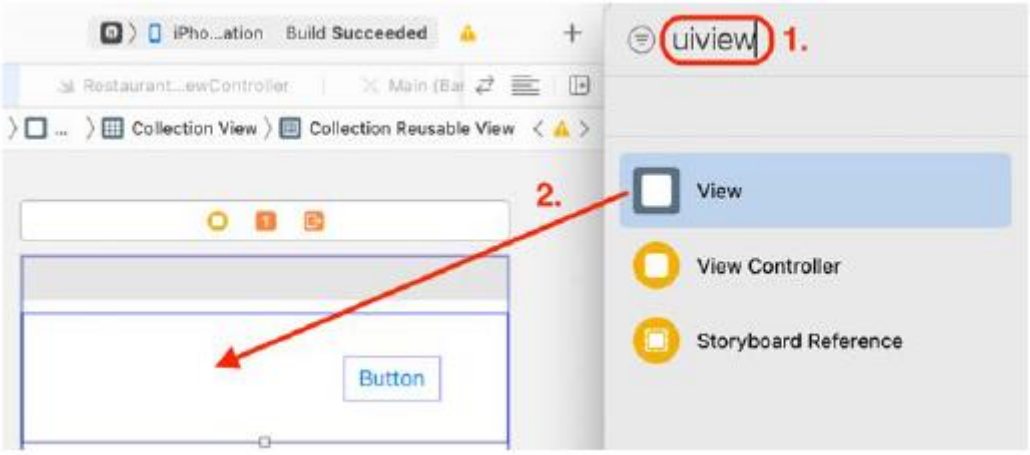
Szerokość: 0

Wzrost: 100

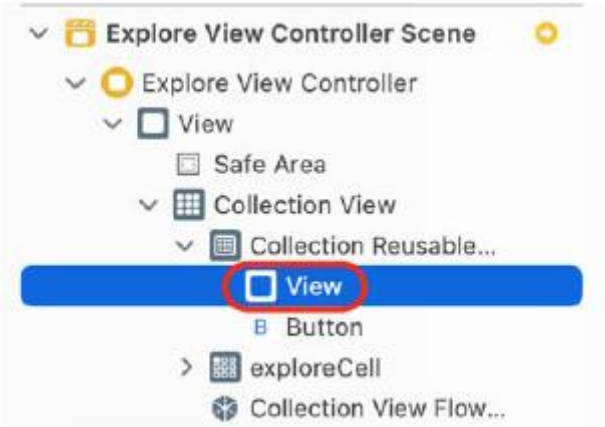


Pamiętaj, że używanymi jednostkami są punkty. Ustawienie szerokości nagłówka sekcji widoku kolekcji na 0 automatycznie sprawi, że będzie on miał taką samą szerokość jak ekran.

3. Do nagłówka sekcji widoku kolekcji dodasz widok, który będzie pełnił rolę kontenera dla wszystkich innych dodanych elementów interfejsu użytkownika. Kliknij przycisk Biblioteka. Wpisz uiview w polu filtra. W wynikach pojawi się obiekt View. Przeciągnij go do nagłówka sekcji widoku kolekcji:



4. W konspekcie dokumentu przeciągnij widok w górę, aby stał się pierwszą pozycją na liście widoków podrzędnych widoku kolekcji. Pierwsza pozycja na liście widoków podrzędnych zostanie narysowana jako pierwsza na ekranie, co gwarantuje, że nie będzie nachodzić na przycisk:



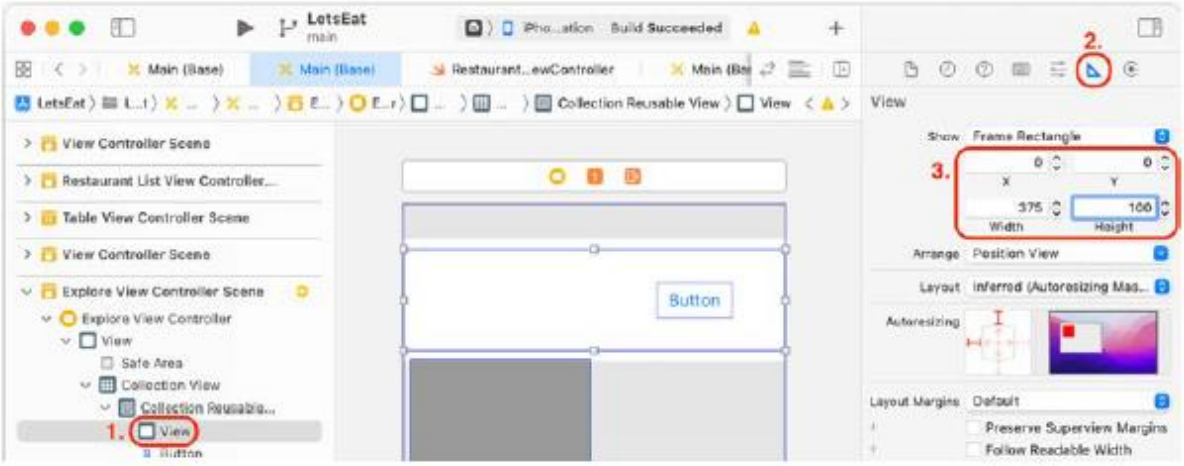
5. Zmień rozmiar widoku na nagłówek sekcji widoku kolekcji. Po wybraniu widoku w konspekcie dokumentu kliknij przycisk Inspektora rozmiaru. W sekcji Widok zaktualizuj następujące wartości:

X: 0

Y: 0

Szerokość: 375

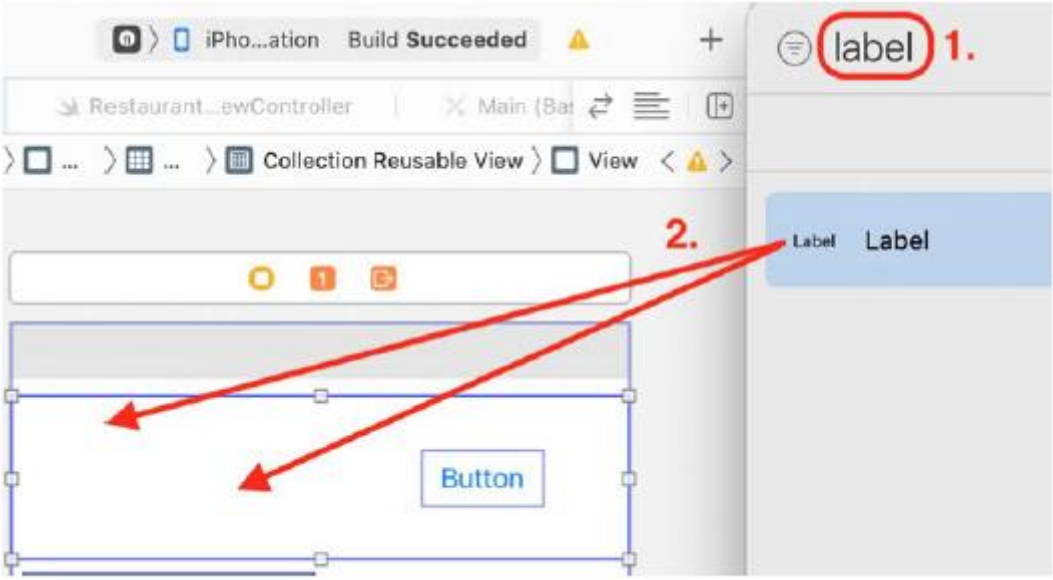
Wzrost: 100



Wartości X i Y określają poziome i pionowe przesunięcie widoku względem lewego górnego rogu nagłówka sekcji widoku kolekcji, a wartości Szerokość i Wysokość określają szerokość i wysokość widoku.

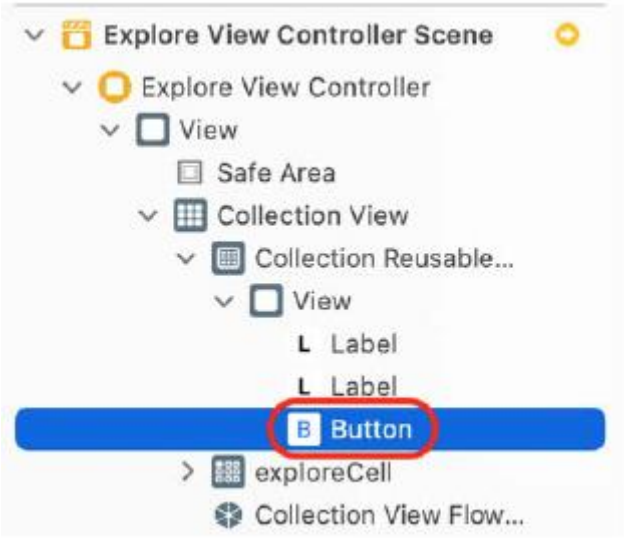
To sprawia, że lewy górny róg tego widoku jest taki sam, jak lewy górny róg nagłówka sekcji widoku kolekcji, ustawia szerokość widoku na tę samą szerokość co ekran (375 punktów) i ustawia wysokość widzenia do 100 punktów. Ułatwi to późniejsze dodawanie ograniczeń, ponieważ dodawane widoki będą ustawiane względem widoku kontenera.

6. Będziesz potrzebować etykiet do wyświetlenia. WYBIERZ LOKALIZACJĘ i ODKRYJ w nagłówku sekcji widoku kolekcji. Kliknij przycisk Biblioteka. Wpisz etykietę w polu filtra. W wynikach pojawi się obiekt Etykieta. Przeciągnij dwa obiekty Label do przeciągniętego wcześniej widoku:

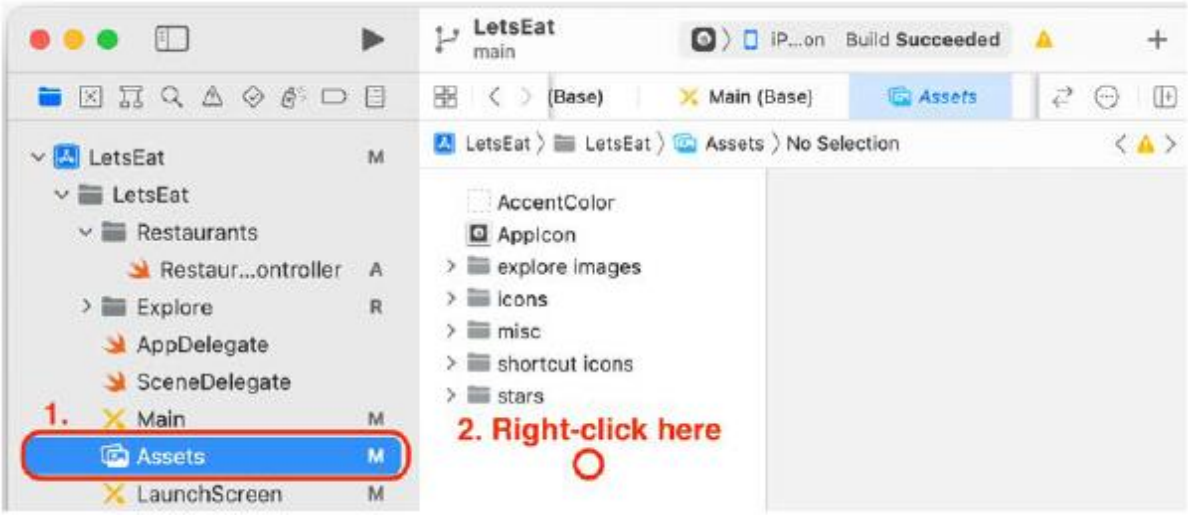


7. Obie etykiety muszą być widokami podrzędnymi widoku kontenera, ponieważ zostaną do nich zastosowane ograniczenia w stosunku do ich pozycji w widoku kontenera. W konspekcie dokumentu sprawdź, czy oba obiekty Label są widokami podrzędnymi widoku, a widok jest widokiem podrzędnym widoku kolekcji do ponownego użycia:

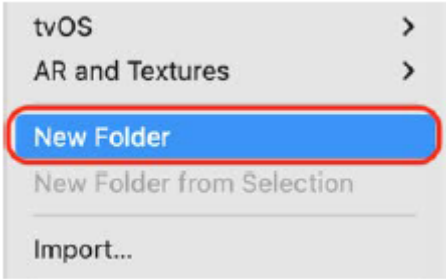
8. Przycisk musi być także widokiem podrzędnym widoku kontenera, ponieważ również zostaną do niego nałożone ograniczenia w stosunku do widoku kontenera. Wybierz przycisk w konspekcie dokumentu i przeciągnij go do widoku, aby uczynić go widokiem podrzędnym widoku. Kiedy już skończysz, powinno to wyglądać tak:



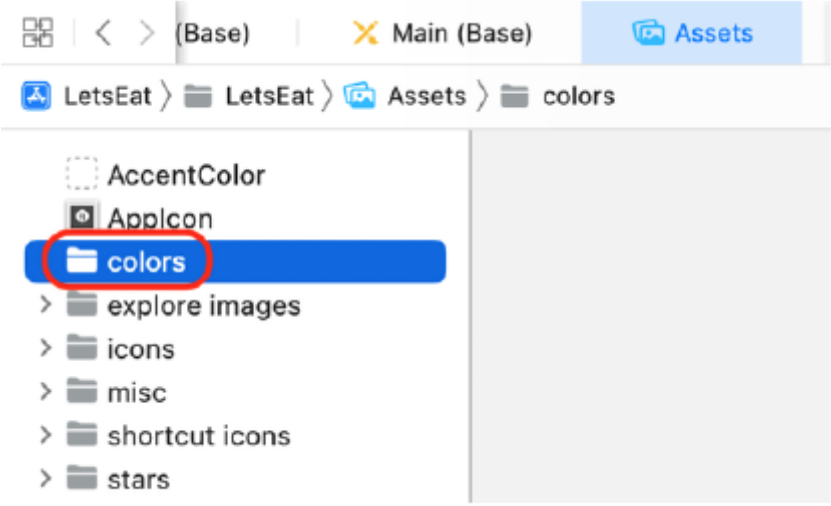
9. Jedna z etykiet w nagłówku sekcji widoku kolekcji powinna być ustawiona w niestandardowym szarym kolorze, aby pasowała do projektu pokazanego w przewodniku po aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika. Utworzysz folder w swoim katalogu zasobów i dodasz do niego nowe niestandardowe kolory. Kliknij plik Assets.xcassets. Kliknij prawym przyciskiem myszy czysty biały obszar konspektu dokumentu, jak pokazano:



10. Wybierz opcję Nowy folder z wyskakującego menu, aby utworzyć nowy folder:

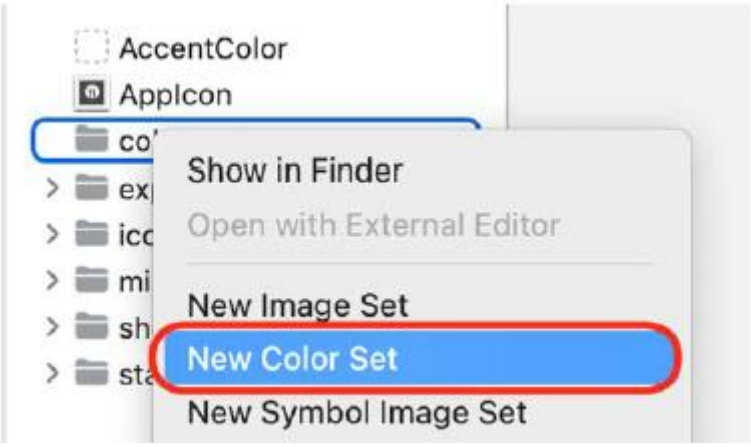


11. Zmień nazwę folderu na kolory:

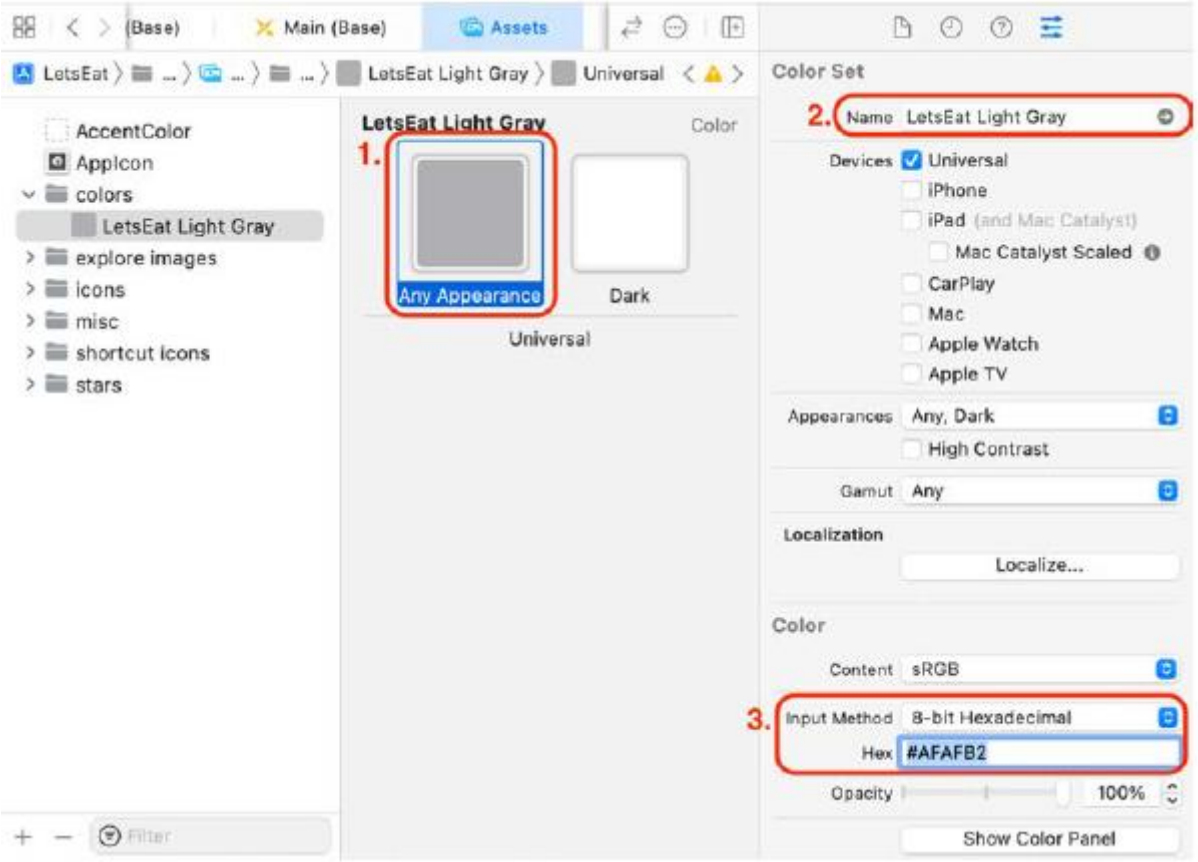


W tym folderze umieścisz wszystkie niestandardowe kolory.

12. Teraz dodasz nowy, niestandardowy kolor do swojego projektu. Kliknij prawym przyciskiem myszy folder kolorów i wybierz Nowy zestaw kolorów:

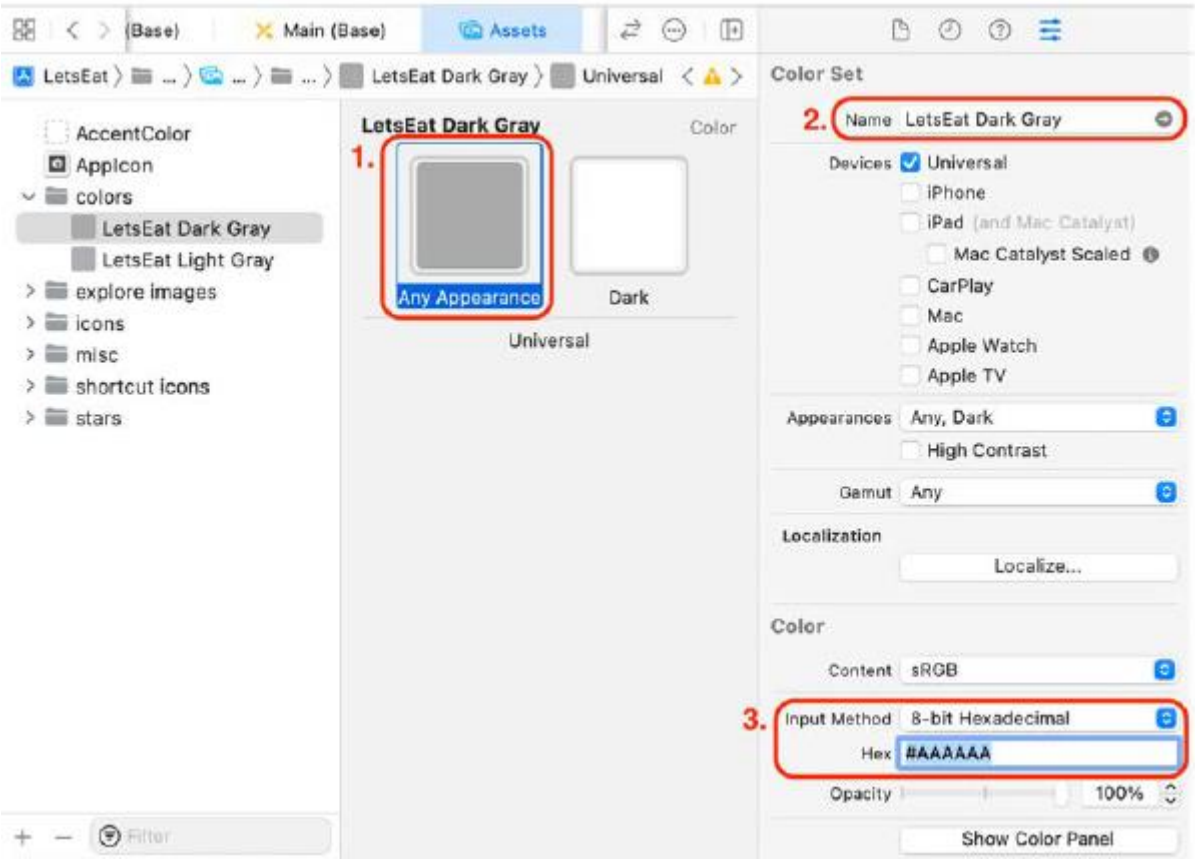


13. Kliknij nowo utworzony zestaw kolorów. Upewnij się, że wybrano opcję Dowolny wygląd. W Inspektorze atrybutów nazwij kolor LetsEat Light Grey, ustaw metodę wprowadzania na 8-bitową wartość szesnastkową i wpisz #AFAFB2 w polu szesnastkowym, naciskając Return, gdy skończysz:

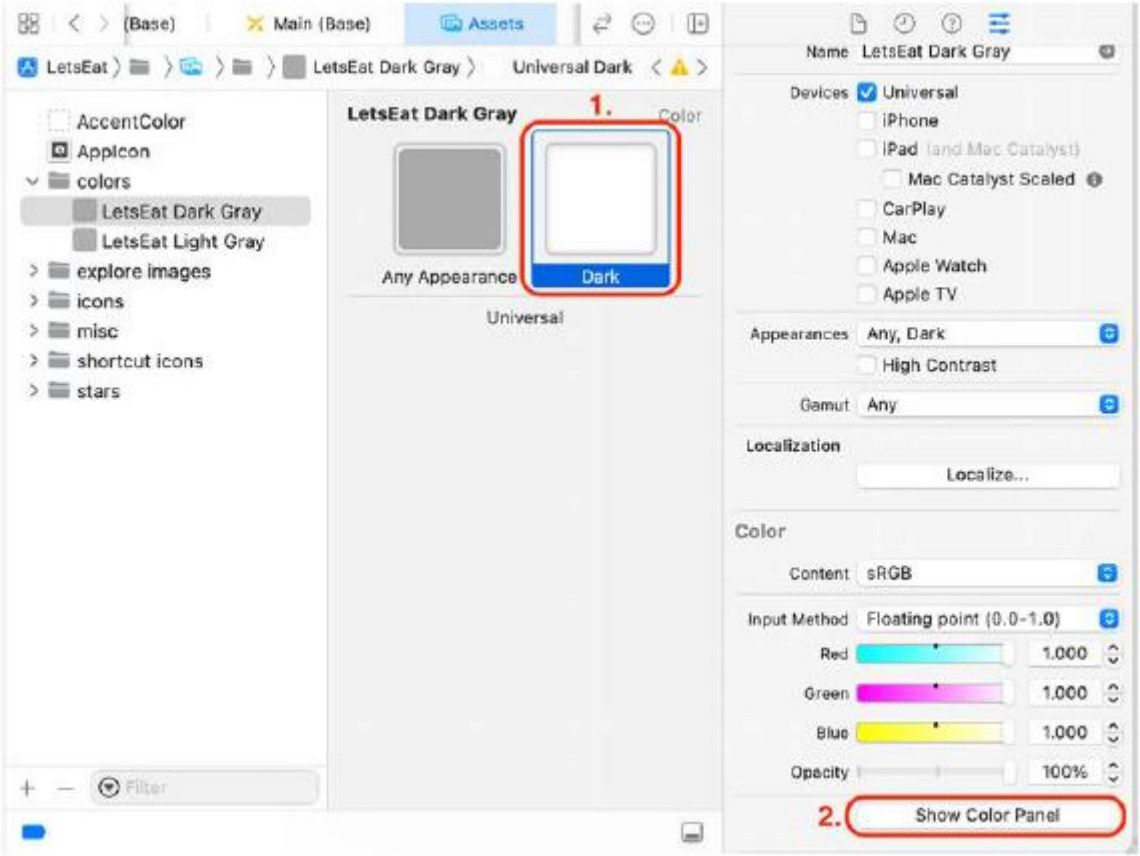


Zwróć uwagę, że obok pola koloru Dowolny wygląd znajduje się pole koloru Ciemny wygląd. Kolor w polu koloru Wygląd ciemny jest używany, jeśli użytkownik włączy tryb ciemny. Pozostaw wartość domyślną.

14. Utworzysz drugi kolor, którego będziesz używać podczas późniejszej modyfikacji ekranu szczegółów restauracji. Powtórz kroki 12 i 13, ale tym razem nazwij kolor LetsEat Dark Grey i ustaw wartość szesnastkową w sekcji Kolor na #AAAAAA:

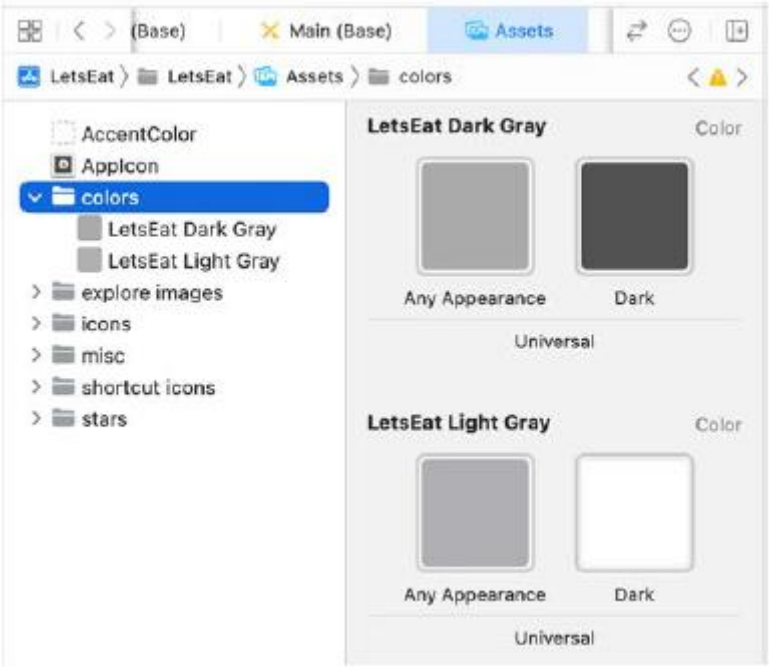


15. W przypadku LetsEat Dark Grey zmienisz kolor Dark Appearance na ciemniejszy szary. Kliknij pole koloru Ciemny wygląd i kliknij przycisk Pokaż panel kolorów:



16. Aby ustawić kolor, wybierz panel suwaków, wybierz opcję Suwak skali szarości z wyskakującego menu i kliknij drugi szary odcień od lewej, jak pokazano:

17. Sprawdź, czy zawartość folderu kolorów jest następująca:

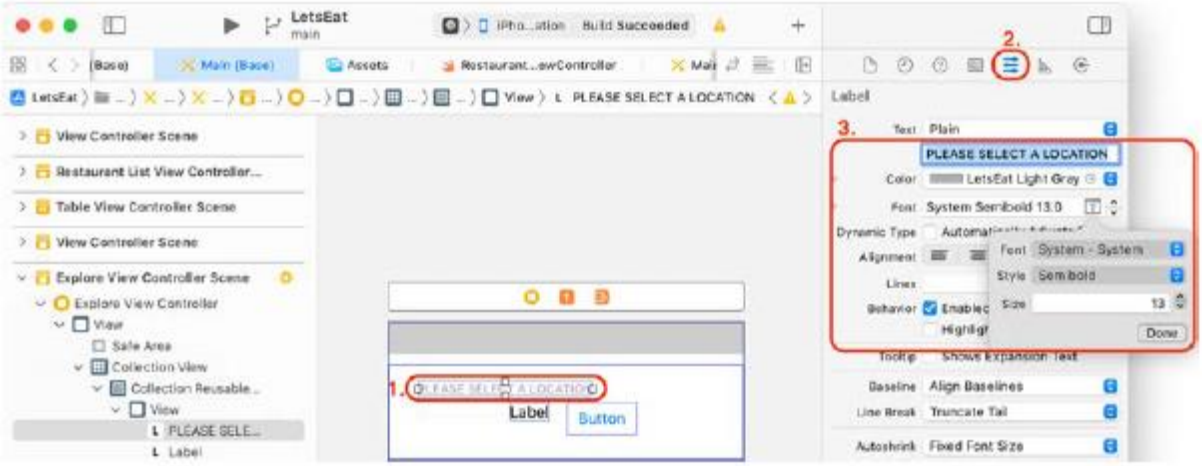


18. Aby skonfigurować etykietę jako podtytuł, kliknij główny plik scenorysu i wybierz jedną z etykiet. Kliknij przycisk Inspektora atrybutów i zaktualizuj następujące wartości:

Tekst: Zwykły, a następnie dodaj. PROSZĘ WYBRAĆ LOKALIZACJĘ w pustym polu tekstowym poniżej

Kolor: LetsEat jasnoszary

Czcionka: System Semibold 13.0



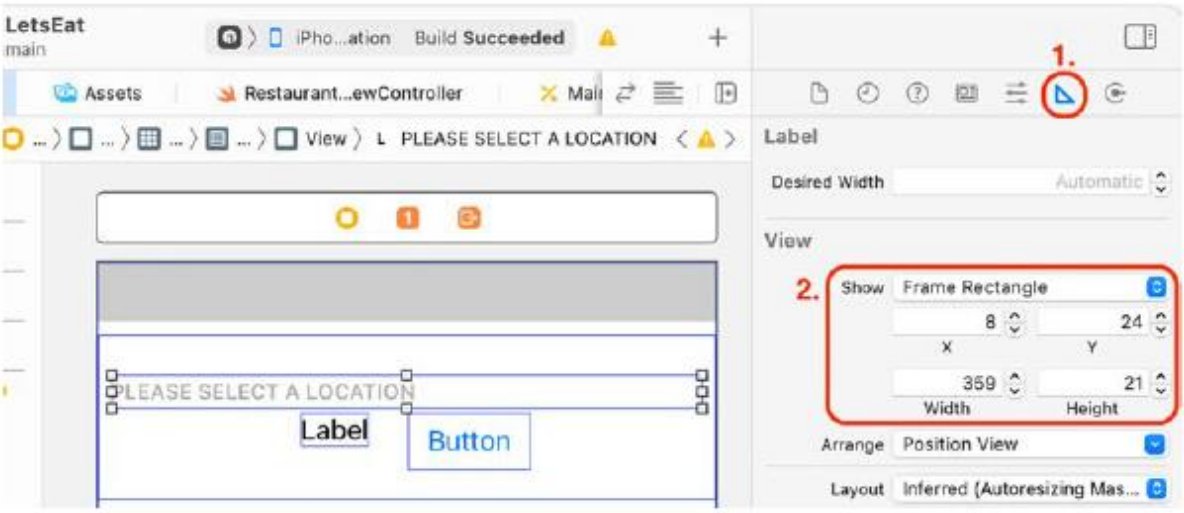
19. Gdy etykieta jest nadal zaznaczona, kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok:

X: 8

Y: 24

Szerokość: 359

Wzrost: 21

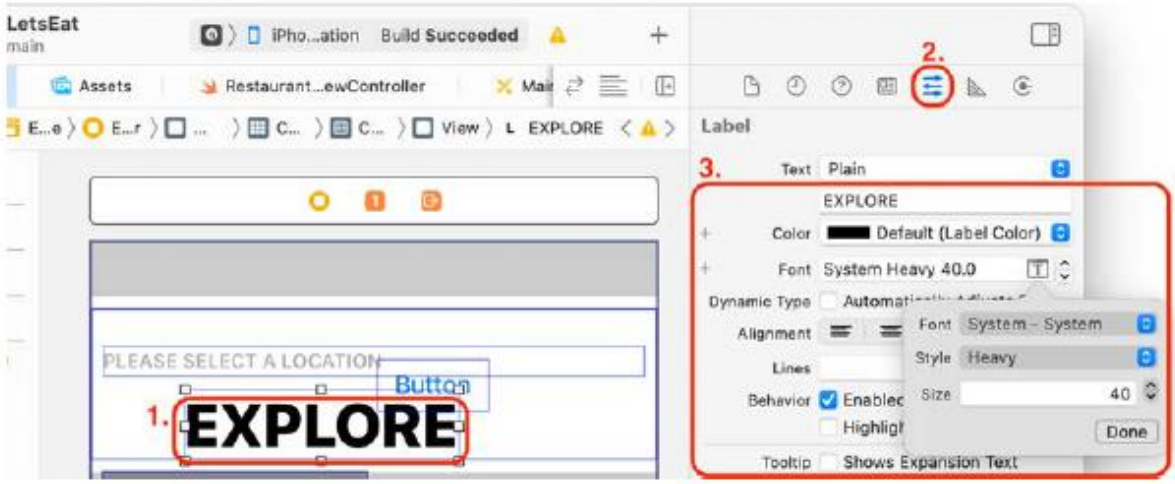


Ta etykieta jest widokiem podrzędnym widoku kontenera dodanego wcześniej do nagłówka sekcji widoku kolekcji. Oznacza to, że położenie etykiety będzie zależne od tego widoku. Lewy górny róg etykiety zostanie przesunięty o 8 punktów w poziomie i 24 punkty w pionie i pojawi się poniżej i na prawo od lewego górnego rogu nagłówka sekcji widoku kolekcji. Szerokość etykiety wyniesie 359 punktów, a wysokość 21 punktów.

20. Aby ustawić drugą etykietę jako tytuł, zaznacz ją i kliknij przycisk Inspektora atrybutów. Zaktualizuj następujące wartości:

Tekst: Zwykły, a następnie dodaj ODKRYJ w pustym polu tekstowym poniżej.

Czcionka: System Heavy 40.0



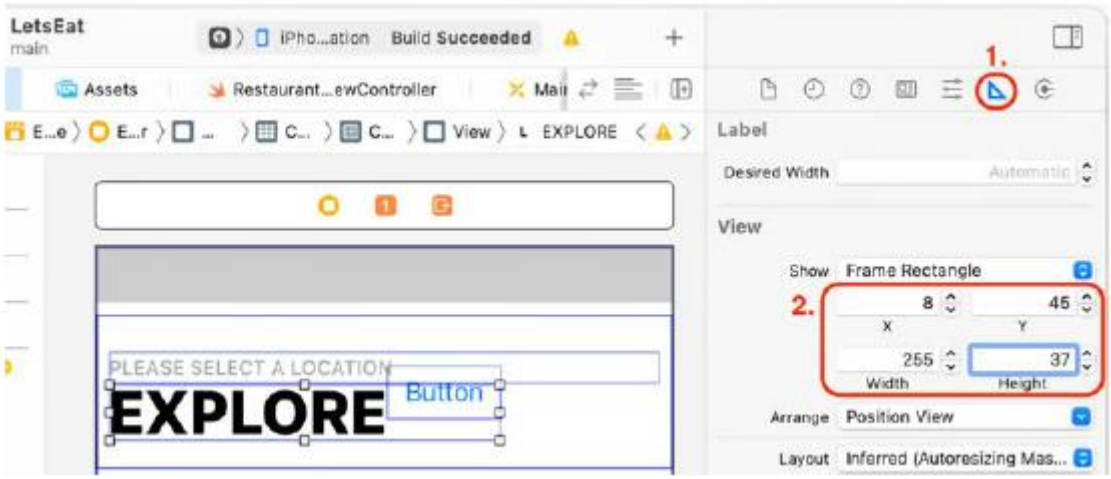
21. Gdy etykieta jest nadal zaznaczona, kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji widoku:

X: 8

Y: 45

Szerokość: 255

Wzrost: 37

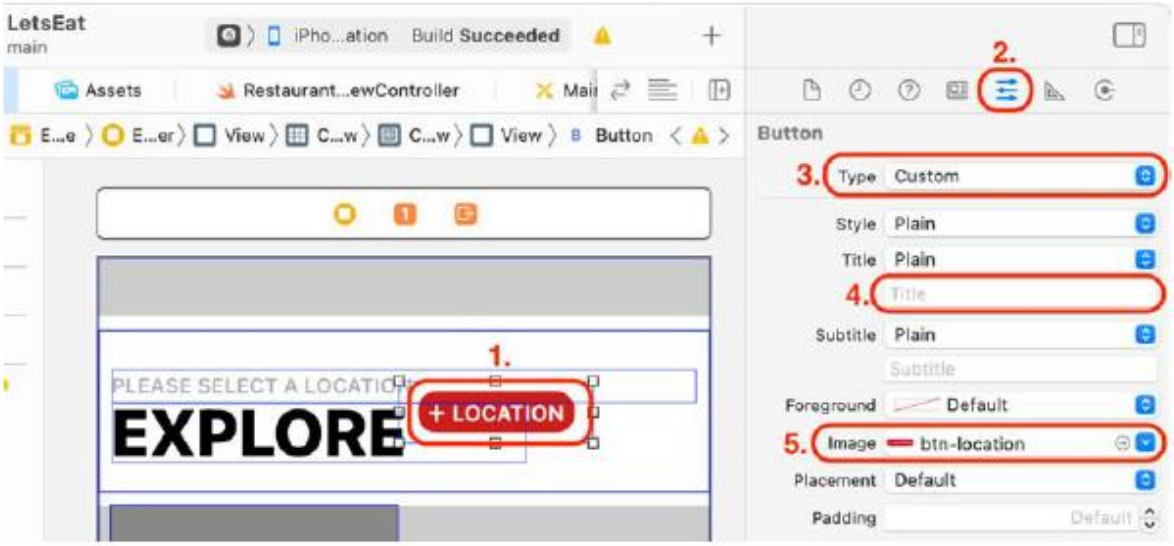


Te ustawienia przesuwają tę etykietę o 8 punktów w prawo i 45 punktów poniżej lewego górnego rogu widoku kontenera, umieszczając ją tuż pod pierwszą etykietą. Należy pamiętać, że nie rozciąga się on całkowicie w prawo, pozostawiając trochę miejsca na przycisk, który zostanie skonfigurowany później.

22. Aby skonfigurować niestandardowy obraz przycisku, zaznacz go i w Inspektorze atrybutów zaktualizuj następujące wartości w sekcji Przycisk:

Wpisz: Niestandardowy i usuń tekst z pola w wyskakującym menu Tytuł

Obraz: btn-location



Ten obraz znajduje się w pliku Assets.xcassets dodanym do aplikacji w rozdziale 10, Konfigurowanie interfejsu użytkownika.

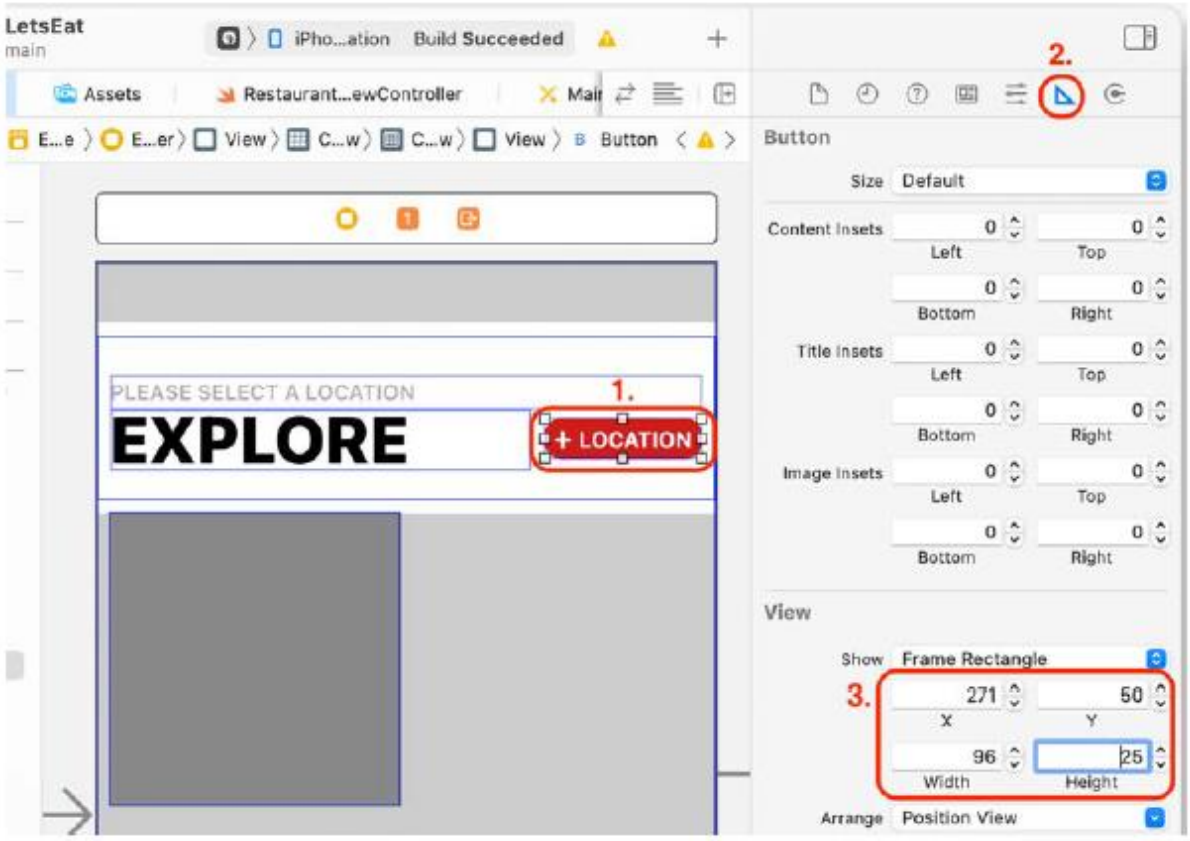
23. Gdy przycisk jest nadal zaznaczony, kliknij przycisk Inspektor rozmiaru. Zaktualizuj następujące wartości w sekcji Widok, aby umieścić przycisk po prawej stronie etykiety ODKRYJ:

X: 271

Y: 50

Szerokość: 96

Wzrost: 25



24. Ostatnią rzeczą, którą dodasz, będzie cienka szara linia u dołu nagłówka sekcji widoku kolekcji. Kliknij przycisk Biblioteka. Wpisz uiview w polu filtra. W wynikach pojawi się obiekt View. Przeciągnij obiekt View do widoku kontenera:

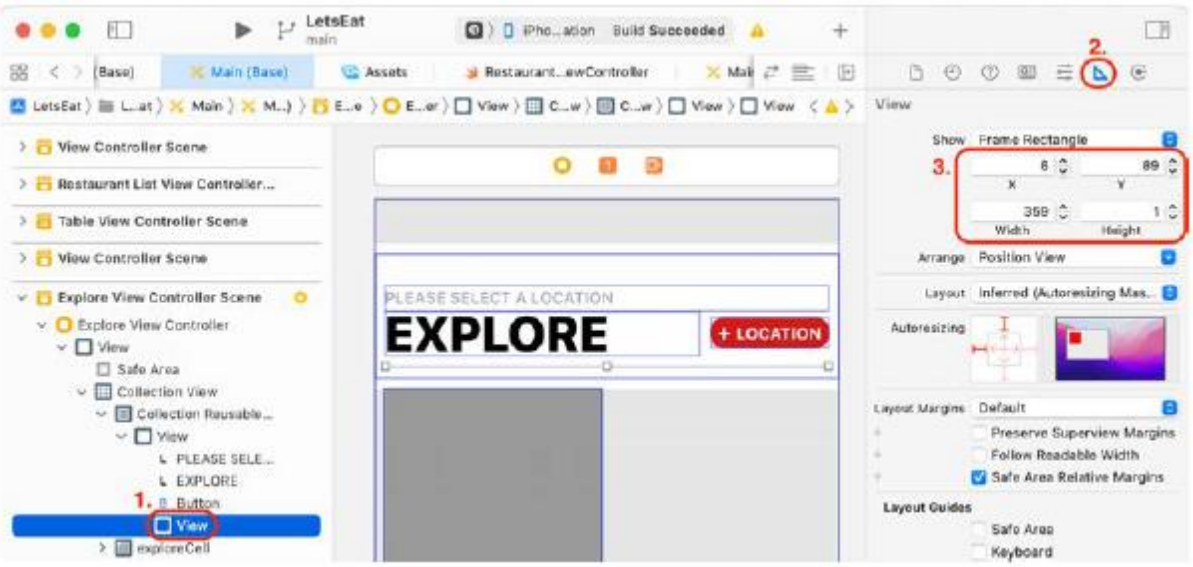
25. Inspektor rozmiaru powinien być nadal wybrany. Aby umieścić nowo dodany widok we właściwym miejscu, zaktualizuj następujące wartości w sekcji Widok:

X: 8

Y: 89

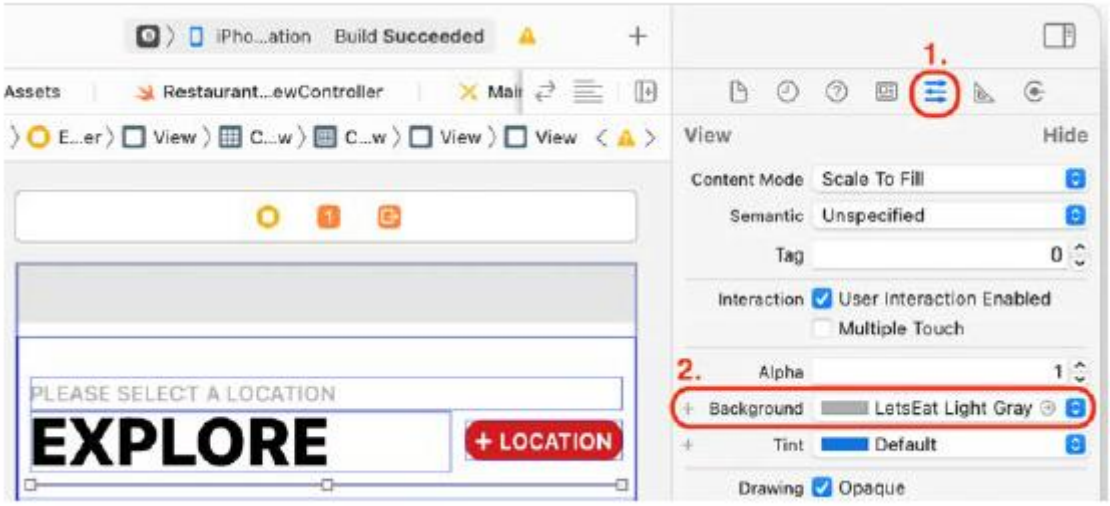
Szerokość: 359

Wysokość: 1

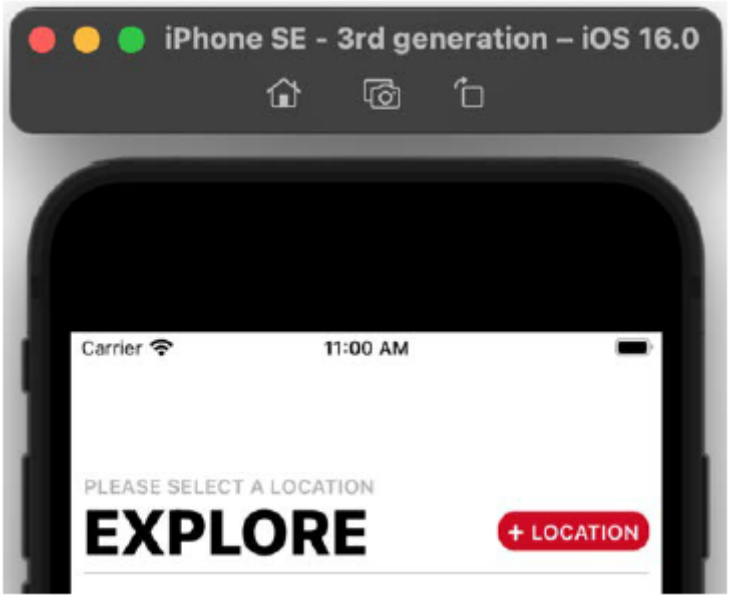


Spowoduje to umieszczenie widoku poniżej wszystkich pozostałych elementów, ale będzie on niewidoczny.

26. Aby ustawić kolor widoku, kliknij przycisk Inspektora atrybutów i ustaw Tło na Jasnoszary LetsEat:



Dodano wszystkie wymagane elementy interfejsu użytkownika. Kompiluj i uruchamiaj swoją aplikację. Nagłówek sekcji widoku kolekcji powinien teraz wyglądać następująco:



Jak widać, nagłówek sekcji widoku kolekcji jest teraz zgodny z projektem pokazanym w prezentacji aplikacji. Działa świetnie przy użyciu symulatora iPhone'a SE (3. generacji), ale aby mieć pewność, że będzie działać na ekranach o innych rozmiarach, dodasz ograniczenia Auto Layout. Zrobisz to w następnej sekcji.

Dodanie układu automatycznego do nagłówka sekcji ekranu Eksploruj

Jeśli zbudujesz i uruchomisz swoją aplikację teraz w symulatorze iPhone'a SE (3. generacji), nagłówek sekcji widoku kolekcji będzie wyglądał świetnie, ale jeśli przełączysz się na symulator z większym ekranem, zobaczysz, że niektóre elementy graficzne nie są prawidłowo rozmieszczone . Jak widzieliście w poprzednich rozdziałach, automatyczny układ zapewnia dostosowanie interfejsu użytkownika do rozmiaru i orientacji ekranu urządzenia. Na przykład logo aplikacji Let’s Eat na ekranie uruchamiania pozostaje dokładnie na środku ekranu, niezależnie od urządzenia, a widok tabeli na ekranie Lokalizacje zajmuje całą dostępną przestrzeń ekranu nawet po obróceniu urządzenia. Do tej pory używałeś ograniczeń automatycznego układu tylko z pojedynczymi elementami interfejsu użytkownika. W tej sekcji dodasz je do wielu elementów interfejsu użytkownika w nagłówku sekcji widoku kolekcji. Zaczniesz od dodania ograniczeń do widoku kontenera, a następnie przystąpisz do dodawania ograniczeń do wszystkich pozostałych elementów w nim zawartych.

Wykonaj następujące kroki:

1. Wybierz widok zawierający inne widoki w konspekcie dokumentu:

2. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia dla tego widoku:

Góra: 0

Po lewej: 0

Po prawej: 0

Wzrost: 90

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Spowoduje to powiązanie górnej, lewej i prawej krawędzi widoku kontenera z krawędziami nagłówka sekcji widoku kolekcji. Wysokość widoku jest ustawiona na 90 punktów, co określa położenie dolnej krawędzi.

3. W konspekcie dokumentu wybierz etykietę PROSZĘ WYBRAĆ LOKALIZACJĘ:

4. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia dla tej etykiety:

U góry: 24

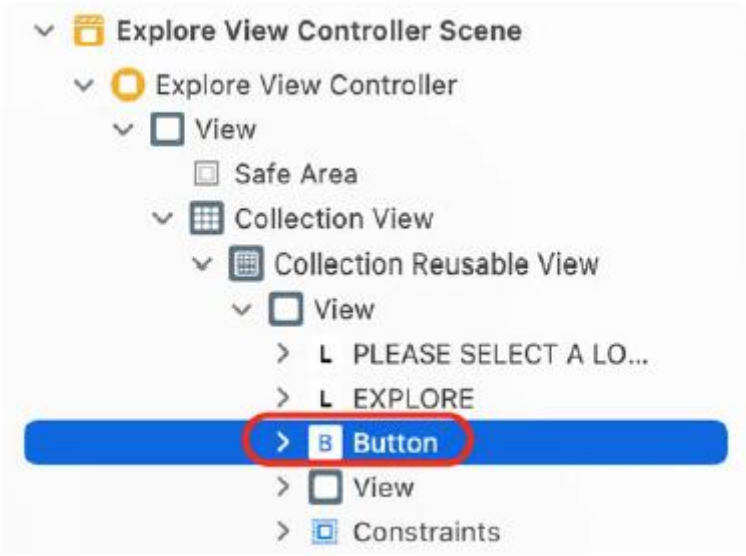
Po lewej: 8

Po prawej: 8

Wzrost: 21

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Odstępy pomiędzy górną, lewą i prawą krawędzią etykiety oraz odpowiadającymi im krawędziami widoku kontenera są ustawione odpowiednio na 24 punkty, 8 punktów i 8 punktów. Zwróć uwagę, że szerokość etykiety nie jest ustawiona, co pozwala na jej zmianę, jeśli uruchomisz ją na symulatorze z mniejszym lub większym ekranem. Tak jak poprzednio, ustawienie ograniczenia wysokości określa położenie dolnej krawędzi etykiety.

5. Wybierz przycisk LOKALIZACJA w konspekcie dokumentu:



6. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia dla tego przycisku:

Top 5

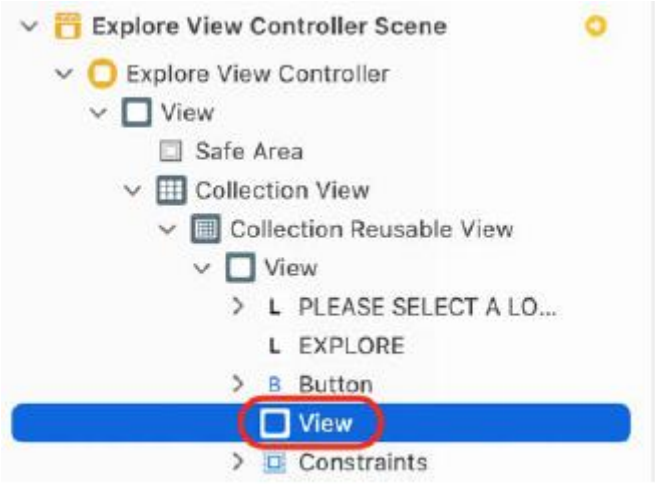
Po prawej: 8

Szerokość: 96

Wzrost: 25

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Ponieważ przycisk LOKALIZACJA znajduje się pod etykietą PROSZĘ WYBRAĆ LOKALIZACJĘ, górne ograniczenie określa odstęp pomiędzy górną krawędzią przycisku LOKALIZACJA a dolną krawędzią etykiety, a nie górną krawędzią widoku kontenera. Odstęp pomiędzy prawą krawędzią przycisku LOKALIZACJA a prawą krawędzią widoku kontenera ustawiony jest na 8 punktów, a ograniczenia szerokości i wysokości określają położenie lewej i dolnej krawędzi przycisku LOKALIZACJA.

7. Wybierz szarą linię Widok w konspekcie dokumentu:



8. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia dla tego widoku:

Po lewej: 8

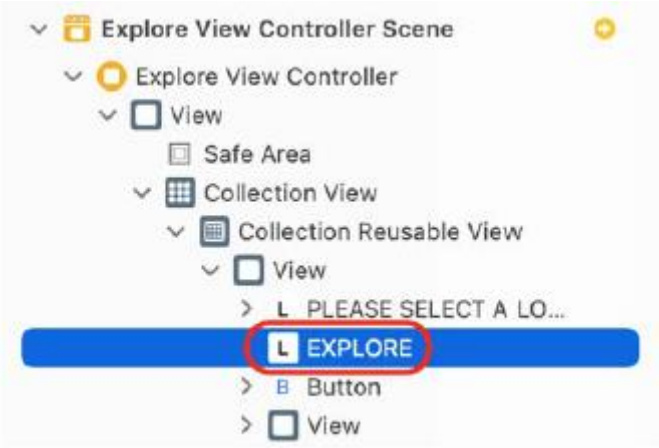
Po prawej: 8

Dół: 0

Wysokość: 1

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Odstęp pomiędzy lewą i prawą krawędzią widoku oraz lewą i prawą krawędzią widoku kontenera wynosi 8 punktów, a dolna krawędź widoku jest powiązana z dolną krawędzią widoku kontenera. Ograniczenie wysokości określa położenie górnej krawędzi widoku.

9. Wybierz etykietę ODKRYJ w konspekcie dokumentu:



10. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia dla tej etykiety:

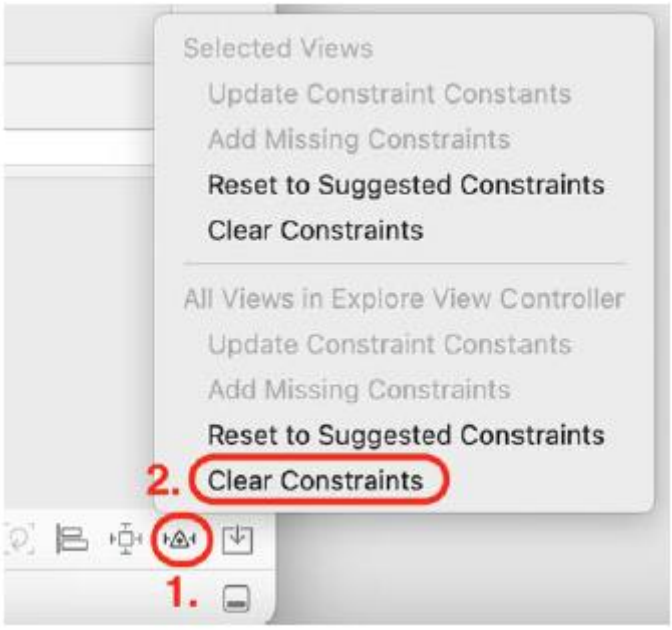
Góra: 0

Po lewej: 8

Po prawej: 8

Wzrost: 37

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Górna krawędź etykiety jest połączona z dolną krawędzią etykiety PROSZĘ WYBRAĆ LOKALIZACJĘ. Odstęp pomiędzy lewą krawędzią etykiety a lewą krawędzią widoku kontenera wynosi 8 punktów. Odstęp pomiędzy prawą krawędzią etykiety a lewą krawędzią przycisku LOKALIZACJA wynosi 8 punktów. Ograniczenie wysokości określa położenie dolnej krawędzi etykiety. Zakończyłeś dodawanie ograniczeń automatycznego układu do wszystkich widoków w nagłówku sekcji widoku kolekcji na ekranie Eksploruj. Spróbuj uruchomić aplikację przy użyciu różnych symulatorów, aby zobaczyć, jak interfejs użytkownika dostosowuje się do różnych rozmiarów ekranu. Fajny! Być może zastanawiasz się, dlaczego przed dodaniem ograniczeń konieczne było ustawienie położenia elementów interfejsu użytkownika za pomocą Inspektora rozmiaru. Właściwie nie jest to konieczne, ale dzięki temu znacznie łatwiej jest dodawać ograniczenia, ponieważ wartości ograniczeń widoczne po kliknięciu przycisku Dodaj nowe ograniczenia pochodzą z bieżącej przestrzeni pomiędzy elementami interfejsu użytkownika. Praca z Auto Layout może być wyzwaniem dla początkujących programistów. Nie spiesz się, robiąc to. Jeśli to nie działa poprawnie, usuń wszystkie ograniczenia i zacznij od nowa. Aby to zrobić, kliknij przycisk Rozwiąż problemy z automatycznym układem u dołu ekranu i wybierz opcję Wyczyść ograniczenia:



Dodałeś wszystkie wymagane elementy i ograniczenia interfejsu użytkownika do nagłówka sekcji widoku kolekcji na ekranie Eksploruj. Po zakończeniu modyfikowania nagłówka sekcji widoku kolekcji zmodyfikujmy komórkę widoku kolekcji exploreCell w następnej sekcji. Dodasz kilka elementów interfejsu użytkownika, aby dopasować je do komórki pokazanej w przewodniku po aplikacji.

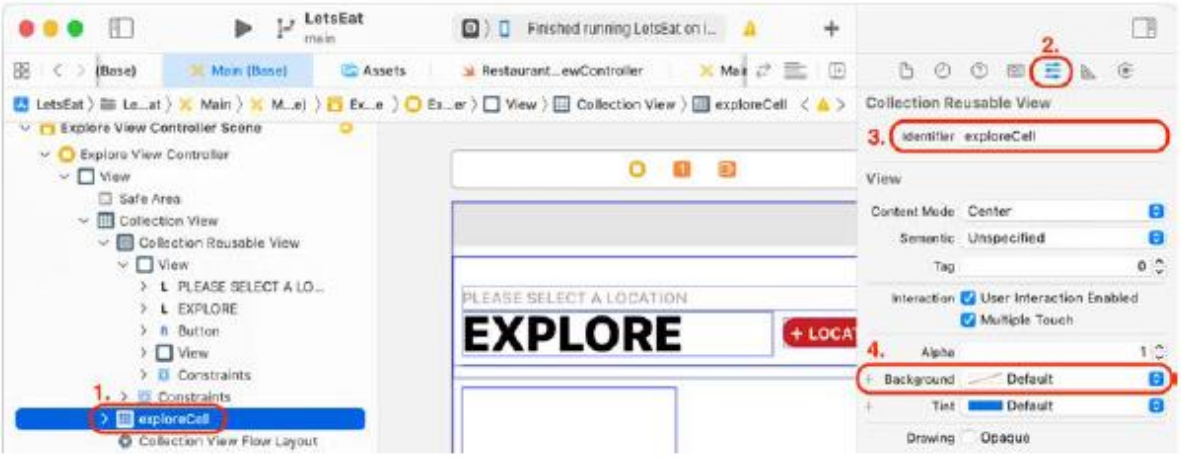
Modyfikowanie komórki widoku kolekcji exploreCell

Przyjrzyjmy się, jak wygląda komórka widoku kolekcji exploreCell podczas prezentacji aplikacji:

Bistro

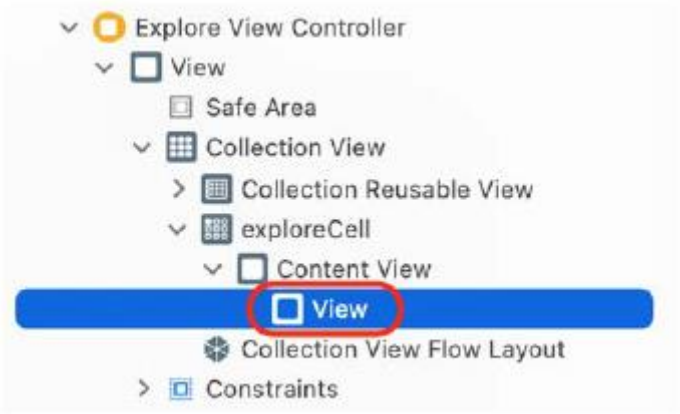
W poprzednim rozdziale ustawiłeś kolor tła komórki widoku kolekcji exploreCell i skonfigurowałeś widok kolekcji tak, aby wyświetlał siatkę 20 komórek. Teraz usuniesz kolor tła i dodasz elementy interfejsu użytkownika do komórki widoku kolekcji exploreCell, aby dopasować je do projektu pokazanego w prezentacji aplikacji. Wykonaj następujące kroki:

1. Zanim zaczniesz, sprawdź ustawienia początkowe komórki widoku kolekcji exploreCell. Wybierz opcję exploreCell w konspekcie dokumentu sceny kontrolera widoku Eksploruj. Kliknij przycisk Inspektora atrybutów. Potwierdź, że identyfikator jest ustawiony na exploreCell. Ustaw tło na domyślne:



2. Dodasz widok kontenera do komórki widoku kolekcji exploreCell. Kliknij przycisk Biblioteka. Wpisz uiview w polu filtra. W wynikach pojawi się obiekt View. Przeciągnij go do komórki prototypowej:

3. Aby upewnić się, że ograniczenia nowo dodanego widoku mogą być ustawione prawidłowo, sprawdź, czy właśnie dodany widok jest widokiem podrzędnym widoku zawartości exploreCell i jest wybrany:



4. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić wiązania dla nowo dodanego widoku:

Góra: 0

Po lewej: 0

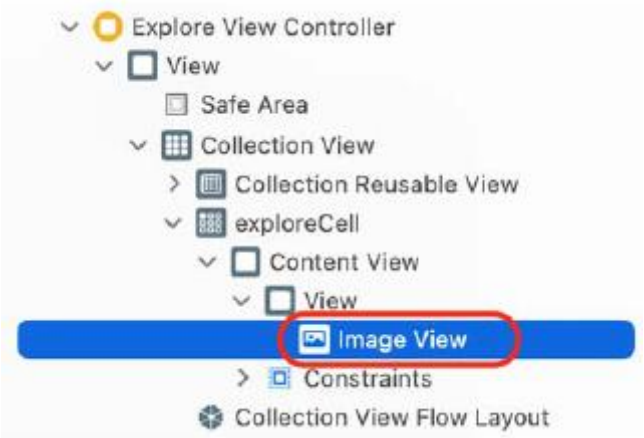
Po prawej: 0

Dół: 40

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Spowoduje to powiązanie górnej, lewej i prawej krawędzi widoku z odpowiednimi krawędziami komórki widoku kolekcji ExploreCell. Położenie dolnej krawędzi jest określane przez ograniczenie dolne, które określa odległość między dolną krawędzią widoku a dolną krawędzią komórki widoku kolekcji exploreCell. Później dodasz etykietę w tym miejscu.

5. Dodasz widok obrazu, aby wyświetlić zdjęcie kuchni. Kliknij przycisk Biblioteka. Wpisz obraz w polu filtra. W wynikach pojawi się obiekt Image View. Przeciągnij go na widok dodany wcześniej:

6. Aby zapewnić prawidłowe ustawienie ograniczeń widoku obrazu, sprawdź, czy widok obrazu jest widokiem dodanym wcześniej i czy jest wybrany:



7. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, dla których chcesz ustawić ograniczenia

widok obrazu:

Góra: 0

Po lewej: 0

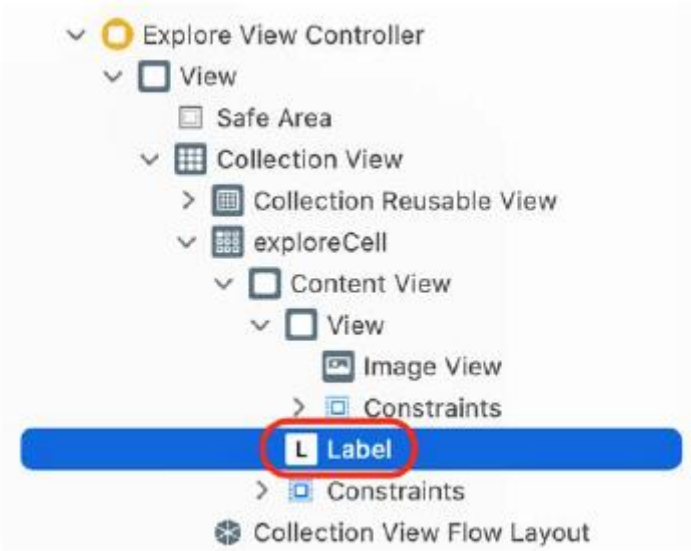
Po prawej: 0

Dół: 0

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Spowoduje to powiązanie krawędzi widoku obrazu z krawędziami widoku dodanego wcześniej.

8. Dodasz etykietę wyświetlającą rodzaj kuchni. Kliknij przycisk Biblioteka. Wpisz etykietę w polu filtra. W wynikach pojawi się obiekt Etykieta. Przeciągnij go w miejsce pomiędzy właśnie dodanym widokiem obrazu a dolną krawędzią komórki:

9. Aby upewnić się, że ograniczenia etykiety można ustawić prawidłowo, sprawdź, czy etykieta jest zaznaczona i czy jest widokiem podrzędnym widoku zawartości komórki widoku kolekcji exploreCell, a nie widokiem podrzędnym widoku dodanego wcześniej:



10. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości, aby ustawić ograniczenia

dla etykiety:

Góra: 9

Po lewej: 8

Po prawej: 8

Wzrost: 21

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Odstęp pomiędzy górną krawędzią etykiety a dolną krawędzią dodanego wcześniej widoku jest ustawiony na 9 punktów. Odstęp pomiędzy lewą i prawą krawędzią etykiety oraz odpowiadającymi mu krawędziami widoku zawartości exploreCell jest ustawiony na 8 punktów. Ograniczenie wysokości określa położenie dolnej krawędzi etykiety poprzez

ustawienie odstępu pomiędzy górną i dolną krawędzią etykiety. Dodano wszystkie niezbędne ograniczenia. Zbuduj i uruchom swoją aplikację:

Jak widać, ekran Eksploruj jest teraz bardziej zgodny z wyglądem pokazanym w prezentacji aplikacji. Każda komórka ma teraz widok obrazu i etykietę tuż pod nią, a także dodano wszystkie niezbędne ograniczenia. Fantastyczny! Zwróć uwagę, że w przeciwieństwie do poprzedniej sekcji, przed dodaniem wiązań nie ustawiłeś położenia elementów interfejsu użytkownika za pomocą Inspektora rozmiaru. Można to zrobić, ponieważ dodajesz tylko kilka elementów do komórki, a względne położenie każdego elementu względem siebie jest jednoznaczne. Dodałeś wszystkie wymagane elementy interfejsu użytkownika i ograniczenia do komórki widoku kolekcji exploreCell. Po zakończeniu modyfikowania komórki widoku kolekcji exploreCell zmodyfikujmy następnie komórkę widoku kolekcji RestaurantCell, dodając do niej pewne elementy interfejsu użytkownika w następnej sekcji.

Modyfikowanie komórki widoku kolekcji RestaurantCell

Przyjrzyjmy się, jak wygląda komórka widoku kolekcji RestaurantCell podczas prezentacji aplikacji:

Jak widać, komórka widoku kolekcji RestaurantCell składa się z wielu elementów. Teraz zmodyfikujesz go, aby pasował do projektu pokazanego w prezentacji aplikacji. Podsumowanie wymaganych zmian jest następujące:

• Zmień rozmiar komórki widoku kolekcji RestaurantCell, aby ją powiększyć i zmień kolor tła na domyślny.

• Dodaj widok, następnie dodaj etykietę i widok stosu zawierający trzy przyciski pokazujące dostępne terminy rezerwacji.

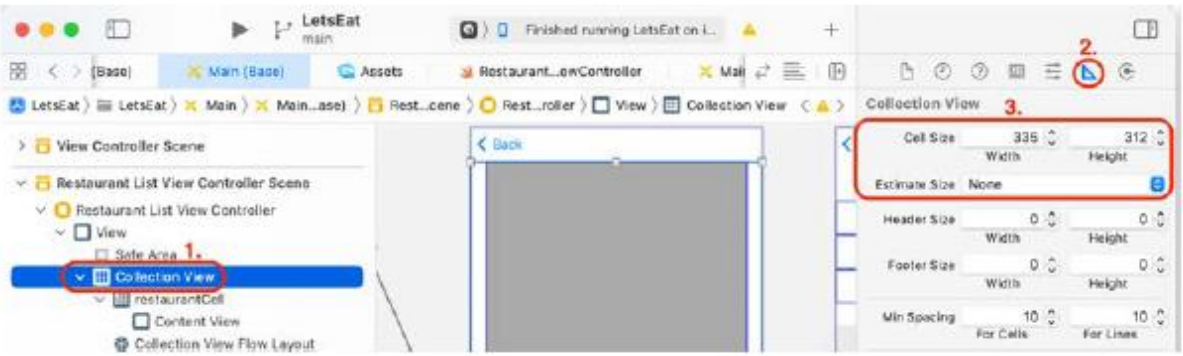
• Dodaj widok, a następnie dodaj widok obrazu, aby pokazać zdjęcie restauracji.

• Dodaj etykietę w lewym górnym rogu, aby pokazać nazwę restauracji.

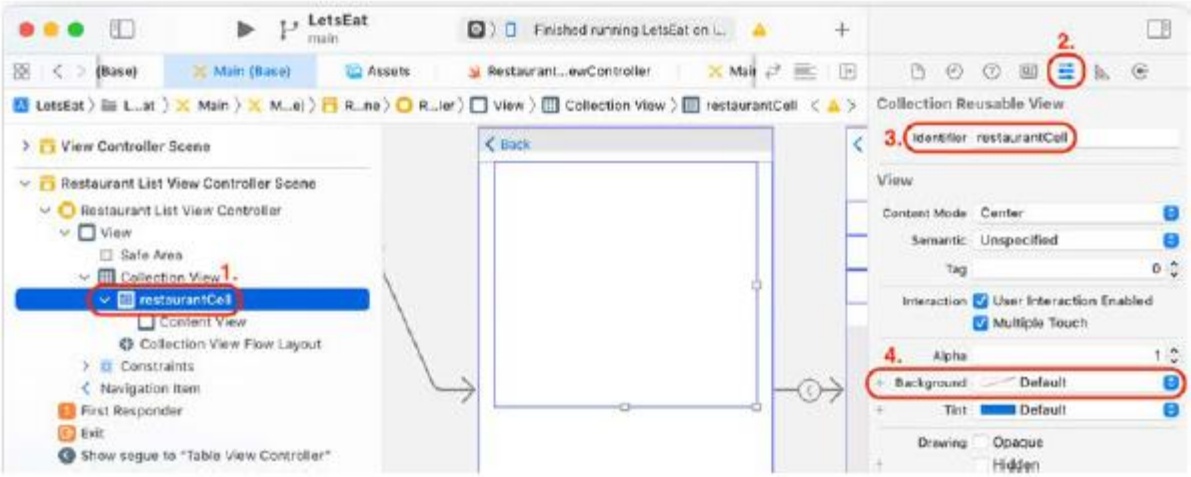
• Dodaj etykietę tuż pod etykietą z nazwą, aby pokazać kuchnię oferowaną przez restaurację.

Do rozmieszczenia wszystkich elementów będziesz używać Inspektora rozmiaru, co ułatwi późniejsze dodanie niezbędnych wiązań układu automatycznego. Nie spiesz się, aby to zrobić, aby zmniejszyć ryzyko popełnienia błędu. Wykonaj następujące kroki:

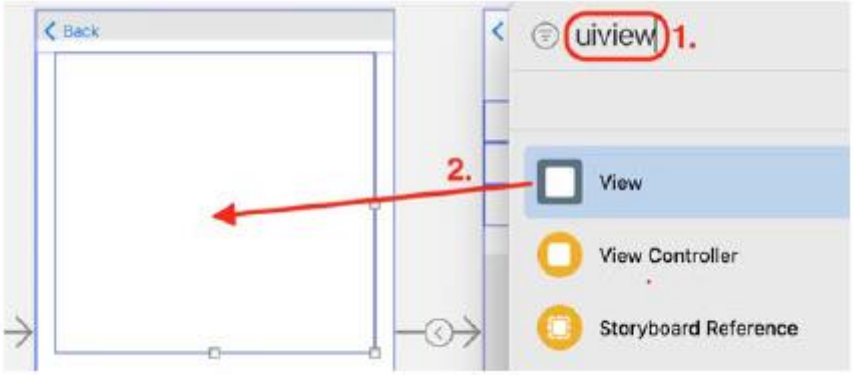
1. Zaczniesz od ustawienia rozmiaru komórki widoku kolekcji RestaurantCell. W głównym pliku scenorysu kliknij widok kolekcji dla sceny kontrolera widoku listy restauracji w konspekcie dokumentu. Kliknij przycisk Inspektora rozmiaru. W obszarze Rozmiar komórki ustaw Szerokość na 335 i Wysokość na 312. Ustaw Szacowany rozmiar na Brak:



2. Aby sprawdzić identyfikator i kolor tła komórki widoku kolekcji RestaurantCell, kliknij restauracjęCell w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. Upewnij się, że identyfikator jest ustawiony na RestaurantCell. Ustaw tło na domyślne:



3. Dodasz widok kontenera dla etykiety Dostępne terminy i przycisków rezerwacji. Kliknij przycisk Biblioteka. Wpisz uiview w polu filtra. W wynikach pojawi się obiekt View. Przeciągnij go do komórki prototypowej:



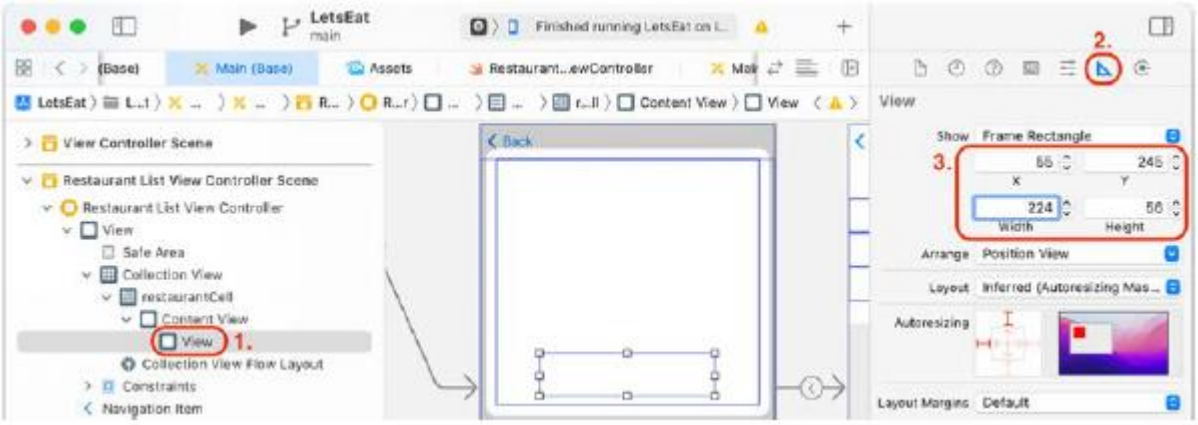
4. Po wybraniu nowo dodanego widoku kliknij przycisk Inspektor rozmiaru. Zaktualizuj następujące wartości w sekcji Widok, aby umieścić ją w komórce:

X: 55

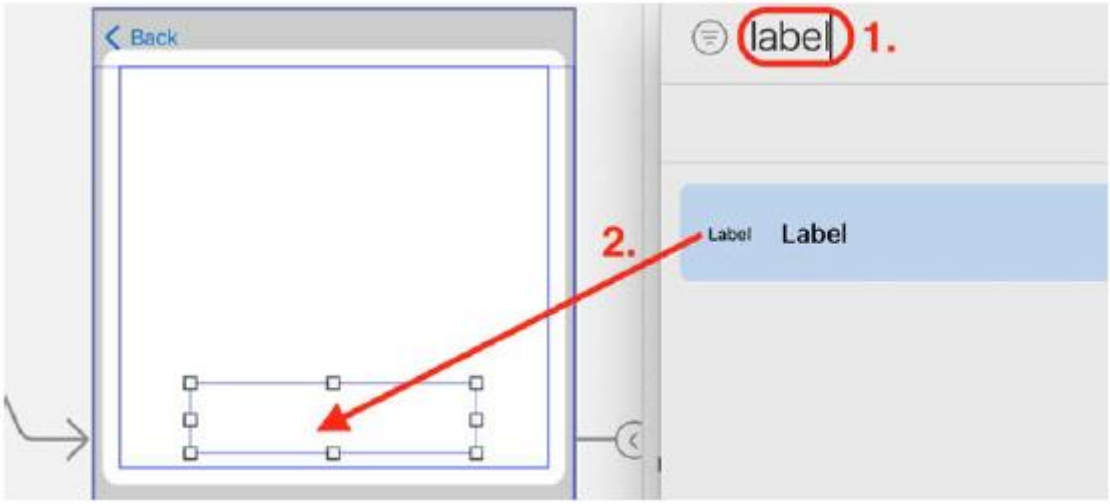
Y: 245

Szerokość: 224

Wzrost: 56



5. Do widoku dodasz etykietę zawierającą tekst Dostępne terminy. Kliknij przycisk Biblioteka. Wpisz etykietę w polu filtra. W wynikach pojawi się obiekt Etykieta. Przeciągnij go do właśnie dodanego widoku:



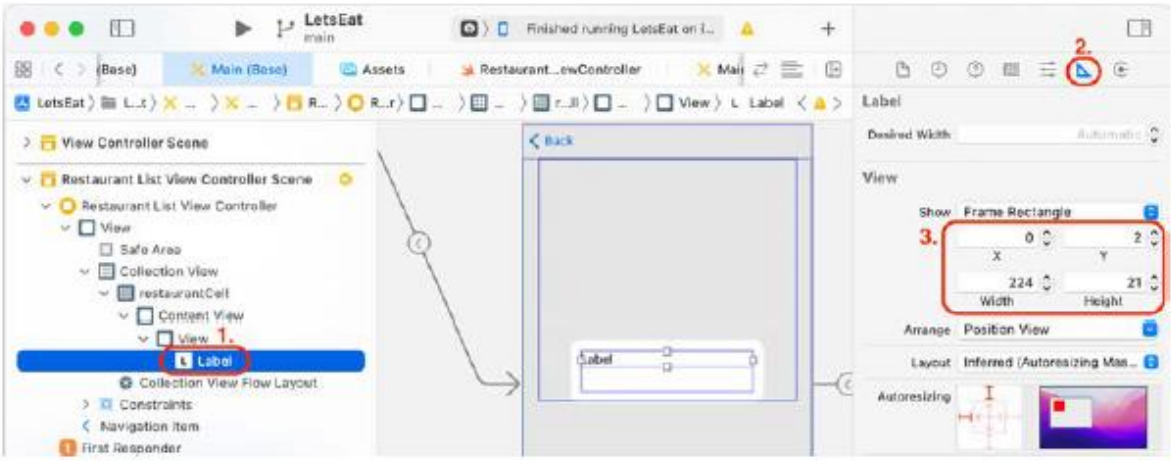
6. Po zaznaczeniu etykiety kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok, aby ustawić położenie etykiety w widoku:

X: 0

Y: 2

Szerokość: 224

Wzrost: 21

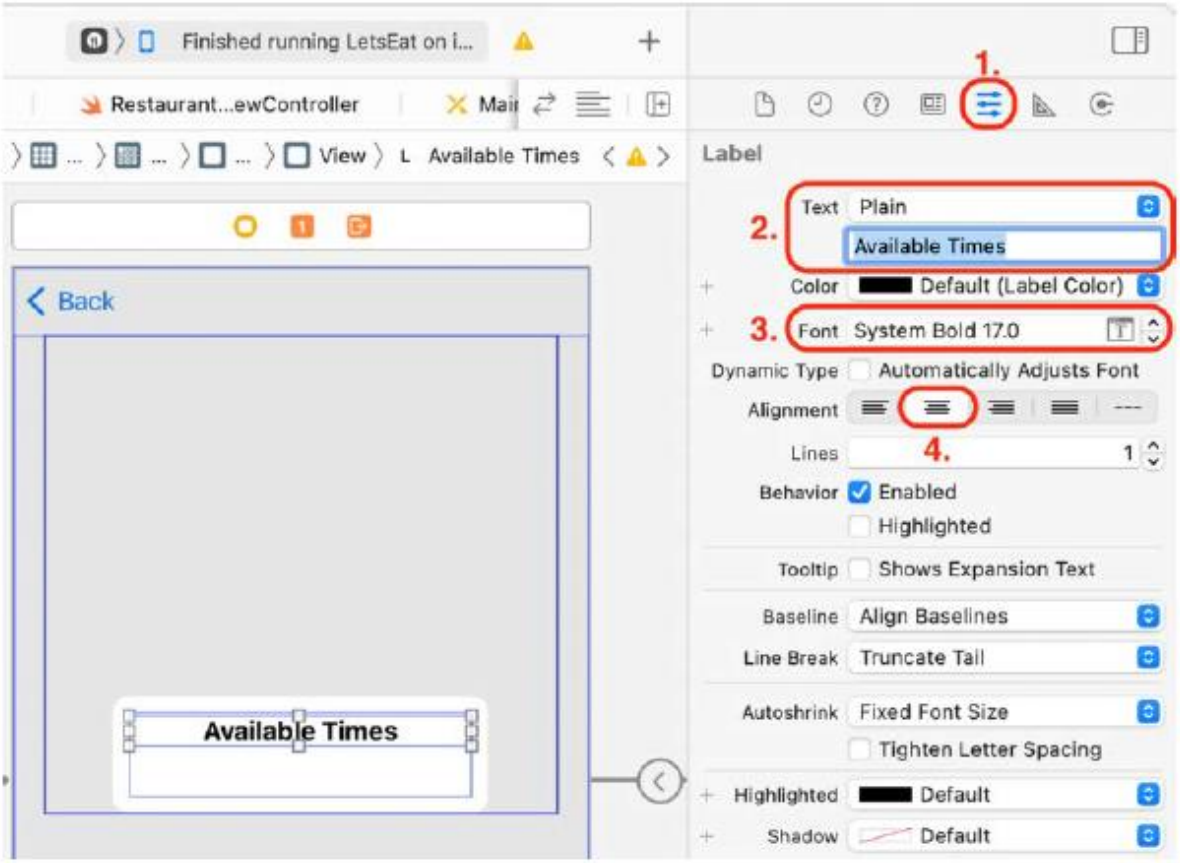


7. Aby ustawić i skonfigurować tekst Dostępne godziny dla etykiety, kliknij przycisk Inspektora atrybutów. Zaktualizuj następujące wartości:

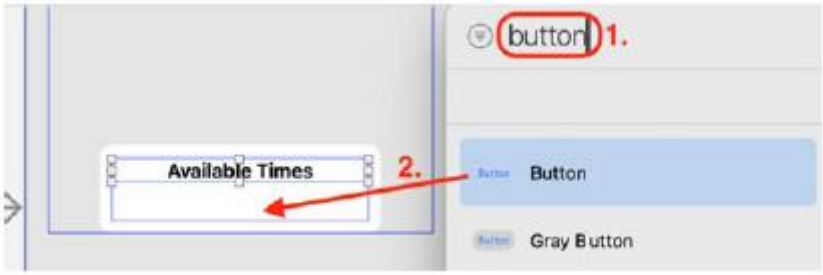
Tekst: Zwykły, a następnie dodaj Dostępne terminy w pustym polu tekstowym poniżej

Wyrównanie: środek

Czcionka: Systemowa pogrubiona 17.0



8. Do widoku dodasz przyciski z terminami rezerwacji. Kliknij przycisk Biblioteka. Wpisz przycisk w polu filtra. W wynikach pojawi się obiekt Button. Przeciągnij go do tego samego widoku, w którym znajdowała się etykieta Dostępne terminy:



9. Aby ustawić tekst i tło przycisku, zaznacz go i kliknij przycisk Inspektora atrybutów.

Zaktualizuj następujące wartości:

Typ: Systemowy

Styl: domyślny

Tytuł: Zwykły, a następnie dodaj godzinę 19:30 w pustym polu tekstowym poniżej

Czcionka: Systemowa pogrubiona 15.0

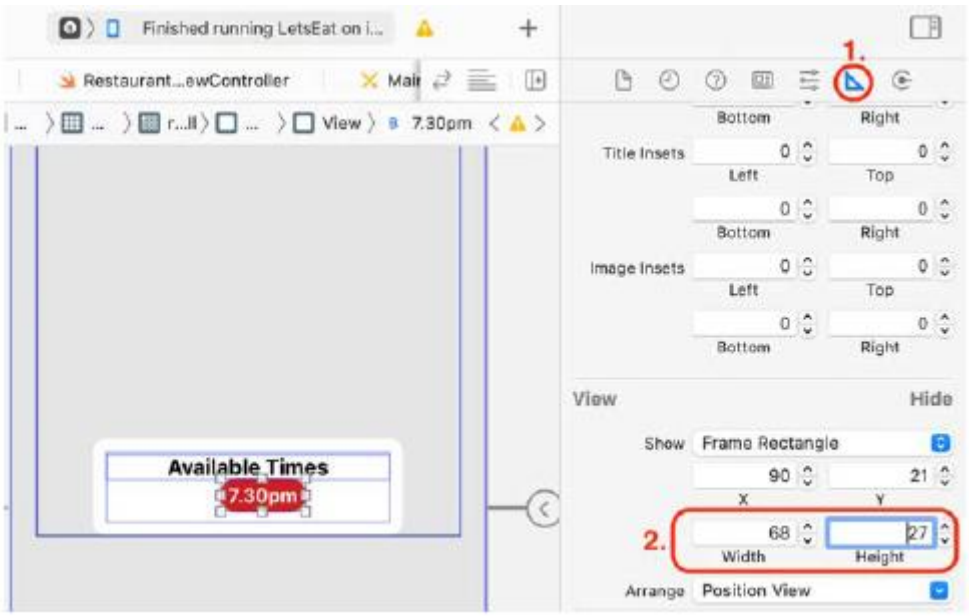
Kolor tekstu: Kolor biały

Tło: time-bg

10. Aby zmienić szerokość i wysokość przycisku, kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok:

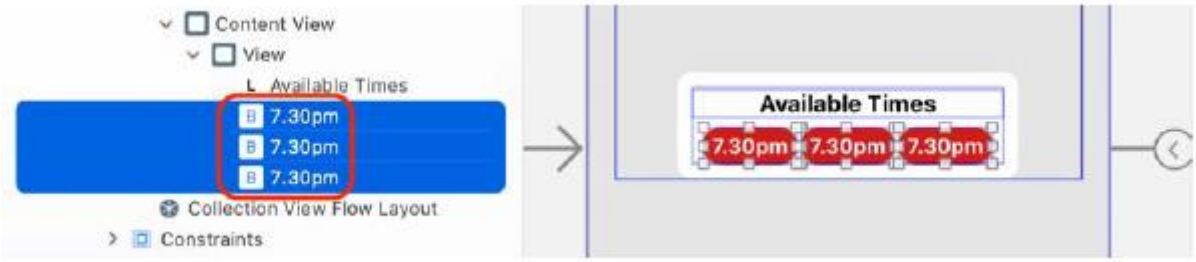
Szerokość: 68

Wzrost: 27



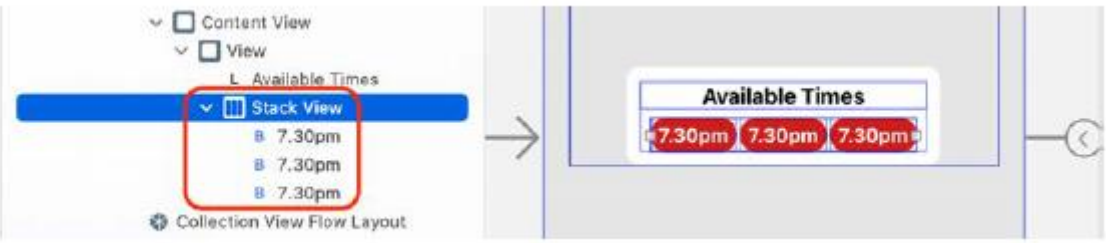
11. Jak widać w prezentacji aplikacji, znajdują się trzy przyciski rezerwacji. Wybierz przycisk i naciśnij Command + C, aby skopiować. Naciśnij dwukrotnie Command + V, aby wkleić. Powinieneś teraz mieć trzy przyciski. Ułóż je w następujący sposób:

12. Osadzanie elementów interfejsu użytkownika w widokach stosu ułatwia zarządzanie nimi. Wszystkie przyciski osadzisz w widoku stosu. Kliknij jeden przycisk, następnie naciśnij Shift i kliknij pozostałe dwa. Wszystkie trzy przyciski powinny być teraz wybrane:



13. Wybierz opcję Osadź w | Widok stosu z menu Edytor. Spowoduje to umieszczenie wszystkich trzech przycisków w widoku stosu, który ma siatkę komórek z 1 wierszem i 3 kolumnami.

14. Sprawdź, czy wszystkie przyciski są teraz widokami podrzędnymi widoku stosu, sprawdzając podwidoki widoku stosu w konspekcie dokumentu:



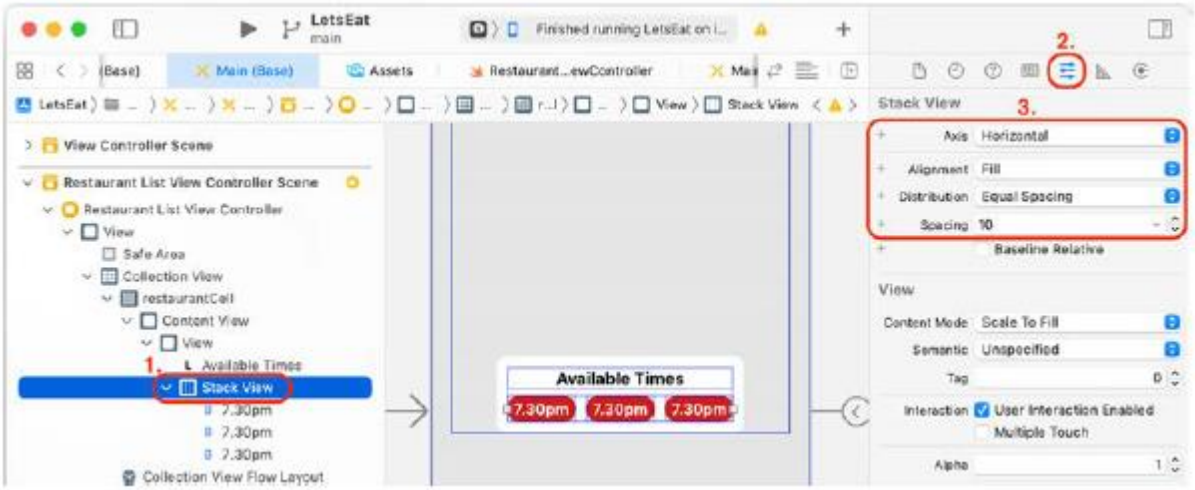
15. Wybierz Widok stosu w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. Zaktualizuj następujące wartości, aby równomiernie rozmieścić przyciski:

Oś: pozioma

Wyrównanie: Wypełnienie

Dystrybucja: Równe odstępy

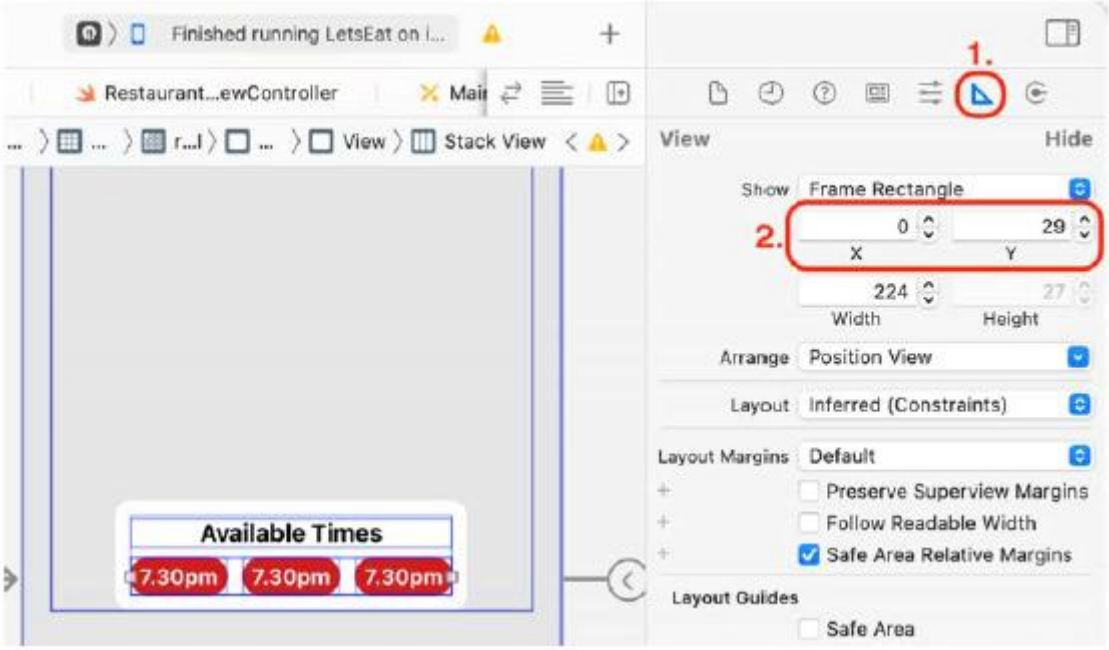
Rozstaw: 10



16. Aby umieścić widok stosu w widoku otaczającym, kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok:

X: 0

Y: 29



17. Dodasz widok kontenera do komórki widoku kolekcji RestaurantCell. Widok ten będzie zawierał widok obrazu przedstawiający zdjęcie restauracji. Kliknij przycisk Biblioteka. Wpisz uiview w polu filtra. W wynikach pojawi się obiekt View. Przeciągnij go do komórki prototypowej:

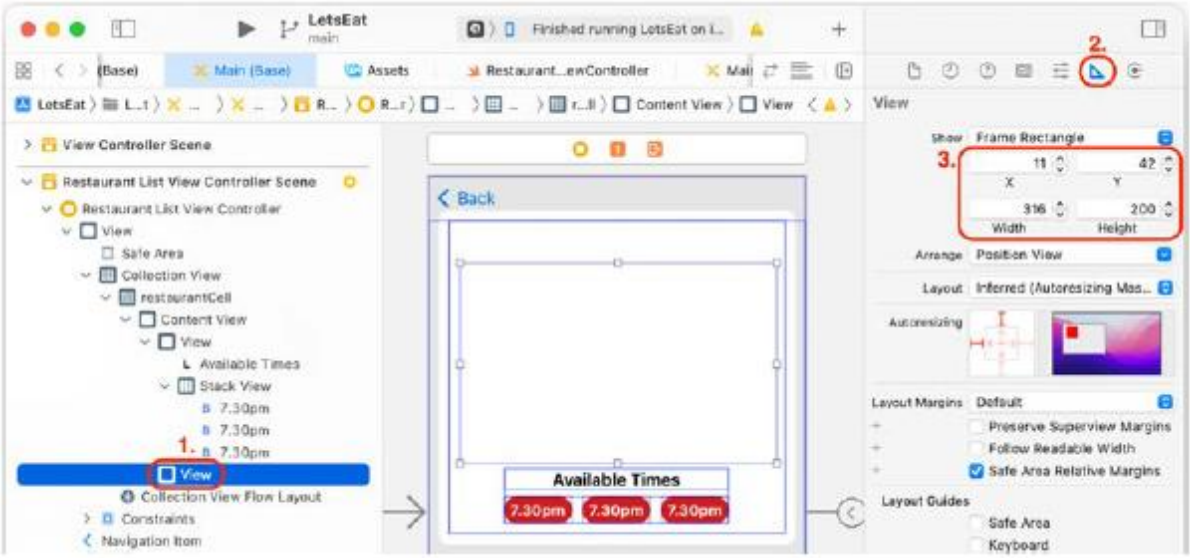
18. Po wybraniu widoku kliknij przycisk Inspektor rozmiaru. Zaktualizuj następujące wartości w sekcji Widok, aby umieścić widok w komórce:

X: 11

Y: 42

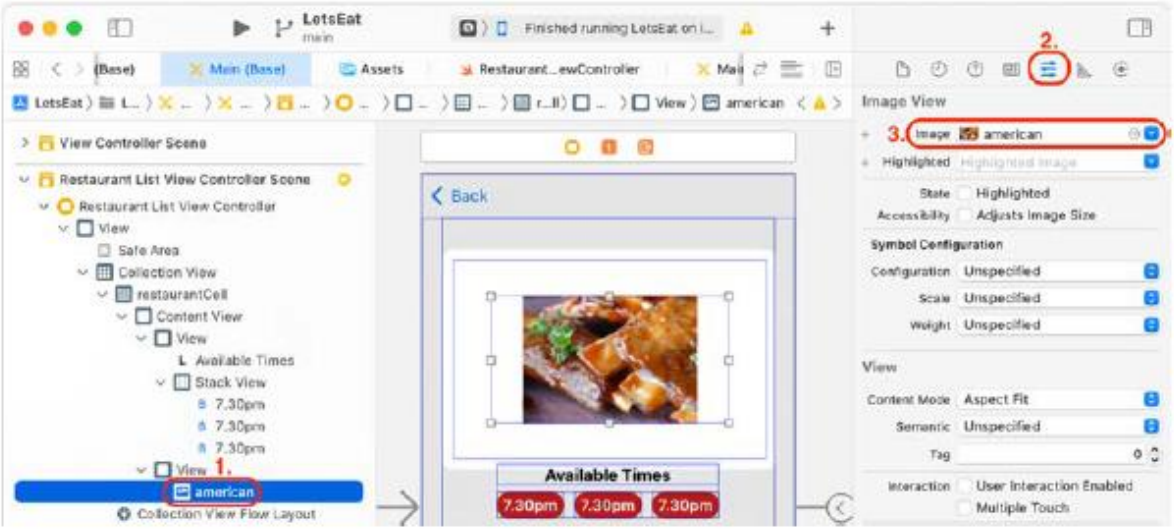
Szerokość: 316

Wzrost: 200



19. Dodasz widok obrazu do dodanego wcześniej widoku kontenera. Kliknij przycisk Biblioteka. Wpisz obraz w polu filtra. W wynikach pojawi się obiekt Image View. Przeciągnij go do właśnie dodanego widoku:

20. Aby ustawić tymczasowy obraz zastępczy dla widoku obrazu, zaznacz go i kliknij przycisk Inspektora atrybutów. Ustaw obraz na amerykański.



Obrazy załadujesz za pomocą kodu w rozdziale 15, Pobieranie danych do widoków kolekcji.

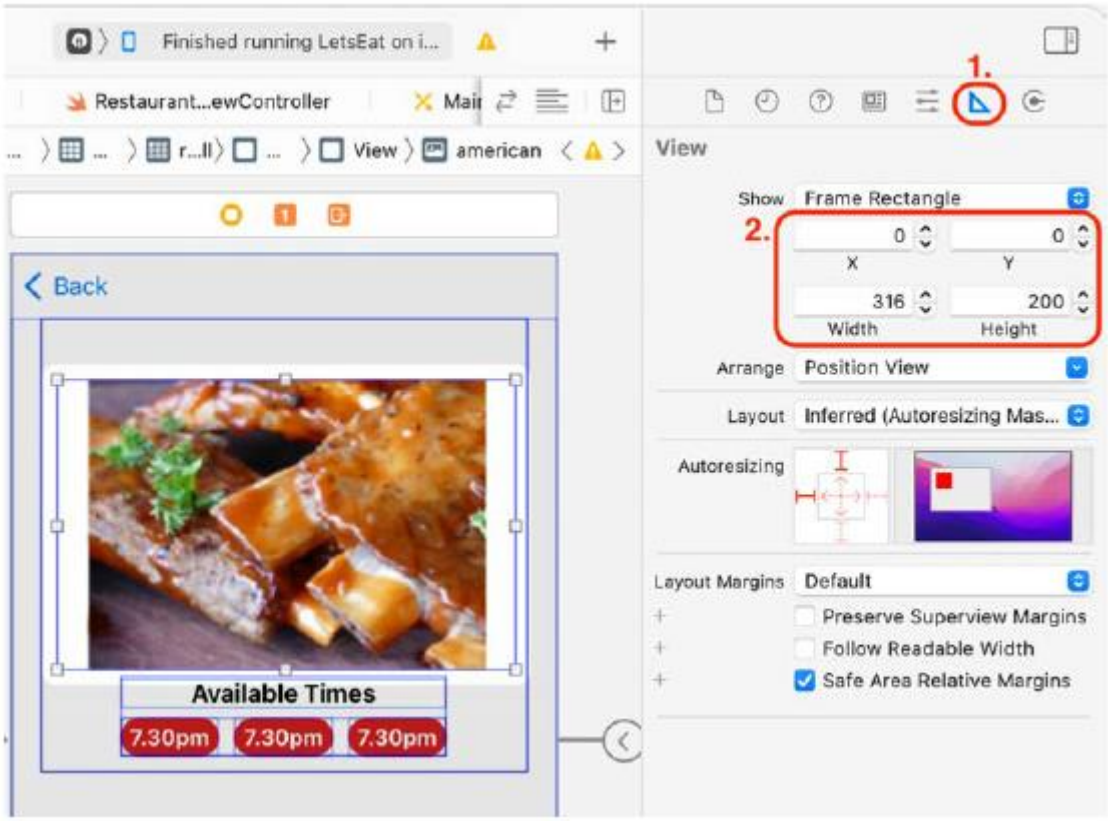
21. Aby umieścić widok obrazu w widoku kontenera, zaznacz go i kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok:

X: 0

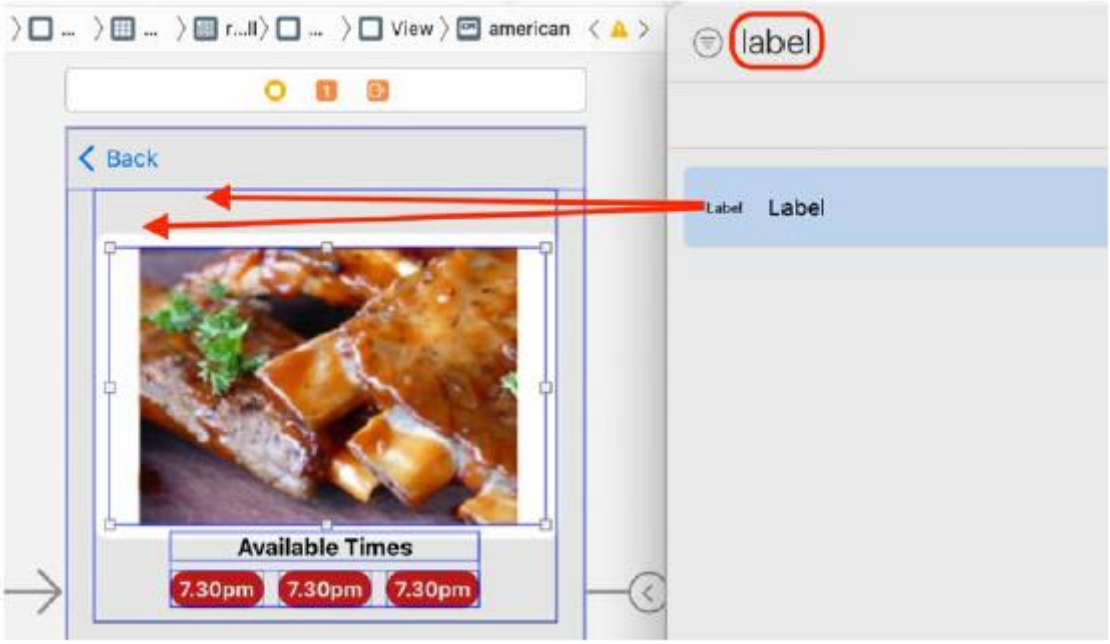
Y: 0

Szerokość: 316

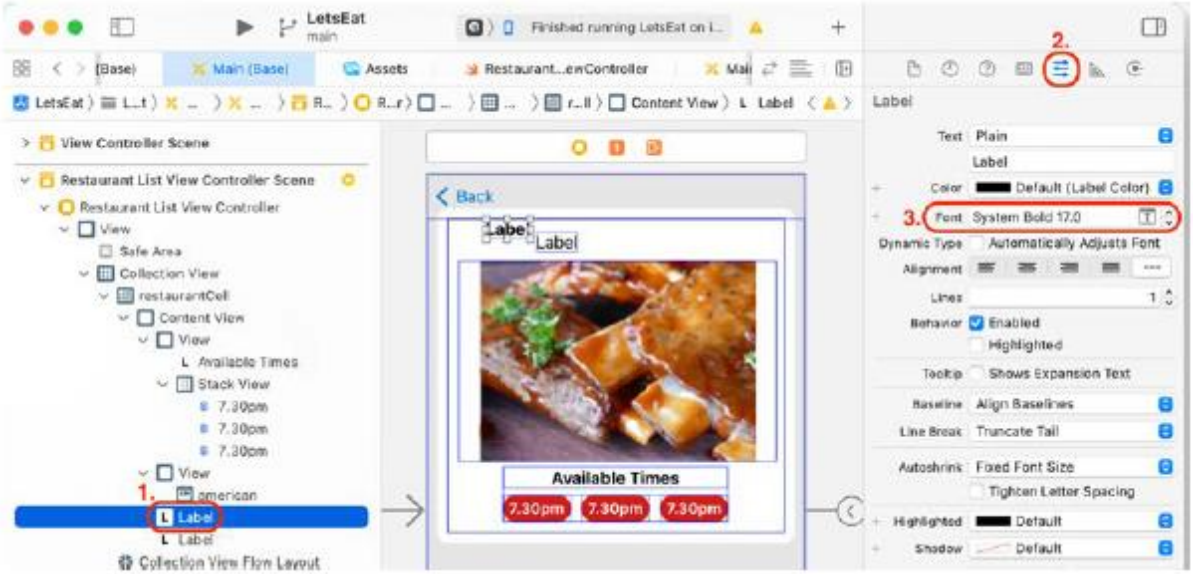
Wzrost: 200



22. Dodasz etykiety, które będą wyświetlać nazwę restauracji i rodzaj oferowanej przez nią kuchni. Kliknij przycisk Biblioteka. Wpisz etykietę w polu filtra. W wynikach pojawi się obiekt Etykieta. Przeciągnij dwa obiekty Label do komórki prototypowej:



23. Jedna z etykiet zostanie użyta jako nazwa restauracji. Wybierz etykietę i kliknij przycisk Inspektora atrybutów. Ustaw czcionkę na System Pogrubienie 17.0, aby skonfigurować styl czcionki dla tej etykiety.



24. Aby umieścić tę etykietę w komórce widoku kolekcji RestaurantCell, wybierz etykietę i kliknij

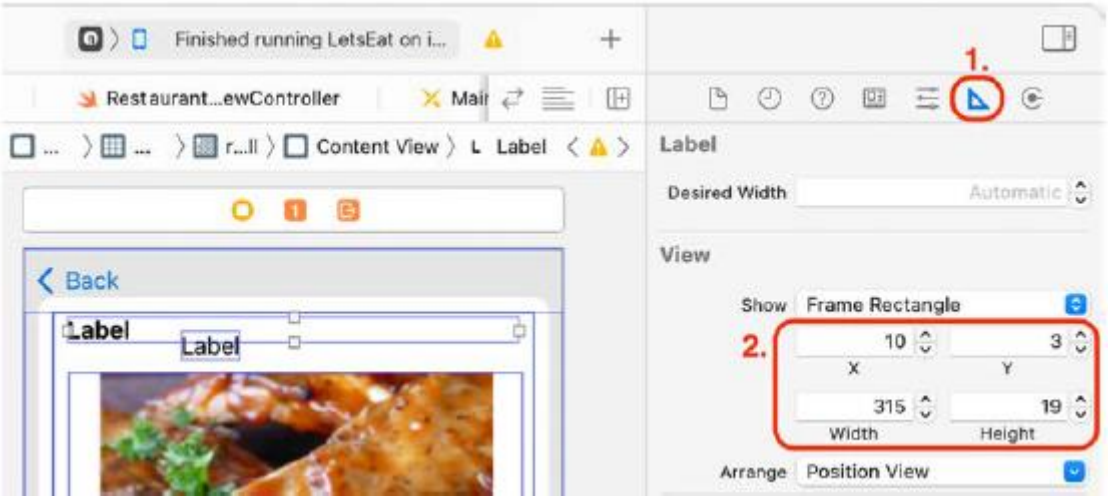
przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok:

X: 10

Y: 3

Szerokość: 315

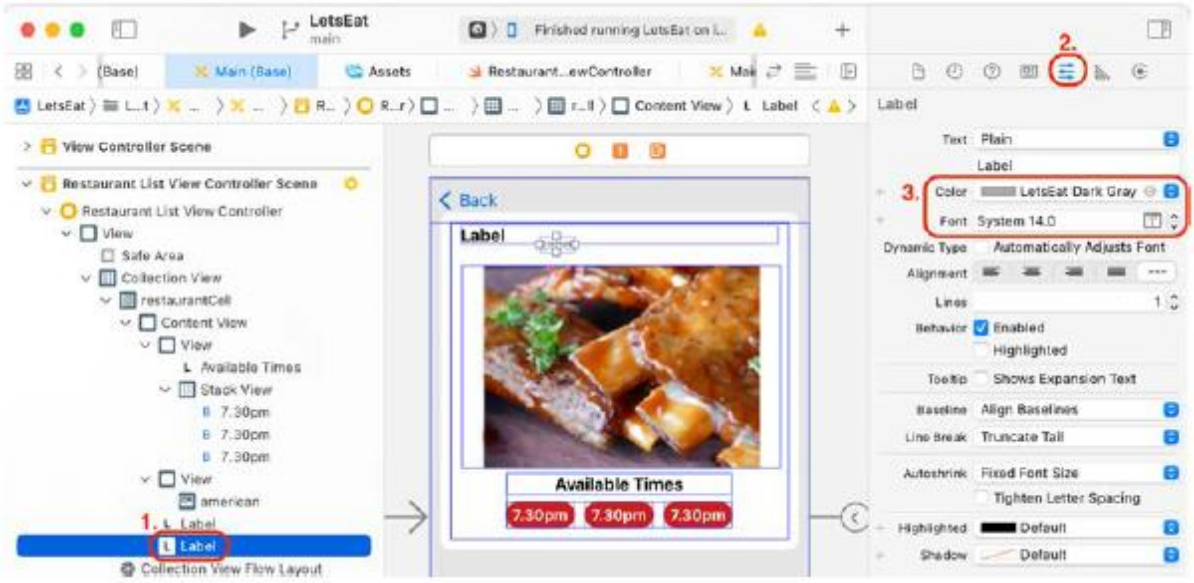
Wzrost: 19



25. Druga etykieta będzie wykorzystywana do prezentacji kuchni oferowanej przez restaurację. Wybierz drugą etykietę i kliknij przycisk Inspektora atrybutów. Zaktualizuj następujące wartości, aby skonfigurować czcionkę i kolor:

Kolor: LetsEat ciemnoszary

Czcionka: System 14.0



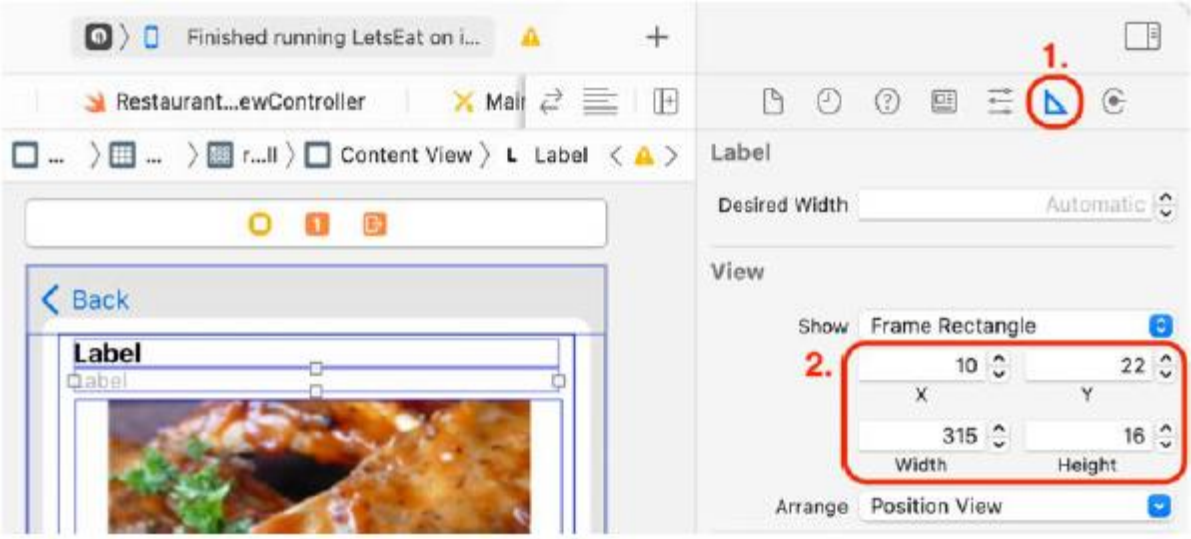
26. Etykieta ta powinna znajdować się pod etykietą zawierającą nazwę restauracji. Po zaznaczeniu etykiety kliknij przycisk Inspektora rozmiaru. Zaktualizuj następujące wartości w sekcji Widok, aby umieścić tę etykietę w komórce widoku kolekcji RestaurantCell:

X: 10

Y: 22

Szerokość: 315

Wzrost: 16



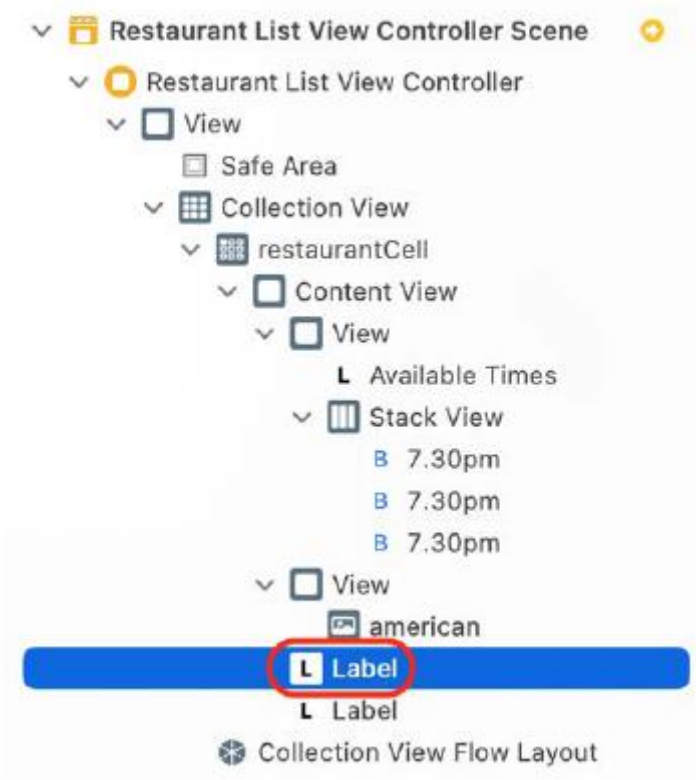
Dodałeś wszystkie elementy restauracjiCell i ustawiłeś ich położenie za pomocą Inspektora rozmiaru. Teraz musisz dodać do nich ograniczenia Auto Layout, aby zapewnić dostosowanie interfejsu użytkownika do rozmiaru i orientacji ekranu urządzenia. Zrobisz to w następnej sekcji.

Dodanie ograniczeń automatycznego układu do kolekcji RestaurantCell

Podobnie jak poprzednio w przypadku komórki widoku kolekcji ExploreCell, teraz dodasz ograniczenia automatycznego układu do wszystkich elementów restauracjiCell. Ponieważ do pozycjonowania elementów użyłeś Inspektora rozmiaru, ich położenie względem siebie oraz wartości wiązań powinny być już poprawnie ustawione, co ułatwi Ci dodanie wiązań. Ponieważ restauracjaCell zawiera wiele elementów, nie spiesz się w tej sekcji.

Wykonaj następujące kroki:

1. Aby ustawić ograniczenia dla etykiety zawierającej nazwę restauracji, wybierz górną Etykietę w konspekcie dokumentu:



Pozycja etykiety w konspekcie dokumentu nie określa pozycji etykiety w komórce widoku kolekcji RestaurantCell. Sprawdź, czy została wybrana właściwa etykieta. Powinien być czarny, a nie jasnoszary.

2. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 3 (odstęp pomiędzy górną krawędzią a górną krawędzią widoku otaczającego)

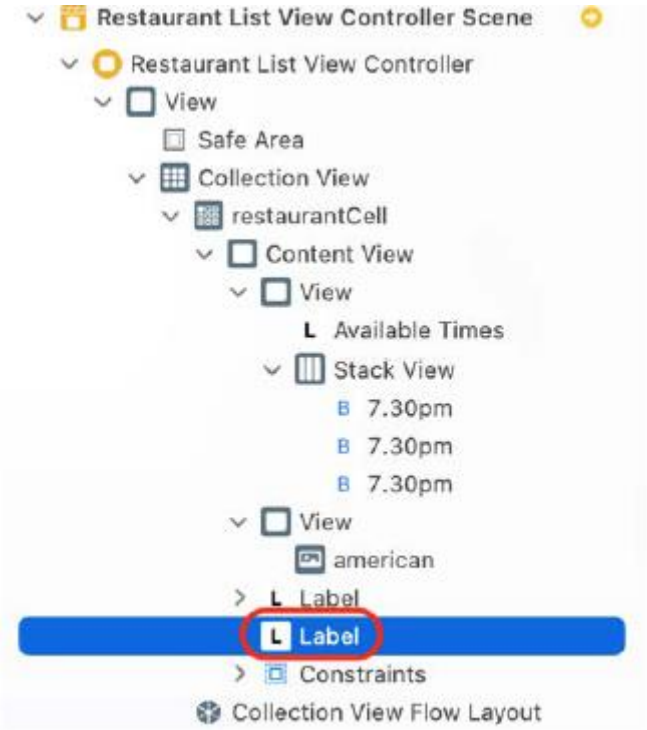
Po lewej: 10 (odstęp między lewą krawędzią a lewą krawędzią widoku otaczającego)

Po prawej: 10 (odstęp między prawą krawędzią a prawą krawędzią widoku otaczającego)

Wysokość: 19 (przestrzeń pomiędzy górną a dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia.

3. Aby ustawić ograniczenia dla etykiety zawierającej kuchnię restauracji, wybierz Etykietę pod poprzednią Etykietą w konspekcie dokumentu:



Pozycja etykiety w konspekcie dokumentu nie określa pozycji etykiety w komórce widoku kolekcji RestaurantCell. Sprawdź, czy została wybrana właściwa etykieta. Powinien być jasnoszary, a nie czarny.

4. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 0 (wiąże górną krawędź z dolną krawędzią poprzedniej etykiety)

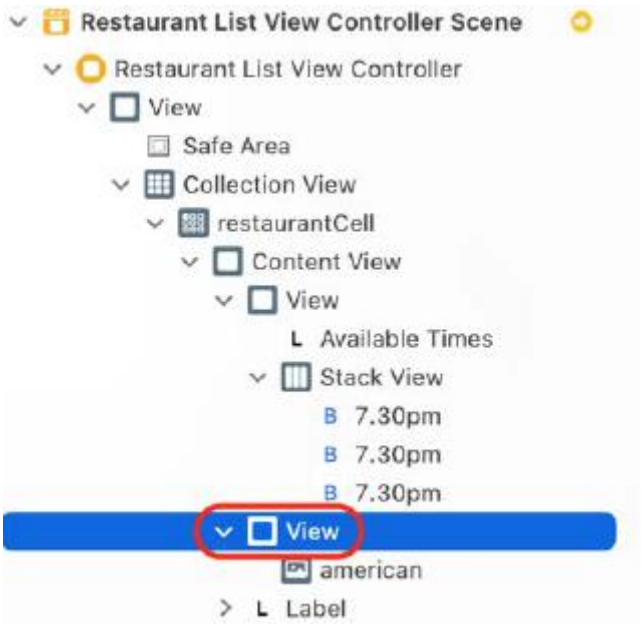
Po lewej: 10 (odstęp między lewą krawędzią a lewą krawędzią widoku otaczającego)

Po prawej: 10 (odstęp między prawą krawędzią a prawą krawędzią widoku otaczającego)

Wysokość: 16 (przestrzeń pomiędzy górną i dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia.

5. Aby ustawić ograniczenia dla widoku zawierającego widok obrazu ze zdjęciem restauracji, wybierz Widok zawierający widok obrazu w konspekcie dokumentu:



6. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 4 (odstęp pomiędzy górną a dolną krawędzią etykiety)

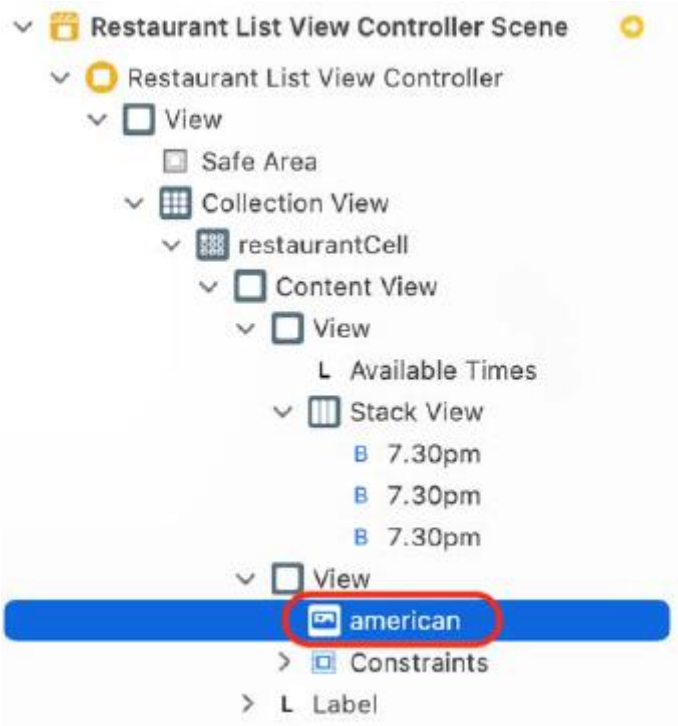
Szerokość: 316 (odstęp pomiędzy lewą i prawą krawędzią)

Wysokość: 200 (przestrzeń pomiędzy górną i dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 3 wiązania. Należy pamiętać, że pozycja pozioma tego widoku nie została jeszcze ustawiona.

7. Kliknij przycisk Wyrównaj, zaznacz opcję Poziomo w kontenerze i wprowadź następującą wartość: 0. Po zakończeniu kliknij przycisk Dodaj 1 ograniczenie. Ustawia poziome położenie widoku na środek otaczającego widoku. Ponieważ szerokość tego widoku jest ustawiona, położenie lewej i prawej krawędzi można określić automatycznie.

8. Aby ustawić ograniczenia dla widoku obrazu zawierającego zdjęcie restauracji, wybierz Widok obrazu w konspekcie dokumentu:



9. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 0

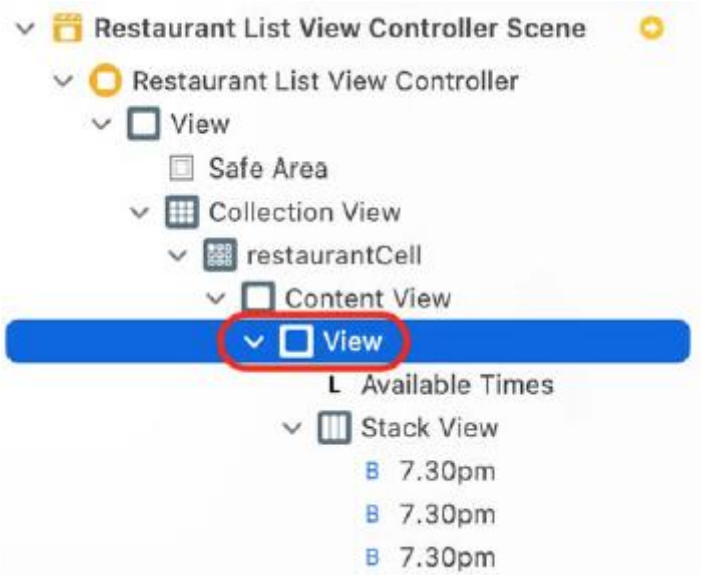
Po lewej: 0

Po prawej: 0

Dół: 0

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia. Spowoduje to powiązanie krawędzi widoku obrazu z widokiem otaczającym.

10. Aby ustawić ograniczenia dla widoku zawierającego etykietę i przyciski, wybierz w konspekcie dokumentu Widok zawierający etykietę Dostępne terminy i Widok stosu:



11. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 3 (odstęp pomiędzy górną a dolną krawędzią widoku zawierającego widok obrazu)

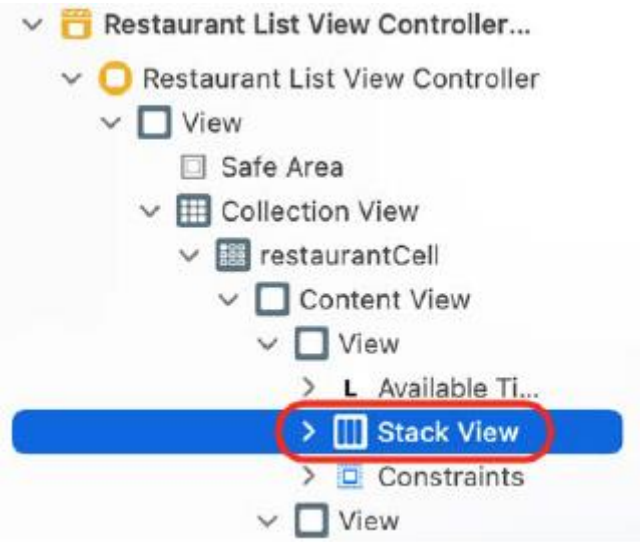
Szerokość: 224 (odstęp pomiędzy lewą i prawą krawędzią)

Wysokość: 56 (przestrzeń pomiędzy górną a dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 3 wiązania. Należy pamiętać, że pozioma pozycja widoku nie została jeszcze ustawiona.

12. Kliknij przycisk Wyrównaj. Zaznacz opcję Poziomo w kontenerze i wprowadź następującą wartość: 0. Po zakończeniu kliknij przycisk Dodaj 1 ograniczenie. Ustawia poziomą pozycję widoku na środek widoku zawartości. Ponieważ szerokość tego widoku jest ustawiona, położenie lewej i prawej krawędzi można określić automatycznie.

13. Aby ustawić ograniczenia dla widoku stosu zawierającego przyciski, wybierz Widok stosu w konspekcie dokumentu:



14. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 6 (odstęp między górną a dolną krawędzią etykiety Dostępne terminy)

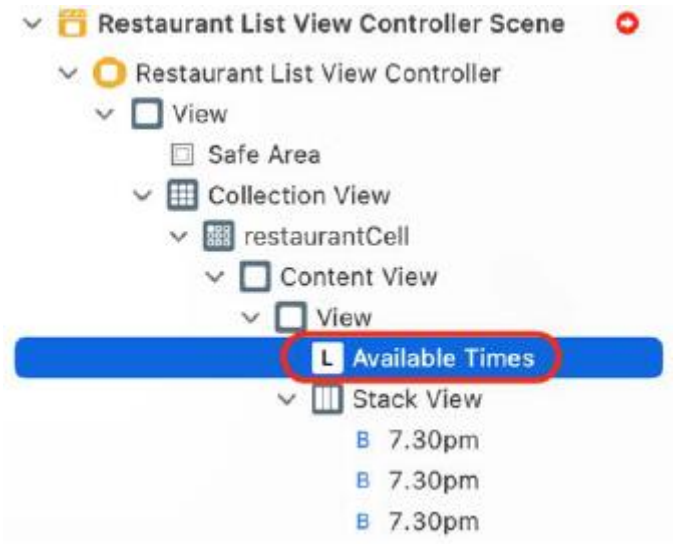
Po lewej: 0 (odstęp między lewą krawędzią a lewą krawędzią widoku otaczającego)

Po prawej: 0 (odstęp między prawą krawędzią a prawą krawędzią widoku otaczającego)

Wysokość: 27 (przestrzeń pomiędzy górną i dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia.

15. Aby ustawić ograniczenia dla etykiety Dostępne terminy, wybierz etykietę Dostępne terminy w konspekcie dokumentu:



16. Kliknij przycisk Dodaj nowe ograniczenia i wprowadź następujące wartości:

Góra: 2 (odstęp pomiędzy górną krawędzią a górną krawędzią widoku otaczającego)

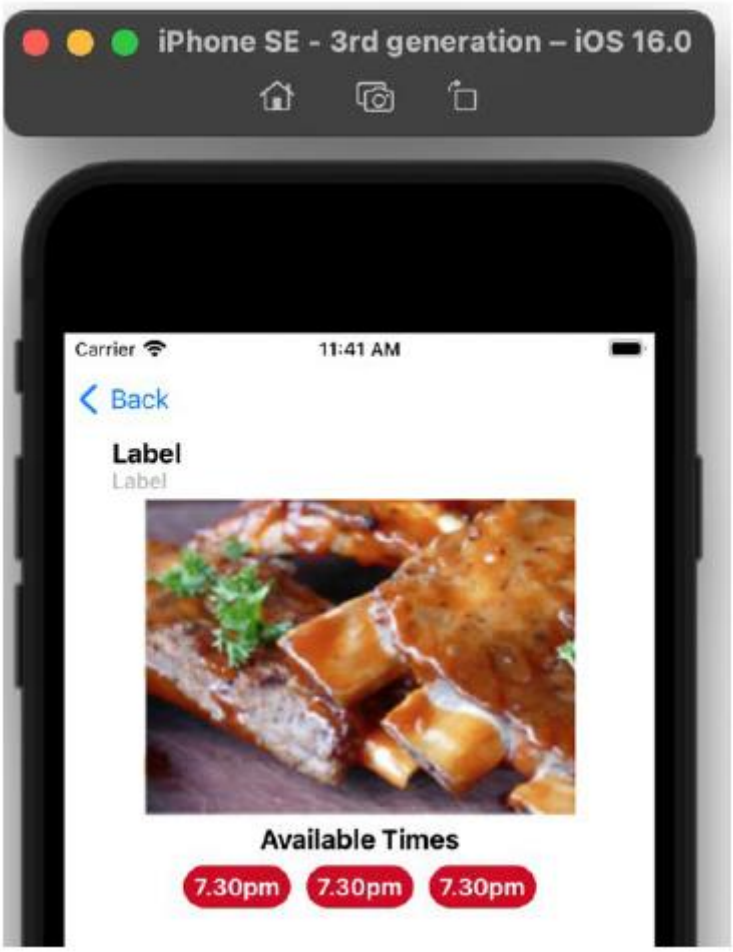
Lewa: 0 (wiąże lewą krawędź z lewą krawędzią otaczającego widoku)

Prawa: 0 (wiąże prawą krawędź z prawą krawędzią otaczającego widoku)

Wysokość: 21 (przestrzeń pomiędzy górną i dolną krawędzią)

Po zakończeniu kliknij przycisk Dodaj 4 ograniczenia.

Wszystkie ograniczenia układu automatycznego dla komórki widoku kolekcji RestaurantCell zostały skonfigurowane. Zbuduj i uruchom aplikację, a następnie przejdź do ekranu Lista restauracji. Powinieneś zobaczyć następujące informacje:



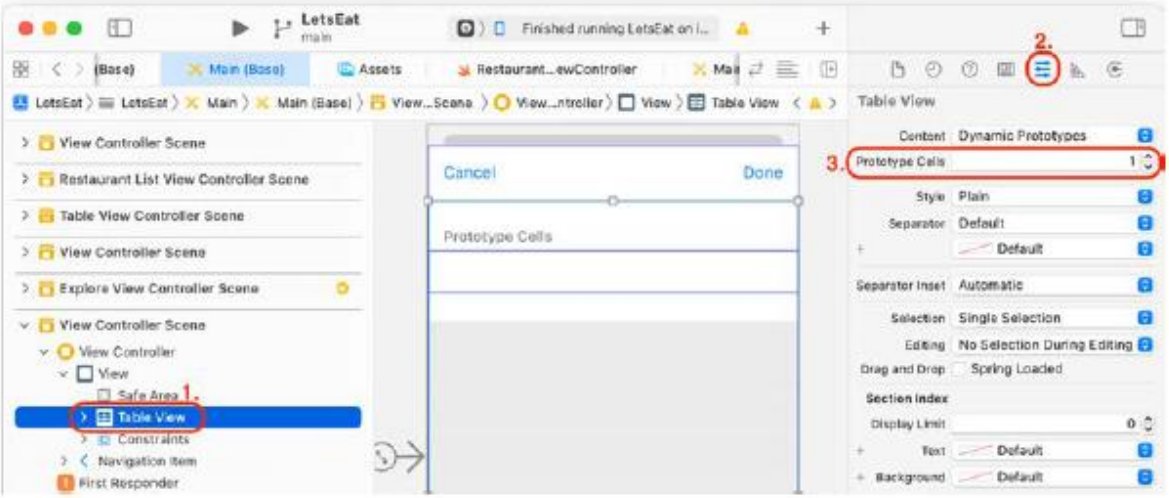
Jak widać, ekran Lista restauracji jest teraz bardziej zgodny z wyglądem pokazanym w przewodniku po aplikacji. Komórka widoku kolekcji RestaurantCell wygląda teraz tak samo jak projekt wycieczki po aplikacji i dodano wszystkie niezbędne ograniczenia. Wspaniały!

Dodałeś wszystkie wymagane elementy interfejsu użytkownika i ograniczenia do restauracjiCell. Po zakończeniu modyfikowania RestaurantCell zmodyfikujmy komórki widoku tabeli na ekranie Lokalizacje w następnej sekcji.

Konfigurowanie komórki widoku tabeli LocationCell

Ostatnią rzeczą do zrobienia w tym rozdziale jest konfiguracja komórek widoku tabeli na ekranie Lokalizacje. Jak widziałeś podczas prezentacji aplikacji, każda komórka widoku tabeli zawiera tylko tekst, więc jedyne, co musisz teraz zrobić, to włączyć prototypową komórkę widoku tabeli i ustawić identyfikator komórki widoku tabeli na LocationCell. Wykonaj następujące kroki:

1. Znajdź scenę kontrolera widoku aktywowaną przyciskiem w Eksploruj scenę kontrolera widoku. Wybierz Widok tabeli w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. Ustaw komórki prototypowe na 1:

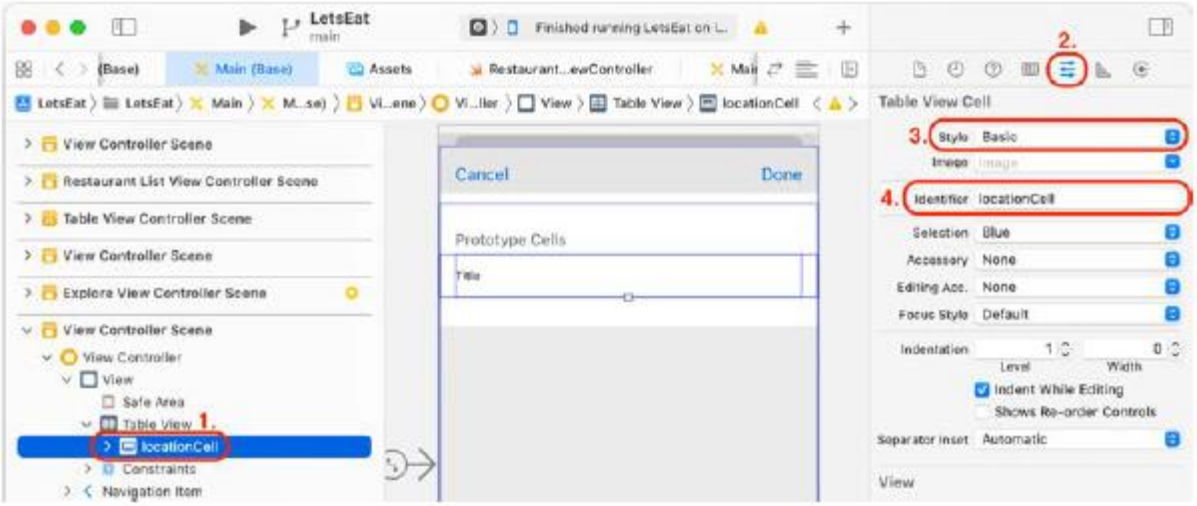


W widoku tabeli pojawi się prototypowa komórka widoku tabeli.

2. Aby ustawić styl i identyfikator komórki, kliknij opcję Widok tabeli Komórka w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów i wprowadź następujące wartości:

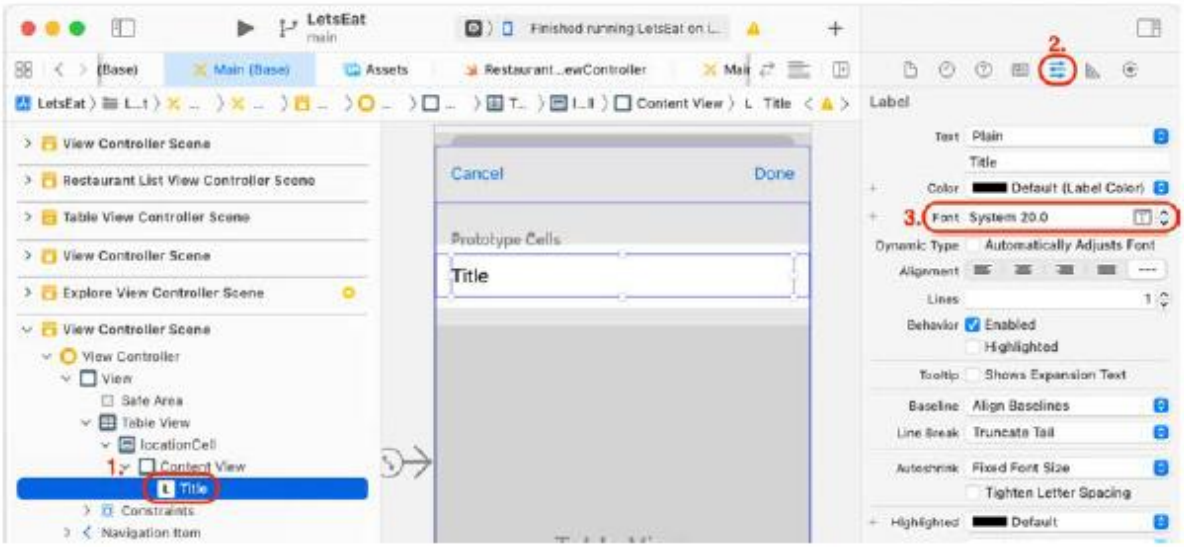
Styl: podstawowy

Identyfikator: LocationCell



Należy pamiętać, że nazwa komórki widoku tabeli prototypowej zostanie zmieniona na LocationCell. Po zmianie stylu z Niestandardowy na Podstawowy w komórce powinno pojawić się słowo Tytuł. To tylko symbol zastępczy. Później zmienisz tę wartość w kodzie.

3. Aby ustawić rozmiar czcionki komórki, kliknij Tytuł w widoku ContentView lokalizacjiCell w konspekcie dokumentu. W Inspektorze atrybutów zmień rozmiar czcionki na 20:



Jeśli teraz zbudujesz i uruchomisz aplikację, a następnie przejdziesz do ekranu Lokalizacje, będzie on nadal pusty, ponieważ nie dodałeś jeszcze żadnego kodu wyświetlającego dane w komórkach. Zrobisz to w Rozdziale 16, Pierwsze kroki z widokami tabel.

Streszczenie

Zmodyfikowałeś komórki na ekranach Eksploruj, Lista restauracji i Lokalizacje, aby dopasować je do wyglądu pokazanego w przewodniku po aplikacji. W przypadku ekranu Eksploruj dodałeś etykiety i widok do nagłówka sekcji widoku kolekcji, skonfigurowałeś przycisk z niestandardowym obrazem i zmodyfikowałeś komórkę widoku kolekcji exploreCell, dodając do niej widok obrazu i etykietę, a także wymagane ograniczenia . W przypadku ekranu Lista restauracji zmodyfikowałeś komórkę widoku kolekcji RestaurantCell, dodając do niej etykiety, przyciski i widok obrazu, skonfigurowałeś ją tak, aby wyświetlała obraz domyślny i dodałeś niezbędne ograniczenia. Na ekranie Lokalizacje skonfigurowałeś prototypową komórkę dla widoku tabeli i ustawiłeś identyfikator komórek widoku tabeli na LocationCell. Masz już doświadczenie w używaniu Konstruktora interfejsów do dodawania i konfigurowania wielu elementów interfejsu użytkownika, ustawiania ich rozmiarów i położenia za pomocą Inspektora rozmiaru oraz stosowania niezbędnych ograniczeń za pomocą przycisków Dodaj nowe ograniczenia i Wyrównaj, aby zapewnić zgodność z różnymi rozmiarami ekranów i orientacje. Będzie to przydatne podczas projektowania własnych interfejsów użytkownika. Powinieneś także być w stanie łatwo prototypować wygląd i przepływ własnych aplikacji. W tym momencie skończyłeś już tworzenie scenorysu i konfiguracji projektu. Możesz przejrzeć każdy ekran, który powinna zawierać Twoja aplikacja, i zobaczyć, jak wyglądają, nawet jeśli żaden z ekranów nie zawiera rzeczywistych danych. Jeśli ta aplikacja przedstawiała dom w budowie, to tak, jakbyś zbudował wszystkie ściany i podłogi, a dom jest teraz gotowy do wykończenia wnętrza. Dobra robota! Na tym kończy się część 2 tej książki. W następnej części zaczniesz wpisywać cały kod wymagany do działania aplikacji. W następnej części zaczniesz od dowiedzenia się więcej o wzorcu projektowym Model-Widok-Kontroler. Dowiesz się także, jak działają widoki kolekcji, które są kluczowe dla zrozumienia działania ekranów Eksploruj i Lista restauracji.

Pierwsze kroki z MVC i widokami kolekcji

W poprzednim rozdziale zmodyfikowałeś komórki na ekranie Eksploracja, ekranie Lista restauracji i ekranie Lokalizacje, aby dopasować je do wycieczki po aplikacji opisanej w Rozdziale 10, Konfigurowanie interfejsu użytkownika. Ukończyłeś początkowy interfejs aplikacji Let's Eat i na tym zakończyłeś część 2 tej książki. Ten rozdział rozpoczyna część 3 tej książki, w której skupisz się na kodzie, który sprawia, że Twoja aplikacja działa. W tym rozdziale dowiesz się o wzorcu projektowym Model-View-Controller (MVC) oraz o tym, jak różne części aplikacji współdziałają ze sobą. Następnie programowo zaimplementujesz widok kolekcji (co oznacza wdrożenie go przy użyciu kodu zamiast scenorysów) przy użyciu placu zabaw, aby zrozumieć, jak działają widoki kolekcji. Na koniec ponownie przejrzysz widoki kolekcji zaimplementowane na ekranach Eksploruj i Lista restauracji, dzięki czemu będziesz mógł zobaczyć, jakie są różnice między ich implementacją w scenorysie a implementacją programową. Pod koniec tego rozdziału zrozumiesz wzorzec projektowy MVC, dowiesz się, jak programowo utworzyć kontroler widoku kolekcji oraz dowiesz się, jak używać delegatów widoku kolekcji i protokołów źródła danych.

Zrozumienie wzorca projektowego Model-View-Controlle

Wzorzec projektowy Model-View-Controller (MVC) jest powszechnym podejściem używanym do tworzenia aplikacji dla systemu iOS. MVC dzieli aplikację na trzy różne części:

• Model: obsługuje zadania przechowywania i reprezentacji danych oraz przetwarzania danych.

• Widok: obejmuje wszystkie elementy widoczne na ekranie, z którymi użytkownik może wchodzić w interakcję.

• Kontroler: zarządza przepływem informacji pomiędzy modelem a widokiem.

Godną uwagi cechą MVC jest to, że widok i model nie oddziałują ze sobą; zamiast tego całą komunikacją zarządza kontroler. Wyobraź sobie na przykład, że jesteś w restauracji. Patrzysz na menu i wybierasz coś, co chcesz. Potem przychodzi kelner, przyjmuje zamówienie i wysyła je kucharzowi. Kucharz przygotowuje Twoje zamówienie, a kiedy jest gotowe, kelner przyjmuje zamówienie i przynosi je Tobie. W tym scenariuszu menu to widok, kelner to kontroler, a kucharz to model. Pamiętaj też, że wszystkie interakcje między Tobą a kuchnią odbywają się wyłącznie za pośrednictwem kelnera; nie ma żadnej interakcji między tobą a kucharzem. Aby zobaczyć jak działa MVC, dowiedzmy się więcej o kontrolerach i klasach. Zobaczysz, co trzeba zrobić, aby zaimplementować kontroler widoku wymagany do zarządzania widokiem kolekcji, który jest używany na ekranie Eksploruj i ekranie Lista restauracji.

Odkrywanie kontrolerów i klas

Do tej pory zaimplementowałeś sceny kontrolera widoku w głównym pliku scenorysu za pomocą Konstruktora interfejsów. Dodano ExploreViewController, kontroler widoku, który zarządza widokiem kolekcji na ekranie Eksploruj, oraz RestaurantListViewController, kontroler widoku, który zarządza widokiem kolekcji na ekranie Ekran Lista restauracji do Twojego projektu. Jednak nadal nie wiesz, jak działa kod dodany do każdego kontrolera widoku, więc przyjrzyjmy się temu teraz. Po uruchomieniu typowej aplikacji na iOS ładowany jest kontroler widoku dla pierwszego wyświetlanego ekranu. Kontroler widoku ma właściwość widoku i automatycznie ładuje instancję widoku przypisaną do jego właściwości widoku. Widok ten może zawierać widoki podrzędne, które również są ładowane. Jeśli jeden z widoków podrzędnych jest widokiem kolekcji, będzie miał właściwości dataSource i delegata. Właściwość dataSource jest przypisana do obiektu dostarczającego dane do widoku kolekcji. Właściwość delegat jest przypisana do obiektu, który obsługuje interakcję użytkownika z widokiem kolekcji. Zazwyczaj kontroler widoku dla widoku kolekcji zostanie również przypisany do właściwości

dataSource i delegata widoku kolekcji. Wywołania metod, które widok kolekcji wyśle do kontrolera widoku, są deklarowane w protokołach UICollectionViewDataSource i UICollectionViewDelegate. Pamiętaj, że protokoły dostarczają jedynie deklaracji metod; implementacja tych wywołań metod znajduje się w kontrolerze widoku. Kontroler widoku dostarczy następnie żądane dane do widoku kolekcji lub obsłuży interakcję z użytkownikiem. Przyjrzyjmy się bliżej widokom kolekcji i protokołom widoku kolekcji w następnej sekcji.

Zrozumienie widoków kolekcji

Widok kolekcji wyświetla uporządkowaną kolekcję komórek widoku kolekcji przy użyciu dostosowywalnych układów. Układ widoku kolekcji jest podyktowany UICollectionViewFlowLayout. Określa kierunek przepływu i rozmiar elementów w widoku kolekcji. Dane wyświetlane w widoku kolekcji są zwykle dostarczane przez kontroler widoku. Kontroler widoku dostarczający dane dla widoku kolekcji musi być zgodny z protokołem UICollectionViewDataSource. Protokół ten deklaruje listę metod, które informują widok kolekcji, ile komórek ma wyświetlić i co wyświetlić w każdej komórce. Obejmuje także tworzenie i konfigurację widoków dodatkowych (takich jak nagłówek sekcji widoku kolekcji). Aby zapewnić interakcję użytkownika, kontroler widoku dla widoku kolekcji musi również być zgodny z protokołem UICollectionViewDelegate, który deklaruje listę metod wyzwalanych, gdy użytkownik wchodzi w interakcję z widokiem kolekcji. Aby zrozumieć, jak działają widoki kolekcji, zaimplementuj kontroler widoku kontrolujący widok kolekcji na placu zabaw CollectionViewBasics. Następnie porównasz to z implementacją kontrolerów widoku na ekranach Eksploruj i Lista restauracji w następnej sekcji. Ponieważ na placu zabaw nie ma scenorysu, nie możesz dodawać elementów interfejsu użytkownika w sposób, w jaki robiłeś to w poprzednich rozdziałach. Zamiast tego dodasz je programowo.

Zaczniesz od utworzenia klasy CollectionViewExampleController, implementacji kontrolera widoku zarządzającego widokiem kolekcji. Następnie utworzysz instancję CollectionViewExampleController i sprawisz, że będzie wyświetlać widok kolekcji zawierający pojedynczą komórkę widoku kolekcji w podglądzie na żywo placu zabaw. Aby to zrobić, wykonaj następujące kroki:

1. Otwórz plac zabaw CollectionViewBasics, który utworzyłeś na początku tego rozdziału. Na samej górze placu zabaw usuń instrukcję var i dodaj instrukcję import PlaygroundSupport. Twój plac zabaw powinien teraz zawierać następujące elementy:

```
import UIKit

import PlaygroundSupport
```

Pierwsza instrukcja importu importuje interfejs API do tworzenia aplikacji na iOS. Druga instrukcja umożliwia wyświetlanie na placu zabaw podglądu na żywo, którego użyjesz do wyświetlenia widoku kolekcji.

2. Dodaj następujący kod po instrukcjach importu, aby zadeklarować CollectionViewExampleController:

```
class CollectionViewExampleController:

UIViewController {

}
```

Ta klasa jest podklasą UIViewController, klasy udostępnianej przez firmę Apple do zarządzania widokami na ekranie.

3. Dodaj następujący kod w nawiasach klamrowych, aby dodać opcjonalną właściwość CollectionView do klasy CollectionViewExampleController:

var collectionView: UICollectionView?

Instancja widoku kolekcji zostanie później przypisana do tej właściwości.

4. Sprawdź, czy Twój kod wygląda następująco:

class CollectionViewExampleController: UIViewController {

var collectionView: UICollectionView?

}

Właśnie zadeklarowałeś i zdefiniowałeś początkową implementację klasy CollectionViewExampleController. W następnej sekcji dowiesz się, jak ustawić liczbę komórek wyświetlanych w widoku kolekcji oraz jak ustawić zawartość każdej komórki.

Zgodny z protokołem UICollectionViewDataSource

Widok kolekcji wyświetla na ekranie siatkę komórek widoku kolekcji. Jednak zanim będzie mógł to zrobić, musi wiedzieć, ile komórek wyświetlić i co umieścić w każdej komórce. Aby udostępnić te informacje widokowi kolekcji, należy dostosować klasę CollectionViewExampleController do protokołu UICollectionViewDataSource. Ten protokół ma dwie wymagane metody:

• kolekcjaView(_:numberOfItemsInSection:) jest wywoływana przez widok kolekcji w celu określenia, ile komórek widoku kolekcji powinno zostać wyświetlonych.

• kolekcjaView(_:cellForItemAt:) jest wywoływana przez widok kolekcji w celu określenia, co ma być wyświetlane w każdej komórce widoku kolekcji.

Dodajmy trochę kodu, aby CollectionViewExampleController był zgodny z protokołem UICollectionViewDataSource. Wykonaj następujące kroki:

1. Aby kontroler CollectionViewExampleController obsługiwał protokół UICollectionViewDataSource, wpisz przecinek (,) po deklaracji nadklasy, a następnie wpisz UICollectionViewDataSource. Kiedy skończysz, Twój kod powinien wyglądać następująco:

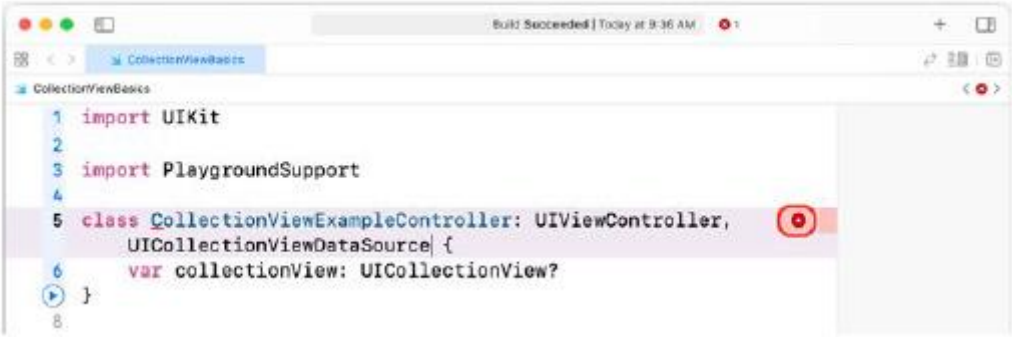
class CollectionViewExampleController: UIViewController,

UICollectionViewDataSource {

var collectionView:UICollectionView?

}

2. Pojawi się błąd, ponieważ nie zaimplementowałeś jeszcze dwóch wymaganych metod. Kliknij ikonę błędu:



3. Komunikat o błędzie informuje, że brakuje wymaganych metod dla protokołu UICollectionViewDataSource. Kliknij przycisk Napraw, aby dodać wymagane metody:

4. Sprawdź, czy Twój kod wygląda następująco

```
class CollectionViewExampleController: UIViewController,

UICollectionViewDataSource {

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

code

}

func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell{

code

}

var collectionView:UICollectionView?

}
```

5. Konwencja w definicji klasy stanowi, że właściwości deklaruje się na początku klasy, przed deklaracjami metod. Zmień układ kodu tak, aby deklaracja właściwości CollectionView znajdowała się na górze, w następujący sposób:

```
class CollectionViewExampleController: UIViewController,

UICollectionViewDataSource {

var collectionView:UICollectionView?

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

code

}
```

6. W kolekcjiView(_:numberOfItemsInSection:) kliknij na słowo kod i wpisz 1. Gotowa metoda powinna wyglądać następująco:

```
func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

1

}
```

Spowoduje to, że instancja CollectionView wyświetli pojedynczą komórkę widoku kolekcji. Zwykle liczba wyświetlanych komórek jest określana przez obiekt modelu. Więcej na ich temat dowiesz się w rozdziale 15, Wprowadzanie danych do widoków kolekcji.

7. W kolekcjiView(_:cellForItemAt:) kliknij słowo code i zmodyfikuj metodę w następujący sposób:

```
func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell{

let cell = collectionView.dequeueReusableCell

(withReuseIdentifier: "BoxCell", for: indexPath)

cell.backgroundColor = .red

return cell

}
```

Oto jak działa ta metoda. Wyobraź sobie, że masz 1000 pozycji do wyświetlenia w widoku kolekcji. Nie potrzebujesz 1000 komórek widoku kolekcji; potrzebujesz tylko tyle, aby wypełnić ekran. Komórki widoku kolekcji przewijane u góry ekranu można ponownie wykorzystać do wyświetlania elementów wyświetlanych u dołu ekranu. Aby mieć pewność, że używasz właściwego typu komórki, używasz

identyfikatora ponownego użycia w celu zidentyfikowania typu komórki. Identyfikator ponownego użycia musi zostać zarejestrowany w widoku kolekcji, co zrobisz później. Następna linia kodu ustawia kolor tła komórki na czerwony, a następna linia zwraca komórkę, która jest następnie wyświetlana na ekranie. Proces powtarza się dla liczby komórek podanej w pierwszej metodzie, która w tym przypadku wynosi 1.

8. Sprawdź, czy klasa CollectionViewExampleController wygląda następująco:

```
class CollectionViewExampleController:

UIViewController, UICollectionViewDataSource {

var collectionView: UICollectionView?

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

1

}

func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell{

let cell = collectionView.dequeueReusableCell

(withReuseIdentifier: "BoxCell", for:

indexPath)

cell.backgroundColor = .red

return cell

}

}
```

Zakończyłeś implementację klasy CollectionViewExampleController. W następnej sekcji dowiesz się, jak utworzyć instancję tej klasy.

Tworzenie instancji CollectionViewExampleController

Teraz, gdy zadeklarowałeś i zdefiniowałeś klasę CollectionViewExampleController, napiszesz metodę tworzenia jej instancji. Wykonaj następujące kroki:

1. Wpisz następujący kod po deklaracji zmiennej CollectionView, aby zadeklarować nową metodę:

```
func createCollectionView() {

}
```

To deklaruje nową metodę createCollectionView(), której użyjesz do utworzenia instancji widoku kolekcji i przypisania jej do właściwości CollectionView.

2. Wpisz następujący kod po otwierającym nawiasie klamrowym, aby zdefiniować treść tej metody:

```
self.collectionView = UICollectionView(

frame: CGRect(x: 0, y: 0, width:

self.view.frame.width, height:

self.view.frame.height),

collectionViewLayout: UICollectionViewFlowLayout())
```

Spowoduje to utworzenie nowej instancji widoku kolekcji i przypisanie jej do kolekcjiView. Wymiary tego widoku kolekcji są dokładnie takie same, jak jego widoku otaczającego, z domyślnym układem przepływu. Układ przepływu określa kolejność wyświetlania komórek widoku kolekcji (od lewej do prawej).

3. Przejdź do następnej linii, a następnie wpisz następujący kod, aby ustawić właściwość dataSource widoku kolekcji na instancję CollectionViewExampleController:

```
self.collectionView?.dataSource = self
```

Właściwość dataSource widoku kolekcji określi, który obiekt zawiera implementację wymaganych metod UIViewControllerDataSource.

4. Przejdź do następnej linii, a następnie wpisz poniższy kod, aby ustawić kolor tła widoku kolekcji na biały:

```
self.collectionView?.backgroundColor = .white
```

5. Przejdź do następnej linii, a następnie wpisz następujący kod, aby ustawić identyfikator komórek widoku kolekcji w widoku kolekcji na BoxCell:

```
self.collectionView?.register(UICollectionViewCell.self,

forCellWithReuseIdentifier: „BoxCell”)
```

Identyfikator ten zostanie użyty w metodzie CollectionView(_:cellForItemAt:) w celu zidentyfikowania typu komórek widoku kolekcji, które mają zostać ponownie wykorzystane.

6. Przejdź do następnej linii, a następnie wpisz poniższy kod, aby dodać widok kolekcji jako widok podrzędny do widoku instancji CollectionViewExampleController:

```
self.view.addSubview(self.collectionView!)
```

Kiedy do pamięci ładowana jest instancja kontrolera widoku, ładowany jest także jego widok wraz ze wszystkimi widokami podrzędnymi. W takim przypadku instancja CollectionViewExampleController automatycznie załaduje swój widok, a ponieważ widok kolekcji jest widokiem podrzędnym jej widoku, widok kolekcji również zostanie załadowany.

7. Sprawdź, czy ukończona metoda wygląda następująco:

```
UICollectionViewFlowLayout())

self.collectionView?.dataSource = self

self.collectionView?.backgroundColor = .white
```

```
self.collectionView?.register(

UICollectionViewCell.self,

forCellWithReuseIdentifier: "BoxCell")

self.view.addSubview(self.collectionView!)

}
```

Teraz potrzebujesz odpowiedniego miejsca, aby wywołać tę metodę. Kontrolery widoku mają właściwość widoku. Widok przypisany do właściwości view zostanie automatycznie załadowany po załadowaniu kontrolera widoku. Po pomyślnym załadowaniu widoku zostanie wywołana metoda viewDidLoad() kontrolera widoku. Zastąpisz metodę viewDidLoad() w klasie CollectionViewControllerExample, którą chcesz wywołać

utwórzCollectionView(). Wykonaj następujące kroki:

1. Tuż przed metodą createCollectionView() wpisz poniższy kod:

```
override func viewDidLoad(){

super.viewDidLoad()

self.view.bounds = CGRect(x: 0, y: 0, width: 375,

height: 667)

createCollectionView()

}
```

Spowoduje to ustawienie rozmiaru widoku na żywo, utworzenie instancji widoku kolekcji, przypisanie jej do kolekcjiView i dodanie jej jako widoku podrzędnego do widoku instancji CollectionViewExampleController. Widok kolekcji wywołuje następnie metody źródła danych, aby określić, ile komórek widoku kolekcji ma zostać wyświetlonych i co wyświetlić w każdej komórce. kolekcjaView(_:numberOfItemsInSection:) zwraca 1, więc zostanie wyświetlona pojedyncza komórka widoku kolekcji. kolekcjaView(_:cellForItemAt:) tworzy komórkę, ustawia kolor tła komórki na czerwony i zwraca ją do wyświetlenia.

2. Sprawdź, czy ukończony plac zabaw wygląda następująco:

```
import UIKit

import PlaygroundSupport

class CollectionViewExampleController:

UIViewController, UICollectionViewDataSource{

var collectionView: UICollectionView?

override func viewDidLoad(){

super.viewDidLoad()

self.view.bounds = CGRect(x: 0, y: 0,

width: 375, height: 667)
```

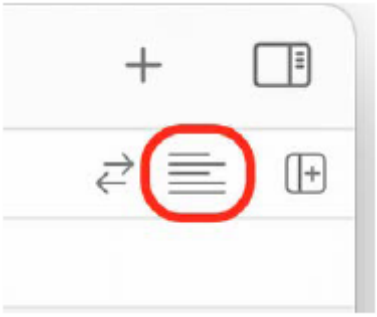
```
createCollectionView()
}
func createCollectionView(){
self.collectionView = UICollectionView(frame:
CGRect(x: 0, y: 0, width:
self.view.frame.width,
height: self.view.frame.height),
collectionViewLayout:
UICollectionViewFlowLayout())
self.collectionView?.dataSource = self
self.collectionView?.backgroundColor = .white
self.collectionView?.register(
UICollectionViewCell.self,
forCellWithReuseIdentifier: "BoxCell")
self.view.addSubview(self.collectionView!)
}
func collectionView(_ collectionView:
UICollectionView, numberOfItemsInSection
section: Int) -> Int {
1
}
func collectionView(_ collectionView:
UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
let cell = collectionView.dequeueReusableCell(
withReuseIdentifier: "BoxCell", for:
indexPath)
cell.backgroundColor = .red
return cell
}
}
```

3. Teraz czas zobaczyć to w akcji. Wpisz następujący kod po całym innym kodzie na placu zabaw:

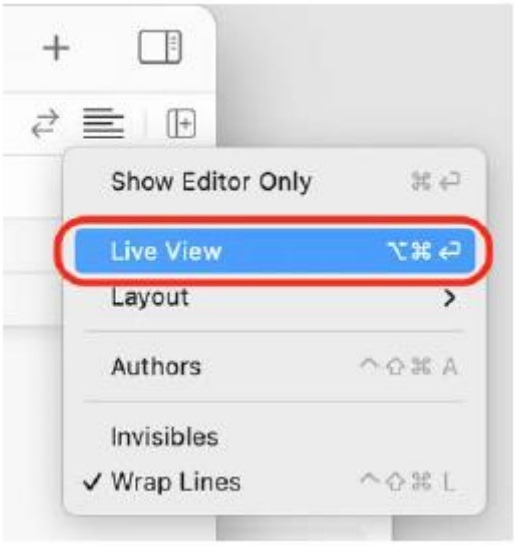
PlaygroundPage.current.liveView = CollectionViewExampleController()

To polecenie tworzy instancję CollectionViewExampleController i wyświetla jej widok w podglądzie na żywo placu zabaw. Metoda createCollectionView() utworzy widok kolekcji i doda go jako widok podrzędny do widoku instancji CollectionViewExampleController, który pojawi się na ekranie.

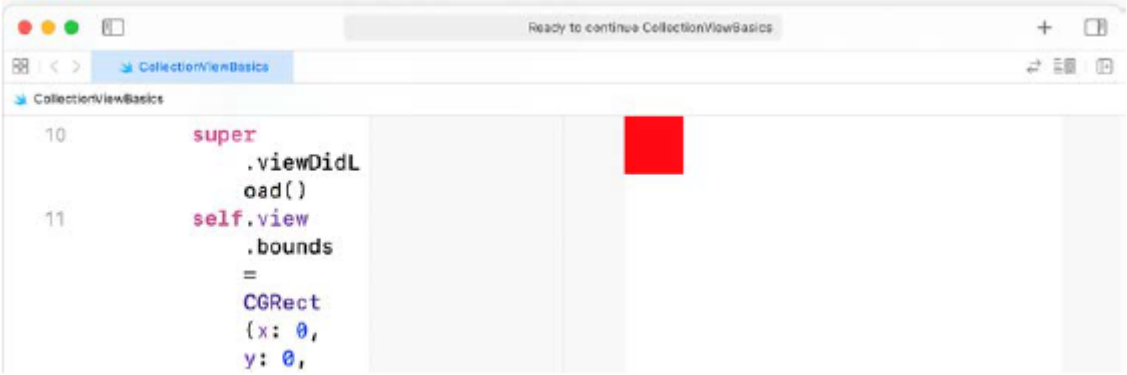
4. Uruchom plac zabaw. Jeśli nie widzisz widoku kolekcji na ekranie, musisz włączyć podgląd placu zabaw na żywo. Kliknij przycisk Dostosuj opcje edytora:



5. Upewnij się, że wybrany jest widok na żywo:



6. W widoku na żywo widok kolekcji będzie zawierał jedną czerwoną komórkę widoku kolekcji:



Właśnie utworzyłeś kontroler widoku dla widoku kolekcji, utworzyłeś jego instancję i wyświetliłeś widok kolekcji w podglądzie na żywo placu zabaw. Dobra robota! W następnej sekcji ponownie omówisz sposób użycia kontrolerów widoku kolekcji na ekranach Eksploruj i Lista restauracji, które zaimplementowałeś w Rozdziale 11, Tworzenie interfejsu użytkownika i Rozdziale 12, Kończenie interfejsu użytkownika. Używając tego, czego nauczyłeś się w tej sekcji jako odniesienia, powinieneś być w stanie zrozumieć, jak one działają.

Ponowne przeglądanie ekranów Eksploruj i Lista restauracji

Pamiętasz klasę ExploreViewController dodaną w rozdziale 11, Tworzenie interfejsu użytkownika, oraz klasę RestaurantListViewController dodaną w rozdziale 12, Kończenie interfejsu użytkownika? Obydwa są przykładami kontrolerów widoku, które zarządzają widokiem kolekcji. Pamiętaj, że kod w obu przypadkach jest bardzo podobny do tego na Twoim placu zabaw. Różnice są następujące:

• Kolor tła komórki ustawiasz programowo w kolekcjiView(_:cellForItemAt:), zamiast ustawiać go w Inspektorze atrybutów.

• Programowo utworzyłeś i przypisałeś widok kolekcji do właściwości CollectionView w CollectionViewExampleController.

• Wymiary widoku kolekcji ustawiasz programowo w UICollectionView(frame: kolekcjaViewLayout:), zamiast używać Inspektora rozmiaru.

• Programowo podłączyłeś źródło danych do kontrolera widoku, zamiast używać Inspektora połączeń.

• Kolor tła widoku kolekcji ustawiasz programowo, zamiast korzystać z Inspektora atrybutów.

• Ustawiasz identyfikator ponownego wykorzystania komórki widoku kolekcji programowo, zamiast korzystać z Inspektora atrybutów.

• Dodano widok kolekcji jako widok podrzędny widoku CollectionViewExampleController programowo, zamiast przeciągać obiekt widoku kolekcji z biblioteki.

Otwórz projekt LetsEat. Przejrzyj jeszcze raz rozdział 11, Tworzenie interfejsu użytkownika i rozdział 12, Kończenie interfejsu użytkownika, aby porównać i skontrastować implementację widoku kolekcji za pomocą scenorysu i wykonując implementację programowo, tak jak zrobiłeś to w tym rozdział.

Streszczenie

Poznałeś szczegółowo wzorce projektowe MVC i kontrolery widoku kolekcji. Następnie ponownie przejrzałeś widoki kolekcji używane na ekranach Eksploruj i Lista restauracji i dowiedziałeś się jak one pracują. Powinieneś teraz zrozumieć wzorzec projektowy MVC, sposób tworzenia kontrolera widoku kolekcji i korzystania z protokołu źródła danych widoku kolekcji. Umożliwi to zaimplementowanie kontrolerów widoku kolekcji dla własnych aplikacji. Do tego momentu skonfigurowałeś widoki i kontrolery widoku dla ekranów Eksploruj i Lista restauracji, ale ekran Eksploruj wyświetla tylko siatkę komórek, a ekran Lista restauracji wyświetla pojedynczą komórkę z obrazem zastępczym. W następnym rozdziale zaimplementujesz obiekty modelu dla ekranu Eksploruj, aby mógł wyświetlać listę kuchni. W tym celu odczytasz dane z pliku zapisanego na urządzeniu z systemem iOS, utworzysz struktury do przechowywania tych danych, a na koniec dostarczysz je do instancji ExploreViewController, aby mogły zostać wyświetlone w widoku kolekcji na ekranie Eksploruj.

Pobieranie danych do widoków kolekcji

Poznałeś wzorzec projektowy Model-View-Controller (MVC) oraz widoki kolekcji. Ponownie odwiedziłeś ekrany Eksploruj i Lista restauracji i zobaczyłeś, jak działają widoki kolekcji na obu ekranach. Jednak w tym momencie oba ekrany wyświetlają tylko komórki, które nie zawierają żadnych danych. Jak pokazano w prezentacji aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika, ekran Eksploruj powinien wyświetlać listę kuchni, a ekran Lista restauracji powinien wyświetlać listę restauracji. Zaimplementujesz obiekty modelu na ekranie Eksploruj, aby wyświetlić listę kuchni. Zaczniesz od poznania obiektów modelu, których będziesz używać. Następnie dowiesz się o listach właściwości i zobaczysz, w jaki sposób są one wykorzystywane do przechowywania danych dotyczących kuchni, a także utworzysz strukturę Swift, która będzie mogła przechowywać instancje kuchni. Następnie utworzysz klasę menedżera danych, która odczytuje dane z listy właściwości i wypełnia tablicę struktur. Ta tablica struktur zostanie następnie wykorzystana jako źródło danych dla widoku kolekcji na ekranie Eksploruj. Pod koniec dowiesz się, w jaki sposób listy właściwości służą do przechowywania danych, jak tworzyć obiekty modelu, jak tworzyć klasę menedżera danych, która może ładować dane z listy właściwości do tablicy obiektów modelu, jak aby skonfigurować kontrolery widoku, aby udostępniać obiekty modelu widokom kolekcji oraz jak skonfigurować widoki kolekcji, aby wyświetlać dane na ekranie.

Zrozumienie obiektów modelu

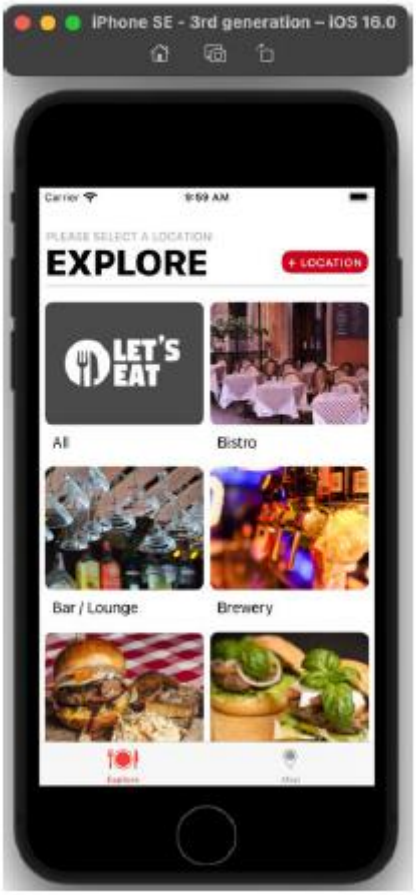
Jak dowiedziałeś się z rozdziału 14, Pierwsze kroki z MVC i widokami kolekcji, typowym wzorcem projektowym dla aplikacji na iOS jest Model-View-Controller, w skrócie MVC. Podsumowując, MVC dzieli aplikację na trzy różne części:

• Model: obsługuje zadania przechowywania, reprezentacji i przetwarzania danych.

• Widok: jest to wszystko, co znajduje się na ekranie, z czym użytkownik może wchodzić w interakcję.

• Kontroler: zarządza przepływem informacji pomiędzy modelem i widokiem.

Przyjrzyjmy się ponownie projektowi ekranu Eksploruj, który widziałeś podczas prezentacji aplikacji i który wygląda następująco:



Zbuduj i uruchom aplikację, a ekran Eksploruj będzie wyglądał następująco:

Jak widać, wszystkie komórki są obecnie puste. W oparciu o wzorzec projektowy MVC zakończono implementację widoków (nagłówek sekcji widoku kolekcji i widok kolekcji) oraz kontrolera (klasa ExploreViewController). Teraz dodasz obiekty modelu, które dostarczą dane do wyświetlenia. Najpierw dodasz do projektu listę właściwości ExploreData.plist, która będzie zawierać nazwę i nazwę pliku obrazu dla każdej kuchni. Same obrazy kuchni są już obecne w Twoich zasobach. folderze xcassets. Następnie utworzysz obiekt modelu ExploreItem, który będzie strukturą z dwiema właściwościami. Jedna właściwość będzie używana do przechowywania nazw kuchni, a druga do przechowywania nazw plików obrazów. Następnie utworzysz klasę menedżera danych ExploreDataManager, która załaduje dane z ExploreData.plist, umieści je w tablicy instancji ExploreItem i udostępni tablicę instancji ExploreViewController. Na koniec zmodyfikujesz klasę ExploreViewController, aby mogła dostarczać dane do wyświetlenia widoku kolekcji. Aby dowiedzieć się więcej o plikach list właściwości, dodaj teraz ExploreData.plist do Twojego projektu i zobacz, jak przechowuje on dane dotyczące kuchni.

Zrozumienie plików .plist

Firma Apple opracowała listy właściwości do przechowywania struktur danych lub stanów obiektów w celu późniejszego odtworzenia i przesłania. Są powszechnie używane do przechowywania preferencji aplikacji. Pliki list właściwości mają rozszerzenie nazwy pliku .plist i dlatego często nazywane są plikami .plist. W swoim projekcie będziesz używać pliku .plist zawierającego dane dotyczące kuchni, ExploreData.plist. Aby uzyskać plik ExploreData.plist, musisz pobrać pakiet kodu tej książki. Następnie możesz użyć Xcode, aby wyświetlić jego zawartość. Wykonaj następujące kroki:

1. Jeśli jeszcze tego nie zrobiłeś, pobierz pliki zasobów i ukończ projekt Xcode z tego linku: https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Seventh-Edition.

2. Otwórz folder Chapter15 i zajrzyj do folderu Resources, aby znaleźć plik ExploreData.plist. W tym pliku przechowywane są nazwy kuchni i nazwy plików obrazów.

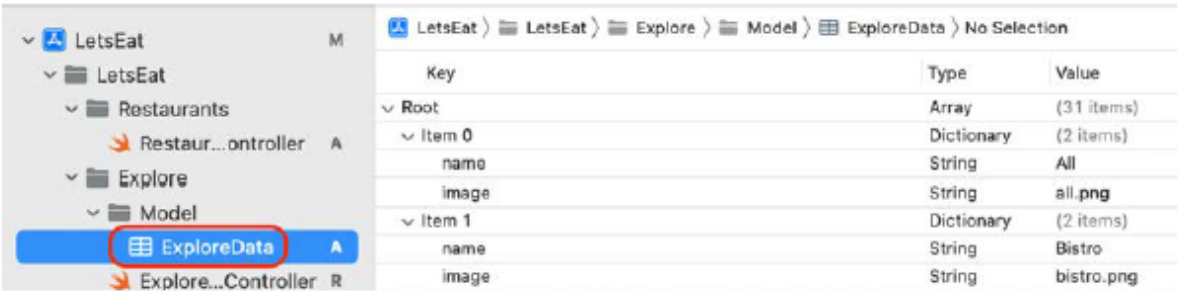
3. Otwórz projekt LetsEat, kliknij prawym przyciskiem myszy folder Eksploruj w nawigatorze projektu i wybierz Nowa grupa.

4. Zmień nazwę nowej grupy, do której właśnie dodałeś Model.

5. Przeciągnij plik ExploreData.plist do tej grupy.

6. Upewnij się, że jest zaznaczona opcja Kopiuj elementy, jeśli to konieczne, i kliknij Zakończ.

Kiedy klikniesz ExploreData.plist w nawigatorze projektu, zobaczysz tablicę zawierającą słowniki, jak pokazano:



Każdy słownik składa się z dwóch elementów. Pierwszy element zawiera klucz, nazwę i wartość opisującą rodzaj kuchni. Drugi element ma klucz, obraz i wartość zawierającą nazwę pliku obrazu kuchni. Wszystkie obrazy kuchni są przechowywane w pliku Assets.xcassets w Twoim projekcie. Aby

wykorzystać dane zawarte w ExploreData.plist, musisz utworzyć strukturę przechowującą je w aplikacji, tak aby instancja ExploreViewController mogła uzyskać do nich później dostęp. Utworzysz go w następnej sekcji.

Tworzenie struktury reprezentującej kuchnię

Aby utworzyć obiekt modelu, który może reprezentować kuchnię w aplikacji, dodasz nowy plik do projektu ExploreItem i zadeklarujesz strukturę ExploreItem zawierającą właściwości nazwy i obrazu kuchni. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift, a następnie kliknij Dalej.

3. Nazwij plik ExploreItem i kliknij Utwórz. Pojawi się w nawigatorze projektu, a jego zawartość w obszarze Edytora. Jedyną linią w tym pliku jest instrukcja importu.

4. Dodaj do pliku następujący kod, aby zadeklarować strukturę o nazwie ExploreItem:

```
struct ExploreItem {

}
```

5. Dodaj następujący kod przed ostatnim nawiasem klamrowym, aby dodać dwie opcjonalne właściwości String do struktury ExploreItem:

```
struct ExploreItem {

let name: String?

let image: String?

}
```

Właściwość name będzie przechowywać nazwę kuchni, a właściwość image będzie przechowywać nazwę pliku obrazu z pliku Assets.xcassets.

Struktury automatycznie otrzymują domyślny inicjator. Możesz utworzyć instancję ExploreItem, używając następującej instrukcji:

```
let myExploreItem = ExploreItem(name:"name",

image:"image")
```

Jednak nazwa i nazwa pliku obrazu w ExploreData.plist są przechowywane jako elementy słownika, jak pokazano w poniższym przykładzie:

```
["name": "All", "image": "all.png"]
```

Utworzysz niestandardowy inicjator, który przyjmuje słownik jako parametr i przypisuje wartości uzyskane z elementów słownika do właściwości w instancji ExploreItem. Użyjesz rozszerzenia, aby dodać ten niestandardowy inicjator do struktury ExploreItem

Wykonaj następujące kroki:

1. Wpisz następujące polecenie po deklaracji struktury ExploreItem, aby dodać rozszerzenie:

```
extension ExploreItem {
```

}

Użyjesz tego rozszerzenia, aby dodać metodę inicjatora do struktury ExploreItem.

2. Dodaj następujący wpis między nawiasami klamrowymi rozszerzenia, aby zadeklarować niestandardową metodę inicjującą:

init(dict: [String: String]) {

}

Zignoruj wyświetlony błąd, ponieważ wkrótce go naprawisz. Ten inicjator przyjmuje słownik jako argument. Należy pamiętać, że zarówno klucz, jak i wartość są ciągami znaków.

3. Dodaj następujący wpis pomiędzy nawiasami klamrowymi inicjatora:

self.name = dict["name"]

self.image = dict["image"]

Spowoduje to przypisanie wartości elementu słownika z kluczem „name" do właściwości name, a wartość elementu słownika z kluczem „image" do właściwości image.

Gotowe rozszerzenie powinno wyglądać następująco:

extension ExploreItem {

init(dict: [String: String]) {

self.name = dict["name"]

self.image = dict["image"]

}

}

W tym momencie masz strukturę ExploreItem z dwiema właściwościami String, nazwą i obrazem. Tworząc instancję tej struktury, przekażesz słownik zawierający elementy z kluczem typu String i wartością typu AnyObject. Wartość klucza nazwy zostanie przypisana do właściwości name, a wartość klucza obrazu zostanie przypisana do właściwości image. W następnej sekcji zaimplementujesz klasę menedżera danych. Spowoduje to odczytanie tablicy słowników z pliku ExploreData.plist i przypisanie wartości elementów słownika do instancji ExploreItem.

Implementacja klasy menedżera danych do odczytu danych z pliku .plist

Do swojego projektu dodałeś plik .plist ExploreData.plist zawierający dane o kuchni i utworzyłeś strukturę ExploreItem, w której można przechowywać szczegółowe informacje o każdej kuchni. Teraz musisz utworzyć nową klasę ExploreDataManager, która będzie w stanie odczytać dane z pliku .plist i zapisać je w tablicy instancji ExploreItem. Tę klasę będziesz nazywał klasą menedżera danych. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift, a następnie kliknij Dalej.

3. Nazwij plik ExploreDataManager i kliknij Utwórz, aby wyświetlić jego zawartość w obszarze Edytor.

4. Wpisz w pliku następujący kod, aby zadeklarować klasę ExploreDataManager:

```
class ExploreDataManager {

}
```

5. Wpisz następujący kod w nawiasach klamrowych. Implementuje to metodę loadingData(), która odczyta zawartość pliku ExploreData.plist i zwróci tablicę słowników:

```
private func loadData() -> [[String: String]] {

let decoder = PropertyListDecoder()

if let path = Bundle.main.path(forResource:

"ExploreData", ofType: "plist"),

let exploreData = FileManager.default.contents(

atPath: path),

let exploreItems = try? decoder.decode([[String:

String]].self, from: exploreData) {

return exploreItems

}

return [[:]]

}
```

Podzielmy tę metodę na części:

private

Słowo kluczowe private oznacza, że metody można używać tylko w obrębie tej klasy.

func loadingData() -> [[String: String]]

Deklaracja metody LoadData() nie zawiera argumentów i zwraca tablicę słowników, a każdy słownik zawiera elementy z kluczem i wartością typu String.

let decoder = PropertyListDecoder()

Klasa PropertyListDecoder jest udostępniana przez firmę Apple. Ta instrukcja tworzy instancję dekodera listy właściwości, która będzie używana do dekodowania danych w pliku ExploreData.plist.

if let path = Bundle.main.path(forResource:

"ExploreData", ofType: "plist"),

Wynikiem tworzenia aplikacji jest folder zawierający wszystkie zasoby aplikacji, zwany pakietem aplikacji. ExploreData.plist znajduje się w tym pakiecie. Ta instrukcja próbuje uzyskać ścieżkę do pliku ExploreData.plist i przypisać ją do stałej ścieżki. Wykorzystuje klasę FileManager dostarczoną przez firmę Apple, która ułatwia pracę z systemem plików.

let exploreData = FileManager.default.contents(

atPath: path),

Ta instrukcja próbuje zapisać plik ExploreData.plist w ścieżce i przypisać go do stałej exploreData.

let exploreItems = try? decoder.decode([[String:

String]].self, from: exploreData) {

Jeśli klikniesz ExploreData.plist w swoim projekcie, zwróć uwagę, że obiekt na poziomie głównym jest tablicą, a każdy element tablicy jest słownikiem, jak pokazano:

Key	Type	Value
Root	Array	(31 items)
Item 0	Dictionary	(2 items)
name	String	All
image	String	all.png

Ta instrukcja próbuje utworzyć tablicę na podstawie zawartości pliku ExploreData.plist i przypisać ją do stałej exploreItems.

return exploreItems

}

Jeśli opcjonalne powiązanie zakończy się pomyślnie, instrukcja ta zwróci exploreItems jako tablicę słowników, a każdy słownik jest typu [String: String].

return [[:]]

}

Jeśli opcjonalne powiązanie nie powiedzie się, zwracana jest pusta tablica słowników. W tym momencie masz klasę menedżera danych ExploreDataManager zawierającą metodę ładującą dane z pliku ExploreData.plist i przypisując je do tablicy exploreItems. W następnej sekcji przyjrzymy się, jak używać słowników znajdujących się w tej tablicy do inicjowania instancji ExploreItem, które ostatecznie zostaną przekazane do instancji ExploreViewController zarządzającej ekranem Eksploruj.

Używanie menedżera danych do inicjowania instancji ExploreItem

Obecnie masz klasę menedżera danych ExploreDataManager. Ta klasa zawiera metodę LoadData(), która odczytuje dane z ExploreData.plist i zwraca tablicę słowników. Dodasz metodę, która tworzy i inicjuje instancje ExploreItem ze słownikami w tej tablicy. Wykonaj następujące kroki:

1. W definicji klasy ExploreDataManager dodaj następujący kod przed funkcją LoadData()

metoda implementacji metody fetch():

func fetch() {

for data in loadData() {

print(data)

}

}

Ta metoda wywoła funkcję LoadData(), która zwróci tablicę słowników. Następnie używana jest pętla for do wydrukowania zawartości każdego słownika w tablicy w obszarze Debug, aby upewnić się, że plik ExploreData.plist został pomyślnie odczytany.

2. Kliknij plik ExploreViewController w nawigatorze projektu. Dodaj następujący kod do metody viewDidLoad(). Spowoduje to utworzenie instancji ExploreDataManager i wywołanie jej metody fetch(), gdy instancja ExploreViewController wczyta swój widok:

```
override func viewDidLoad() {

super.viewDidLoad()

let manager = ExploreDataManager()
```

Kompiluj i uruchamiaj swoją aplikację. Zawartość pliku ExploreData.plist jest odczytywana i drukowana w obszarze Debugowanie, jak pokazano:

Teraz przypiszesz ciągi nazw i obrazów z każdego słownika w tablicy do instancji ExploreItem.

3. Kliknij plik ExploreDataManager. Dodaj deklarację właściwości, aby przechowywać tablicę instancji ExploreItem tuż przed metodą fetch():

```
private var exploreItems: [ExploreItem] = []
```

Spowoduje to dodanie właściwości exploreItems do klasy ExploreDataManager i przypisanie do niej pustej tablicy.

4. Wewnątrz metody fetch() zastąp instrukcję print() poniższą instrukcją, która inicjuje instancje ExploreItem i dołącza je do tablicy exploreItems:

```
exploreItems.append(ExploreItem(dict: data))
```

Niestandardowy inicjator w klasie ExploreItem przypisuje ciągi nazw i obrazów w każdym słowniku odczytanym z ExploreData.plist do właściwości name i image instancji ExploreData.

5. Sprawdź, czy zawartość pliku ExploreDataManager wygląda następująco:

```
import Foundation

class ExploreDataManager {

private var exploreItems: [ExploreItem] = []

func fetch() {

for data in loadData() {

exploreItems.append(ExploreItem(dict:
```

```
data))
}
}
private func loadData() -> [[String: String]] {
let decoder = PropertyListDecoder()
if let path = Bundle.main.path(forResource:
"ExploreData", ofType: "plist"),
let exploreData =
FileManager.default.contents(atPath: path),
let exploreItems = try?
decoder.decode([[String: String]].self,
from: exploreData) {
return exploreItems
}
return [[:]]
}
}
```

W tym momencie masz klasę ExploreDataManager, która odczytuje dane z ExploreData.plist i przechowuje je w exploreItems, czyli tablicy instancji ExploreItem. Ta tablica będzie źródłem danych dla widoku kolekcji zarządzanego przez instancję ExploreViewController. Informacje o kuchni, które zawiera, zostaną ostatecznie wyświetlone na ekranie Eksploruj. Zobaczmy, jak możesz to zrobić w następnej sekcji.

Wyświetlanie danych w widoku kolekcji

Zaimplementowałeś klasę menedżera danych ExploreDataManager, która odczytuje dane dotyczące kuchni z pliku ExploreData.plist i przechowuje je w tablicy instancji ExploreItem. Teraz zmodyfikujesz klasę ExploreViewController, aby używać tej tablicy jako źródła danych dla widoku kolekcji na ekranie Eksploruj. Obecnie widok kolekcji na ekranie Eksploruj wyświetla 20 komórek widoku kolekcji, a każda komórka zawiera pusty widok obrazu i etykietę. Potrzebujesz sposobu na ustawienie wartości widoku obrazu i etykiety w komórkach, dlatego utworzysz w tym celu nową klasę ExploreCell. Następnie możesz skonfigurować kontroler widoku dla widoku kolekcji, ExploreViewController, aby uzyskać szczegółowe informacje o kuchni z instancji ExploreDataManager i udostępnić je widokowi kolekcji w celu wyświetlenia. Aby utworzyć ExploreCell, wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Eksploruj w nawigatorze projektu i wybierz opcję Nowa grupa.

2. Zmień nazwę nowego widoku grupowego.

3. Kliknij prawym przyciskiem myszy folder Widok i wybierz Nowy plik.

4. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch, a następnie kliknij Dalej.

5. Skonfiguruj klasę jak pokazano poniżej:

Zajęcia: ExploreCell

Podklasa: UICollectionViewCell

Utwórz także XIB: Niezaznaczone

Język: Swift

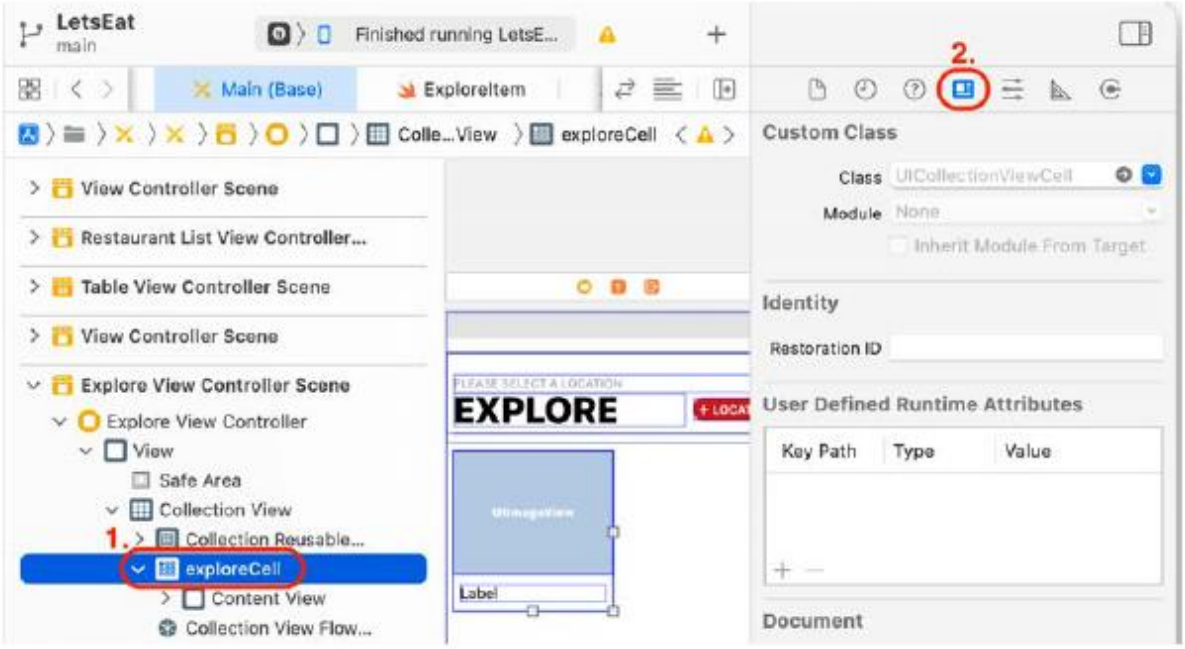
Kliknij Następny.

6. Kliknij Utwórz.

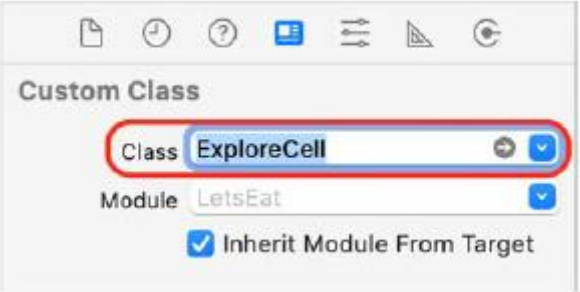
7. Do Twojego projektu zostanie dodany nowy plik ExploreCell. Wewnątrz zobaczysz następujące informacje:

```
import UIKit

class ExploreCell: UICollectionViewCell {

}
```

8. Teraz przypiszesz klasę ExploreCell jako tożsamość komórki widoku kolekcji exploreCell. Kliknij plik głównego scenorysu w nawigatorze projektu i kliknij exploreCell wewnątrz sceny Eksploruj kontroler widoku w konspekcie dokumentu. Kliknij przycisk Inspektora tożsamości:



9. W sekcji Klasa niestandardowa ustaw opcję Klasa na ExploreCell. Spowoduje to ustawienie instancji ExploreCell jako menedżera ExploreCell. Po zakończeniu naciśnij klawisz Return:

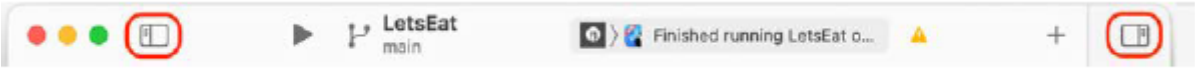


Właśnie zadeklarowałeś i zdefiniowałeś klasę ExploreCell i przypisałeś ją jako menedżera komórki widoku kolekcji ExploreCell. Teraz utworzysz w tej klasie punkty sprzedaży, które będą połączone z widokiem obrazu i etykietą w komórce widoku kolekcji exploreCell, dzięki czemu będziesz mógł kontrolować to, co wyświetlają.

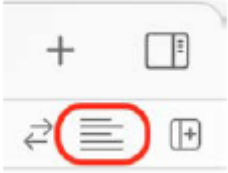
Podłączanie gniazd w exploreCell

Aby zarządzać tym, co jest wyświetlane w komórkach widoku kolekcji na ekranie Eksploruj, połącz widok obrazu i etykietę w komórce widoku kolekcji ExploreCell z gniazdami w klasie ExploreCell. Używasz do tego asystenta edytora. Wykonaj następujące kroki:

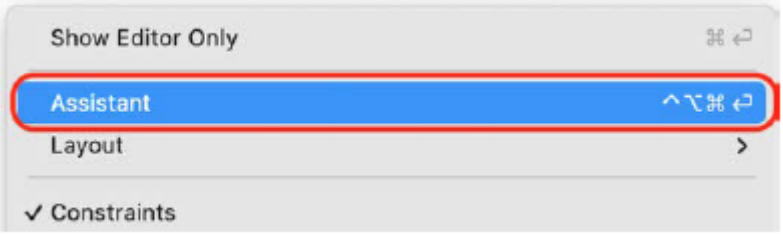
1. Kliknij przyciski Nawigator i Inspektor, aby ukryć obszary Nawigatora i Inspektora. Dzięki temu zyskasz więcej miejsca do pracy:



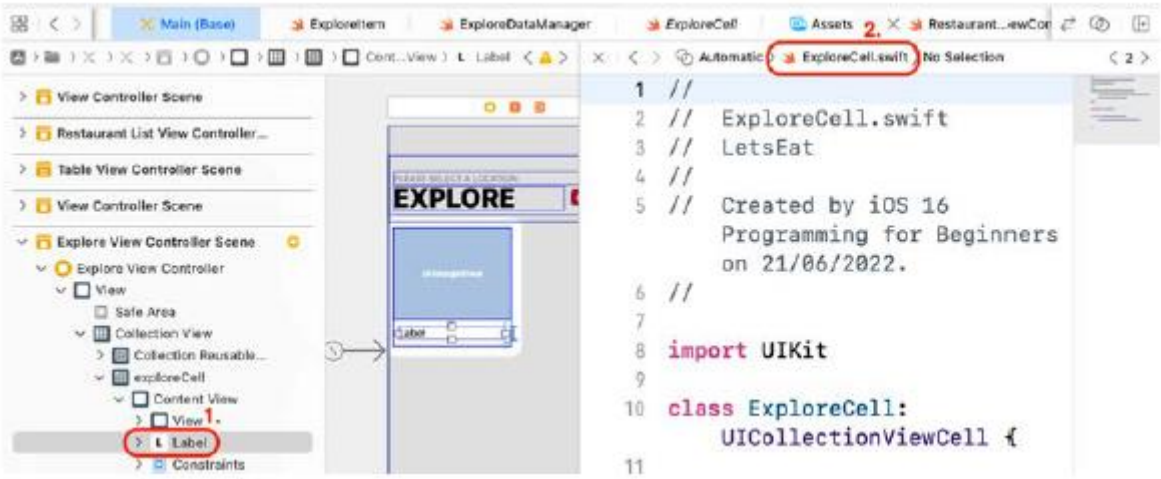
2. Kliknij przycisk Dostosuj opcje edytora, aby wyświetlić menu:



3. Wybierz z menu opcję Asystent, aby wyświetlić asystenta edytora:



4. Aby utworzyć outlety dla klasy ExploreCell, ścieżkę asystenta edytora należy ustawić na Automatic > ExploreCell.swift. Jeśli nie widzisz pliku ExploreCell.swift w ścieżce, wybierz Etykietę komórki widoku kolekcji ExploreCell w konspekcie dokumentu i wybierz opcję ExploreCell. Swift z menu rozwijanego ścieżki asystenta edytora:

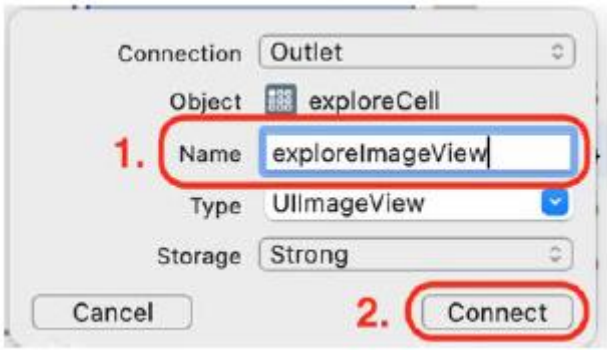


5. Ctrl + Przeciągnij element Etykieta w komórce widoku kolekcji do przestrzeni pomiędzy nawiasami klamrowymi, jak pokazano. To tworzy dla niego ujście:

6. W wyskakującym oknie dialogowym wpisz exploreNameLabel w polu Nazwa, aby ustawić nazwę punktu sprzedaży:

7. Ctrl + Przeciągnij element UIImageView w komórce widoku kolekcji do miejsca tuż za właściwością exploreNameLabel. To tworzy dla niego ujście:

8. W wyskakującym oknie dialogowym wpisz exploreImageView w polu Nazwa, aby ustawić nazwę punktu sprzedaży:



9. Wyjścia exploreNameLabel i exploreImageView zostały dodane do klasy ExploreCell i połączone z elementami widoku obrazu i etykiety komórki widoku kolekcji exploreCell, jak pokazano:

```
import UIKit

class ExploreCell:
    UICollectionViewCell {

    @IBOutlet var
        exploreNameLabel:
        UILabel!

    @IBOutlet var
        exploreImageView:
        UIImageView!
}
```

10. Kliknij przycisk x, aby zamknąć edytor asystenta:

Komórka widoku kolekcji exploreCell w głównym pliku scenorysu została teraz skonfigurowana z klasą ExploreCell. Utworzono i przypisano także wyjścia dla widoku obrazu i etykiety komórki widoku kolekcji. Teraz możesz ustawić punkty eksploracjiNameLabel i exploreImageView w instancji ExploreCell tak, aby po uruchomieniu aplikacji wyświetlał obraz i nazwę kuchni w każdej komórce. Poprawność podłączenia gniazd możesz sprawdzić w Inspektorze połączeń.

W następnej sekcji dodasz kod do ExploreDataManager, aby podać liczbę komórek wyświetlanych przez CollectionView i udostępnisz instancję ExploreItem, której właściwości zostaną użyte do określenia, jaki obraz i etykieta będą wyświetlane w komórce.

Implementacja dodatkowych metod zarządzania danymi

Jak dowiedziałeś się z rozdziału 14, Pierwsze kroki z MVC i widokami kolekcji, widok kolekcji musi wiedzieć, ile komórek wyświetlić i co umieścić w każdej komórce. Do klasy ExploreDataManager dodasz dwie metody, które podają liczbę instancji xploreItem w tablicy exploreItems i zwrócą instancję ExploreItem o określonym indeksie tablicy. Kliknij plik ExploreDataManager w nawigatorze projektu i dodaj te dwie metody po metodzie loadingData():

```
func numberOfExploreItems() -> Int {

exploreItems.count

}

func exploreItem(at index: Int) -> ExploreItem {

exploreItems[index]

}
```

Pierwsza metoda, numberOfExploreItems(), określi liczbę komórek wyświetlanych w widoku kolekcji.

Druga metoda, exploreItem(at:), zwróci instancję ExploreItem odpowiadającą pozycji komórki w widoku kolekcji. W następnej sekcji zaktualizujesz metody źródła danych widoku kolekcji w klasie ExploreViewController, aby podać liczbę komórek wyświetlanych w widoku kolekcji oraz podać nazwę kuchni i obraz dla każdej komórki.

Aktualizowanie metod źródła danych w ExploreViewController

Metody źródła danych w klasie ExploreViewController są obecnie ustawione tak, aby wyświetlały 20 komórek, a każda komórka zawiera pusty widok obrazu i etykietę. Zaktualizujesz je, aby uzyskać liczbę komórek do wyświetlenia i dane do umieszczenia w każdej komórce z instancji ExploreDataManager. Wykonaj następujące kroki:

1. Kliknij plik ExploreViewController i znajdź metodę viewDidLoad(). To powinno wyglądać tak:

```
override func viewDidLoad() {

super.viewDidLoad()

let manager = ExploreDataManager()

manager.fetch()

}
```

Oznacza to, że instancja ExploreDataManager jest dostępna tylko w ramach metody viewDidLoad(). Musisz udostępnić je całej klasie.

2. Przesuń wiersz let manager = ExploreDataManager() tuż pod deklarację właściwości CollectionView, aby udostępnić instancję ExploreDataManager wszystkim metodom w obrębie

Klasa ExploreViewController, jak pokazano:

```
@IBOutlet var collectionView: UICollectionView!

let manager = ExploreDataManager()
```

3. Zaktualizuj CollectionView(_:numberOfItemsInSection:) jak pokazano. Spowoduje to, że widok kolekcji wyświetli komórkę dla każdego elementu tablicy items:

```
func collectionView(_ collectionView: UICollectionView,

numberOfItemsInSection section: Int) -> Int {

manager.numberOfExploreItems()

}
```

4. Zaktualizuj CollectionView(_:cellForItemAt:) jak pokazano, aby ustawić widok obrazu i etykietę dla każdej komórki, korzystając z danych z odpowiedniego elementu w tablicy items:

```
func collectionView(_ collectionView: UICollectionView, cellForItemAt

indexPath: IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(

withReuseIdentifier: "exploreCell", for:

indexPath) as! ExploreCell

let exploreItem = manager.exploreItem(at:

indexPath.row)

cell.exploreNameLabel.text = exploreItem.name

cell.exploreImageView.image = UIImage(named:

exploreItem.image!)

return cell
```

}

Rozbijmy to:

let cell = collectionView.dequeueReusableCell(

withReuseIdentifier:"exploreCell", for: indexPath)

as! ExploreCell

Określa, że komórka, która jest usuwana z kolejki, jest instancją ExploreCell.

let exploreItem = manager.exploreItem(at: indexPath.row)

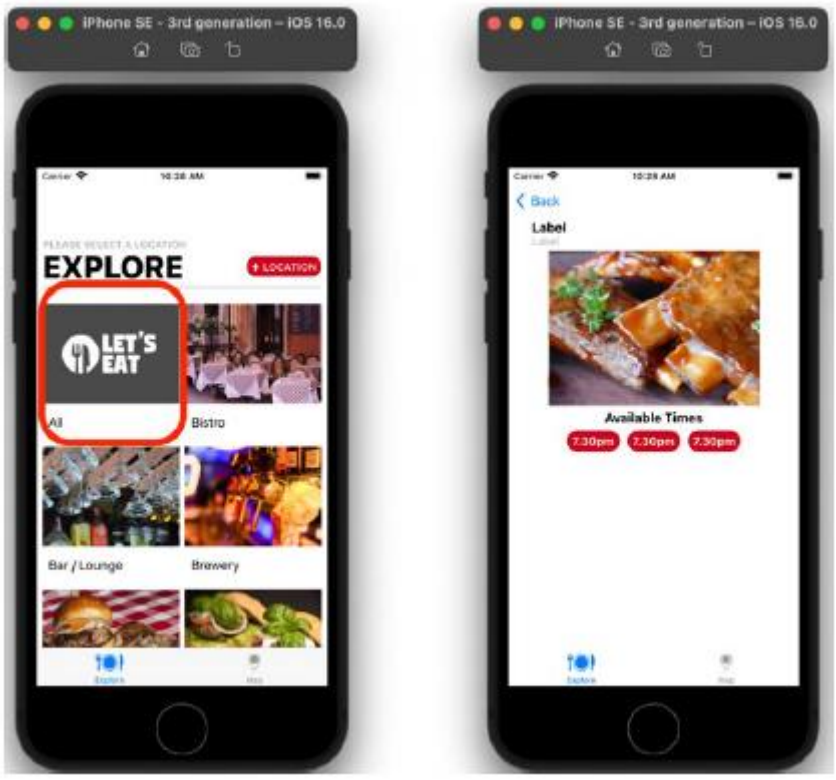
Pobiera wystąpienie ExploreItem odpowiadające bieżącej komórce w widoku kolekcji. Innymi słowy, pierwsza komórka w widoku kolekcji odpowiada pierwszej instancji ExploreItem w tablicy exploreItems, druga komórka odpowiada drugiej instancji ExploreItem i tak dalej.

cell.exploreNameLabel.text = exploreItem.name

Ustawia właściwość tekstową komórki nameLabel na nazwę instancji ExploreItem.

cell.exploreImageView.image = UIImage(named: exploreItem.image!)

Pobiera ciąg obrazu z instancji ExploreItem, pobiera odpowiedni obraz z pliku Assets.xcassets i przypisuje go do obrazu właściwości imgExplore komórki. Kompiluj i uruchamiaj aplikację. Widok kolekcji zobaczysz na ekranie Eksploruj, wyświetlając obrazy i teksty dotyczące różnych kuchni. Dotknięcie komórki widoku kolekcji spowoduje wyświetlenie ekranu Lista restauracji:



W tym momencie ekran Eksploruj wyświetla obrazy i tekst różnych kuchni na podstawie danych uzyskanych z pliku ExploreData.plist. W rozdziale 18, Pierwsze kroki z plikami JSON, zmodyfikujesz klasę RestaurantListViewController, aby na ekranie Lista restauracji wyświetlana była lista restauracji

oferujących wybraną kuchnię. Ale zanim to zrobisz, musisz ustawić lokalizację na ekranie Lokalizacje, który wyświetli listę wszystkich dostępnych restauracji w tej lokalizacji. Zostanie to omówione w następnym rozdziale.

Streszczenie

Dodałeś do swojego projektu plik listy właściwości ExploreData.plist. Zaimplementowałeś strukturę ExploreItem, w tym obiekty modelu dla ekranu Eksploruj. Utworzono klasę menedżera danych ExploreDataManager do odczytywania danych z ExploreData.plist, umieszczania danych w tablicy instancji ExploreItem i udostępniania ich do ExploreViewController. Utworzyłeś klasę dla komórki widoku kolekcji exploreCell. Na koniec skonfigurowano metody źródła danych w klasie ExploreViewController tak, aby korzystały z danych z tablicy instancji ExploreItem w celu wypełnienia widoku kolekcji, a ekran Eksploruj wyświetla teraz listę kuchni. Dobra robota! Teraz wiesz, jak dostarczać dane do aplikacji przy użyciu plików .plist, tworzyć obiekty modelu, tworzyć klasy menedżerów danych, które ładują pliki .plist do obiektów modelu, konfigurować widoki kolekcji w celu wyświetlania załadowanych danych oraz konfigurować kontrolery widoku dla widoków kolekcji . Będzie to przydatne, jeśli chcesz tworzyć własne aplikacje korzystające z widoków kolekcji. W następnym rozdziale przyjrzysz się widokom tabel, które pod pewnymi względami są podobne do widoków kolekcji, oraz skonfigurujesz ekran Lokalizacje tak, aby wyświetlał listę lokalizacji w widoku tabeli po dotknięciu przycisku LOKALIZACJA na ekranie Eksploruj.

Pierwsze kroki z widokami tabeli

Skonfigurowałeś klasę ExploreViewController, kontroler widoku dla ekranu Eksploruj, aby wyświetlać informacje o kuchni dostarczane przez ExploreData.plist w widoku kolekcji. W tym rozdziale zaczniesz od poznania widoków tabeli i kontrolerów widoku tabeli. Zaimplementujesz widok tabeli programowo (co oznacza wdrożenie go przy użyciu kodu zamiast scenorysów) przy użyciu placu zabaw, aby zrozumieć, jak działają widoki tabeli. Następnie utworzysz kontroler widoku tabeli dla ekranu Lokalizacje, utworzysz od podstaw plik .plist do przechowywania listy lokalizacji, utworzysz klasę menedżera danych do odczytywania danych z pliku .plist i skonfigurujesz kontroler widoku tabeli tak, aby pobierz dane od menedżera danych i udostępnij je widokowi tabeli. Na ekranie Lokalizacje wyświetli się lista lokalizacji restauracji. Pod koniec dowiesz się, jak tworzyć pliki .plist i jak implementować kontrolery widoku tabeli. Umożliwi to implementację plików .plist i widoków tabel korzystających z plików .plist jako źródła danych we własnych aplikacjach.

Zrozumienie widoków tabel

Aplikacja Let's Eat wykorzystuje widok tabeli na ekranie Lokalizacje do wyświetlania listy lokalizacji restauracji. Widok tabeli przedstawia komórki widoku tabeli przy użyciu wierszy ułożonych w jednej kolumnie. Dane wyświetlane w widoku tabeli są zwykle dostarczane przez kontroler widoku. Kontroler widoku dostarczający dane dla widoku tabeli musi być zgodny z protokołem UITableViewDataSource. Protokół ten deklaruje listę metod, które informują widok tabeli, ile komórek wyświetlić i co wyświetlić w każdej komórce. Aby zapewnić interakcję użytkownika, kontroler widoku dla widoku tabeli musi być również zgodny z protokołem UITableViewDelegate, który deklaruje listę metod wyzwalanych, gdy użytkownik wchodzi w interakcję z widokiem tabeli. Aby dowiedzieć się, jak działają widoki tabeli, zaimplementuj podklasę kontrolera widoku, która kontroluje widok tabeli na placu zabaw TableViewBasics. Ponieważ na placu zabaw nie ma scenorysu, nie można dodawać elementów interfejsu użytkownika za pomocą biblioteki, tak jak to robiłeś w poprzednich rozdziałach. Zamiast tego zrobisz wszystko programowo. Wykonaj następujące kroki:

1. Otwórz plac zabaw TableViewBasics, który utworzyłeś na początku tego rozdziału. Na samej górze placu zabaw usuń instrukcję var i dodaj instrukcję import PlaygroundSupport. Twój plac zabaw powinien teraz zawierać następujące elementy:

```
import UIKit

import PlaygroundSupport
```

Pierwsza instrukcja importu importuje interfejs API do tworzenia aplikacji na iOS. Drugi umożliwia wyświetlanie na placu zabaw podglądu na żywo, którego użyjesz do wyświetlenia widoku stołu.

2. Dodaj następujący kod po instrukcjach import, aby zadeklarować klasę TableViewExampleController:

```
class TableViewExampleController: UIViewController {

}
```

Ta klasa jest podklasą UIViewController, klasy udostępnianej przez firmę Apple do zarządzania widokami na ekranie.

3. Dodaj następujący kod w nawiasach klamrowych, aby zadeklarować właściwość widoku tabeli i właściwość tablicy do klasy TableViewExampleController:

```
class TableViewExampleController: UIViewController {
```

var tableView: UITableView?

var names: [String] = ["Divij","Aamir","Shubham"]

}

Właściwość tableView jest właściwością opcjonalną, do której zostanie przypisana instancja UITableView. Tablica nazw jest obiektem modelu, który będzie używany do dostarczania danych do widoku tabeli. Widok tabeli wyświetla na ekranie pojedynczą kolumnę wierszy, a każdy wiersz zawiera instancję komórki widoku tabeli. Podobnie jak widoki kolekcji, widoki tabel muszą wiedzieć, ile wierszy wyświetlić i co umieścić w każdym wierszu. Aby udostępnić te informacje widokowi tabeli, należy dostosować klasę TableViewExampleController do protokołu UITableViewDataSourceProtocol. Ten protokół ma dwie wymagane metody:

• tableview(_:numberOfRowsInSection:) jest wywoływana przez widok tabeli w celu określenia liczby wyświetlanych komórek widoku tabeli.

• tableView(_:cellForRowAt:) jest wywoływana przez widok tabeli w celu określenia, co ma być wyświetlane w każdej komórce widoku tabeli.

Dodajmy trochę kodu, aby klasa TableViewExampleController była zgodna z protokołem UITableViewDataSource. Wykonaj następujące kroki:

1. Aby klasa TableViewExampleController przyjęła protokół UITableViewDataSource, wpisz przecinek po nazwie nadklasy, a następnie wpisz UITableViewDataSource. Twój kod powinien wyglądać tak:

class TableViewExampleController: UIViewController, UITableViewDataSource

{

2. Pojawi się błąd, ponieważ nie zaimplementowałeś dwóch wymaganych metod. Kliknij ikonę błędu:

```
3 import PlaygroundSupport
4
5 class TableViewExampleController:
      UIViewController,
      UITableViewDataSource {
```

3. Pojawiający się komunikat o błędzie informuje, że brakuje wymaganych metod dla protokołu UITableViewDataSource. Kliknij przycisk Napraw, aby dodać wymagane kody pośredniczące metod do klasy:

```
5 class TableViewExampleController:
      UIViewController,
      UITableViewDataSource {
6   Type 'TableViewExampleController' does
7   not conform to protocol
8   'UITableViewDataSource'
    Do you want to add protocol stubs?   Fix
10
```

4. Sprawdź, czy Twój kod wygląda następująco:

class TableViewExampleController: UIViewController,

```
UITableViewDataSource {

func tableView(_ tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {

code

}

func tableView(_ tableView: UITableView,

cellForRowAt indexPath: IndexPath) ->

UITableViewCell {

code

}

var tableView: UITableView?

var names: [String] = ["Divij","Aamir","Shubham"]

}
```

5. Konwencja w definicji klasy stanowi, że właściwości deklarowane są na górze, przed deklaracjami metod. Zmień układ kodu tak, aby deklaracje właściwości znajdowały się na górze, w następujący sposób:

```
class TableViewExampleController: UIViewController,

UITableViewDataSource {

var tableView: UITableView?

var names: [String] = ["Divij","Aamir","Shubham"]

func tableView(_ tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {
```

6. Aby widok tabeli wyświetlał wiersz dla każdego elementu wewnątrz tablicy Names, kliknij kod słowa w definicji metody tableView(_:numberOfRowsInSection:) i wpisz Names.count. Gotowa metoda powinna wyglądać następująco:

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section:

Int) -> Int {

names.count

}
```

Names.count zwraca liczbę elementów wewnątrz tablicy Names. Ponieważ znajdują się w nim trzy nazwy, widok tabeli spowoduje wyświetlenie trzech wierszy.

7. Aby widok tabeli wyświetlał nazwy w każdej komórce, kliknij kod słowny w definicji metody tableView(_:cellForRowAt:) i wpisz następujące polecenie:

Rozbijmy to:

let cell = tableView.dequeueReusableCell(

withIdentifier: "Cell", for: indexPath)

Spowoduje to utworzenie nowej komórki widoku tabeli lub ponowne wykorzystanie istniejącej komórki widoku tabeli i przypisanie jej do komórki. Wyobraź sobie, że masz 1000 pozycji do wyświetlenia w widoku tabeli. Nie potrzebujesz 1000 wierszy zawierających 1000 komórek widoku tabeli — wystarczy ich tylko tyle, aby wypełnić ekran. Komórki widoku tabeli przewijane u góry ekranu można ponownie wykorzystać do wyświetlenia elementów wyświetlanych u dołu ekranu. Aby mieć pewność, że używasz właściwego typu komórki, ustawiasz identyfikator ponownego użycia na Komórka. To ponowne wykorzystanie identyfikatora zostanie później zarejestrowane w widoku tabeli.

let name = names[indexPath.row]

Wartość indeksPath lokalizuje wiersz w widoku tabeli. Pierwszy wiersz zawiera indeksPath zawierający sekcję 0 i wiersz 0. IndexPath.row zwraca 0 dla pierwszego wiersza, więc name zostaje przypisane do pierwszego elementu w tablicy Names.

cell.textLabel?.text = name

Spowoduje to przypisanie nazwy do właściwości tekstowej textLabel komórki widoku tabeli.

return cell

Spowoduje to zwrócenie komórki widoku tabeli, która zostanie następnie wyświetlona na ekranie.

Ta metoda jest wykonywana dla każdego wiersza w widoku tabeli.

8. Sprawdź, czy klasa TableViewExampleController wygląda następująco:

class TableViewExampleController: UIViewController,

UITableViewDataSource {

var tableView: UITableView?

var names: [String] = ["Divij","Aamir","Shubham"]

func tableView(_ tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {

names.count

}

func tableView(_ tableView: UITableView,

cellForRowAt indexPath: IndexPath) ->

UITableViewCell {

let cell = tableView.dequeueReusableCell(

withIdentifier: "Cell", for:indexPath)

```
let name = names[indexPath.row]

cell.textLabel?.text = name

return cell

}

}
```

Zakończyłeś implementację klasy TableViewExampleController. Teraz napiszesz metodę createTableView(), aby utworzyć jej instancję. Wpisz następujący kod po deklaracjach właściwości, aby zadeklarować i zdefiniować metodę createTableView():

```
func createTableView() {

self.tableView = UITableView(frame: CGRect(x: 0, y: 0,

width: self.view.frame.width,

height: self.view.frame.height))

self.tableView?.dataSource = self

self.tableView?.backgroundColor = .white

self.tableView?.register(UITableViewCell.self,

forCellReuseIdentifier: "Cell")

self.view.addSubview(self.tableView!)

}
```

Rozbijmy to:

```
self.tableView = UITableView(frame: CGRect(x: 0, y: 0, width: self.view.frame.

width, height: self.view.frame.height))
```

Spowoduje to utworzenie nowej instancji UITableView o dokładnie takim samym rozmiarze jak otaczający ją widok i przypisanie jej do tableView.

```
self.tableView?.dataSource = self
```

Informuje to widok tabeli, że jego źródłem danych jest instancja TableViewExampleController.

```
self.tableView?.backgroundColor = .white
```

Spowoduje to ustawienie koloru tła widoku tabeli na biały.

```
self.tableView?.register(UITableViewCell.self, forCellReuseIdentifier: „Cell”)
```

Spowoduje to ustawienie identyfikatora ponownego wykorzystania komórek widoku tabeli na „Komórka”. Ten identyfikator ponownego użycia zostanie użyty w metodzie tableView(_:cellForRowAt:) w celu zidentyfikowania komórek, które można ponownie wykorzystać.

```
self.view.addSubview(self.tableView!)
```

Spowoduje to dodanie widoku tabeli jako widoku podrzędnego do widoku instancji TableViewExampleController. Teraz musisz wywołać tę metodę. Klasa UIViewController posiada metodę viewDidLoad(), która jest wywoływana po załadowaniu widoku. Ta metoda jest dziedziczona przez klasę TableViewExampleController i należy ją zastąpić, aby wywołać metodę createTableView(). Wykonaj następujące kroki:

1. Wpisz poniższy kod tuż nad metodą createTableView():

```
override func viewDidLoad() {

super.viewDidLoad()

self.view.bounds = CGRect(x: 0, y: 0,

width: 375, height: 667)

createTableView()

}
```

Spowoduje to ustawienie rozmiaru widoku na żywo, utworzenie widoku tabeli i dodanie go jako widoku podrzędnego do widoku instancji TableViewExampleController. Metody źródła danych są następnie wykorzystywane do określenia liczby wyświetlanych komórek widoku tabeli oraz tego, co należy umieścić w każdej komórce widoku tabeli. tableView(_:numberOfRowsInSection:) zwraca 3, więc wyświetlane są trzy wiersze. tableView(_:cellForRowAt:) ustawia tekst każdej komórki na odpowiednią nazwę w tablicy nazw.

2. Sprawdź, czy wypełniony kod wygląda następująco:

```
import UIKit

import PlaygroundSupport

class TableViewExampleController: UIViewController,

UITableViewDataSource {

var tableView: UITableView?

var names: [String] = ["Divij","Aamir","Shubham"]

override func viewDidLoad() {

super.viewDidLoad()

self.view.bounds = CGRect(x: 0, y: 0,

width: 375, height: 667)

createTableView()

}

func createTableView() {

self.tableView = UITableView(frame: CGRect(

x: 0, y: 0,
```

```
width: self.view.frame.width,
height: self.view.frame.height))
self.tableView?.dataSource = self
self.tableView?.backgroundColor = .white
self.tableView?.register(
UITableViewCell.self,
forCellReuseIdentifier: "Cell")
self.view.addSubview(self.tableView!)
}
func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
names.count
}
func tableView(_ tableView: UITableView,
cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
let cell = tableView.dequeueReusableCell(
withIdentifier: "Cell", for:indexPath)
let name = names[indexPath.row]
cell.textLabel?.text = name
return cell
}
}
```

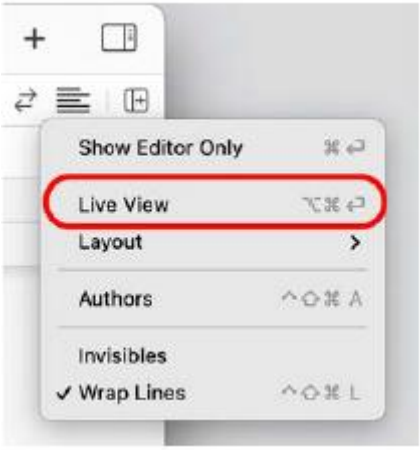
3. Wpisz następujący kod po całym innym kodzie na placu zabaw:

```
PlaygroundPage.current.liveView = TableViewExampleController()
```

To polecenie tworzy instancję TableViewExampleController i wyświetla jej widok w podglądzie na żywo placu zabaw.

4. Uruchom plac zabaw. Jeśli nie widzisz widoku tabeli, kliknij przycisk Dostosuj opcje edytora:

5. Upewnij się, że z wyskakującego menu wybrano opcję Widok na żywo:

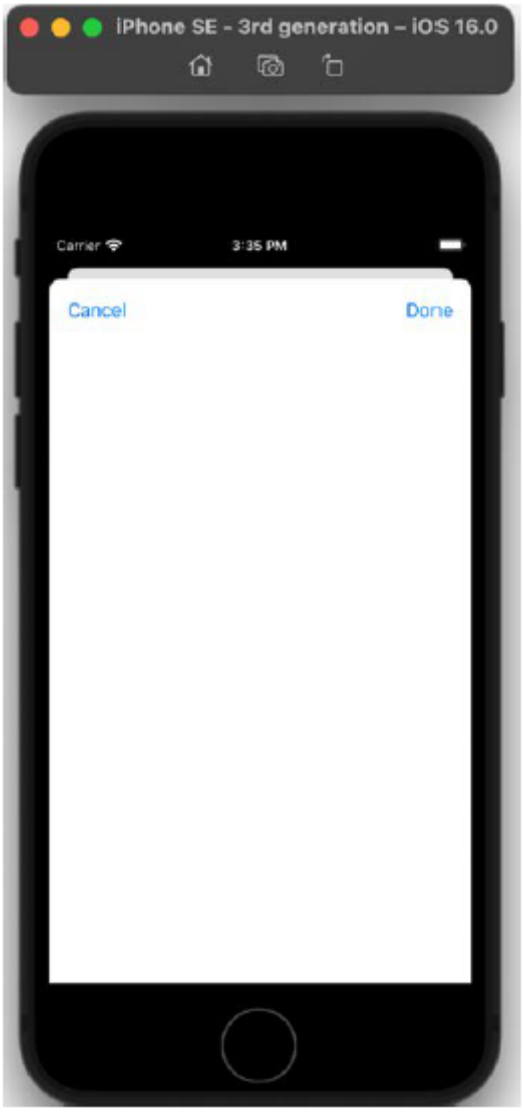


6. Zobaczysz widok tabeli przedstawiający tabelę z trzema wierszami zawierającymi nazwy, jak pokazano:

Świetnie! Skoro już wiesz, jak działają widoki tabel, dokończmy implementację ekranu Lokalizacje. W następnej sekcji zaczniesz od utworzenia kontrolera widoku dla tego ekranu, aby mógł on zarządzać tym, co będzie wyświetlane w widoku tabeli.

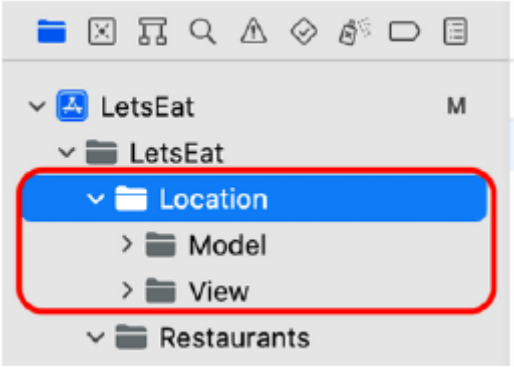
Tworzenie klasy LocationViewController

Jak pokazano w prezentacji aplikacji w Rozdziale 10, Konfigurowanie interfejsu użytkownika, ekran Lokalizacje wyświetla listę lokalizacji w widoku tabeli. Na końcu rozdziału 13, Modyfikowanie i konfigurowanie komórek, skonfigurowałeś ekran Lokalizacje tak, aby wyświetlał widok tabeli i ustawiłeś identyfikator komórek widoku tabeli na LocationCell. Odnosząc się do wzorca projektowego Model-View-Controller (MVC), ukończyłeś wymagane widoki, ale nie ukończyłeś jeszcze kontrolera ani modelu. W tej chwili po kliknięciu przycisku LOKALIZACJA na ekranie Eksploruj wyświetli się pusty widok tabeli:



Utworzysz klasę LocationViewController jako kontroler widoku dla ekranu Lokalizacje, dodasz do niej punkt wyjściowy dla widoku tabeli i skonfigurujesz ją jako źródło danych i delegata widoku tabeli. Wykonaj następujące kroki:

1. Otwórz swój projekt LetsEat z poprzedniego rozdziału. Utwórz nowy folder „Lokalizacja” w swoim projekcie, klikając prawym przyciskiem myszy folder LetsEat i wybierając opcję Nowa grupa. Utwórz dwa kolejne foldery, View i Model, w folderze Location. Kiedy skończysz, zobaczysz następującą strukturę folderów:



2. Kliknij prawym przyciskiem myszy folder Lokalizacja i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

4. Skonfiguruj klasę, podając następujące szczegóły:

Klasa: LocationViewController

Podklasa: UIViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

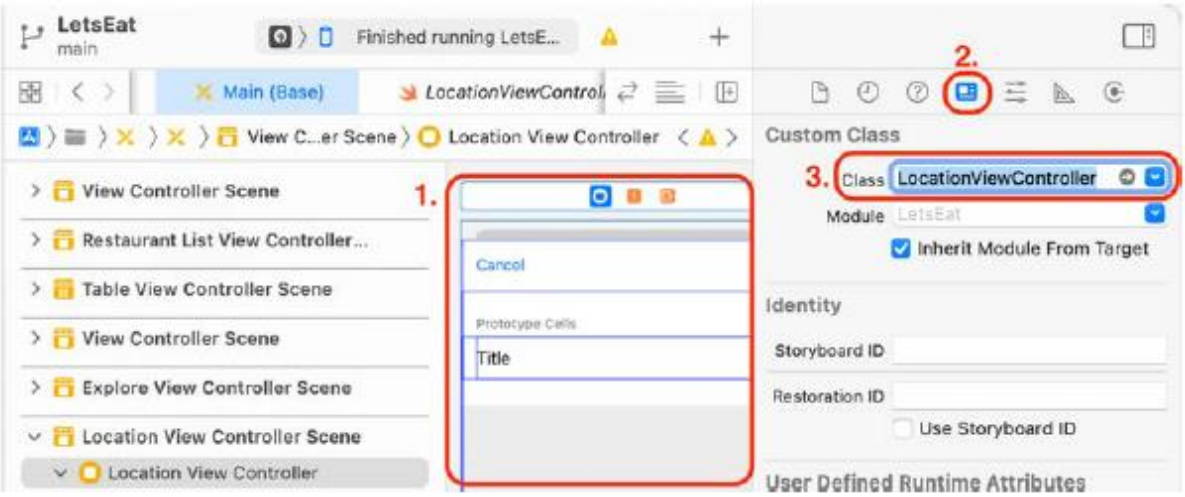
Kliknij Następny.

5. Kliknij Utwórz. Plik LocationViewController pojawi się w nawigatorze projektu.

Utworzono plik LocationViewController, w którym znajduje się deklaracja klasy LocationViewController. Teraz ustawisz tożsamość sceny kontrolera widoku wyświetlanej po dotknięciu przycisku LOKALIZACJA dla tej klasy. Wykonaj następujące kroki:

1. Otwórz plik głównego scenorysu w nawigatorze projektu.

2. Wybierz scenę kontrolera widoku, która wyświetli się po kliknięciu przycisku LOKALIZACJA (jest to ta z przyciskami Anuluj i Gotowe). Kliknij przycisk Inspektora tożsamości i w obszarze Klasa niestandardowa ustaw opcję Klasa na LocationViewController. Uwaga: nazwa sceny zmieni się na Scena kontrolera widoku lokalizacji:

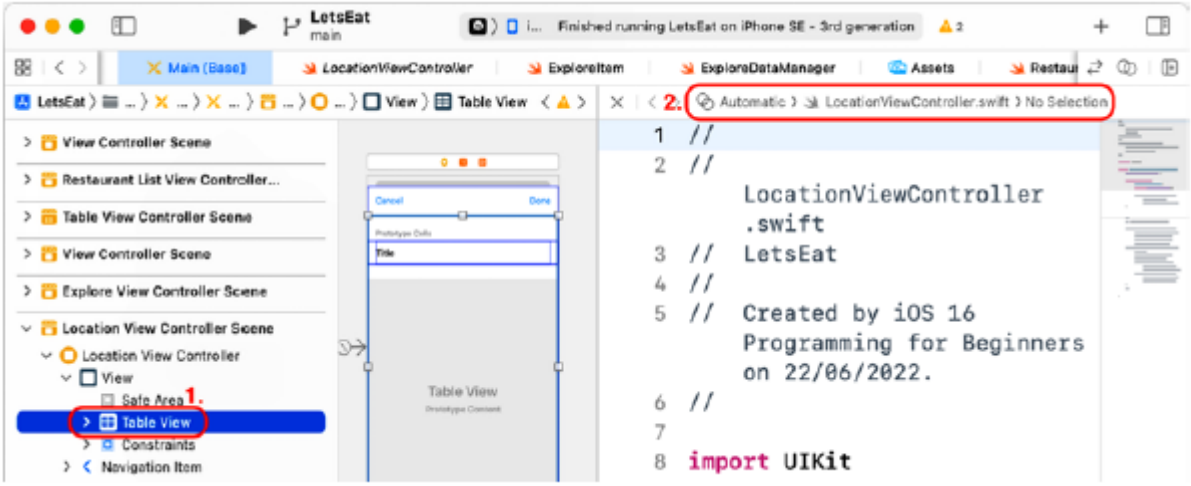


Fajnie! W następnej sekcji połączmy widok tabeli z gniazdem w klasie LocationViewController. W ten sposób instancja LocationViewController dla ekranu Lokalizacje będzie mogła zarządzać widokiem tabeli. Podłączenie widoku tabeli do klasy LocationViewController Obecnie instancja LocationViewController dla ekranu Locations nie ma możliwości komunikacji z znajdującym się w niej widokiem tabeli. Utworzysz nowy outlet w klasie LocationViewController i przypiszesz do niego widok tabeli. Wykonaj następujące kroki:

1. Kliknij przyciski Nawigator i Inspektor, aby w razie potrzeby ukryć obszary Nawigatora i Inspektora.

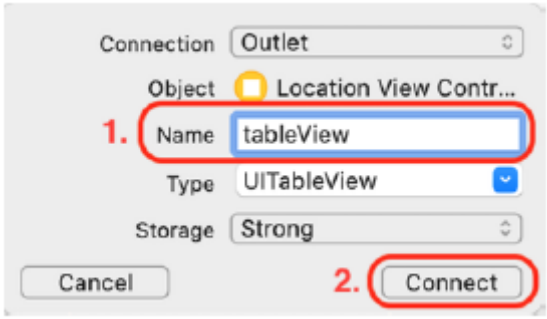
2. Kliknij przycisk Dostosuj opcje edytora i wybierz z menu opcję Asystent.

3. W głównym pliku scenorysu kliknij widok tabeli w konspekcie dokumentu. Edytor asystenta powinien być ustawiony na Automatyczny > LocationViewController.swift, jak pokazano:



4. Ctrl + Przeciągnij z widoku tabeli do miejsca tuż nad viewDidLoad():

5. W wyskakującym menu wpisz tableView w polu Nazwa i kliknij Połącz:



6. Sprawdź, czy wyjście tableView zostało dodane do klasy LocationViewController i połączone z widokiem tabeli w scenorysie:

```
10  class LocationViewController:
        UIViewController {
11
        @IBOutlet var tableView:
            UITableView!
13
14      override func
```

7. Kliknij przycisk x, aby zamknąć edytor asystenta:

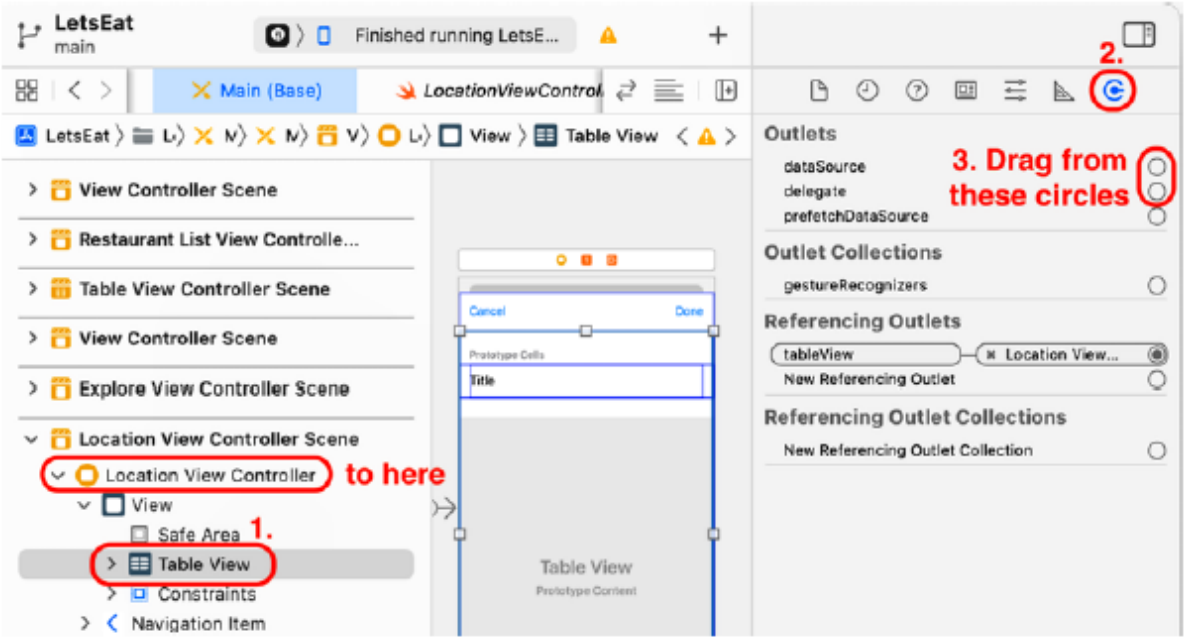
```
Au...tic 〉 LocationViewController.swift 〉 LocationViewController
 6  //
 7
 8  import UIKit
 9
10  class LocationViewController:
        UIViewController {
```

Podłączyłeś widok tabeli do gniazdka w klasie LocationViewController. Aby widok tabeli wyświetlał dane i reagował na interakcję użytkownika, klasa LocationViewController musi być zgodna z protokołami UITableViewDataSource i UITableViewDelegate oraz implementować wymagane metody. Zrobisz to w następnej sekcji. Dodawanie źródła danych i metod delegowania Kontroler widoku dla widoku tabeli musi przyjąć protokoły UITableViewDataSource i UITableViewDelegate, a także zaimplementować wymagane metody, aby umożliwić wyświetlanie danych i interakcję użytkownika. W tej sekcji połączysz LocationViewController ze źródłem danych widoku tabeli i delegujesz punkty sprzedaży oraz zaimplementujesz wymagane metody źródła danych. Metody delegowania zaimplementujesz w rozdziale 18, Pierwsze kroki z plikami JSON. Wykonaj następujące kroki:

1. Kliknij przyciski Nawigator i Inspektor, aby w razie potrzeby wyświetlić obszary Nawigatora i Inspektora.

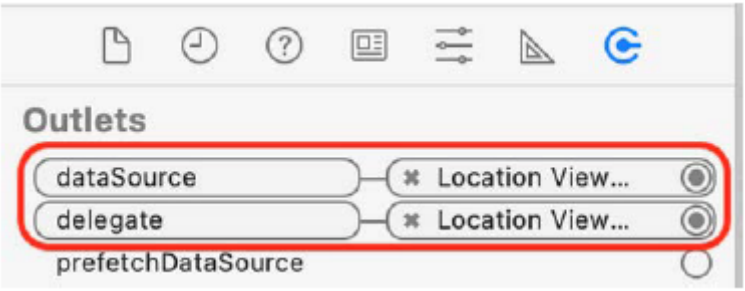
2. Upewnij się, że w głównym pliku scenorysu w konspekcie dokumentu wybrano widok tabeli.

Kliknij przycisk Inspektora połączeń. Kliknij i przeciągnij ze źródła danych i deleguj punkty sprzedaży do ikony LocationViewController w konspekcie dokumentu:



Spowoduje to połączenie widoku tabeli z gniazdami klasy LocationViewController.

3. Sprawdź, czy właściwości dataSource i delegat widoku tabeli zostały połączone z gniazdami w klasie LocationViewController:



Następnie sprawisz, że LocationViewController będzie zgodny z protokołem UITableViewDataSource i zaimplementujesz wymagane metody dla tego protokołu. Wykonaj następujące kroki:

1. Kliknij plik LocationViewController w nawigatorze projektu i usuń cały skomentowany kod z definicji klasy LocationViewController, tak aby pozostało tylko:

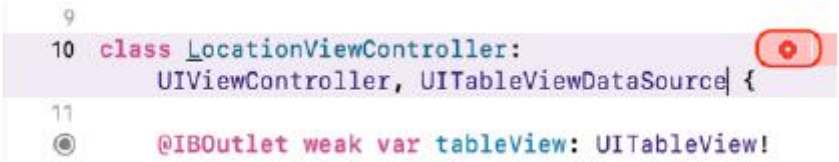
```
class LocationViewController: UIViewController {

@IBOutlet weak var tableView: UITableView!

override func viewDidLoad(){

super.viewDidLoad()

}

}
```

2. Aby LocationViewController przyjął protokół UITableViewDataSource, wpisz przecinek po nazwie nadklasy UIViewController i wpisz UITableViewDataSource. Kiedy skończysz, Twój kod powinien wyglądać następująco:

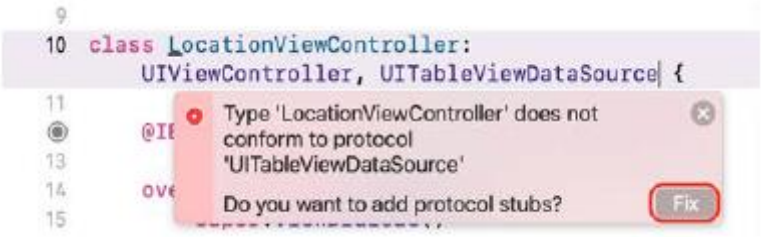
class LocationViewController: UIViewController,

UITableViewDataSource {

3. Pojawi się błąd, ponieważ nie zaimplementowałeś jeszcze dwóch wymaganych metod. Kliknij ikonę błędu, aby wyświetlić komunikat o błędzie:



4. Kliknij przycisk Napraw, aby dodać do klasy wymagane kody pośredniczące metod:



5. Zmień układ kodu tak, aby deklaracje właściwości i metoda viewDidLoad() znajdowały się na górze. Jest to zgodne z ogólnymi konwencjami kodowania dla programistów iOS i ułatwia utrzymanie kodu. Kiedy skończysz, sprawdź, czy Twój kod wygląda następująco:

class LocationViewController: UIViewController,

UITableViewDataSource {

@IBOutlet var tableView: UITableView!

override func viewDidLoad() {

super.viewDidLoad()

}

func tableView(_ tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {

code

}

func tableView(_ tableView: UITableView,

cellForRowAt indexPath: IndexPath) ->

UITableViewCell {

code

}

}

6. Wewnątrz pierwszej wymaganej metody kliknij słowo code i wpisz 10. Spowoduje to wyświetlenie w widoku tabeli 10 wierszy. Kompletna metoda powinna wyglądać następująco:

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section:

Int) -> Int {

10

}
```

7. W drugiej wymaganej metodzie kliknij kod słowa i wpisz następujące polecenie, aby widok tabeli wyświetlał ciąg „Komórka” w każdym wierszu:

```
func tableView(_ tableView: UITableView, cellForRowAt

indexPath: IndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCell(

withIdentifier: "locationCell", for: indexPath)

cell.textLabel?.text = "A Cell"

return cell

}
```

Rozbijmy to:

```
let cell = tableView.dequeueReusableCell(

withIdentifier: "locationCell", for: indexPath)
```

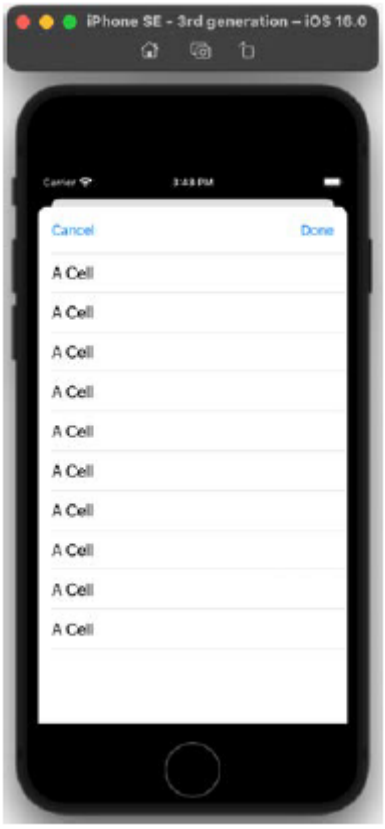
Spowoduje to utworzenie nowej komórki widoku tabeli lub ponowne wykorzystanie istniejącej komórki widoku tabeli z identyfikatorem LocationCell i przypisanie jej do komórki. Ustawiasz ten identyfikator w scenie kontrolera widoku lokalizacji w głównym pliku scenorysu w rozdziale 13, Modyfikowanie i konfigurowanie komórek.

```
cell.textLabel?.text = "A Cell"
```

Spowoduje to przypisanie ciągu „A Cell” do właściwości tekstowej textLabel komórki widoku tabeli.

```
return cell
```

Spowoduje to zwrócenie komórki, która zostanie następnie wyświetlona na ekranie. Proces ten powtarza się dla liczby komórek podanej w pierwszej metodzie, czyli w tym przypadku wynoszącej 10. Zbuduj i uruchom swój projekt. Kliknij przycisk LOKALIZACJA na ekranie Eksploruj. Powinieneś zobaczyć, że widok tabeli wyświetla 10 wierszy, z których każdy zawiera komórkę A, jak pokazano:



Zakończyłeś implementację klasy LocationsViewController i w widoku tabeli wyświetlane są teraz komórki widoku tabeli. Świetnie! Teraz, gdy kontroler widoku widoku tabeli został skonfigurowany, w następnej sekcji utwórzmy kilka obiektów modelu, abyś mógł dostarczyć do niego dane.

Dodawanie danych lokalizacyjnych do widoku tabeli

W tym momencie utworzyłeś i skonfigurowałeś klasę LocationViewController. Instancja tej klasy będzie działać jako źródło danych dla widoku tabeli na ekranie Lokalizacje. Widoki i kontroler dla tego ekranu zostały skonfigurowane, więc teraz utworzysz obiekty modelu, aby widok tabeli wyświetlał listę rzeczywistych lokalizacji. Podobnie jak w poprzednim rozdziale, użyjesz pliku .plist zawierającego dane o lokalizacji, ale zamiast korzystać z istniejącego pliku .plist, utworzysz go od podstaw i dodasz do niego dane o lokalizacji. Wykonaj następujące kroki:

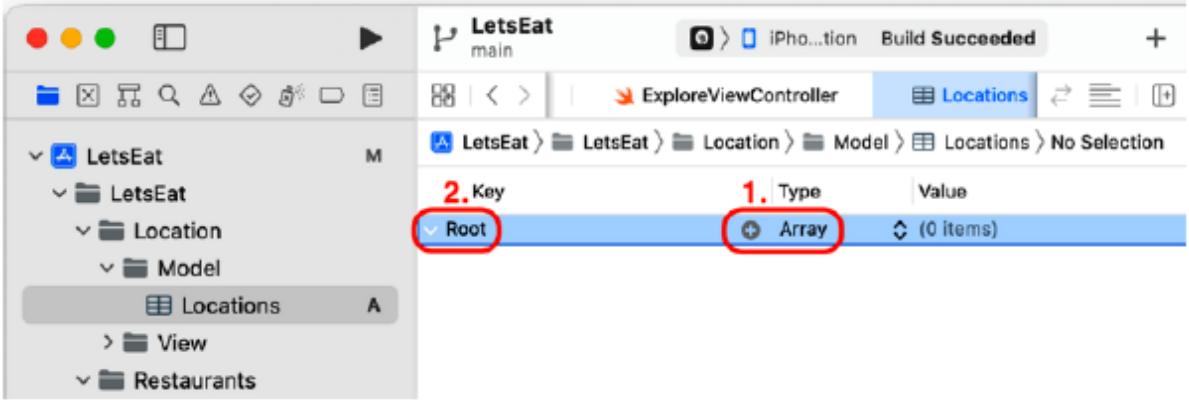
1. Kliknij prawym przyciskiem myszy folder Model w folderze Lokalizacja i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wpisz właściwy w polu filtra; W oknie pojawi się Lista właściwości. Wybierz Lista właściwości i kliknij Dalej.

3. Nazwij plik Lokalizacje i kliknij Utwórz.

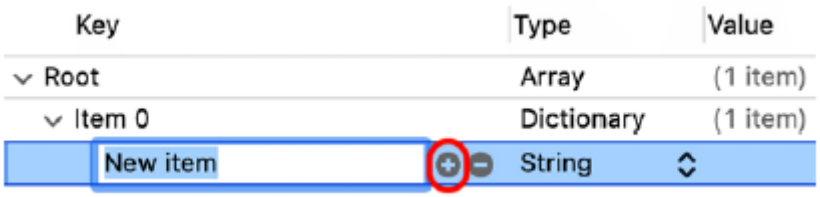
Do projektu dodano plik Locations.plist. W poprzednim rozdziale widziałeś, jak ExploreData.plist przechowuje dane w postaci tablicy słowników. Skonfigurujesz plik Locations.plist tak, aby przechowywał dane z ekranu Lokalizacje w tym samym formacie, a następnie dodasz do niego wszystkie lokalizacje restauracji. Wykonaj następujące kroki:

1. Kliknij Słownik w Locations.plist i zmień go na Array. Zwróć uwagę, że trójkąt odsłonięcia po lewej stronie powinien być skierowany w dół. Kliknij przycisk +:

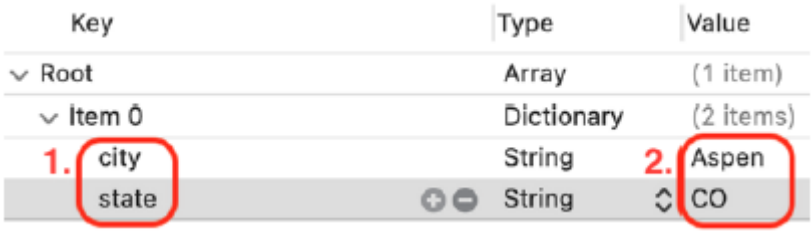


2. Do tablicy zostanie dodany nowy element, Pozycja 0. Zmień typ na Słownik. Kliknij trójkąt rozwijania, aby był skierowany w dół. Kliknij przycisk +:

3. Do słownika Pozycja 0 zostanie dodana nowa pozycja, Nowa pozycja. Kliknij przycisk +:



4. Do słownika Pozycja 0 zostanie dodana druga pozycja. Dla pierwszego elementu zmień klucz na miasto i wartość na Aspen. W przypadku drugiego elementu zmień klucz na stan i wartość na CO:



5. Kliknij trójkąt rozwijania obok słownika Pozycja 0, aby go zwinąć:

6. Wybierz element 0 i naciśnij Command + C na klawiaturze, aby go skopiować, i Command + V, aby wkleić. Zobaczysz nowy element, Pozycja 1

Key	Type	Value
Root	Array	(2 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(2 items)

7. Kliknij trójkąt rozwijania obok słownika Element 1, aby go rozwinąć. Zmień miasto na Boston i stan na MA:

Key	Type	Value
Root	Array	(2 items)
Item 0	Dictionary	(2 items)
Item 1	Dictionary	(2 items)
city	String	Boston
state	String	MA

8. Kontynuuj ten sam proces, dodając następujące miasta i stany:

Item	City	State
Item 2	Charleston	NC
Item 3	Chicago	IL
Item 4	Houston	TX
Item 5	Las Vegas	NV
Item 6	Los Angeles	CA
Item 7	Miami	FL
Item 8	New Orleans	LA
Item 9	New York	NY
Item 10	Philadelphia	PA
Item 11	Portland	OR
Item 12	San Antonio	TX
Item 13	San Francisco	CA

Wypełniony plik .plist powinien wyglądać następująco:

Key	Type	Value
Root	Array	(14 items)
Item 0	Dictionary	(2 items)
city	String	Aspen
state	String	CO
Item 1	Dictionary	(2 items)
city	String	Boston
state	String	MA
Item 2	Dictionary	(2 items)
city	String	Charleston
state	String	NC

Plik Locations.plist jest gotowy. W następnej sekcji utworzysz klasę menedżera danych, podobną do tej, którą utworzyłeś w poprzednim rozdziale, która odczyta plik Locations.plist i udostępni go instancji LocationViewController na ekranie Lokalizacje.

Tworzenie klasy LocationDataManager

Podobnie jak w poprzednim rozdziale, utworzysz klasę menedżera danych, aby załadować dane o lokalizacji z pliku Locations.plist i dostarczyć je do instancji LocationsViewController dla ekranu lokalizacji. Dane zostaną następnie wykorzystane do wypełnienia widoku tabeli na ekranie Lokalizacje. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze Lokalizacja i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik LocationDataManager i kliknij Utwórz.

4. Kliknij plik LocationDataManager w nawigatorze projektu i po instrukcji importu wpisz poniższe polecenie, aby zadeklarować klasę LocationDataManager:

```
class LocationDataManager {

}
```

5. Wewnątrz nawiasów klamrowych dodaj właściwość tablicową „locations”, aby przechowywać listę lokalizacji:

```
private var locations: [String] = []
```

Słowo kluczowe private oznacza, że dostęp do właściwości Locations można uzyskać wyłącznie metodami z tej klasy.

6. Dodaj następujące metody po deklaracji właściwości:

```
private func loadData() -> [[String: String]] {

let decoder = PropertyListDecoder()

if let path = Bundle.main.path(forResource:

"Locations", ofType: "plist"),

let locationsData = FileManager.default.contents(

atPath: path),

let locations = try? decoder.decode([[String:

String]].self, from: locationsData) {

return locations

}

return [[:]]

}

func fetch() {
```

```
for location in loadData() {
if let city = location["city"], let
state = location["state"] {
locations.append("\(city), \(state)")
}
}
}
func numberOfLocationItems() -> Int {
locations.count
}
func locationItem(at index: Int) -> String {
locations[index]
}
```

Metody te są podobne do metod ExploreDataManager. Rozbijmy to:

loadData()

Ładuje zawartość Locations.plist i zwraca tablicę słowników. Każdy słownik przechowuje miasto i stan danej lokalizacji.

fetch()

Pobiera tablicę dostarczoną przez funkcję LoadData(), łączy miasto i stan dla każdego elementu i dołącza wynikowy ciąg do tablicy lokalizacji.

numberOfLocationItems()

Zwraca liczbę elementów w tablicy lokalizacji.

locationItem(at:)

Zwraca ciąg znaków przechowywany w tablicy lokalizacji pod danym indeksem tablicy. Teraz, gdy klasa LocationDataManager jest już gotowa, skonfigurujmy klasę LocationViewController tak, aby mogła pobierać dane z instancji LocationDataManager i dostarczać je do widoku tabeli. Zrobisz to w następnej sekcji.

Wyświetlanie danych w widoku tabeli

Obecnie na ekranie Lokalizacje wyświetlanych jest 10 komórek zawierających ciąg „Komórka”. Zaktualizujesz klasę LocationViewController, aby używać instancji LocationDataManager jako źródła danych. Podążać za tymi krokami:

1. Kliknij plik LocationViewController w nawigatorze projektu. W definicji klasy LocationViewController przed metodą viewDidLoad() utwórz instancję LocationDataManager i przypisz ją do właściwości manager, wpisując:

let manager = LocationDataManager()

2. W metodzie viewDidLoad() pobierz dane do widoku tabeli, wywołując metodę manager.fetch():

```
override func viewDidLoad() {

super.viewDidLoad()

manager.fetch()

}
```

3. Zmodyfikuj tableView(_:numberOfRowsInSection:) tak, aby mógł uzyskać od menedżera liczbę wierszy do wyświetlenia w widoku tabeli:

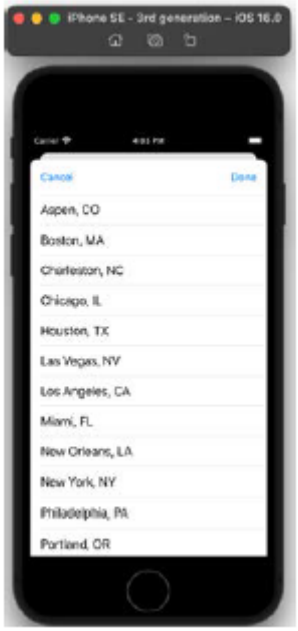
```
func tableView(_ tableView: UITableView,

sekcja numberOfRowsInSection: Int) -> Int {

menedżer.numberOfLocationItems()

}
```

4. Zmodyfikuj tableView(_:cellForRowAt:) jak pokazano, aby widok tabeli wyświetlał ciąg znaków zawierający miasto i stan w każdej komórce widoku tabeli:

```
func tableView(_ tableView: UITableView, cellForRowAt

IndexPath: IndexPath) -> UITableViewCell {

niech komórka = tableView.dequeueReusableCell(

withIdentifier: „locationCell”, dla: IndexPath)

cell.textLabel?.text =

menadżer.lokalizacjaItem(at:indexPath.row)

komórka zwrotna

}
```

Ustawia to właściwość tekstową komórki widoku tabeli na odpowiedni element w tablicy lokalizacji. IndexPath zwraca numer sekcji i wiersza określonego wiersza w widoku tabeli. Na przykład pierwszy wiersz ma wartość IndexPath zawierającą sekcję 0 i wiersz 0. IndexPath.row zwraca 0 dla pierwszego wiersza, więc menedżer zwraca ciąg znaków przechowywany pod indeksem 0 w tablicy Locations. Ten ciąg znaków jest następnie przypisywany do właściwości tekstowej etykiety tekstowej pierwszej komórki widoku tabeli.

Kompiluj i uruchamiaj swoją aplikację. W widoku tabeli powinny zostać wyświetlone lokalizacje z ExploreData.plist:



Streszczenie

Poznałeś widoki tabeli i kontrolery widoku tabeli, a także zaimplementowałeś kontroler widoku dla widoku tabeli na placu zabaw. Następnie zaimplementowano klasę LocationsViewController, kontroler widoku tabeli dla ekranu Lokalizacje, i utworzono od podstaw plik .plist o nazwie Locations.plist, w którym przechowywana jest lista lokalizacji. Utworzyłeś klasę menedżera danych, LocationsDataManager, do odczytu danych z pliku .plist. Na koniec skonfigurowano klasę LocationsViewController tak, aby pobierała dane z instancji LocationsDataManager i dostarczała je do widoku tabeli, dzięki czemu na ekranie Lokalizacje wyświetlana jest lista lokalizacji restauracji. Umożliwi to tworzenie od podstaw plików .plist do przechowywania danych oraz implementowanie widoków tabel korzystających z plików .plist jako źródła danych dla własnych aplikacji. Wspaniały! W następnej części dodasz widok mapy do ekranu Mapa i skonfigurujesz go tak, aby wyświetlał lokalizacje restauracji. Skonfigurujesz także niestandardowe adnotacje na ekranie Mapa i skonfigurujesz ekran Szczegóły restauracji, który będzie wyświetlany po dotknięciu przycisku w objaśnieniu adnotacji

Pierwsze kroki z MapKitem

Zapoznałeś się z widokami tabeli i kontrolerami widoku tabeli oraz ukończyłeś implementację ekranu Lokalizacje. Wyświetla teraz listę lokalizacji restauracji.Tu będziesz wyświetlać lokalizacje restauracji na ekranie mapy za pomocą niestandardowych pinezek. Po dotknięciu pinezki zobaczysz ekran pokazujący szczegóły konkretnej restauracji. Firma Apple udostępnia protokół MKAnnotation, który umożliwia powiązanie utworzonych klas z określoną lokalizacją na mapie. Utworzysz nową klasę RestaurantItem, która jest zgodna z tym protokołem. Następnie utworzysz MapDataManager, klasę menedżera danych, która ładuje dane restauracji z pliku .plist i umieszcza je w tablicy instancji RestaurantItem. Utworzysz nowy protokół DataManager do odczytu plików .plist i zaktualizujesz klasy MapDataManager i ExploreDataManager, aby uniknąć zbędnego kodu (refaktoryzacja). Następnie utworzysz klasę MapViewController, kontroler widoku dla ekranu mapy i skonfigurujesz ją tak, aby wyświetlała niestandardowe piny. Skonfigurujesz pinezki, aby wyświetlały objaśnienia, i skonfigurujesz przyciski w objaśnieniach, aby po dotknięciu wyświetlały ekran szczegółów restauracji. Następnie utworzysz klasę RestaurantDetailViewController, kontroler widoku dla ekranu Szczegóły restauracji i przekażesz do niej dane restauracji z instancji MapViewController. Na koniec uporządkujesz i uporządkujesz kod za pomocą rozszerzeń, aby ułatwić jego czytanie i konserwację. Pod koniec tego rozdziału dowiesz się, jak tworzyć niestandardowe widoki adnotacji map i dodawać je do mapy, jak używać odniesień do scenorysów do łączenia ze sobą scenorysów oraz jak używać rozszerzeń do organizowania kodu, co ułatwia czytanie

Zrozumienie i tworzenie adnotacji

Wcześniej dodałeś widok mapy do ekranu Mapa. Widok mapy jest instancją klasy MKMapView. Możesz zobaczyć, jak to wygląda, w aplikacji Apple Maps. Kiedy zbudujesz i uruchomisz aplikację, na ekranie zobaczysz mapę. Część mapy widoczną na ekranie można określić, ustawiając właściwość regionu mapy. Pinezki na ekranie Mapa służą do oznaczania określonych lokalizacji i są instancjami klasy MKAnnotationView. Aby dodać pinezkę do widoku mapy, potrzebny jest obiekt zgodny z protokołem MKAnnotation. Protokół ten umożliwia powiązanie obiektu z konkretną lokalizacją na mapie. Każdy obiekt może być zgodny z protokołem MKAnnotation poprzez implementację właściwości współrzędnych, która zawiera lokalizację na mapie. Opcjonalne właściwości protokołu MKAnnotation to title, ciąg znaków zawierający tytuł adnotacji; i subtitle, ciąg znaków zawierający podtytuł adnotacji. Gdy obiekt zgodny z protokołem MKAnnotation znajduje się w obszarze mapy widocznym na ekranie, widok mapy prosi swojego delegata (zazwyczaj kontroler widoku) o podanie odpowiedniej instancji klasy MKAnnotationView. Ta instancja pojawia się jako pinezka na mapie. Jeśli użytkownik przewinie mapę, a instancja KAnnotationView zniknie z ekranu, zostanie ona umieszczona w kolejce ponownego użycia i ponownie wykorzystana później, podobnie jak komórki widoku tabeli i komórki widoku kolekcji. Instancję MKAnnotationView można dostosować tak, aby wyświetlała niestandardowe ikony i dymki objaśnień po dotknięciu. Dymki objaśnień mogą zawierać przyciski umożliwiające wykonywanie czynności, takich jak wyświetlanie ekranu. Dla swojej aplikacji utworzysz nową klasę RestaurantItem zgodną z rotokołem MKAnnotation. Zobaczmy, jak utworzyć tę klasę w następnej sekcji.

Tworzenie klasy RestaurantItem

Aby reprezentować lokalizacje restauracji na ekranie Mapa, utworzysz klasę RestaurantItem zgodną z protokołem MKAnnotation. Ta klasa będzie miała właściwość Coord do przechowywania lokalizacji restauracji, właściwość Title do przechowywania nazwy restauracji i właściwość subtitle do przechowywania oferowanych kuchni. Aby ustawić właściwość współrzędnych instancji RestaurantItem, potrzebujesz lokalizacji restauracji. Dane restauracji (w tym jej lokalizacja) zostaną

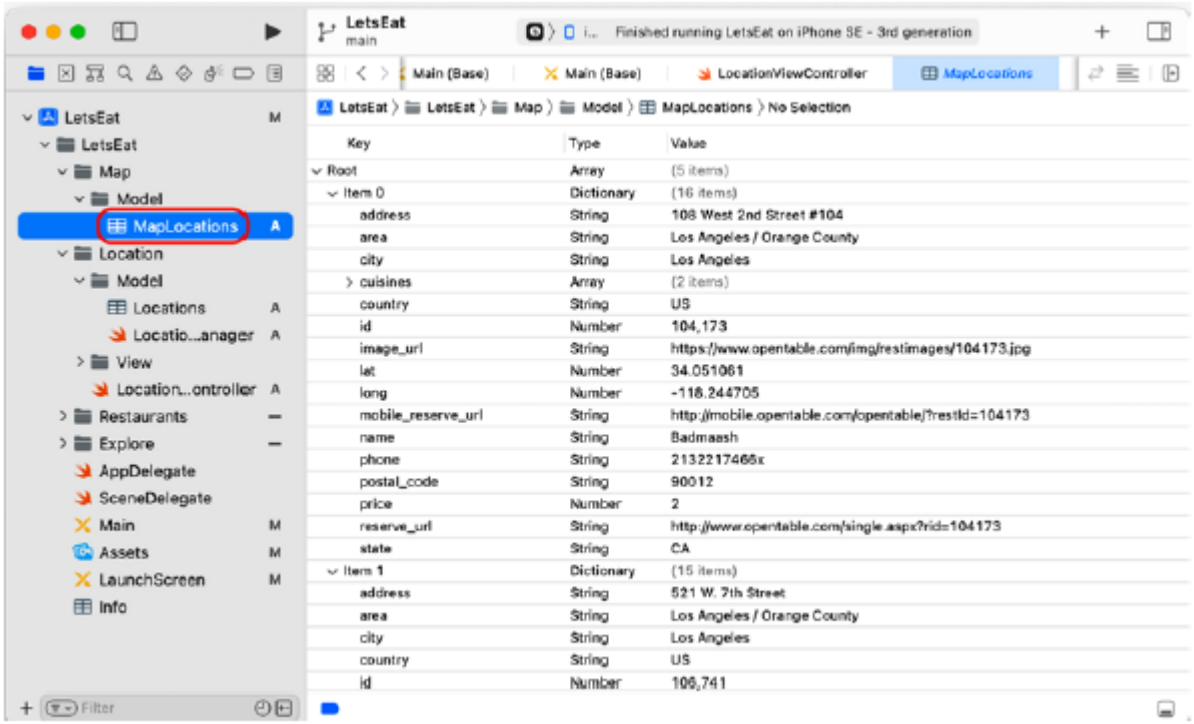
dostarczone w postaci pliku .plist. Zanim utworzysz klasę RestaurantItem, musisz zaimportować ten plik .plist do swojej aplikacji. Wykonaj następujące kroki:

1. Otwórz projekt LetsEat. W nawigatorze projektu kliknij prawym przyciskiem myszy folder LetsEat i utwórz nową grupę o nazwie Mapa.

2. Kliknij prawym przyciskiem myszy folder Mapa i utwórz nową grupę o nazwie Model.

3. Jeśli jeszcze tego nie zrobiłeś, pobierz ukończony projekt i zasoby projektu z https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Seventh- Edition i znajdź plik Maplocations.plist wewnątrz folder Resources w Chapter 17

4. Przeciągnij plik Maplocations.plist do folderu Model w swoim projekcie i kliknij go, aby wyświetlić jego zawartość. Zobaczysz, że jest to szereg słowników, a każdy słownik zawiera szczegółowe informacje o restauracji (w tym jej lokalizację). W klasie estaurantItem utworzysz właściwości dla danych, których będziesz używać, a które ostatecznie zostaną wyświetlone na ekranie Szczegóły restauracji:



.Utwórzmy klasę RestaurantItem, wykonując następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: RestaurantItem

Podklasa: NSObject

Utwórz także XIB: wyszarzony

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. Plik RestaurantItem pojawi się w nawigatorze projektu.

5. W pliku RestaurantItem wpisz następujące polecenie po instrukcji import UIKit, która ma zostać zaimportowana

framework MapKit:

import MapKit

Daje to dostęp do protokołów, takich jak MKAnnotation i MKMapViewDelegate.

6. Zmodyfikuj deklarację klasy w następujący sposób, aby przyjąć protokół MKAnnotation:

klasa RestaurantItem: NSObject, MKAnnotation {

Zobaczysz błąd, ponieważ nie zaimplementowałeś jeszcze właściwości współrzędnych, która jest wymagana, aby zachować zgodność z MKAnnotacją. Zrobisz to wkrótce.

7. Wpisz następujące polecenie w nawiasach klamrowych:

let name: String?

let cuisines: [String]

let lat: Double?

let long: Double?

let address: String?

let postalCode: String?

let state: String?

let imageURL: String?

let restaurantID: Int?

Te właściwości będą przechowywać dane uzyskane z pliku Maplocations.plist. Zobaczmy do czego służą:

name przechowuje nazwę restauracji.

cusine przechowuje dania oferowane przez restaurację.

lat i long przechowuje szerokość i długość geograficzną lokalizacji restauracji.

address przechowuje adres restauracji.

postalCode przechowuje kod pocztowy restauracji.

state przechowuje stan, w którym znajduje się restauracja.

imageURL przechowuje zdjęcie restauracji.

RestaurantID przechowuje unikalny numer używany jako identyfikator restauracji.

Pamiętaj, że nie utworzyłeś właściwości do przechowywania wszystkich szczegółów restauracji zawartych w pliku Maplocations.plist i to jest w porządku. Wystarczy utworzyć właściwości dla szczegółów, które pojawią się na ekranie Szczegóły restauracji.

8. Użyjesz niestandardowego inicjatora, aby zainicjować instancje RestaurantItem danymi z pliku .plist. Wpisz następujące polecenie po ostatniej deklaracji właściwości:

```
init(dict: [String: AnyObject]) {

self.lat = dict["lat"] as? Double

self.long = dict["long"] as? Double

self.name = dict["name"] as? String

self.cuisines = dict["cuisines"] as? [String] ?? []

self.address = dict["address"] as? String

self.postalCode = dict["postalCode"] as? String

self.state = dict["state"] as? String

self.imageURL = dict["image_url"] as? String

self.restaurantID = dict["id"] as? Int

}
```

Mimo że ten inicjator wygląda na skomplikowany, w rzeczywistości jest całkiem prosty. Każda linia szuka określonego klucza elementu słownika i przypisuje jego wartość do odpowiedniej właściwości. Na przykład pierwsza linia szuka elementu słownika z kluczem zawierającym lat i przypisuje powiązaną wartość do właściwości lat.

9. Użyjesz właściwości lat i long, aby utworzyć wartość właściwości współrzędnych, która jest wymagana do zgodności z MKAnnotacją. Wpisz następujące polecenie po metodzie init(dict:) aby ją zaimplementować:

```
var coordinate: CLLocationCoordinate2D {

guard let lat = lat, let long = long else {

return CLLocationCoordinate2D()

}

return CLLocationCoordinate2D(latitude: lat,

longitude: long)

}
```

Właściwość współrzędnych jest typu CLLocationCooperative2D i przechowuje lokalizację geograficzną. Wartość właściwości współrzędnych nie jest przypisana bezpośrednio; instrukcja Guard pobiera wartości szerokości i długości geograficznej z właściwości lat i long, które są następnie wykorzystywane do tworzenia wartości właściwości współrzędnych. Takie właściwości nazywane są właściwościami obliczonymi. 10. Zaimplementuj właściwość tytułu, dodając następujący kod po właściwości współrzędnych:

```
var title: String? {
name
}
```

title to obliczona właściwość, która zwraca zawartość właściwości name.

11. Na koniec zaimplementuj właściwość subtitle, dodając następujący kod po właściwości title:

```
var subtitle: String? {
if cuisines.isEmpty {
return ""
} else if cuisines.count == 1 {
return cuisines.first
} else {
return cuisines.joined(separator: ", ")
}
}
```

subtitle jest również właściwością obliczaną. Pierwsza linia sprawdza, czy właściwość cuisines jest pusta i jeśli tak, zwraca pusty ciąg znaków. Jeśli właściwość cuisines zawiera pojedynczy element, element ten zostanie zwrócony. Jeśli właściwość cuisines zawiera więcej niż jeden element, każdy element jest dodawany do ciągu znaków, oddzielany przecinkiem. Na przykład, jeśli kuchnie zawierały tablicę ["American", "Bistro", "Burgers"], wygenerowany ciąg będzie miał postać "American, Bistro, Burgers". Twoja klasa RestaurantItem jest teraz kompletna i wolna od błędów i powinna wyglądać tak:

```
import UIKit
import MapKit
class RestaurantItem: NSObject, MKAnnotation {
let name: String?
let cuisines: [String]
let lat: Double?
let long: Double?
let address: String?
let postalCode: String?
let state: String?
let imageURL: String?
let restaurantID: Int?
init(dict: [String: AnyObject]) {
```

```
self.lat = dict["lat"] as? Double
self.long = dict["long"] as? Double
self.name = dict["name"] as? String
self.cuisines = dict["cuisines"] as? [String]
?? []
self.address = dict["address"] as? String
self.postalCode = dict["postalCode"] as? String
self.state = dict["state"] as? String
self.imageURL = dict["image_url"] as? String
self.restaurantID = dict["id"] as? Int
}
var coordinate: CLLocationCoordinate2D {
guard let lat = lat, let long = long else {
return CLLocationCoordinate2D()
}
return CLLocationCoordinate2D(latitude: lat,
longitude: long)
}
var title: String? {
name
}
var subtitle: String? {
if cuisines.isEmpty {
return ""
} else if cuisines.count == 1 {
return cuisines.first
} else {
return cuisines.joined(separator: ", ")
}
}
}
```

W tym momencie dodałeś plik Maplocations.plist do swojej aplikacji i utworzyłeś klasę RestaurantItem. Następnie utwórzmy klasę menedżera danych, która odczytuje dane restauracji z pliku Maplocations.plist i umieszcza je w tablicy instancji RestaurantItem do wykorzystania przez Twoją aplikację.

Tworzenie klasy MapDataManager

Podobnie jak w poprzednich rozdziałach, utworzysz klasę menedżera danych, MapDataManager, która załaduje dane restauracji z pliku Maplocations.plist i umieści je w tablicy instancji RestaurantItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze Map i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik MapDataManager. Kliknij Utwórz. Plik MapDataManager pojawi się w nawigatorze projektu.

4. W pliku MapDataManager dodaj po instrukcji import następujące polecenie, aby zadeklarować klasę MapDataManager:

```
class MapDataManager {

}
```

5. Dodaj następujące właściwości między nawiasami klamrowymi, aby przechowywać instancje RestaurantItem który zostanie odczytany z pliku .plist:

```
private var items: [RestaurantItem] = []

var annotations: [RestaurantItem] {

items

}
```

Tablica items będzie zawierać instancje RestaurantItem. private sprawia, że tablica elementów jest dostępna tylko w klasie MapDataManager, a adnotacje to właściwość obliczana, która zwraca kopię tablicy elementów po uzyskaniu dostępu. Dzięki temu inne obiekty mogą uzyskać dostęp do zawartości tablicy items, ale nie mogą jej modyfikować.

6. Dodaj następujące metody po deklaracjach właściwości, aby załadować plik .plist, odczytać zawarte w nim dane i zapisać je w tablicy instancji RestaurantItem:

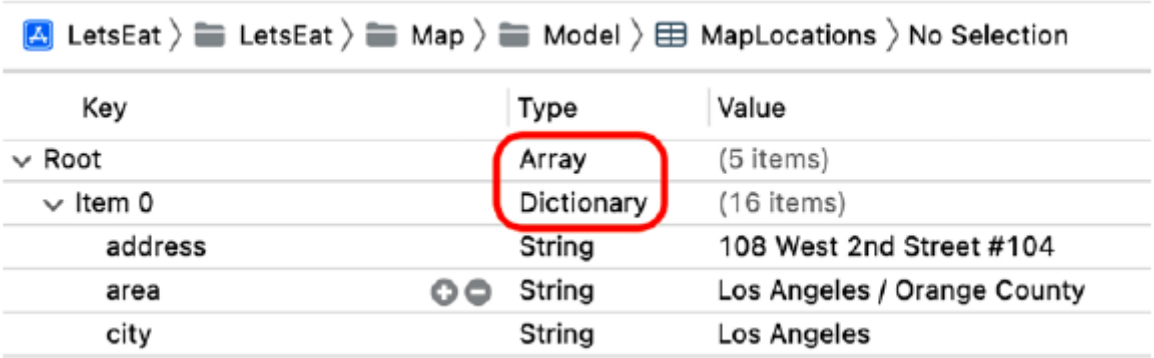
```
private func loadData() -> [[String: AnyObject]] {

guard let path = Bundle.main.path(forResource:

"MapLocations", ofType: "plist"),

let itemsData = FileManager.default.contents(

atPath: path),

let items = try! PropertyListSerialization

.propertyList(from: itemsData, format: nil) as?
```

```
[[String: AnyObject]] else {
return [[:]]
}
return items
}
func fetch(completion: (_ annotations:
[RestaurantItem]) -> ()) {
if !items.isEmpty {
items.removeAll()
}
for data in loadData() {
items.append(RestaurantItem(dict: data))
}
completion(items)
}
```

Metody loadingData() i fetch(completion:) wykonują te same zadania, co metody loadingData() i fetch() w klasie ExploreDataManager. Jednakże zastosowana tutaj metoda loadingData() może zwrócić tablicę zawierającą słowniki, których wartości są typu AnyObject. Jest to konieczne, ponieważ plik MapLocations.plist, w odróżnieniu od pliku ExploreData.plist, nie zawiera wyłącznie słowników typu [String: String]. Ponadto zastosowana tutaj metoda etch(completion:) ma jako parametr zamknięcie zakończenia, które może zaakceptować dowolną funkcję lub zamknięcie, które przyjmuje jako parametr tablicę RestaurantItems:

```
(_ annotations:[RestaurantItem]) -> ())
```

Czasami nie wiadomo, kiedy operacja się zakończy. Na przykład musisz wykonać jakąś czynność po pobraniu pliku z Internetu, ale nie wiesz, ile czasu zajmie pobranie. Można określić zamknięcie zakończenia, które ma zostać zastosowane po zakończeniu operacji. W tym przypadku zamknięcie zakończenia przetworzy tablicę items po wczytaniu wszystkich danych z pliku .plist. Teraz rozważ jeszcze raz plik MapLocations.plist:



Ten plik ma taką samą strukturę jak ExploreData.plist. Element główny jest tablicą zawierającą słowniki. Ponieważ zarówno ExploreData.plist, jak i MapLocations.plist mają tablicę słowników, bardziej efektywne byłoby utworzenie jednej metody ładowania plików .plist i używania ich tam, gdzie jest to potrzebne. Zrobisz to w następnej sekcji.

Tworzenie protokołu DataManager

Zamiast tworzyć w każdej klasie metodę ładującą plik .plist, utworzysz nowy protokół DataManager do obsługi ładowania pliku .plist. Ten protokół zaimplementuje metodę ładowania plików .plist przy użyciu rozszerzenia. Po utworzeniu protokołu DataManager może go przyjąć dowolna klasa, która musi załadować plik .plist. Zmodyfikujesz zarówno klasy ExploreDataManager, jak i MapDataManager, aby przyjąć ten protokół. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder LetsEat i utwórz nową grupę o nazwie Różne.

2. Kliknij prawym przyciskiem myszy folder Misc i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

4. Nazwij ten plik DataManager. Kliknij Utwórz.

5. Kliknij plik DataManager w nawigatorze projektu i zadeklaruj protokół DataManager w następujący sposób:

```
import Foundation

protocol DataManager {

func loadPlist(file name: String) ->

[[String: AnyObject]]

}
```

Protokół ten wymaga, aby każdy zgodny obiekt miał metodę o nazwie loadingPlist(file:) która przyjmuje ciąg znaków jako parametr i zwraca tablicę słowników. Ciąg będzie zawierał nazwę pliku .plist, który ma zostać załadowany.

6. Po deklaracji protokołu dodaj rozszerzenie zawierające implementację metody loadingPlist(file:)

```
import Foundation

protocol DataManager {

func loadPlist(file name: String) ->

[[String: AnyObject]]

}

extension DataManager {

func loadPlist(file name: String) ->

[[String: AnyObject]] {

guard let path = Bundle.main.path(forResource:

name, ofType: "plist"),
```

```
let itemsData = FileManager.default

.contents(atPath: path),

let items = try! PropertyListSerialization

.propertyList(from: itemsData, format: nil)

as? [[String: AnyObject]] else {

return [[:]]

}

return items

}

}
```

Każda klasa, która przyjmie ten protokół, otrzyma metodę LoadPlist(file:) . Ta metoda szuka pliku .plist określonego w parametrze name w pakiecie aplikacji. Jeśli plik nie zostanie znaleziony, zwracana jest pusta tablica słowników. W przeciwnym razie zawartość pliku .plist jest ładowana do tablicy słowników typu [String: AnyObject] i zwracana. Teraz, gdy masz już ten protokół, zmodyfikujesz klasy MapDataManager i ExploreDataManager, aby go zastosować. Kiedy bierzesz istniejący kod i modyfikujesz go, aby osiągnąć to samo wydajniej, proces ten nazywa się refaktoryzacją. W następnej sekcji zaczniesz od refaktoryzacji klasy MapDataManager w celu dostosowania jej do protokołu DataManager.

Refaktoryzacja klasy MapDataManager

Klasa MapDataManager ma już metodę loadingData(), która jest zakodowana na stałe w celu odczytania pliku Maplocations.plist. Teraz, gdy utworzyłeś protokół DataManager, zmodyfikujesz klasę MapDataManager, aby zamiast tego go używać. Wykonaj następujące kroki:

1. Po wybraniu pliku MapDataManager w nawigatorze projektu znajdź i usuń metodę loadingData(). Zobaczysz błąd, ponieważ metoda fetch() wywołuje metodę loadingData(), którą właśnie usunąłeś. Wkrótce to naprawisz.

2. Dodaj protokół DataManager do deklaracji klasy w następujący sposób:

class MapDataManager: DataManager Dzięki temu metoda loadingPlist(file:) będzie dostępna dla klasy MapDataManager.

3. Zmodyfikuj wiersz for data w funkcji loadingData() w metodzie fetch() w następujący sposób, aby naprawić błąd:

```
for data in loadPlist(file: "MapLocations")
```

Twoja zaktualizowana klasa MapDataManager powinna wyglądać następująco:

```
import Foundation

class MapDataManager: DataManager {

private var items: [RestaurantItem] = []

var annotations: [RestaurantItem] {
```

```
items
}
func fetch(completion: (_ annotations:
[RestaurantItem]) -> ()){
if !items.isEmpty {
items.removeAll()
}
for data in loadPlist(file: "MapLocations") {
items.append(RestaurantItem(dict: data))
}
completion(items)
}
}
```

Błąd powinien zniknąć. W następnej sekcji dokonasz także refaktoryzacji klasy ExploreDataManager, aby była zgodna z protokołem DataManager.

Refaktoryzacja klasy ExploreDataManager

Podobnie jak klasa MapDataManager, klasa ExploreDataManager posiada metodę loadingData(), która jest zakodowana na stałe w celu odczytu pliku ExploreData.plist.

Musisz wprowadzić te same zmiany w klasie ExploreDataManager, które wprowadziłeś w klasie MapDataManager. Wykonaj następujące kroki:

1. Po wybraniu pliku ExploreDataManager w nawigatorze projektu znajdź i usuń metodę loadingData(). Zignoruj błąd, ponieważ wkrótce zostanie on naprawiony.

2. Dodaj protokół DataManager do deklaracji klasy w następujący sposób:

```
class ExploreDataManager: DataManager
```

3. Zmodyfikuj metodę fetch() w następujący sposób, aby naprawić błąd:

```
func fetch() {
for data in loadPlist(file: "ExploreData") {
exploreItems.append(ExploreItem(dict: data
as! [String: String]))
}
}
```

Należy pamiętać, że dane są rzutowane jako [String: String], dzięki czemu można ich używać do inicjowania instancji klasy ExploreItem. Możesz teraz sprawić, że każda klasa, która musi załadować

plik .plist zawierający tablicę słowników, przyjmie protokół DataManager, tak jak zrobiłeś to w przypadku klas MapDataManager i ExploreDataManager. Nie zawsze jest jasne, kiedy należy przeprowadzić refaktoryzację, ale im więcej masz doświadczenia, tym staje się to łatwiejsze. Jedną z oznak konieczności refaktoryzacji jest sytuacja, gdy piszesz ten sam kod w więcej niż jednej klasie. Zakończyłeś implementację klasy MapDataManager, utworzyłeś protokół DataManager i dokonałeś refaktoryzacji klas MapDataManager i ExploreDataManager w celu zapewnienia zgodności z tym protokołem. Za pomocą klasy MapDataManager można załadować dane z pliku MapLocations.plist i zwrócić tablicę instancji RestaurantItem. Zobaczmy teraz, jak użyć tej tablicy, aby dodać adnotacje do widoku mapy, które będą wyświetlane jako pinezki na ekranie mapy.

Dodawanie adnotacji do widoku mapy

Wcześniej dodałeś widok mapy do ekranu Mapa. W poprzednich sekcjach dodałeś plik MapLocations.plist do swojego projektu i utworzyłeś klasy RestaurantItem i MapDataManager. Pamiętasz wzorzec projektowy MVC? W tym momencie utworzyłeś widoki i modele dla ekranu Mapa, więc wszystko, czego teraz potrzebujesz, to kontroler widoku. Kontroler widoku będzie odpowiedzialny za następujące zadania:

• Dodawanie do widoku mapy instancji RestaurantItem zgodnych z protokołem MKAnnotation.

• W przypadku instancji RestaurantItem w regionie wyświetlanym w widoku mapy podaj instancje MKAnnotationView wymagane w widoku mapy.

• Zapewnij niestandardowe instancje MKAnnotationView, które po dotknięciu wyświetlają dymek zawierający przycisk i wyświetlają ekran szczegółów restauracji po dotknięciu przycisku. W następnej sekcji zaczniesz od utworzenia klasy MapViewController jako kontrolera widoku dla ekranu Mapa.

Tworzenie klasy MapViewController

Utworzyłeś obiekty widoku i modelu dla ekranu Mapa, a jedyne, co pozostaje, to utworzenie dla niego kontrolera widoku. Utworzysz nową klasę, MapViewController, która będzie kontrolerem widoku na ekranie mapy. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Mapa i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: MapViewController

Podklasa: UIViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. Plik MapViewController pojawi się w nawigatorze projektu.

5. W pliku MapViewController po zaimportowaniu UIKit dodaj następujący wiersz, aby zaimportować framework MapKit:

import MapKit

6. Zmodyfikuj deklarację klasy w następujący sposób, aby klasa MapViewController przyjęła protokół MKMapViewDelegate:

class MapViewController: UIViewController,

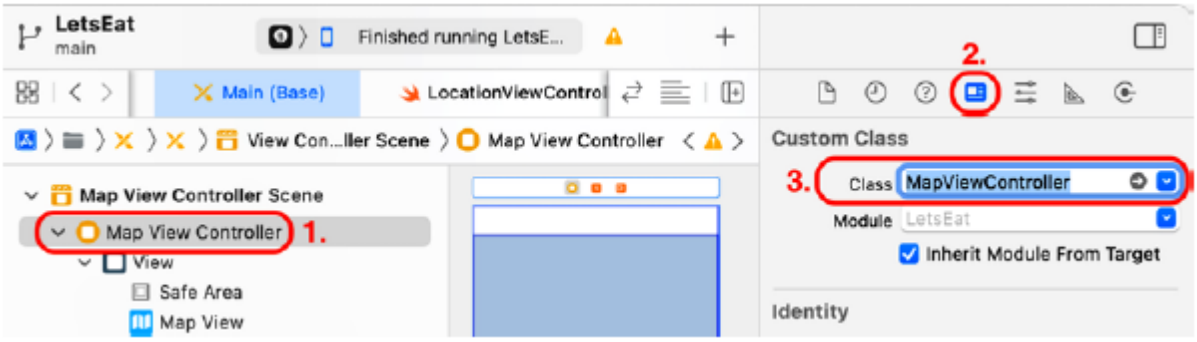
MKMapViewDelegate {

Zadeklarowałeś klasę MapViewController. W następnej sekcji przypiszesz tę klasę jako kontroler widoku dla ekranu mapy i utworzysz punkt wyjścia dla widoku mapy.

Podłączenie gniazd dla widoku mapy

Scena kontrolera widoku na ekranie Mapa wyświetla mapę, ale obecnie nie ma możliwości ustawienia wyświetlanego regionu mapy ani wyświetlania adnotacji. Przypiszmy klasę MapViewController jako kontroler widoku dla ekranu Mapa i dodajmy do niej wyjście dla widoku mapy. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu. Kliknij ikonę kontrolera widoku w scenie kontrolera widoku na ekranie mapy. W Inspektorze tożsamości w obszarze Klasa niestandardowa ustaw klasę na MapViewController:

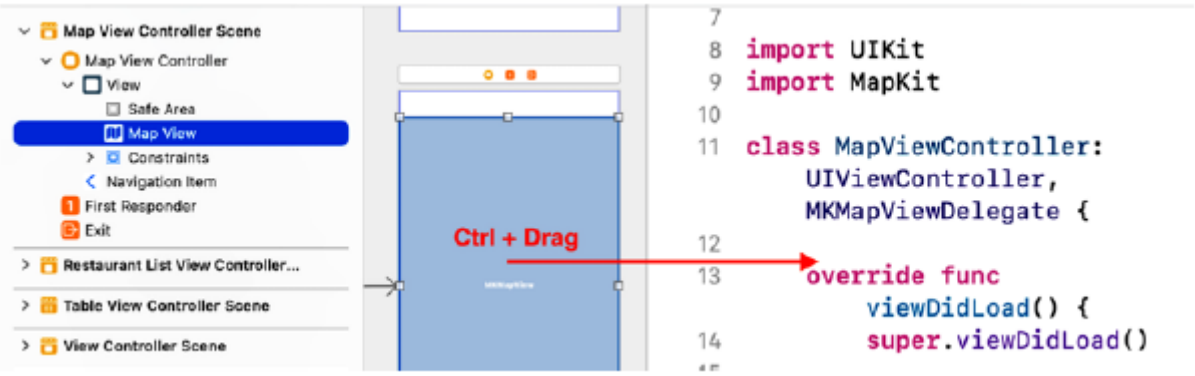


2. Wybierz Widok mapy w konspekcie dokumentu:

3. Kliknij przycisk Dostosuj opcje edytora.

4. Z wyskakującego menu wybierz opcję Asystent.

5. Pojawi się asystent edytora pokazujący zawartość pliku MapViewController. Ctrl + Przeciągnij z widoku mapy do miejsca tuż pod deklaracją klasy:



6. Wpisz mapView w polu Nazwa i kliknij Połącz:

7. Widok mapy został podłączony do gniazda mapView w klasie MapViewController. Kliknij przycisk x, aby zamknąć edytor asystenta:

Klasa MapViewController ma teraz punkt wyjścia, mapView, który jest połączony z widokiem mapy na ekranie Mapa. W następnej sekcji zmodyfikujesz klasę MapDataManager, dodając metodę generowania nowego regionu na podstawie lokalizacji restauracji, aby mogła ona zapewnić region mapy do wyświetlenia w widoku mapy

Ustawianie wyświetlanego regionu widoku mapy

W widoku mapy część mapy widoczna na ekranie nazywana jest regionem. Aby określić region, potrzebne są współrzędne środka regionu oraz rozpiętość pozioma i pionowa reprezentująca wymiary mapy, która ma zostać wyświetlona. Metoda fetch(completion:) w klasie MapDataManager zwraca tablicę instancji RestaurantItem. Zaimplementujesz metodę inicjującąRegion(latDelta:longDelta:), aby pobrać pierwszą instancję RestaurantItem z tej tablicy, uzyskać współrzędne restauracji i użyć ich do utworzenia regionu. Wykonaj następujące kroki:

1. Kliknij plik MapDataManager w nawigatorze projektu. Po instrukcji import Foundation dodaj import MapKit.

2. Tuż przed zamykającym nawiasem klamrowym zaimplementuj metodę initialRegion(latDelta:longDelta:) w następujący sposób:

```
func initialRegion(latDelta: CLLocationDegrees,

longDelta: CLLocationDegrees) -> MKCoordinateRegion {

guard let item = items.first else {

return MKCoordinateRegion()

}

let span = MKCoordinateSpan(latitudeDelta:

latDelta, longitudeDelta: longDelta)

return MKCoordinateRegion(center: item.coordinate,

span: span)

}
```

Rozbijmy to:

```
func initialRegion(latDelta: CLLocationDegrees, longDelta:

CLLocationDegrees) -> MKCoordinateRegion
```

Ta metoda przyjmuje dwa parametry i zwraca instancję MKCooperativeRegion. latDelta określa odległość z północy na południe (mierzoną w stopniach) wyświetlaną dla obszaru mapy. Jeden stopień to około 69 mil. longDelta określa odległość ze wschodu na zachód (mierzoną w stopniach) wyświetlaną dla obszaru mapy. Zwracana instancja MKCooperativeRegion określa region, który pojawi się na ekranie.

```
guard let item = items.first else { return MKCoordinateRegion() }
```

Instrukcja Guard pobiera pierwszy element z tablicy instancji RestaurantItem i przypisuje go do elementu. Jeśli tablica jest pusta, zwracana jest pusta instancja MKCooperativeRegion.

```
let span = MKCoordinateSpan(latitudeDelta: latDelta, longitudeDelta:

longDelta)
```

latDelta i longDelta służą do tworzenia instancji MKCooperativeSpan, która określa poziomy i pionowy zakres tworzonego regionu.

```
return MKCoordinateRegion(center: item.coordinate, span: span)
```

Instancja MKCooperativeRegion jest tworzona i zwracana przy użyciu właściwości współrzędnych elementu i instancji MKCooperativeSpan. Teraz, gdy region mapy został określony, możesz określić, które instancje RestaurantItem znajdują się w tym regionie, na podstawie ich właściwości współrzędnych. Pamiętaj, że klasa RestaurantItem jest zgodna z MKAnnotation. Jako kontroler widoku mapy, klasa MapViewController jest odpowiedzialna za dostarczanie instancji MKAnnotationView dla dowolnych instancji RestaurantItem w tym regionie.

W następnej sekcji zmodyfikujesz klasę MapViewController, aby zapewnić MKAnnotationViews dla instancji RestaurantItem w regionie wyświetlanym w widoku mapy.

Wyświetlanie instancji MKAnnotationView na widoku mapy

W tym momencie masz klasę MapViewController do zarządzania widokiem mapy na ekranie Mapa i możesz wywołać metodę inicjującąRegion(latDelta:longDelta:) w klasie MapDataManager, aby ustawić region mapy. Teraz zmodyfikujesz klasę MapViewController, aby uzyskać tablicę instancji RestaurantItem z klasy MapDataManager i dodać ją do widoku mapy. Wykonaj następujące kroki:

1. Kliknij plik MapViewController w nawigatorze projektu i usuń skomentowany kod.

2. Zaraz po deklaracji właściwości mapView dodaj następującą komendę, aby utworzyć instancję klasy MapDataManager i przypisać ją menadżerowi:

```
private let manager = MapDataManager()
```

3. Dodaj następującą metodę po funkcji viewDidLoad(). Ta metoda doda instancje RestaurantItem (zgodne z protokołem MKAnnotation) do widoku mapy:

```
func setupMap(_ annotations: [RestaurantItem]) {

mapView.setRegion(manager.initialRegion(

latDelta: 0.5, longDelta: 0.5), animated: true)

mapView.addAnnotations(manager.annotations)

}
```

Metoda setupMap(_:) przyjmuje parametr adnotacje, który jest tablicą instancji RestaurantItem. Ustawia region mapy, który ma być wyświetlany w widoku mapy przy użyciu metody inicjującejRegion(latDelta:longDelta:) klasy MapDataManager, a następnie dodaje każdą instancję RestaurantItem w tablicy adnotacji do widoku mapy. Następnie delegat widoku mapy (w tym przypadku klasa apViewController) udostępnia dane automatycznie instancję MKAnnotationView dla każdej instancji RestaurantItem w regionie.

4. Dodaj następującą metodę przed metodą setupMap(_:). Wywołuje to metodę fetch(completion:) instancji MapDataManager i przekazuje metodę setupMap(_:) jako zamknięcie zakończenia:

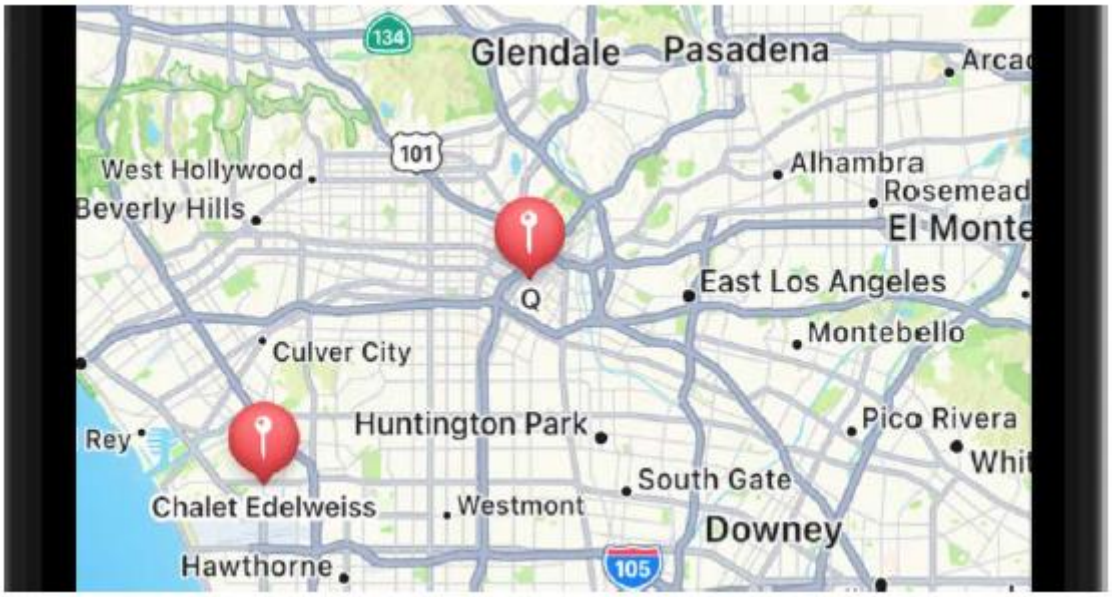
```
func initialize() {

manager.fetch {(annotations) in

setupMap(annotations)

}

}
```

Metoda fetch(completion:) ładuje plik MapLocations.plist oraz tworzy i przypisuje tablicę instancji RestaurantItem do tablicy items. Właściwość adnotacje zwraca kopię tablicy items. Tablica ta jest następnie przetwarzana przez metodę setupMap(_:) przekazaną jako zamknięcie zakończenia.

5. Wywołaj metodę inicjalizacji() wewnątrz funkcji viewDidLoad(), aby została wywołana po załadowaniu widoku mapy:

```
override func viewDidLoad() {

super.viewDidLoad()

initialize()

}
```

Zbuduj i uruchom aplikację. Powinieneś zobaczyć instancje pinów MKAnnotationView) na ekranie mapy:



Do każdej instancji RestaurantItem w obszarze mapy dodano instancję MKAnnotationView. Każda instancja MKAnnotationView jest reprezentowana przez pin. Masz teraz pinezki pokazujące lokalizacje restauracji na mapie, ale musisz dodać kod, aby wyświetlać niestandardowe pinezki, jak pokazano w przewodniku po aplikacji. Zrobisz to w następnej sekcji.

Tworzenie niestandardowych instancji MKAnnotationView

Obecnie na ekranie Mapa wyświetlane są standardowe instancje MKAnnotationView, które wyglądają jak pinezki. Możesz zastąpić standardowy obraz pinów obrazem niestandardowym. W zasobach znajduje się niestandardowy obraz xcassets i skonfigurujesz klasę MapViewController, aby z niego korzystała. Spowoduje to, że pinezki na ekranie będą zgodne z tymi w przewodniku po aplikacji. Skonfigurujesz także każdy pin tak, aby po dotknięciu wyświetlał dymek z objaśnieniem. Wykonaj następujące kroki:

1. Kliknij plik MapViewController w nawigatorze projektu.

2. Dodaj następujący kod do metody inicjalizacji() po otwierającym nawiasie klamrowym. To sprawia, że klasa MapViewController jest delegatem widoku mapy:

```
func initialize() {

mapView.delegate = self
```

3. Dodaj następującą metodę po metodzie setupMap(_:). Ta metoda zwraca niestandardową instancję MKAnnotationView dla każdej instancji MKAnnotation w regionie wyświetlanym w widoku mapy:

```
MKAnnotation) -> MKAnnotationView? {
let identifier = "custompin"
guard !annotation.isKind(of: MKUserLocation.self)
else {
return nil
}
let annotationView: MKAnnotationView
if let customAnnotationView =
mapView.dequeueReusableAnnotationView(
withIdentifier: identifier) {
annotationView = customAnnotationView
annotationView.annotation = annotation
} else {
let av = MKAnnotationView(annotation:
annotation, reuseIdentifier: identifier)
av.rightCalloutAccessoryView =
UIButton(type: .detailDisclosure)
annotationView = av
}
annotationView.canShowCallout = true
if let image = UIImage(named:
"custom-annotation") {
annotationView.image = image
annotationView.centerOffset = CGPoint(
x: -image.size.width / 2,
y: -image.size.height / 2)
}
return annotationView
}
```

Rozbijmy to:

```
func mapView(_ mapView: MKMapView, viewFor
```

annotation: MKAnnotation) -> MKAnnotationView?

Jest to jedna z metod delegowania określonych w protokole MKMapViewDelegate. Jest wyzwalana, gdy instancja MKAnnotation znajduje się w obszarze mapy i zwraca KAnnotationView instancję, którą użytkownik zobaczy na ekranie. Użyjesz tej metody, aby zastąpić domyślne szpilki niestandardowymi szpilkami.

let identifier = "custompin"

Stałej identyfikatorowi przypisany jest ciąg „custompin”. Będzie to identyfikator ponownego użycia.

guard !annotation.isKind(of: MKUserLocation.self)

else {

return nil

}

Oprócz określonych adnotacji instancja MKMapView doda także adnotację dotyczącą lokalizacji użytkownika. To oświadczenie zabezpieczające sprawdza, czy adnotacja przedstawia lokalizację użytkownika. Jeśli tak, zwracane jest zero, ponieważ lokalizacja użytkownika nie jest lokalizacją restauracji.

let annotationView: MKAnnotationView

annotationView jest stałą typu MKAnnotationView. Tworzysz to, aby móc skonfigurować i zwrócić je później.

if let customAnnotationView =

mapView.dequeueReusableAnnotationView (withIdentifier:

identifier) {

annotationView = customAnnotationView

annotationView.annotation = annotation

}

Instrukcja if sprawdza, czy istnieją adnotacje, które były początkowo widoczne, ale nie są już widoczne na ekranie. Jeśli tak, instancja MKAnnotationView dla tej adnotacji może zostać ponownie wykorzystana i przypisana do zmiennej annotationView. Parametr annotation jest przypisany do właściwości adnotation obiektu annotationView.

else {

let av = MKAnnotationView(annotation: annotation,

reuseIdentifier: identifier)

av.rightCalloutAccessoryView =

UIButton(type: .detailDisclosure)

annotationView = av

}

Klauzula else jest wykonywana, jeśli nie istnieją żadne instancje MKAnnotationView, które można ponownie wykorzystać. Tworzona jest nowa instancja MKAnnotationView z określonym wcześniej identyfikatorem ponownego użycia (custompin). Instancja MKAnnotationView jest konfigurowana za pomocą objaśnienia. Po dotknięciu pinezki na mapie pojawi się dymek z tytułem (nazwa restauracji), podtytułem (kuchnie) i przyciskiem. Później zaprogramujesz przycisk tak, aby wyświetlał ekran szczegółów restauracji.

```
annotationView.canShowCallout = true

if let image = UIImage(named: "custom-annotation") {

annotationView.image = image

annotationView.centerOffset = CGPoint(

x: -image.size.width / 2,

y: -image.size.height / 2)

}
```

Spowoduje to skonfigurowanie utworzonej właśnie instancji MKAnnotationView do wyświetlania dodatkowych informacji w dymku objaśnienia i ustawienie obrazu niestandardowego na obraz niestandardowej adnotacji przechowywany w pliku Assets.xcassets. Podczas dodawania niestandardowego obrazu adnotacja wykorzystuje środek obrazu jako punkt docelowy, więc właściwość centerOffset służy do ustawiania prawidłowego położenia punktu, na końcu pinezki.

```
return annotationView
```

Zwracana jest niestandardowa instancja MKAnnotationView.

Skompiluj i uruchamiaj swoją aplikację. Możesz zobaczyć niestandardowe pinezki na swojej mapie:

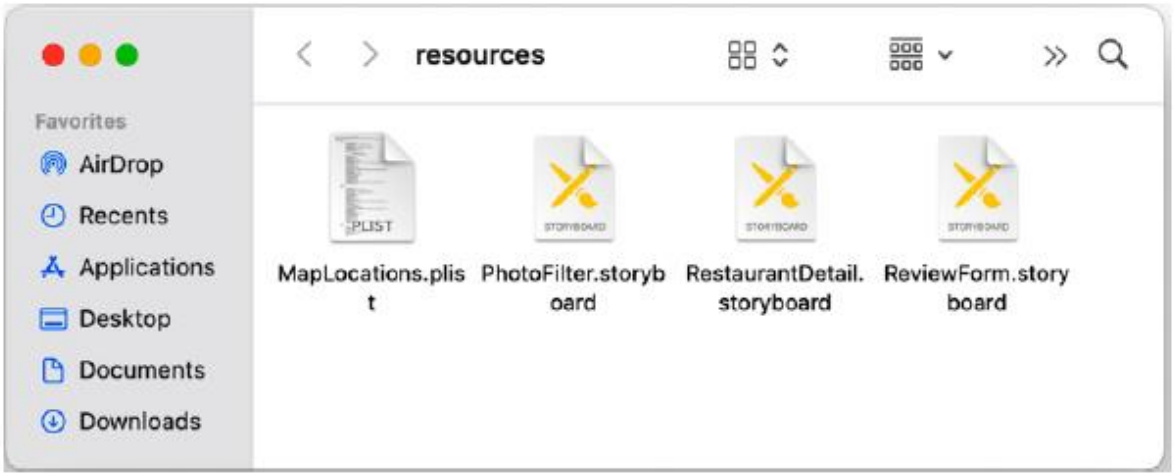
Skonfigurowałeś ekran Mapa tak, aby wyświetlał region zawierający niestandardowe instancje MKAnnotationView przy użyciu danych uzyskanych z klasy MapDataManager. Kliknięcie pinezki powoduje wyświetlenie dymka z nazwą restauracji i oferowaną przez nią kuchnią. Dotknięcie przycisku

w dymku objaśnienia jeszcze nic nie daje. W następnej sekcji skonfigurujesz przycisk tak, aby wyświetlał ekran szczegółów restauracji

Przejście z ekranu Mapa do ekranu Szczegóły restauracji

Na ekranie Mapa wyświetlane są teraz niestandardowe instancje MKAnnotationView, a dotknięcie jednego z nich powoduje wyświetlenie dymka objaśniającego przedstawiającego szczegóły restauracji. Jednak przycisk w dymku objaśnienia jeszcze nie działa. W pobranym wcześniej folderze zasobów znajdziesz gotowe scenorysy o nazwach RestaurantDetail.storyboard, PhotoFilter.storyboard i ReviewForm.storyboard, które dodasz do swojego projektu. Te scenorysy zawierają sceny z ekranu Szczegóły restauracji, ekranu Filtru zdjęć i ekranu Formularza recenzji. Aby wyświetlić ekran szczegółów restauracji za pomocą przycisku objaśnienia, dodasz odniesienie do scenorysu do swojego projektu i połączysz z nim plik scenorysu RestaurantDetail. Zrobisz to w następnej sekcji.

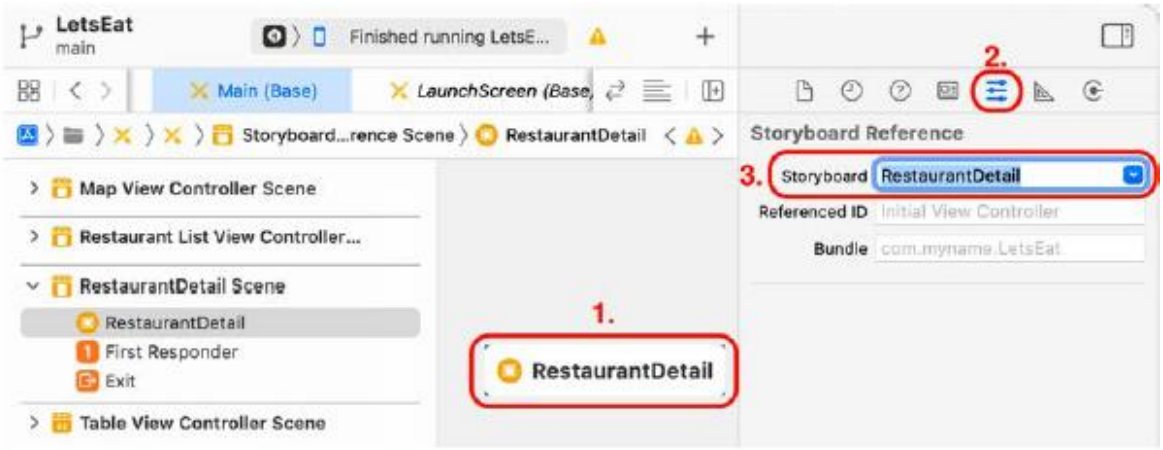
4. Otwórz pobrany wcześniej folder zasobów i zlokalizuj w nim trzy pliki scenorysów, które dodasz do swojego projektu (RestaurantDetail.storyboard, PhotoFilter. scenorys i ReviewForm.storyboard):



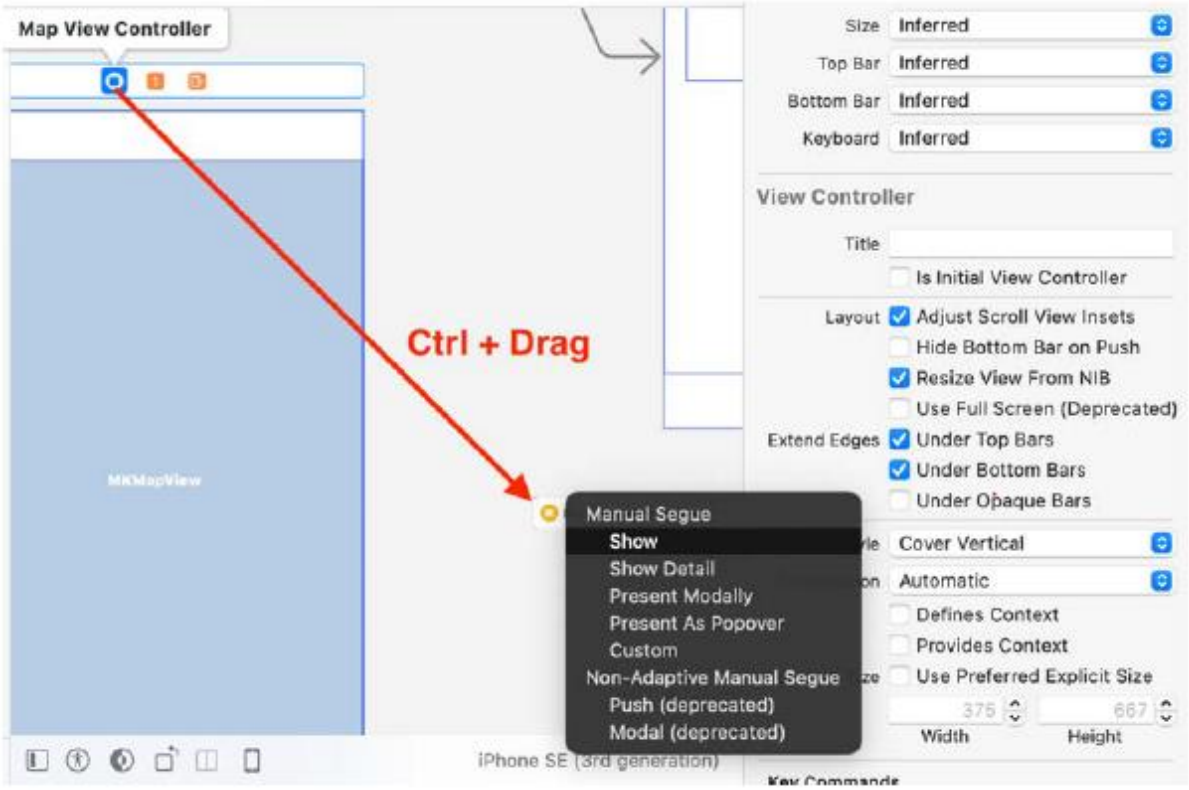
5. W nawigatorze projektu utwórz nowy folder w folderze LetsEat o nazwie RestaurantDetail i skopiuj do niego plik scenorysu RestaurantDetail:

6. Utwórz nowy folder w swoim folderze LetsEat o nazwie ReviewForm i skopiuj do niego plik scenorysu ReviewForm, a następnie utwórz nowy folder w swoim folderze LetsEat o nazwie PhotoFilter i skopiuj do niego plik scenorysu PhotoFilter:

7. Teraz przypiszesz plik scenorysu RestaurantDetail do odniesienia do scenorysu dodanego wcześniej do projektu. Kliknij plik głównego scenorysu, wybierz dodane wcześniej odniesienie do scenorysu i kliknij przycisk Inspektora atrybutów. W sekcji Storyboard Reference ustaw Storyboard na RestaurantDetail:

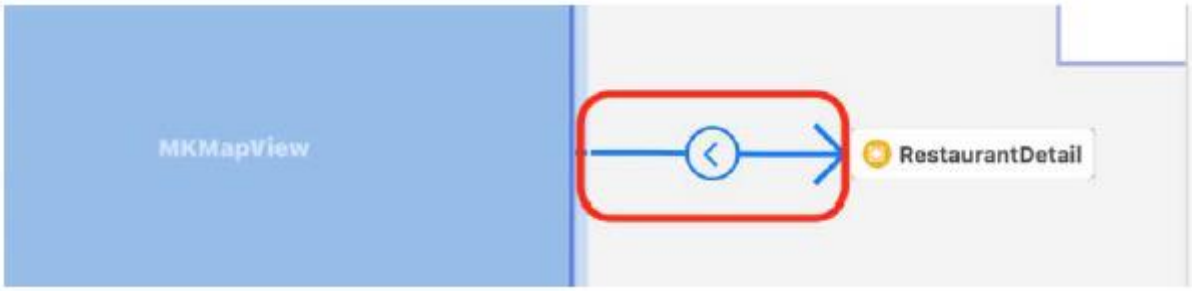


8. Ctrl + Przeciągnij ikonę kontrolera widoku mapy do odniesienia do scenorysu i wybierz Pokaż z wyskakującego menu, aby dodać przejście pomiędzy sceną kontrolera widoku mapy a odniesieniem do scenorysu:

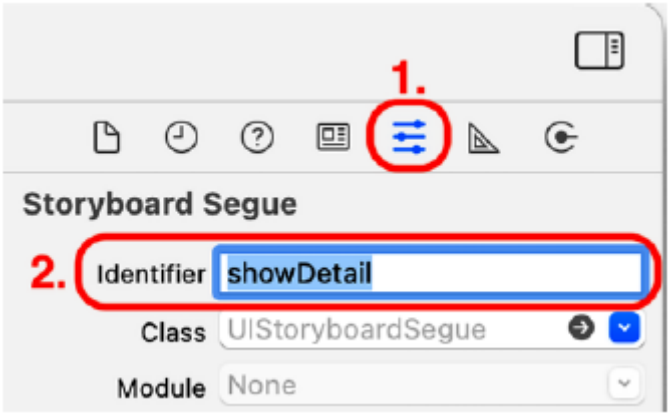


Jest to wymagane, aby wyświetlić ekran szczegółów restauracji po dotknięciu przycisku objaśnienia instancji MKAnnotationView na ekranie mapy.

9. Ustawisz identyfikator dla tego przejścia. Później dodasz metodę, która po dotknięciu przycisku dymku z objaśnieniem wykona przejście z tym identyfikatorem. Wybierz połączenie łączące scenę kontrolera widoku mapy z odniesieniem do scenorysu:



10. W Inspektorze atrybutów, w sekcji Storyboard Segue ustaw Identyfikator na showDetail:



Połączyłeś teraz scenę kontrolera widoku dla ekranu Mapa ze sceną kontrolera widoku dla ekranu Szczegóły restauracji za pomocą przejścia. W następnej sekcji zaimplementujesz metodę wyświetlania ekranu szczegółów restauracji po dotknięciu przycisku dymku z objaśnieniem.

Wykonywanie sekwencji showDetail

Połączyłeś scenę kontrolera widoku dla ekranu Mapa ze sceną kontrolera widoku dla ekranu Szczegóły restauracji za pomocą przejścia. Ustawiłeś także identyfikator przejścia na showDetail. Teraz potrzebujesz metody, aby wykonać to przejście, ale zanim ją zaimplementujesz, utworzysz wyliczenie zawierające wszystkie identyfikatory przejścia dla tego projektu. Zmniejsza to potencjalne błędy, włączając autouzupełnianie po wpisaniu kolejnych identyfikatorów w kodzie. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Misc w folderze LetsEat i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik Segue. Kliknij Utwórz. Plik Segue pojawi się w nawigatorze projektu.

4. Dodaj następujące polecenie po instrukcji import, aby zadeklarować i zdefiniować wyliczenie Segue:

```
enum Segue: String {

case showDetail

case showRating

case showReview

case showAllReviews

case restaurantList

case locationList

case showPhotoReview

case showPhotoFilter

}
```

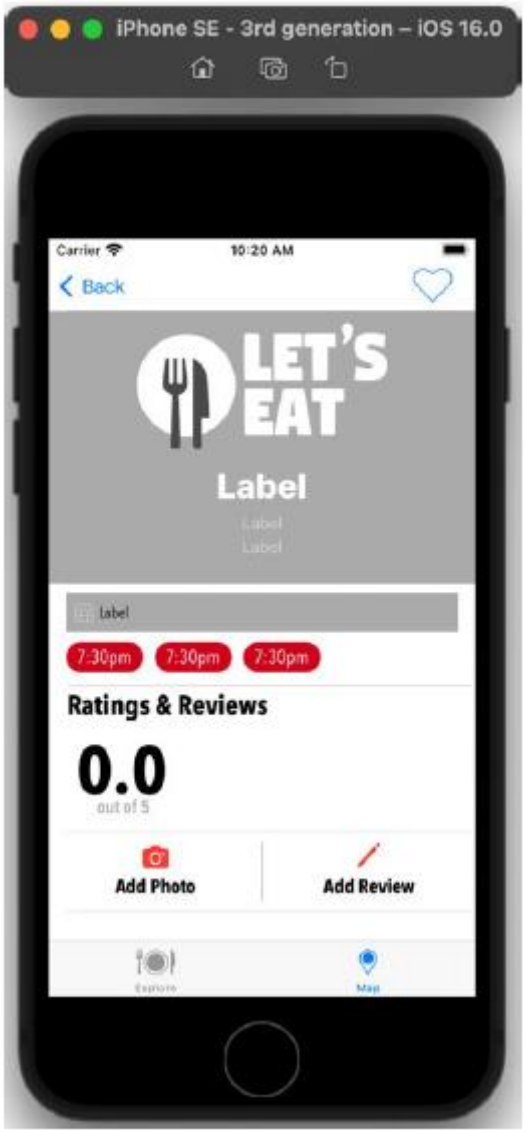
Należy pamiętać, że typem wyliczenia Segue jest String, zatem surowymi wartościami dla każdego przypadku są ciągi znaków. Na przykład surowa wartość przypadku showDetail to „showDetail”. Teraz możesz dodać metodę wykonywania przejścia showDetail po dotknięciu przycisku objaśnienia. Kliknij

plik MapViewController w nawigatorze projektu i dodaj następującą metodę po metodzie setupMap(_:):

```
func mapView(_ mapView: MKMapView, annotationView view:

MKAnnotationView, calloutAccessoryControlTapped control:

UIControl) {

self.performSegue(withIdentifier:

Segue.showDetail.rawValue, sender: self)

}
```

mapView(_:annotationView:calloutAccessoryControlTapped:) to kolejna metoda określona w protokole MKMapViewDelegate. Uruchamia się, gdy użytkownik dotknie przycisku dymku z objaśnieniem. self.performSegue(withIdentifier: Segue.showDetail.rawValue, nadawca: self) wykonuje sekwencję z identyfikatorem „showDetail”, który wyświetla ekran szczegółów restauracji. Zbuduj i uruchom swój projekt. Na ekranie Mapa dotknij pinezki, a następnie dotknij przycisku wewnątrz dymku objaśnienia:

Pojawi się nowy ekran szczegółów restauracji, ale nie będzie zawierał żadnych szczegółów na temat restauracji:



W następnej części sprawisz, że ekran szczegółów restauracji będzie wyświetlał szczegółowe informacje o restauracji, ale na razie po prostu przekażmy dane o wybranej restauracji do kontrolera widoku ekranu szczegółów restauracji i wydrukujmy je w obszarze debugowania. Zrobisz to w następnej sekcji.

Przekazywanie danych do ekranu szczegółów restauracji

Na ekranie Mapa wyświetlane są teraz niestandardowe instancje MKAnnotationView, które po dotknięciu wyświetlają dymki z objaśnieniami. Po dotknięciu przycisku w dymku objaśnienia pojawia się ekran szczegółów restauracji, ale jest on wyświetlany nie zawiera żadnych danych o restauracji. Będziesz musiał przekazać dane restauracji z powiązanej instancji RestaurantItem do kontrolera widoku ekranu Szczegóły restauracji, co nie zostało jeszcze stworzone. Wykonaj następujące kroki, aby utworzyć go teraz:

1. Kliknij prawym przyciskiem myszy folder RestaurantDetail i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: RestaurantDetailViewController

Podklasa: UITableViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. W nawigatorze projektu pojawi się plik RestaurantDetailViewController.

5. Usuń kod z pliku, aż będzie wyglądał tak:

```
import UIKit

class RestaurantDetailViewController:

UITableViewController {

override func viewDidLoad() {

super.viewDidLoad()

}

}
```

6. Zadeklaruj właściwość o nazwie wybranejRestaurant przed metodą viewDidLoad():

```
var selectedRestaurant: RestaurantItem?
```

Ta właściwość przechowuje instancję RestaurantItem, która zostanie przekazana do instancji RestaurantDetailViewController z instancji MapViewController.

7. Dodaj następujący kod do metody viewDidLoad() przed zamykającym nawiasem klamrowym, aby wydrukować zawartość instancji RestaurantItem w obszarze Debug:

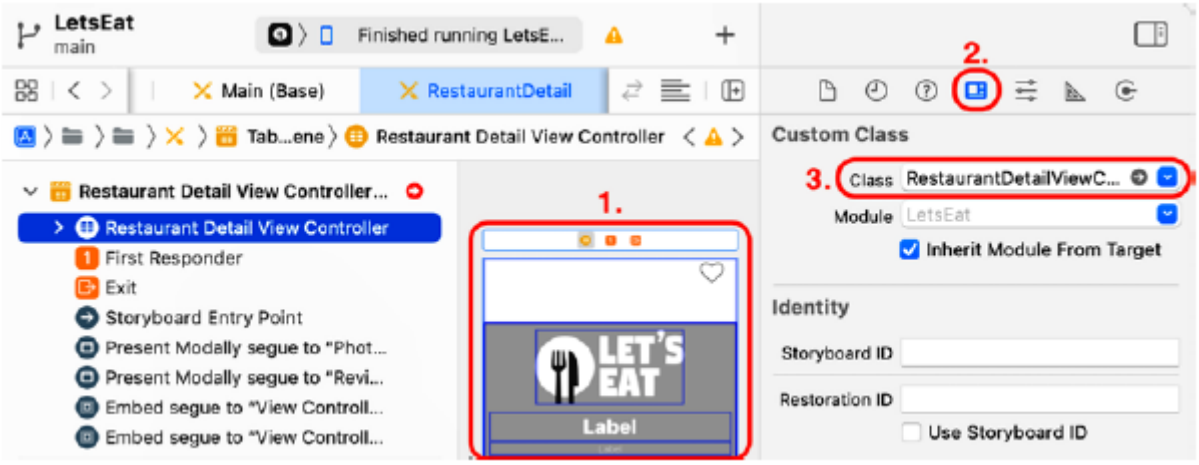
```
dump(selectedRestaurant as Any)
```

Potwierdza to, że instancja MapViewController pomyślnie przekazała instancję RestaurantItem do instancji RestaurantDetailViewController.

8. Sprawdź, czy plik wygląda następująco:

```
import UIKit

class RestaurantDetailViewController: UITableViewController {

var selectedRestaurant: RestaurantItem?

override func viewDidLoad() {

super.viewDidLoad()

dump(selectedRestaurant as Any)

}

}
```

9. Kliknij plik scenorysu RestaurantDetail w folderze RestaurantDetail. Wybierz scenę kontrolera widoku tabeli w scenorysie. Kliknij przycisk Inspektora tożsamości. W obszarze Klasa niestandardowa ustaw klasę na RestaurantDetailViewController:



Należy pamiętać, że nazwa sceny zmieni się na Scena kontrolera widoku szczegółów restauracji.

10. Kliknij plik MapViewController w nawigatorze projektu.

11. Dodaj właściwość do przechowywania instancji RestaurantItem po private let manager =

Instrukcja MapDataManager():

var selectedRestaurant: RestaurantItem?

12. Dodaj następujący kod do metody func mapView (_:annotationView:calloutAccessoryControlTapped:) przed wywołaniem metody self.performSegue(withIdentifier:sender:):

```
func mapView(_ mapView: MKMapView, annotationView

view: MKAnnotationView, calloutAccessoryControlTapped

control: UIControl) {

guard let annotation =

mapView.selectedAnnotations.first else {

return

}

selectedRestaurant = annotation as? RestaurantItem

self.performSegue(withIdentifier:

Segue.showDetail.rawValue, sender: self)

}
```

Spowoduje to pobranie instancji RestaurantItem powiązanej z instancją MKAnnotationView, która została dotknięta i przypisanie jej do wybranej restauracji.

13. Aby przekazać instancję RestaurantItem z instancji MapViewController do instancji RestaurantDetailViewController, należy zastąpić metodę UIViewController o nazwie przygotowanie(for:sender:). Wpisz następujący kod po funkcji viewDidLoad():

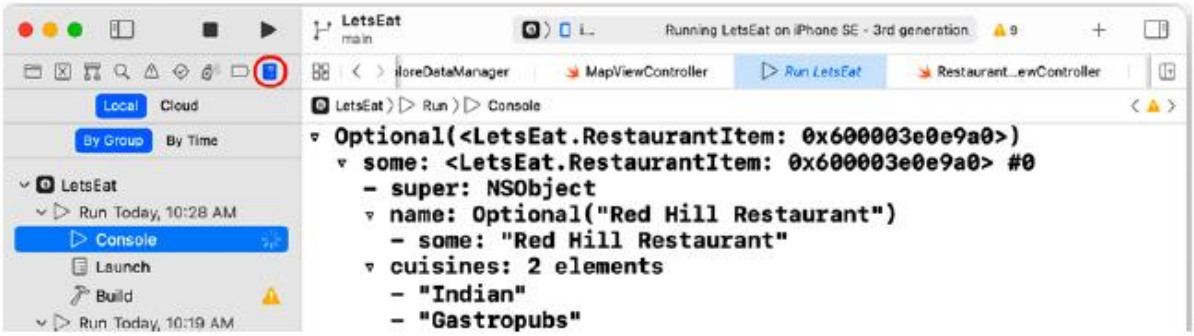
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?){

switch segue.identifier! {

case Segue.showDetail.rawValue:

showRestaurantDetail(segue: segue)

default:

print("Segue not added")

}

}
```

Metoda przygotowania(for:sender:) jest wykonywana przez kontroler widoku przed przejściem do innego kontrolera widoku. W tym przypadku metoda ta jest wywoływana przed przejściem ekranu Mapa do ekranu Szczegóły restauracji. Jeżeli identyfikatorem segue jest showDetail, a tak jest w tym przypadku, wywoływana jest metoda showRestaurantDetail(segue:). Ta metoda ustawi właściwość wybranąRestaurant dla instancji RestaurantDetailViewController. Zobaczysz błąd, ponieważ showRestaurantDetail(segue:) nie zostało jeszcze utworzone.

14. Dodaj następujący kod po metodzie setupMap(_:) aby zaimplementować showRestaurantDetail(segue:):

```
func showRestaurantDetail(segue: UIStoryboardSegue) {

if let viewController = segue.destination as?

RestaurantDetailViewController, let restaurant =

selectedRestaurant {

viewController.selectedRestaurant = restaurant

}

}
```

Sprawdza to, czy docelowym miejscem docelowym jest instancja RestaurantDetailViewController. Jeśli tak, do tymczasowej stałej restauracji przypisana jest właściwość wybranej restauracji instancja MapViewController. restauracja zostaje następnie przypisana do właściwości wybranej restauracji w instancji RestaurantDetailViewController. Innymi słowy, szczegóły restauracji uzyskane z instancji RestaurantItem są przekazywane do instancji RestaurantDetailViewController. Kompiluj i uruchamiaj swoją aplikację. Na ekranie Mapa dotknij pinezki, a następnie dotknij przycisku objaśnienia. Pojawi się ekran szczegółów restauracji. Kliknij nawigator raportów i kliknij pierwszy wpis, jak pokazano. Szczegóły restauracji powinieneś zobaczyć w obszarze Edytora:



Dodałeś scenorys dla ekranu Szczegóły restauracji do swojego projektu, połączyłeś go z ekranem Mapa i pomyślnie przekazałeś dane z wybranej restauracji na ekranie Mapa do ekranu Szczegóły restauracji. Instancja RestaurantDetailViewController zawiera teraz dane z instancji RestaurantItem wybranej na ekranie Mapa. Świetnie! Skonfigurujesz Restaurację. Ekran szczegółów umożliwiający wyświetlenie tych danych w następnej części.

Wykonałeś dużo pracy , więc zanim przejdziesz do następnej części, uporządkujmy napisany przez Ciebie kod, aby był łatwiejszy do zrozumienia. W tym celu użyjesz rozszerzeń w pliku w następneja sekcji.

Porządkowanie kodu

W miarę jak Twoje programy staną się bardziej złożone, do uporządkowania kodu będziesz używać rozszerzeń (omówionych w rozdziale 8, Protokoły, rozszerzenia i obsługa błędów). Rozszerzenia mogą pomóc w zwiększeniu czytelności kodu i uniknięciu bałaganu. Zorganizujesz cztery klasy: ExploreViewController, RestaurantListViewController, LocationViewController i MapViewController. Będziesz segregować bloki powiązanego kodu za pomocą rozszerzeń. Zacznijmy od klasy ExploreViewController w następnej sekcji.

Refaktoryzacja klasy ExploreViewController

Kod w pliku ExploreViewController podzielisz na odrębne sekcje za pomocą rozszerzeń. Wykonaj następujące kroki:

1. Kliknij plik ExploreViewController w nawigatorze projektu. Po ostatnim nawiasie klamrowym dodaj:

```
// MARK: Private Extension

private extension ExploreViewController {

// code goes here

}

// MARK: UICollectionViewDataSource

extension ExploreViewController:

UICollectionViewDataSource {

// code goes here

}
```

Tutaj tworzysz dwa rozszerzenia. Pierwsze rozszerzenie będzie prywatne, co oznacza, że zawartość tego rozszerzenia będzie dostępna tylko dla klasy ExploreViewController. Drugie rozszerzenie będzie zawierać wszystkie metody UICollectionViewDataSource.

2. Otrzymasz błąd, ponieważ UICollectionViewDataSource pojawia się w dwóch miejscach. Usuń UICollectionViewDataSource z deklaracji klasy na górze pliku. Deklaracja klasy powinna wyglądać następująco:

```
class ExploreViewController: UIViewController, UICollectionViewDelegate {
```

3. Przenieś wszystkie metody UICollectionViewDataSource do drugiego rozszerzenia. To powinno wyglądać tak:

```
// MARK: UICollectionViewDataSource

extension ExploreViewController:

UICollectionViewDataSource {

func collectionView(_ collectionView:

UICollectionView, viewForSupplementaryElementOfKind

kind: String, at indexPath: IndexPath) ->

UICollectionReusableView {

let headerView = collectionView.

dequeueReusableSupplementaryView( ofKind: kind,

withReuseIdentifier: "header", for: indexPath)

return headerView

}

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

manager.numberOfExploreItems()

}

func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(

withReuseIdentifier: "exploreCell", for:

indexPath) as! ExploreCell

let exploreItem = manager.exploreItem(at:
```

```
indexPath.row)

cell.exploreNameLabel.text = exploreItem.name

cell.exploreImageView.image = UIImage(

named: exploreItem.image!)

return cell

}

}
```

Aby zachować możliwie największą czystość funkcji viewDidLoad(), utworzysz metodę inicjalizacji() wewnątrz rozszerzenia prywatnego i umieścisz w niej wszystko, czego potrzebujesz do zainicjowania kontrolera widoku. Następnie wywołasz inicjalizację() w viewDidLoad().

4. Dodaj metodę inicjalizacji() wewnątrz rozszerzenia prywatnego:

```
func initialize() {

manager.fetch()

}
```

5. Przenieś metodę unwindLocationCancel(segue:) również do rozszerzenia prywatnego.

6. Sprawdź, czy rozszerzenie prywatne wygląda następująco:

```
// MARK: Private Extension

private extension ExploreViewController {

func initialize() {

manager.fetch()

}

@IBAction func unwindLocationCancel(segue:

UIStoryboardSegue) {

}

}
```

7. Na koniec zmodyfikuj funkcję viewDidLoad() w następujący sposób:

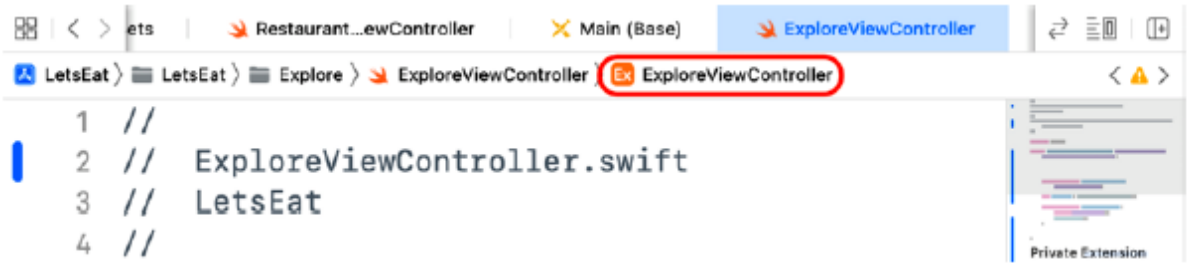
```
override func viewDidLoad() {

super.viewDidLoad()

initialize()

}
```

Korzyści z segregowania kodu w ten sposób mogą nie wydawać się teraz oczywiste, ale gdy klasy staną się bardziej złożone, łatwiej będzie znaleźć konkretną metodę i utrzymać kod. Zanim zrobisz to samo z innymi plikami, zobaczmy, jak używana jest składnia // MARK: w następnej sekcji.

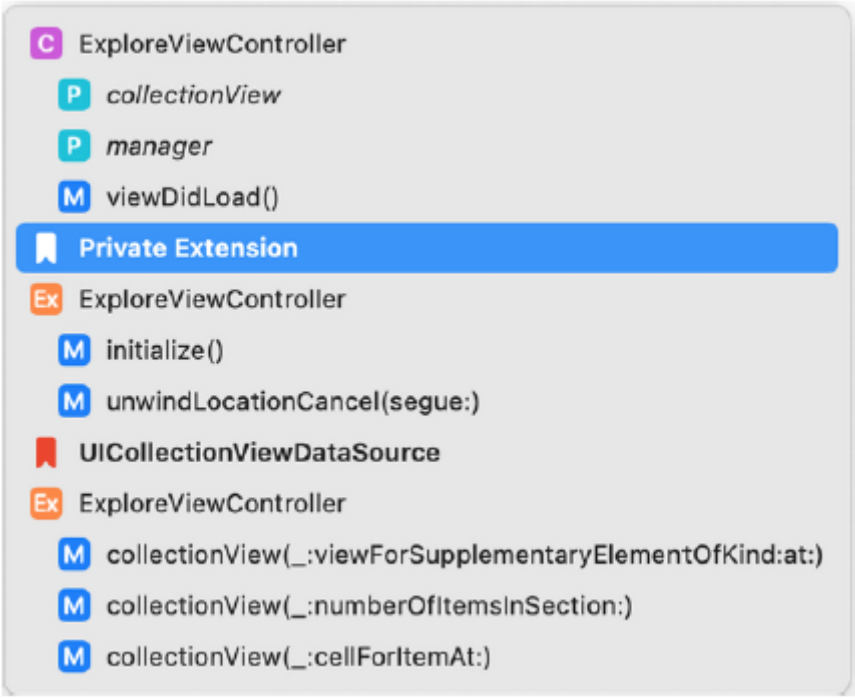
Korzystanie ze składni // MARK:

Składnia //MARK: służy do łatwego poruszania się pomiędzy różnymi częściami kodu. Zobaczmy, co robi:

1. Spójrz na ścieżkę widoczną tuż pod paskiem narzędzi i kliknij ostatnią część, jak pokazano:



2. Wyświetlone zostanie menu, w którym zobaczysz zarówno rozszerzenie prywatne, jak i UICollectionViewDataSource, wygenerowane za pomocą składni //MARK:. Dzięki temu możesz łatwo przejść do następujących sekcji:



Zorganizowałeś klasę ExploreViewController, więc teraz zajmiemy się klasą RestaurantListViewController, refaktoryzując ją i dodając rozszerzenia.

Refaktoryzacja klasy RestaurantListViewController

Do klasy RestaurantListViewController dodasz dwa rozszerzenia, podobne do tych, które dodałeś do klasy ExploreViewController. Wykonaj następujące kroki:

1. Kliknij plik RestaurantListViewController w nawigatorze projektu. Po ostatnim nawiasie klamrowym:

```
// MARK: Private Extension

private extension RestaurantListViewController {

// code goes here

}

// MARK: UICollectionViewDataSource

extension RestaurantListViewController:

UICollectionViewDataSource {

// code goes here

}
```

W pierwszym rozszerzeniu umieścisz prywatne metody klasy RestaurantListViewController, a w drugim wszystkie metody UICollectionViewDataSource.

2. Usuń UICollectionViewDataSource z deklaracji klasy na górze pliku. Deklaracja klasy powinna wyglądać następująco:

```
class RestaurantListViewController: UIViewController,

UICollectionViewDelegate {
```

3. Przenieś wszystkie metody UICollectionViewDataSource do drugiego rozszerzenia. Po zakończeniu powinno to wyglądać tak:

```
// MARK: UICollectionViewDataSource extension

RestaurantListViewController:

UICollectionViewDataSource {

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {

1

}

func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath:

IndexPath) -> UICollectionViewCell {

collectionView.dequeueReusableCell(

withReuseIdentifier: "restaurantCell",

for: indexPath)

}
```

}

Skończyłeś porządkowanie klasy RestaurantListViewController, więc w następnej sekcji uporządkujmy klasę LocationViewController.

Refaktoryzacja klasy LocationViewController

Podobnie jak poprzednio, dodasz dwa rozszerzenia do pliku LocationViewController. Wykonaj następujące kroki:

1. Kliknij plik LocationViewController w nawigatorze projektu. Po ostatnim nawiasie klamrowym dodaj:

```
// MARK: Private Extension

private extension LocationViewController {

// code goes here

}

// MARK: UITableViewDataSource

extension LocationViewController:

UITableViewDataSource {

// code goes here

}
```

Pierwsze rozszerzenie będzie zawierać prywatne metody dla klasy LocationViewController. Drugie rozszerzenie będzie zawierać wszystkie metody UITableViewDataSource.

2. Usuń UITableViewDataSource z deklaracji klasy na górze pliku. Deklaracja klasy powinna wyglądać następująco:

```
class LocationViewController: UIViewController {
```

3. Przenieś wszystkie metody UITableViewDataSource do drugiego rozszerzenia. To powinno wyglądać tak:

```
// MARK: UITableViewDataSource

extension LocationViewController: UITableViewDataSource

{

func tableView(_ tableView: UITableView,

numberOfRowsInSection section: Int) -> Int {

manager.numberOfLocationItems()

}

func tableView(_ tableView: UITableView,

cellForRowAt indexPath: IndexPath) ->

UITableViewCell {
```

```
let cell = tableView.dequeueReusableCell(

withIdentifier: "locationCell", for: indexPath)

cell.textLabel?.text =

manager.locationItem(at: indexPath.row)
```



```
return cell

}

}
```

4. Podobnie jak w klasie ExploreViewController, utworzysz metodę inicjalizacji() w pierwszym rozszerzeniu i umieścisz w niej wszystko, czego potrzebujesz, aby zainicjować w niej klasę LocationViewController. Dodaj następujące elementy w pierwszym rozszerzeniu:

```
// MARK: Private Extension

private extension LocationViewController {

func initialize() {

manager.fetch()

}

}
```

5. Zmodyfikuj funkcję viewDidLoad() w następujący sposób, aby wywołać metodę inicjalizacji():

```
override func viewDidLoad() {

super.viewDidLoad()

initialize()

}
```

Skończyłeś porządkowanie klasy LocationViewController, więc w następnej sekcji uporządkujmy klasę MapViewController.

Refaktoryzacja klasy MapViewController

Podobnie jak poprzednio w przypadku innych klas, dodasz dwa rozszerzenia do klasy MapViewController. Wykonaj następujące kroki:

1. Kliknij plik MapViewController w nawigatorze projektu. Po ostatnim nawiasie klamrowym dodaj:

```
// MARK: Private Extension

private extension MapViewController {

// code goes here

}
```

```
// MARK: MKMapViewDelegate

extension MapViewController: MKMapViewDelegate {

// code goes here

}
```

Pierwsze rozszerzenie będzie zawierało prywatne metody dla klasy MapViewController. Druga będzie zawierać wszystkie metody MKMapViewDelegate.

2. Usuń MKMapViewDelegate z deklaracji klasy na górze pliku. Definicja Twojej klasy powinna wyglądać następująco:

```
class MapViewController: UIViewController {
```

3. Przenieś wszystkie metody MKMapViewDelegate do drugiego rozszerzenia. To powinno wyglądać tak:

```
// MARK: MKMapViewDelegate

extension MapViewController: MKMapViewDelegate {

func mapView(_ mapView: MKMapView, annotationView

view: MKAnnotationView,

calloutAccessoryControlTapped control: UIControl){

guard let annotation =

mapView.selectedAnnotations.first else

{

return

}

selectedRestaurant = annotation as?

RestaurantItem

self.performSegue(withIdentifier:

Segue.showDetail.rawValue, sender: self)

}

func mapView(_ mapView: MKMapView, viewFor

annotation:MKAnnotation) -> MKAnnotationView? {

let identifier = "custompin"

guard !annotation.isKind(of:

MKUserLocation.self) else {

return nil
```

```
}
let annotationView: MKAnnotationView
if let customAnnotationView = mapView.
dequeueReusableAnnotationView(withIdentifier:
identifier) {
annotationView = customAnnotationView
annotationView.annotation = annotation
} else {
let av = MKAnnotationView(annotation:
annotation, reuseIdentifier: identifier)
av.rightCalloutAccessoryView =
UIButton(type: .detailDisclosure)
annotationView = av
}
annotationView.canShowCallout = true
if let image = UIImage(named: "customannotation")
{
annotationView.image = image
annotationView.centerOffset =
CGPoint(x: -image.size.width / 2,
y: -image.size height / 2 )
}
return annotationView
}
}
```

4. Przenieś metody inicjalizacji(), setupMap(_:) i showRestaurantDetail(segue:) do pierwszego rozszerzenia. To powinno wyglądać tak:

```
// MARK: Private Extension
private extension MapViewController {
func initialize() {
mapView.delegate = self
```

```
manager.fetch { (annotations) in
setupMap(annotations) }
}
func setupMap(_ annotations: [RestaurantItem]) {
mapView.setRegion(manager.currentRegion(
latDelta: 0.5, longDelta: 0.5), animated: true)
mapView.addAnnotations(manager.annotations)
}
func showRestaurantDetail(segue:UIStoryboardSegue){
if let viewController = segue.destination as?
RestaurantDetailViewController, let restaurant
= selectedRestaurant {
viewController.selectedRestaurant
= restaurant
}
}
}
```

Wszystkie cztery kontrolery widoku (ExploreViewController, RestaurantListViewController, LocationViewController i MapViewController) zorganizowałeś za pomocą rozszerzeń. Dobra robota!

Streszczenie

Utworzyliśmy nową klasę RestaurantItem, która jest zgodna z protokołem MKAnnotation. Następnie utworzono MapDataManager, klasę menedżera danych, która ładuje dane restauracji z pliku .plist i umieszcza je w tablicy instancji RestaurantItem. Utworzono protokół DataManager i dokonano refaktoryzacji klas MapDataManager i ExploreDataManager, aby używać tego protokołu. Następnie utworzyłeś klasę MapViewController, kontroler widoku dla ekranu Mapa i skonfigurowałeś ją do wyświetlania niestandardowych adnotacji. Skonfigurowałeś przyciski objaśnień w niestandardowych adnotacjach, aby wyświetlały ekran szczegółów restauracji. Następnie utworzyłeś klasę RestaurantDetailViewController, kontroler widoku dla ekranu Szczegóły restauracji i przekazałeś do niej dane z instancji MapViewController. W tym momencie wiesz, jak tworzyć obiekty zgodne z protokołem MKAnnotation, jak dodawać je do widoku mapy i jak tworzyć niestandardowe MKAnnotationViews, które umożliwiają dodawanie map z adnotacjami do własnych projektów. Dodałeś także pliki scenorysów do swojego projektu, nauczyłeś się korzystać z odniesień do scenorysów i uporządkowałeś klasy kontrolerów widoku (ExploreViewController, RestaurantListViewController, LocationViewController i MapViewController) za pomocą rozszerzeń. Pomoże Ci to uporządkować scenorysy i kod dla dużych projektów, dzięki czemu będzie łatwiejszy do odczytania i utrzymania. W następnej części dowiesz się o plikach JSON i o tym, jak załadować z nich

dane, aby ekrany Lista restauracji i Mapa mogły wyświetlać szczegółowe informacje na temat konkretnej restauracji.

Pierwsze kroki z plikami JSON

W poprzedniej części skonfigurowałeś ekran Mapa tak, aby wyświetlał listę restauracji wykorzystującą dane z pliku .plist. Skonfigurowałeś niestandardowe adnotacje dla każdej lokalizacji restauracji i skonfigurowałeś w nich przyciski objaśnień, aby po dotknięciu wyświetlał ekran szczegółów restauracji. Uporządkowałeś także swój kod za pomocą rozszerzeń, aby ułatwić jego czytanie i konserwację. W tym rozdziale użyjesz danych przechowywanych w formacie JavaScript Object Notation (JSON) do wypełnienia ekranów Mapa i Szczegóły restauracji. Zaczniesz od poznania formatu JSON, utworzenia klasy menedżera danych, która może ładować dane z plików JSON i zmodyfikowania klasy MapViewController tak, aby wyświetlała listę restauracji z pliku JSON zamiast z pliku .plist. Następnie skonfigurujesz klasę LocationViewController tak, aby przechowywała lokalizację wybraną przez użytkownika na ekranie Lokalizacje i przekazała ją do instancji ExploreViewController po naciśnięciu przycisku Gotowe. Następnie skonfigurujesz klasę ExploreViewController tak, aby po wybraniu rodzaju kuchni przekazywała wybraną lokalizację i kuchnię do instancji RestaurantListViewController. Na koniec klasa RestaurantListViewController zostanie następnie zmodyfikowana, aby uzyskać listę restauracji z pliku JSON odpowiadającą wybranej lokalizacji i kuchni i wyświetlić je na ekranie Lista restauracji. Na koniec będziesz wiedział, jak ładować i analizować dane z plików JSON do wykorzystania we własnych aplikacjach. Dowiesz się także o metodach UITableViewDelegate i sposobach przekazywania danych z jednego kontrolera widoku do drugiego.

Pobieranie danych z plików JSON

Ddowiedziałeś się, jak załadować plik, odczytać z niego dane za pomocą klasy menedżera danych i umieścić je w obiektach w aplikacji. W tym rozdziale również zrobisz to samo, z tą różnicą, że będziesz czytać dane z pliku JSON, a nie z pliku .plist. Spowoduje to symulację odczytu danych z usługi internetowej, w której powszechnie używanym formatem jest JSON. Zacznijmy od dowiedzenia się więcej o plikach JSON i ich działaniu. Zrozumienie formatu JSON.

Notacja obiektowa JavaScript (JSON) to sposób porządkowania danych w pliku, który może być łatwo odczytany zarówno przez ludzi, jak i komputery. Wiele aplikacji na iOS współpracuje z usługą internetową online, aby uzyskać dostęp do plików JSON, które są następnie wykorzystywane do dostarczania aplikacji danych. W tej części nie dowiesz się, jak połączyć się z usługą online. Zamiast tego użyjesz przykładowych plików JSON pobranych ze strony http://opentable.herokuapp.com, które zostały zmodyfikowane przez Craiga Claytona na potrzeby tej książki. Jak zobaczysz, praca z plikami JSON jest podobna do pracy z plikami .plist. Aby pomóc Ci zrozumieć format JSON, dodasz przykładowe pliki JSON do swojego projektu i spójrz na strukturę jednego z nich. Wykonaj następujące kroki:

1. W nawigatorze projektu utwórz nową grupę w folderze Misc i nadaj jej nazwę JSON.

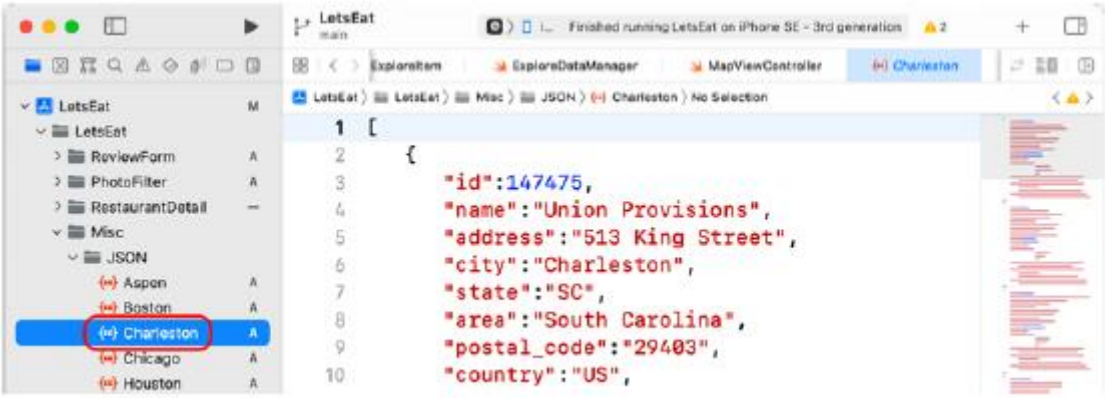
2. Jeśli jeszcze nie pobrałeś plików projektu do tego rozdziału, pobierz je za pomocą tego linku: https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Wydanie siódme.

3. Rozpakuj folder i otwórz folder zasobów w folderze Chapter18. Wewnątrz powinieneś zobaczyć kilka plików JSON.

4. Przeciągnij wszystkie pliki JSON do właśnie utworzonego folderu JSON.

5. Kliknij Zakończ na wyświetlonym ekranie.

6. Każdy plik JSON zawiera dane restauracji dla konkretnego miasta. Kliknij Charleston.json i powinieneś zobaczyć następujące informacje:



Jak widać, plik zaczyna się otwierającym nawiasem kwadratowym, a każdy element w środku składa się z par klucz-wartość zawierających informacje o restauracji, ujętych w nawiasy klamrowe i oddzielonych przecinkami. Na samym końcu pliku widać zamykający nawias kwadratowy. Nawiasy kwadratowe oznaczają tablice, a nawiasy klamrowe oznaczają słowniki. Innymi słowy, plik JSON zawiera tablicę słowników, dokładnie taką samą, jak pliki .plist, których używałeś wcześniej. Teraz, gdy już wiesz, jak wygląda plik JSON, w następnej sekcji utwórzmy klasę menedżera danych, która będzie ładować dane z plików JSON do Twojej aplikacji.

Tworzenie klasy RestaurantDataManager

O tym, jak utworzyć klasę menedżera danych do ładowania danych z pliku .plist, dowiedziałeś się w rozdziale 15, Wprowadzanie danych do widoków kolekcji. Utworzysz teraz RestaurantDataManager, klasę menedżera danych, która ładuje dane z plików JSON dodanych wcześniej do projektu. Jak zobaczysz, ładowanie danych z pliku JSON jest podobne do ładowania danych z pliku .plist. Zanim utworzysz klasę RestaurantDataManager, musisz zmodyfikować klasę RestaurantItem, tak aby była zgodna z protokołem Decodable. Przyjęcie tego protokołu umożliwia użycie klasy JSONDecoder do zapełniania instancji RestaurantItem przy użyciu danych z plików JSON. Aby zmodyfikować klasę RestaurantItem tak, aby była zgodna z protokołem Decodable, wykonaj następujące kroki:

1. W nawigatorze projektu kliknij plik RestaurantItem w folderze Model w folderze Map. Zmodyfikuj deklarację klasy dla RestaurantItem, jak pokazano, aby przyjąć protokół Decodable:

class RestaurantItem: NSObject, MKAnnotation,

Decodable {

2. Usuń metodę init() i dodaj następujące wyliczenie, aby klasa RestaurantItem była zgodna z protokołem Decodable:

enum CodingKeys: String, CodingKey {

case name

case cuisines

case lat

case long

case address

```
case postalCode = "postal_code"

case state

case imageURL = "image_url"

case restaurantID = "id"

}
```

Wyliczenie CodingKeys dopasowuje właściwości klasy RestaurantItem do kluczy w pliku JSON. Dzięki temu instancja JSONDecoder może pobrać wartości z pliku JSON i przypisać je do właściwości w klasie RestaurantItem. Jeśli nazwa klucza nie jest zgodna z nazwą właściwości, możesz zamapować klucz na właściwość, jak pokazano w poprzednim bloku kodu dla postalCode, imageURL i RestaurantID. Po zmodyfikowaniu klasy RestaurantItem zobaczysz błąd w metodzie fetch(completion:) w pliku MapDataManager. Nie martw się tym, ponieważ naprawisz to w następnej sekcji. Stwórzmy teraz klasę RestaurantDataManager, która odczyta plik JSON i umieści dane w tablicy instancji RestaurantItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Restauracje i utwórz nową grupę o nazwie Model. Następnie kliknij prawym przyciskiem myszy folder Model i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik RestaurantDataManager. Kliknij Utwórz. Plik RestaurantDataManager pojawi się w nawigatorze projektu.

4. Dodaj następujący wpis po instrukcji importu, aby zadeklarować klasę RestaurantDataManager:

RestaurantDataManager class:

```
class RestaurantDataManager {

}
```

5. Dodaj następującą właściwość pomiędzy nawiasami klamrowymi, aby przechowywać tablicę instancji RestaurantItem:

```
private var restaurantItems: [RestaurantItem] = []
```

Tablica RestaurantItems będzie przechowywać instancje RestaurantItem uzyskane z pliku JSON. Słowo kluczowe private oznacza, że jest ono dostępne tylko z poziomu tej klasy.

6. Dodaj następującą metodę po właściwości RestaurantItems, aby odczytać plik JSON i zwrócić tablicę instancji RestaurantItem:

```
func fetch(location: String, selectedCuisine:

String = "All", completionHandler: (_

restaurantItems: [RestaurantItem]) -> Void) {

if let file = Bundle.main.url(forResource:

location, withExtension: "json") {

do {
```

```
let data = try Data(contentsOf: file)

let restaurants = try JSONDecoder().decode(

[RestaurantItem].self, from: data)

if selectedCuisine != "All" {

restaurantItems = restaurants.filter {

($0.cuisines.contains(selectedCuisine))

}

} else {

restaurantItems = restaurants

}

} catch {

print("There was an error \(error)")

}

}

completionHandler(restaurantItems)

}
```

Rozbijmy to:

```
func fetch(location: String, selectedCuisine:

String = "All", completionHandler: (_ restaurantItems:

[RestaurantItem]) -> Void)
```

Metoda ta przyjmuje trzy parametry: lokalizację, ciąg znaków zawierający lokalizację restauracji, wybraneCuisine, ciąg znaków zawierający kuchnię wybraną przez użytkownika oraz zakończeniuHandler, zamknięcie służące do przetworzenia wyniku tej metody po zakończeniu jej wykonywania. Jeśli nie podasz wartości dla wybranej kuchni, domyślnie zostanie ona ustawiona na „Wszystkie”.

```
if let file = Bundle.main.url(forResource: location,

withExtension: "json")
```

Spowoduje to pobranie adresu URL pliku JSON w pakiecie aplikacji i przypisanie go do pliku.

do code block:

Pierwsza instrukcja próbuje przypisać zawartość pliku do danych. Następna instrukcja próbuje użyć instancji JSONDecoder do przeanalizowania danych i zapisania ich jako tablicy instancji RestaurantItem, która jest przypisana do restauracji. W następnym oświadczeniu, jeśli zostanie wybranaKuchnia jest not All, metoda filtrowania jest stosowana do tablicy restauracji przy użyciu zamknięcia {($0.cuisines. zawiera(selectedCuisine))}. W rezultacie powstaje tablica instancji

RestaurantItem, w których właściwość cuisines zawiera kuchnię wybraną przez użytkownika, a tablica ta jest przypisana do restauracjiItems. W przeciwnym razie cała tablica restauracje jest przypisana do restauracjiItems.

catch code block:

Spowoduje to wyświetlenie komunikatu o błędzie w obszarze Debugowanie, jeśli blok kodu do nie powiedzie się.

completionHandler(restaurantItems)

Ta instrukcja przetwarza tablicę RestaurantItems przy użyciu dostarczonego zamknięcia. Kiedy wywołujesz tę metodę w Xcode, funkcja autouzupełniania daje dwie możliwe możliwości; taki, który zawiera wybrany parametrCuisine: (który pobiera ciąg znaków zawierający wybraną kuchnię) i taki, który go nie zawiera (selectedCuisine jest ustawiony na All).

7. Po metodzie fetch(location:selectedCuisine:completionHandler:) dodaj metodę zwracającą liczbę pozycji w tablicy RestaurantItems:

```
func numberOfRestaurantItems() -> Int {

restaurantItems.count

}
```

Metodę tę wywołasz, aby określić liczbę komórek widoku kolekcji, które mają być wyświetlane na ekranie Restaurant List

8. Zaraz po metodzie numberOfRestaurantItems() dodaj metodę, która zwróci instancję RestaurantItem z tablicy RestaurantItems pod podanym indeksem:

```
func restaurantItem(at index: Int) ->

RestaurantItem {

restaurantItems[index]

}
```

Metodę tę wywołasz, aby skonfigurować zawartość każdej komórki widoku kolekcji na ekranie Lista restauracji. Utworzono klasę RestaurantDataManager, która umożliwia odczytanie danych zapisanych w plikach JSON i umieszczenie ich w tablicy instancji RestaurantItem. Zanim będziesz mógł z niego skorzystać, będziesz musiał nieco zmodyfikować swój projekt. Zobaczmy, co jest wymagane do wyświetlenia informacji o restauracji na ekranach Mapa i Lista restauracji w następnej sekcji.

Korzystanie z danych z plików JSON w Twojej aplikacji

Przyjrzyjmy się, jak działa aplikacja. Na ekranie Mapa użytkownik zobaczy wszystkie restauracje w pobliżu jego lokalizacji. Dotknięcie restauracji spowoduje wyświetlenie dymka z objaśnieniem, a dotknięcie przycisku objaśnienia spowoduje wyświetlenie szczegółów danej restauracji na ekranie szczegółów restauracji. Na ekranie Eksploruj użytkownik dotknie przycisku LOKALIZACJA i wybierz lokalizację, taką jak Charleston, Karolina Północna na ekranie Lokalizacje. Po wybraniu lokalizacji użytkownik klika Gotowe i powraca do ekranu Eksploruj. Następnie użytkownik wybierze kuchnię na ekranie Eksploruj, a lista restauracji w tej lokalizacji oferujących tę kuchnię zostanie wyświetlona na ekranie Lista restauracji. Dotknięcie restauracji spowoduje wyświetlenie jej szczegółów na ekranie Szczegóły restauracji. Wykonasz następujące czynności:

• Skonfiguruj klasę MapViewController, aby uzyskać listę restauracji z pliku JSON zamiast z pliku .plist. Spowoduje to również naprawienie błędu w metodzie fetch(completion:) klasy MapDataManager.

• Skonfiguruj klasę LocationViewController tak, aby przechowywała lokalizację wybraną przez użytkownika.

• Przekaż wybraną lokalizację do instancji ExploreViewController.

• Skonfiguruj klasę ExploreViewController, aby przekazywać wybraną lokalizację i kuchnię do instancji RestaurantListViewController.

• Skonfiguruj klasę RestaurantListViewController, aby uzyskać listę restauracji z pliku JSON odpowiadającą wybranej lokalizacji.

• Skonfiguruj klasę RestaurantListViewController, aby wyświetlała listę restauracji na podstawie wybranej lokalizacji i kuchni.

Może to wydawać się trudne, więc będziesz robić wszystko krok po kroku. Na początek skonfigurujesz klasę MapDataManager tak, aby odczytywała dane z plików JSON zamiast z plików .plist.

Konfigurowanie instancji MapDataManager do korzystania z danych z instancji RestaurantDataManager

Obecnie występuje błąd w pliku MapDataManager. Dzieje się tak, ponieważ metoda fetch(completion:) w klasie MapDataManager wywołuje metodę inicjującą, którą usunąłeś z klasy RestaurantItem. Teraz zaktualizujesz klasę MapDataManager, aby używała instancji RestaurantDataManager jako źródła danych, naprawiając błąd w procesie. Kliknij plik MapDataManager (w folderze Model w folderze Map) w nawigatorze projektu i zaktualizuj metodę fetch(completion:) w następujący sposób:

```
func fetch(completion: (_ annotations: [RestaurantItem]) -> ()){

let manager = RestaurantDataManager()

manager.fetch(location: "Boston", completionHandler: {

(restaurantItems) in self.items = restaurantItems

completion(items)

})

}
```

Rozbijmy to:

```
func fetch(completion: (_ annotations: [RestaurantItem]) -> ())
```

Metoda ta posiada parametr metody uzupełniania. Metoda uzupełniania zostanie użyta do przetworzenia wyniku po zakończeniu wykonywania metody.

```
let manager = RestaurantDataManager()
```

Spowoduje to utworzenie instancji klasy RestaurantDataManager i przypisanie jej do menedżera.

```
manager.fetch(location: "Boston", completionHandler: {

(restaurantItems) in self.items = restaurantItems
```

```
completion(items)

})
```

Wywołuje to metodę fetch() instancji menedżera w celu pobrania listy restauracji z pliku Boston.json. Na razie jest to zakodowane na stałe, ponieważ symulator iOS nie ma funkcjonalnego GPS. Aby zobaczyć restauracje w innej lokalizacji, zmień nazwę używanego pliku JSON na inną lokalizację. Tablica instancji RestaurantItem zwrócona przez tę metodę jest przypisana do tablicy items instancji MapViewController, a przekazana metoda uzupełniania jest używana do przetwarzania tej tablicy. Jak widziałeś w poprzedniej części, wygeneruje to adnotacje, które zostaną dodane do widoku mapy na ekranie Mapa.

Jeśli uruchomisz teraz aplikację i wybierzesz ekran Mapa, powinieneś zobaczyć pinezki restauracji w Bostonie. W następnej sekcji skonfigurujesz klasę LocationViewController tak, aby mogła przechowywać lokalizację wybraną przez użytkownika.

Zapisywanie lokalizacji wybranej przez użytkownika

Obecnie klasa LocationDataManager ładuje dane z Locations.plist i przechowuje informacje o lokalizacji w tablicy ciągów znaków. Utworzysz nową strukturę LocationItem i skonfigurujesz klasę LocationDataManager do przechowywania lokalizacji w tablicy instancji LocationItem. Następnie zmodyfikujesz klasę LocationViewController tak, aby mogła przechowywać instancję LocationItem zawierającą lokalizację wybraną przez użytkownika. Następnie możesz przekazać tę instancję do instancji RestaurantListViewController w swojej aplikacji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze Lokalizacja i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik LocationItem. Kliknij Utwórz. Plik LocationItem pojawi się w nawigatorze projektu.

4. Kliknij plik LocationItem i po instrukcji import dodaj następujący wpis, aby zadeklarować i zdefiniować strukturę LocationItem:

```
struct LocationItem {

let city: String?

let state: String?

}

extension LocationItem {

init(dict: [String: String]) {

self.city = dict["city"]

self.state = dict["state"]

}

var cityAndState: String {

guard let city = self.city, let state =

self.state else {
```

```
return ""

}

return "\(city), \(state)"

}

}
```

Struktura LocationItem ma dwie właściwości String, miasto i stan. Metoda init() przyjmuje słownik dict jako parametr i przypisuje wartości kluczy miasta i stanu do właściwości miasta i stanu. Obliczona właściwość cityAndState zwraca ciąg znaków utworzony z połączenia wartości miasta i stanu. Teraz zaktualizujesz klasę LocationDataManager, tak aby mogła przechowywać informacje o mieście i stanie w tablicy instancji LocationItem zamiast w ciągach znaków. Wykonaj następujące kroki:

1. Kliknij menedżera LocationDataManager w nawigatorze projektu i zmodyfikuj tablicę lokalizacji, aby przechowywać instancje LocationItem zamiast ciągów znaków:

```
private var locations:[LocationItem] = []
```

2. Metoda fetch() wyświetli teraz błąd. Zmodyfikuj metodę fetch(), aby działała z instancjami LocationItem zamiast z ciągami znaków:

```
for location in loadData() {

locations.append(LocationItem(dict: location))

}

}
```

Każdy słownik udostępniany przez metodę loadingData() jest teraz używany do inicjowania instancji LocationItem zamiast ciągów znaków, które są następnie dołączane do tablicy LocationItem. Metoda LoadData() w dalszym ciągu korzysta z tego samego pliku Locations.plist, który utworzyłeś w rozdziale 16, Pierwsze kroki z widokami tabel.

3. Metoda LocationItem(at:) wyświetla teraz błąd. Zmodyfikuj go tak, aby zamiast ciągu zwracał instancję LocationItem:

```
func locationItem(at index: Int) ->

LocationItem {

locations[index]

}
```

W tym momencie skończyłeś z klasą LocationDataManager. Następnie zaktualizujesz klasę LocationViewController, aby do wypełniania komórek widoku tabeli używała instancji LocationItem zamiast ciągów znaków. Wykonaj następujące kroki:

1. Kliknij plik LocationViewController w nawigatorze projektu.

2. Zobaczysz błąd w metodzie tableView(_:cellForRowAtIndexPath:). Ten błąd wynika z tego, że nie można przypisać instancji LocationItem do właściwości textLabel komórki. Zmodyfikuj tę metodę w następujący sposób:

```
func tableView(_ tableView: UITableView, cellForRowAt

indexPath: IndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCell(

withIdentifier: "locationCell", for: indexPath)

let location = manager.locationItem(at:

indexPath.row)

cell.textLabel?.text = location.cityAndState

return cell

}
```

Właściwość cityAndState struktury LocationItem zwraca ciąg znaków, który łączy ciągi znaków miasta i stanu lokalizacji, naprawiając błąd.

3. Potrzebujesz właściwości, aby śledzić wybór użytkownika. Dodaj następującą deklarację właściwości zaraz po deklaracji menedżera:

```
let manager = LocationDataManager()

var selectedCity: LocationItem?
```

Aby obsłużyć interakcję użytkownika z widokiem tabeli, należy dostosować klasę LocationViewController do protokołu UITableViewDelegate.

Protokół UITableViewDelegate określa komunikaty, które widok tabeli wyśle do swojego delegata, gdy użytkownik wejdzie w interakcję z zawartymi w nim wierszami. Aby go przyjąć, wykonaj następujące kroki:

1. Dodaj następujące rozszerzenie po rozszerzeniu UITableViewDataSource:

```
// MARK: UITableViewDelegate

extension LocationViewController:

UITableViewDelegate {

}
```

Rozszerzenie pomaga utrzymać porządek w kodzie, a składnia //MARK: tworzy to rozszerzenie które łatwo znaleźć w obszarze Edytora.

2. Metoda UITableViewDelegate wywoływana, gdy użytkownik dotknie wiersza w widoku tabeli, to tableView(_:didSelectRowAt:). Dodaj tę metodę między nawiasami klamrowymi rozszerzenia. Powinno to wyglądać następująco:

```
// MARK: UITableViewDelegate

extension LocationViewController: UITableViewDelegate

{
```

```
func tableView(_ tableView: UITableView,

didSelectRowAt indexPath:IndexPath) {

if let cell = tableView.cellForRow(at:

indexPath) {

cell.accessoryType = .checkmark

selectedCity =

manager.locationItem(at: indexPath.row)

}

}

}
```

Gdy użytkownik dotknie wiersza na ekranie Lokalizacje, w tym wierszu pojawi się znacznik wyboru, a właściwość wybraneCity zostanie przypisana do odpowiedniej instancji LocationItem w tablicy lokalizacji. Na przykład, jeśli dotkniesz trzeciego wiersza, instancja LocationItem z wartościami „Charleston" i „NC" zostanie przypisana do wybranego miasta. Klasa LocationViewController może teraz przechowywać lokalizację wybraną przez użytkownika, ale gdy wybierzesz lokalizację na ekranie Lokalizacje i klikniesz Gotowe, nic się nie dzieje. Dzieje się tak, ponieważ nie utworzyłeś jeszcze ani nie przypisałeś akcji do przycisku Gotowe. W następnej sekcji utworzysz akcję unwind w klasie ExploreViewController i przypiszesz ją do przycisku Gotowe, ale zanim to zrobisz, najpierw utworzysz nowy kontroler widoku dla nagłówka sekcji widoku kolekcji na ekranie Eksploruj. Umożliwi to wyświetlenie wybranej przez użytkownika lokalizacji na ekranie Eksploruj.

Dodawanie podklasy UICollectionReusableView dla nagłówka sekcji na ekranie Eksploruj

Widok kolekcji na ekranie Eksploruj zawiera nagłówek sekcji widoku kolekcji, który zawiera etykietę podtytułu, etykietę tytułu i przycisk LOKALIZACJA. Obecnie nie ma możliwości ustawienia tekstu etykiety napisów. Utworzysz podklasę UICollectionReusableView dla nagłówka sekcji widoku kolekcji i skonfigurujesz wylot dla etykiety podtytułu, dzięki czemu będziesz mógł wyświetlić wybraną przez użytkownika lokalizację w nagłówku sekcji widoku kolekcji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Widok w folderze Eksploruj i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: ExploreHeaderView

Podklasa: UICollectionReusableView

Utwórz także XIB: Niezaznaczone

Język: Swift

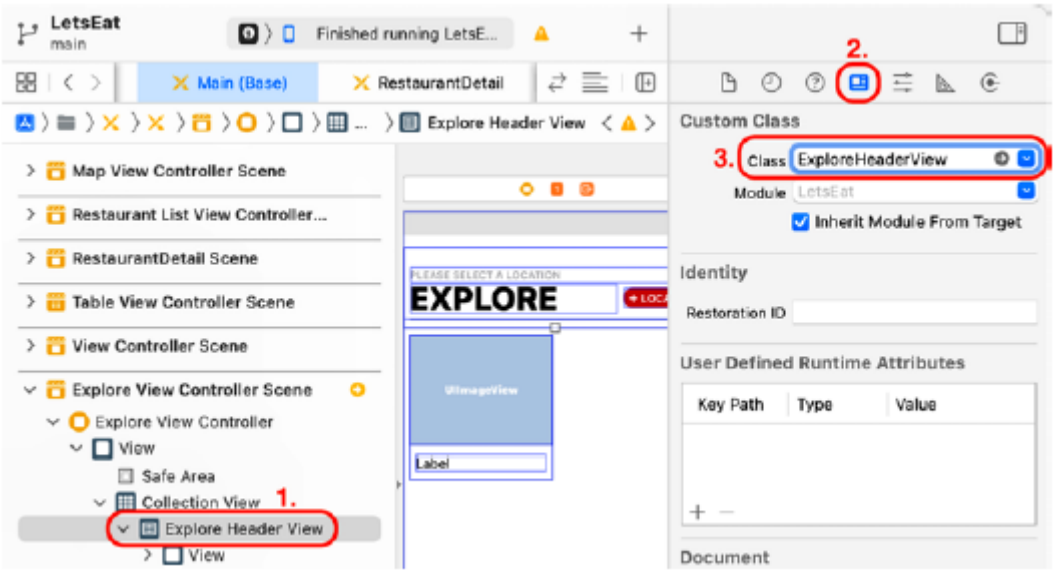
Kliknij Następny.

4. Kliknij Utwórz. Plik ExploreHeaderView pojawi się w nawigatorze projektu. Sprawdź, czy zawiera następujące elementy:

class ExploreHeaderView: UICollectionReusableView {

}

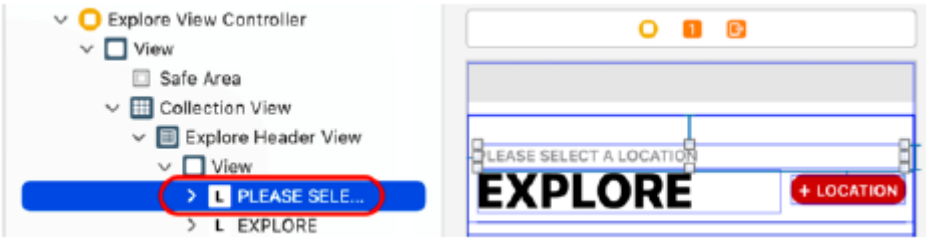
5. Kliknij plik głównego scenorysu w nawigatorze projektu. Wybierz widok kolekcji do ponownego użycia w scenie kontrolera widoku Eksploruj. Kliknij przycisk Inspektora tożsamości i w obszarze Klasa niestandardowa ustaw opcję Klasa na ExploreHeaderView:



Należy pamiętać, że widok kolekcji do ponownego wykorzystania zmieni się w widoku nagłówka Eksploruj w konspekcie dokumentu. Klasa ExploreHeaderView zarządza teraz nagłówkiem sekcji widoku kolekcji. W następnej sekcji połączysz etykietę napisów w nagłówku sekcji widoku kolekcji z gniazdem w klasie ExploreHeaderView i dodasz właściwość dla klasy ExploreHeaderView w klasie ExploreViewController. Podłączenie etykiety nagłówka sekcji do klasy ExploreViewController

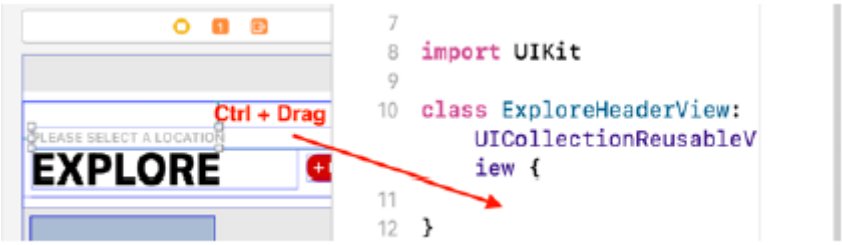
Aby wyświetlić miasto wybrane przez użytkownika w nagłówku sekcji widoku kolekcji, połącz etykietę napisów z gniazdem w klasie ExploreHeaderView, a następnie dodaj jej właściwość w klasie ExploreViewController. Wykonaj następujące kroki:

1. W konspekcie dokumentu kliknij etykietę podtytułu widoku nagłówka Eksploruj (jest to etykieta zawierająca tekst PROSZĘ WYBRAĆ LOKALIZACJĘ):



2. Kliknij przycisk Dostosuj opcje edytora i wybierz z menu opcję Asystent. Po prawej stronie ekranu pojawi się asystent edytora. Upewnij się, że pokazuje zawartość pliku ExploreHeaderView.

3. Ctrl + Przeciągnij etykietę PROSZĘ WYBRAĆ LOKALIZACJĘ do miejsca pomiędzy nawiasami klamrowymi:



4. W wyświetlonym oknie wpisz nazwę lokalizacjiLabel i kliknij Połącz.

5. W klasie ExploreHeaderView utworzono outletlocationLabel. Zamknij edytor asystenta, klikając przycisk x.

6. Kliknij plik ExploreViewController w nawigatorze projektu. Dodaj następującą deklarację właściwości po deklaracji menedżera, aby przechowywać lokalizację przekazaną do instancji ExploreViewController przez instancję LocationViewController:

let manager = ExploreDataManager()

var selectedCity: LocationItem?

7. Po deklaracji właściwości wybranegoCity zadeklaruj właściwość, do której zostanie przypisana instancja ExploreHeaderView, która umożliwi instancji ExploreViewController ustawienie wartości dla LocationLabel:

let manager = ExploreDataManager()

var selectedCity: LocationItem?

var headerView: ExploreHeaderView!

Następnie skonfigurujmy przycisk Gotowe, aby zamknąć ekran Lokalizacje i po dotknięciu ustawić etykietę lokalizacji na wybrane przez użytkownika miasto i stan. Następnie pojawi się ekran Eksploruj z wybranym miastem wyświetlonym w nagłówku sekcji widoku kolekcji. Zrobisz to w następnej sekcji.

Dodanie metody akcji unwind do przycisku Gotowe

Dodaliśmy metodę akcji unwind dla przycisku Anuluj w klasie ExploreViewController, która po dotknięciu zamyka ekran Lokalizacje. Teraz, dodasz metodę akcji unwind dla przycisku Gotowe, która zamknie ekran Lokalizacje i ustawi właściwość wybraneCity w instancji ExploreViewController na lokalizację wybraną przez użytkownika. Wykonaj następujące kroki:

1. Dodaj następujący tekst zaraz po metodzie unwindLocationCancel(segue:), aby zaimplementować akcję unwind dla przycisku Gotowe:

@IBAction func

unwindLocationDone(segue: UIStoryboardSegue) {

if let viewController = segue.source as?

LocationViewController {

```
selectedCity = viewController.selectedCity

if let location = selectedCity {

headerView.locationLabel.text =

location.cityAndState

}

}

}
```

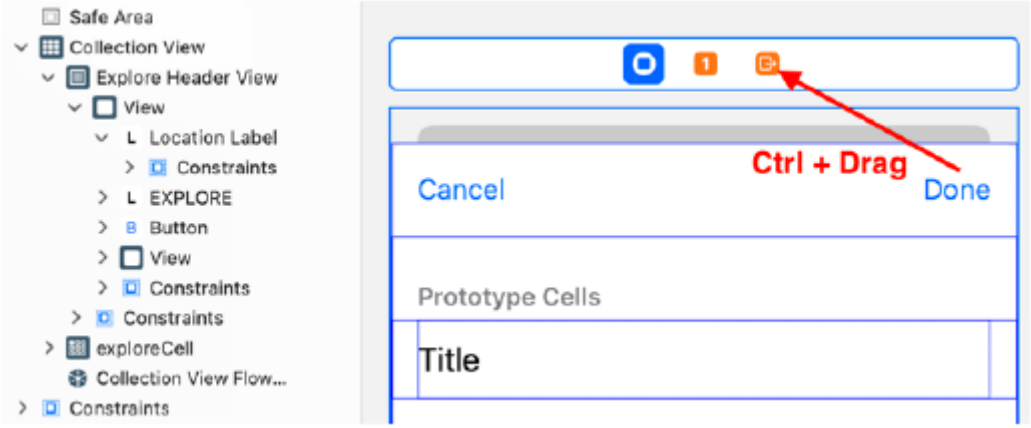
Później przypiszesz tę metodę do przycisku Gotowe. Kontrolerem widoku źródłowego w tym przypadku jest instancja LocationViewController, a kontrolerem widoku docelowego jest instancja ExploreViewController. Ta metoda najpierw sprawdza, czy kontroler widoku źródłowego jest instancją LocationViewController. Jeśli tak, wartość właściwości wybraneCity instancji LocationViewController jest przypisana do właściwości wybraneCity instancji ExploreViewController, jeśli taka istnieje. Jeśli właściwość wybraneCity instancji ExploreViewController ma wartość, jest ona przypisana do lokalizacji, a tekst etykiety napisów jest ustawiany na właściwość cityAndState lokalizacji.

2. Zmodyfikuj metodę CollectionView(_:viewForSupplementaryElementOfKind:at:) w następujący sposób:

```
func collectionView(_ collectionView: UICollectionView,

viewForSupplementaryElementOfKind kind: String, at

indexPath: IndexPath) -> UICollectionReusableView {

let header =

collectionView.dequeueReusableSupplementaryView

(ofKind: kind, withReuseIdentifier: "header", for:

indexPath)

headerView = header as? ExploreHeaderView

return headerView

}
```

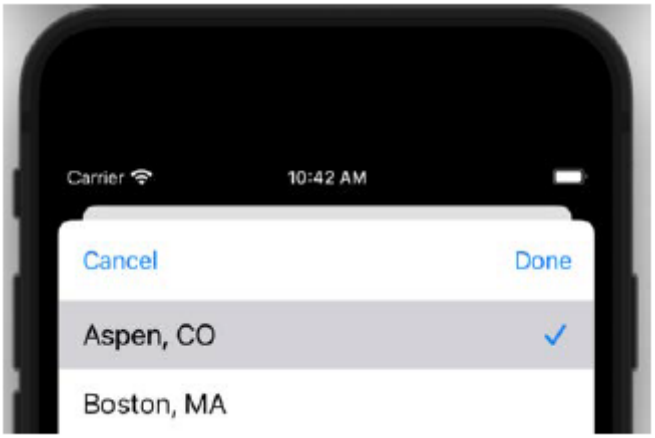
Ta metoda jest jedną z metod źródła danych zadeklarowanych w protokole UICollectionViewDataSource. Zwraca widok, który będzie używany jako nagłówek sekcji widoku kolekcji. W tym przypadku nagłówek sekcji widoku kolekcji na ekranie Eksploruj jest ustawiony na instancję ExploreHeaderView.

3. Kliknij plik głównego scenorysu. Wybierz scenę kontrolera widoku lokalizacji. Aby ustawić metodę wyzwalaną przez przycisk Gotowe, Ctrl + przeciągnij z przycisku Gotowe do ikony Wyjdź w Doku Sceny:

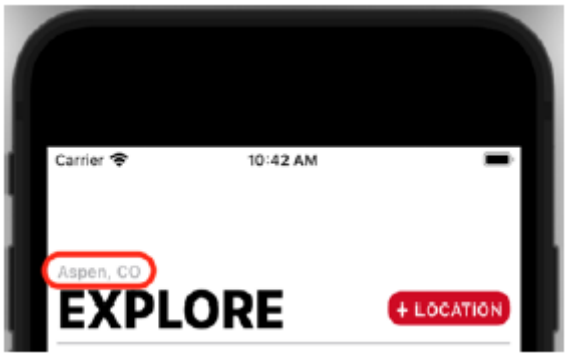


4. Wybierz unwindLocationDoneWithSegue: z wyskakującego menu. Spowoduje to połączenie przycisku Gotowe z akcją unwindLocationDone(segue:) unwind w klasie ExploreViewController:

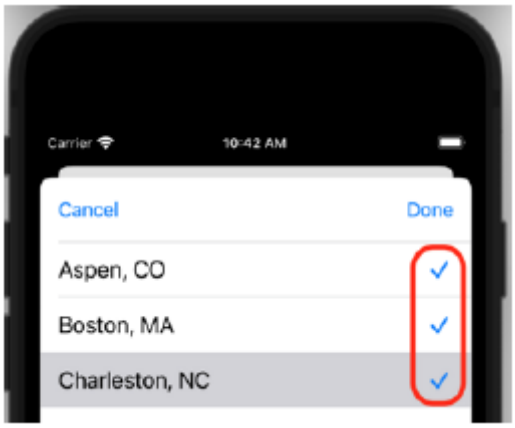
Zbuduj i uruchom aplikację, a następnie dotknij przycisku LOKALIZACJA. Kliknij miasto, a w wierszu pojawi się znacznik wyboru. Kliknij Gotowe:



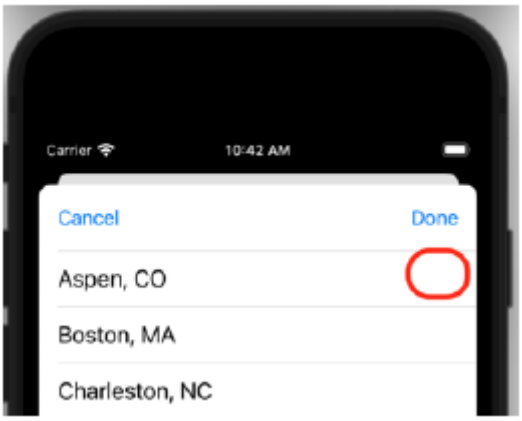
Wybrana nazwa miasta i stan zastąpią tekst PROSZĘ WYBRAĆ LOKALIZACJĘ na etykiecie napisów wewnątrz nagłówka sekcji widoku kolekcji:



Chociaż to działa, istnieją dwa problemy, które należy rozwiązać przy wyborze lokalizacji. Pierwszą kwestią jest to, że możesz wybrać wiele lokalizacji:



Chcesz, aby użytkownik wybrał tylko jedną lokalizację, a jeśli zostanie wybrana inna, wcześniej wybrana lokalizacja powinna zostać odznaczona. Drugi problem polega na tym, że znacznik wyboru obok lokalizacji wybranej przez użytkownika znika, jeśli klikniesz Gotowe na ekranie Lokalizacje i ponownie klikniesz przycisk LOKALIZACJA na ekranie Eksploruj:



Gdy ponownie przejdziesz do ekranu Lokalizacje, ostatnia wybrana lokalizacja powinna być zaznaczona. Zmodyfikujmy klasę LocationDataManager, aby mieć pewność, że w następnej sekcji można wybrać tylko jedną lokalizację naraz.

Wybieranie tylko jednej lokalizacji na ekranie Lokalizacje

Obecnie na ekranie Lokalizacje można wybrać wiele lokalizacji. Powinieneś mieć możliwość wybrania tylko jednej lokalizacji, a jeśli zostanie wybrana inna, poprzednio wybrana lokalizacja powinna zostać odznaczona. Wykonaj następujące kroki:

1. Kliknij plik LocationItem. Dostosuj LocationItem do protokołu Equatable, modyfikując strukturę LocationItem w następujący sposób:

struktura LocationItem: Równa {

struct LocationItem: Equatable {

let city: String?

let state: String?

}

Pozwala to sprawdzić, czy dwie instancje LocationItem są równe.

2. Kliknij plik LocationViewController. Po metodzie viewDidLoad() dodaj metodę, która ustawia znacznik tylko w wierszu zawierającym wybrane miasto:

private func setCheckmark(for cell: UITableViewCell,

location: LocationItem) {

if selectedCity == location {

cell.accessoryType = .checkmark

} else {

cell.accessoryType = .none

}

}

Metoda setCheckmark(for:location:) przyjmuje jako argumenty komórkę, komórkę widoku tabeli i lokalizację, instancję LocationItem. Jeśli lokalizacja i wybrane miasto są równe, znacznik wyboru dla tego wiersza jest ustawiony. W przeciwnym razie znacznik wyboru nie jest ustawiony.

3. W metodzie tableView(_:cellForRowAt:) zmodyfikuj kod w następujący sposób, aby wywołać setCheckmark(for:location:) po wierszu ustawiającym tekst właściwości textLabel komórki w następujący sposób:

func tableView(_ tableView: UITableView, cellForRowAt

indexPath: IndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCell(

withIdentifier: "locationCell", for: indexPath)

let location = manager.locationItem(at

indexPath.row)

```
cell.textLabel?.text = location.cityAndState

setCheckmark(for: cell, location: location)

return cell

}
```

Oznacza to, że setCheckmark(for:at:) zostanie wywołany podczas renderowania każdego wiersza w widoku tabeli, a znacznik wyboru zostanie ustawiony tylko w wierszu zawierającym wybraną lokalizację.

4. W metodzie tableView(_:didSelectRowAt:) zmodyfikuj kod w instrukcji if w następujący sposób, aby ponownie załadować widok tabeli (w ten sposób renderując wszystkie zawarte w nim wiersze) po wybraniu lokalizacji:

```
if let cell = tableView.cellForRow(at: indexPath) {

cell.accessoryType = .checkmark

selectedCity = manager.locationItem(at:

indexPath.row)

tableView.reloadData()

}
```

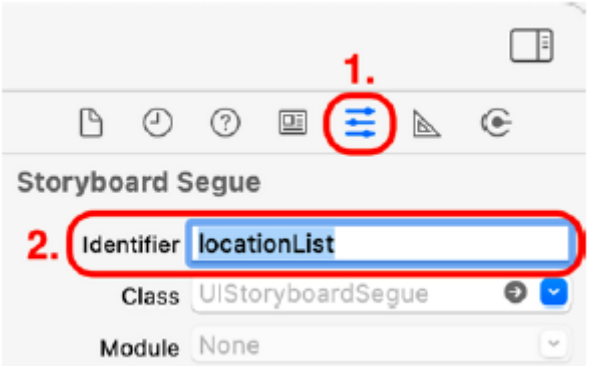
Zbuduj i uruchom swój projekt. Teraz powinno być możliwe ustawienie tylko jednej lokalizacji. Jeśli wybierzesz inną lokalizację, wybrana wcześniej lokalizacja zostanie odznaczona. Drugi problem rozwiążesz w następnej sekcji, dzięki czemu po wybraniu lokalizacji pozostanie ona wybrana, gdy wrócisz do ekranu Lokalizacje. Przekażesz także informacje o lokalizacji i kuchni do instancji RestaurantListViewController, dzięki czemu może ona ostatecznie wyświetlić listę restauracji w określonej lokalizacji pasujących do kuchni wybranej przez użytkownika.

Przekazywanie informacji o lokalizacji i kuchni do instancji RestaurantListViewController

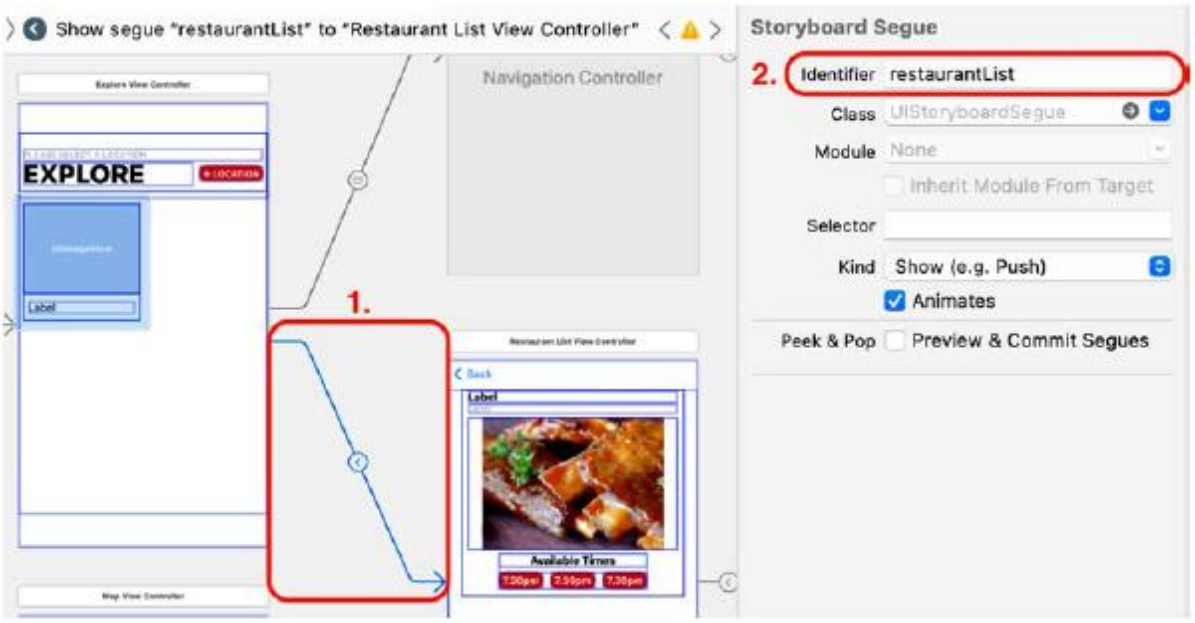
Obecnie możesz ustawić lokalizację na ekranie Lokalizacje i wyświetlić ją w nagłówku sekcji widoku kolekcji na ekranie Eksploruj. Teraz dodasz kod, dzięki któremu będziesz mógł przekazać wartości lokalizacji i kuchni do instancji RestaurantListViewController, która następnie wyświetli restauracje w wybranej lokalizacji oferujące wybraną kuchnię. Sprawisz także, że znacznik wyboru obok wybranej lokalizacji pojawi się ponownie, jeśli wcześniej wybrałeś lokalizację na ekranie Lokalizacje. Wykonaj następujące kroki:

1. Dodasz identyfikatory dla każdego przejścia podłączonego do ekranu Eksploruj, aby wiedzieć, które przejście nastąpi później. Otwórz plik głównego scenorysu i kliknij opcję Eksploruj scenę kontrolera widoku. Wybierz przejście pomiędzy eksploracją sceny kontrolera widoku a sceną kontrolera widoku lokalizacji:

2. Kliknij przycisk Inspektora atrybutów. W obszarze Storyboard Segue ustaw Identyfikator na LocationList:



3. Wybierz opcję przeglądania sceny kontrolera widoku i sceny kontrolera widoku listy restauracji. Kliknij przycisk Inspektora atrybutów. W obszarze Storyboard Segue ustaw Identyfikator na RestaurantList:



Możesz teraz użyć identyfikatorów przejścia, aby określić, które przejście ma miejsce. Gdy już wiesz, który ciąg ma miejsce, możesz określić metody, które mają być wykonane dla każdego przejścia. Utworzysz dwie metody, showLocationList(segue:) i showRestaurantList(segue:), a następnie użyjesz metody przygotowania(for:sender:) do wykonania żądanej metody w zależności od tego, która sekwencja ma miejsce. Kliknij plik ExploreViewController w nawigatorze projektu. Wewnątrz rozszerzenia prywatnego zadeklaruj i zdefiniuj metodę showLocationList(segue:) przed metodą unwindLocationCancel() w celu przekazania wybranego przez użytkownika miasta z powrotem do instancji LocationViewController, jeśli zostało ustawione wcześniej:

```
func showLocationList(segue: UIStoryboardSegue) {

guard let navController = segue.destination as?

UINavigationController, let viewController =

navController.topViewController as?

LocationViewController else {

return

}

viewController.selectedCity = selectedCity

}
```

Metoda showLocationList(segue:) zostanie wywołana przed przejściem ekranu Eksploracja do ekranu Lokalizacje. Zobaczmy jak to działa. Pamiętaj, że w głównym pliku scenorysu osadziłeś scenę kontrolera widoku lokalizacji w kontrolerze nawigacji. kontroler nawigacyjny ma właściwość viewControllers, która przechowuje tablicę kontrolerów widoku, a widok ostatniego kontrolera widoku w tablicy jest widoczny na ekranie. Dostęp do ostatniego kontrolera widoku w tablicy można uzyskać za pomocą właściwości topViewController kontrolera nawigacyjnego. Instrukcja Guard sprawdza, czy miejsce docelowe przejścia jest instancją UINavigationController i czy topViewController jest instancją

LocationViewController. Jeśli tak jest, sprawdzana jest właściwość wybraneCity instancji ExploreViewController, aby sprawdzić, czy zawiera ona wartość. Jeśli tak, ta wartość jest przypisana do właściwości wybraneCity instancji LocationViewController. Pamiętaj, że dla każdego wiersza w widoku tabeli zostanie wywołana metoda setCheckmark(for:at:) instancji LocationViewController, co spowoduje ustawienie znacznika wyboru w wierszu zawierającym wybrane miasto. To rozwiąże drugi problem z ekranem Lokalizacje. Dodajmy teraz kod, który przekaże lokalizację i kuchnię do instancji RestaurantListViewController. Wykonaj następujące kroki:

1. Kliknij plik RestaurantListViewController w nawigatorze projektu. Dodaj następujące właściwości wewnątrz klasy RestaurantListViewController tuż przed deklaracją @IBOutlet:

```
var selectedRestaurant: RestaurantItem?

var selectedCity: LocationItem?

var selectedCuisine: String?

@IBOutlet var collectionView: UICollectionView!
```

selectedRestaurant zostanie ustawiona, jeśli wybierzesz restaurację na ekranie Lista restauracji i zostanie później przekazana do ekranu Szczegóły restauracji. Kod, który to zrobi, dodasz w następnej części.

selectedCity przechowuje miasto wybrane na ekranie Lokalizacje.

selectedCuisine przechowuje kuchnię wybraną na ekranie Eksploruj.

2. Dodaj następujący kod po metodzie viewDidLoad(), aby wydrukować wybrane miasto i kuchnię w obszarze Debug, gdy na ekranie pojawi się widok instancji RestaurantListViewController:

```
override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

print("selected city \(selectedCity as Any)")

print("selected cuisine \(selectedCuisine as Any)")

}
```

Funkcja viewDidAppear() jest wywoływana za każdym razem, gdy na ekranie pojawia się widok kontrolera widoku, natomiast funkcja viewDidLoad() jest wywoływana tylko raz, gdy kontroler widoku początkowo ładuje swój widok. Użyto tutaj funkcji viewDidAppear(), ponieważ instancja RestaurantListViewController będzie wyświetlać inną listę restauracji za każdym razem, gdy jej widok pojawi się na ekranie, w zależności od wyborów dokonanych przez użytkownika. W tej chwili kod po prostu wypisuje wybraną lokalizację i kuchnię do obszaru Debug, dzięki czemu można zobaczyć, czy te wartości są przekazywane poprawnie.

3. Kliknij plik ExploreViewController w nawigatorze projektu. Zadeklaruj i zdefiniuj metodę showRestaurantList(segue:) po metodzie showLocationList(segue:) w celu ustawienia właściwości wybraneCuisine i wybraneCity instancji RestaurantListViewController:

```
func showRestaurantList(segue: UIStoryboardSegue) {

if let viewController = segue.destination as?
```

```
RestaurantListViewController, let city =

selectedCity, let index =

collectionView.indexPathsForSelectedItems?.first?

.row {

viewController.selectedCuisine =

manager.exploreItem(at: index).name

viewController.selectedCity = city

}

}
```

Metodę tę wywołasz przed przejściem ekranu Eksploracja do ekranu Lista restauracji. Instrukcja if-let sprawdza, czy docelowym kontrolerem widoku jest instancja RestaurantListViewController, ustawia miasto na wartość wybraneCity instancji ExploreViewController, jeśli tak jest, i pobiera indeks komórki widoku kolekcji, którą kliknął użytkownik. Jeśli instrukcja się powiedzie, właściwość wybraneCuisine instancji RestaurantListViewController zostanie ustawiona na nazwę instancji ExploreItem znajdującej się pod tym indeksem w tablicy items. W kolejnym wierszu właściwości wybraneCity instancji RestaurantListViewController zostanie przypisana wartość przechowywana w mieście. Aby ta metoda zadziałała, przed przejściem do ekranu Szczegóły restauracji należy najpierw ustawić właściwość wybraneCity instancji ExploreViewController. Poinformujesz użytkownika o konieczności ustawienia miasta przed wyborem kuchni. Wykonaj następujące kroki:

1. Kliknij plik ExploreViewController w nawigatorze projektu. Dodaj następującą metodę przed unwindLocationCancel(), aby wyświetlić alert:

```
func showLocationRequiredAlert() {

let alertController = UIAlertController(title:

"Location Needed", message: "Please select a

location.", preferredStyle: .alert)

let okAction = UIAlertAction(title: "OK", style:

.default, handler: nil)

alertController.addAction(okAction)

present(alertController, animated: true,

completion: nil)

}
```

Metoda showLocationRequiredAlert() tworzy instancję UIAlertController z tytułem ustawionym na „Wymagana lokalizacja” i komunikatem „Wybierz lokalizację”. Następnie do instancji UIAlertController dodawana jest instancja UIAlertAction z przyciskiem OK. Na koniec alert jest prezentowany użytkownikowi i naciśnięcie przycisku OK powoduje jego odrzucenie.

2. Dodaj następujący kod po funkcji viewDidLoad(), aby wyświetlić ten alert, jeśli nie wybrano lokalizacji:

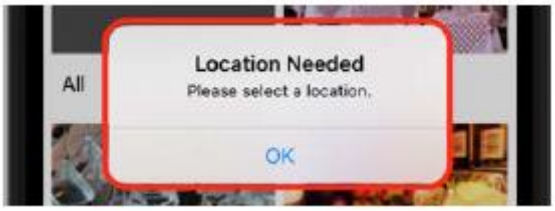
```
override func shouldPerformSegue(withIdentifier

identifier: String, sender: Any?) -> Bool {

if identifier == Segue.restaurantList.rawValue,

selectedCity == nil {

showLocationRequiredAlert()

return false

}

return true

}
```

Metoda powinnaPerformSegue(withIdentifier:sender:) służy do sprawdzenia, czy ekran Eksploruj powinien przejść do ekranu Lista restauracji. Najpierw sprawdzasz, czy identyfikator przejścia między tymi dwoma ekranami jest zgodny z restauracjąList i czy ustawiono wybrane miasto; jeśli nie, wywoływana jest metoda showLocationRequiredAlert() i ShouldP erformSegue(withIdentifier:sender:) zwraca wartość false. W przeciwnym razie metoda ShouldPerformSegue(withIdentifier:sender:) zwraca wartość true i pojawia się ekran Lista restauracji. Teraz, gdy masz już metody showLocationList(segue:) i showRestaurantList(segue:) dodasz kod implementujący metodę przygotowania(for:sender:) po viewDidLoad() i przed powinnoPerformSegue(withIdentifier:). Ta metoda wywołuje showLocationList(segue:) lub showRestaurantList(segue:) w zależności od sekwencji, która zostanie wykonana:

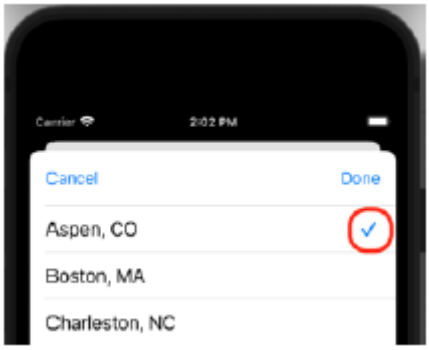
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

switch segue.identifier! {

case Segue.locationList.rawValue:

showLocationList(segue: segue)

case Segue.restaurantList.rawValue:

showRestaurantList(segue: segue)

default:

print("Segue not added")

}

}
```

Po dotknięciu przycisku LOKALIZACJA identyfikatorem przejścia jest lokalizacjaList, zatem przed przejściem do ekranu Lokalizacje wykonywana jest metoda showLocationList(segue:), co powoduje zaznaczenie wybranego miasta w widoku tabeli. Po dotknięciu komórki na ekranie Eksploruj kolejnym identyfikatorem jest restauracjaList, zatem przed przejściem do ekranu Lista restauracji wykonywana jest metoda showRestaurantListing(segue:). Spowoduje to ustawienie właściwości wybranego typu i

wybranego miasta w instancji RestaurantListViewController, które zostaną wydrukowane w obszarze debugowania. Zbuduj i uruchom swój projekt. Jeśli spróbujesz wybrać kuchnię, zobaczysz ten alert informujący, że musisz wybrać lokalizację:



Jeśli wybierzesz lokalizację, dotknij Gotowe, a następnie ponownie dotknij przycisku LOKALIZACJA. Wybrana wcześniej lokalizacja powinna nadal być wybrana:



Jeśli wybierzesz kuchnię, zobaczysz ekran Lista restauracji:

Wybrana lokalizacja i kuchnia pojawią się w obszarze debugowania:

Teraz, gdy instancja RestaurantListViewController ma lokalizację, możesz pobrać dane restauracji dla tej lokalizacji z instancji RestaurantDataManager. Kliknij plik RestaurantListViewController w nawigatorze projektu i zaktualizuj funkcję viewDidAppear() w następujący sposób. Spowoduje to wydrukowanie listy restauracji z wybranej lokalizacji serwujących wybraną kuchnię:

```
override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

guard let city = selectedCity?.city, let cuisine =

selectedCuisine else {

return

}

let manager = RestaurantDataManager()

manager.fetch(location: city, selectedCuisine: cuisine) {

restaurantItems in if !restaurantItems.isEmpty {

for restaurantItem in restaurantItems {

if let restaurantName = restaurantItem.name {

print(restaurantName)

}

}

} else {

print("No data")

}

}

}
```

Instrukcja strażnika sprawdza, czy wartości do miasta i kuchni zostały pomyślnie przypisane, i powraca, jeśli tak się nie stało. Następnie tworzona jest instancja RestaurantDataManager i przydzielana menedżerowi. Metoda fetch(location:selectedCuisine:completion:) zwraca tablicę instancji RestaurantItem dla wybranego miasta i kuchni, a pętla for wyświetla nazwy restauracji w obszarze debugowania. Jeśli nie ma restauracji spełniających kryteria, w obszarze Debugowanie nie zostaną wydrukowane żadne dane. Zbuduj i uruchom swój projekt, wybierz miasto, wybierz kuchnię i zanotuj wyniki w obszarze Debugowanie. Wyniki możesz także zobaczyć w nawigatorze raportów. Kliknij przycisk nawigatora raportów i wybierz pierwszy wpis w konsoli, jak pokazano:

W obszarze Edytora zobaczysz listę restauracji lub Brak danych. Zatem w tym momencie instancja RestaurantListViewController pomyślnie pobiera dane potrzebne do wyświetlenia listy restauracji. Teraz, gdy masz już te dane, musisz skonfigurować widok kolekcji, aby wyświetlał je użytkownikowi. Aby to zrobić, musisz utworzyć kontroler widoku dla komórek widoku kolekcji i skonfigurować instancję RestaurantListViewController w celu ich zapełnienia. Zrobisz to w następnej sekcji.

Tworzenie kontrolera widoku dla komórek na ekranie Lista restauracji

Obecnie ekran Lista restauracji nie ma klasy umożliwiającej zarządzanie komórkami widoku kolekcji w widoku kolekcji. W tym celu utworzysz klasę RestaurantCell. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Restauracje i wybierz Nowa grupa. Nazwij go Widok.

2. Kliknij prawym przyciskiem myszy folder Widok i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

4. Skonfiguruj plik w następujący sposób:

Klasa: RestauracjaCell

Podklasa: UICollectionViewCell

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

5. Kliknij Utwórz. Plik RestaurantCell pojawi się w nawigatorze projektu. Zawiera implementację klasy RestaurantCell:

```
import UIKit

class RestaurantCell: UICollectionViewCell {

}
```

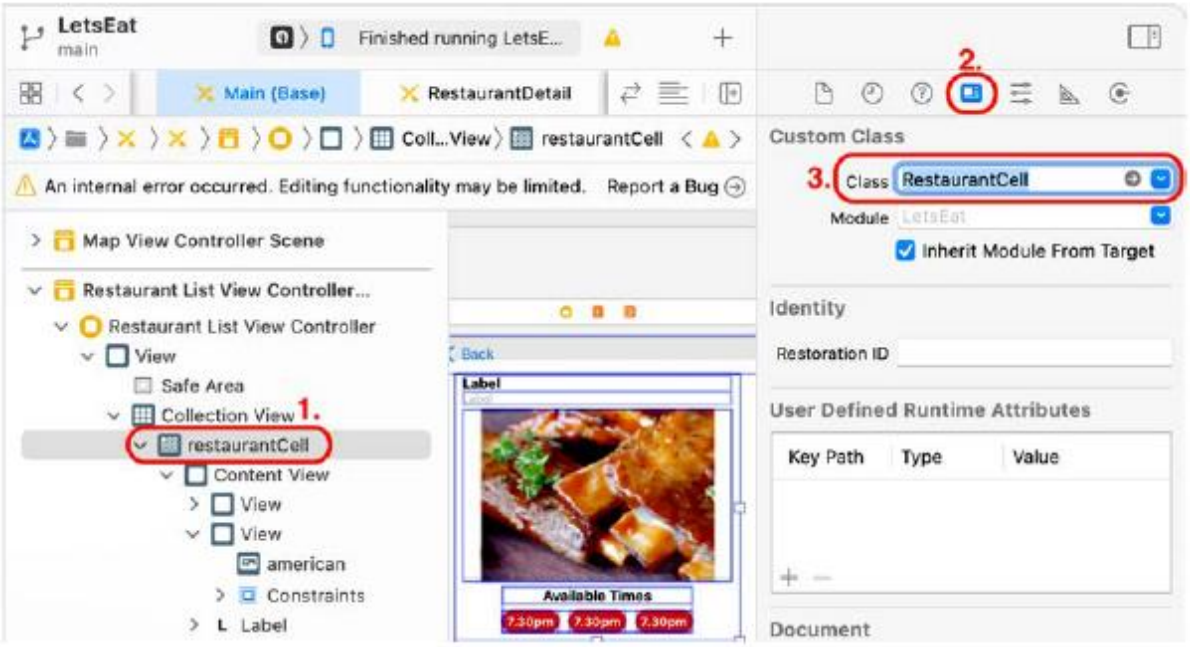
Utwórzmy teraz punkty sprzedaży dla komórki widoku kolekcji na ekranie Lista restauracji, aby klasa RestaurantCell mogła zarządzać ich zawartością. Zrobisz to w następnej sekcji.

Podłączenie gniazd dla klasy RestaurantCell

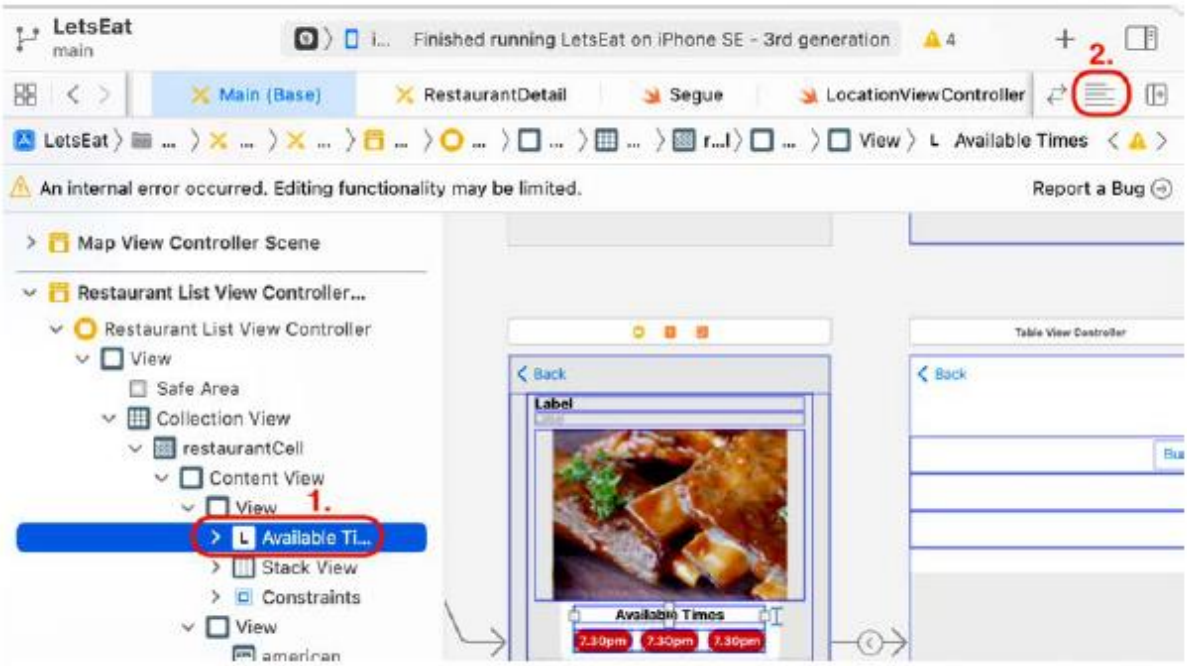
Po utworzeniu klasy RestaurantCell musisz utworzyć w niej punkty sprzedaży i połączyć je z elementami interfejsu użytkownika w komórkach widoku kolekcji na ekranie Lista restauracji. To pozwoli

Instancje RestaurantCell do zarządzania tym, co jest wyświetlane w komórce widoku kolekcji. Wykonaj następujące kroki:

1. Kliknij plik głównego scenorysu. Kliknij restauracjęKomórka w scenie kontrolera widoku listy restauracji. W Inspektorze tożsamości w obszarze Klasa niestandardowa ustaw klasę na RestaurantCell:

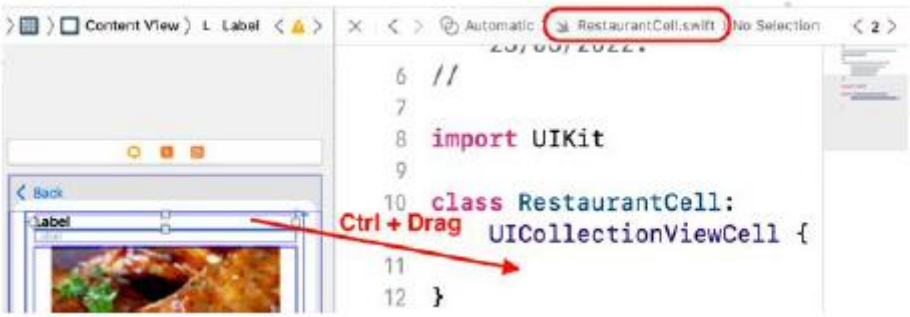


2. W scenie kontrolera widoku listy restauracji kliknij etykietę z tekstem Dostępne terminy w konspekcie dokumentu. Ma to na celu zapewnienie możliwości wybrania prawidłowego pliku (RestaurantCell) do wyświetlenia w edytorze asystenta po kliknięciu przycisku Dostosuj opcje edytora:

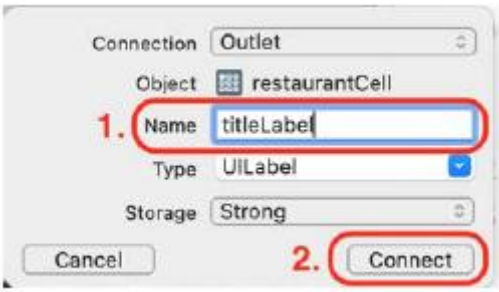


3. Kliknij przycisk Dostosuj opcje edytora i wybierz z menu opcję Asystent.

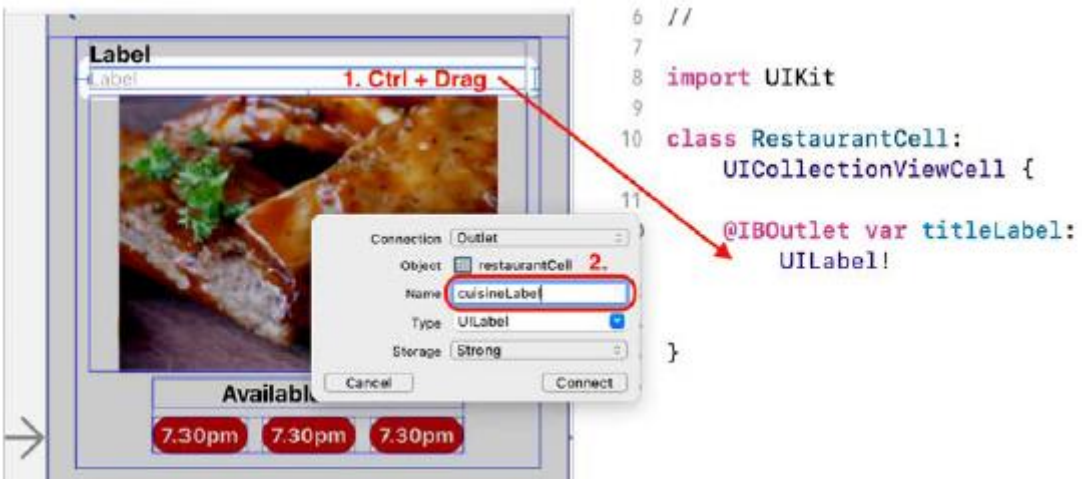
4. Pojawi się asystent redaktora. Pasek ścieżki u góry powinien pokazywać opcję Automatyczny > RestaurantCell. szybki. Jeśli nie, wybierz go z wyskakującego menu. Ctrl + Przeciągnij etykietę tytułową do przestrzeni pomiędzy nawiasami klamrowymi w pliku RestaurantCell:



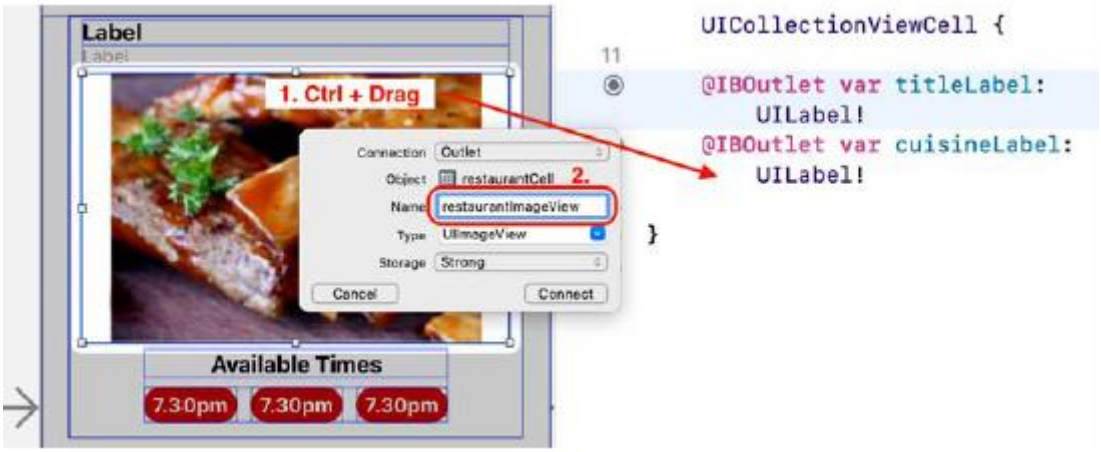
5. W wyświetlonym oknie wpisz tytułLabel w polu Nazwa i kliknij Połącz.



6. Ctrl + Przeciągnij etykietę napisów tuż za właśnie utworzoną właściwością titleLabel. W wyświetlonym oknie wpisz cuisineLabel w polu Nazwa i kliknij Połącz:



7. Ctrl + Przeciągnij z amerykańskiego widoku obrazu tuż za innymi utworzonymi właściwościami. W wyświetlonym oknie wpisz restauracjaImageView w polu Nazwa i kliknij Połącz:



8. Kliknij przycisk x, aby zamknąć asystenta edytora.

Punkty sprzedaży klasy RestaurantCell są teraz połączone z elementami interfejsu użytkownika w komórce widoku kolekcji. Później skonfigurujesz instancję RestaurantListViewController, aby zapełniła ten widok kolekcji, ale zanim to zrobisz, istnieje możliwość rozważenia. Dokonane przez użytkownika wybory dotyczące lokalizacji i kuchni mogą nie dać żadnych wyników, dlatego zaimplementujesz ekran informujący użytkownika, gdy nie ma danych do wyświetlenia. Zrobisz to w następnej sekcji.

Wyświetlanie niestandardowego widoku UIView w celu wskazania braku dostępnych danych

Wybór lokalizacji i kuchni dokonany przez użytkownika na ekranie Eksploruj może nie zwrócić żadnych wyników. W takim przypadku na ekranie Lista restauracji zostanie wyświetlony niestandardowy widok informujący o tym fakcie.

Aby to zrobić, utworzysz podklasę UIView i towarzyszący jej plik XIB. XIB oznacza Xcode Interface Builder, a pliki XIB zostały użyte do stworzenia interfejsu użytkownika przed wdrożeniem scenorysów. Utwórzmy teraz oba pliki, wykonując następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Różne i wybierz Nowa grupa. Nazwij to Brak danych.

2. Kliknij prawym przyciskiem myszy folder Brak danych i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

4. Skonfiguruj plik w następujący sposób:

Klasa: NoDataView

Podklasa: UIView

Utwórz także XIB: Wyszarzone

Język: Swift

Kliknij Następny.

5. Kliknij Utwórz. W nawigatorze projektu pojawi się plik NoDataView.

6. Kliknij prawym przyciskiem myszy folder Brak danych i utwórz nowy plik.

7. iOS powinien być już wybrany. Wybierz opcję Widok, a następnie kliknij przycisk Dalej.

8. Nazwij ten plik NoDataView. Kliknij Utwórz. W nawigatorze projektu pojawi się plik NoDataView XIB.

9. Kliknij plik NoDataView w nawigatorze projektu i zadeklaruj oraz zdefiniuj klasę NoDataView w następujący sposób:

```
class NoDataView: UIView {

var view: UIView!

@IBOutlet var titleLabel: UILabel!

@IBOutlet var descLabel: UILabel!

override init(frame: CGRect) {

super.init(frame: frame)

setupView()

}

required init?(coder: NSCoder) {

super.init(coder: coder)

setupView()

}

func loadViewFromNib() -> UIView {

let nib = UINib(nibName: "NoDataView", bundle:

Bundle.main)

let view = nib.instantiate(withOwner: self,

options: nil) [0] as! UIView
```

```
return view
}
func setupView() {
view = loadViewFromNib()
view.frame = bounds
view.autoresizingMask = [.flexibleWidth,
.flexibleHeight]
addSubview(view)
}
func set(title: String, desc: String) {
titleLabel.text = title
descLabel.text = desc
}
}
```

Ta klasa jest podklasą klasy UIView i będzie zarządzać widokiem w pliku NoDataView XIB.

Rozbijmy to:

```
var view: UIView!
@IBOutlet var titleLabel: UILabel!
@IBOutlet var descLabel: UILabel!
```

view zostanie przypisany widok z pliku NoDataView XIB podczas inicjalizacji.

titleLabel i descLabel zostaną przypisane do dwóch instancji UILabel, które zostaną umieszczone w pliku w NoDataView XIB podczas tworzenia interfejsu użytkownika w następnej sekcji.

```
override init(frame: CGRect) {
super.init(frame:frame)
setupView()
}
required init?(coder: NSCoder) {
super.init(coder: coder)
setupView()
}
```

Klasa NoDataView jest podklasą UIView. Obiekt UIView ma dwie metody init: pierwsza obsługuje programowe tworzenie widoków, a druga obsługuje ładowanie plików XIB z pakietu aplikacji zapisane na urządzeniu. W tym przypadku obie metody wywołają setupView().

```
func loadViewFromNib() -> UIView {

let nib = UINib(nibName: "NoDataView", bundle:

Bundle.main)

let view = nib.instantiate(withOwner: self,

options: nil) [0] as! UIView

return view

}
```

Ta metoda wyszukuje i ładuje plik NoDataView XIB z pakietu aplikacji i zwraca instancję UIView przechowywaną w nim.

```
func setupView() {

view = loadViewFromNib()

view.frame = bounds

view.autoresizingMask = [.flexibleWidth, .flexibleHeight]

addSubview(view)

}
```

Ta metoda wywołuje funkcję LoadViewFromNib(), konfiguruje widok tak, aby miał ten sam rozmiar co ekran urządzenia, zapewnia elastyczność szerokości i wysokości widoku w celu dostosowania do zmian rozmiaru i orientacji oraz dodaje go do hierarchii widoków urządzenia, aby był widoczny na ekranie.

```
func set(title: String, desc: String) {

titleLabel.text = title

descLabel.text = desc

}
```

Ta metoda ustawia tekst właściwości titleLabel i descLabel.

Teraz skonfigurujmy plik NoDataView XIB. Możesz zajrzeć do Rozdziału 13, Modyfikowanie i konfigurowanie komórek, który bardziej szczegółowo opisuje korzystanie z Inspektora rozmiaru i menu wiązań Autoukładu.

Wykonaj następujące kroki:

1. Kliknij plik NoDataView XIB w nawigatorze projektu.

2. Wybierz Właściciela pliku w konspekcie dokumentu. W Inspektorze tożsamości, w obszarze Klasa niestandardowa ustaw opcję Klasa na NoDataView i naciśnij klawisz Return.

3. Kliknij przycisk Biblioteka, aby wyświetlić bibliotekę. Wpisz etykietę w polu filtra. W wynikach pojawi się obiekt Etykieta.

4. Przeciągnij dwa obiekty Label do widoku, tak aby jeden obiekt Label znajdował się nad drugim.

5. Wybierz górną etykietę reprezentującą tytuł. W Inspektorze atrybutów zaktualizuj następujące wartości:

Tekst: dodaj TITLE GOES Here do pola tekstowego w ustawieniu Tekst

Kolor: domyślny (kolor etykiety)

Wyrównanie: środek

Czcionka: Systemowa pogrubiona 26.0

6. Po wybraniu tej samej etykiety zaktualizuj następujące wartości w Inspektorze rozmiaru:

Szerokość: 335

Wzrost: 36

7. Wybierz dolną etykietę. Będzie to reprezentować opis. W Inspektorze atrybutów zaktualizuj następujące wartości:

Tekst: Dodaj opis trafia tutaj do pola tekstowego w ustawieniu Tekst

Kolor: domyślny (kolor etykiety)

Wyrównanie: środek

Czcionka: System Thin 17.0

8. W Inspektorze rozmiaru zaktualizuj następujące wartości:

Szerokość: 335

Wzrost: 21

9. Zaznacz obie etykiety, klikając pierwszą etykietę i przytrzymując klawisz Shift podczas wybierania drugiej etykiety.

10. Mając zaznaczone obie etykiety, kliknij menu Edytor i wybierz Osadź w | Widok stosu.

11. Wybierz Widok stosu w konspekcie dokumentu i kliknij przycisk Inspektora atrybutów. W obszarze Widok stosu ustaw następujące wartości:

Oś: pionowa

Wyrównanie: środek

Rozstaw: 8

12. Po wybraniu widoku stosu kliknij przycisk Dodaj nowe ograniczenia. Ustaw następujące wartości:

Po lewej: 10

Po prawej: 10

Kliknij przycisk Dodaj 2 ograniczenia.

13. Mając nadal wybrany widok stosu, kliknij przycisk Wyrównaj. Ustaw następujące wartości:

Poziomo w pojemniku (zaznaczone)

Pionowo w kontenerze (zaznaczone)

Kliknij przycisk Dodaj 2 ograniczenia.

14. Wybierz Właściciela pliku w konspekcie dokumentu.

15. Otwórz Inspektora połączeń i połącz titleLabel z etykietą z napisem TITLE GOES Here.

16. Połącz descLabel z drugą etykietą.

Kiedy skończysz, powinieneś zobaczyć następujące informacje:

Zakończono konfigurowanie pliku NoDataView.xib. Teraz złóżmy to wszystko w jedną całość tak, aby na ekranie Lista Restauracji wyświetliła się lista restauracji na podstawie wybranej lokalizacji i kuchni lub wyświetliła NoDataView, jeśli w danej lokalizacji nie ma restauracji oferujących wybraną kuchnię. Zrobisz to w następnej sekcji.

Wyświetlanie listy restauracji na ekranie Lista restauracji

Masz teraz wszystko, czego potrzebujesz, aby wyświetlić listę restauracji na podstawie wybranej lokalizacji i kuchni na ekranie Lista restauracji. Nadszedł więc czas, aby to wszystko połączyć w jedną całość. Wykonaj następujące kroki:

1. Kliknij plik RestaurantListViewController w nawigatorze projektu. Przed właściwością wybranej restauracji dodaj następujące polecenie, aby utworzyć instancję RestaurantDataManager i przypisać ją do właściwości menedżera:

```
private let manager = RestaurantDataManager()

var selectedRestaurant: RestaurantItem?
```

2. Dodaj następującą metodę w rozszerzeniu prywatnym, aby wypełnić tablicę elementów właściwości menedżera i ustawić widok tła dla ekranu Lista restauracji:

```
func createData() {

guard let city = selectedCity?.city, let

cuisine = selectedCuisine else {

return

}
```

```
manager.fetch(location: city,
selectedCuisine: cuisine) {restaurantItems in
if !restaurantItems.isEmpty {
collectionView.backgroundView = nil
} else {
let view = NoDataView(frame: CGRect(x: 0,
y: 0, width: collectionView.frame.width,
height: collectionView.frame.height))
view.set(title: "Restaurants", desc:
"No restaurants found.")
collectionView.backgroundView = view
}
collectionView.reloadData()
}
}
```

Rozbijmy to:

```
guard let city = selectedCity?.city, let cuisine =
selectedCuisine else {
return
}
```

Sprawdza to, czy właściwości wybraneCity i wybraneCuisine są ustawione; jeśli tak, przypisz wybrane miasto do miasta i wybraną kuchnię do kuchni. W przeciwnym razie zakończ metodę.

```
manager.fetch(location: city, selectedCuisine: cuisine)
```

Wywołuje to metodę fetch(location:selectedCuisine:completion:) instancji RestaurantDataManager, która ładuje odpowiednie instancje RestaurantItem do tablicy RestaurantItems.

```
{{ restaurantItems in
if !restaurantItems.isEmpty {
collectionView.backgroundView = nil
} else {
let view = NoDataView(frame: CGRect(x: 0, y: 0,
width: collectionView.frame.width, height:
collectionView.frame.height))
```

```
view.set(title: "Restaurants", desc:

"No restaurants found.")

collectionView.backgroundView = view

}
```

Jeśli tablica RestaurantItems instancji RestaurantDataManager nie jest pusta, ustaw wartość tłaView instancji CollectionView na nil. W przeciwnym razie utwórz instancję NoDataView, ustaw jej tytuł i opis oraz ustaw ją jako właściwość backView instancji kolekcjiView.

```
collectionView.reloadData()
```

To mówi kolekcjiView, aby odświeżyła swój widok.

3. Zaktualizuj CollectionView(_:cellForItemAt:) w następujący sposób, aby ustawić właściwości instancji RestaurantCell:

```
func collectionView(_ collectionView:

UICollectionView, cellForItemAt indexPath: IndexPath)

-> UICollectionViewCell {

let cell = collectionView.dequeueReusableCell(

withReuseIdentifier: "restaurantCell", for:

indexPath) as! RestaurantCell

let restaurantItem = manager.restaurantItem(at:

indexPath.row)

cell.titleLabel.text = restaurantItem.name

if let cuisine = restaurantItem.subtitle {

cell.cuisineLabel.text = cuisine

}

if let imageURL = restaurantItem.imageURL {

Task {

guard let url = URL(string: imageURL)

else {

return

}

let (imageData, response) = try await

URLSession.shared.data(from: url)

guard let httpResponse = response as?
```

HTTPURLResponse, httpResponse.statusCode

== 200 else {

return

}

guard let cellImage = UIImage(data:

imageData) else {

return

}

cell.restaurantImageView.image = cellImage

}

}

return cell

}

Rozbijmy to:

let restaurantItem = manager.restaurantItem(at: indexPath.row)

Spowoduje to pobranie instancji RestaurantItem z tablicy RestaurantItems instancji RestaurantDataManager odpowiadającej pozycji instancji RestaurantCell.

cell.titleLabel.text = restaurantItem.name

Ustawia to tekst titleLabel instancji RestaurantCell na wartość nazwy instancji RestaurantItem.

if let cuisine = restaurantItem.subtitle {

cell.cuisineLabel.text = cuisine

}

Ustawia to tekst instancji KitchenLabel instancji RestaurantCell na wartość podtytułu instancji RestaurantItem.

if let imageURL = restaurantItem.imageURL {

Spowoduje to pobranie adresu URL określonego w imageURL instancji RestaurantItem i przypisanie go do imageURL.

Task {

This creates a unit of asynchronous work.

guard let url = URL(string: imageURL)

else {

return

}

Ta instrukcja Guard tworzy adres URL na podstawie właściwości imageURL instancji RestaurantItem i przypisuje go do adresu URL, a jeśli nie jest w stanie tego zrobić, zwraca adres URL.

let (imageData, response) = try await

URLSession.shared.data(from: url)

Spowoduje to asynchroniczne pobranie danych z adresu URL zapisanego w adresie URL i przypisanie ich do imageData. Odpowiedź z serwera jest przypisana do odpowiedzi.

guard let httpResponse = response as? HTTPURLResponse,

httpResponse.statusCode == 200 else {

return

}

Ta instrukcja strażnika sprawdza, czy kod odpowiedzi serwera wynosi 200 (co oznacza, że pobieranie powiodło się) i zwraca, jeśli tak nie jest.

guard let cellImage = UIImage(data: imageData) else {

return

}

Ta instrukcja Guard tworzy instancję UIImage z danych przechowywanych w imageData i przypisuje ją do cellImage, a następnie zwraca, jeśli nie jest w stanie tego zrobić.

cell.restaurantImageView.image = cellImage

Spowoduje to przypisanie UIImage przechowywanego w cellImage do właściwości restauracyjnej instancji RestaurantCell, która będzie wyświetlona w widoku kolekcji na ekranie Lista restauracji. W przeciwnym razie zostanie wyświetlony domyślny obraz ustawiony dla właściwości restauracjaImageView.

return cell

}

Zwraca to komórkę widoku kolekcji.

4. Zaktualizuj CollectionView(_:numberOfItemsInSection:) w następujący sposób, aby uzyskać liczbę widoków kolekcji do wyświetlenia od menedżera:

func collectionView(_ collectionView: UICollectionView,

numberOfItemsInSection section: Int) -> Int {

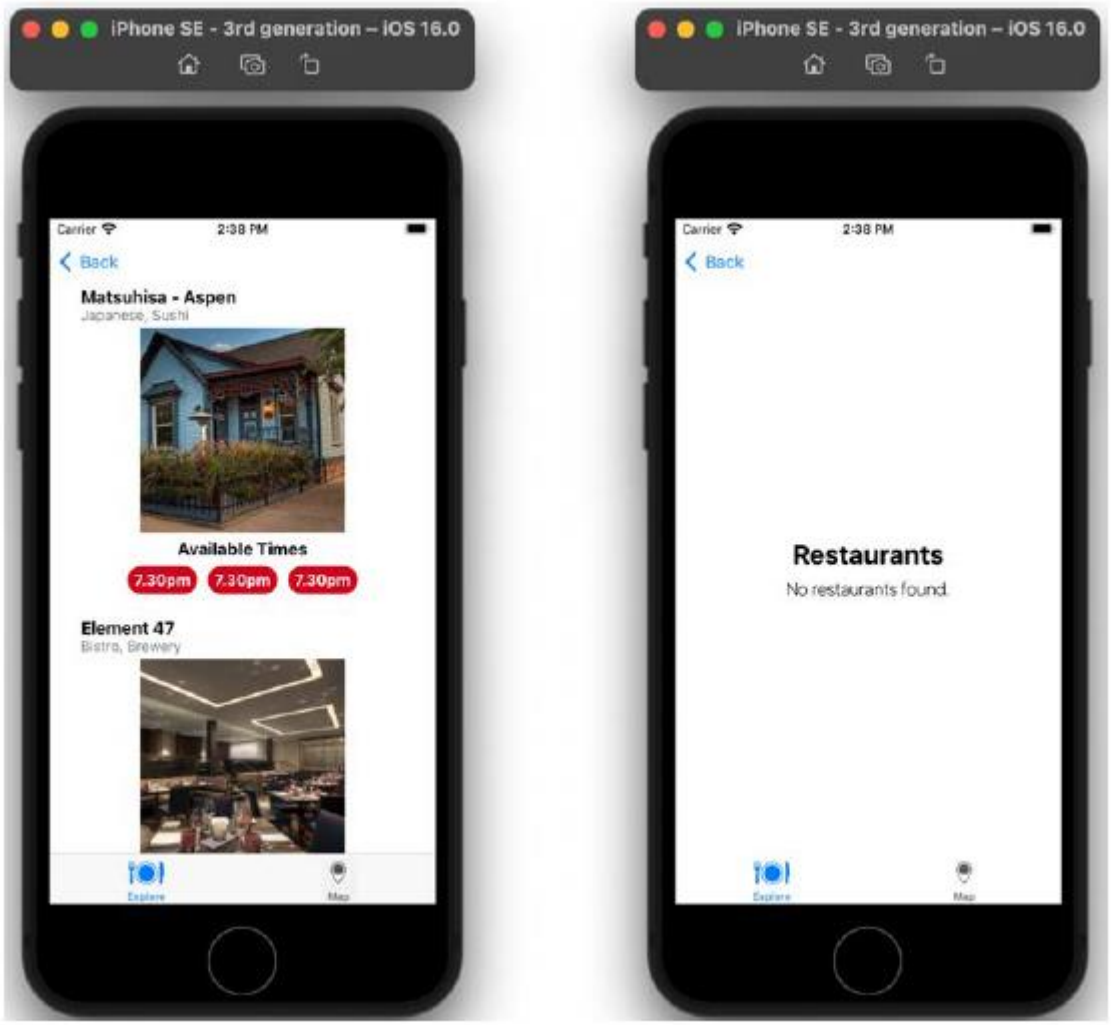
manager.numberOfRestaurantItems()

}

5. Zaktualizuj funkcję viewDidAppear() w następujący sposób, aby wywołać funkcję createData(), gdy na ekranie pojawi się widok kolekcji:

```
override func viewDidAppear(animated: Bool) {

super.viewDidAppear(animated)

createData()

}
```

Kompiluj i uruchamiaj swoją aplikację. Ustaw lokalizację i kliknij kuchnię. Jeśli istnieją restauracje spełniające wybrane kryteria, zobaczysz je na ekranie Lista restauracji. W przeciwnym razie zostanie wyświetlona instancja NoDataView.



Zanim skończysz z klasą RestaurantListViewController, jest jeszcze jedna rzecz. Byłoby miło, gdyby wybrane miasto było pokazane na ekranie Lista Restauracji. Dodajmy kod, aby wyświetlić go u góry paska nawigacyjnego ekranu Listy Restauracji z dużymi tytułami. Wykonaj następujące kroki:

1. W pliku RestaurantListViewController dodaj następującą metodę do rozszerzenia prywatnego po funkcji createData(), aby wyświetlić wybrane miasto na pasku nawigacyjnym:

```
func setupTitle() {

navigationController?.setNavigationBarHidden(

false, animated: false)
```

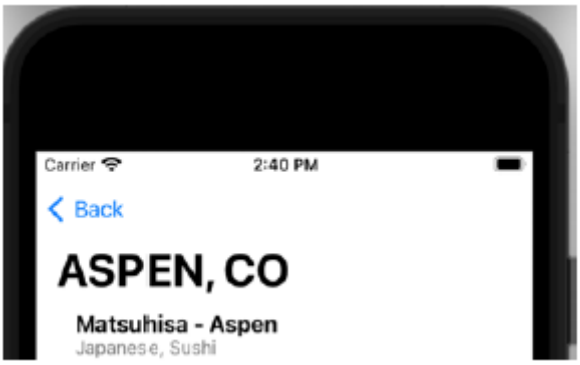
```
title = selectedCity?.cityAndState.uppercased()

navigationController?.navigationBar.

prefersLargeTitles = true

}
```

Każda instancja UIViewController ma właściwość title i jeśli pasek nawigacyjny jest widoczny, tytuł również będzie widoczny. Ta metoda wyświetla pasek nawigacyjny i ustawia tytuł instancji RestaurantListViewController na ciąg znaków zawierający nazwy miast i stanów pisane wielkimi literami.

2. Wywołaj setupTitle() po createData() w metodzie viewDidAppear():

```
override func viewDidAppear(_ animated: Bool){

super.viewDidAppear(animated)

createData()

setupTitle()

}
```

Spowoduje to wywołanie metody setupTitle() po wyświetleniu ekranu Lista restauracji. Kompiluj i uruchamiaj swoją aplikację. Wybierz lokalizację i kuchnię. Miasto i stan powinny być widoczne wielkimi literami u góry ekranu z listą restauracji:



Zakończyłeś wdrażanie ekranu Lista restauracji i dotarłeś do końca tego rozdziału. Dobra robota!

Streszczenie

Osiągnąłeś wiele w tym rozdziale. Zacząłeś od poznania formatu JSON i utworzyłeś klasę RestaurantDataManager, klasę menedżera danych, która może ładować dane z plików JSON. Skonfigurowałeś klasę MapViewController tak, aby pobierała dane z instancji RestaurantDataManager w celu wyświetlenia listy restauracji na ekranie Mapa. Następnie skonfigurowano klasę LocationViewController do przechowywania lokalizacji wybranej przez użytkownika i przekazywania jej do instancji ExploreViewController po naciśnięciu przycisku Gotowe. Następnie skonfigurowałeś klasę ExploreViewController do przekazywania wybraną lokalizację i kuchnię do instancji RestaurantListViewController po wybraniu typu kuchni. Na koniec skonfigurowałeś klasę RestaurantListViewController, aby uzyskać listę restauracji z instancji RestaurantDataManager i wyświetlić ją na ekranie Lista restauracji, przefiltrowaną według wybranej kuchni. Utworzyłeś także

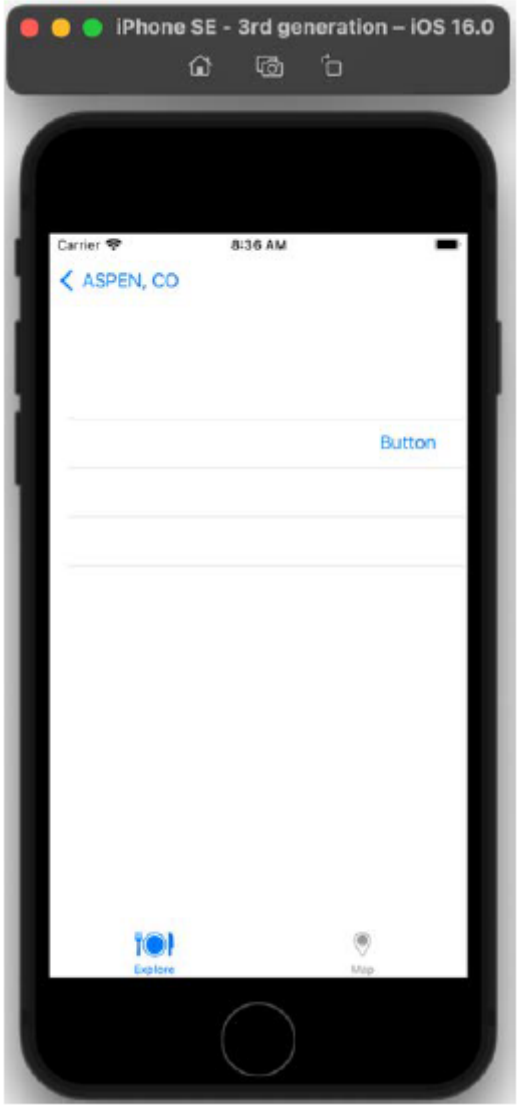
klasę i widok NoDataView, który wyświetla się, jeśli w danej lokalizacji nie ma restauracji oferujących wybraną kuchnię. Możesz teraz ładować i odczytywać dane z plików JSON oraz przekazywać je między różnymi kontrolerami widoku w aplikacji w celu wyświetlenia w widokach kolekcji i widokach map. Nauczyłeś się także, jak używać metod delegowania UITableViewController do obsługi interakcji użytkownika z widokami tabel. Przyda się to podczas tworzenia własnych aplikacji. W następnym rozdziale zaimplementujesz ekran Szczegóły restauracji, który wyświetla szczegółowe informacje o konkretnej restauracji za pomocą widoku tabeli zawierającej komórki statyczne.

Wyświetlanie danych w statycznym widoku tabeli

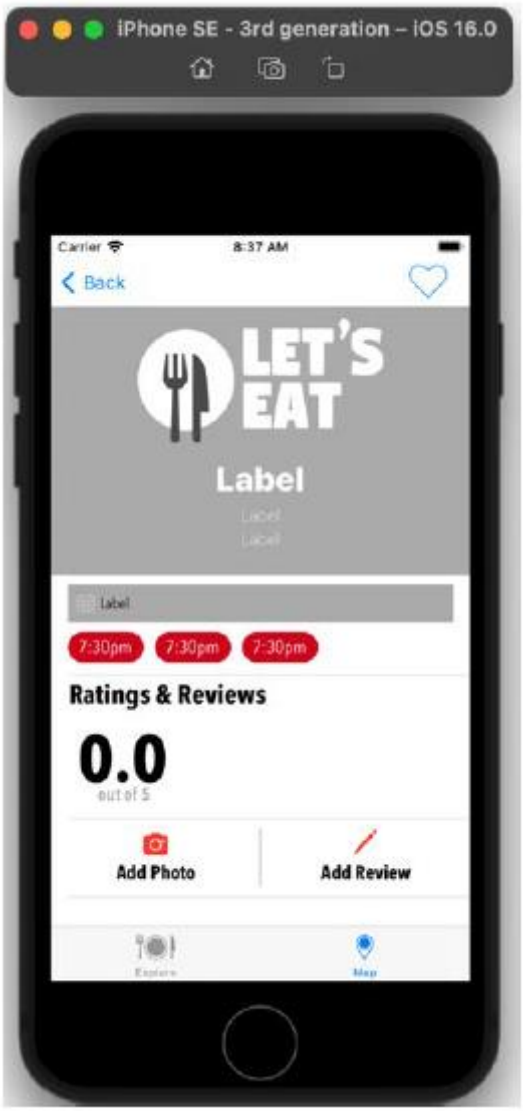
Przebyłeś długą drogę, a Twoja aplikacja zawiera dane na wszystkich ekranach z wyjątkiem ekranu „Szczegóły restauracji”. W tym rozdziale skonfigurujesz klasę RestaurantDetailViewController do zarządzania widokami na ekranie Szczegóły restauracji. Następnie dodasz metody do metody viewDidLoad(), aby wypełnić widok tabeli po wyświetleniu ekranu szczegółów restauracji. Na koniec przekażesz odpowiednią instancję RestaurantItem z instancji RestaurantListViewController i MapViewController do instancji RestaurantDetailViewController, która wyświetli dane z tej instancji RestaurantItem na ekranie Szczegóły restauracji. Pod koniec tego rozdziału dowiesz się, jak tworzyć widoki tabel z komórkami statycznymi, aby wyświetlać dane, i jak utworzyć niestandardowy obraz mapy. Dzięki temu będziesz mógł wdrożyć te funkcje we własnych aplikacjach.

Konfigurowanie punktów sprzedaży dla klasy RestaurantDetailViewController

Twoja aplikacja zawiera dane na wszystkich ekranach z wyjątkiem ekranu „Szczegóły restauracji”. Dostęp do tego ekranu można uzyskać, dotykając restauracji na ekranie Lista restauracji lub dotykając przycisku objaśnienia widoku adnotacji restauracji na ekranie Mapa. Jeśli utworzysz i uruchomisz aplikację, dotknięcie restauracji na ekranie Lista restauracji spowoduje wyświetlenie zastępczego ekranu szczegółów restauracji:



Dotknięcie przycisku w dymku widoku adnotacji restauracji na ekranie mapy powoduje wyświetlenie rzeczywistego ekranu szczegółów restauracji, ale nie zawiera on żadnych danych o restauracji:



Aby to naprawić, skonfigurujmy outlety dla klasy RestaurantDetailViewController. Kliknij plik RestaurantDetailViewController w nawigatorze projektu. Dodaj następujące punkty sprzedaży po deklaracji klasy i przed deklaracją właściwości wybranej restauracji:

```
// Pasek nawigacji

@IBOutlet var heartButton: UIBarButtonItem!

// Komórka pierwsza

@IBOutlet var nameLabel: UILabel!

@IBOutlet var cuisineLabel: UILabel!

@IBOutlet var headerAddressLabel: UILabel!

// Komórka druga

@IBOutlet var tableDetailsLabel: UILabel!
```

// Komórka trzecia

@IBOutlet var generalRatingLabel: UILabel!

// Komórka ósma

@IBOutlet var adresLabel: UILabel!

// Komórka dziewiąta

@IBOutlet var LocationMapImageView: UIImageView!

Punkty sprzedaży, które właśnie skonfigurowałeś, są następujące:

• heartButton to miejsce dla przycisku w kształcie serca na pasku nawigacyjnym. Służy do wskazania, czy lubisz konkretną restaurację. W tej książce nie będziesz wdrażał jego funkcjonalności, ale możesz później popracować nad tym samodzielnie.

• nameLabel to wylot etykiety, która wyświetla nazwę restauracji w pierwszej komórce.

• cuisineLabel to outlet etykiety, która w pierwszej komórce wyświetla kuchnie oferowane przez restaurację.

• headerAddressLabel to wylot etykiety, która wyświetla adres restauracji w pierwszej komórce.

• tableDetailsLabel to wylot etykiety wyświetlającej w drugiej komórce szczegółowe informacje o stole restauracji.

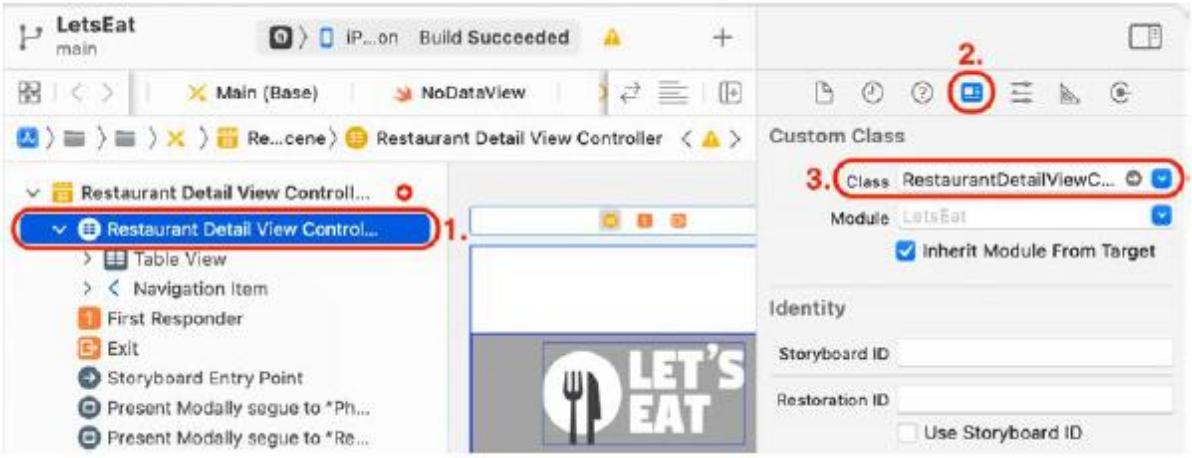
• totalRatingLabel to miejsce sprzedaży etykiety, która wyświetla ogólną ocenę restauracji w trzeciej komórce.

• adresLabel to wylot etykiety, która wyświetla adres restauracji w ósmej komórce.

• LocationMapImageView to widok obrazu, który wyświetla mapę lokalizacji restauracji w dziewiątej komórce. Metody generowania tej mapy napiszesz w dalszej części tej części.

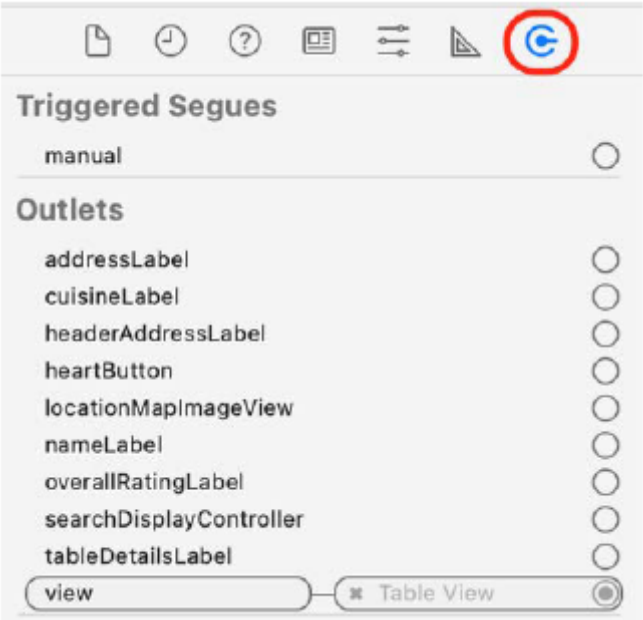
Po utworzeniu punktów sprzedaży połącz je z elementami interfejsu użytkownika w scenie kontrolera widoku szczegółów restauracji w pliku scenorysu RestaurantDetail. Wykonaj następujące kroki:

1. Rozwiń folder RestaurantDetail w nawigatorze projektu. Kliknij plik scenorysu RestaurantDetail. Następnie kliknij ikonę kontrolera widoku w scenie kontrolera widoku szczegółów restauracji. Następnie kliknij przycisk Inspektor tożsamości. W obszarze Klasa niestandardowa potwierdź, że klasa została ustawiona na klasę RestaurantDetailViewController:



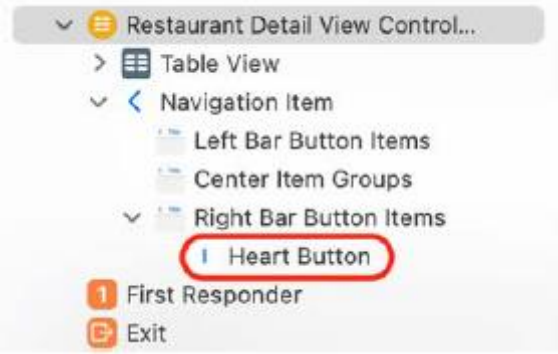
Należy pamiętać, że po ustawieniu klasy nazwa kontrolera widoku zmieni się na Kontroler widoku szczegółowego restauracji. W przeciwieństwie do widoku tabeli w scenie kontrolera widoku lokalizacji, widok tabeli w scenie kontrolera widoku szczegółów restauracji zawiera komórki statyczne, co oznacza, że liczba komórek nie jest generowana dynamicznie na podstawie danych z obiektu modelu. Jak widać na schemacie dokumentu, jest dziewięć komórek, a każda komórka została już skonfigurowana z odpowiednimi obiektami widoku. Kliknięcie każdej komórki widoku tabeli w konspekcie dokumentu spowoduje wyświetlenie tej komórki w obszarze Edytora.

2. Kliknij przycisk Inspektora połączeń. Wszystkie punkty sprzedaży, które dodałeś wcześniej, zobaczysz w klasie RestaurantDetailViewController:

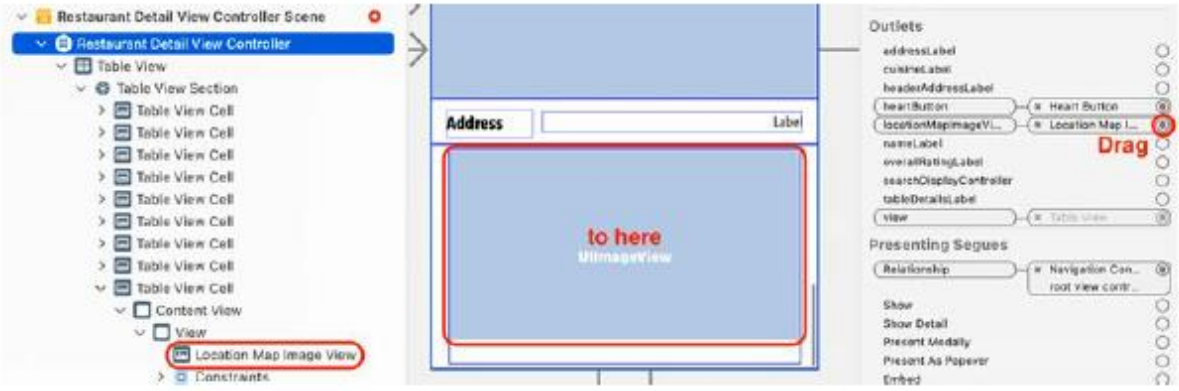


3. Kliknij i przeciągnij z outletu heartButton do serca na pasku nawigacyjnym:

4. Gniazdo heartButton jest teraz podłączone. Pamiętaj, że opis widoku zmieni się na Przycisk Serca w konspekcie dokumentu:



5. Kliknij ostatnią komórkę widoku tabeli w konspekcie dokumentu, aby wyświetlić dół widoku tabeli, a następnie kliknij opcję Kontroler widoku szczegółów restauracji (może być konieczne kilkakrotne kliknięcie). Powinieneś zobaczyć UIImageView. Kliknij i przeciągnij z wyjścia LocationMapImageView do widoku obrazu w ostatniej komórce, aby je połączyć. Należy pamiętać, że nazwa zmieni się z Widoku obrazu na Widok obrazu mapy lokalizacji w konspekcie dokumentu:



6. Kliknij ósmą komórkę widoku stołu w konspekcie dokumentu i kliknij Kontroler widoku szczegółów restauracji. Kliknij i przeciągnij z wyjścia adresuLabel do etykiety w ósmej komórce, aby je połączyć. Należy pamiętać, że nazwa zmieni się z Etykieta na Etykieta adresowa w konspekcie dokumentu:

7. Kliknij pierwszą komórkę widoku tabeli w konspekcie dokumentu i kliknij Kontroler widoku szczegółów restauracji. Kliknij i przeciągnij z gniazdka cuisineLabel do drugiej etykiety w pierwszej komórce, aby je połączyć. Należy pamiętać, że nazwa w konspekcie dokumentu zmieni się z Etykieta na Etykieta kuchni:

8. Kliknij i przeciągnij z wyjścia headerAddressLabel do trzeciej etykiety w pierwszej komórce, aby je połączyć. Pamiętaj, że nazwa w konspekcie dokumentu zmieni się z Etykieta na Etykieta adresu nagłówka:

9. Kliknij i przeciągnij z gniazda nameLabel do pierwszej etykiety w pierwszej komórce, aby je połączyć. Należy pamiętać, że nazwa w konspekcie dokumentu zmieni się z Etykieta na Etykieta z nazwą:

10. Kliknij trzecią komórkę widoku stołu w konspekcie dokumentu i kliknij Kontroler widoku szczegółów restauracji. Kliknij i przeciągnij z wyjścia totalRatingLabel do etykiety z dużym czarnym 0.0 w środku, aby je połączyć. Należy pamiętać, że nazwa w konspekcie dokumentu zmieni się z Etykieta na Etykieta oceny ogólnej:

11. Kliknij drugą komórkę widoku tabeli w konspekcie dokumentu i kliknij Kontroler widoku szczegółów restauracji. Kliknij i przeciągnij z wyjścia tableDetailsLabel do etykiety tuż nad trzema czerwonymi przyciskami w drugiej komórce, aby je połączyć. Należy pamiętać, że nazwa w konspekcie dokumentu zmieni się z Etykieta na Etykieta szczegółów tabeli:

Wszystkie punkty sprzedaży dla klasy RestaurantDetailViewController zostały już skonfigurowane. W następnej sekcji zmodyfikujesz klasę RestaurantDetailViewController, aby odbierać dane restauracji z instancji RestaurantListViewController i MapViewController i wyświetlać je na ekranie Szczegóły restauracji.

Wyświetlanie danych w statycznym widoku tabeli

Pomyślnie połączyłeś wszystkie punkty sprzedaży w klasie RestaurantDetailViewController z elementami interfejsu użytkownika na ekranie Szczegóły restauracji. Ponieważ jest to statyczny widok tabeli, nie będziesz stosować protokołu UITableViewDataSource do zapełniania gniazd. Zamiast tego napiszesz w tym celu niestandardowe metody. Wykonaj następujące kroki:

1. Kliknij plik RestaurantDetailViewController w nawigatorze projektu.

2. Dodaj kod importujący framework MapKit po istniejącej instrukcji importu:

zaimportuj MapKit

Jest to wymagane, ponieważ będziesz używać właściwości i metod środowiska MapKit do generowania obrazu mapy dla widoku obrazu w ostatniej komórce.

3. Dodaj prywatne rozszerzenie zawierające kod umożliwiający ustawienie etykiet na ekranie Szczegóły restauracji po ostatnim nawiasie klamrowym:

```
private extension RestaurantDetailViewController {

func setupLabels() {

guard let restaurant =

selectedRestaurant else {

return

}

title = restaurant.name

nameLabel.text = restaurant.name

cuisineLabel.text = restaurant.subtitle

headerAddressLabel.text = restaurant.address

tableDetailsLabel.text = "Table for 7, tonight

at 10:00 PM"

addressLabel.text = restaurant.address

}

}
```

Metoda setupLabels() jest całkiem prosta; pobiera wartości z instancji RestaurantItem i umieszcza je w punktach wyjściowych instancji RestaurantDetailViewController, z wyjątkiem tableDetailsLabel, do którego właśnie przypisano ciąg znaków.

4. W ostatniej komórce wyświetlisz obraz mapy. Aby to zrobić, wygenerujesz obraz z obszaru mapy i ustawisz lokalizację lokalizacjiMapImageLabel tak, aby wyświetlała ten obraz. Na tym obrazie będzie także wyświetlany ten sam obraz niestandardowej adnotacji, którego użyłeś na ekranie Mapa. Dodaj następującą metodę po setupLabels() i przed ostatnim nawiasem klamrowym:

```
func createMap() {

guard let annotation = selectedRestaurant, let long

= annotation.long, let lat = annotation.lat else {

return

}
```

```
let location = CLLocationCoordinate2D(latitude:

lat, longitude: long)

takeSnapshot(with: location)

}
```

Ta metoda tworzy instancję CLLocationCooperative2D przy użyciu właściwości lat i long właściwości wybranejRestaurant i przypisuje ją do lokalizacji. Następnie wywołuje metodę takeSnapshot(with:), przekazując lokalizację jako parametr.

5. Zobaczysz błąd, ponieważ funkcja takeSnapShot(with:) nie została jeszcze zaimplementowana, więc dodaj następujący kod po funkcji createMap(), aby ją zaimplementować:

```
func takeSnapshot(with location:

CLLocationCoordinate2D) {

let mapSnapshotOptions = MKMapSnapshotter.Options()

var loc = location

let polyline = MKPolyline(coordinates: &loc, count:

1 )

let region = MKCoordinateRegion(polyline.

boundingMapRect)

mapSnapshotOptions.region = region

mapSnapshotOptions.scale = UIScreen.main.scale

mapSnapshotOptions.size = CGSize(width: 340,

height: 208)

mapSnapshotOptions.showsBuildings = true

mapSnapshotOptions.pointOfInterestFilter =

.includingAll

let snapShotter = MKMapSnapshotter(options:

mapSnapshotOptions)

snapShotter.start() { snapshot, error in

guard let snapshot = snapshot else {

return

}

UIGraphicsBeginImageContextWithOptions(

mapSnapshotOptions.size, true, 0)
```

```
snapshot.image.draw(at: .zero)
let identifier = "custompin"
let annotation = MKPointAnnotation()
annotation.coordinate = location
let pinView = MKMarkerAnnotationView(
annotation: annotation,
reuseIdentifier: identifier)
pinView.image = UIImage(named: "customannotation")!
let pinImage = pinView.image
var point = snapshot.point(for: location)
let rect = self.locationMapImageView.bounds
if rect.contains(point) {
let pinCenterOffset = pinView.centerOffset
point.x -= pinView.bounds.size.width / 2
point.y -= pinView.bounds.size.height / 2
point.x += pinCenterOffset.x
point.y += pinCenterOffset.y
pinImage?.draw(at: point)
}
if let image =
UIGraphicsGetImageFromCurrentImageContext() {
UIGraphicsEndImageContext()
DispatchQueue.main.async {
self.locationMapImageView.image = image
}
}
}
}
```

Pełny opis tej metody wykracza poza zakres tej książki, ale poniżej znajduje się proste wyjaśnienie jej działania. Biorąc pod uwagę lokalizację, wykonuje migawkę mapy w tej lokalizacji, dodaje niestandardową adnotację użytą wcześniej na ekranie Mapa, konwertuje ją na obraz i przypisuje do gniazda LocationMapImageView w instancji RestaurantDetailViewController.

6. Napisałeś wszystkie metody wymagane dla klasy RestaurantDetailViewController do wyświetlenia żądanych szczegółów instancji RestaurantItem na ekranie Szczegóły restauracji. W rozszerzeniu prywatnym przed definicją metody setupLabels() dodaj metodę inicjalizacji(), która wywołuje metody setupLabels() i createMap():

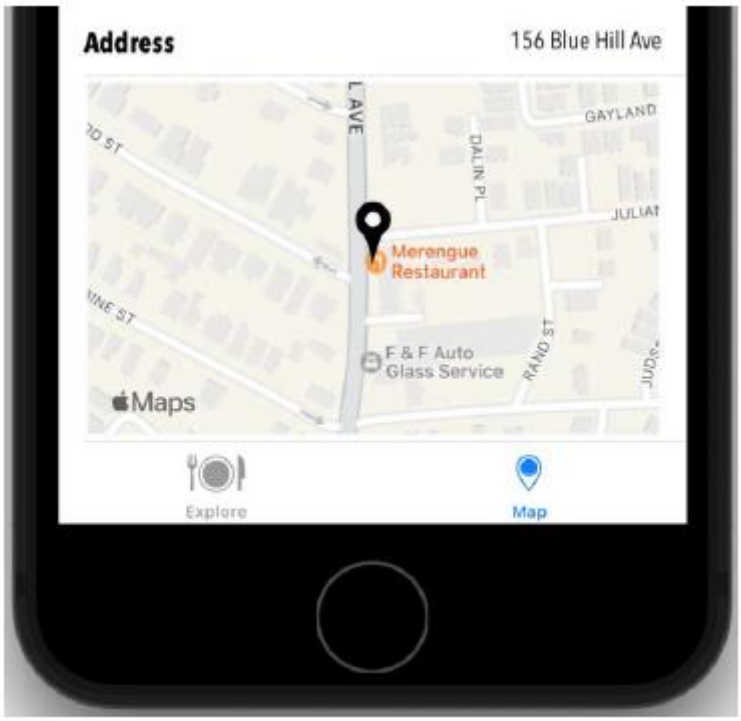
```
func initialize() {

setupLabels()

createMap()

}
```

7. Zmodyfikuj metodę viewDidLoad(), aby wywoływała metodę inicjalizacji(), gdy instancja RestaurantDetailViewController ładuje swój widok:

```
override func viewDidLoad() {

super.viewDidLoad()

initialize()

}
```

Przypomnij sobie, że skonfigurowałeś już klasę MapViewController do przekazywania instancji RestaurantItem do instancji RestaurantDetailViewController. Zbuduj i uruchom aplikację, a następnie przejdź do ekranu Mapa. Kliknij jedną z restauracji, aby wyświetlić dymek z objaśnieniem. Kliknij przycisk w dymku objaśnienia, a na ekranie Szczegóły restauracji powinny pojawić się szczegółowe informacje o restauracji:

Jeśli przewiniesz w dół, w ostatniej komórce zobaczysz obraz mapy:



Zakończyłeś modyfikowanie klasy RestaurantDetailViewController, ale nadal musisz przekazać wybraną instancję RestaurantItem z instancji RestaurantListViewController do instancji RestaurantDetailViewController. Zrobisz to w następnej sekcji.

Przekazywanie danych do instancji RestaurantDetailViewController

Dodałeś i połączyłeś wyjścia dla ekranu Szczegóły restauracji w klasie RestaurantDetailViewController. Dodałeś także kod do tej klasy, aby pobrać dane restauracji z instancji RestaurantItem i użyć ich do zapełnienia jej punktów sprzedaży. Ostatnią rzeczą, którą musisz zrobić, to przekazać wybraną instancję RestaurantItem z instancji RestaurantListViewController do instancji RestaurantDetailViewController. Wykonaj następujące kroki:

1. Kliknij plik RestaurantListViewController w nawigatorze projektu.

2. Dodaj następujący kod po funkcji viewDidAppear(): aby wywołać metodę showRestaurantDetail(segue:) jeśli identyfikatorem segue jest showDetail:

```
override func prepare(for segue: UIStoryboardSegue,

sender: Any?) {

if let identifier = segue.identifier {

switch identifier {

case Segue.showDetail.rawValue:

showRestaurantDetail(segue: segue)

default:

print("Segue not added")

}
```

}

}

Przypomnij sobie, że w scenorysie dodano przejście pomiędzy sceną kontrolera widoku listy restauracji a sceną kontrolera widoku szczegółów restauracji. Zanim instancja RestaurantListViewController przejdzie do innego kontrolera widoku, sprawdzany jest identyfikator przejścia. Jeżeli kolejnym identyfikatorem jest showDetail, wykonywana jest metoda showRestaurantDetail. Tylko przejście między sceną kontrolera widoku listy restauracji a sceną kontrolera widoku szczegółów restauracji ma identyfikator showDetail, więc docelowym kontrolerem widoku musi być instancja RestaurantDetailViewController.

3. Zobaczysz błąd, ponieważ metoda showRestaurantDetail(segue:) nie została zaimplementowana. Ta metoda przekaże instancję RestaurantItem z instancji RestaurantListViewController do instancji RestaurantDetailViewController. Dodaj go po otwierającym nawiasie klamrowym rozszerzenia prywatnego w klasie RestaurantListViewController:

func showRestaurantDetail(segue: UIStoryboardSegue) {

if let viewController = segue.destination as?

RestaurantDetailViewController, let indexPath =

collectionView.indexPathsForSelectedItems?.first {

selectedRestaurant = manager.restaurantItem(at:

indexPath.row)

viewController.selectedRestaurant =

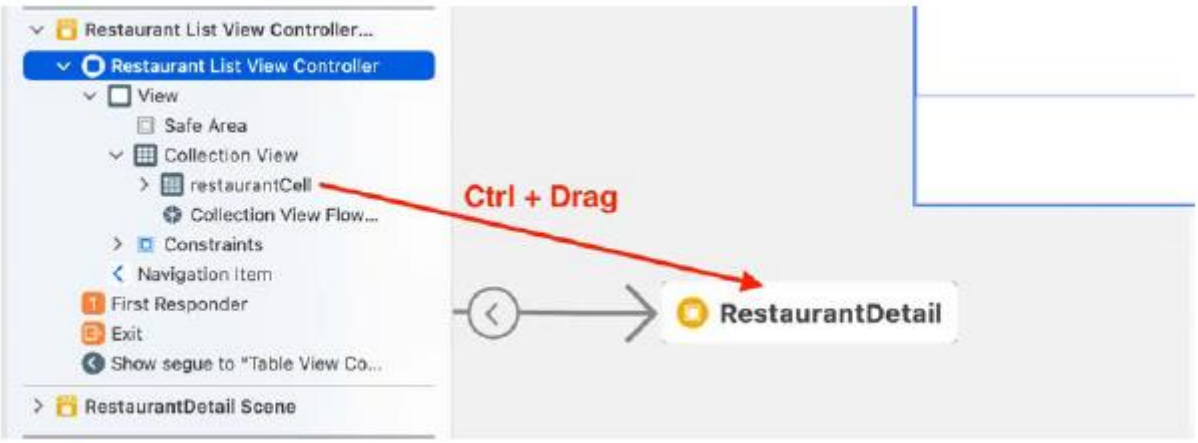
selectedRestaurant

}

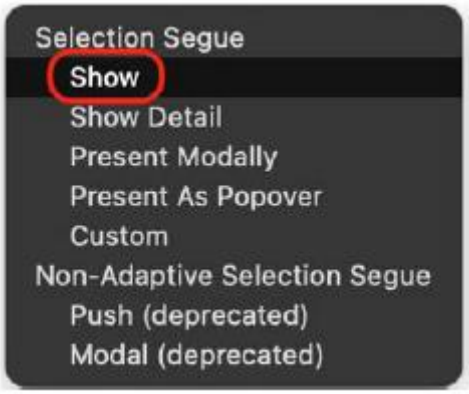
}

Ta metoda najpierw sprawdza, czy miejsce docelowe przejścia jest instancją RestaurantDetailViewController i pobiera indeks dotkniętej komórki widoku kolekcji. Następnie menedżer zwraca instancję RestaurantItem przechowywaną pod tym indeksem, która jest przypisana do wybranej restauracji. Właściwość wybraneRestaurant instancji RestaurantDetailViewController jest następnie ustawiana na tę instancję. Przyjrzyjmy się teraz scenie kontrolera widoku listy restauracji w głównym pliku scenorysu. Jest obecnie podłączony do zastępczej sceny kontrolera widoku. Zaktualizujesz główny plik scenorysu, aby usunąć element zastępczy i połączyć scenę kontrolera widoku listy restauracji ze sceną kontrolera widoku szczegółów restauracji w pliku scenorysu RestaurantDetail. Wykonaj następujące kroki:

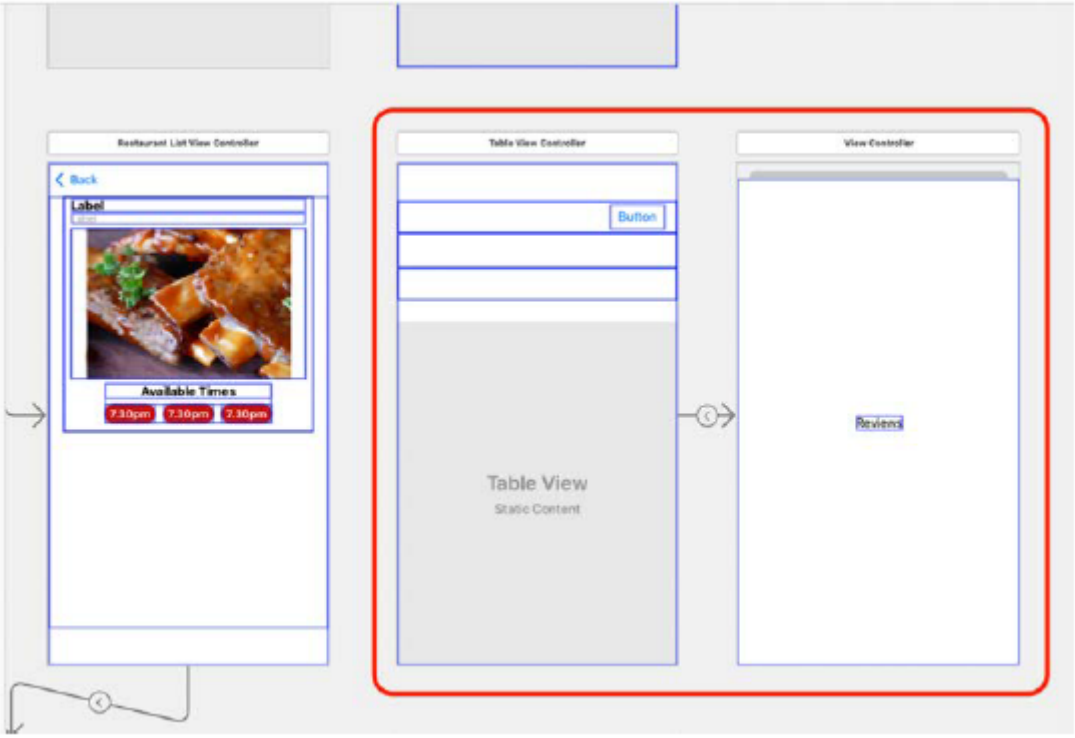
1. Kliknij plik głównego scenorysu i zlokalizuj scenę kontrolera widoku listy restauracji. Kliknij restauracjęKomórka w konspekcie dokumentu. Następnie naciśnij klawisz Ctrl i przeciągnij z RestaurantCell do odniesienia do scenorysu RestaurantDetail



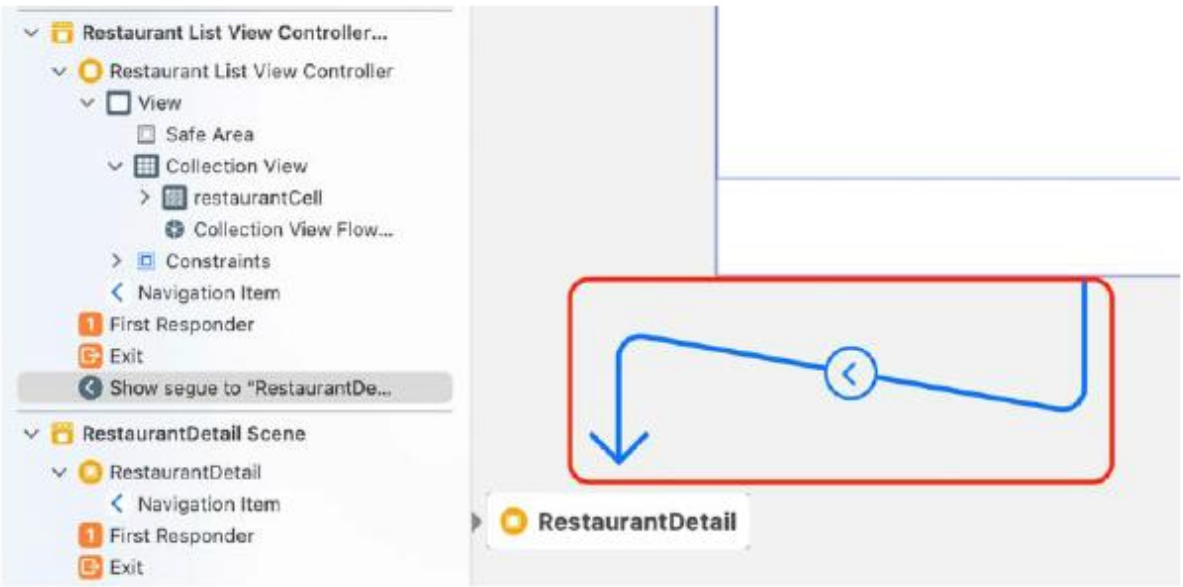
2. Wybierz opcję Pokaż z wyświetlonego menu:



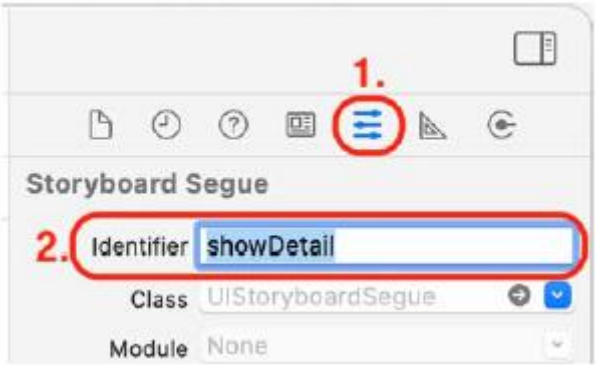
3. Usuń sceny zastępcze ze scenorysu, zaznaczając je i naciskając klawisz Delete na klawiaturze, ponieważ nie są już potrzebne:



4. Ustawisz identyfikator przejścia na showDetail. Jak wspomniano wcześniej, spowoduje to ustawienie właściwości wybranej restauracji instancji RestaurantDetailViewController. Wybierz właśnie dodany ciąg:



5. Kliknij przycisk Inspektora atrybutów. W obszarze Storyboard Segue ustaw Identyfikator na showDetail:



Zbuduj i uruchom swój projekt. Wybierz miasto i rodzaj kuchni. Kliknij jedną z restauracji na ekranie Lista restauracji. Szczegóły tej restauracji pojawią się na ekranie Szczegóły restauracji:

Implementacja ekranów szczegółów restauracji została zakończona. Po wybraniu restauracji na ekranie Mapa lub Lista restauracji szczegółowe informacje o tej restauracji zostaną wyświetlone na ekranie Szczegóły restauracji. Wspaniale!

Streszczenie

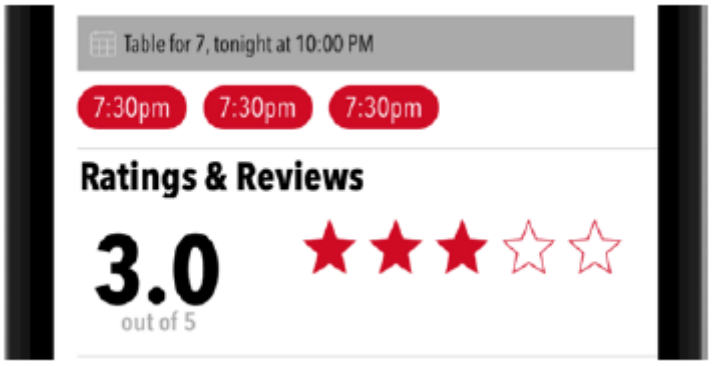
W tej części podłączyłeś outlety w klasie RestaurantDetailViewController do ekranu Szczegóły restauracji. Następnie dodano metody do metody viewDidLoad() w celu wypełnienia widoku tabeli po wyświetleniu ekranu Szczegóły restauracji. Na koniec przekazałeś odpowiednią instancję RestaurantItem z instancji RestaurantListViewController i MapViewController do instancji RestaurantDetailViewController, umożliwiając jej wyświetlenie danych z tej instancji RestaurantItem na ekranie szczegółów restauracji. W ten sposób nauczyłeś się, jak tworzyć widoki tabel ze statycznymi komórkami, aby wyświetlać dane, a także jak tworzyć niestandardowy obraz mapy, który możesz teraz zaimplementować we własnych aplikacjach. Gratulacje! Wszystkie ekrany w Twojej aplikacji wyświetlają teraz dane. Jeśli jednak spojrzysz na ekran Szczegóły restauracji, nie będzie tam żadnych ocen, recenzji ani zdjęć restauracji i nie będzie możliwości ich dodania. Zaczniesz to wdrażać począwszy od następnego rozdziału, w którym utworzysz niestandardową kontrolkę, która umożliwi dodawanie gwiazdek dla restauracji na ekranach Szczegóły restauracji i Formularz recenzji.

Pierwsze kroki z niestandardowymi UIControls

W tym momencie Twoja aplikacja zawiera dane na wszystkich ekranach, ale ekran Szczegóły restauracji jest niekompletny. Nie możesz ustawić liczby gwiazdek dla restauracji, nie możesz dodawać zdjęć ani recenzji. Do tej pory korzystałeś ze standardowych elementów interfejsu użytkownika Apple. W tej części utworzysz niestandardową podklasę klasy UIControl, która będzie wyświetlać oceny restauracji w formie gwiazdek. Zmodyfikujesz tę podklasę, aby użytkownicy mogli ustawić ocenę restauracji, dotykając jej. Następnie wdrożysz formularz recenzji, który umożliwi użytkownikom przesyłanie recenzji restauracji. Pod koniec tego rozdziału dowiesz się, jak tworzyć niestandardowe klasy UIControl, obsługiwać zdarzenia dotykowe i wdrażać formularze recenzji dla własnych aplikacji. Zacznijmy od nauczenia się, jak utworzyć niestandardową podklasę UIControl, która wyświetli ocenę w postaci gwiazdek na ekranie.

Tworzenie niestandardowej podklasy UIControl

Do tej pory korzystałeś tylko z predefiniowanych elementów interfejsu użytkownika Apple, takich jak etykiety i przyciski. Wszystko, co musiałeś zrobić, to kliknąć przycisk Biblioteka, wyszukać żądany obiekt i przeciągnąć go do scenorysu. Mogą się jednak zdarzyć przypadki, gdy przedmioty dostarczone przez Apple będą nieodpowiednie lub w ogóle nie będą istnieć. W takich przypadkach będziesz musiał zbudować własny. Przyjrzyjmy się ekranowi szczegółów restauracji, który widziałeś podczas prezentacji aplikacji:



Tuż nad przyciskiem Dodaj recenzję widać grupę pięciu gwiazdek. Obecnie scena kontrolera widoku szczegółów restauracji w pliku scenorysu RestaurantDetail i scena kontrolera widoku stołu w pliku scenorysu ReviewForm znajdują się puste obiekty widoku w miejscach, w których powinny znajdować się gwiazdy. Utworzysz klasę RatingsView, niestandardową podklasę klasy UIControl, której będziesz używać w obu scenach. Klasa UIControl jest podklasą klasy UIView i jest używana jako nadklasa dla klasy RatingsView, ponieważ instancje RatingsView muszą reagować, gdy użytkownik je dotknie. Instancja RatingsView będzie wyświetlać oceny w postaci gwiazdek. Użytkownik będzie mógł także wybrać półgwiazdki. Zacznijmy od utworzenia podklasy klasy UIControl. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Formularz recenzji i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: RatingsView

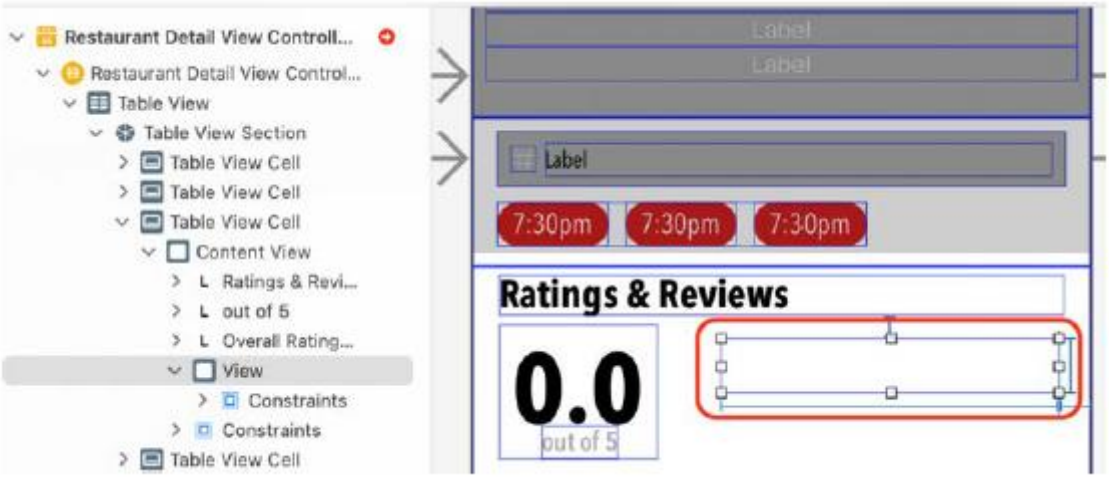
Podklasa: UIControl

Język: Swift

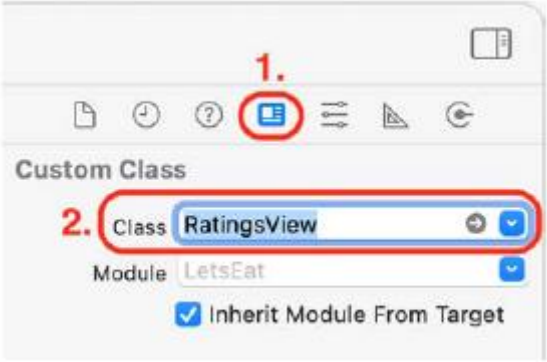
Kliknij Następny.

4. Kliknij Utwórz. Plik RatingsView pojawi się w nawigatorze projektu. Teraz musisz ustawić tożsamość obiektu widoku obok etykiety 0.0 w scenie kontrolera widoku szczegółów restauracji na RatingsView. Wykonaj następujące kroki:

1. Rozwiń folder RestaurantDetail w nawigatorze projektu. Kliknij plik scenorysu RestaurantDetail i wybierz obiekt Widok obok etykiety 0.0, jak pokazano:



2. Kliknij przycisk Inspektor tożsamości. W obszarze Klasa niestandardowa ustaw klasę na RatingsView:



Zmodyfikujmy teraz klasę RatingsView tak, aby wyświetlała gwiazdki. W następnej sekcji użyjesz do tego zasobów graficznych znajdujących się w folderze Assets.xcassets.

Wyświetlanie gwiazdek w niestandardowej podklasie UIControl

Jak dotąd w swoim projekcie utworzyłeś nową podklasę UIControl o nazwie RatingsView. Przypisałeś także klasę obiektu widoku obok etykiety 0.0 na ekranie Szczegóły restauracji do klasy RatingsView. W dalszej części tej części instancja klasy RatingsView będzie nazywana widokiem ocen (w ten sam sposób instancja klasy UIButton nazywana jest przyciskiem). W tej sekcji dodasz kod do klasy RatingsView, aby widok ocen wyświetlał gwiazdki.

Wykonaj następujące kroki:

1. Kliknij plik RatingsView w nawigatorze projektu i usuń cały skomentowany kod.

2. Wpisz następujące polecenie po deklaracji klasy RatingsView, aby zadeklarować właściwości klasy:

```
private let filledStarImage = UIImage(named:

"filled-star")

private let halfStarImage = UIImage(named:

"half-star")

private let emptyStarImage = UIImage(named:

"empty-star")

private var totalStars = 5

var rating = 0.0
```

Do pierwszych trzech właściwości, fillStarImage, halfStarImage i pustyStarImage, przypisane są obrazy gwiazd przechowywane w pliku Assets.xcassets. Właściwość totalStars określa całkowitą liczbę gwiazdek do wylosowania. Właściwość ratingowa służy do przechowywania oceny restauracji. Rodzaj wylosowanych gwiazdek zostanie określony na podstawie wartości oceny. Na przykład, jeśli ocena wynosi 3,5, w widoku ocen zostaną wyświetlone trzy gwiazdki wypełnione, jedna gwiazda w połowie wypełniona i jedna gwiazdka pusta. Następnie utwórzmy metodę, która będzie rysować widok ocen na ekranie. Wszystkie podklasy UIView posiadają metodę remis(_:) odpowiedzialną za rysowanie ich widoków na ekranie. Zastąpisz implementację tej metody w nadklasie dla klasy RatingsView. Wykonaj następujące kroki:

1. Dodaj następujący kod w deklaracji klasy po deklaracjach właściwości:

```
override func draw(_ rect: CGRect) {

let context = UIGraphicsGetCurrentContext()

context!.setFillColor(UIColor.systemBackground.

cgColor)

context!.fill(rect)

let ratingsViewWidth = rect.size.width

let availableWidthForStar = ratingsViewWidth /

Double(totalStars)

let starSidelength = (availableWidthForStar <=

rect.size.height) ? availableWidthForStar :

rect.size.height

for index in 0..<totalStars {

let starOriginX = (availableWidthForStar *

Double(index)) + ((availableWidthForStar -

starSidelength) / 2)
```

```
let starOriginY = ((rect.size.height -
starSidelength) / 2)
let frame = CGRect(x: starOriginX,
y: starOriginY, width: starSidelength,
height: starSidelength)
var starToDraw: UIImage!
if (Double(index + 1) <= self.rating) {
starToDraw = filledStarImage
} else if (Double(index + 1) <=
self.rating.rounded()) {
starToDraw = halfStarImage
} else {
starToDraw = emptyStarImage
}
starToDraw.draw(in: frame)
}
}
```

Rozbijmy to:

```
let context = UIGraphicsGetCurrentContext()
```

Tworzy to instancję UIGraphicsGetCurrentContext i przypisuje ją do kontekstu. Można o nim myśleć jak o szkicowniku, w którym będziesz komponował elementy interfejsu użytkownika.

```
context!.setFillColor(UIColor.systemBackground.cgColor)
```

Ustawia kolor wypełnienia kontekstu na domyślny systemowy kolor tła.

```
context!.fill(rect)
```

Spowoduje to wypełnienie prostokątnego obszaru określonego przez rect kolorem wypełnienia.

```
let ratingsViewWidth = rect.size.width
let availableWidthForStar = ratingsViewWidth /
Double(totalStars)
let starSidelength = (availableWidthForStar <=
rect.size.height) ? availableWidthForStar :
rect.size.height
```

Stwierdzenia te określają, jak duża powinna być każda gwiazda. Pierwsza instrukcja pobiera szerokość widoku ocen i przypisuje ją do ratingsViewWidth. Następna instrukcja oblicza szerokość dostępną dla każdej gwiazdy poprzez podzielenie szerokości widoku ocen przez liczbę gwiazdek, które należy narysować. Ta wartość jest przypisana do availableWidthForStar. W przypadku trzeciego stwierdzenia wyobraź sobie, że każda gwiazda jest zamknięta w prostokącie. Ta instrukcja oblicza, jak długi powinien być każdy bok tego prostokąta, aby zmieścił się w widoku ocen. Jeśli availableWidthForStar jest mniejszy lub równy wysokości widoku ocen, starSideLength jest ustawiany na availableWidthForStar; w przeciwnym razie jest równa wysokości widoku ocen.

Załóżmy na przykład, że widok ocen ma szerokość 200 punktów i wysokość 50 punktów. availableWidthForStar będzie wynosić 200/5 = 40. Ponieważ 40 <= 50 ma wartość true, starSideLength zostanie ustawiona na 40.

```
let starOriginX = (availableWidthForStar *

Double(index)) + (availableWidthForStar –

starSidelength) / 2

let starOriginY = ((rect.size.height - starSidelength)

/ 2)

let frame = CGRect(x: starOriginX, y: starOriginY,

width: starSidelength, height: starSidelength)
```

Instrukcje te obliczają początek i rozmiar prostokąta, w którym powinna zostać narysowana każda gwiazda w widoku ocen. Jest to następnie przypisywane do ramki. Wartości początkowe są odsunięte od lewego górnego rogu widoku ocen, a szerokość i wysokość są ustawione na starSidelength. Na przykład dla pierwszej gwiazdy starOriginX wynosi (40*0,0) + (40-40)/2 = 0. starOriginY wynosi (50 – 40)/2 = 5. ramka byłaby zatem CGRect, gdzie x wynosi 0, y wynosi 5, szerokość wynosi 40, a wysokość wynosi 40.

```
var starToDraw: UIImage!

if (Double(index + 1) <= self.rating) {

starToDraw = filledStarImage

} else if (Double(index + 1) <= self.rating.rounded())

{

starToDraw = halfStarImage

} else {

starToDraw = emptyStarImage

}
```

W zależności od wartości właściwości oceny widoku ocen, instrukcje te określają, czy rysowana gwiazda jest wypełniona, w połowie wypełniona czy pusta. Załóżmy na przykład, że ocena wynosi 3,5. Pierwsza gwiazda ma indeks 0. Oznacza to, że Double(0 + 1) <= 3,5 będzie wynosić 1,0 <= 3,5, co daje wartość true. Oznacza to, że pierwsza narysowana gwiazda będzie gwiazdą wypełnioną. To samo

dotyczy drugiej i trzeciej gwiazdy. Czwarta gwiazda ma indeks 3. Oznacza to, że Double(3 + 1) <= 3,5 będzie wynosić 4,0 <= 3,5, co daje wartość false. Klauzula else ocenia wartość Double(3 + 1) <= 4,0, co daje wartość true, więc czwarta wylosowana gwiazda będzie gwiazdą w połowie wypełnioną. Piąta gwiazda ma indeks 4. Oznacza to, że Double(4 + 1) <= 3,5 będzie wynosić 5,0 <= 3,5, co daje wartość false. Klauzula else zwraca wartość Double(5 + 1) <= 4,0, co również daje wartość false, więc piąta narysowana gwiazda będzie gwiazdą pustą.

starToDraw.draw(in: frame)

Ta instrukcja rysuje gwiazdę w określonej ramce.

To cały kod potrzebny dla klasy RatingsView. Dodajmy teraz outlet do klasy RestaurantDetailViewController, aby mogła ona zarządzać tym, co wyświetla widok ocen. Wykonaj następujące kroki:

1. Kliknij plik RestaurantDetailViewController w nawigatorze projektu.

2. Wpisz następujący kod po wylocie totalRatingLabel:

@IBOutlet var ratingsView: RatingsView!

Tworzy to miejsce w klasie RestaurantDetailViewController dla widoku ocen. Masz teraz punkt sprzedaży o nazwie ratingsView typu RatingsView, który później połączysz z widokiem ocen w scenorysie.

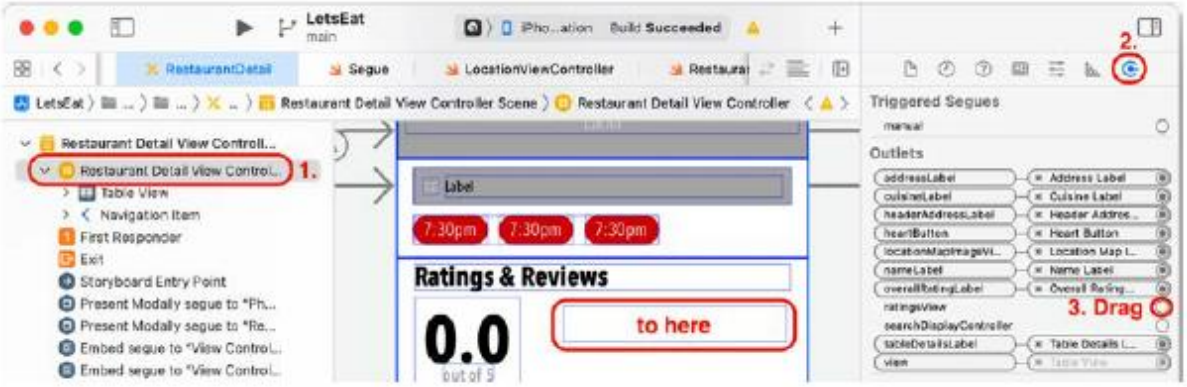
3. Dodaj metodę przypisania wartości 3,5 do właściwości ratingowej instancji ratingsView. Wpisz następujące polecenie w swoim prywatnym rozszerzeniu po metodzie inicjalizacji():

```
func createRating() {

ratingsView.rating = 3.5

}
```

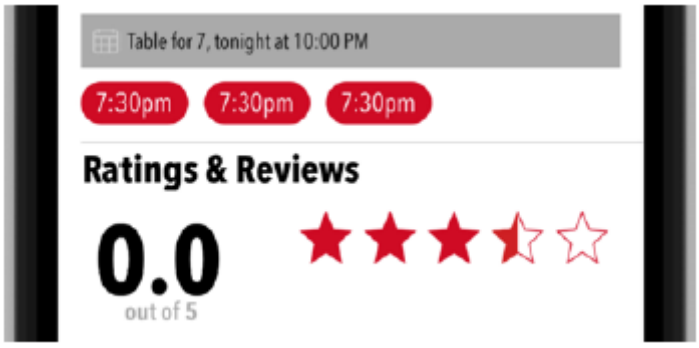
4. Zmodyfikuj metodę initalize(), aby wywołać metodę createRating():

```
func initialize() {

setupLabels()

createMap()

createRating()

}
```

5. Otwórz plik scenorysu RestaurantDetail i wybierz opcję Kontroler widoku szczegółów restauracji w konspekcie dokumentu. Kliknij przycisk Inspektora połączeń. Przeciągnij z gniazdka ratingsView do widoku ocen:



Zbuduj i uruchom swój projekt, a następnie przejdź do RestaurantDetailView dla dowolnej restauracji. Widok ocen powinien wyświetlać trzy i pół gwiazdki:



Utworzyłeś i zaimplementowałeś widok ocen dla ekranu Szczegóły restauracji. Wygląda świetnie, ale w tej chwili widok ocen nie reaguje na dotknięcie. W następnej sekcji sprawisz, że będzie reagował na zdarzenia dotykowe, aby użytkownik mógł wybrać ocenę.

Dodanie obsługi zdarzeń dotykowych

Obecnie klasa RestaurantDetailViewController ma punkt wyjściowy ratingsView, połączony z widokiem ocen na ekranie Szczegóły restauracji. Wyświetla ocenę trzech i pół gwiazdki, ale nie można jej zmienić. Aby widok ocen reagował na dotknięcia, musisz obsługiwać zdarzenia dotykowe. Aby obsługiwać zdarzenia dotykowe, zmodyfikuj klasę RatingsView tak, aby śledziła dotknięcia użytkownika na ekranie i wykorzystywała je do ustalania oceny. Wykonaj następujące kroki:

1. Kliknij plik RatingsView w nawigatorze projektu i po metodzie remis(_:) dodaj następującą właściwość:

```
override var canBecomeFirstResponder: Bool {

true

}
```

canBecomeFirstResponder to właściwość UIControl, która określa, czy obiekt może zostać osobą reagującą jako pierwszy. Widok ocen musi być pierwszą osobą reagującą na zdarzenia dotykowe. Ta metoda domyślnie zwraca wartość false, ponieważ nie wszystkie elementy interfejsu użytkownika muszą reagować na dotknięcia. Zastępujesz tę metodę, aby zwróciła wartość true, dzięki czemu widok ocen może stać się pierwszą odpowiedzią.

2. Aby śledzić dotknięcia użytkownika na ekranie, dodaj następujący kod po właśnie dodanej właściwości canBecomeFirstResponder.

```
override func beginTracking(_ touch: UITouch, with

event: UIEvent?) -> Bool {

guard self.isEnabled else {

return false

}

super.beginTracking(touch, with: event)

handle(with: touch)

return true

}
```

Rozbijmy to:

```
override func beginTracking(_ touch: UITouch, with event: UIEvent?) ->

Bool {
```

Metoda ta jest jedną z metod zadeklarowanych w klasie UIControl. Jest wywoływana, gdy dotyk użytkownika znajduje się w granicach instancji UIControl. Lokalizacja, rozmiar, ruch i siła dotyku na ekranie są przechowywane w instancji UITouch. Ta metoda musi zwracać wartość true, jeśli chcesz śledzić dotknięcia użytkownika. Zastępujesz tę metodę, aby móc zdefiniować niestandardowe zachowanie, gdy użytkownik dotknie widoku ocen.

```
guard self.isEnabled else {

return false

}
```

W tym oświadczeniu ochronnym sprawdzana jest właściwość isEnabled, aby sprawdzić, czy widok ocen jest włączony. Jeśli widok ocen nie jest włączony, dotknięcia użytkownika nie będą śledzone.

```
super.beginTracking(touch, with: event)
```

Nazywa się to implementacją tej metody w nadklasie. Zajmie się to inicjalizacją wymaganą przez klasę nadrzędną.

```
handle(with: touch)
```

Przekażesz instancję UITouch do tej metody, która będzie wykonywana przy każdym dotknięciu. W następnym kroku zadeklarujesz i zdefiniujesz tę metodę.

```
return true
```

Śledzi dotknięcia użytkownika, gdy włączony jest widok ocen.

3. Zobaczysz błąd, ponieważ nie zaimplementowałeś jeszcze handle(with:), więc utwórz prywatne rozszerzenie dla RatingsView po całym innym kodzie w pliku i wpisz w nim następujący kod:

```
private extension RatingsView {

func handle(with touch: UITouch) {

let starRectWidth = self.bounds.size.width /

Double(totalStars)

let location = touch.location(in: self)

var value = location.x / starRectWidth

if (value + 0.5) < value.rounded(.up) {

value = floor(value) + 0.5

} else {

value = value.rounded(.up)

}

updateRating(with: value)

}

}
```

handle(with:) obliczy wartość oceny na podstawie miejsca dotknięcia użytkownika. Jako parametr przyjmuje instancję UITouch. Najpierw do parametru starRectWidth przypisywana jest szerokość widoku ocen podzielona przez 5. Następnie przydzielana jest lokalizacja instancji UITouch w widoku ocen Lokalizacja. Następnie przypisuje się wartość x pozycji lokalizacji podzielonej przez starRectWidth. Oznacza to, że wartość będzie zawierać zakres wartości od 0 do 5. Następnie instrukcja if oblicza ocenę odpowiadającą pozycji dotknięcia i wywołuje funkcję updateRating(with:), przekazując jej wartość. W następnym kroku zaimplementujesz updateRating(with:) Aby zrozumieć, jak działa instrukcja if, załóżmy, że szerokość widoku ocen wynosi 200. starRectWidth zostanie ustawiony na 200/5 = 40. Załóżmy, że użytkownik dotknął ekranu w pozycji x = 130, y = 17, co odpowiada punktowi pomiędzy trzecią i czwartą gwiazdą. wartość zostanie przypisana 130/40 = 3,25. Zatem instrukcja if obliczyłaby (3,25 + 0,5 < 0,25.zaokrąglone(.up), co daje (3,75 < 4,0), co zwraca wartość true, a zatem wartość zostałaby ustawiona na Floor(3,25) + 0,5, co daje 3,0 + 0,5, czyli 3,5. Zatem do parametru updateRating(with:) zostanie przekazana wartość 3,5.

4. Zobaczysz błąd, ponieważ nie zaimplementowałeś jeszcze updateRating(with:), więc wpisz następujący kod do prywatnego rozszerzenia po metodzie handle(with:):

```
func updateRating(with newValue: Double) {

if (self.rating != newValue && newValue >= 0 &&

newValue <= Double(totalStars)) {

self.rating = newValue

}

}
```

updateRating(with:) sprawdza, czy wartość nie jest równa aktualnej ocenie i mieści się w przedziale od 0 do 5. Jeśli tak, wartość jest przypisywana do oceny. Kontynuując poprzedni przykład, ponieważ 3,5 mieści się w przedziale od 0 do 5, zostanie przypisane do oceny, jeśli nie jest równe bieżącej wartości oceny.

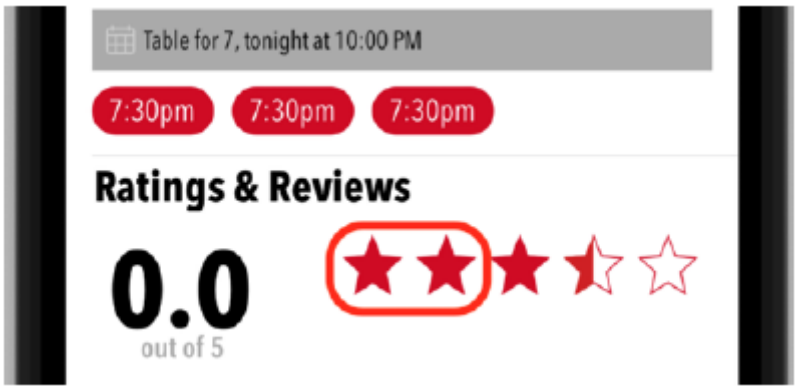
5. Po zmianie oceny konieczne będzie ponowne narysowanie widoku ocen, aby wyświetlić prawidłowy stan gwiazdek. Zmodyfikuj deklarację właściwości ratingu w następujący sposób:

```
var rating = 0.0 {

didSet {

setNeedsDisplay()

}

}
```

W tym miejscu zdefiniowano obserwatora właściwości, który będzie monitorował zmiany wartości ratingu. Za każdym razem, gdy zmienia się ocena, wywoływana jest metoda setNeedsDisplay() i widok ocen jest rysowany na nowo. Ponieważ ekran jest odświeżany tylko w przypadku zmiany wartości oceny, istnieje niewielka korzyść w zakresie wydajności. Dodałeś cały kod niezbędny, aby widok ocen reagował na dotknięcia. Teraz musisz zaktualizować klasę RestaurantDetailViewController, aby ustawić właściwość isEnabled dla widoku ocen. Kliknij plik RestaurantDetailViewController w nawigatorze projektu i zmodyfikuj metodę createRating() w następujący sposób:

```
func createRating() {

ratingsView.rating = 3.5

ratingsView.isEnabled = true

}
```

Ustawienie właściwości isEnabled na true pozwala widokowi ocen stać się pierwszą odpowiedzią i rozpocząć śledzenie dotknięć, co uruchomi funkcję handle(with:) w celu obliczenia oceny na podstawie pozycji dotknięcia, co z kolei wywoła funkcję updateRating(with:) zaktualizować widok ocen. Zbuduj i uruchom swój projekt. Kliknięcie widoku ocen zmienia teraz ocenę w zależności od miejsca dotknięcia. Kliknij pomiędzy pierwszą i drugą gwiazdką, jak pokazano:

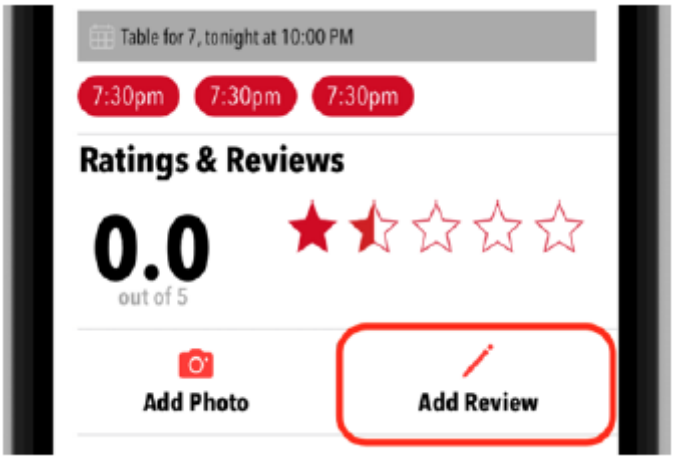


Ocena zmieni się na półtorej gwiazdki. Ostatecznie obliczysz ogólną ocenę, sumując wszystkie oceny przesłane przez użytkowników na ekranie Formularza recenzji. Jeśli dotkniesz przycisku Dodaj recenzję,

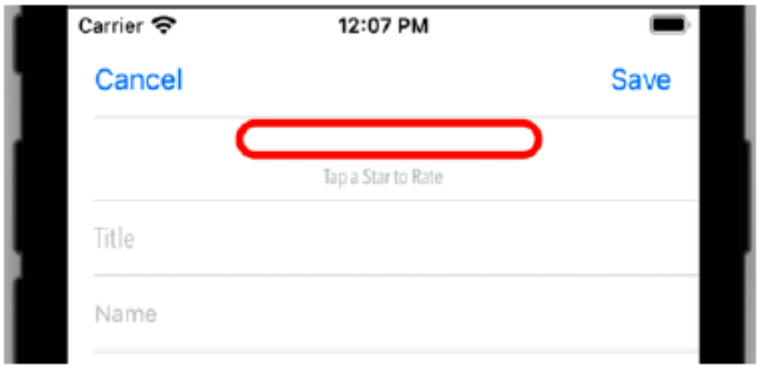
wyświetli się ekran Formularz recenzji, ale nie będziesz mógł go odrzucić ani ustawić oceny. W następnej sekcji skonfigurujesz przycisk Anuluj, aby po dotknięciu zamknąć ekran Formularza recenzji.

Implementacja metody unwind dla przycisku Anuluj

Przyjrzyjmy się ekranowi formularza recenzji. Przejście pomiędzy przyciskiem Dodaj recenzję a ekranem Formularza recenzji zostało już dla Ciebie wykonane. Zbuduj i uruchom swój projekt, przejdź do ekranu Szczegóły restauracji i dotknij przycisku Dodaj recenzję:



Zostanie wyświetlony ekran Formularz recenzji (zwróć uwagę, że w górnej komórce widoku tabeli znajduje się puste miejsce w miejscu, w którym powinien znajdować się widok ocen, który dodasz później):



Gdy na ekranie pojawi się ekran Formularz recenzji, nie można go zamknąć, ponieważ akcje przycisków Zapisz i Anuluj nie zostały skonfigurowane. Podobnie jak w przypadku ekranu Lokalizacje, należy zaimplementować metodę rozwijania, aby zamknąć ekran Formularz recenzji. Wykonaj następujące kroki:

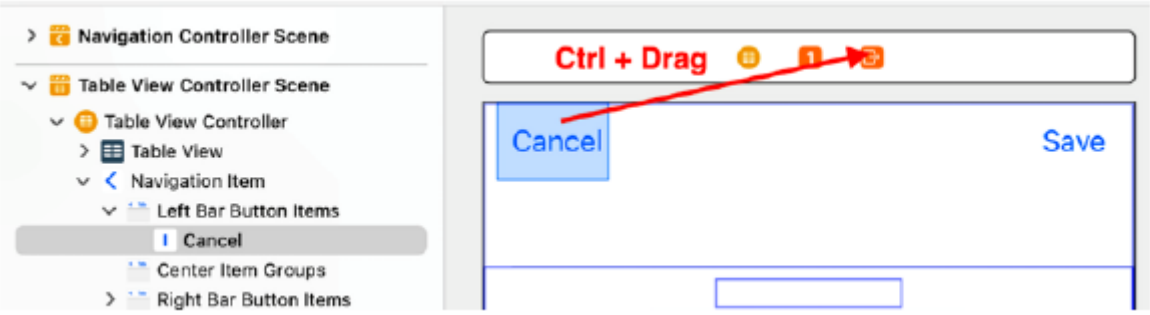
1. Kliknij plik RestaurantDetailViewController w nawigatorze projektu.

2. Zaimplementuj metodę unwind w rozszerzeniu prywatnym, przed metodą createRating() w następujący sposób:

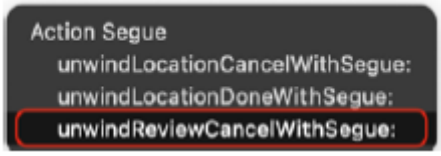
```
@IBAction func unwindReviewCancel(segue:

UIStoryboardSegue) {

}
```

Ta metoda zostanie wywołana, gdy ekran formularza recenzji przejdzie do ekranu szczegółów restauracji.

3. Otwórz plik scenorysu ReviewForm i naciśnij Ctrl + przeciągnij od przycisku Anuluj do ikony Wyjdź w Doku Sceny, jak pokazano:



4. Z wyskakującego menu wybierz opcję unwindReviewCancelWithSegue:



Zbuduj i uruchom swój projekt. Możesz teraz zamknąć ekran Formularza recenzji, dotykając przycisku Anuluj. Następnie przyjrzyjmy się przyciskowi Zapisz. Utworzysz kontroler widoku dla ekranu Formularza recenzji, aby przetwarzać dane w polach ekranu Formularza recenzji po dotknięciu przycisku Zapisz. Zrobisz to w następnej sekcji.

Tworzenie klasy ReviewFormViewController

Aby przetwarzać dane wejściowe użytkownika, utworzysz klasę ReviewFormViewController, która będzie kontrolerem widoku na ekranie formularza recenzji. Na razie skonfigurujesz tę klasę tak, aby pobierała wszystkie wartości z pliku

Przejrzyj pola na ekranie formularza i wydrukuj je w obszarze debugowania. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder ReviewForm i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: ReviewFormViewController

Podklasa: UITableViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. Plik ReviewFormViewController pojawi się w nawigatorze projektu.

5. Usuń cały kod i komentarze po metodzie viewDidLoad(). Dodaj następujące gniazda po deklaracji klasy. Odpowiadają one polom na ekranie Formularza recenzji:

@IBOutlet var ratingsView: RatingsView!

@IBOutlet var titleTextField: UITextField!

@IBOutlet var nameTextField: UITextField!

@IBOutlet var reviewTextView: UITextView!

6. Należy także skonfigurować akcję dla przycisku Zapisz. Dodaj następujący kod po metodzie viewDidLoad():

@IBAction func onSaveTapped(_ sender: Any) {

print(ratingsView.rating)

print(titleTextField.text as Any)

print(nameTextField.text as Any)

print(reviewTextView.text as Any)

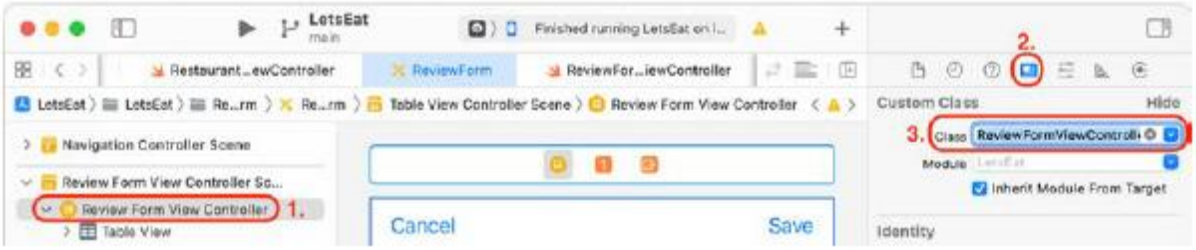
dismiss(animated: true, completion: nil)

}

Ta metoda drukuje zawartość pól ekranu formularza recenzji w obszarze debugowania i odrzuca ją.

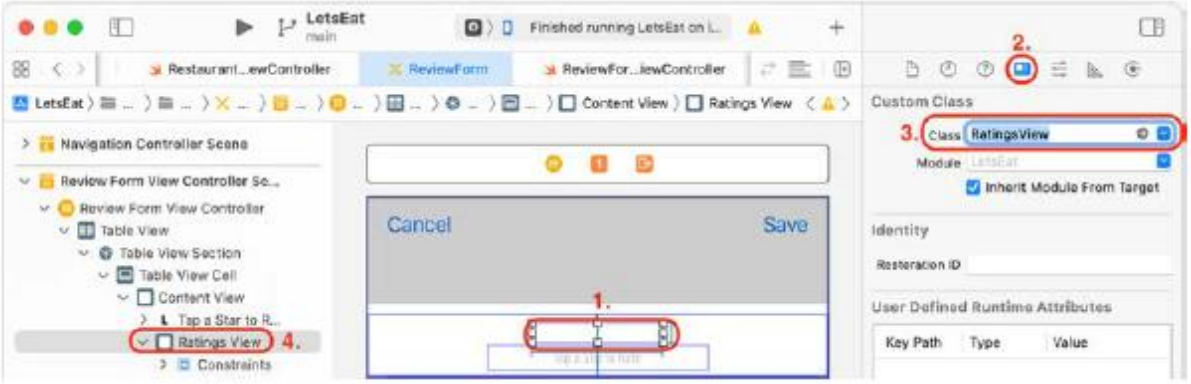
Teraz połączmy wyjścia w klasie ReviewFormViewController z elementami interfejsu użytkownika w scenie kontrolera widoku tabeli w pliku scenorysu ReviewForm w następujący sposób:

1. Kliknij plik scenorysu ReviewForm w nawigatorze projektu i kliknij ikonę Kontrolera widoku tabeli w scenie kontrolera widoku tabeli. Kliknij przycisk Inspektora tożsamości. W obszarze Klasa niestandardowa ustaw klasę na ReviewFormViewController.



Należy pamiętać, że nazwa Scena kontrolera widoku tabeli zmieni się na Scena kontrolera widoku formularza recenzji.

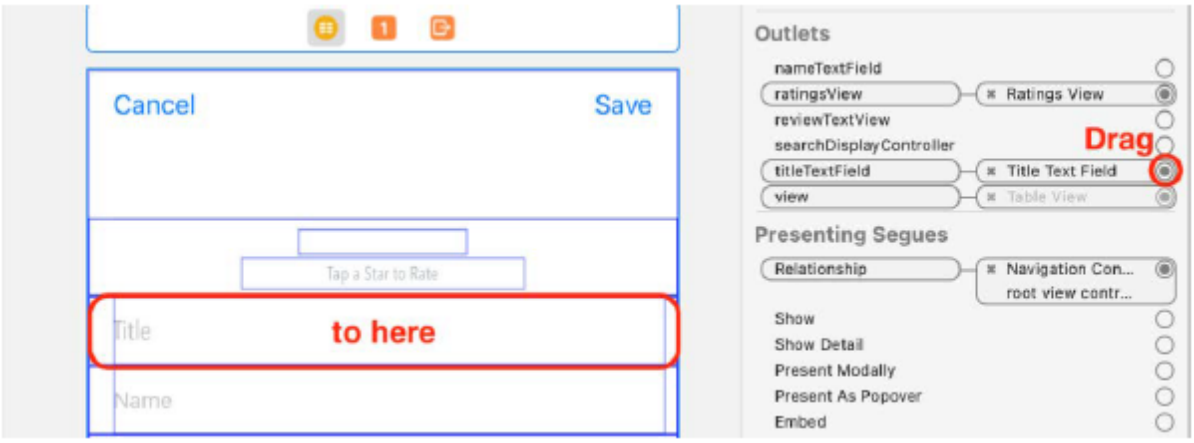
2. Kliknij opcję Widok w widoku zawartości pierwszej komórki widoku tabeli, jak pokazano na rysunku, kliknij przycisk Inspektor tożsamości i w obszarze Klasa niestandardowa ustaw opcję Klasa na RatingsView. Nazwa widoku zmieni się na Widok ocen:



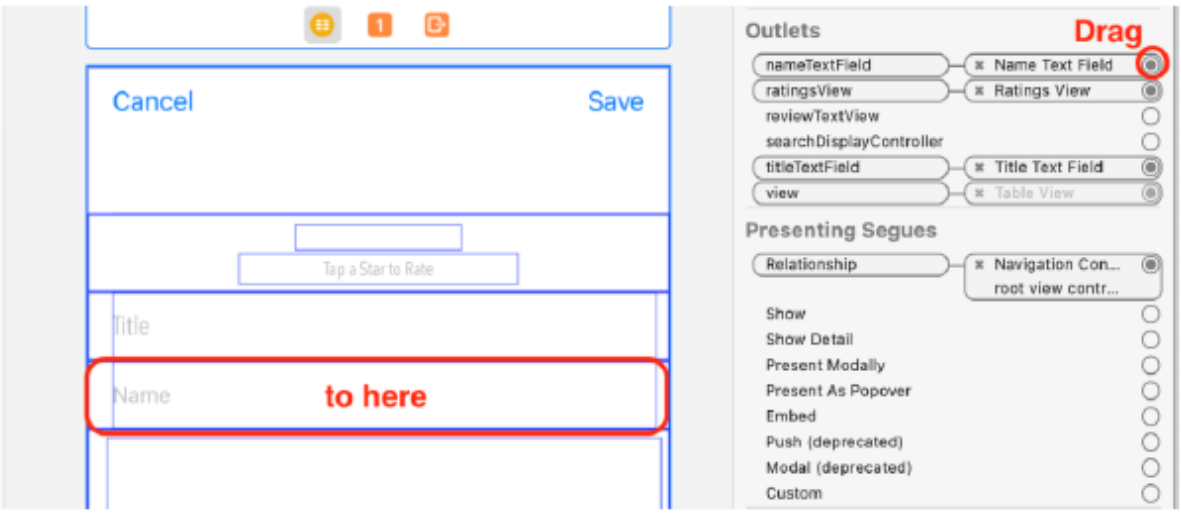
3. Następnie połączysz gniazda. Kliknij ikonę Kontroler widoku formularza recenzji w konspekcie dokumentu, a następnie kliknij przycisk Inspektora połączeń:

4. Połącz wyjście ratingsView z widokiem ocen:

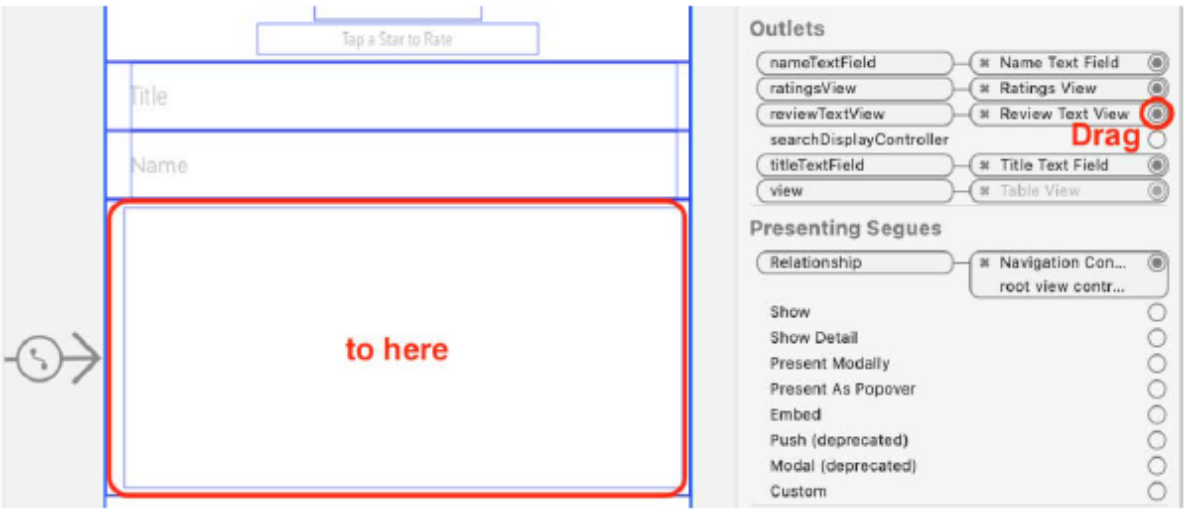
5. Podłącz wyjście titleTextField do pierwszego pola tekstowego:



6. Podłącz wyjście nameTextField do drugiego pola tekstowego:

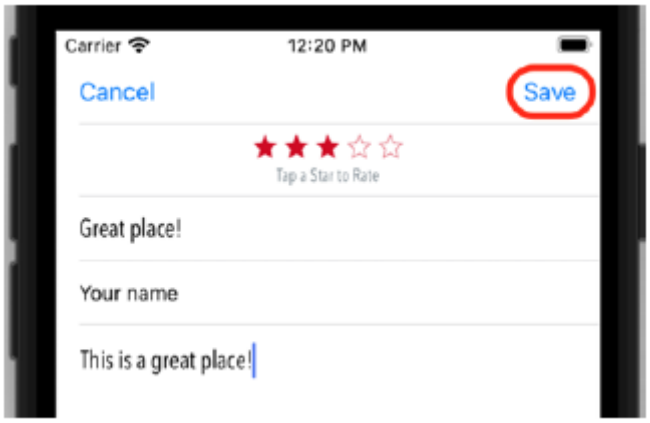


7. Podłącz wyjście reviewTextView do Text View:



8. Na koniec połącz akcję onSaveTapped: z przyciskiem Zapisz:

Kompiluj i uruchamiaj swoją aplikację. Przejdź do ekranu Formularz recenzji, ustaw ocenę, dodaj przykładowy tekst do pól i dotknij przycisku Zapisz:



Zobaczysz, że wprowadzone dane pojawią się w obszarze Debugowanie:

```
3.0
Optional("Great place!")
Optional("Your name")
Optional("This is a great place!")
```

Gratulacje! Na ekranie formularza recenzji można teraz akceptować dane wprowadzane przez użytkownika.

Streszczenie

W tej części utworzyłeś od podstaw nową niestandardową podklasę UIControl, RatingsView i dodałeś ją do ekranów Szczegóły restauracji i Formularz recenzji. Skonfigurowałeś go tak, aby reagował na dotknięcia, dzięki czemu użytkownik może ustawić ocenę restauracji na ekranie formularza recenzji. Na koniec zaimplementowano klasę ReviewFormViewController, kontroler widoku dla ekranu formularza recenzji, oraz skonfigurowano akcje przycisków Anuluj i Zapisz, aby użytkownik mógł zamknąć ekran formularza recenzji lub przesłać recenzję. Teraz dobrze wiesz, jak tworzyć niestandardowe klasy UIControl, jak sprawić, by reagowały na interakcję użytkownika i jak zaimplementować formularz recenzji, który akceptuje dane wprowadzane przez użytkownika. Będzie to przydatne podczas pisania własnych aplikacji. W następnym rozdziale dowiesz się, jak pracować ze zdjęciami z aparatu lub Biblioteki zdjęć, a także jak stosować filtry fotograficzne do posiadanych zdjęć.

Pierwsze kroki z aparatami i bibliotekami zdjęć

W poprzedniej części utworzyłeś klasę RatingsView i dodałeś ją do ekranów Szczegóły restauracji i Formularz recenzji. Umożliwiłeś także użytkownikowi przesłanie recenzji za pomocą ekranu Formularz recenzji, chociaż przesłana recenzja jest na razie drukowana tylko w obszarze Debugowanie. W tej części zakończysz implementację ekranu Filtr zdjęć, dzięki czemu będziesz mógł pobrać zdjęcie z aparatu lub biblioteki zdjęć i zastosować do niego filtr. Zaczniesz od zaimportowania pliku .plist zawierającego filtry, których chcesz użyć, utworzenia klasy obiektu filtra do przechowywania danych filtrów i utworzenia klasy menedżera danych do odczytania pliku .plist i zapełnienia tablicy obiektów filtrów. Następnie utworzysz protokół z metodą stosowania filtrów do obrazów. Następnie utworzysz kontrolery widoku dla ekranu Filtru zdjęć i znajdującego się na nim widoku kolekcji, zaimplementujesz protokół UIImagePickerDelegate, który umożliwia pobieranie zdjęć z aparatu lub biblioteki zdjęć, oraz zaimplementujesz metody zastosowania wybranego filtra do zdjęcia . Pamiętaj, że zdjęcie nie zostanie zapisane. W następnym rozdziale dowiesz się, jak zapisywać recenzje i zdjęcia. Pod koniec tej części dowiesz się, jak importować zdjęcia do własnych aplikacji i jak stosować do nich filtry.

Zrozumienie filtrów

iOS ma szereg wbudowanych filtrów, których możesz używać do ulepszania zdjęć. Filtry te są dostępne w bibliotece Core Image. Core Image to technologia przetwarzania i analizy obrazu, która zapewnia wysoką wydajność przetwarzania obrazów nieruchomych i wideo. W programie Core Image dostępnych jest ponad 170 filtrów, które umożliwiają zastosowanie szerokiej gamy ciekawych efektów do zdjęć. W tej aplikacji będziesz używać tylko 10 filtrów. Szczegóły tych filtrów znajdują się w pliku .plist.

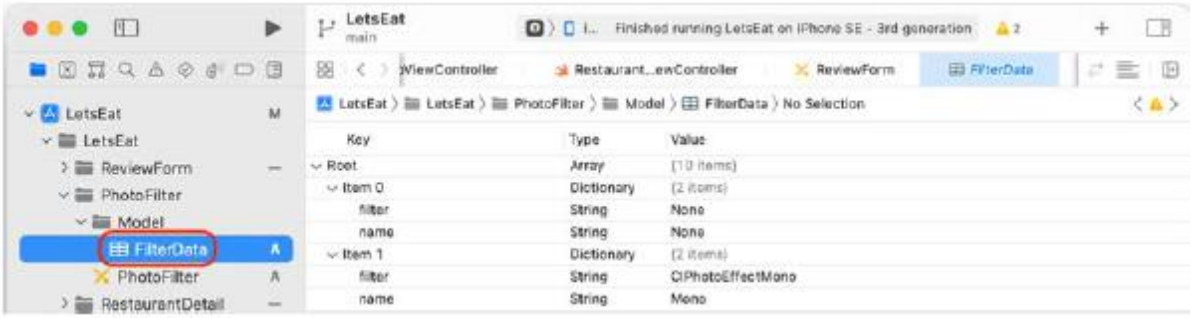
Zaimportuj ten plik do swojej aplikacji, wykonując następujące kroki:

1. Jeśli jeszcze tego nie zrobiłeś, pobierz i rozpakuj pakiet kodu tej książki pod tym linkiem: https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Seventh-Edition. Plik FilterData.plist znajdziesz w folderze Resources w folderze Chapter21.

2. W nawigatorze projektu utwórz nową grupę w folderze PhotoFilter i nadaj jej nazwę Model.

3. Przeciągnij plik FilterData.plist do folderu Model. Upewnij się, że jest zaznaczona opcja Kopiuj elementy, jeśli to konieczne, i kliknij Zakończ.

4. Kliknij FilterData.plist w nawigatorze projektu, aby zobaczyć, co zawiera:



Jak widać, FilterData.plist to tablica słowników. Każdy słownik zawiera nazwę filtra i etykietę opisową. W następnej sekcji zobaczysz, jak możesz wykorzystać informacje zawarte w pliku FilterData.plist w swojej aplikacji.

Tworzenie obiektów modelu dla ekranu Filtru zdjęć

Aby pobrać informacje z pliku FilterData.plist do swojej aplikacji, utworzysz strukturę FilterItem, w której będą przechowywane szczegółowe informacje o filtrze, oraz klasę menedżera danych FilterManager, która będzie ładować FilterData. plist i utwórz tablicę instancji FilterItem. Jest to podobne do metody ładowania informacji o kuchni i lokalizacji do aplikacji. Zacznijmy od stworzenia struktury FilterItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik FilterItem. Kliknij Utwórz. Plik FilterItem pojawi się w nawigatorze projektu.

4. W pliku FilterItem wpisz następujący kod po instrukcji import, aby zadeklarować i

zdefiniuj strukturę FilterItem:

```
struct FilterItem {

let filter: String?

let name: String?

init(dict: [String: String]) {

self.filter = dict["filter"]

self.name = dict["name"]

}

}
```

Struktura ta ma dwie właściwości i inicjator. Właściwość filter będzie przechowywać nazwy filtrów, a właściwość name będzie przechowywać krótki opis filtra. Inicjator przyjmuje słownik jako parametr, aby ustawić nazwę i właściwości filtra podczas tworzenia instancji tej klasy. Po utworzeniu klasy FilterItem utworzysz klasę menedżera danych FilterDataManager. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik FilterDataManager. Kliknij Utwórz. Plik FilterDataManager pojawi się w nawigatorze projektu.

4. W pliku FilterDataManager wpisz następujący kod po instrukcji import, aby zadeklarować i zdefiniować klasę FilterDataManager:

```
class FilterDataManager: DataManager {

func fetch() -> [FilterItem] {

var filterItems: [FilterItem] = []

for data in loadPlist(file: "FilterData") {

filterItems.append(FilterItem(dict:

data as! [String: String]))
```

```
}

return filterItems

}

}
```

Klasa FilterDataManager przyjmuje protokół DataManager utworzony wcześniej w rozdziale 17, Pierwsze kroki z MapKit. Wywołanie metody fetch() ładuje dane z pliku FilterData.plist, tworzy tablicę instancji FilterItem i zwraca je. W następnej sekcji utworzysz protokół z metodą zastosowania filtra do obrazu.

Tworzenie protokołu ImageFiltering

Potrzebujesz sposobu na zastosowanie filtra do obrazu. W tym celu utworzysz protokół ImageFiltering, który implementuje metodę Apply(filter:to:). Każda klasa, która przyjmuje ten protokół, będzie miała dostęp do tej metody, która stosuje określony filtr do obrazu. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik ImageFiltering. Kliknij Utwórz. Plik ImageFiltering pojawi się w nawigatorze projektu.

4. Zmodyfikuj kod w tym pliku, aby zadeklarować i zdefiniować protokół ImageFiltering:

```
import UIKit

import CoreImage

protocol ImageFiltering {

func apply(filter: String, originalImage:

UIImage) -> UIImage

}

extension ImageFiltering {

func apply(filter: String, originalImage:

UIImage) -> UIImage {

let initialCIImage = CIImage(image:

originalImage, options: nil)

let originalOrientation =

originalImage.imageOrientation

guard let ciFilter = CIFilter(name:

filter) else {

print("filter not found")

return originalImage
```

```
}
ciFilter.setValue(initialCIImage, forKey:
kCIInputImageKey)
let context = CIContext()
let filteredCIImage =
(ciFilter.outputImage)!
let filteredCGImage =
context.createCGImage(filteredCIImage,
from: filteredCIImage.extent)
return UIImage(cgImage: filteredCGImage!,
scale: 1.0, orientation:
originalOrientation)
}
}
```

Rozbijmy to:

```
import UIKit
```

Struktura UIKit zapewnia wymaganą infrastrukturę dla aplikacji na iOS. Importujesz UIKit zamiast Foundation, ponieważ obsługa klasy UIImage nie jest dostępna w Foundation.

```
import CoreImage
```

Core Image to technologia przetwarzania i analizy obrazu, która zapewnia wysoką wydajność przetwarzania obrazów nieruchomych i wideo. Importujesz CoreImage, ponieważ jest on wymagany, aby uzyskać dostęp do wbudowanych filtrów zdjęć.

```
protocol ImageFiltering {
func apply(filter: String, originalImage:
UIImage) -> UIImage
}
```

Tutaj deklarujesz protokół o nazwie ImageFiltering. Protokół ten określa metodę Apply(filter:originalImage:), która przyjmuje jako parametry nazwę filtra i obraz.

```
extension ImageFiltering {
func apply(filter: String, originalImage:
UIImage) -> UIImage {
```

To rozszerzenie protokołu ImageFiltering zawiera implementację metody Apply(filter:originalImage:). Oznacza to, że każda klasa, która przyjmuje protokół ImageFiltering, będzie mogła wykonać tę metodę.

let initialCIImage = CIImage(image:

originalImage, options: nil)

Ta instrukcja konwertuje oryginalny obraz na instancję CIImage, dzięki czemu można do niej zastosować filtry, i przypisuje go do inicjałuCIImage.

let initialCIImage = CIImage(image:

originalImage, options: nil)

Ta instrukcja przechowuje oryginalną orientację obrazu w OriginalOrientation.

guard let ciFilter = CIFilter(name: filter)

else {

print("filter not found")

return originalImage

}

Ta instrukcja Guard pobiera filtr o tej samej nazwie co filter i przypisuje go do ciFilter i zwraca oryginalny obraz, jeśli filtr nie zostanie znaleziony.

ciFilter.setValue(initialCIImage, forKey:

kCIInputImageKey)

let context = CIContext()

let filteredCIImage =

(ciFilter.outputImage)!

Te instrukcje stosują wybrany filtr do initiaCIImage i przechowują wynik w filteredCIImage.

let filteredCGImage =

context.createCGImage(filteredCIImage, from:

filteredCIImage.extent)

return UIImage(cgImage: filteredCGImage!,

scale: 1.0, orientation:

originalOrientation)

Te instrukcje konwertują instancję CIImage przechowywaną w filteredCIImage z powrotem na instancję UIImage i zwraca ją.

To kończy implementację protokołu ImageFiltering i metody Apply(filter:originalImage:). W tym momencie masz co następuje:

• FilterData.plist, który zawiera dane filtrów zdjęć w Twojej aplikacji.

• FilterItem, klasa, która może przechowywać filtr i opis filtra.

• FilterDataManager, klasa menedżera danych, która ładuje dane z pliku FilterData.plist i generuje tablicę instancji FilterItem.

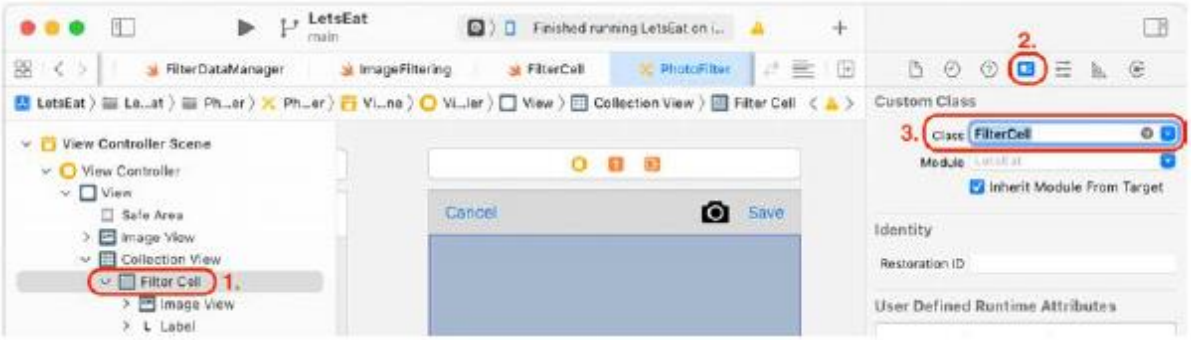
• ImageFiltering, protokół zawierający metodę Apply(filter:originalImage:), która stosuje filtr do obrazu.

W następnej sekcji utworzysz klasy dla elementów interfejsu użytkownika na ekranie Filtru zdjęć, co umożliwi zarządzanie tym ekranem i znajdującym się na nim widokiem kolekcji.

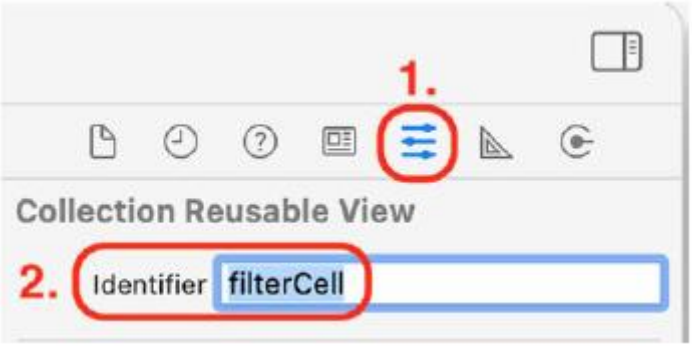
Tworzenie klas dla ekranu Filtru zdjęć

Do tej pory zaimportowałeś plik FilterData.plist do swojej aplikacji, utworzyłeś klasy FilterItem i FilterDataManager oraz utworzyłeś protokół ImageFiltering. W tej sekcji skonfigurujesz klasy dla ekranu Filtru zdjęć, co pozwoli Ci zarządzać tym ekranem i widokiem kolekcji na nim. Pamiętaj, że dodałeś plik scenorysu PhotoFilter do swojego projektu w Rozdziale 17, Pierwsze kroki z Mapkit. Zawiera scenę składającą się z dużego widoku obrazu, w którym będzie przechowywane zdjęcie wybrane przez użytkownika, oraz widoku kolekcji, w którym będą wyświetlane podglądy filtrów. Poniższy zrzut ekranu pokazuje, jak to będzie wyglądać po zakończeniu implementacji:

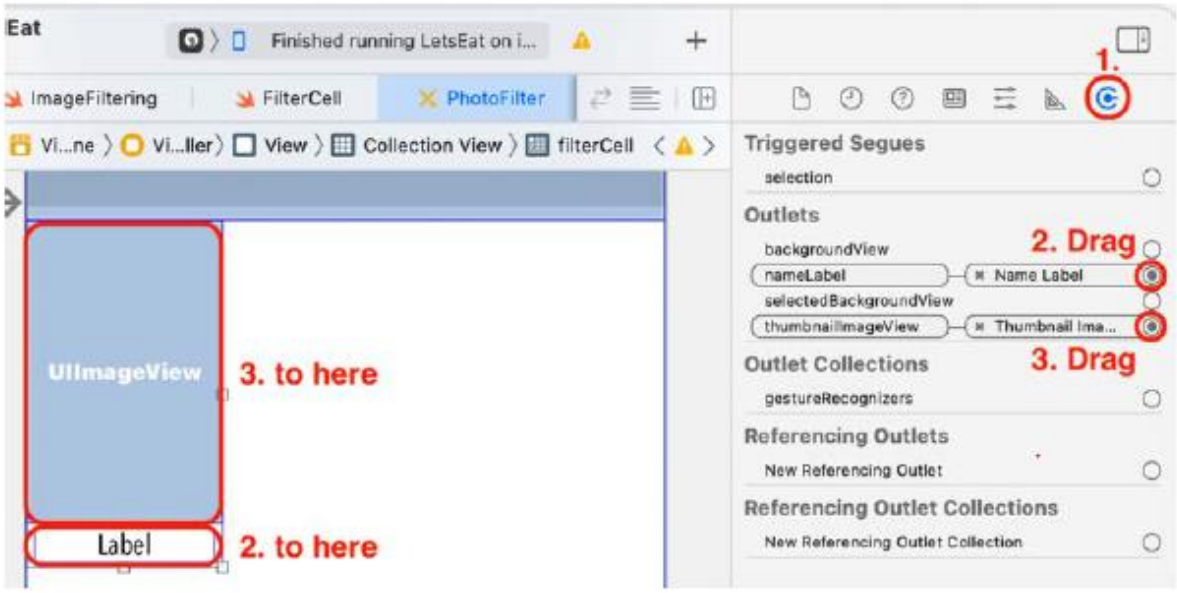
Ten ekran działa w następujący sposób. Kiedy dotkniesz przycisku Dodaj zdjęcie na ekranie Szczegóły restauracji i wybierzesz zdjęcie, pojawi się ekran Filtr zdjęć, pokazujący wybrane zdjęcie z przewijaną listą filtrów tuż pod nim. Każdy filtr na liście przewijanej jest wyświetlany w komórce widoku kolekcji. Kliknięcie filtra na przewijanej liście spowoduje zastosowanie wybranego filtra do zdjęcia. W następnej sekcji utworzysz i skonfigurujesz klasę do zarządzania komórkami widoku kolekcji. W każdej komórce zostanie wyświetlony podgląd miniatury zdjęcia po zastosowaniu filtra.



8. Kliknij przycisk Inspektora atrybutów. Ustaw identyfikator na filterCell:



9. Kliknij przycisk Inspektora połączeń. Połącz gniazda nameLabel i miniaturaImageView z odpowiadającymi im elementami interfejsu użytkownika, jak pokazano:



Zakończyłeś konfigurowanie komórek widoku kolekcji. W następnej sekcji utworzysz kontroler widoku dla ekranu Filtru zdjęć. Umożliwi to wybranie zdjęcia i wybranie filtra, który zostanie do niego zastosowany.

Tworzenie kontrolera widoku dla ekranu Filtru zdjęć

Jak dotąd utworzyłeś klasę FilterCell do zarządzania komórkami widoku kolekcji na ekranie Filtru zdjęć. Teraz utworzysz kontroler widoku, aby zarządzać zawartością tego ekranu. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch i kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: PhotoFilterViewController

Podklasa: UIViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. Plik PhotoFilterViewController pojawi się w nawigatorze projektu. Usuń cały skomentowany kod po metodzie viewDidLoad().

5. Dodaj do pliku poniższy kod, aby zadeklarować i zdefiniować klasę PhotoFilterViewController oraz jej właściwości:

```
import UIKit

import AVFoundation

class PhotoFilterViewController: UIViewController {

@IBOutlet var mainImageView: UIImageView!

@IBOutlet var collectionView: UICollectionView!

private let manager = FilterDataManager()

var selectedRestaurantID: Int?

private var mainImage: UIImage?

private var thumbnail: UIImage?

private var filters: [FilterItem] = []

override func viewDidLoad() {

super.viewDidLoad()

initialize()

}

}
```

Rozbijmy to:

```
import AV Foundation
```

Ta instrukcja importuje framework AVFoundation. Ten framework zawiera metody przechwytywania, przetwarzania, syntezy, kontrolowania, importowania i eksportowania multimediów audiowizualnych na platformach Apple.

class PhotoFilterViewController: UIViewController {

Ta instrukcja deklaruje klasę PhotoFilterViewController, podklasę klasy UIViewController.

@IBOutlet var mainImageView: UIImageView!

Jest to wyjście dla widoku obrazu, w którym zostanie wyświetlone wybrane przez użytkownika zdjęcie z zastosowanym filtrem.

@IBOutlet var collectionView: UICollectionView!

To jest miejsce na widok kolekcji, w którym będą wyświetlane miniatury podglądu każdego filtra.

private let manager = FilterDataManager()

Ta instrukcja przypisuje instancję klasy FilterDataManager do właściwości manager.

var selectedRestaurantID:Int?

Każda restauracja posiada unikalny identyfikator numeryczny.

private var mainImage: UIImage?

Ta właściwość służy do przechowywania tego identyfikatora. W następnym rozdziale zobaczysz, jak jest on używany podczas przechowywania zdjęć przy użyciu danych podstawowych.

private var thumbnail: UIImage?

Ta właściwość przechowuje zdjęcie wybrane przez użytkownika.

private var thumbnail: UIImage?

Ta właściwość przechowuje miniaturę zdjęcia wybranego przez użytkownika.

private var filters: [FilterItem] = []

Ta właściwość przechowuje tablicę instancji FilterItem dostarczonych przez menedżera.

override func viewDidLoad() {

super.viewDidLoad()

initialize()

}

Ta metoda wywołuje metodę inicjalizacji(), gdy instancja PhotoFilterViewController ładuje swój widok. Pamiętaj, że spowoduje to wygenerowanie błędu, ponieważ funkcja inicjowania() nie została jeszcze zaimplementowana.

6. Podobnie jak poprzednio, do uporządkowania kodu użyjesz rozszerzeń. Dodaj następujące rozszerzenie prywatne zawierające metodę inicjalizacji() po zamykającym nawiasie klamrowym:

// MARK: - Private Extension

private extension PhotoFilterViewController {

func initialize() {

setupCollectionView()

```
checkSource()
}
}
```

To rozszerzenie zawiera implementację metody inicjalizacji(), która wywołuje dwie inne metody. setupCollectionView() konfiguruje widok kolekcji używany do wyświetlania listy filtrów. checkSource() sprawdza status autoryzacji użytkownika do korzystania z kamery. Notatka

że będą one generować błędy, ponieważ nie zostały jeszcze wdrożone. Metody te zaimplementujesz w następnym kroku.

7. Zaimplementuj metody setupCollectionView() i checkSource() w rozszerzeniu prywatnym po metodzie inicjalizacji():

```
func setupCollectionView() {
let layout = UICollectionViewFlowLayout()
layout.scrollDirection = .horizontal
layout.sectionInset = UIEdgeInsets(top: 7,
left: 7, bottom: 7, right: 7)
layout.minimumInteritemSpacing = 0
layout.minimumLineSpacing = 7
collectionView.collectionViewLayout = layout
collectionView.dataSource = self
collectionView.delegate = self
}
func checkSource() {
let cameraMediaType = AVMediaType.video
let cameraAuthorizationStatus =
AVCaptureDevice.authorizationStatus(for:
cameraMediaType)
switch cameraAuthorizationStatus {
case .notDetermined:
AVCaptureDevice.requestAccess(for:
cameraMediaType) { granted in
if granted {
DispatchQueue.main.async {
```

self.showCameraUserInterface()

}

}

}

case .authorized:

self.showCameraUserInterface()

default:

break

}

}

Rozbijmy to:

setupCollectionView()

Spowoduje to ustawienie widoku kolekcji używanego do wyświetlania miniaturowych podglądów filtrów. Tutaj tworzysz instancję UICollectionViewFlowLayout, ustawiasz kierunek przewijania, wstawki sekcji, odstępy między elementami i odstępy między wierszami, a następnie przypisujesz je do widoku kolekcji. Następnie ustawiasz klasę PhotoFilterViewController jako delegata i źródło danych dla tego widoku kolekcji. Pamiętaj, że ustawiasz delegata i źródło danych programowo, a nie używając scenorysu; każde podejście jest akceptowalne. Nie martw się błędami, pojawiają się, ponieważ nie zaadoptowałeś jeszcze protokołów UICollectionViewDataSource i UICollectionViewDelegate dla tej klasy. Naprawisz to później.

checkSource()

Służy do sprawdzania stanu autoryzacji użytkownika w zakresie korzystania z kamery. Możliwe przypadki są następujące:

.notDetermined oznacza, że użytkownik nie został poproszony o dostęp do kamery.

.authorized oznacza, że użytkownik udzielił wcześniej dostępu do kamery.

.restricted oznacza, że użytkownik nie może uzyskać dostępu ze względu na ograniczenia ustawione na urządzeniu.

.denied oznacza, że użytkownik wcześniej odmówił aplikacji dostępu do kamery.

Jeśli status to .notDetermined, aplikacja zapyta użytkownika o pozwolenie, a jeśli zostanie ono wydane, zostanie wywołana metoda showCameraUserInterface(). Jeśli status to .authorized, wywoływana jest metoda showCameraUserInterface(). Należy pamiętać, że spowoduje to wygenerowanie błędu, ponieważ funkcja showCameraUserInterface() nie została jeszcze zaimplementowana. Jeśli status to .restricted lub .denied, mieści się on w domyślnym przypadku: case i metoda kończy działanie.

8. Wymaganych jest jeszcze kilka metod pomocniczych. Dodaj następujący kod do rozszerzenia prywatnego, aby je zaimplementować po metodzie checkSource():

func showApplyFilterInterface() {

```
filters = manager.fetch()

if let mainImage = self.mainImage {

mainImageView.image = mainImage

collectionView.reloadData()

}

}

@IBAction func onPhotoTapped(_ sender: Any) {

checkSource()

}
```

Rozbijmy to:

showApplyFilterInterface()

Metoda ta zostanie wywołana po wybraniu przez użytkownika zdjęcia z aparatu lub biblioteki zdjęć. Wywołuje metodę fetch() instancji FilterManager, która ładuje plik FilterData.plist i umieszcza jego zawartość w tablicy instancji FilterItem. Tablica ta jest następnie przypisywana do właściwości filter instancji PhotoFilterViewController, która później zostanie użyta do wypełnienia widoku kolekcji miniaturowymi podglądami filtrów. Następna instrukcja przypisuje właściwość mainImage instancji PhotoFilterViewController do mainImageView, który jest punktem wyjścia dla widoku obrazu znajdującego się nad widokiem kolekcji, jeśli ustawiono mainImage. Ostatnia instrukcja informuje widok kolekcji, aby się przerysował.

onPhotoTapped()

Metoda ta wywołuje zaimplementowaną wcześniej metodę checkSource(), która wywołuje metodę showCameraUserInterface() w przypadku udzielenia autoryzacji. Przypiszesz to do przycisku aparatu w scenie kontrolera widoku filtra zdjęć później.

Apple zastrzega, że aplikacje korzystające z aparatu lub biblioteki zdjęć muszą wcześniej poprosić użytkownika o pozwolenie. Później zmodyfikujesz aplikację tak, aby wyświetlała okno dialogowe z prośbą o pozwolenie na korzystanie z aparatu lub biblioteki zdjęć, implementując klucze NSCameraUsageDescription i NSMicrophoneUsageDescription w pliku Info.plist.

9. Zastosujesz protokół UICollectionViewDataSource i zaimplementujesz wymagane metody, aby widok kolekcji wyświetlał miniaturowe podglądy filtrów. Dodaj nowe rozszerzenie po rozszerzeniu prywatnym i zaimplementuj je w następujący sposób:

```
extension PhotoFilterViewController:

UICollectionViewDataSource {

func collectionView(_ collectionView:

UICollectionView, numberOfItemsInSection

section: Int) -> Int {
```

```
filters.count
}
func collectionView(_ collectionView:
UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
let cell = collectionView
.dequeueReusableCell
(withReuseIdentifier: "filterCell",
for: indexPath) as! FilterCell
let filterItem = filters[indexPath.row]
if let thumbnail = thumbnail {
cell.set(filterItem: filterItem,
imageForThumbnail: thumbnail)
}
return cell
}
}
```

Poniższe informacje powinny być Ci znane, ponieważ robiłeś to już wcześniej, ale przeanalizujmy to jeszcze raz:

kolekcjaView(_:numberOfItemsInSection:)

Określa to liczbę elementów, które ma wyświetlać widok kolekcji, która jest taka sama jak liczba elementów FilterItem w tablicy filtrów instancji PhotoFilterViewController.

kolekcjaView(_:cellForItemAt:)

To określa, co umieścić w każdej komórce. Tutaj otrzymujesz instancję FilterItem odpowiadającą pozycji komórki w widoku kolekcji i przekazujesz ją wraz z właściwością miniatury instancji PhotoFilterViewController do metody set(filterItem:imageForThumbnail:) , która ustawia obraz i etykietę dla komórki widoku kolekcji .

10. Wcześniej skonfigurowałeś widok kolekcji przy użyciu instancji UICollectionViewFlowLayout. Teraz ustawisz rozmiar komórek widoku kolekcji. Dodaj następujące rozszerzenie po rozszerzeniu zawierającym metody źródła danych:

```
rozszerzenie PhotoFilterViewController:
UICollectionViewDelegateFlowLayout {
func kolekcjaView(_ kolekcjaView:
UICollectionView, układ
```

kolekcjaViewUkład:

UICollectionViewLayout, sizeForItemAt

IndexPath: IndexPath) -> CGSize {

niech kolekcjaViewHeight =

kolekcjaWidok.rozmiar.ramki.wysokość

niech topInset = 14,0

niech cellHeight = kolekcjaViewHeight -

topWstawka

zwróć CGSize(szerokość: 150, wysokość:

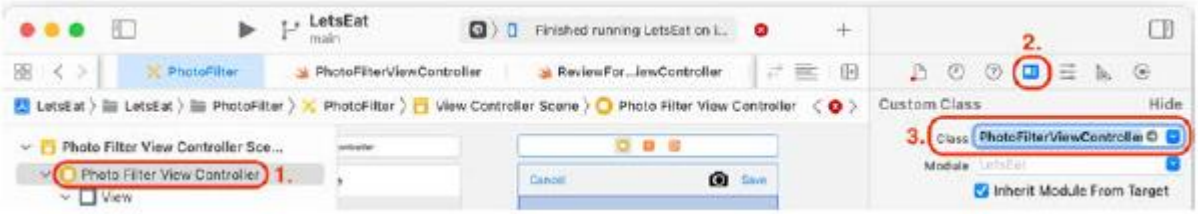
wysokość komórki)

}

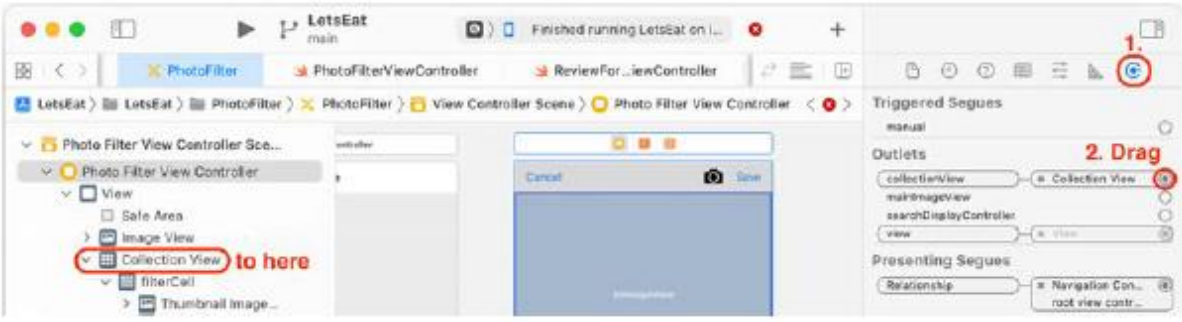
}

kolekcjaView(_:layout:sizeForItemAt:) zwraca rozmiar, jaki powinna mieć każda komórka widoku kolekcji. Najpierw wysokość widoku kolekcji jest przypisywana do kolekcjiViewHeight. Następnie wartość topInset zostaje ustalona na 14,0 punktów. Wysokość komórki widoku kolekcji jest obliczana poprzez odjęcie topInset od kolekcjiViewHeight. Powoduje to 14-punktową przerwę między górą komórek widoku kolekcji a górą widoku kolekcji. Na koniec instancja CGSize z szerokością ustawioną na 150 punktów i wysokością ustawioną na cellHeight jest zwracana jako rozmiar komórki widoku kolekcji. Poprzednio robiło się to za pomocą Inspektora rozmiaru; teraz robisz to programowo. Teraz połączysz wyjścia i akcje w tej klasie z elementami interfejsu użytkownika w pliku scenorysu PhotoFilter. kolekcjaView to punkt wyjścia dla widoku kolekcji, który wyświetla listę filtrów. mainImageView służy do widoku obrazu tuż nad nim, który pokazuje obraz wybrany przez użytkownika. onPhotoTapped() jest przyciskiem aparatu na pasku nawigacyjnym. Skonfigurujesz także przycisk Anuluj, aby zamknąć ekran Filtru zdjęć. Wykonaj następujące kroki:

1. Kliknij plik scenorysu PhotoFilter w nawigatorze projektu. Wybierz ikonę kontrolera widoku sceny kontrolera widoku w konspekcie dokumentu. Kliknij przycisk Inspektora tożsamości. W obszarze Klasa niestandardowa ustaw klasę na PhotoFilterViewController:



2. Wybierz Inspektora połączeń. Kliknij i przeciągnij z gniazda CollectionView do widoku kolekcji w konspekcie dokumentu:

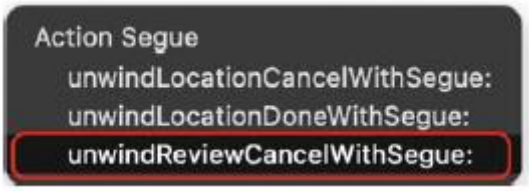


3. Kliknij i przeciągnij z głównego wyjścia ImageView do widoku obrazu w konspekcie dokumentu:

4. Kliknij i przeciągnij z akcji onPhotoTapped: na przycisk aparatu:

5. Przycisk Anuluj służy do wyjścia z tego ekranu, jeśli użytkownik nie chce dokonywać wyboru. Połączysz przycisk Anuluj z metodą rozwijania zaimplementowaną w poprzednim rozdziale, co spowoduje zamknięcie tego ekranu i powrót użytkownika do ekranu Szczegóły restauracji. Ctrl + Przeciągnij z przycisku Anuluj do ikony Wyjdź w Doku Sceny:

6. Wybierz opcję unwindReviewCancelWithSegue: z wyskakującego menu:



Wszystkie wyjścia i akcje klasy PhotoFilterViewController zostały połączone. Następnie zaimplementujesz następujące metody:

• showCameraUserInterface(), metoda wyświetlająca widok z kamery urządzenia lub biblioteki zdjęć w interfejsie selektora obrazów.

• Dwie metody protokołu UIImagePickerControllerDelegate, które zostaną wywołane po wybraniu obrazu w interfejsie selektora obrazów lub kliknięciu przycisku Anuluj.

Aby zaimplementować metody showCameraUserInterface() i UIImagePickerControllerDelegate, kliknij plik PhotoFilterViewController w nawigatorze projektu i dodaj następujące rozszerzenie po rozszerzeniu UICollectionViewDelegateFlowLayout:

```
extension PhotoFilterViewController: UIImagePickerControllerDelegate,

UINavigationControllerDelegate {

func showCameraUserInterface() {

let imagePicker = UIImagePickerController()

imagePicker.delegate = self

#if targetEnvironment(simulator)

imagePicker.sourceType =

UIImagePickerController.SourceType.photoLibrary

#else

imagePicker.sourceType =

UIImagePickerController.SourceType.camera

imagePicker.showsCameraControls = true

#endif

imagePicker.mediaTypes = ["public.image"]

imagePicker.allowsEditing = true

self.present(imagePicker, animated: true,

completion: nil)

}

func imagePickerControllerDidCancel(_ picker:

UIImagePickerController) {

picker.dismiss(animated: true, completion: nil)

}

func imagePickerController(_ picker:
```

```
UIImagePickerController,
didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
if let selectedImage =
info[UIImagePickerController.InfoKey
.editedImage] as? UIImage {
self.thumbnail =
selectedImage.preparingThumbnail(of:
CGSize(width: 100, height: 100))
let mainImageViewSize =
mainImageView.frame.size
self.mainImage =
selectedImage.preparingThumbnail(of:
mainImageViewSize)
}
picker.dismiss(animated: true) {
self.showApplyFilterInterface()
}
}
}
```

Porozmawiajmy najpierw o showCameraUserInterface(). Ta metoda jest uruchamiana po dotknięciu przycisku aparatu, co powoduje wyświetlenie selektora obrazów na ekranie. Ten selektor obrazów to standardowy selektor obrazów w systemie iOS, który pojawia się, gdy chcesz użyć obrazu — na przykład, aby dodać obraz do postu na Facebooku lub tweeta.

Rozbijmy to:

```
let imagePicker = UIImagePickerController()
```

Spowoduje to utworzenie instancji klasy UIImagePickerController i przypisanie jej do imagePicker.

```
imagePicker.delegate = self
```

Spowoduje to ustawienie właściwości delegata instancji imagePicker na instancję PhotoFilterViewController.

```
#if targetEnvironment(simulator)
imagePicker.sourceType =
```

UIImagePickerController.SourceType.photoLibrary

#else

imagePicker.sourceType =

UIImagePickerController.SourceType.camera

imagePicker.showsCameraControls = true

#endif

Ten blok kodu nazywany jest blokiem kompilacji warunkowej. Zaczyna się od dyrektywy kompilacji #if i kończy się dyrektywą kompilacji #endif. Jeśli używasz symulatora, tylko

Kompilowana jest instrukcja ustawiająca właściwość sourceType instancji imagePicker na bibliotekę zdjęć. Jeśli używasz rzeczywistego urządzenia, kompilowane są instrukcje ustawiające właściwość sourceType instancji imagePicker dla kamery i wyświetlające elementy sterujące kamery.

imagePicker.mediaTypes = ["public.image"]

Ustawia interfejs aparatu do przechwytywania zdjęć.

imagePicker.mediaTypes = ["public.image"]

Sets the camera interface to capture still images.

imagePicker.allowsEditing = true

Indicates the user is allowed to edit the selected image.

self.present(imagePicker, animated: true, completion: nil)

Wyświetla imagePicker na ekranie.

Gdy na ekranie pojawi się selektor obrazów, możesz wybrać zdjęcie lub anulować. Jeśli anulujesz, zostanie wyzwolony imagePickerControllerDidCancel(_:) i selektor obrazów zostanie odrzucony. Jeśli wybierzesz zdjęcie, zostanie uruchomiony imagePickerController(_:didFinishPickingMediaWithInfo:) a zdjęcie zostanie zwrócone i przypisane do wybranego Obrazu. Następnie metodaprzygotowującaThumbnail(of:) instancji wybranejImage zostanie użyta do utworzenia małego obrazka o szerokości i wysokości 100 punktów. Zostanie to następnie przypisane do właściwości miniatury. Następnie z wybranego obrazu zostanie utworzony obraz o tym samym rozmiarze co mainImageView przy użyciu metody przygotowanieThumbnail(of:) . Zostanie to przypisane do właściwości mainImage, a selektor obrazów zostanie odrzucony.

Następnie zaimplementujesz filterMainImage(filterItem:), metodę zastosowania filtra do obrazu w mainImageView. Dodaj rozszerzenie zawierające tę metodę po UIImagePickerControllerDelegate

extension PhotoFilterViewController: ImageFiltering {

func filterMainImage(filterItem: FilterItem) {

if let mainImage = mainImage, let filter =

filterItem.filter {

if filter != "None" {

```
mainImageView.image =

self.apply(filter: filter,

originalImage: mainImage)

} else {

mainImageView.image = mainImage

}

}

}

}
```

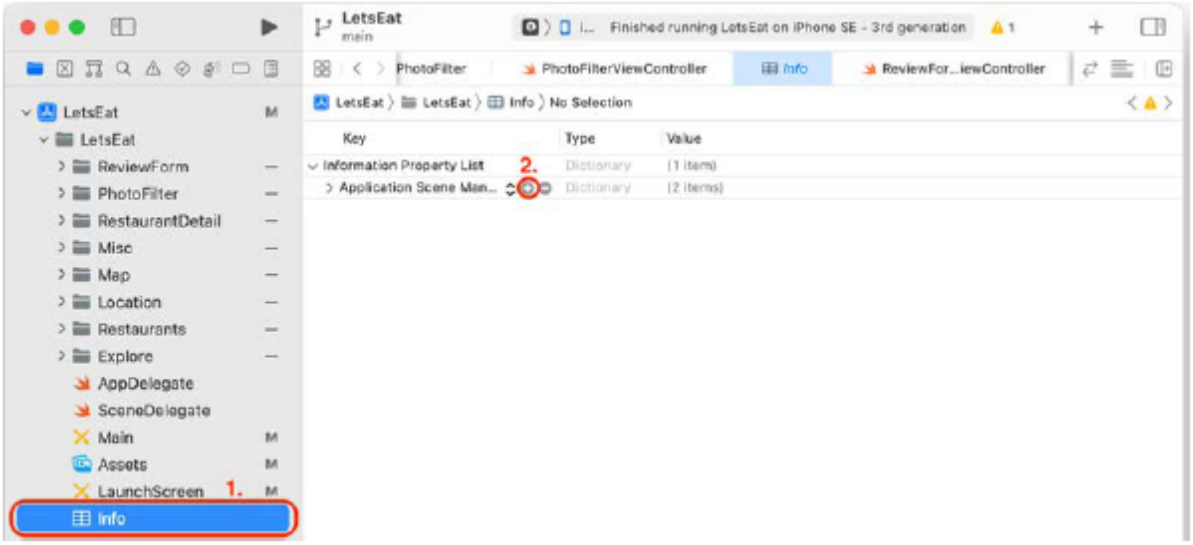
To sprawia, że klasa PhotoFilterViewController przyjmuje protokół ImageFiltering. Pamiętaj, że każda klasa, która przyjmuje ten protokół, otrzymuje metodę Apply(filter:originalImage:). Metoda filterMainImage(filterItem:) wykorzystuje tę metodę do zastosowania wybranego filtru do zdjęcia przechowywanego we właściwości mainImage instancji PhotoFilterViewController, a wynik jest przypisywany do wyjścia mainImageView, dzięki czemu jest widoczny na ekranie. Jeśli wybrałeś filtr Brak, mainImage zostanie przypisany do wyjścia mainImageView. Nadal musisz wiedzieć, który filtr wybrał użytkownik, więc sprawisz, że klasa PhotoFilterViewController przyjmie protokół UICollectionViewDelegate i zaimplementujesz metodę identyfikującą, która komórka w widoku kolekcji została dotknięta. Dodaj następujące rozszerzenie zawierające tę metodę po rozszerzeniu ImageFiltering:

```
extension PhotoFilterViewController:

UICollectionViewDelegate {

func collectionView(_ collectionView:

UICollectionView, didSelectItemAt

indexPath: IndexPath) {

let filterItem = self.filters[indexPath.row]

filterMainImage(filterItem: filterItem)

}

}
```

Metoda CollectionView(_:didSelectItemAt:) jest wywoływana za każdym razem, gdy użytkownik dotknie komórki w widoku kolekcji. Element FilterItem odpowiadający dotkniętej komórce jest następnie przekazywany do filterMainImage(filterItem:). Implementacja klasy PhotoFilterViewController została już ukończona, ale pamiętaj o tym ,ŻE musisz poprosić o pozwolenie na użycie aparatu lub dostęp do biblioteki zdjęć. Zmodyfikujesz informacje. plist w swoim projekcie, aby komunikaty były wyświetlane użytkownikowi, gdy aplikacja spróbuje tego dokonać aby uzyskać dostęp do aparatu lub biblioteki zdjęć. Uzyskiwanie pozwolenia na korzystanie z aparatu lub biblioteki zdjęć Jak wspomniano wcześniej, firma Apple wymaga, aby aplikacja informowała użytkownika, jeśli chce uzyskać dostęp do aparatu lub biblioteki zdjęć. Jeśli tego nie zrobisz, Twoja aplikacja zostanie

odrzucona i nie będzie dostępna w App Store. Zmodyfikujesz plik Info.plist w swoim projekcie, aby aplikacja wyświetlała komunikaty przy próbie uzyskania dostępu do aparatu lub biblioteki zdjęć. Wykonaj następujące kroki:

1. Kliknij plik Info.plist w nawigatorze projektu, aby wyświetlić listę kluczy. Przesuń wskaźnik myszy nad dowolny istniejący klawisz i kliknij przycisk +:



2. Powinno pojawić się pole umożliwiające wprowadzenie dodatkowego klucza:

3. Wprowadź następujące klucze:

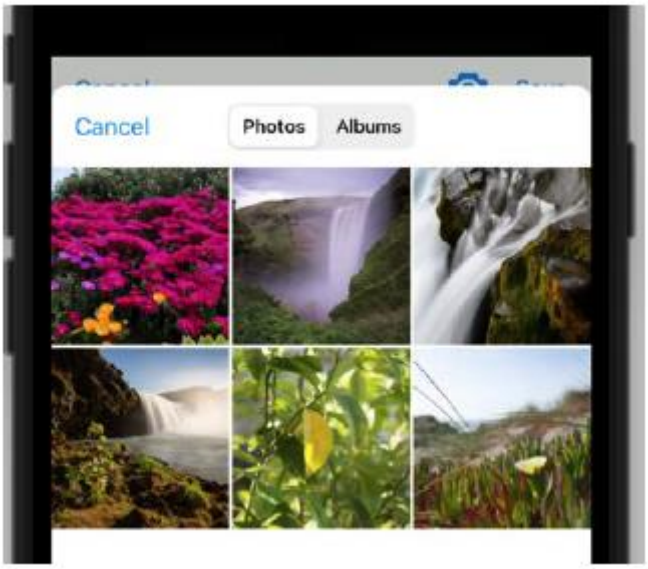
NSPhotoLibraryUsageDescription

NSCameraUsageDescription

4. Dla wartości każdego klucza wprowadź ciąg wyjaśniający użytkownikowi, dlaczego chcesz korzystać z aparatu lub biblioteki zdjęć:

Zbuduj i uruchom projekt. Przejdź do ekranu Szczegóły restauracji i dotknij przycisku Dodaj zdjęcie. Powinieneś zobaczyć następujący alert:

Kliknij OK. Pojawi się selektor obrazów:



Wybierz zdjęcie, a ekran Filtr zdjęć wyświetli zdjęcie i listę miniatur z zastosowanymi różnymi filtrami. Dotknięcie filtra spowoduje zastosowanie jego efektu do zdjęcia:

Zmodyfikowałeś plik info.plist w swoim projekcie i Twoja aplikacja prosi teraz o pozwolenie przed użyciem aparatu lub biblioteki zdjęć. Możesz użyć przycisku Anuluj, aby zamknąć ekran Filtru zdjęć i

wróć do ekranu szczegółów restauracji. Nie możesz jednak jeszcze używać przycisku Zapisz; jego funkcjonalność zaimplementujesz w następnym rozdziale.

Streszczenie

Zakończyłeś implementację ekranu Filtru zdjęć. Zaimportowałeś FilterData. plist, plik .plist zawierający filtry, których chcesz użyć, utworzył klasę FilterItem do przechowywania danych filtrów i utworzył klasę menedżera danych FilterManager w celu odczytania pliku .plist i zapełnienia tablicy instancji FilterItem. Następnie utworzono protokół ImageFiltering z metodą stosowania filtrów do obrazów. Następnie utworzyłeś klasy FilterCell i PhotoFilterViewController w celu zarządzania komórkami widoku kolekcji i ekranem Photo Filter. Następnie dostosowałeś klasę PhotoFilterViewController do przyjęcia protokołu UIImagePickerDelegate i dodałeś metody, dzięki którym możesz używać zdjęć z aparatu lub biblioteki zdjęć w swojej aplikacji. Na koniec dodano kod do PhotoFilterViewController, aby zastosować wybrany filtr do obrazu. Możesz teraz pisać własne aplikacje, które importują zdjęcia z aparatu lub biblioteki zdjęć i stosują do nich filtry. Należy pamiętać, że wybranego zdjęcia nie można zapisać. W następnym rozdziale dowiesz się, jak zapisywać recenzje i zdjęcia przy użyciu danych podstawowych, aby pojawiły się ponownie po zamknięciu i ponownym uruchomieniu aplikacji.

Zrozumienie podstawowych danych

Twoja aplikacja jest prawie gotowa! Każdy ekran działa zgodnie z prezentacją aplikacji. Jednakże jest jeszcze jedna rzecz, którą musisz zrobić. Zaimplementowałeś ekran Formularz recenzji, który umożliwia wprowadzenie recenzji dla konkretnej restauracji. Zaimplementowałeś ekran Filtr zdjęć, który pozwala pobrać zdjęcie z aparatu lub biblioteki zdjęć i dodać do niego filtr. Jednak obecnie nie ma możliwości zapisania recenzji ani zdjęć, ponieważ zostaną one utracone po zamknięciu aplikacji. Tu użyjesz danych podstawowych do zapisywania recenzji i zdjęć w swojej aplikacji. Najpierw dowiesz się o danych podstawowych i ich różnych komponentach. Następnie utworzysz model danych dla recenzji i zdjęć oraz utworzysz odpowiednie obiekty modelu dla swojej aplikacji. Następnie skonfigurujesz komponenty Core Data dla swojej aplikacji. Dowiesz się wówczas o mechanizmie służącym do zapisywania recenzji i zdjęć konkretnej restauracji za pomocą identyfikatora restauracji. Następnie zaktualizujesz klasy ReviewFormViewController i PhotoFilterViewController, aby zapisywać recenzje i zdjęcia dla konkretnej restauracji, oraz zmodyfikujesz klasę RestaurantDetailViewController, aby ładować i wyświetlać recenzje dla konkretnej restauracji. Obliczysz także i wyświetlisz ogólną ocenę tej restauracji. Na koniec samodzielnie zmodyfikujesz klasę RestaurantDetailViewController, aby ładować i wyświetlać zdjęcia dla konkretnej restauracji. Pod koniec tego rozdziału zrozumiesz, jak działają dane podstawowe. Będziesz także mieć możliwość skonfigurowania komponentów Core Data i umożliwienia interfejsu między aplikacją a komponentami Core Data za pomocą klasy menedżera danych. Nauczysz się także zapisywać i ładować recenzje i zdjęcia za pomocą Core Data, które będziesz wtedy mógł wdrożyć we własnych aplikacjach.

Przedstawiamy podstawowe dane

Core Data to mechanizm Apple służący do zapisywania danych aplikacji na Twoim urządzeniu. Zapewnia trwałość, cofanie/ponawianie, zadania w tle, synchronizację widoków, wersjonowanie i migrację. Możesz zdefiniować typy danych i relacje za pomocą edytora modeli danych Xcode, a Core Data automatycznie wygeneruje definicje klas dla Twoich typów danych. Core Data może następnie tworzyć instancje obiektów i zarządzać nimi w oparciu o definicje klas.

Core Data udostępnia zestaw klas zwanych zbiorczo stosem Core Data do zarządzania instancjami obiektów, które są następujące:

• NSManagedObjectModel

Opisuje typy aplikacji, w tym ich właściwości i relacje

• Obiekt NSManaged

Klasa używana do implementowania instancji typów aplikacji na podstawie danych z NSManagedObjectModel.

• NSManagedObjectContext

Śledzi zmiany w wystąpieniach typów aplikacji.

• Koordynator NSPersistentStore

Zapisuje i pobiera wystąpienia typów Twojej aplikacji ze sklepów. Core Data udostępnia zestaw klas zwanych zbiorczo stosem Core Data do zarządzania instancjami obiektów, które są następujące:

• NSManagedObjectModel

Opisuje typy aplikacji, w tym ich właściwości i relacje

• Obiekt NSManaged

Klasa używana do implementowania instancji typów aplikacji na podstawie danych z NSManagedObjectModel.

• NSManagedObjectContext

Śledzi zmiany w wystąpieniach typów aplikacji.

• Koordynator NSPersistentStore

Zapisuje i pobiera wystąpienia typów Twojej aplikacji ze sklepów.

W następnej sekcji zaimplementujesz komponenty Core Data wymagane dla Twojej aplikacji do zapisywania recenzji i zdjęć.

Implementowanie komponentów Core Data dla Twojej aplikacji

Zanim zaimplementujesz komponenty Core Data w swojej aplikacji, zastanówmy się, co musisz zrobić, aby zapisywać recenzje lub zdjęcia. Wyobraź sobie, że zapisujesz recenzję lub zdjęcie w programie Microsoft Word. Najpierw tworzysz nowy szablon dokumentu Word z odpowiednimi polami na recenzję lub zdjęcie. Następnie na podstawie szablonów tworzysz nowe dokumenty Word i uzupełniasz dane. Wprowadzasz niezbędne zmiany, na przykład zmieniając tekst recenzji lub zmieniając efekt, jaki chcesz zastosować do zdjęcia. W tym momencie plik nie został jeszcze zapisany. Kiedy będziesz zadowolony ze swojego dokumentu, zapiszesz go na dysku twardym swojego komputera. Następnym razem, gdy będziesz chciał obejrzeć swoją recenzję lub zdjęcie, wyszukaj na dysku twardym odpowiedni dokument i kliknij go dwukrotnie, aby otworzyć go w programie Word i zobaczyć go jeszcze raz. Teraz, gdy masz już pomysł, co musisz zrobić, przyjrzyjmy się krokom wymaganym do wdrożenia tego. Najpierw musisz utworzyć model danych dla recenzji lub zdjęcia. Robisz to, tworząc encje w edytorze modeli danych Xcode, które przypominają szablony Microsoft Word. Jednostki mogą mieć atrybuty, które działają jak pola w szablonach Microsoft Word. Xcode może następnie utworzyć klasę NSManagedObjectModel na podstawie tego modelu danych. Core Data użyje następnie tej klasy NSManagedObjectModel do utworzenia instancji NSManagedObject, podobnie jak szablony Microsoft Word używane do tworzenia plików Microsoft Word. Te instancje NSManagedObject są umieszczane w instancji SManagedObjectContext, do której aplikacja ma do nich dostęp, podobnie jak otwieranie plików Microsoft Word w programie Microsoft Word. Następnie, gdy wyświetlisz ekran Formularz recenzji lub filtr zdjęć, szczegóły recenzji lub zdjęcie z filtrem zostaną zapisane w instancjach NSManagedObject i będziesz mógł je dowolnie modyfikować, podobnie jak w przypadku dokumentów Microsoft Word edytowane w programie Microsoft Word. Kiedy skończysz z recenzją lub zdjęciem, instancje NSManagedObject w instancji NSManagedObjectContext zostaną zapisane w pliku na Twoim urządzeniu iOS, zwanym magazynem trwałym. Przypomina to zapisywanie dokumentów programu Word na dysku twardym, gdy już z nimi skończysz. Instancja NSPersistentStoreCoordinator zarządza przepływem informacji między magazynem trwałym a instancją NSManagedObjectContext. Klasy NSPersistentContainer użyjesz do tworzenia instancji SManagedObjectModel, NSManagedObjectContext i NSPersistentStoreCoordinator dla swojej aplikacji. W następnej sekcji utworzysz encje i atrybuty reprezentujące recenzję lub zdjęcie za pomocą edytora modelu danych Xcode.

Tworzenie modelu danych

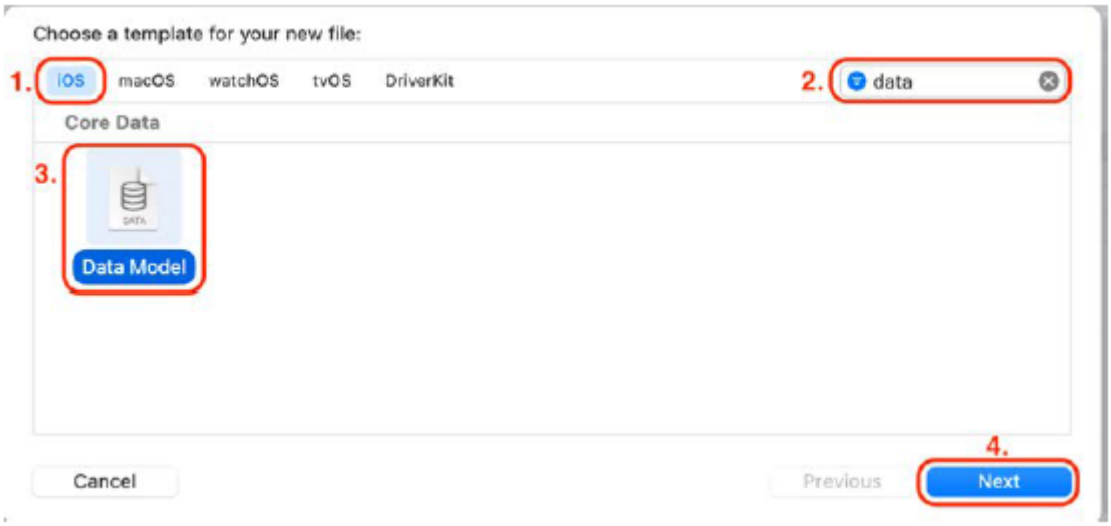
Obecnie, gdy naciśniesz Zapisz na ekranie Formularza recenzji, dane wprowadzone w polach zostaną po prostu wydrukowane w obszarze Debugowanie, a naciśnięcie Zapisz na ekranie Filtru zdjęć nic nie

daje. Pierwszym krokiem jest utworzenie definicji klas dla obiektów do przechowywania danych z ekranu Formularza recenzji i zdjęcia z ekranu Filtru zdjęć. Będziesz tworzyć encje dla recenzji i zdjęć za pomocą edytora modeli danych Xcode, a Xcode automatycznie wygeneruje definicje klas. Najpierw utwórzmy encję dla recenzji. Wykonaj następujące kroki:

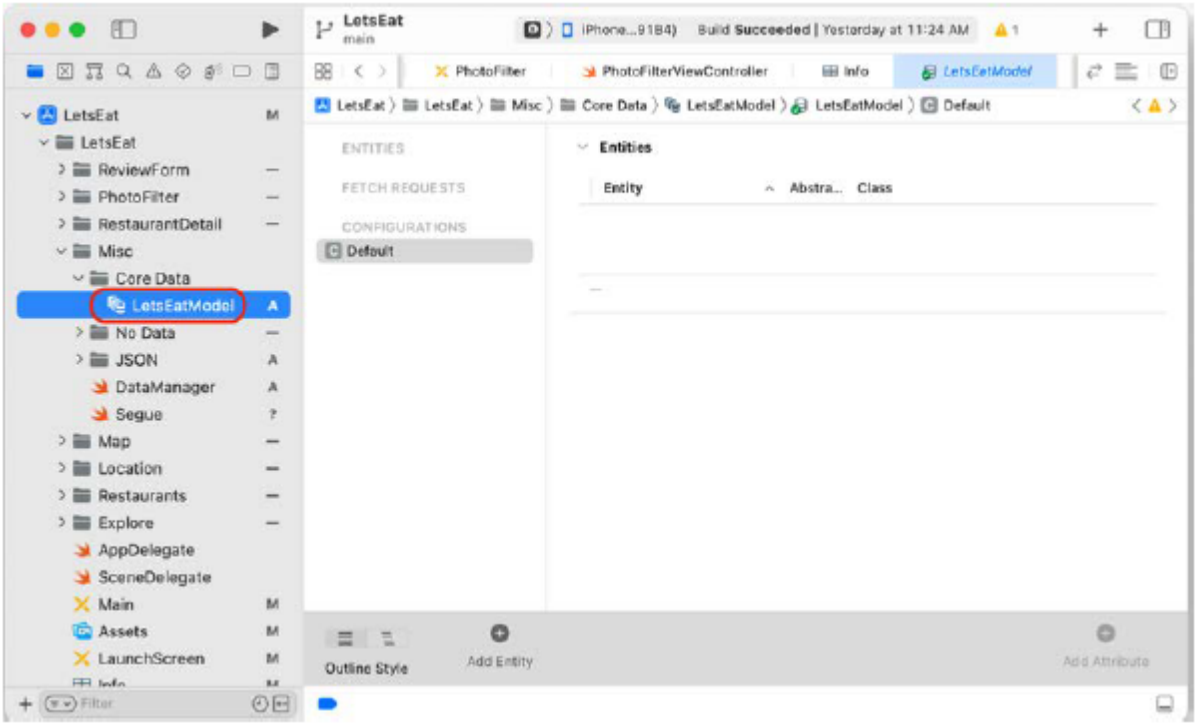
1. Kliknij prawym przyciskiem myszy folder Misc w nawigatorze projektu i wybierz opcję Nowa grupa. Nazwij tę grupę Dane podstawowe.

2. Kliknij prawym przyciskiem myszy grupę Dane podstawowe i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wpisz dane w polu filtru i wybierz Model danych. Kliknij Następny:

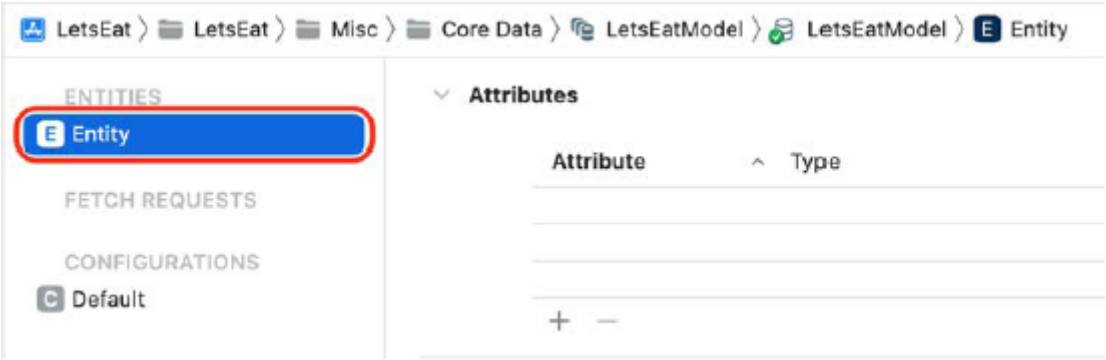


4. Nazwij plik LetsEatModel i kliknij Utwórz. Edytor modelu danych pojawi się w obszarze Edytor:

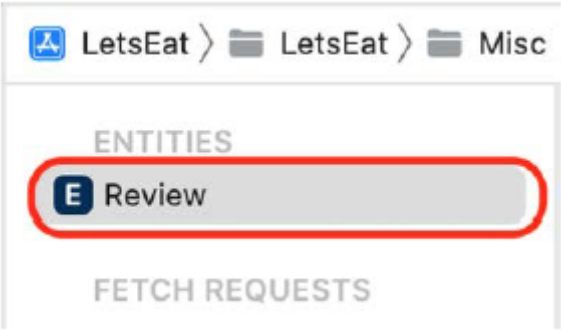


5. Kliknij przycisk Dodaj encję:

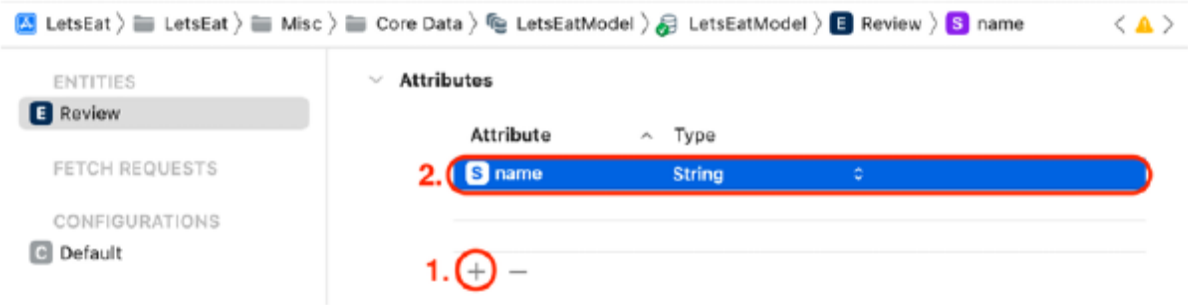
6. W sekcji ENTITIES pojawi się Encja. Atrybuty tej encji pojawiają się po prawej stronie encji:



7. Kliknij dwukrotnie Element i zmień jego nazwę. Review:



8. Kliknij przycisk + w sekcji Atrybuty, aby utworzyć atrybut. Ustaw atrybut na nazwę i typ na ciąg:



9. Musisz utworzyć atrybut dla każdego pola na ekranie Formularza recenzji, a także zapisać identyfikator restauracji, aby powiązać recenzję z restauracją. Dodaj następujące atrybuty i typy:

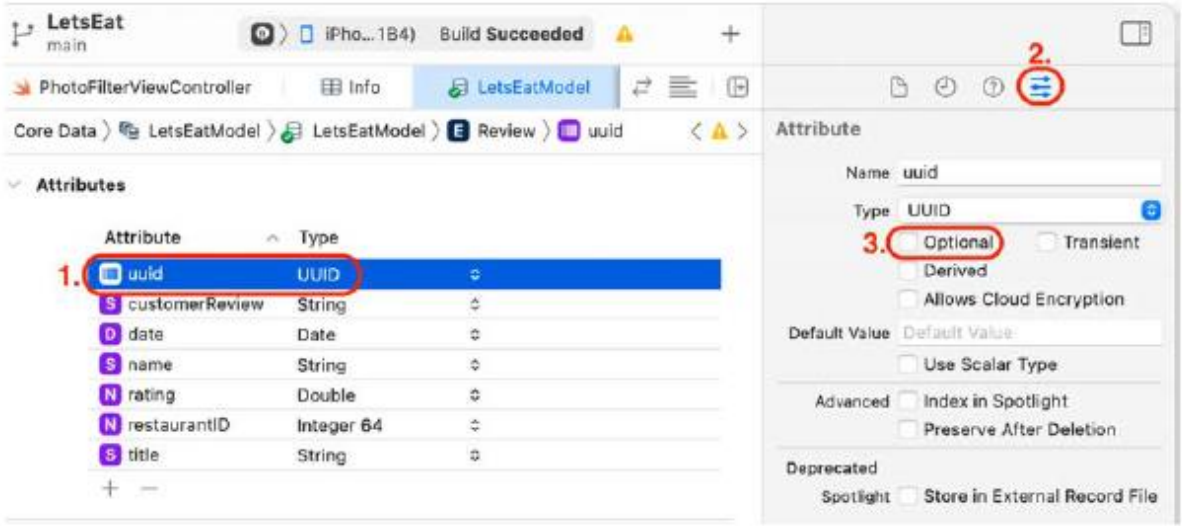
Attribute	Type
customerReview	String
date	Date
rating	Double
restaurantID	Integer 64
title	String

data zostanie automatycznie ustawiona po utworzeniu instancji Review.

10. Po zakończeniu sprawdź, czy atrybuty encji recenzji wyglądają następująco:

Attribute	Type
customerReview	String
date	Date
name	String
rating	Double
restaurantID	Integer 64
title	String

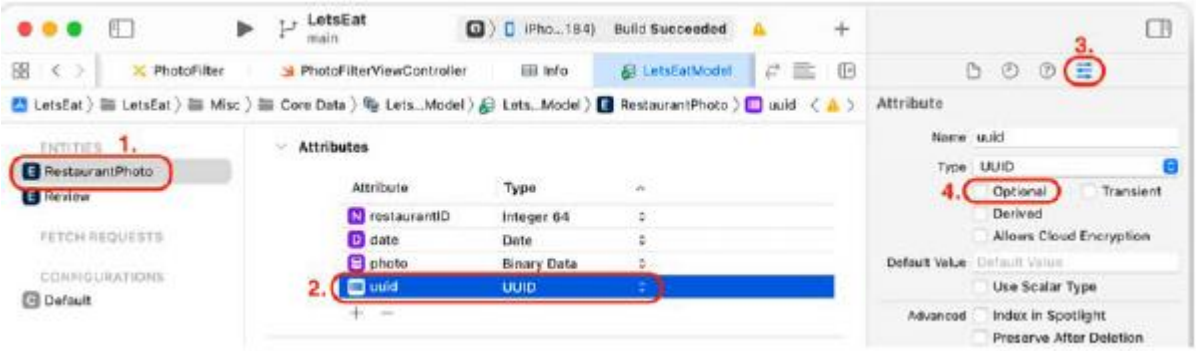
11. Dodaj jeszcze jeden atrybut uuid typu UUID. Jest to używane jako wartość klucza dla każdej instancji Review. Kliknij Inspektora modelu danych i w obszarze Atrybut odznacz Opcjonalne, ponieważ każda instancja recenzji musi mieć wartość klucza:



12. Dodaj drugi obiekt o nazwie RestaurantPhoto z następującymi atrybutami. Dane podstawowe nie mogą przechowywać obiektów UIImage, dlatego typ atrybutu zdjęcia jest ustawiony na Dane binarne. Przekonwertujesz zdjęcia na dane binarne, aby Core Data mogło je przechowywać, i przekonwertujesz je z powrotem na obiekty UIImage, gdy zajdzie taka potrzeba, aby wyświetlić je w aplikacji. data zostanie automatycznie ustawiona po utworzeniu instancji RestaurantPhoto. Użyjesz identyfikatora restauracji, aby powiązać zdjęcie z restauracją, a wartością klucza będzie uuid:

Attribute	Type
date	Date
photo	Binary Data
restaurantID	Integer 64
uuid	UUID

13. W przypadku uuid nie zapomnij odznaczyć opcji Opcjonalne w Inspektorze modelu danych:



Zakończyłeś tworzenie encji potrzebnych dla Twojej aplikacji. Zbuduj swoją aplikację. Pliki klas dla encji Review i RestaurantPhoto zostaną automatycznie utworzone przez Xcode, ale nie będą widoczne w nawigatorze projektu. Aby ułatwić pracę z nimi, utworzysz obiekt modelu dla każdej jednostki, zaczynając od elementu ReviewItem w następnej sekcji.

Tworzenie elementu recenzji

Utworzyłeś dwie encje do przechowywania recenzji i danych zdjęć za pomocą edytora modeli danych Xcode. Następnie Xcode automatycznie wygeneruje dwie definicje klas NSManagedObject na podstawie modelu danych Review

i RestaurantPhoto, ale nie widać ich w nawigatorze projektu. Utworzysz dwa obiekty modelu, ReviewItem i RestaurantPhotoItem, które będą współdziałać z instancjami Review i RestaurantPhoto. Utwórzmy teraz element ReviewItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder ReviewForm w nawigatorze projektu i wybierz opcję Nowa grupa. Nazwij tę grupę Model.

2. Kliknij prawym przyciskiem myszy folder Model w folderze ReviewForm i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

4. Nazwij ten plik ReviewItem. Kliknij Utwórz. Plik ReviewItem pojawi się w nawigatorze projektu.

5. Zmodyfikuj plik jak pokazano:

mport UIKit

struct ReviewItem {

var date: Date?

```
var rating: Double?

var title: String?

var name: String?

var customerReview: String?

var restaurantID: Int64?

var uuid = UUID()

}

extension ReviewItem {

init(review: Review) {

self.date = review.date

self.rating = review.rating

self.title = review.title

self.name = review.name

self.customerReview = review.customerReview

self.restaurantID = review.restaurantID

if let reviewUUID = review.uuid {

self.uuid = reviewUUID

}

}

}
```

Jak widać właściwości struktury ReviewItem są takie same jak atrybuty encji Review. Inicjator tworzy instancję ReviewItem i mapuje atrybuty z Review na właściwości instancji ReviewItem. W następnej sekcji utworzysz drugi obiekt modelu, RestaurantPhotoItem, który będzie obiektem modelu dla encji RestaurantPhoto.

Tworzenie RestaurantPhotoItem

Proces tworzenia RestaurantPhotoItem jest podobny do tworzenia ReviewItem. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Model w folderze PhotoFilter i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik RestaurantPhotoItem. Kliknij Utwórz.

4. Zmodyfikuj plik jak pokazano:

```
import UIKit
```

```
struct RestaurantPhotoItem {
var date: Date?
var photo: UIImage?
var photoData: Data {
guard let photo = photo, let photoData =
photo.pngData() else {
return Data()
}
return photoData
}
var restaurantID: Int64?
var uuid = UUID()
}
extension RestaurantPhotoItem {
init(restaurantPhoto: RestaurantPhoto) {
self.date = restaurantPhoto.date
if let restPhoto = restaurantPhoto.photo {
self.photo = UIImage(data: restPhoto,
scale: 1.0)
}
self.restaurantID =
restaurantPhoto.restaurantID
if let restPhotoUUID = restaurantPhoto.uuid {
self.uuid = restPhotoUUID
}
}
}
```

Podobnie jak w przypadku struktury ReviewItem, właściwości struktury RestaurantPhotoItem są takie same, jak atrybuty encji RestaurantPhoto. Istnieje jedna dodatkowa właściwość obliczana, photoData, która służy do przechowywania reprezentacji zdjęcia w formacie danych binarnych, ponieważ Core Data nie może przechowywać instancji UIImage. Inicjator tworzy instancję RestaurantPhotoItem i mapuje atrybuty z encji RestaurantPhoto na właściwości w instancji RestaurantPhotoItem. Zwróć uwagę na konwersję danych binarnych na UIImage podczas ustawiania wartości zdjęcia. Teraz, gdy

zadeklarowałeś i zdefiniowałeś ReviewItem i RestaurantPhotoItem, w następnej sekcji utwórzmy menedżera Core Data, który skonfiguruje komponenty Core Data dla Twojej aplikacji.

Tworzenie podstawowego menedżera danych

W tym momencie Xcode automatycznie wygenerował definicje klas Review i RestaurantData na podstawie modelu danych, a Ty zadeklarowałeś i zdefiniowałeś odpowiednie obiekty modelu, ReviewItem i RestaurantPhotoItem. Teraz utworzysz klasę CoreDataManager, która skonfiguruje komponenty Core Data dla Twojej aplikacji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Core Data w folderze Misc i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik CoreDataManager. Kliknij Utwórz. Plik CoreDataManager pojawi się w nawigatorze projektu.

4. Dodaj następujący kod po instrukcji importu:

```
zaimportuj CoreData
```

Daje to dostęp do biblioteki Core Data.

5. Dodaj następujący kod pomiędzy nawiasami klamrowymi, aby zadeklarować i zdefiniować strukturę CoreDataManager:

```
struct CoreDataManager {

let container: NSPersistentContainer

init() {

container = NSPersistentContainer(name:

"LetsEatModel")

container.loadPersistentStores {

(storeDesc, error) in

error.map {

print($0)

}

}

}

}
```

Spowoduje to utworzenie i inicjowanie wystąpień NSManagedObjectModel, NSPersistentStoreCoordinator i NSManagedObjectContext.

6. Aby utworzyć instancję struktury CoreDataManager, która będzie dostępna w całej aplikacji, kliknij plik AppDelegate w nawigatorze projektu i dodaj następujący kod po zamykającym nawiasie klamrowym:

```
extension CoreDataManager {

static var shared = CoreDataManager()

}
```

Następnie dodasz metody tworzenia instancji Review i RestaurantPhoto, zapełniasz je instancjami ReviewItem i RestaurantPhotoItem oraz zapisujesz je w trwałym sklepie. Wykonaj następujące kroki:

1. Kliknij plik CoreDataManager w nawigatorze projektu. Dodaj następujący kod po inicjatorze, aby zaimplementować metodę addReview(_:):

```
func addReview(_ reviewItem: ReviewItem) {

let review = Review(context:

container.viewContext)

review.date = Date()

if let reviewItemRating = reviewItem.rating {

review.rating = reviewItemRating

}

review.title = reviewItem.title

review.name = reviewItem.name

review.customerReview =

reviewItem.customerReview

if let reviewItemRestID =

reviewItem.restaurantID {

review.restaurantID = reviewItemRestID

}

review.uuid = reviewItem.uuid

save()

}
```

Ta metoda przyjmuje instancję ReviewItem jako parametr i pobiera pustą instancję Review z instancji NSManagedObjectContext. Właściwości instancji ReviewItem są przypisywane do atrybutów instancji Review, a metoda save() jest wywoływana w celu zapisania zawartości instancji NSManagedObjectContext w magazynie trwałym. Pamiętaj, że zobaczysz błąd, ponieważ nie zaimplementowałeś jeszcze metody save(). Na razie zignoruj ten błąd.

2. Dodaj następujący kod po metodzie addReview(_:) w celu zaimplementowania metody addPhoto(_:):

```
func addPhoto(_ restPhotoItem:

RestaurantPhotoItem) {
```

```
let restPhoto = RestaurantPhoto(context:

container.viewContext)

restPhoto.date = Date()

restPhoto.photo = restPhotoItem.photoData

if let restPhotoID =

restPhotoItem.restaurantID {

restPhoto.restaurantID = restPhotoID

}

restPhoto.uuid = restPhotoItem.uuid

save()

}
```

Ta metoda jest podobna do metody addReview(_:). Przyjmuje instancję RestaurantPhotoItem jako parametr i pobiera pustą instancję RestaurantPhoto z instancji NSManagedObjectContext. Właściwości instancji RestaurantPhotoItem są przypisywane do właściwości instancji RestaurantPhoto, a metoda save() wywoływana jest w celu zapisania zawartości instancji NSManagedObjectContext w magazynie trwałym. Pamiętaj, że zobaczysz błąd, ponieważ nie zaimplementowałeś jeszcze metody save(). Ponownie zignoruj na razie ten błąd.

3. Zaimplementuj metodę save(), dodając następujący kod przed ostatnim nawiasem klamrowym:

```
private func save() {

do {

if container.viewContext.hasChanges {

try container.viewContext.save()

}

} catch let error {

print(error.localizedDescription)

}

}
```

Ten blok do-catch zapisuje zawartość instancji NSManagedObjectContext w magazynie trwałym. Jeżeli zapisywanie nie powiodło się, w obszarze Debugowanie zostanie wydrukowany komunikat o błędzie. Jeśli chcesz pobrać recenzje i zdjęcia ze sklepu stałego, użyj identyfikatora restauracji jako identyfikatora, aby uzyskać recenzje i zdjęcia konkretnej restauracji. Zaimplementujmy teraz potrzebne do tego metody. Dodaj następujący kod po metodzie addPhoto(_:) w celu zaimplementowania metod fetchReviews(by:) i fetchPhotos(by:):

```
func fetchReviews(by identifier: Int) ->

[ReviewItem] {
```

```
let moc = container.viewContext
let request = Review.fetchRequest()
let predicate = NSPredicate(format:
"restaurantID = %i", identifier)
var reviewItems: [ReviewItem] = []
request.sortDescriptors =
[NSSortDescriptor(key: "date",
ascending: false)]
request.predicate = predicate
do {
for review in try moc.fetch(request) {
reviewItems.append(ReviewItem(review:
review))
}
return reviewItems
} catch {
fatalError("Failed to fetch reviews:
\(error)")
}
}
func fetchRestPhotos(by identifier: Int) ->
[RestaurantPhotoItem] {
let moc = container.viewContext
let request = RestaurantPhoto.fetchRequest()
let predicate = NSPredicate(format:
"restaurantID = %i", identifier)
var restPhotoItems: [RestaurantPhotoItem] = []
request.sortDescriptors =
[NSSortDescriptor(key: "date",
ascending: false)]
request.predicate = predicate
```

```
do {
for restPhoto in try moc.fetch(request) {
restPhotoItems.append(RestaurantPhotoItem
(restaurantPhoto: restPhoto))
}
return restPhotoItems
} catch {
fatalError("Failed to fetch restaurant
photos: \(error)")
}
}
```

Rozłóżmy to na czynniki pierwsze, zaczynając od fetchReview(by:):

```
let moc = container.viewContext
```

Pobiera to odwołanie do instancji NSManagedObjectContext.

```
let request = Review.fetchRequest()
```

Spowoduje to utworzenie żądania pobrania, które pobiera instancje recenzji z magazynu trwałego.

```
let predicate = NSPredicate(format: "restaurantID = %i", identifier)
```

Tworzy to predykat pobierania, który pobiera tylko te instancje recenzji z określonym identyfikatorem restauracji.

```
var reviewItems: [ReviewItem] = []
```

Spowoduje to utworzenie tablicy reviewItems, której będziesz używać do przechowywania wyników żądania pobrania.

```
request.sortDescriptors = [NSSortDescriptor(key: "date", ascending: false)]
```

Sortuje to wyniki żądania pobrania według daty, zaczynając od najnowszych elementów.

```
request.predicate = predicate
```

Ustawia predykat dla żądania pobrania.

```
do {
for review in try moc.fetch(request) {
reviewItems.append(ReviewItem(review:
review))
}
return reviewItems
```

} catch {

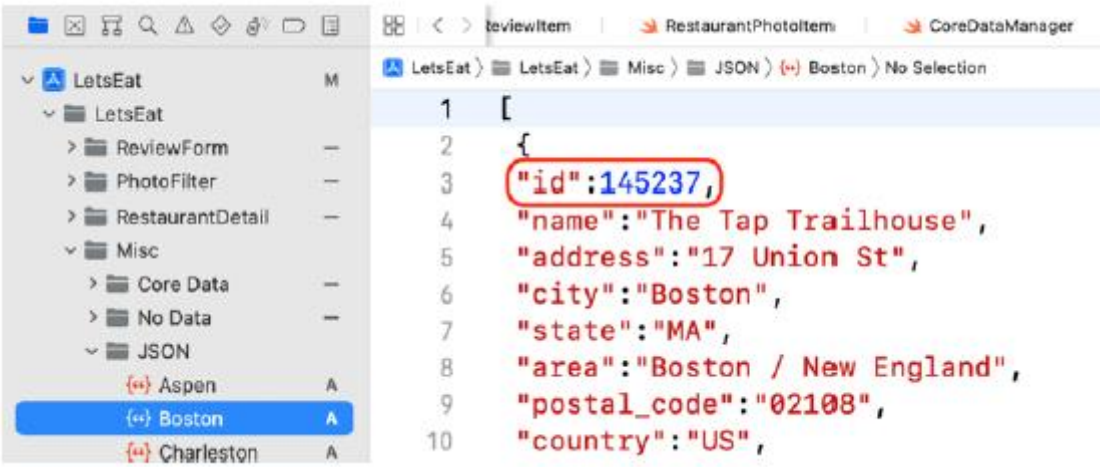
Ten blok do-catch wykonuje żądanie pobrania i umieszcza wyniki w tablicy items. Jeśli się nie powiedzie, aplikacja ulegnie awarii, a w obszarze Debugowanie zostanie wydrukowany komunikat o błędzie.

fetchPhotos(by:) działa w taki sam sposób jak fetchReview(by:), ale zamiast tego zwraca tablicę instancji RestaurantPhotoItems.

Utworzyłeś klasę CoreDataManager, która dodaje dane do magazynu trwałego i pobiera je z niego. Kompiluj i uruchamiaj aplikację, aby przetestować błędy. Powinno działać tak samo jak wcześniej. Zaimplementowałeś wszystkie komponenty Core Data w swojej aplikacji. Następnie skonfigurujesz RestaurantDetailViewController tak, aby używał danych podstawowych do wyświetlania recenzji i zdjęć na ekranie szczegółów restauracji. Zaczniesz od dowiedzenia się, w jaki sposób ekran szczegółów restauracji będzie wyświetlał recenzje i zdjęcia konkretnej restauracji.

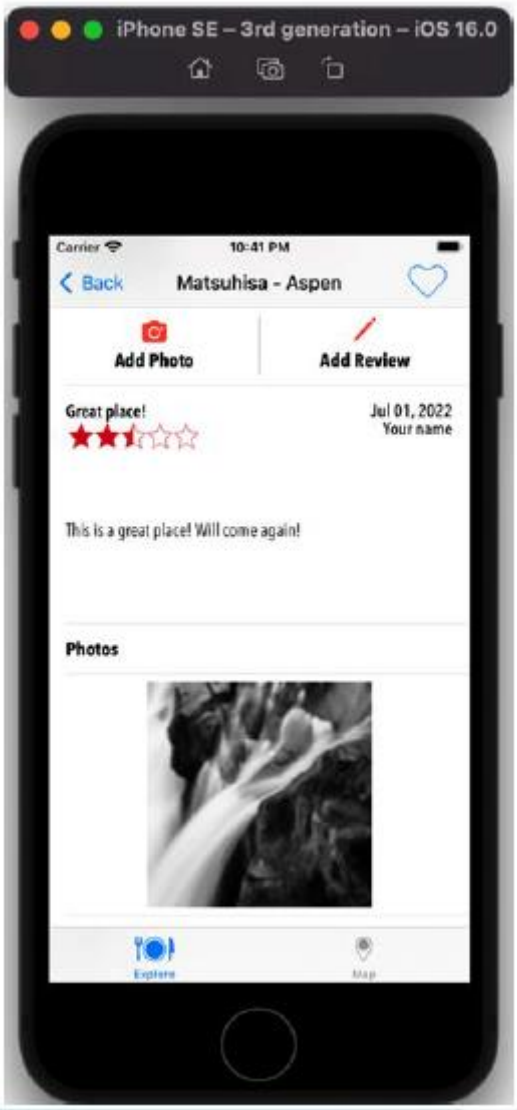
Zrozumienie, jak działa zapisywanie i ładowanie

Przyjrzyjmy się temu, co zrobiłeś do tej pory. Utworzono encje Review i RestaurantPhoto za pomocą edytora modelu danych i utworzyłeś dla nich odpowiednie obiekty modelu o nazwach ReviewItem i RestaurantPhotoItem. Utworzono klasę CoreDataManager, aby dodawać i pobierać instancje Review i RestaurantPhoto ze sklepu stałego. Klasa CoreDataManager wykorzystuje identyfikator restauracji do kojarzenia recenzji i zdjęć restauracji z konkretną restauracją, ale skąd on pochodzi? Otwórz folder Misc w swoim projekcie i otwórz folder JSON. Jeśli klikniesz którykolwiek z znajdujących się w nim plików JSON, zobaczysz, że każda restauracja ma unikalny identyfikator numeryczny. Na przykład identyfikator restauracji The Tap Trailhouse to 145237, jak pokazano na zrzucie ekranu poniżej:

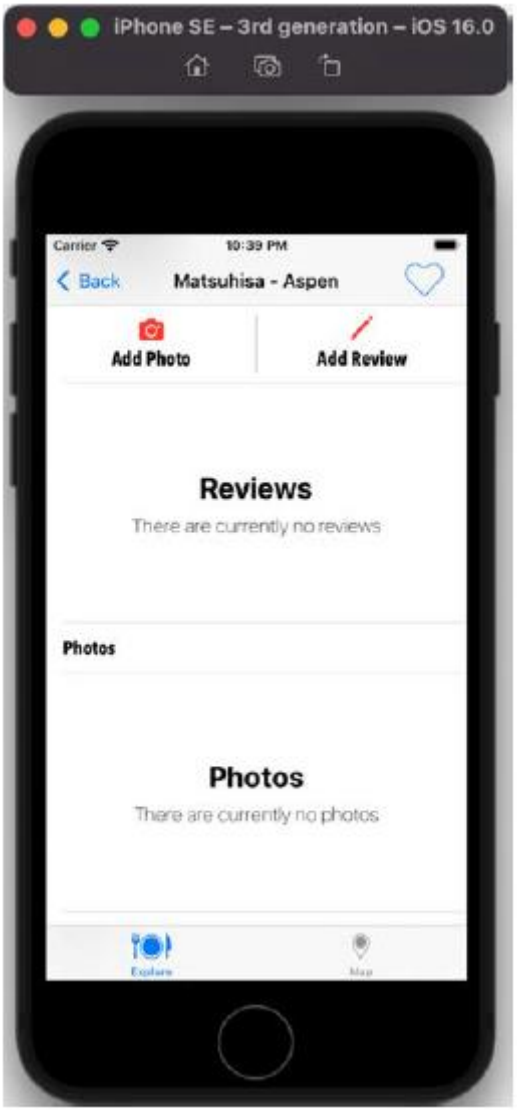


Zapisując zdjęcia restauracji i recenzje do sklepu stałego, zapiszesz je razem z tym identyfikatorem. Następnie, gdy konkretna restauracja zostanie wyświetlona na ekranie Szczegóły restauracji,

RestaurantDetailViewController użyje instancji ReviewDataManager do pobrania recenzji i zdjęć restauracji tej restauracji i wyświetlenia ich w widokach kolekcji, jak pokazano na zrzucie ekranu:



Jeśli nie ma żadnych recenzji ani zdjęć, użyjesz NoDataView, aby poinformować użytkownika, że nie ma żadnych recenzji ani zdjęć, jak pokazano na zrzucie ekranu:



RestaurantItem ma właściwość RestaurantID do przechowywania identyfikatorów restauracji. Gdy RestaurantDataManager ładuje plik JSON i tworzy tablicę instancji RestaurantItem, identyfikator każdej restauracji jest pobierany z pliku JSON i zapisywany we właściwości RestaurantID. W następnej sekcji zaktualizujesz ReviewFormViewController, aby zapisać recenzję z identyfikatorem restauracji w trwałym sklepie.

Aktualizowanie klasy ReviewFormViewController w celu zapisywania recenzji

Przycisk Zapisz na ekranie formularza recenzji obecnie po dotknięciu drukuje recenzję w obszarze debugowania. Aby zapisać recenzje, musisz zmodyfikować metodę onSaveTapped(_:) w celu zapisania recenzji w sklepie trwałym po dotknięciu przycisku Zapisz. Wykonaj następujące kroki:

1. Kliknij plik ReviewFormViewController w nawigatorze projektu. Dodaj następującą właściwość do klasy ReviewFormViewController przed deklaracjami outlet, aby przechowywać identyfikator restauracji:

var selectedRestaurantID: Int?

2. Utwórz rozszerzenie prywatne, przenieś do niego metodę onSaveTapped(_:) i zmodyfikuj ją w następujący sposób:

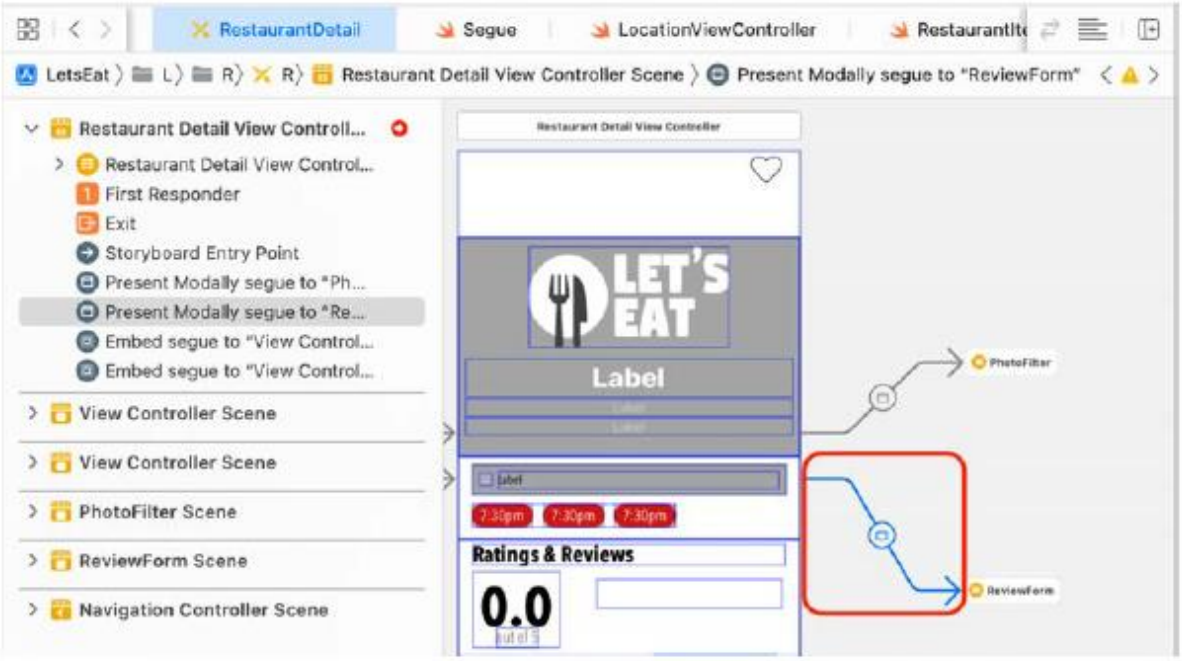
```
private extension ReviewFormViewController {

@IBAction func onSaveTapped(_ sender: Any) {

var reviewItem = ReviewItem()

reviewItem.rating = ratingsView.rating

reviewItem.title = titleTextField.text

reviewItem.name = nameTextField.text

reviewItem.customerReview =

reviewTextView.text

if let selRestID = selectedRestaurantID {

reviewItem.restaurantID =

Int64(selRestID)

}

CoreDataManager.shared.addReview(reviewItem)

dismiss(animated: true, completion: nil)

}

}
```

Zamiast drukować szczegóły recenzji w obszarze Debugowanie, onSaveTapped(_:) utworzy teraz instancję ReviewItem, przypisze dane uzyskane z ekranu formularza recenzji do jego właściwości i wywoła CoreDataManager.shared.addReview(reviewItem), aby zapisać recenzję w trwały sklep. Należy pamiętać, że obecnie nie ma mechanizmu przekazywania identyfikatora restauracji do kontrolera ReviewFormViewController. W następnej sekcji zobaczysz, jak uzyskać identyfikator restauracji z RestaurantDetailViewController i przekazać go do ReviewFormViewController.

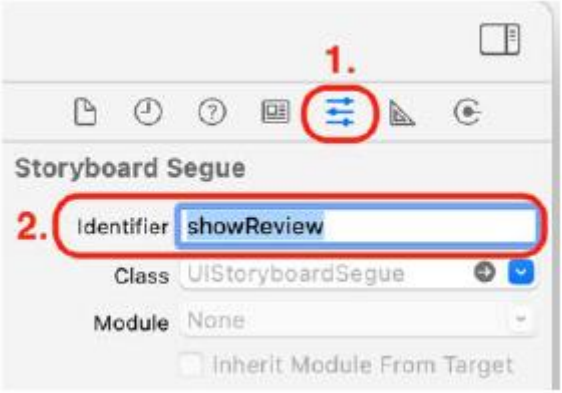
Przekazywanie RestaurantID do instancji ReviewFormViewController

Przycisk Zapisz na ekranie formularza recenzji umożliwia teraz zapisanie recenzji z identyfikatorem restauracji, ale skąd ReviewFormViewController pobiera ten identyfikator? Musisz przekazać wartość identyfikatora z RestaurantDetailViewController do ReviewFormViewController, aby mógł on zapisywać recenzje z identyfikatorem tej restauracji. Tak jak to zrobiłeś wcześniej w rozdziale 18, Pierwsze kroki z plikami JSON, użyjesz identyfikatorów przejścia, aby określić, które przejście ma miejsce, a następnie zaimplementujesz metody przekazywania wartości identyfikatora między dwoma kontrolerami widoku. Wykonaj następujące kroki:

1. Otwórz plik scenorysu RestaurantDetail i wybierz przejście używane do przejścia do sceny ReviewForm:



2. W Inspektorze atrybutów ustaw Identyfikator w obszarze Storyboard Segue na showReview i naciśnij Return:



3. Kliknij plik RestaurantDetailViewController w nawigatorze projektu. Dodaj następujący kod po funkcji viewDidLoad(), aby zaimplementować metodę przygotowania(for:sender:):

```
override func prepare(for segue:

UIStoryboardSegue, sender: Any?) {

if let identifier = segue.identifier {

switch identifier {

case Segue.showReview.rawValue:

showReview(segue: segue)

default:

print("Segue not added")
```

```
}

}

}
```

Metoda przygotowania(for:sender:) sprawdza, czy ciąg ma identyfikator showReview. Jeżeli tak, przed przejściem z ekranu „Szczegóły restauracji” do ekranu „Formularz recenzji” wykonywana jest metoda showReview(segue:) Wystąpi błąd, ponieważ funkcja showReview(segue:) nie została jeszcze zaimplementowana. Dodasz to dalej.

4. Dodaj metodę showReview(segue:) wewnątrz rozszerzenia prywatnego, przed metodą createRating():

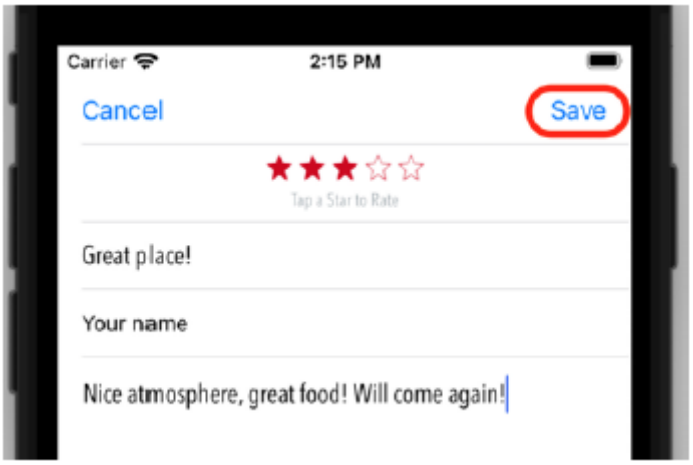
```
func showReview(segue: UIStoryboardSegue) {

guard let navController = segue.destination as?

UINavigationController, let viewController =

navController.topViewController as?

ReviewFormViewController else {

return

}

viewController.selectedRestaurantID =

selectedRestaurant?.restaurantID

}
```

Ustawia to właściwość RestaurantID elementu ReviewFormViewController na identyfikator wybranej restauracji.

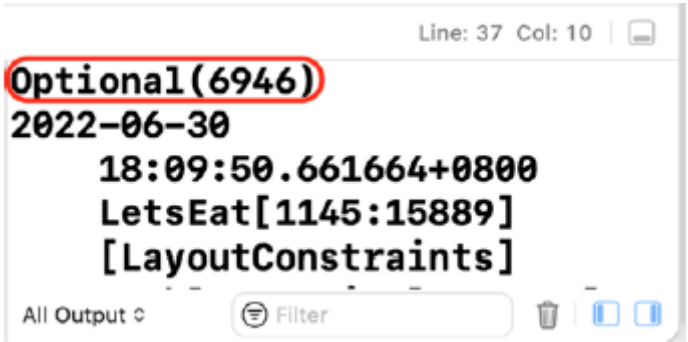
5. Kliknij plik ReviewFormViewController w nawigatorze projektu. Dodaj następujący kod do metody viewDidLoad(), aby wydrukować identyfikator restauracji w obszarze debugowania:

```
super.viewDidLoad()

print(selectedRestaurantID as Any)
```

Dzięki temu dowiesz się, że pomyślnie przekazałeś wartość identyfikatora do kontrolera ReviewFormViewController. Zbuduj i uruchom swój projekt, ustaw lokalizację i dotknij Wszystkie. Stuknij restaurację, a następnie stuknij przycisk Dodaj recenzję. Na ekranie Formularz recenzji wprowadź recenzję i dotknij opcji Zapisz:



Identyfikator restauracji pojawi się w obszarze debugowania (być może trzeba będzie przewinąć, aby go zobaczyć):



Pomyślnie przekazałeś identyfikator restauracji z RestaurantDetailViewController do ReviewFormViewController. Teraz zróbmy to samo ze zdjęciami. Zaktualizujesz PhotoFilterViewController, aby zapisywać zdjęcia z identyfikatorem restauracji w sklepie stałym, gdy w następnej sekcji zostanie naciśnięty przycisk Zapisz.

Aktualizacja klasy PhotoFilterViewController w celu zapisywania zdjęć

Kod umożliwiający klasie PhotoFilterViewController zapisywanie zdjęć w magazynie trwałym jest podobny do kodu zaimplementowanego w klasie ReviewFormViewController służącego do zapisywania recenzji. Teraz zaktualizujesz klasę PhotoFilterViewController, aby zapisywać zdjęcia po dotknięciu przycisku Zapisz. Wykonaj następujące kroki:

1. Kliknij plik PhotoFilterViewController w nawigatorze projektu. Dodaj następującą metodę wewnątrz rozszerzenia prywatnego po metodzie inicjalizacji():

```
func saveSelectedPhoto() {

if let mainImage = self.mainImageView.image {

var restPhotoItem =

RestaurantPhotoItem()

restPhotoItem.date = Date()
```

```
restPhotoItem.photo =

mainImage.preparingThumbnail(of:

CGSize(width: 100, height: 100))

if let selRestID = selectedRestaurantID

{

restPhotoItem.restaurantID =

Int64(selRestID)

}

CoreDataManager.shared

.addPhoto(restPhotoItem)

}

dismiss(animated: true, completion: nil)

}
```

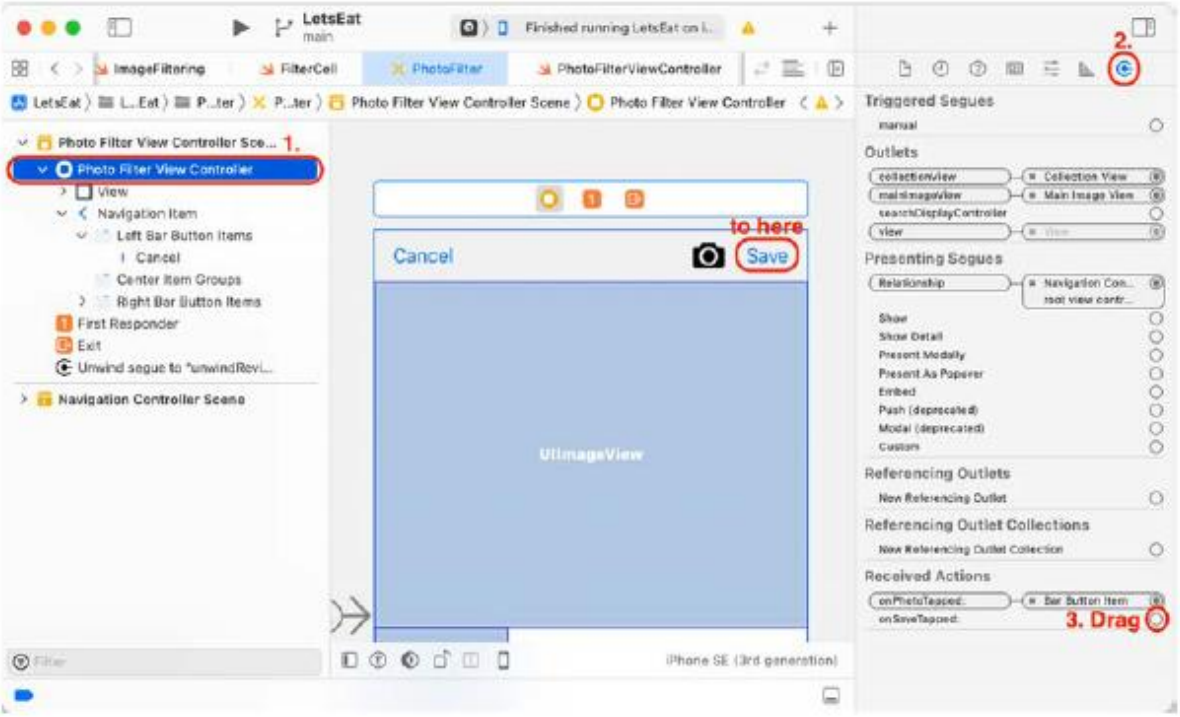
Pamiętaj, że mainImageView jest wyjściem dla dużego widoku obrazu na ekranie Filtru zdjęć. Metoda saveSelectedPhoto() najpierw sprawdza, czy ustawiono właściwość image obiektu mainImageView. Jeśli tak, obraz jest przypisywany do mainImage. Następnie tworzona jest instancja RestaurantPhotoItem i przypisana do restPhotoItem, a bieżąca data jest przypisywana do właściwości date instancji restPhotoItem. Metoda przygotowanieThumbnail(of:) instancji mainImage służy do utworzenia mniejszej wersji obrazu, która jest przypisana do właściwości photo instancji restPhotoItem. Następnie właściwość restPhotoItem instancji restPhotoItem jest ustawiana na identyfikator wybranej restauracji. Na koniec wywoływana jest metoda CoreDataManager.shared.addPhoto(_:) w celu zapisania zdjęcia w magazynie trwałym, a ekran Filtr zdjęć zostaje zamknięty.

2. Musisz uruchomić tę metodę po dotknięciu przycisku Zapisz. Dodaj następującą metodę w rozszerzeniu prywatnym po metodzie onPhotoTapped(_:):

```
@IBAction func onSaveTapped(_ sender: Any) {

saveSelectedPhoto()

}
```

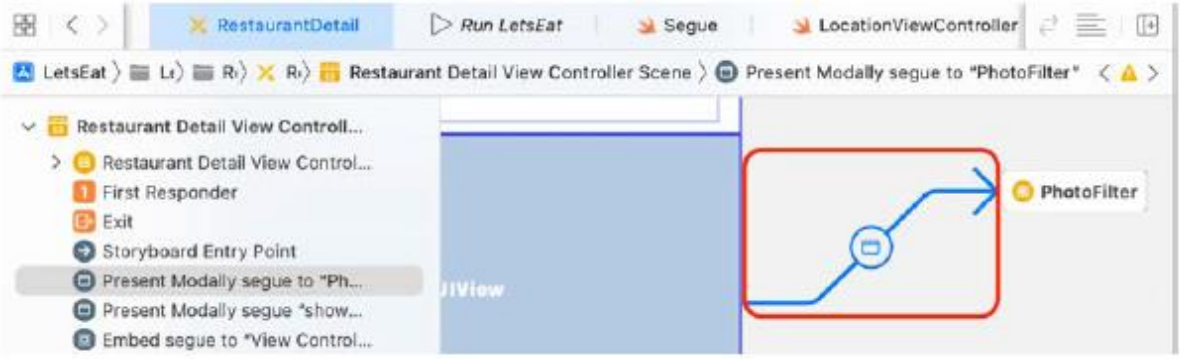
Ta metoda zostanie później połączona z przyciskiem Zapisz.

3. Aby przypisać metodę onSaveTapped(_:) do przycisku Zapisz, otwórz plik scenorysu PhotoFilter i kliknij ikonę Kontroler widoku filtru zdjęć w scenie kontrolera widoku filtru zdjęć. Otwórz Inspektora połączeń. Przeciągnij z akcji onSaveTapped na przycisk Zapisz:

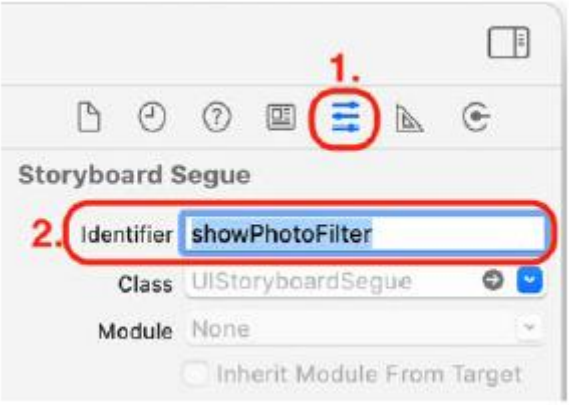


Zanim będziesz mógł zapisać, musisz przekazać identyfikator restauracji do PhotoFilterViewController. Podobnie jak poprzednio, użyjesz identyfikatorów przejścia, aby określić, które przejście ma miejsce, a następnie zaimplementujesz metody przekazywania wartości identyfikatora między dwoma kontrolerami widoku. Wykonaj następujące kroki:

1. Kliknij plik scenorysu RestaurantDetail w nawigatorze projektu i wybierz przejście używane do przejścia do ekranu Filtr zdjęć:



2. W Inspektorze atrybutów ustaw Identyfikator w obszarze Storyboard Segue, aby pokazaćPhotoFilter i naciśnij Return:



3. Kliknij plik RestaurantDetailViewController w nawigatorze projektu. Zaktualizuj metodę przygotowania(for:sender:) w następujący sposób:

```
override func prepare(for segue: UIStoryboardSegue,

sender: Any?){

if let identifier = segue.identifier {

switch identifier {

case Segue.showReview.rawValue:

showReview(segue: segue)

case Segue.showPhotoFilter.rawValue:

showPhotoFilter(segue: segue)

default:

print("Segue not added")

}

}

}
```

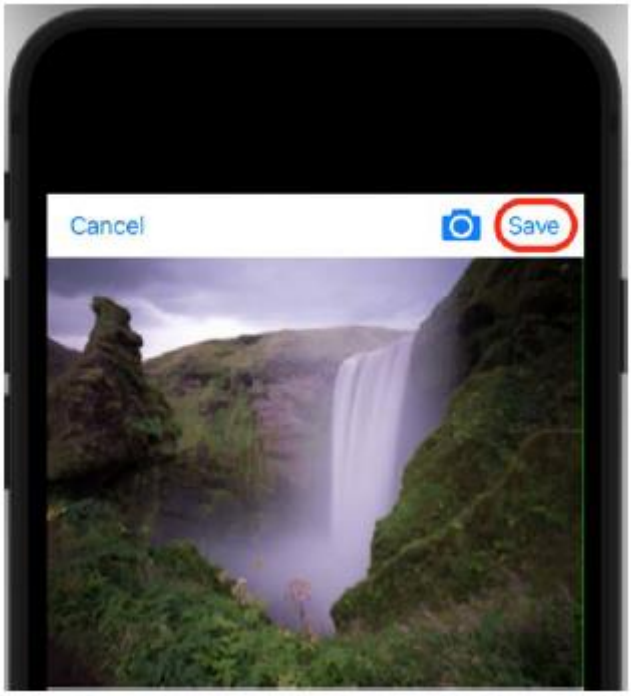
To powoduje, że metoda przygotowania(for:sender:) sprawdza, czy ciąg ma identyfikator showPhotoFilter. Jeżeli tak, przed przejściem z ekranu Szczegóły restauracji do ekranu Filtr zdjęć wykonywana jest metoda showPhotoFilter(segue:) Wystąpi błąd, ponieważ funkcja showPhotoFilter(segue:) nie została jeszcze zaimplementowana.

4. Dodaj metodę showPhotoFilter(segue:) po metodzie showReview() w swoim prywatnym rozszerzeniu:

```
func showPhotoFilter(segue: UIStoryboardSegue) {

guard let navController = segue.destination as?

UINavigationController, let viewController =

navController.topViewController as?
```

```
PhotoFilterViewController else {

return

}

viewController.selectedRestaurantID =

selectedRestaurant?.restaurantID

}
```

Ustawia to właściwość RestaurantID elementu PhotoFilterViewController na identyfikator wybranej restauracji. Zbuduj i uruchom swój projekt, ustaw lokalizację i dotknij Wszystkie. Stuknij restaurację, a następnie stuknij przycisk Dodaj zdjęcie. Wybierz zdjęcie, zastosuj filtr i dotknij Zapisz:



Zdjęcie zostanie zapisane w sklepie trwałym i nastąpi powrót do ekranu Szczegóły restauracji. W tym momencie możesz zapisać recenzje i zdjęcia. Fantastyczny! W następnej sekcji dodasz kod, który wczyta recenzje i zdjęcia ze sklepu stałego, które zostaną wyświetlone na ekranie Szczegóły restauracji.

Wyświetlanie zapisanych recenzji i zdjęć na ekranie szczegółów restauracji

Przyciski Zapisz na ekranach Formularza recenzji i Filtru zdjęć zapisują teraz recenzje i zdjęcia z identyfikatorem restauracji. Teraz musisz skonfigurować ekran szczegółów restauracji, aby je wyświetlać. Jeśli zajrzysz do pliku scenorysu RestaurantDetail, zobaczysz, że widoki kolekcji zostały już skonfigurowane do wyświetlania zdjęć i recenzji w komórkach widoku tabeli statycznej. Wszystko, co musisz zrobić, to zaimplementować odpowiednie kontrolery widoku dla komórek widoku i kolekcji. Zaczniesz od komórek widoku i kolekcji używanych do wyświetlania recenzji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder LetsEat w nawigatorze projektu i wybierz opcję Nowa grupa. Nazwij tę grupę Recenzje.

2. Kliknij folder prawym przyciskiem myszy i wybierz Nowy plik.

3. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch, a następnie kliknij Dalej.

4. Skonfiguruj plik w następujący sposób:

Zajęcia: Komórka przeglądu

Podklasa: UICollectionViewCell

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

5. Kliknij Utwórz. Plik ReviewCell pojawi się w nawigatorze projektu. Wprowadź następujący kod między nawiasami klamrowymi:

```
import UIKit

class ReviewCell: UICollectionViewCell {

@IBOutlet var titleLabel: UILabel!

@IBOutlet var dateLabel: UILabel!

@IBOutlet var nameLabel: UILabel!

@IBOutlet var reviewLabel: UILabel!

@IBOutlet var ratingsView: RatingsView!

}
```

Komórka recenzji ma teraz właściwości dla wszystkich punktów sprzedaży w komórce widoku kolekcji. Następnie utwórzmy ReviewsViewController. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Recenzje i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz klasę Cocoa Touch, a następnie kliknij Dalej.

3. Skonfiguruj plik w następujący sposób:

Klasa: RecenzjeViewController

Podklasa: UIViewController

Utwórz także XIB: Niezaznaczone

Język: Swift

Kliknij Następny.

4. Kliknij Utwórz. W nawigatorze projektu pojawi się plik ReviewsViewController. Zmodyfikuj ten plik w następujący sposób:

```
import UIKit

class ReviewsViewController: UIViewController {
```

```
@IBOutlet var collectionView: UICollectionView!

var selectedRestaurantID: Int?

private var reviewItems: [ReviewItem] = []

private var dateFormatter: DateFormatter = {

let formatter = DateFormatter()

formatter.dateFormat = "MMM dd, yyyy"

return formatter

}()

override func viewDidLoad() {

super.viewDidLoad()

initialize()

}

override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

checkReviews()

}

}
```

Jak widać, implementacja ReviewsViewController jest prosta. Masz ujście dla widoku kolekcji, CollectionView. wybraneRestaurantID przechowuje restaurację

identyfikator. reviewItems zawiera tablicę instancji ReviewItem. dateFormatter jest instancją klasy DateFormatter, która będzie używana do formatowania wyświetlanej daty. Nie przejmuj się błędami, ponieważ w następnym kroku będziesz pisać implementację metod inicjalizacji() i setupDefaults().

5. Dodaj prywatne rozszerzenie z następującym kodem, jak pokazano:

```
private extension ReviewsViewController {

func initialize() {

setupCollectionView()

}

func setupCollectionView() {

let flow = UICollectionViewFlowLayout()

flow.sectionInset = UIEdgeInsets(top: 7,left: 7,

bottom: 7, right: 7)

flow.minimumInteritemSpacing = 0
```

```
flow.minimumLineSpacing = 7

flow.scrollDirection = .horizontal

collectionView.collectionViewLayout = flow

}

}
```

Rozszerzenie prywatne zawiera implementację funkcji inicjowania() i

metody setupCollectionView(). inicjalizacja() po prostu wywołuje setupCollectionView(). setupCollectionView() służy do konfigurowania przepływu i odstępów między widokami kolekcji i jest podobna do kodu, który napisałeś wcześniej.

6. Dodaj następującą metodę po setupCollectionView(), aby zaimplementować checkReviews():

```
func checkReviews() {

let viewController = self.parent as?

RestaurantDetailViewController

if let restaurantID =

viewController?.selectedRestaurant?

.restaurantID {

reviewItems =

CoreDataManager.shared

.fetchReviews(by: restaurantID)

if !reviewItems.isEmpty {

collectionView.backgroundView = nil

} else {

let view = NoDataView(frame:

CGRect(x: 0, y: 0, width:

collectionView.frame.width,

height:

collectionView.frame.height))

view.set(title: "Reviews", desc:

"There are currently no reviews")

collectionView.backgroundView = view

}

}
```

collectionView.reloadData()

}

Ta metoda pobierze wszystkie recenzje restauracji dla określonego identyfikatora restauracji.

Rozbijmy to:

let viewController = self.parent as?

RestaurantDetailViewController

Ta instrukcja przypisuje RestaurantDetailViewController do tymczasowej stałej viewController.

if let restaurantID =

viewController?.selectedRestaurant?.restaurantID {

Ta instrukcja przypisuje identyfikator restauracji pokazany na ekranie Szczegóły restauracji do identyfikatora restauracji.

reviewItems = CoreDataManager.shared

.fetchReviews(by: restaurantID)

Ta instrukcja pobiera tablicę recenzji pasujących do podanego identyfikatora restauracji ze sklepu stałego i przypisuje ją do elementu reviewItems.

if !reviewItems.isEmpty {

collectionView.backgroundView = nil

} else {

let view = NoDataView(frame:

CGRect(x: 0, y: 0, width:

collectionView.frame.width,

height:

collectionView.frame.height))

view.set(title: "Reviews", desc:

"There are currently no reviews")

collectionView.backgroundView = view

}

Jeśli istnieją recenzje tej restauracji, widok tła widoku kolekcji jest ustawiony na zero; w przeciwnym razie utwórz instancję NoDataView, ustaw właściwości title i desc odpowiednio na „Recenzje” i „Obecnie nie ma recenzji” i przypisz ją do widoku tła widoku kolekcji.

collectionView.reloadData()

Ten kod informuje widok kolekcji, aby przerysował się na ekranie.

7. Zaimplementuj metody źródła danych dla widoku kolekcji, dodając następujące rozszerzenie:

```
rozszerzenie RecenzjeViewController:
UICollectionViewDataSource {
func collectionView(_ collectionView:
UICollectionView, numberOfItemsInSection
section: Int) -> Int {
reviewItems.count
}
func collectionView(_ collectionView:
UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell {
let cell = collectionView
.dequeueReusableCell
(withReuseIdentifier: "reviewCell",
for: indexPath) as! ReviewCell
let reviewItem = reviewItems[indexPath.item]
cell.nameLabel.text = reviewItem.name
cell.titleLabel.text = reviewItem.title
cell.reviewLabel.text =
reviewItem.customerReview
if let reviewItemDate = reviewItem.date {
cell.dateLabel.text =
dateFormatter.string(from:
reviewItemDate)
}
if let reviewItemRating = reviewItem.rating
{
cell.ratingsView.rating =
reviewItemRating
}
```

```
return cell
}
}
```

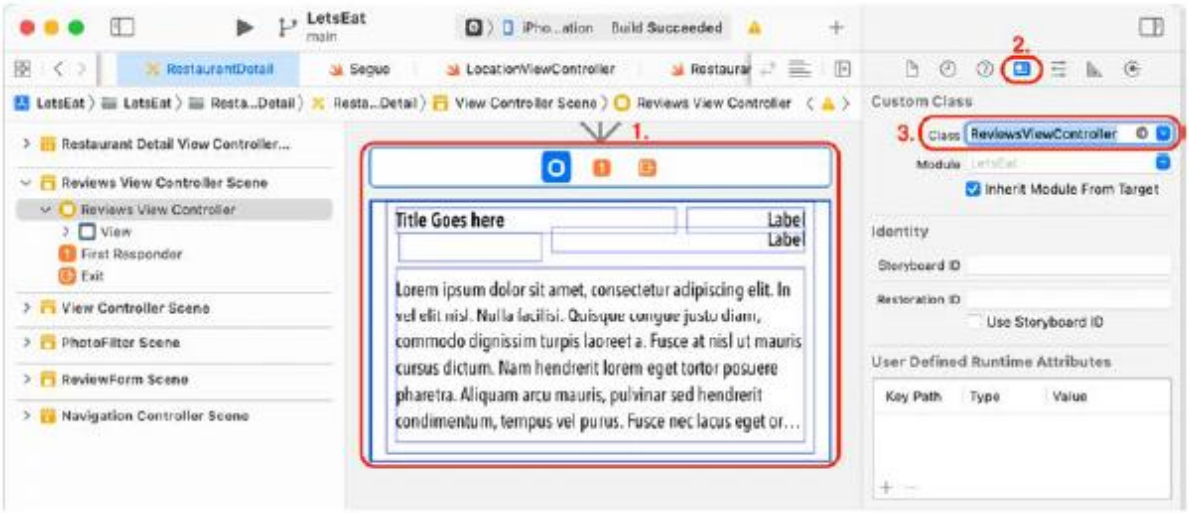
Jest to podobne do tego, co zrobiłeś wcześniej. Liczba komórek wyświetlanych w widoku kolekcji jest taka sama, jak liczba elementów w tablicy reviewItems. Zawartość każdej komórki ustawiasz za pomocą właściwości odpowiedniej instancji ReviewItem.

8. Dodaj metody delegowania układu przepływu do widoku kolekcji, dodając następujące rozszerzenie:

```
extension ReviewsViewController: UICollectionViewDelegateFlowLayout {
func collectionView(_ collectionView:
UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAt
indexPath: IndexPath) -> CGSize {
let edgeInset = 7.0
if reviewItems.count == 1 {
let cellWidth =
collectionView.frame.size.width -
(edgeInset * 2)
return CGSize(width: cellWidth, height:
200)
} else {
let cellWidth =
collectionView.frame.size.width -
(edgeInset * 3)
return CGSize(width: cellWidth, height:
200)
}
}
}
```

Ta metoda zwraca rozmiar wyświetlanej komórki widoku kolekcji. Jeśli w tablicy reviewItems znajduje się tylko jeden element, szerokość komórki jest ustawiana na szerokość widoku kolekcji – 14 punktów, w przeciwnym razie jest ustawiana na szerokość widoku kolekcji – 21 punktów. Wysokość jest ustawiona na 200 punktów. RecenzjeViewController jest już gotowy. Teraz możesz zakończyć implementację pliku scenorysu RestaurantDetail. Wykonaj następujące kroki:

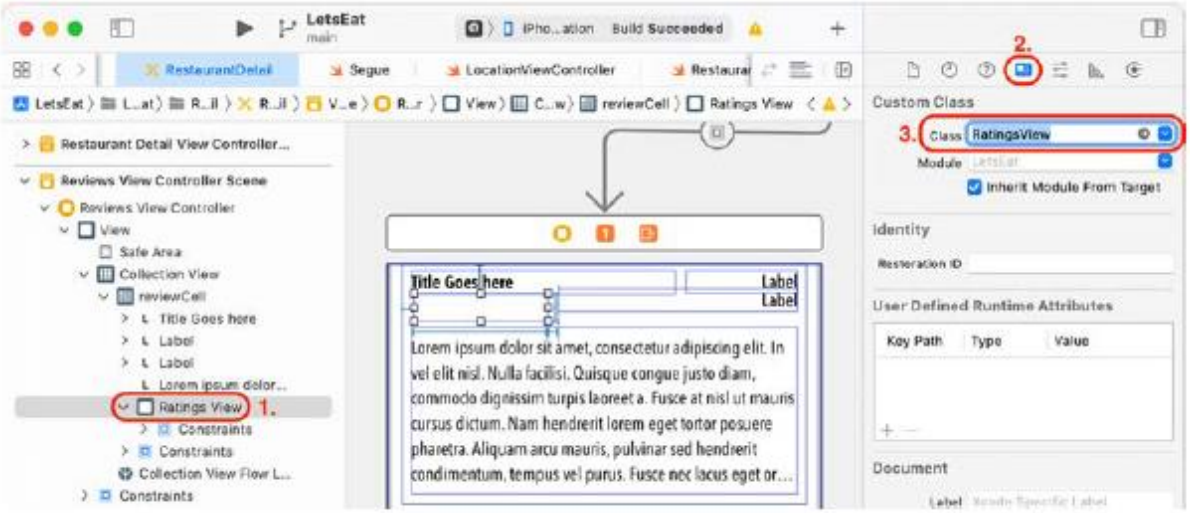
1. Kliknij plik scenorysu RestaurantDetail w nawigatorze projektu. Wybierz scenę kontrolera widoku zawierającą widok kolekcji recenzji. Kliknij przycisk Inspektor tożsamości, ustaw klasę na ReviewsViewController i naciśnij klawisz Return:



Nazwa sceny zmieni się na Recenzje Zobacz scenę kontrolera.

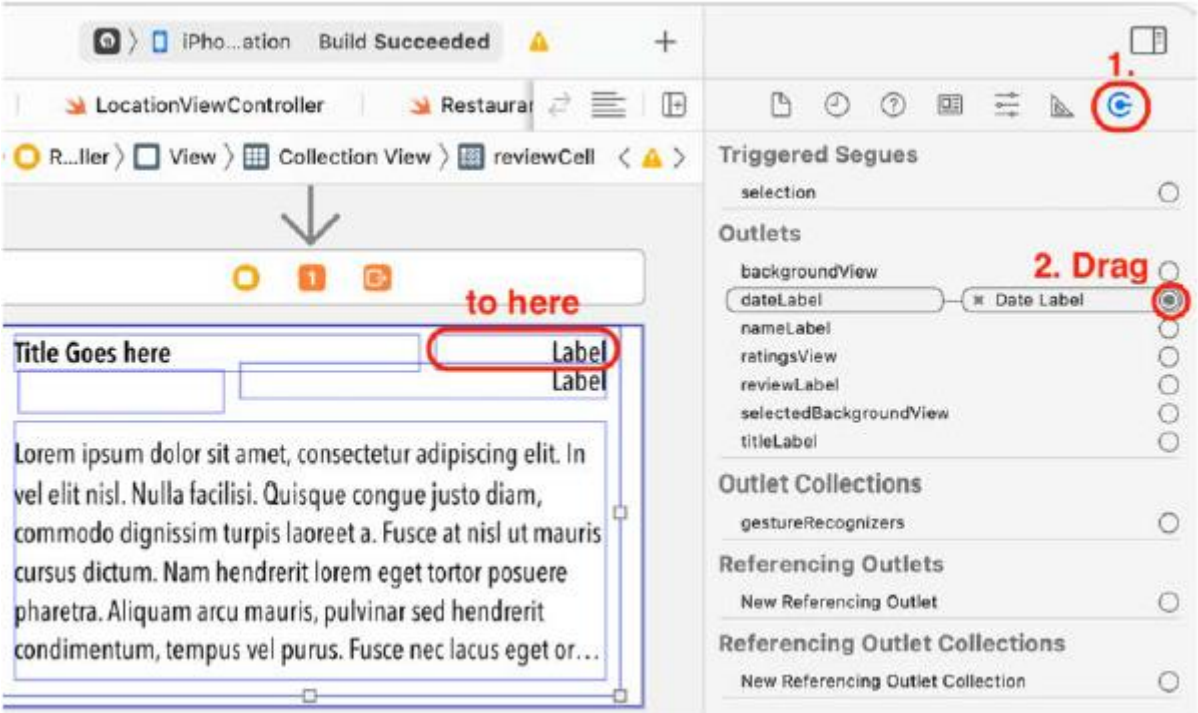
2. Wybierz komórkę widoku kolekcji w konspekcie dokumentu. Kliknij przycisk Inspektor tożsamości, ustaw klasę na ReviewCell i naciśnij klawisz Return:

3. Wybierz widok w konspekcie dokumentu. Kliknij przycisk Inspektora tożsamości, ustaw klasę na RatingsView i naciśnij klawisz Return:

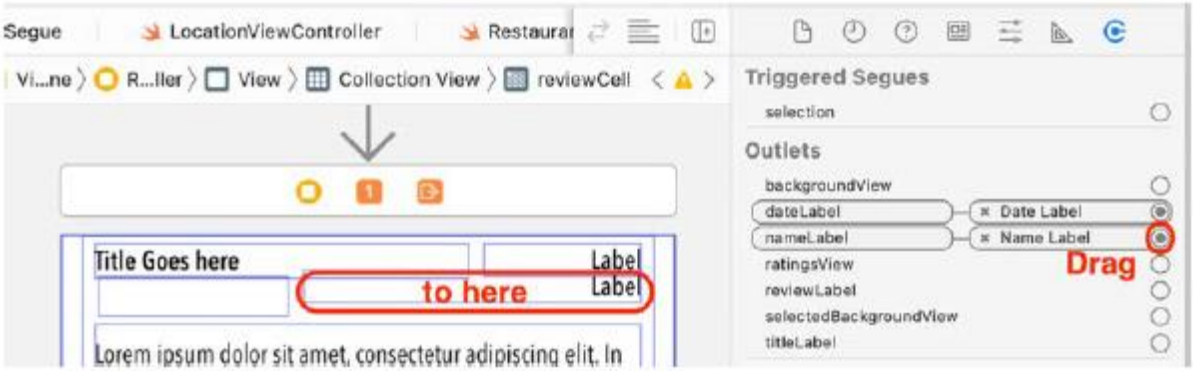


4. Wybierz komórkę recenzji w konspekcie dokumentu. Kliknij przycisk Inspektora atrybutów. Ustaw Identyfikator na reviewCell, jeśli nie jest jeszcze ustawiony:

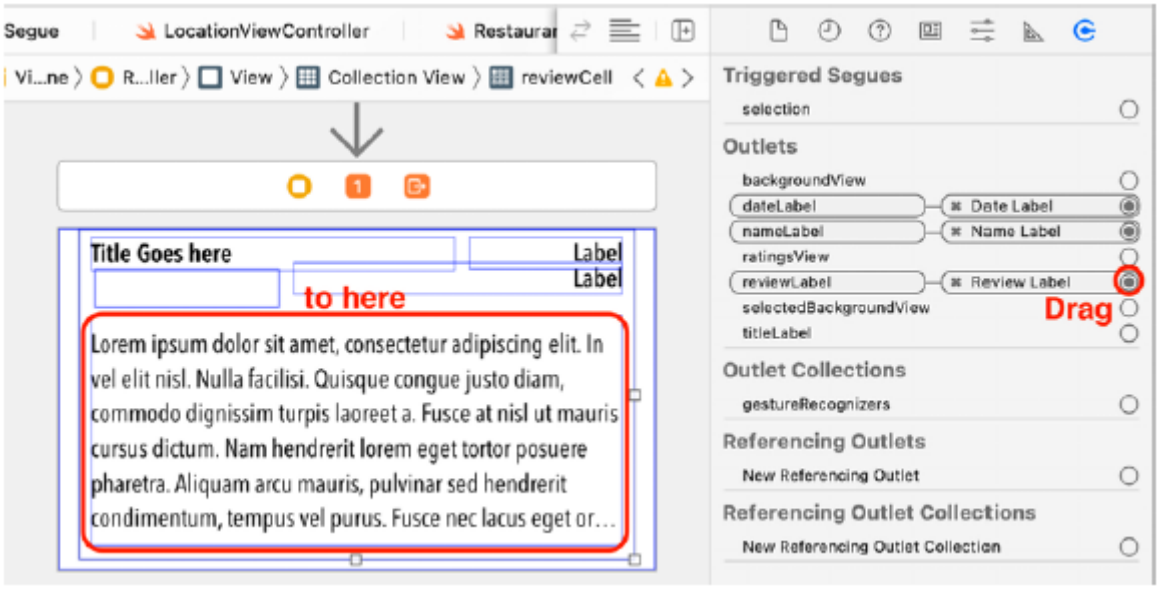
5. Kliknij Inspektora połączeń. Przeciągnij z wyjścia dateLabel na pokazaną etykietę:



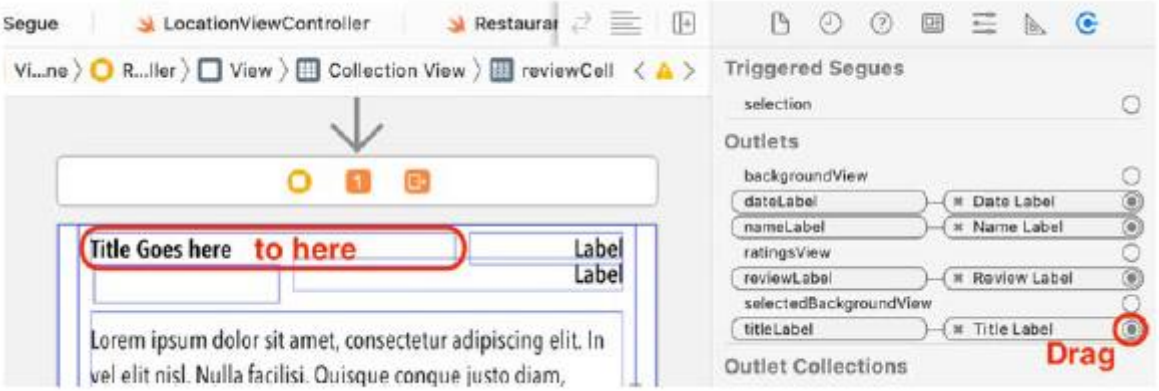
6. Przeciągnij z gniazdka nameLabel na pokazaną etykietę:



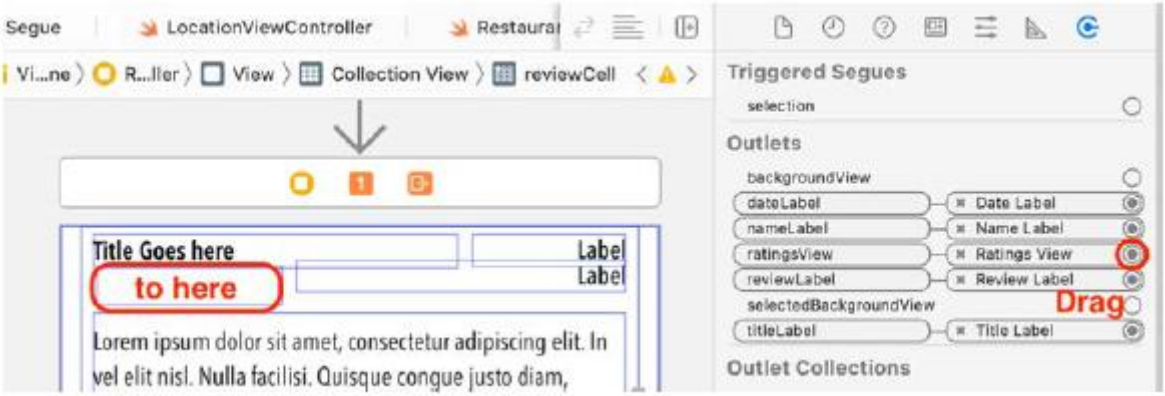
7. Przeciągnij z gniazda recenzjiLabel na pokazaną etykietę:



8. Przeciągnij z wylotu titleLabel do pokazanej etykiety:



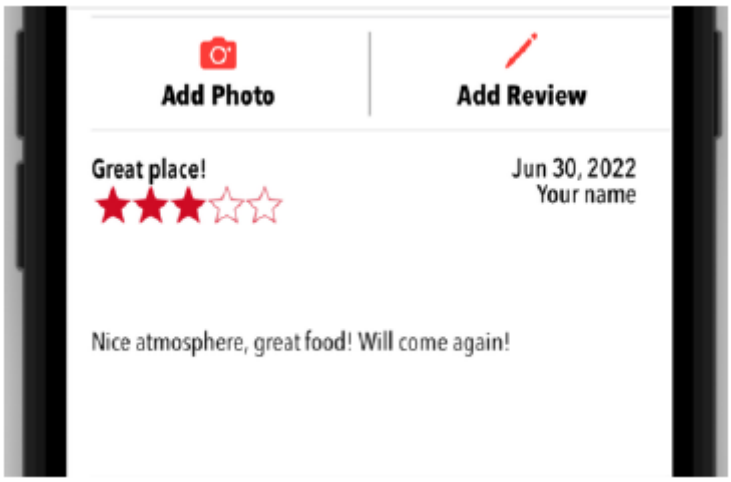
9. Przeciągnij z gniazdka ratingsView do pokazanego widoku ocen:



10. Wybierz ikonę kontrolera widoku recenzji dla sceny kontrolera widoku recenzji w konspekcie dokumentu. Przeciągnij z gniazda CollectionView do widoku kolekcji, jak pokazano:

11. Wybierz Widok kolekcji w konspekcie dokumentu. Przeciągnij z punktów delegowania i źródeł danych na ikonę kontrolera widoku recenzji:

Kompiluj i uruchamiaj swoją aplikację. Powinieneś zobaczyć wcześniej dodane recenzje:



Implementacja kontrolerów widoku dla widoku kolekcji i komórek widoku kolekcji używanych do wyświetlania recenzji została już ukończona, a Twoja aplikacja może teraz wyświetlać recenzje wprowadzone wcześniej za pomocą ekranu formularza recenzji. Jeśli masz więcej niż jedną recenzję, możesz przesuwać palcem w lewo i w prawo, aby zobaczyć każdą recenzję. Ponieważ każda recenzja ma ocenę, możesz ją wykorzystać do obliczenia i dodania ogólnej oceny restauracji. Zmodyfikujmy aplikację, aby zrobić to dalej.

Obliczanie ogólnej oceny restauracji

Na etykiecie ogólnej oceny na ekranie „Szczegóły restauracji" wyświetlana jest wartość 0,0, a w widoku ocen – 3,5 gwiazdki, niezależnie od rzeczywistej oceny. Aby dodać ogólną ocenę, musisz uzyskać oceny ze wszystkich recenzji i uśrednić je. Aby to zrobić, dodajmy nową metodę do CoreDataManager. Wykonaj następujące kroki:

1. Kliknij plik CoreDataManager w nawigatorze projektu (w folderze Core Data w folderze Misc). Dodaj następującą metodę przed metodą addReview(_:):

```
func fetchRestaurantRating(by identifier: Int) ->

Double {

let reviewItems = fetchReviews(by: identifier)

let sum = reviewItems.reduce(0, {$0 +

($1.rating ?? 0)})

return sum / Double(reviewItems.count)

}
```

W tej metodzie wszystkie recenzje dotyczące konkretnej restauracji są pobierane ze sklepu stałego i przypisywane do recenzji. Metoda redukcji() przyjmuje domknięcie, które służy do zsumowania wszystkich ocen z recenzji. Na koniec obliczana jest i zwracana średnia wartość oceny.

2. Kliknij plik RestaurantDetailViewController w nawigatorze projektu (w folderze RestaurantDetail). Zaktualizuj metodę createRating() w następujący sposób:

```
func createRating() {

ratingsView.isEnabled = false

if let restaurantID =

selectedRestaurant?.restaurantID {

let ratingValue =

CoreDataManager.shared.

fetchRestaurantRating(by:

restaurantID)

ratingsView.rating = ratingValue

if ratingValue.isNaN {
```

```
overallRatingLabel.text = "0.0"

} else {

let roundedValue = ((ratingValue *

10).rounded() / 10)

overallRatingLabel.text =

521

"\(roundedValue)"

}

}

}
```

Metoda najpierw przypisuje właściwość RestaurantID instancji wybranej restauracji do identyfikatora restauracji. Jeśli się powiedzie, wywoływana jest metoda CoreDataManager.shared.fetchRestaurantRating(), która pobiera wszystkie recenzje z wartością identyfikatora restauracji RestaurantID i oblicza średnią ocenę. ratingValue jest następnie ustawiany na średnią ocenę i używany do aktualizacji właściwości oceny widoku ocen, która określa liczbę gwiazdek wyświetlanych na ekranie Szczegóły restauracji. Wartość roundedValue jest następnie obliczana na podstawie wartości ratingValue w celu zwrócenia liczby z 1 miejscem dziesiętnym i używana do ustawiania właściwości tekstowej totalRatingLabel.

3. Ogólna ocena również będzie musiała zostać zaktualizowana za każdym razem, gdy użytkownik doda nową recenzję na ekranie Formularza recenzji. Dodaj następujący kod po metodzie viewDidLoad():

```
override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

createRating()

}
```

Spowoduje to ponowne obliczenie oceny po zamknięciu ekranu formularza recenzji i ponownym wyświetleniu ekranu szczegółów restauracji. Zbuduj i uruchom swój projekt, a powinieneś zobaczyć ogólną ocenę restauracji, które mają recenzje, a także odpowiednią liczbę gwiazdek, jak pokazano:

Pozostała jeszcze jedna rzecz do zrobienia, a mianowicie dodanie recenzji zdjęć. Twoim wyzwaniem jest dodanie recenzji zdjęć i wyświetlenie ich w widoku kolekcji tuż pod widokiem kolekcji używanym do recenzji. Można to zrobić bardzo podobnie do sposobu dodawania recenzji

Streszczenie

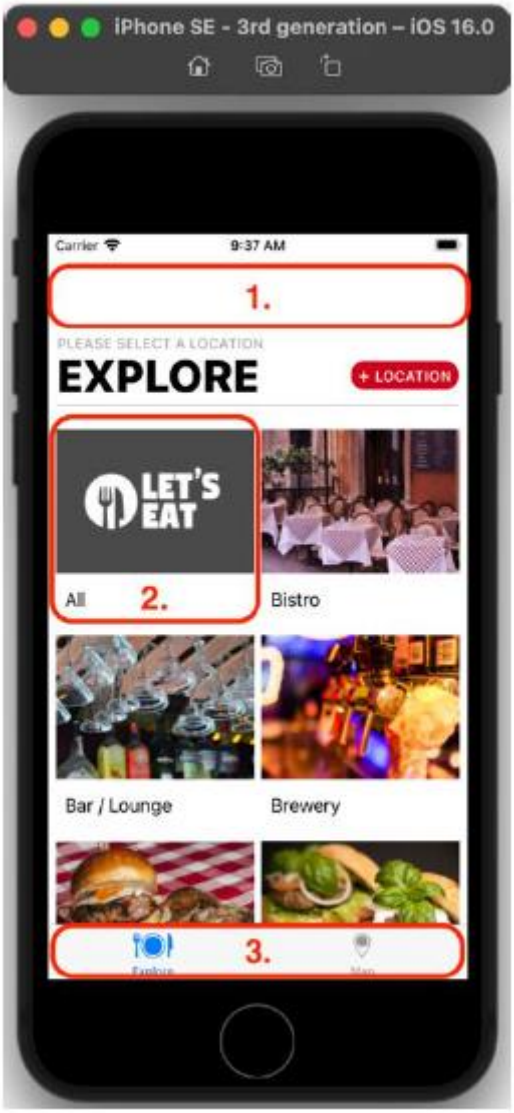
Poznałeś dane podstawowe i ich różne komponenty. Utworzyłeś modele danych dla swojej aplikacji o nazwach Review i RestaurantPhoto oraz utworzyłeś odpowiednie obiekty modelu dla swojej aplikacji o nazwach ReviewItem i RestaurantPhotoItem. Następnie zaimplementowałeś CoreDataManager, aby skonfigurować komponenty Core Data dla swojej aplikacji. Zaktualizowałeś ReviewFormViewController i PhotoFilterViewController, aby zapisywać recenzje i zdjęcia wraz z identyfikatorem restauracji w trwałym sklepie. Zmodyfikowałeś RestaurantDetailViewController, aby ładować recenzje konkretnej restauracji na podstawie identyfikatora restauracji i wyświetlać je w widoku kolekcji. Obliczyłeś także i wyświetliłeś ogólną ocenę tej restauracji. Na koniec samodzielnie zmodyfikowałeś RestaurantDetailViewController, aby ładował zdjęcia dla konkretnej restauracji na podstawie identyfikatora restauracji i wyświetlał je w widoku kolekcji. Masz teraz podstawową wiedzę na temat działania Core Data. Możesz także skonfigurować komponenty Core Data i włączyć interfejs między aplikacją a komponentami Core Data za pomocą klasy menedżera danych. Wiesz także, jak zapisywać i ładować recenzje i zdjęcia przy użyciu Core Data, które teraz będziesz mógł wdrożyć we własnych aplikacjach. Dotarłeś do końca długiej podróży i zakończyłeś tworzenie podstawowej funkcjonalności swojej aplikacji. Wszystkie ekrany działają, a recenzje i zdjęcia są trwałe. Fantastyczna praca!. W następnej części dowiesz się o nowych, ciekawych funkcjach, które Apple wprowadził w iOS 16, oraz o tym, jak dodać je do swojej aplikacji, zaczynając od przygotowania aplikacji na iPady i komputery Mac firmy Apple.

Pierwsze kroki z Mac Catalyst

Funkcja Mac Catalyst firmy Apple umożliwia utworzenie wersji aplikacji na iPada dla komputerów Mac. Pozwala to na współdzielenie tego samego projektu i kodu źródłowego dla obu platform, co ułatwia utrzymanie. Podczas WWDC2022 firma Apple ogłosiła aktualizacje programu Mac Catalyst, które umożliwiają dodanie funkcji klasy komputerowej iPadOS 16, automatyczną optymalizację paska narzędzi oraz nowe funkcje dla aplikacji opartych na dokumentach, takie jak pozycje menu i możliwość zmiany nazw dokumentów na pasku narzędzi. W tym rozdziale skupimy się na tym, jak uruchomić istniejącą aplikację na iPhone'a na iPadzie, aby można było stworzyć jej wersję na komputery Mac. W ten sposób będziesz mógł dotrzeć do ponad 100 milionów aktywnych użytkowników komputerów Mac. W tym rozdziale zmodyfikujesz swoją aplikację, aby działała na iPadach i komputerach Mac. Najpierw rozwiążesz niektóre problemy z interfejsem użytkownika w swojej aplikacji. Następnie dowiesz się, jak sprawić, by interfejs użytkownika aplikacji działał na iPadzie, wykorzystując większy rozmiar ekranu iPada. Następnie użyjesz wersji aplikacji na iPada, aby utworzyć wersję na komputer Mac. Pod koniec tego rozdziału będziesz w stanie zapewnić prawidłowe działanie istniejących aplikacji na iOS na wszystkich urządzeniach z systemem iOS, a także tworzyć aplikacje na Maca z aplikacji na iPada.

Naprawianie problemów z interfejsem użytkownika

Jedną z rzeczy, które zauważysz, jest to, że aplikacja na iOS nigdy nie jest tak naprawdę ukończona. Zawsze znajdziesz sposób na ulepszenie i udoskonalenie swojej aplikacji. Zbuduj i uruchom aplikację, a następnie porównaj ją z projektem pokazanym w prezentacji aplikacji. Po bliższym przyjrzeniu się zauważysz, że ekrany Twojej aplikacji różnią się nieznacznie od ekranów pokazanych w przewodniku po aplikacji (w rozdziale 10, Konfigurowanie interfejsu użytkownika) i wymagają zmian. Zacznijmy od ekranu Eksploruj Twojej aplikacji:



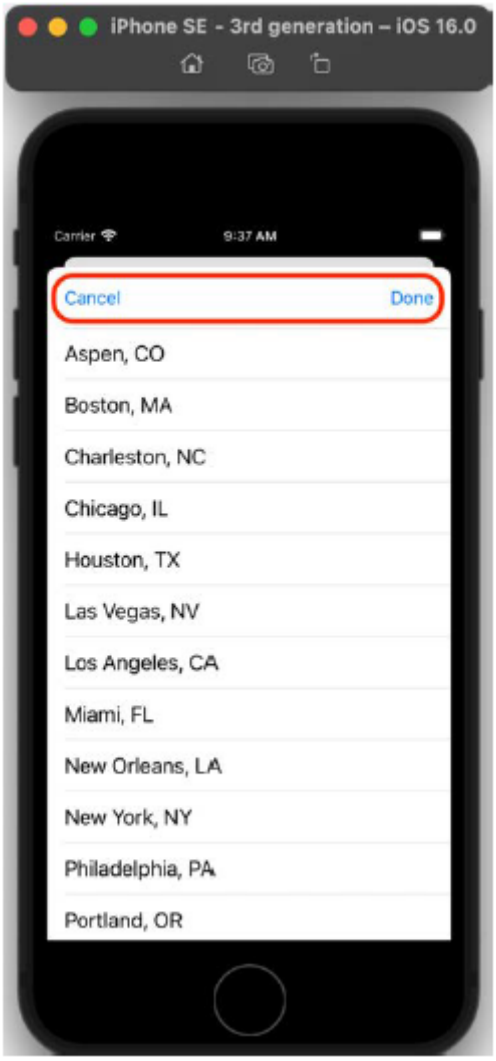
Zmiany wymagane na ekranie Eksploruj są następujące. Sprawdź liczby, aby zobaczyć część wymagającą wymiany:

• Pasek nawigacyjny (1) nie jest obecny na trasie aplikacji i należy go usunąć.

• Komórki widoku kolekcji (2) mają ostre narożniki. Zaimplementujesz zaokrąglone rogi komórek, aby pasowały do komórek pokazanych w prezentacji aplikacji.

• Przyciski paska zakładek są niebieskie (3). Zmienisz kolor przycisku paska kart na czerwony, aby dopasować go do prezentacji aplikacji.

Przyjrzyjmy się teraz ekranowi Lokalizacje Twojej aplikacji:



Brakuje dużego tytułu u góry ekranu Lokalizacje wyświetlanego w przewodniku po aplikacji i konieczne będzie jego dodanie. Jak widać, należy wprowadzić tylko cztery drobne zmiany, a zmiany te są łatwe do wdrożenia. Zaczniesz od modyfikacji ekranu Eksploruj. Wykonaj następujące kroki:

1. Kliknij plik ExploreViewController w folderze Explore w nawigatorze projektu.

2. Dodaj metodę viewWillAppear() po metodzie viewDidLoad() i dodaj kod wewnątrz tej metody, aby ukryć pasek nawigacyjny kontrolera nawigacji:

```
override func viewWillAppear(_ animated: Bool) {

super.viewWillAppear(animated)

navigationController?.setNavigationBarHidden(true,

animated: false)

}
```

Pamiętaj, że jeśli dodasz ten kod do metody viewDidLoad(), pasek nawigacyjny zostanie ukryty tylko wtedy, gdy po raz pierwszy pojawi się ekran Eksploruj, i pojawi się ponownie po przejściu z ekranu Lokalizacje lub ekranu Lista restauracji z powrotem do ekranu Eksploruj.

3. Aby zaokrąglić rogi komórek widoku kolekcji na ekranie Eksploruj, kliknij plik ExploreCell (wewnątrz folderu Widok w folderze Eksploruj) w nawigatorze projektu i po deklaracjach wylotu dodaj następującą metodę:

```
override func awakeFromNib() {

super.awakeFromNib()

exploreImageView.layer.cornerRadius = 9

exploreImageView.layer.masksToBounds = true

}
```

4. Aby zmienić kolory przycisków paska kart, kliknij plik AppDelegate w nawigatorze projektu i po ostatnim nawiasie klamrowym dodaj rozszerzenie prywatne zawierające następujące metody:

```
private extension AppDelegate {

func initialize() {

setupDefaultColors()

}

func setupDefaultColors() {

UITabBar.appearance().tintColor = .systemRed

UITabBarItem.appearance().

setTitleTextAttributes(

[NSAttributedString.Key.foregroundColor:

UIColor.systemRed], for:

UIControl.State.selected)

UINavigationBar.appearance().tintColor =

.systemRed

}

}
```

Plik AppDelegate zawiera deklarację i definicję klasy AppDelegate. Ta klasa obsługuje zdarzenia aplikacji, na przykład to, co dzieje się po uruchomieniu aplikacji, przesłaniu jej do tła, zakończeniu itd. Możesz tutaj dodać kod, aby skonfigurować aplikację podczas jej uruchamiania. Tak jak robiłeś to wcześniej, użyjesz metody inicjalizacji(), aby wywołać wszystkie inne metody konfiguracji. W tym przypadku metoda inicjalizacji() wywołuje metodę setupDefaultColors(). Metoda setupDefaultColors() zmieni kolory odcieni elementów na pasku kart i pasku nawigacji na czerwony i sprawi, że pasek kart

będzie nieprzezroczysty. Wykorzystuje metodę pojawiania się(), która globalnie ustawia atrybuty dla każdej karty i paska nawigacyjnego, który został lub zostanie utworzony.

5. Musisz wywołać metodę inicjalizacji() podczas uruchamiania aplikacji, dlatego zmodyfikuj metodę applicati on(_:didFinishLaunchingWithOptions:) w następujący sposób:

func application(_ application: UIApplication,

didFinishLaunchingWithOptions launchOptions:[UIApplication.

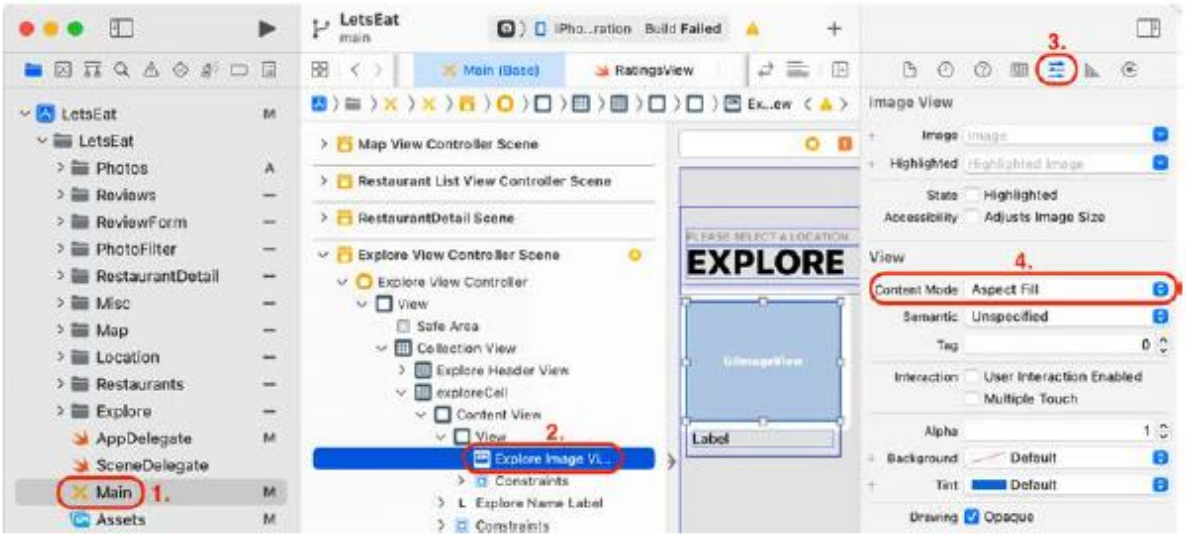
LaunchOptionsKey: Any]?) -> Bool {

initialize()

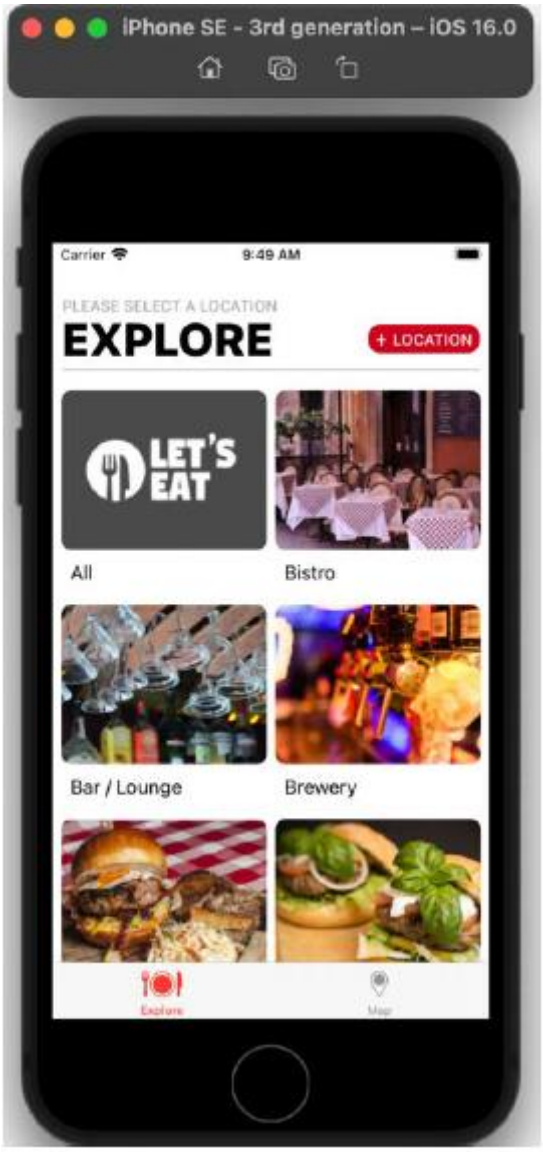
return true

}

6. Kliknij plik głównego scenorysu w nawigatorze projektu. W obszarze Eksploruj scenę kontrolera widoku kliknij opcję Eksploruj widok obrazu. Wybierz Inspektora atrybutów i w obszarze Widok zmień Tryb zawartości na Wypełnienie aspektami:



Dzięki temu obrazy mogą zajmować całą ramkę widoku obrazu i wyświetlać zaokrąglone rogi zakodowane w kroku 3. Kompiluj i uruchamiaj aplikację. Ekran Eksploruj powinien wyglądać następująco:



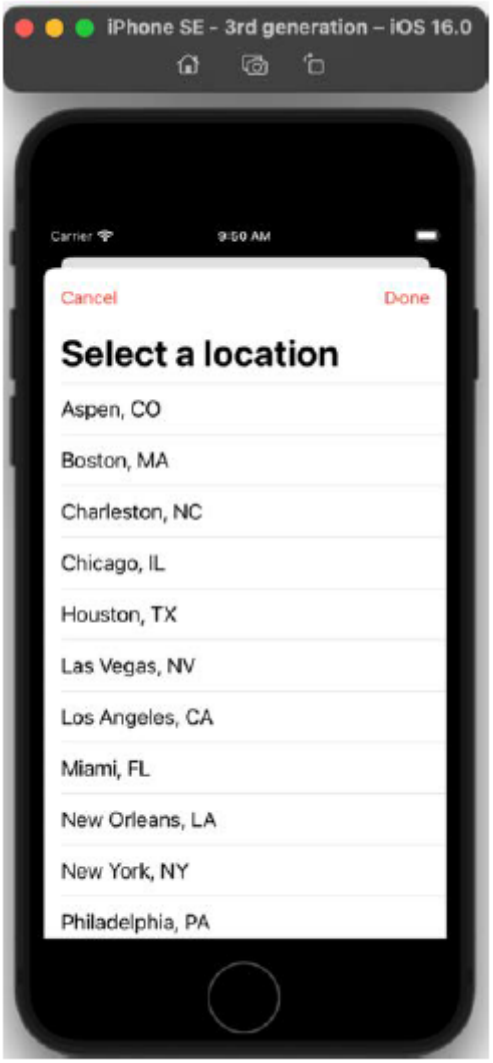
Zobaczysz, że nie ma paska nawigacyjnego, rogi każdej komórki są zaokrąglone, a ikony i tytuły przycisków Eksploruj i Mapa są teraz czerwone po wybraniu.

7. Następnie zaktualizujesz klasę LocationViewController. Kliknij plik LocationViewController w folderze Location w nawigatorze projektu i zmodyfikuj metodę inicjalizacji(), aby ustawić tytuł ekranu Lokalizacje:

```
func initialize() {

manager.fetch()

title = "Select a location"

navigationController?.navigationBar.

prefersLargeTitles = true

}
```

Każdy kontroler widoku ma właściwość tytułu, którą można wyświetlić na pasku nawigacyjnym. Ten kod ustawia tytuł na Wybierz lokalizację i wyświetla go dużymi literami u góry ekranu. Utwórz i

uruchom aplikację, a następnie dotknij przycisku LOKALIZACJA. Ekran Lokalizacje powinien wyglądać następująco:



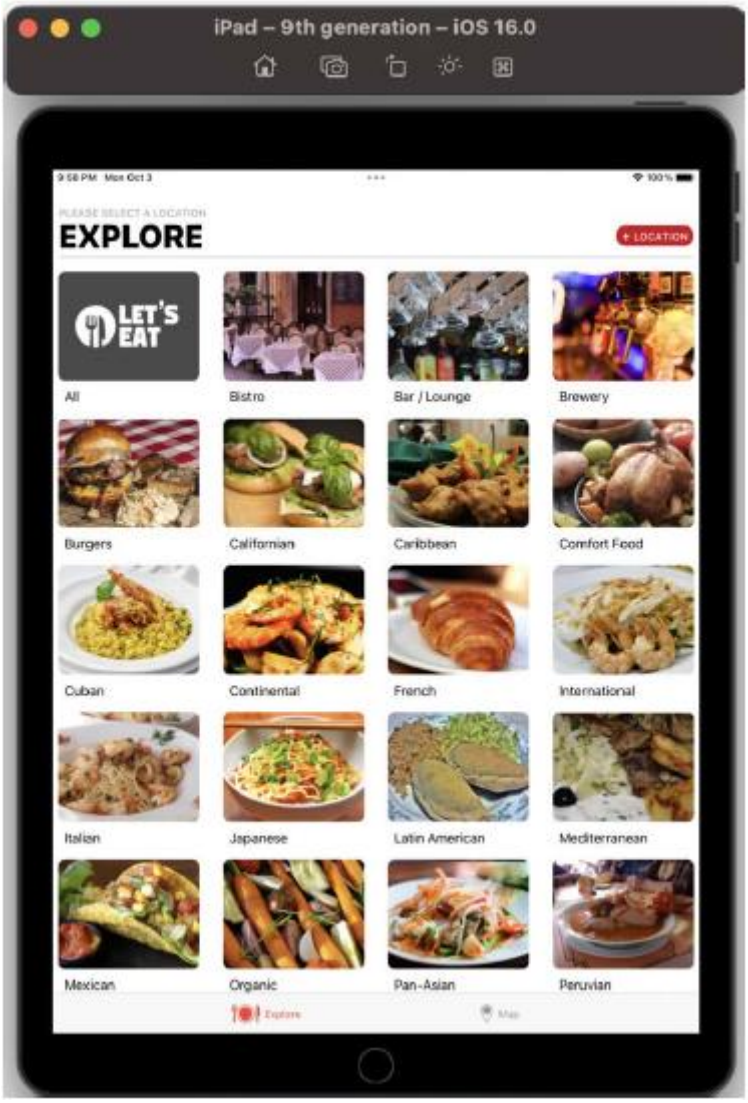
U góry ekranu zobaczysz komunikat Wybierz lokalizację dużymi literami, a przyciski Anuluj i Gotowe są teraz czerwone. Świetnie! Zakończyłeś porządkowanie projektu aplikacji na iPhone'a. Cztery wspomniane wcześniej problemy zostały rozwiązane i ekrany Twojej aplikacji wyglądają teraz dokładnie tak, jak ekrany pokazane w przewodniku po aplikacji. Jak widać, nawet drobne zmiany mogą sprawić, że Twoja aplikacja będzie bardziej atrakcyjna wizualnie. Do tej pory uruchamiałeś swoją aplikację w symulatorze iPhone'a. W następnej sekcji uruchomisz aplikację w symulatorze iPada, aby zobaczyć, jakie zmiany są wymagane. Następnie zmodyfikujesz aplikację tak, aby interfejs użytkownika wykorzystywał większy ekran iPada.

Spraw, aby Twoja aplikacja działała na wszystkich urządzeniach z systemem iOS

Zanim będziesz mógł utworzyć aplikację na Maca z istniejącej aplikacji na iOS, musisz zmodyfikować interfejs użytkownika, aby działał z iPadami. Aby zobaczyć, jakie zmiany należy wprowadzić, zbudujesz i uruchomisz aplikację na symulatorze iPada. Wykonaj następujące kroki:

1. Zamknij symulator, jeśli jest uruchomiony. Wybierz iPada (9. generacji) z listy symulatorów w menu Schemat i uruchom aplikację:

2. Uruchomi się symulator iPada i pojawi się, jak pokazano na poniższym zrzucie ekranu:



Jak widać, widok kolekcji na ekranie Eksploruj automatycznie zajmuje całą szerokość ekranu, a komórki widoku kolekcji mają taki sam rozmiar jak na iPhonie. Chociaż możesz używać dokładnie tego samego interfejsu użytkownika zarówno dla iPhone'a, jak i iPada, byłoby lepiej, gdybyś mógł dostosować go do każdego urządzenia. Aby to zrobić, dodasz kod, dzięki któremu aplikacja będzie mogła zidentyfikować typ urządzenia, na którym działa. Następnie zaktualizujesz interfejs użytkownika aplikacji, aby dopasować go do większego ekranu iPada i sprawisz, że aplikacja będzie automatycznie przełączać

interfejs użytkownika w zależności od typu urządzenia. Zobaczmy, jak sprawić, by aplikacja wykrywała typ urządzenia, na którym działa, w następnej sekcji.

Identyfikacja typu urządzenia

Musisz dodać trochę kodu do swojej aplikacji, aby wiedziała, na jakim urządzeniu jest uruchomiona. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Misc i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift i kliknij Dalej.

3. Nazwij ten plik Urządzenie. Kliknij Utwórz. Plik urządzenia pojawi się w nawigatorze projektu.

4. Zmodyfikuj plik zgodnie z ilustracją, aby utworzyć wyliczenie urządzeń:

```
import UIKit

enum Device {

static var isPhone: Bool {

UIDevice.current.userInterfaceIdiom ==

.phone

}

static var isPad: Bool {

UIDevice.current.userInterfaceIdiom ==

.pad

}

}
```

W tym przypadku zamiast klasy lub struktury używane jest wyliczenie, ponieważ nie można przypadkowo utworzyć jego instancji. Klasa UIDevice reprezentuje urządzenie, na którym działa aplikacja. UIDevice.bieżący. userInterfaceIdiom zwraca .phone, jeśli aplikacja działa na dowolnym iPhonie, i zwraca .pad, jeśli aplikacja działa na dowolnym iPadzie. Zatem isPhone zwraca wartość true, gdy aplikacja jest uruchomiona na iPhonie, a isPad zwraca wartość true, gdy aplikacja jest uruchomiona na iPadzie. Oprócz typu urządzenia należy również wziąć pod uwagę jego orientację. Na przykład iPhone w orientacji poziomej jest szerszy niż iPhone w orientacji pionowej, mimo że jest to ten sam iPhone. W następnej sekcji nauczymy się, jak radzić sobie z orientacją urządzenia za pomocą klas rozmiarów.

Zrozumienie klas wielkości

Chociaż możesz teraz określić, na jakim urządzeniu działa Twoja aplikacja, musisz także wziąć pod uwagę wpływ orientacji urządzenia na interfejs użytkownika. Może to być trudne, ponieważ istnieje wiele różnych rozmiarów ekranów, zarówno w orientacji pionowej, jak i poziomej. Aby to ułatwić, zamiast korzystać z fizycznej rozdzielczości urządzenia, użyjesz klas wielkości. Klasy wielkości to cechy, które są automatycznie przypisywane do widoku. Zdefiniowano dwie klasy opisujące wysokość i szerokość widoku: regularny (przestrzeń ekspansywna) i kompaktowy (przestrzeń ograniczona). Przyjrzyjmy się klasom rozmiarów, aby uzyskać widok pełnoekranowy na różnych urządzeniach:

Device	Portrait	Landscape
iPad	Regular width Regular height	Regular width Regular height
iPhone 11 Pro Max	Compact width Regular height	Regular width Compact height
iPhone SE (2nd generation)	Compact width Regular height	Compact width Compact height

Projektując interfejs użytkownika, będziesz musiał wziąć pod uwagę nie tylko typ urządzenia, ale także klasę wielkości. W następnej sekcji dowiesz się, jak ustawić rozmiar komórki widoku kolekcji na podstawie urządzenia i klasy rozmiaru.

Aktualizacja ekranu Eksploruj

Załóżmy, że na ekranie Eksploruj zdecydowałeś się wyświetlić trzy kolumny na iPadzie, dwie kolumny dla klasy rozmiaru o kompaktowej szerokości i trzy kolumny dla klasy rozmiaru o regularnej szerokości. Dodasz metody ustawiania rozmiaru komórki widoku kolekcji w zależności od urządzenia i orientacji. Wykonaj następujące kroki:

1. Kliknij plik ExploreViewController w nawigatorze projektu i zmodyfikuj metodę inicjalizacji() wewnątrz rozszerzenia prywatnego w następujący sposób:

```
func initialize() {

manager.fetch()

setupCollectionView()

}
```

Zobaczysz błąd, ponieważ funkcja setupCollectionView() nie została jeszcze zadeklarowana ani zdefiniowana. Zrobisz to dalej.

2. Metoda setupCollectionView() zostanie użyta w celu dodania instancji UICollectionViewFlowLayout do widoku kolekcji na ekranie Eksploruj. Zadeklaruj i zdefiniuj tę metodę po metodzie inicjalizacji():

```
func setupCollectionView() {

let flow = UICollectionViewFlowLayout()

flow.sectionInset = UIEdgeInsets(top: 7, left: 7,

bottom: 7, right: 7)

flow.minimumInteritemSpacing = 0

flow.minimumLineSpacing = 7

collectionView.collectionViewLayout = flow

}
```

Ta metoda tworzy instancję klasy UICollectionViewFlowLayout, ustawia wszystkie wstawki krawędzi dla widoku kolekcji na 7 punktów, ustawia minimalne odstępy między elementami na 0 punktów, ustawia minimalny odstęp między wierszami na 7 punktów i przypisuje go do widoku kolekcji. Pamiętaj, że początkowo ustawiasz te wartości dla widoku kolekcji za pomocą Inspektora rozmiaru w rozdziale 11, Tworzenie interfejsu użytkownika.

3. Dodaj rozszerzenie zawierające metody, które ustawią rozmiar komórek widoku kolekcji oraz nagłówek sekcji widoku kolekcji po zamykającym nawiasie klamrowym:

```
extension ExploreViewController: UICollectionViewDelegateFlowLayout {

func collectionView(_ collectionView:

UICollectionView, layout collectionViewLayout:

UICollectionViewLayout, sizeForItemAt indexPath:

IndexPath) -> CGSize {

var columns: CGFloat = 2

if Device.isPad ||

(traitCollection.horizontalSizeClass !=

.compact) {

columns = 3

}

let viewWidth = collectionView.frame.size.width

let inset = 7.0

let contentWidth = viewWidth - inset *

(columns + 1)

let cellWidth = contentWidth / columns

let cellHeight = cellWidth

return CGSize(width: cellWidth, height:

cellHeight)

}

func collectionView(_ collectionView:

UICollectionView, layout collectionViewLayout:

UICollectionViewLayout,

referenceSizeForHeaderInSection section: Int) ->

CGSize {

CGSize(width: collectionView.frame.width,
```

height: 100)

}

}

Metody te są zadeklarowane w protokole UICollectionViewDelegateFlowLayout i definiują rozmiar elementu i odstępy w widoku kolekcji. Zastąpią one ustawienia w Inspektorze rozmiaru. Podzielmy je:

func collectionView(_ collectionView: UICollectionView, layout

collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath:

IndexPath) -> CGSize {

Ta metoda zwraca instancję CGSize, na którą powinien być ustawiony rozmiar komórki widoku kolekcji.

var columns: CGFloat = 2

Zmienna kolumn określa, ile kolumn pojawia się na ekranie i początkowo jest ustawiona na 2.

if Device.isPad || (traitCollection.horizontalSizeClass != .compact) {

Sprawdza to, czy aplikacja działa na iPadzie, czy też posiada właściwość HorizonSizeClass nie jest kompaktowy.

columns = 3

Jeśli aplikacja działa na iPadzie lub klasa rozmiaru poziomego nie jest .compact, ustaw kolumny na 3.

let viewWidth = collectionView.frame.size.width

Pobiera szerokość ekranu i przypisuje ją do viewWidth.

let inset = 7.0

let contentWidth = viewWidth - inset * (columns + 1)

Odejmuje to przestrzeń używaną na wstawki krawędziowe, aby można było określić rozmiar komórki.

let cellWidth = contentWidth / columns

Pobiera szerokość komórki, dzieląc contentWidth przez kolumny i przypisując ją do cellWidth.

let cellHeight = cellWidth

Ustawia to wysokość komórki na taką samą, jak szerokość komórki.

return CGSize(width: cellWidth, height: cellHeight)

}

Zwraca to rozmiar komórki.

Załóżmy, że używasz iPhone'a 13 Pro Max w trybie portretowym. kolumn jest ustawiona na 2. viewWidth zostanie przypisana szerokość ekranu iPhone'a, która wynosi 414 punktów. contentWidth jest ustawione na 414 - (7 x 3) = 393. cellWidth jest ustawione na contentWidth / kolumny = 196,5, a cellHeight jest ustawione na cellWidth, więc zwrócony CGSize będzie wynosić (196,5, 196,5), umożliwiając zmieszczenie dwóch komórek w wiersz. Kiedy obrócisz tego samego iPhone'a do trybu poziomego, kolumny zostaną ustawione na 3. viewWidth zostanie przypisana wysokość ekranu

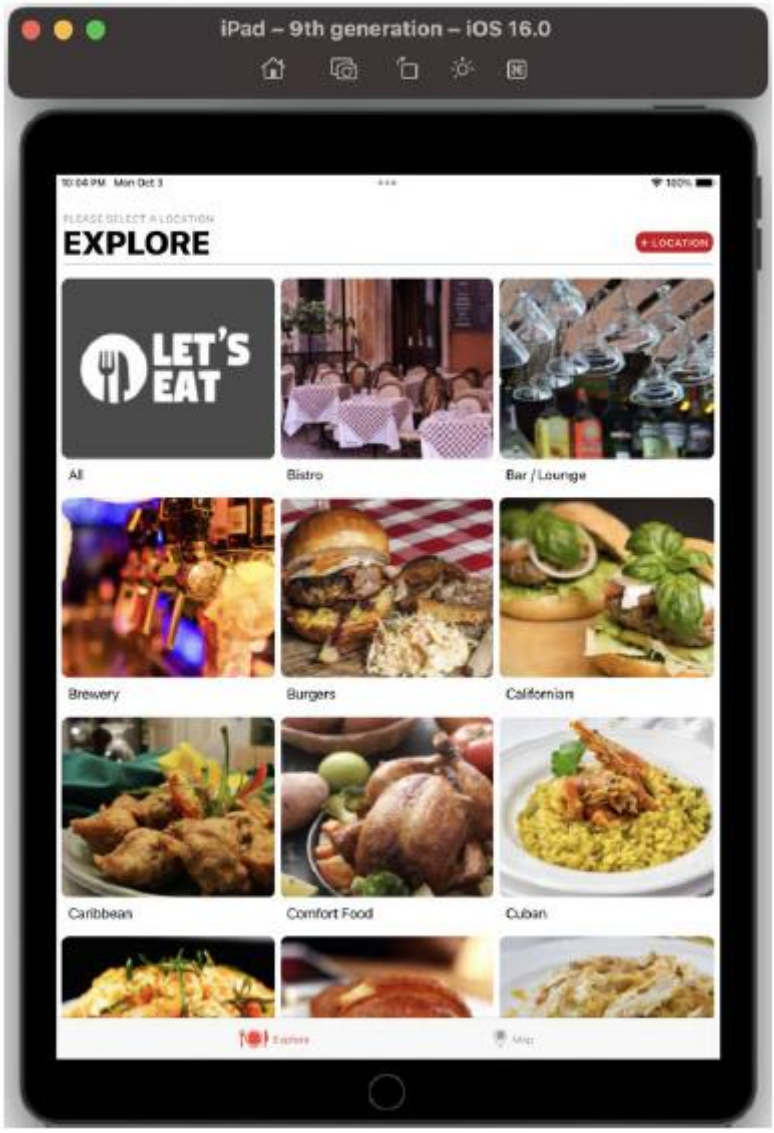
iPhone'a, która wynosi 896 punktów. contentWidth jest ustawione na 896 - (7 x 4) = 868. cellWidth jest ustawione na contentWidth / kolumny = 289,3, a cellHeight jest ustawione na cellWidth, więc zwrócony CGSize będzie wynosić (289,3, 289,3), umożliwiając zmieszczenie trzech komórek w wiersz.

func collectionView(_ collectionView: UICollectionView, layout

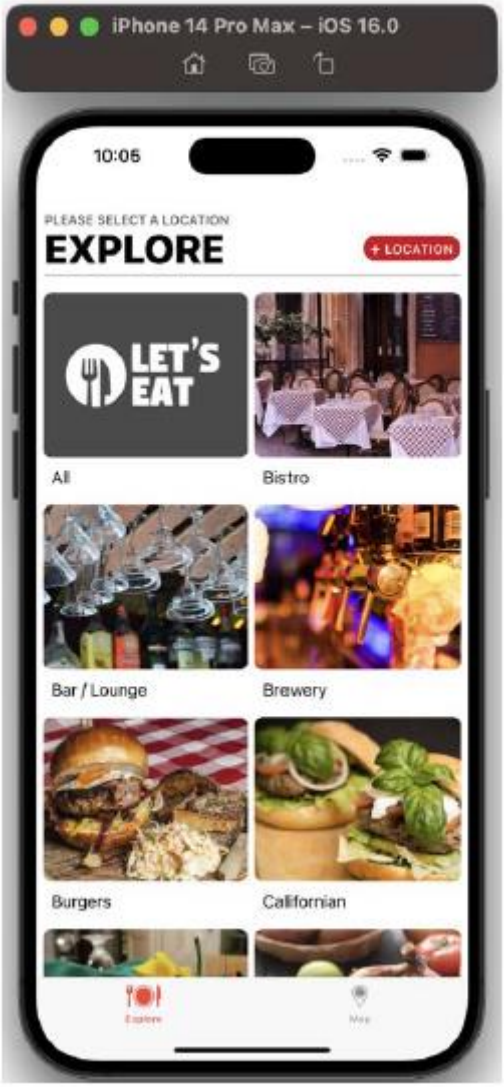
collectionViewLayout: UICollectionViewLayout, referenceSizeForHeaderInSection

section: Int) -> CGSize {

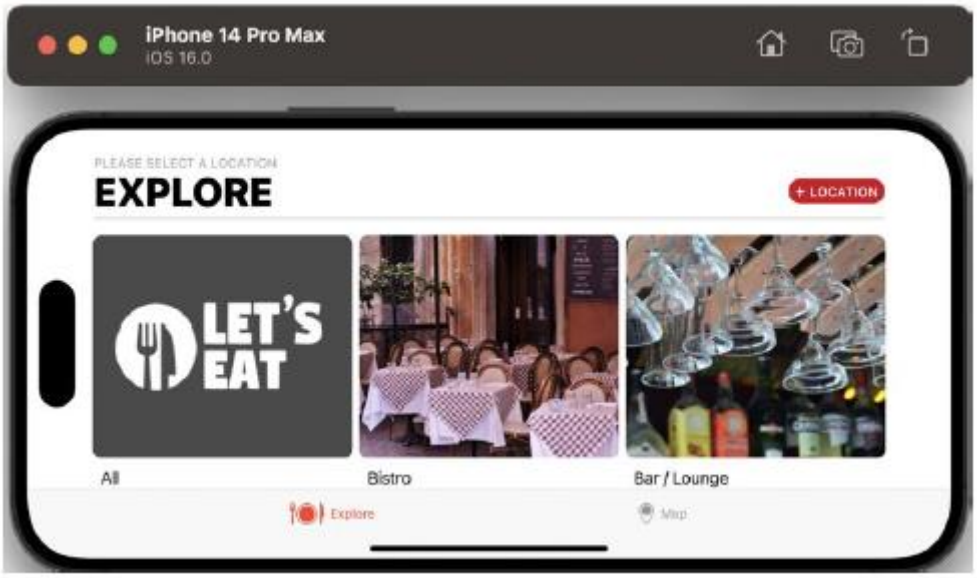
Ta metoda zwraca rozmiar, jaki powinien być ustawiony nagłówek sekcji widoku kolekcji. CGSize(szerokość: kolekcjaView.frame.width, wysokość: 100) Szerokość nagłówka sekcji widoku kolekcji będzie zależeć od orientacji urządzenia, ale wysokość będzie zawsze wynosić 100. Zbuduj i uruchom swoją aplikację na symulatorze iPada. Powinieneś zobaczyć trzy kolumny:



Zbuduj i uruchom swoją aplikację na symulatorze iPhone'a 14 Pro Max; powinieneś zobaczyć dwie kolumny:



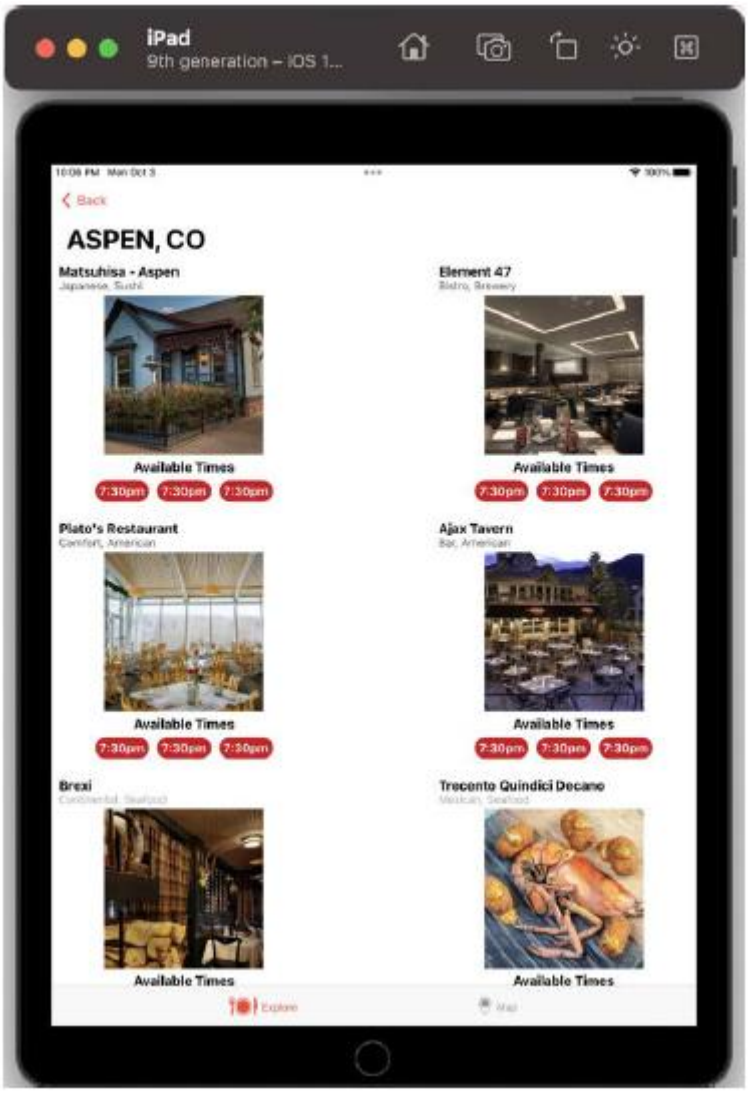
Wybierz urządzenie | Obróć w lewo w menu symulatora; zobaczysz trzy kolumny:



Wybierz urządzenie | Obróć w prawo w menu symulatora, aby powrócić do orientacji pionowej. Zakończyłeś modyfikowanie ekranu Eksploruj. Zobaczmy teraz, jak dostosować ekran Lista restauracji do innego typu i orientacji urządzenia. W następnej sekcji zmodyfikujesz klasę RestaurantListViewController.

Aktualizowanie ekranu listy restauracji

Zmodyfikowałeś już ekran Eksploruj, aby automatycznie dostosowywał się do urządzenia, na którym działa Twoja aplikacja. Teraz zrobisz to samo dla ekranu Lista restauracji. Jeśli zbudujesz i uruchomisz na symulatorze iPada, ekran Lista restauracji będzie wyglądał tak:



Jak widać, są tylko dwie kolumny i pomiędzy nimi znajduje się duża biała przestrzeń. Załóżmy, że chcesz mieć trzy kolumny na iPadzie, jedną kolumnę dla klasy kompaktowej szerokości i 2 kolumny dla

klasa rozmiaru o regularnej szerokości. Wykonaj następujące kroki:

1. Kliknij plik RestaurantListViewController w folderze Restaurants w nawigatorze projektu. Utwórz metodę inicjalizacji() wewnątrz rozszerzenia prywatnego, zanim cały inny kod znajdujący się już w rozszerzeniu:

func initialize() {

```
createData()

setupTitle()

setupCollectionView()

}
```

Obie metody createData() i setupTitle() są wywoływane w funkcji viewDidAppear(), ale później zmodyfikujesz metodę viewDidAppear(), aby zamiast tego wywołała inicjalizację(). Zobaczysz błąd, ponieważ metoda setupCollectionView() nie została jeszcze zadeklarowana ani zdefiniowana.

2. Zadeklaruj i zdefiniuj metodę setupCollectionView() w rozszerzeniu prywatnym po metodzie inicjalizacji():

```
func setupCollectionView() {

let flow = UICollectionViewFlowLayout()

flow.sectionInset = UIEdgeInsets(top: 7, left: 7,

bottom: 7, right: 7)

flow.minimumInteritemSpacing = 0

flow.minimumLineSpacing = 7

collectionView.collectionViewLayout = flow

}
```

Podobnie jak poprzednio, setupCollectionView() tworzy instancję klasy UICollectionViewFlowLayout, konfiguruje ją i przypisuje do widoku kolekcji.

3. Po zamykającym nawiasie klamrowym dodaj rozszerzenie zawierające metody UICollectionViewDelegateFlowLayout:

```
UICollectionViewDelegateFlowLayout {

func collectionView(_ collectionView:

UICollectionView, layout collectionViewLayout:

UICollectionViewLayout, sizeForItemAt indexPath:

IndexPath) -> CGSize {

var columns: CGFloat = 0

if Device.isPad {

columns = 3

} else {

columns =

traitCollection.horizontalSizeClass

== .compact ? 1 : 2
```

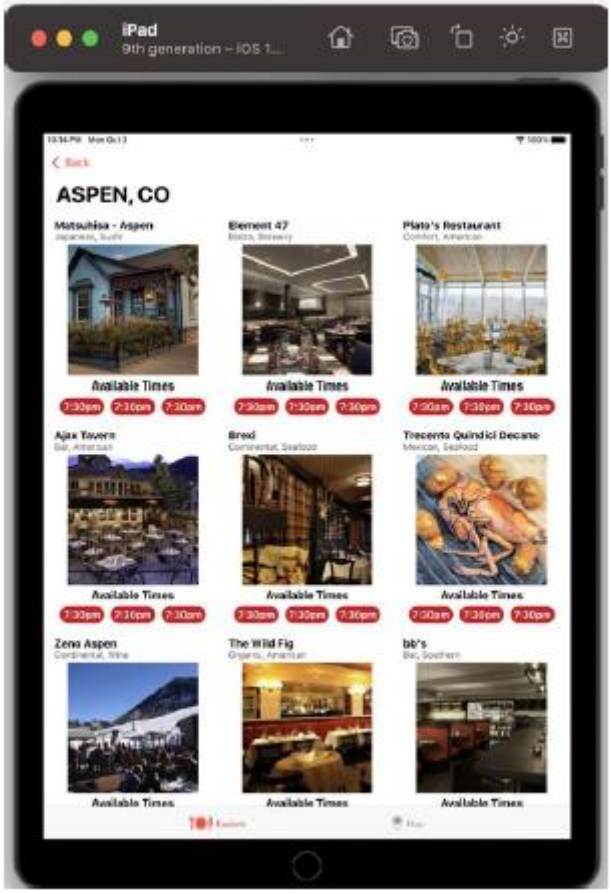
```
}

let viewWidth = collectionView.frame.size.width

let inset = 7.0

let contentWidth = viewWidth - inset *

(columns + 1)

let cellWidth = contentWidth / columns

let cellHeight = 312.0

return CGSize(width: cellWidth, height:

cellHeight)

}

}
```

Zaimplementowana tutaj metoda CollectionView(_:layout:sizeForItemAt:) działa prawie dokładnie tak samo, jak implementacja w klasie ExploreViewController, ale wartość cellHeight jest ustawiona na 312 punktów, a nie na cellWidth. Pamiętaj, że jeśli nie uruchamiasz aplikacji na iPadzie, kolumny zostaną ustawione na 1 dla klasy rozmiaru o kompaktowej szerokości i na 2 dla klasy rozmiaru o zwykłej szerokości.

4. Zaktualizuj funkcję viewDidAppear(), usuwając wywołania metod createData() i setupTitle() oraz dodając wywołanie metody inicjalizacji():

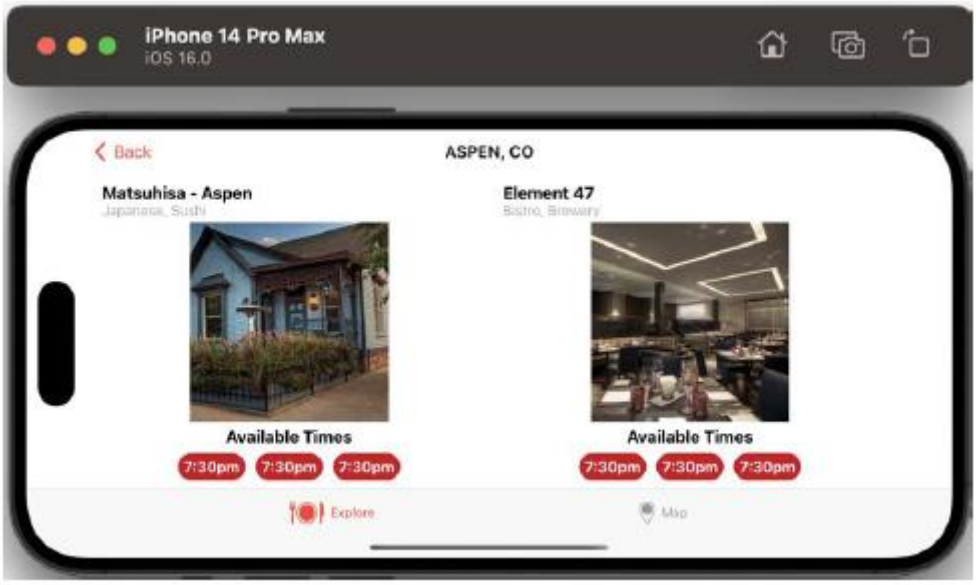
```
override func viewDidAppear(_ animated: Bool) {

super.viewDidAppear(animated)

initialize()

}
```

Zbuduj i uruchom aplikację na symulatorze iPada, jak pokazano:



Są teraz trzy kolumny i szeroka biała szczelina zniknęła. Teraz zbuduj i uruchom swoją aplikację na symulatorze iPhone'a 14 Pro Max. Powinien wyświetlić jedną kolumnę:

Wybierz urządzenie | Obróć w lewo w menu symulatora, a powinieneś zobaczyć dwie kolumny:

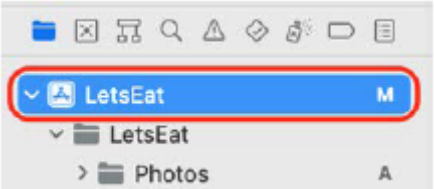


Wybierz urządzenie | Obróć w prawo w menu symulatora i zamknij symulator. Ekrany Eksploruj i ekran Lista restauracji zostały zaktualizowane i teraz Twoja aplikacja wygląda dobrze na iPadzie. Jest to teraz idealny kandydat do przekształcenia w aplikację na komputery Mac. W następnej sekcji zobaczymy, jak możesz zbudować aplikację na Maca na podstawie istniejącej aplikacji na iPada.

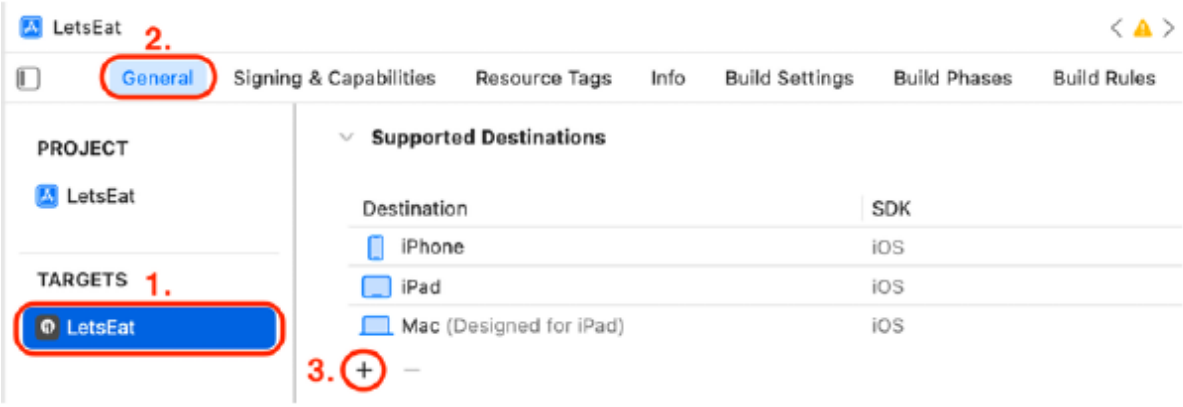
Aktualizacja aplikacji do pracy na macOS

Zmodyfikowałeś ekrany swojej aplikacji, aby działały dobrze na wszystkich urządzeniach z systemem iOS. Teraz dowiesz się, jak uruchomić aplikację na komputerze Mac. Podczas WWDC2022 firma Apple zaktualizowała program Mac Catalyst, który umożliwia tworzenie aplikacji dla komputerów Mac na podstawie istniejącej aplikacji na iPada z optymalizacjami specyficznymi dla komputerów Mac. Jak zobaczysz, obie aplikacje będą korzystać z tego samego projektu i kodu źródłowego. Zanim zaczniesz, pamiętaj, że działa to tylko wtedy, gdy masz bezpłatne lub płatne konto programisty Apple. Jeśli używasz plików projektu w folderze Chapter23 pobranym z GitHub pod adresem https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-Seventh-Edition, musisz wyznaczyć zespół programistów Twojej aplikacji, aby działa na komputerze Mac. Wykonaj następujące kroki:

1. Wybierz swój projekt w nawigatorze projektu:



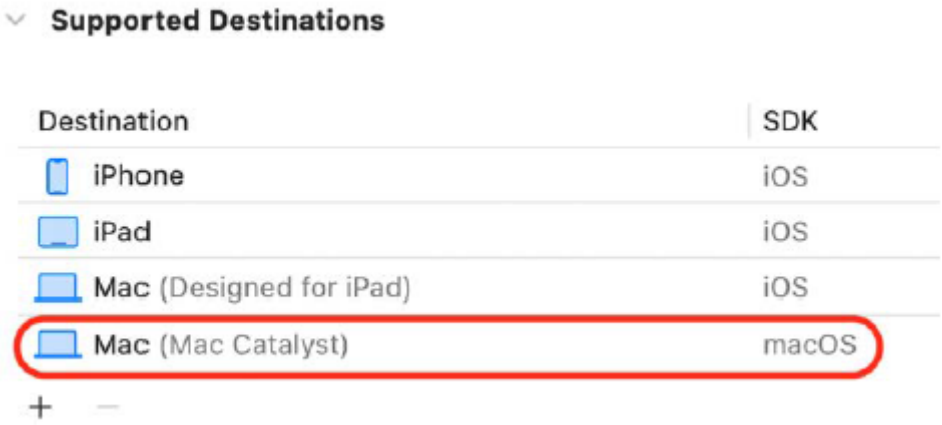
2. Wybierz opcję LetsEat w obszarze CELE. Na karcie Ogólne kliknij przycisk + w sekcji Obsługiwane miejsca docelowe:



3. Wybierz Maca | Mac Catalyst z wyskakującego menu:

4. W opcji Włącz obsługę Mac Catalyst? oknie dialogowym kliknij opcję Włącz:

5. Pamiętaj, że Mac (Mac Catalyst) jest teraz obsługiwanym miejscem docelowym:



Twoja aplikacja zostanie przekompilowana tak, aby działała na komputerze Mac. Zwróć uwagę na miejsce docelowe komputera Mac (Zaprojektowane dla iPada). Jeśli masz komputer Mac Apple Silicon, możesz uruchamiać niezmodyfikowane aplikacje na iPada natywnie na komputerze Mac.

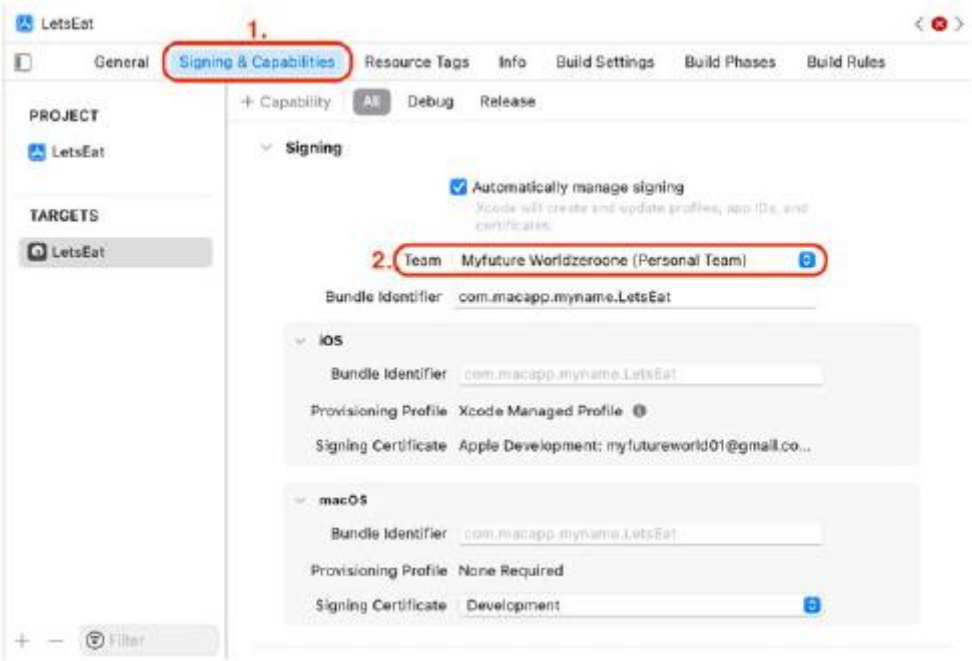
6. W menu Schemat ustaw komputer Mac jako miejsce docelowe uruchamiania:

7. Zbuduj i uruchom swoją aplikację. Jeśli kompilacja projektu nie powiedzie się, kliknij przycisk Nawigator problemów i sprawdź komunikat o błędzie:

Jeśli widzisz pokazany tutaj błąd, oznacza to, że potrzebujesz bezpłatnego lub płatnego konta programisty, aby uruchomić aplikację na rzeczywistym sprzęcie.

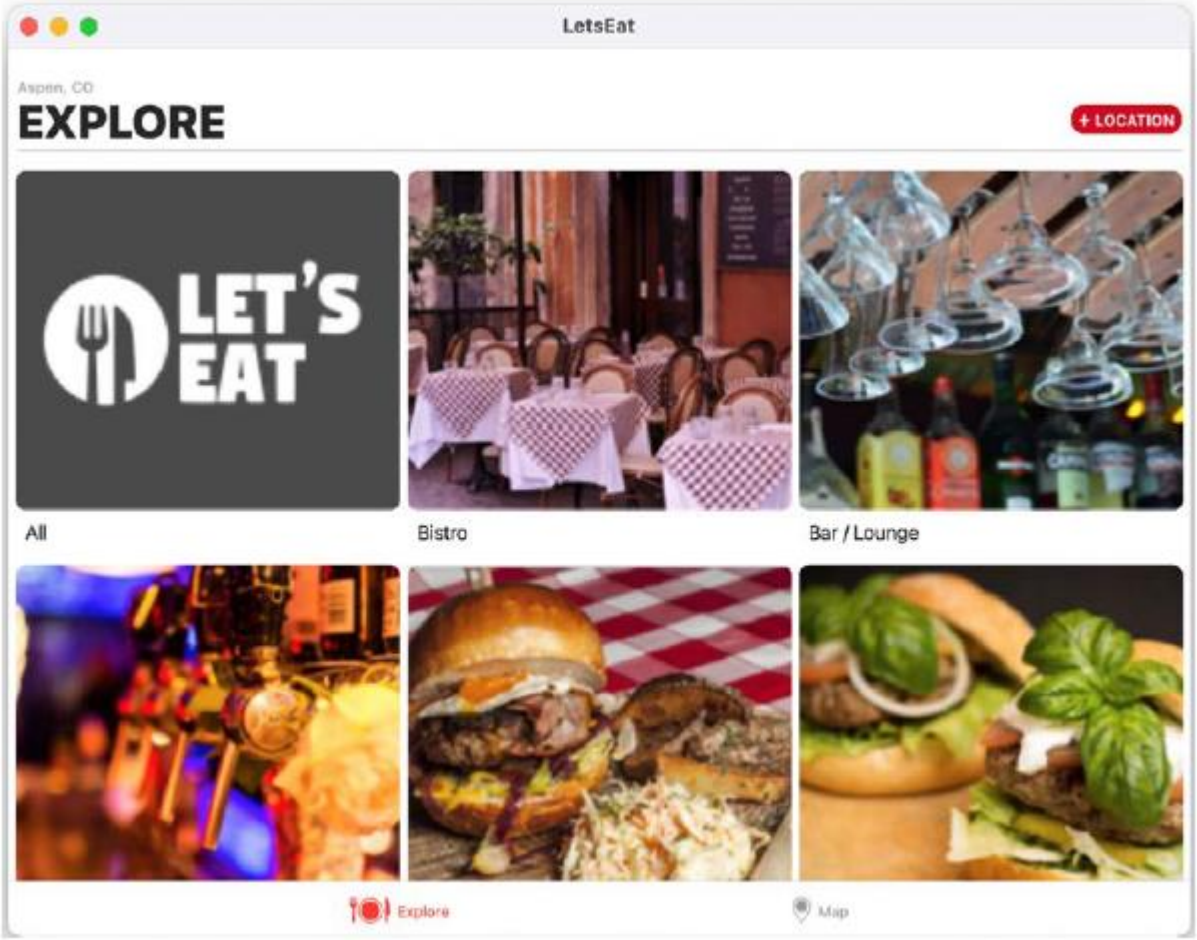
8. Sprawdź, czy Twoje konto programisty zostało dodane do Xcode w Xcode | Ustawienia | Konta.

9. Kliknij kartę Podpisywanie i możliwości. Wybierz swoje płatne lub bezpłatne konto programisty w menu rozwijanym Zespół:



Jeśli nadal widzisz błędy, spróbuj zmienić wartość identyfikatora pakietu na unikalną wartość i spróbuj najpierw uruchomić aplikację na urządzeniu z systemem iOS.

10. Zbuduj i uruchom ponownie, a aplikacja powinna działać na komputerze Mac:



Twoja aplikacja działa teraz na komputerze Mac! Wspaniały!

Streszczenie

Nauczyłeś się, jak zbudować aplikację na Maca na podstawie istniejącej aplikacji na iOS. Zacząłeś od udoskonalenia interfejsu użytkownika aplikacji działającej na iPhonie. Następnie dodałeś kod, dzięki któremu aplikacja wykryje urządzenie, na którym jest uruchomiona, i zmodyfikowałeś ekrany aplikacji, aby działały na wszystkich urządzeniach z systemem iOS. Na koniec użyłeś narzędzia Mac Catalyst do zbudowania aplikacji na komputer Mac z aplikacji na iPada. Twoja aplikacja działa teraz świetnie na iPhonie, iPadzie i komputerze Mac. Możesz teraz sprawić, aby istniejące aplikacje na iPhone'a działały dobrze na iPadzie, a także tworzyć aplikacje na Maca z aplikacji na iPada. Jak widzieliście, gdy już masz aplikację na iPhone'a, możesz przy stosunkowo niewielkim wysiłku sprawić, by działała na iPadzie i komputerze Mac. W następnym rozdziale poznasz zupełnie nowy sposób tworzenia aplikacji przy użyciu SwiftUI, nowoczesnego sposobu pisania aplikacji na dowolną platformę Apple.

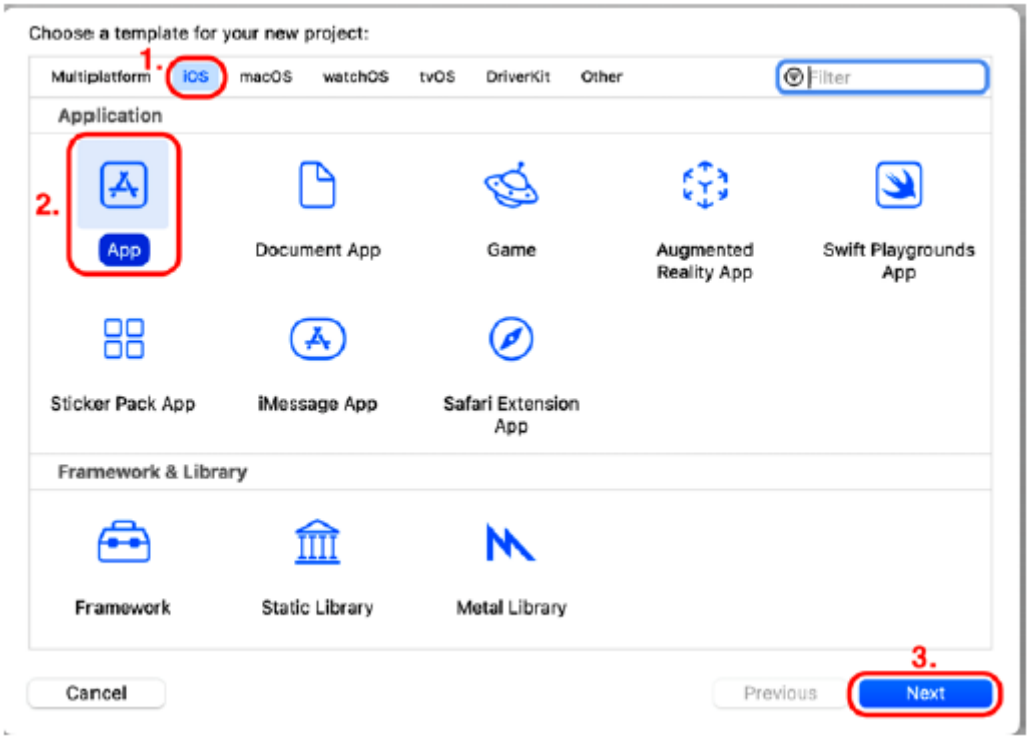
Pierwsze kroki z SwiftUI

Stworzyłeś interfejs użytkownika (UI) aplikacji Let's Eat za pomocą scenorysów. Proces polegał na przeciąganiu obiektów reprezentujących widoki do scenorysu i tworzeniu punktów sprzedaży w kontrolerze widoku plików i połączyć je ze sobą. W tym rozdziale skupimy się na SwiftUI, łatwym i innowacyjnym sposobie tworzenia aplikacji na wszystkich platformach Apple. Zamiast określać interfejs użytkownika za pomocą scenorysów, SwiftUI używa deklaratywnej składni Swift i współpracuje z nowymi narzędziami do projektowania Xcode, aby zapewnić synchronizację kodu i projektu. Funkcje takie jak typ dynamiczny, tryb ciemny, lokalizacja i dostępność są obsługiwane automatycznie. W tym rozdziale zbudujesz uproszczoną wersję aplikacji Let's Eat przy użyciu SwiftUI. Ta aplikacja będzie zawierać tylko ekrany Lista restauracji i Szczegóły restauracji. Ponieważ pisanie aplikacji za pomocą SwiftUI znacznie różni się od tego, co już zrobiłeś, nie będziesz modyfikować projektu LetsEat, nad którym pracowałeś. Zamiast tego utworzysz nowy projekt SwiftUI Xcode. Zaczniesz od dodania i skonfigurowania widoków SwiftUI, aby utworzyć ekran Lista restauracji. Następnie dodasz obiekty modelu do swojej aplikacji i skonfigurujesz nawigację pomiędzy ekranami Lista restauracji i Szczegóły restauracji. Następnie dowiesz się, jak korzystać razem z widoków UIKit i SwiftUI, dodając i konfigurując widok mapy dla ekranu szczegółów restauracji. Na koniec utworzysz ekran szczegółów restauracji. Pod koniec tego rozdziału dowiesz się, jak zbudować aplikację SwiftUI, która odczytuje obiekty modelu, prezentuje je na liście i umożliwia nawigację do drugiego ekranu zawierającego widok mapy. Następnie możesz wdrożyć to w swoich własnych projektach.

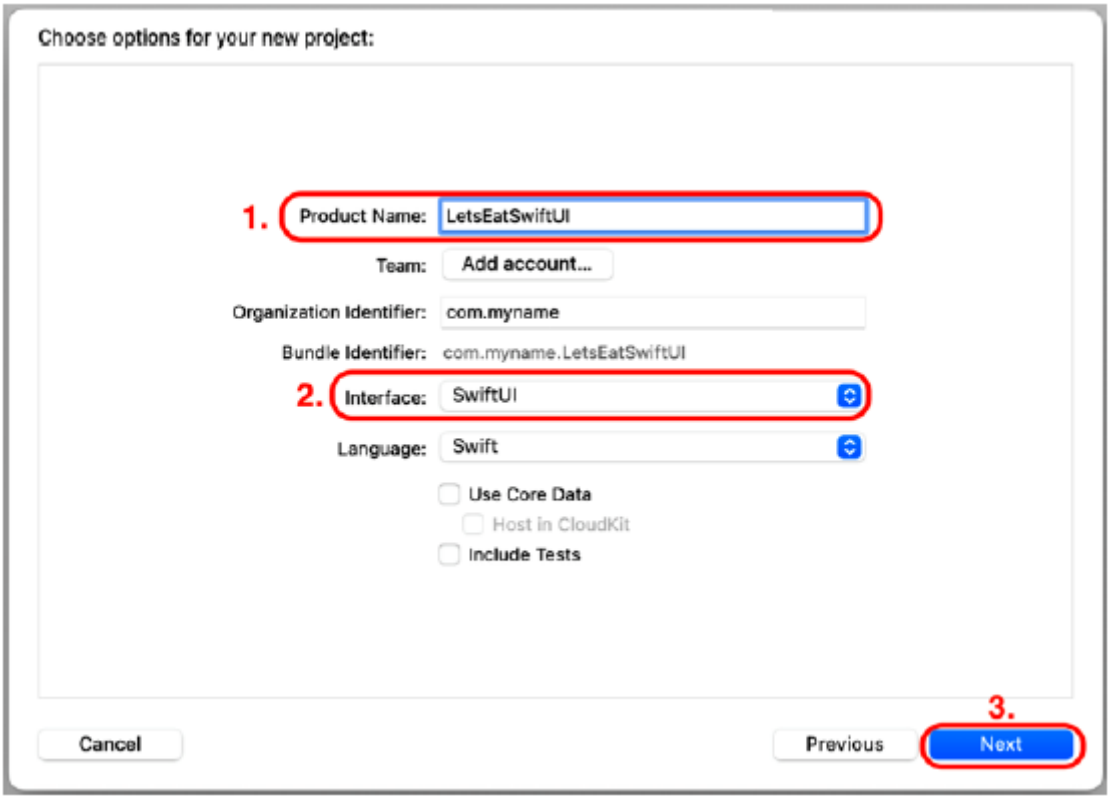
Zacznijmy od utworzenia nowego projektu Xcode SwiftUI. Wykonaj następujące kroki:

1. Utwórz nowy projekt Xcode.

2. Kliknij iOS. Wybierz szablon aplikacji, a następnie kliknij Dalej:



3. Pojawi się ekran Wybierz opcje dla nowego projektu::



Skonfiguruj ten ekran w następujący sposób:

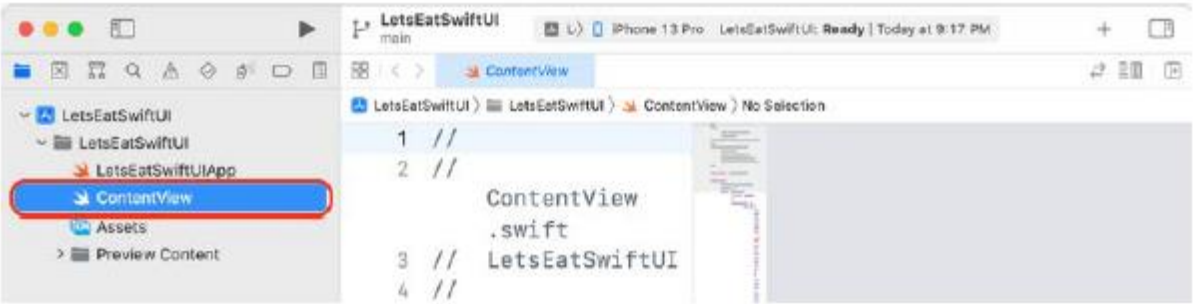
• Nazwa produktu: LetsEatSwiftUI

• Interfejs: SwiftUI

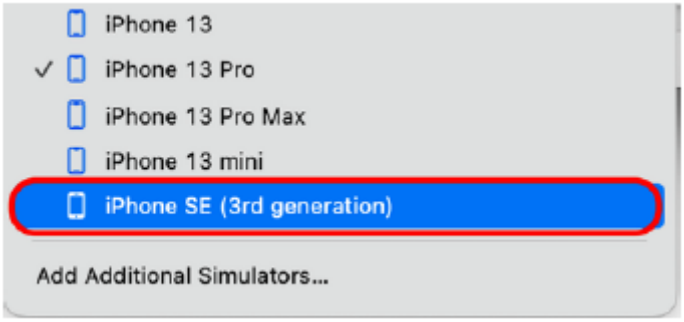
Pozostałe ustawienia powinny być już ustawione. Upewnij się, że wszystkie pola wyboru są odznaczone. Po zakończeniu kliknij Dalej.

4. Wybierz lokalizację, w której chcesz zapisać projekt LetsEatSwiftUI i kliknij Utwórz.

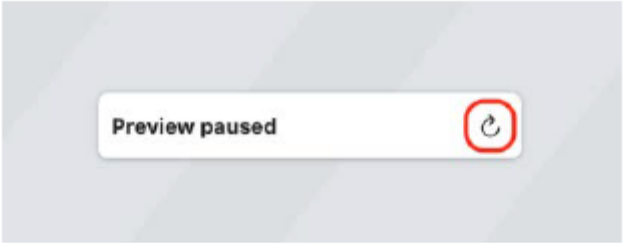
5. Twój projekt pojawi się na ekranie z wybranym plikiem ContentView w nawigatorze projektu. Zawartość tego pliku zobaczysz po lewej stronie obszaru Edytora, a płótno zawierające podgląd po prawej stronie:



6. Plik ContentView zawiera kod, który wygeneruje początkowy widok Twojej aplikacji. Kliknij menu Schemat i wybierz iPhone SE (3. generacji), aby podgląd widoku został wyświetlony na ekranie iPhone'a SE (3. generacji):



7. Jeśli na obszarze roboczym zobaczysz pole Podgląd wstrzymany, kliknij przycisk Wznów, aby wygenerować podgląd:



8. Sprawdź, czy na kanwie wyświetlany jest podgląd aplikacji:

Jeśli płótno nie jest widoczne, wybierz opcję Płótno z menu Dostosuj opcje edytora, aby je wyświetlić. Jeśli używasz MacBooka, możesz użyć gestu szczypania na gładziku, aby zmienić rozmiar symulowanego obrazu.

9. Jeśli potrzebujesz więcej miejsca do pracy, kliknij przyciski Nawigatora i Edytora, aby ukryć obszary Nawigatora i Edytora, a następnie przeciągnij granicę w obszarze Edytora, aby zmienić rozmiar obszaru roboczego:

Przyjrzyjmy się teraz plikowi ContentView. Ten plik zawiera dwie struktury: ContentView i ContentView_Previews. Struktura ContentView opisuje zawartość i układ widoku oraz jest zgodna z protokołem View. Struktura ContentView_Previews deklaruje podgląd struktury ContentView. Podgląd zostanie wyświetlony na kanwie. Aby zobaczyć to w akcji, zmień tekst Hello World na Lets Eat, jak pokazano:

```
struct ContentView: View {

var body: some View {

VStack {

Image(systemName: "globe")

.imageScale(.large)

.foregroundColor(

.accentColor)

Text("Lets Eat")

}

.padding()

}

}
```

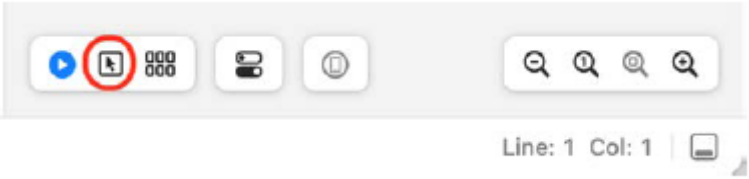
Podgląd w obszarze roboczym jest aktualizowany w celu odzwierciedlenia wprowadzonych zmian:

Udało Ci się stworzyć swój pierwszy projekt SwiftUI! Stwórzmy teraz ekran Lista Restauracji zaczynając od widoku, który wyświetli dane konkretnej restauracji.

Tworzenie ekranu Lista restauracji

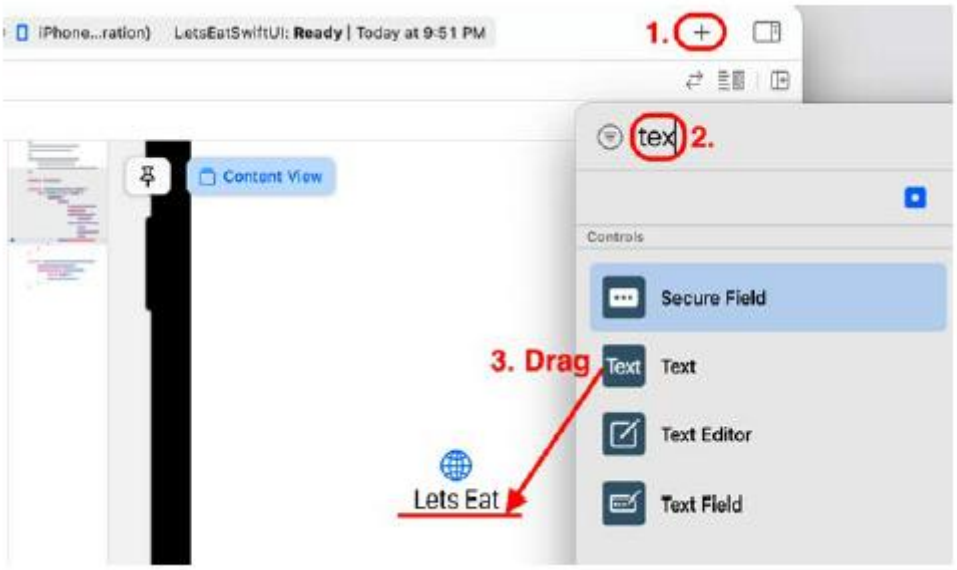
Podczas korzystania ze scenorysów atrybuty widoku modyfikuje się za pomocą Inspektora atrybutów. W SwiftUI możesz modyfikować swój kod lub podgląd na kanwie. Jak widziałeś, zmiana kodu w pliku ContentView natychmiast zaktualizuje podgląd, a modyfikacja podglądu zaktualizuje kod. Dostosujmy strukturę ContentView tak, aby wyświetlała dane konkretnej restauracji. Wykonaj następujące kroki:

1. Kliknij przycisk Do wyboru na obszarze roboczym:



Umożliwia to przeciąganie i upuszczanie elementów interfejsu użytkownika z Biblioteki na kanwę.

2. Kliknij przycisk Biblioteka. Wpisz tex w polu filtru, przeciągnij widok Tekst na kanwę i upuść go pod tekstem Lets Eat:



3. Xcode automatycznie dodał kod do pliku ContentView dla tego widoku tekstowego. Sprawdź, czy Twój kod wygląda następująco:

```
struct ContentView: View {

var body: some View {

VStack{

Image(systemName: "globe")

.imageScale(.large)

.foregroundColor(

.accentColor)

Text("Lets Eat")

Text("Placeholder")
```

```
}
.padding()
}
}
```

Jak widać, po widoku tekstowym zawierającym ciąg „Lets Eat” dodano drugi widok tekstowy, a oba widoki tekstowe i obrazowe są zawarte w widoku VStack. Widok VStack zawiera podwidoki ułożone pionowo i jest podobny do widoku stosu zorientowanego pionowo.

4. Jako przykładowe dane wykorzystasz dane restauracji The Tap Trailhouse w Bostonie. Zmodyfikuj kod w widoku VStack, aby wyświetlić nazwę i kuchnię oferowaną przez restaurację The Tap Trailhouse oraz zmień położenie widoku obrazu:

```
struct ContentView: View {
var body: some View {
VStack{
Text("The Tap Trailhouse")
Text("Brewery, Burgers, American")
Image(systemName: "globe")
.imageScale(.large)
.foregroundColor(
.accentColor)
}
}
}
```

5. Sprawdź, czy zmiany zostały odzwierciedlone na podglądzie:

Należy pamiętać, że widok obrazu ma jeden parametr, nazwasystemu. Ten parametr umożliwia wybranie jednego z obrazów z biblioteki symboli SF firmy Apple. Później zastąpisz ten obraz symboli SF zdjęciem.

6. Aby zmienić wygląd tekstu, zamiast Inspektora atrybutów użyj modyfikatorów. Są to metody zmieniające wygląd lub zachowanie obiektów. Należy pamiętać, że widok obrazu ma już modyfikatory. Zaktualizuj swój kod, jak pokazano, aby ustawić styl i kolor widoków tekstu i usuń modyfikatory widoku obrazu:

struct ContentView: View {

var body: some View {

VStack{

Text("The Tap Trailhouse")

.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "globe")

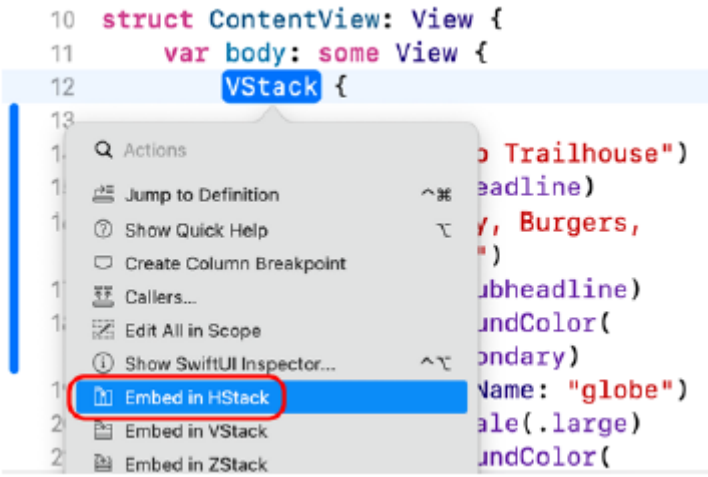
}

}

}

Zwróć uwagę na zmiany w tekście i obrazie w podglądzie.

7. Aby mieć pewność, że widok pozostanie na środku ekranu, osadzisz go w widoku HStack i dodasz obiekty Spacer po obu stronach. Widok HStack zawiera podwidoki ułożone poziomo i jest podobny do widoku stosu zorientowanego poziomo. Obiekt Spacer to elastyczna przestrzeń, która rozwija się poziomo w widoku HStack. Command + kliknij widok VStack i wybierz Osadź w HStack z wyskakującego menu:



8. Sprawdź, czy Twój kod wygląda następująco:

struct ContentView: View {

var body: some View {

HStack {

VStack{

Text("The Tap Trailhouse")

```
.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "globe")

}

}

}

}
```

9. Dodaj dwa obiekty Spacer do widoku HStack, jak pokazano, aby wyśrodkować widok w poziomie na ekranie:

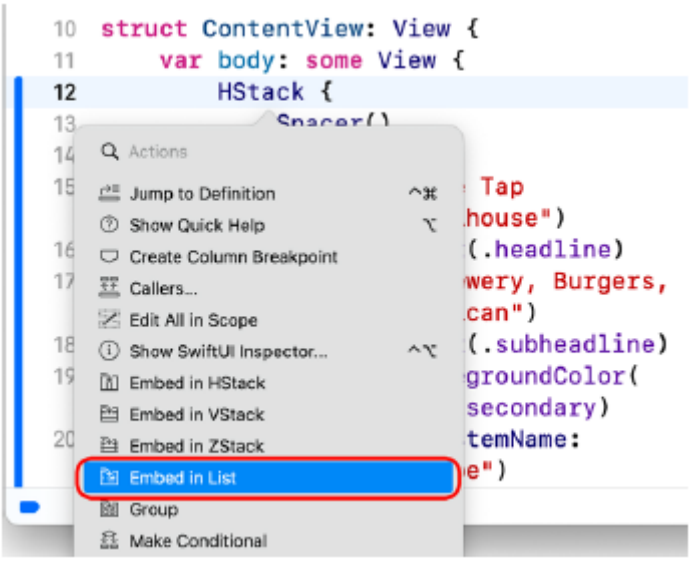
```
HStack {

Spacer()

VStack{

Text("The Tap Trailhouse")

.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "globe")

}

Spacer()

}
```

Twój widok jest już gotowy. W następnej sekcji użyjesz tego widoku jako komórki na ekranie Lista restauracji.

Dodawanie obiektów modelu i konfigurowanie nawigacji

Masz teraz widok, w którym możesz wyświetlić szczegółowe informacje o restauracji. Będziesz używać tego widoku jako komórki na liście SwiftUI, która jest kontenerem prezentującym dane w jednej kolumnie. Skonfigurujesz także obiekty modelu, aby wypełnić tę listę. Wykonaj poniższe kroki.

1. Command + kliknij widok HStack i wybierz opcję Osadź na liście, aby wyświetlić listę zawierającą pięć komórek w obszarze roboczym i usunąć obiekty Spacer:



2. Sprawdź, czy Twój kod wygląda następująco:

struct ContentView: View {

var body: some View {

List(0 ..<5) { item in

VStack{

Text("The Tap Trailhouse")

.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "globe")

}

}

}

}

Jak widać, widok utworzony w poprzedniej sekcji jest teraz zamknięty na liście skonfigurowanej do wyświetlania pięciu elementów, a widok HStack nie jest już potrzebny. Należy pamiętać, że do wyświetlania danych na liście nie są wymagani delegaci ani źródła danych.

3. Otwórz folder zasobów zawarty w folderze Chapter23 pakietu kodu pobranego z https://github.com/PacktPublishing/iOS-16-Programming-for-Beginners-

Wydanie siódme. Przeciągnij plik RestaurantItem.swift do nawigatora projektu i po wyświetleniu monitu o dodanie go do projektu kliknij przycisk Zakończ.

4. Kliknij plik RestaurantItem w nawigatorze projektu. Powinieneś zobaczyć w nim następujący kod:

```
import Foundation
import MapKit
struct RestaurantItem: Identifiable, Hashable {
var id = UUID()
var name: String
var address: String
var city: String
var cuisines: [String] = []
var lat: CLLocationDegrees
var long: CLLocationDegrees
var imageURLString: String
var title: String {
return name
}
var subtitle: String {
if cuisines.isEmpty { return "" }
else if cuisines.count == 1 { return
cuisines.first! }
else { return cuisines.joined(
separator: ", ")}
}
}
let testData = [
RestaurantItem(name: "The Tap Trailhouse", address: "17 Union St", city:
"Boston", cuisines: ["Brewery","Burgers","American"], lat: 42.360847,
long: -71.056819, imageURLString: "https://resizer.otstatic.com/v2/
profiles/legacy/145237.jpg"),
RestaurantItem(name: "o ya", address: "9 East Street", city: "Boston",
cuisines: ["Japanese","Sushi","Int'l"], lat: 42.351353, long: -71.056941,
```

imageURLString: "https://resizer.otstatic.com/v2/profiles/legacy/28066"), RestaurantItem(name: "Skipjack's Boston", address: "199 Clarendon St.", city: "Boston", cuisines: ["American", "Burgers","Brewery"], lat: 42.349887, long: -71.07484, imageURLString: "https://resizer.otstatic.com/v2/profiles/legacy/11656"), RestaurantItem(name: "The Elephant Walk", address: "900 Beacon Street", city: "Boston", cuisines: ["Panasian", "Vietnamese","Int'l"], lat:

42.346541, long: -71.105827, imageURLString: "https://resizer.otstatic.com/v2/profiles/legacy/1635"), RestaurantItem(name: "Metropolis Cafe", address: "584 Tremont Street", city: "Boston", cuisines: ["Mediterranean", "Int'l","Tapas"], lat:

42.3432, long: -71.0727, imageURLString: "https://resizer.otstatic.com/v2/profiles/legacy/2829")

]

Plik RestaurantItem zawiera strukturę RestaurantItem i tablicę testData. Struktura RestaurantItem jest podobna do klasy RestaurantItem użytej w projekcie LetsEat. Aby użyć tej struktury na liście, musisz dostosować ją do protokołu Identible. Ten protokół określa, że element listy musi mieć właściwość id, która umożliwia identyfikację konkretnego elementu. Instancja UUID jest przypisana do każdej instancji RestaurantItem po utworzeniu, aby zapewnić unikatowość każdego identyfikatora. Należy pamiętać, że ta struktura jest zgodna z protokołem Hashable. Zostanie to później wykorzystane do określenia danych wyświetlanych po dotknięciu komórki. testData to tablica zawierająca pięć instancji RestaurantItem reprezentujących pięć restauracji w rejonie Bostonu. Spełnia tę samą funkcję, co pliki JSON, których używałeś we wcześniejszych rozdziałach tej książki.

5. Kliknij plik ContentView w nawigatorze projektu. Dodaj do swojego widoku właściwość RestaurantItems, aby przechowywać dane dla listy po otwierającym nawiasie klamrowym struktury ContentView:

```
struct ContentView: View {

var restaurantItems: [RestaurantItem] = []

var body: some View {
```

6. Zmodyfikuj kod zgodnie z ilustracją, aby wypełnić listę danymi testowymi i wyświetlić dane restauracji w każdej komórce:

```
struct ContentView: View {

var restaurantItems: [RestaurantItem] = []

var body: some View {

List(restaurantItems) { restaurantItem in

VStack{

Text(restaurantItem.title)

.font(.headline)

Text(restaurantItem.subtitle)

.font(.subheadline)

.foregroundColor(.secondary)
```

```
AsyncImage(url: URL(string:
restaurantItem.imageURLString))
.mask(RoundedRectangle
(cornerRadius: 9))
}
}
}
}
struct ContentView_Previews: PreviewProvider {
static var previews: some View {
ContentView(restaurantItems: testData)
}
}
struktura ContentView_Previews: PreviewProvider {
podglądy statycznych zmiennych: niektóre Zobacz {
ContentView(restaurantItems: testData)
}
}
```

Zobaczmy, jak to działa.

Struktura ContentView przechowuje tablicę instancji RestaurantItem we właściwości RestaurantItems. Ta tablica jest przekazywana do listy. Dla każdego elementu w tablicy RestaurantItems tworzony jest widok, do którego przypisywane są dane z właściwości elementu. Obraz każdej restauracji jest pobierany z adresu URL przechowywanego we właściwości imageURLString elementu i wyświetlany przy użyciu nowego widoku AsyncImage wprowadzonego w iOS 15. Ponieważ tablica zawiera pięć elementów, na kanwie pojawia się pięć widoków.

Struktura ContentView_Previews przekazuje tablicę testData (przechowywaną w pliku RestaurantItem) do struktury ContentView, która jest następnie używana do zapełniania widoku.

7. Kiedy dokonasz większych zmian w kodzie, automatyczna aktualizacja canvasu zostanie wstrzymana. Kliknij przycisk Wznów, aby wznowić, jeśli to konieczne. Należy pamiętać, że rozmiar komórki zmienił się w celu dopasowania do rozmiaru obrazu restauracji. Następnie zaimplementujesz nawigację, dzięki której po dotknięciu komórki wyświetli się drugi ekran, który pokaże szczegóły konkretnej restauracji. Wykonaj następujące kroki:

1. Zmodyfikuj kod, jak pokazano, aby zawinąć listę w stos nawigacyjny:

```
var body: some View {
```

```
NavigationStack {
List(restaurantItems) { restaurantItem in
VStack{
Text(restaurantItem.title)
.font(.headline)
Text(restaurantItem.subtitle)
.font(.subheadline)
.foregroundColor(.secondary)
AsyncImage(url: URL(string:
restaurantItem.imageURLString)
.mask(RoundedRectangle
(cornerRadius: 9))
}
}
}
}
```

Stos nawigacyjny jest podobny do klasy UINavigation, której używałeś wcześniej w swojej aplikacji.

2. Dodaj modyfikator, aby ustawić właściwość tytułu listy tak, aby wyświetlała Boston w stanie Massachusetts na górze ekranu:

```
.mask(RoundedRectangle(cornerRadius: 9))
}
}.navigationTitle("Boston, MA")
```

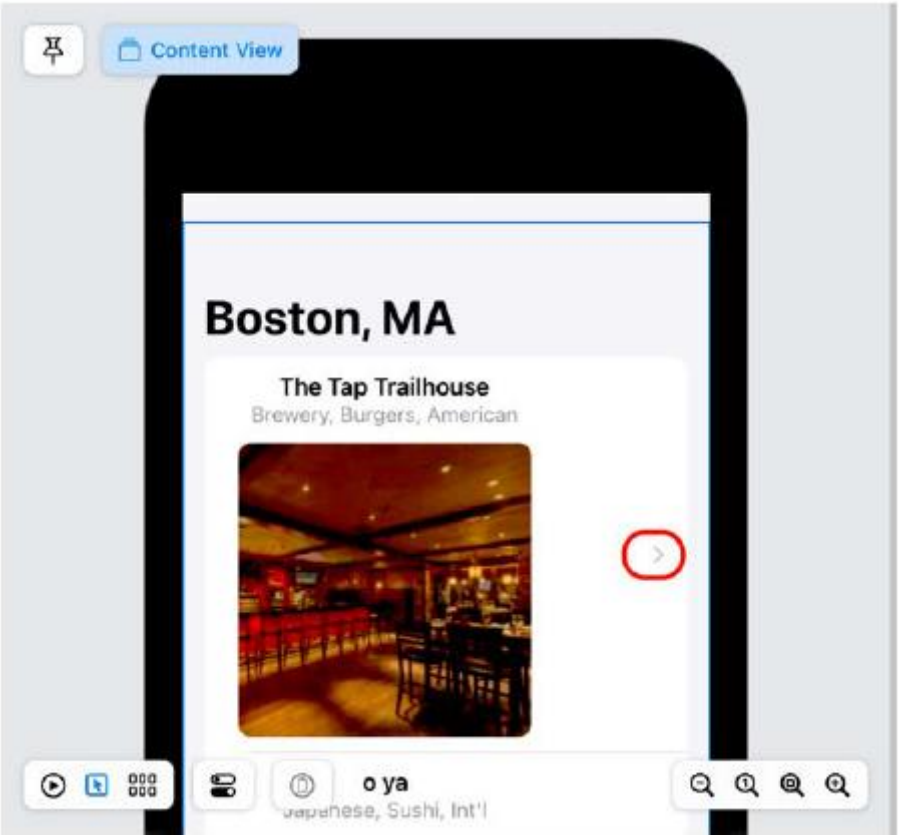
3. Zawiń komórkę w widok łącza nawigacyjnego, jak pokazano, i dodaj modyfikator .navigationDestination(for:d estination:) , aby po dotknięciu komórki wyświetlić nazwę restauracji:

```
List(restaurantItems) { restaurantItem in
NavigationLink(value:
restaurantItem) {
VStack{
Text(restaurantItem.title)
.font(.headline)
.fixedSize()
Text(restaurantItem.subtitle)
```

```
.font(.subheadline)
.foregroundColor(.secondary)
.fixedSize()
AsyncImage(url: URL(string:
restaurantItem.imageURLString))
.mask(RoundedRectangle
(cornerRadius: 9))
}
}
}.navigationTitle("Boston, MA")
.navigationDestination(for:
RestaurantItem.self) {
restaurantItem in
Text(restaurantItem.title)
}
```

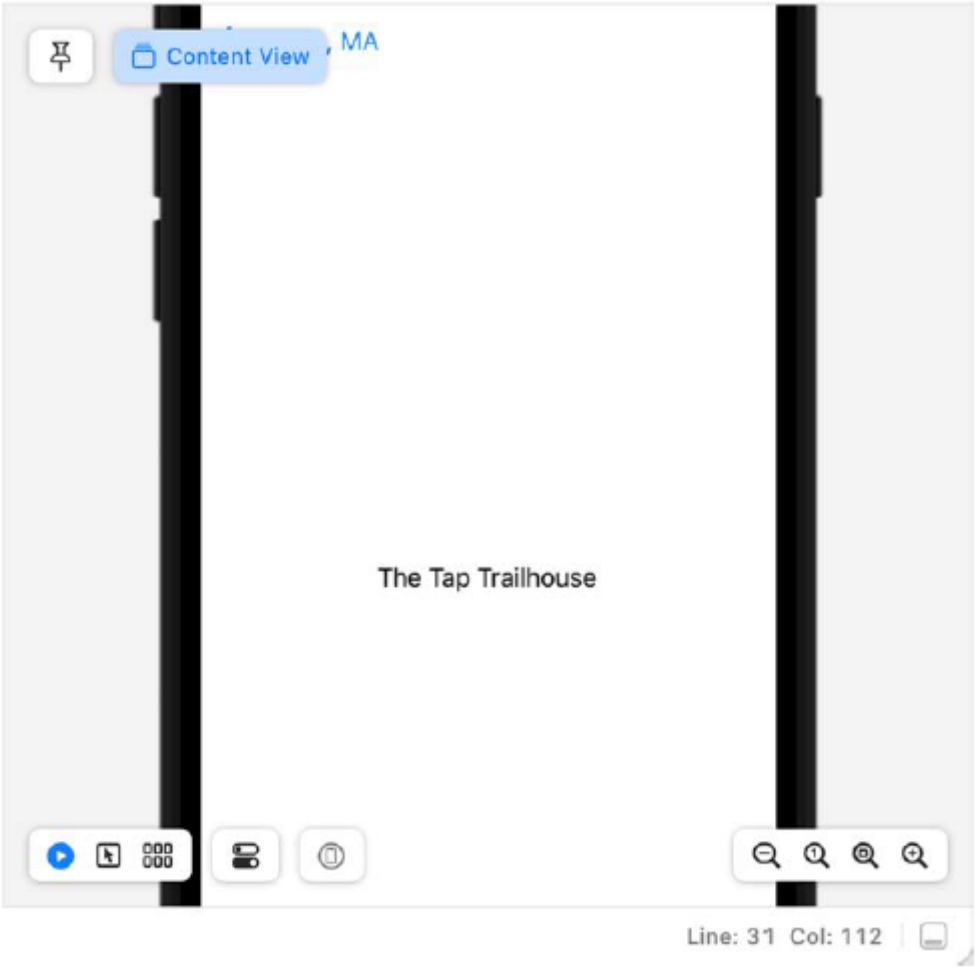
Modyfikator .fixedSize() zapewnia, że tekst w komórce nie zostanie obcięty.

4. Zwróć uwagę, że na liście w obszarze roboczym automatycznie pojawiają się strzałki ujawniania:



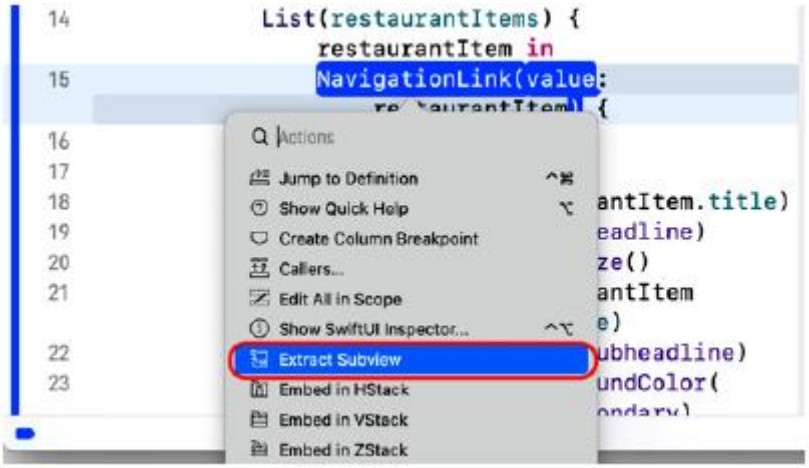
5. Aby zobaczyć, jak to działa tak, jak powinno w aplikacji, kliknij przycisk Podgląd na żywo w obszarze roboczym:

6. Kliknij dowolną komórkę podglądu, aby wyświetlić tekst zawierający nazwę wybranej restauracji:



To świetny sposób, aby upewnić się, że lista działa zgodnie z oczekiwaniami.

7. Kod widoku zaczyna wyglądać na zaśmiecony, dlatego wyodrębnisz komórkę do osobnego widoku. Command + kliknij widok nawigacji i wybierz opcję Wyodrębnij widok podrzędny:



8. Cały kod widoku komórki został przeniesiony do osobnego widoku o nazwie ExtractedView:

```
8  import SwiftUI
9
10 struct ContentView: View {
11     var restaurantItems: [RestaurantItem] = []
12     var body: some View {
13         NavigationStack {
14             List(restaurantItems) { restaurantItem
                   in
15                 ExtractedView()
16             }.navigationTitle("Boston, MA")
17                 .navigationDestination(for:
                       RestaurantItem.self) {
                       restaurantItem in
                       Text(restaurantItem.title)
18                 }
19         }
20     }
21 }
```

9. Zmień nazwę wywołania metody i wyodrębnionego widoku na RestaurantCell. Twój kod powinien wyglądać tak:

var body: some View {

NavigationStack {

List(restaurantItems) { restaurantItem in

RestaurantCell()

}.navigationTitle("Boston, MA")

.navigationDestination(for:

RestaurantItem.self) {

restaurantItem in

Text(restaurantItem.title)

```
}
}
}
}
struct ContentView_Previews: PreviewProvider {
static var previews: some View {
ContentView(restaurantItems: testData)
}
}
struct RestaurantCell: View {
var body: some View {
NavigationLink(value:
```

Nie przejmuj się błędem, naprawisz go w następnym kroku.

10. Dodaj właściwość do widoku RestaurantCell, aby przechowywać instancję RestaurantItem:

```
struct RestaurantCell: View {
var restaurantItem: RestaurantItem
```

11. Dodaj kod do struktury ContentView, aby przekazać instancję RestaurantItem do

Widok komórki restauracyjnej, jak pokazano:

```
struct ContentView: View {
var restaurantItems: [RestaurantItem] = []
var body: some View {
NavigationStack{
List(restaurantItems) { restaurantItem in
RestaurantCell(restaurantItem:
restaurantItem)
}.navigationTitle("Boston, MA"
.navigationDestination(for:
RestaurantItem.self) {
restaurantItem in
Text(restaurantItem.title)
}
```

}

}

}

12. Sprawdź, czy podgląd nadal działa tak samo jak wcześniej.

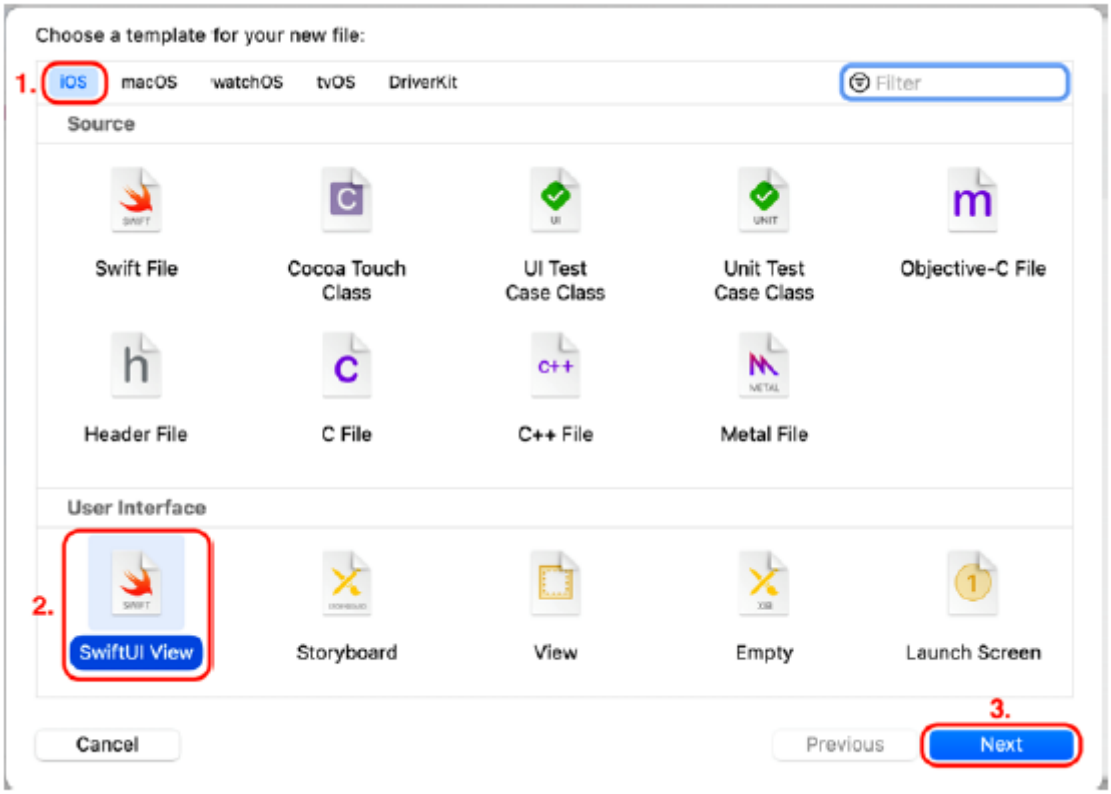
Zakończyłeś wdrażanie ekranu Lista restauracji. Następnie zobaczysz, jak możesz połączyć widoki UIKit i SwiftUI, aby utworzyć widok mapy, którego będziesz używać na ekranie Szczegóły restauracji.

Łączne używanie widoków UIKit i SwiftUI

W tym momencie utworzyłeś ekran Lista restauracji i dotknięcie każdej komórki na tym ekranie powoduje wyświetlenie nazwy restauracji na drugim ekranie. Zmodyfikujesz aplikację tak, aby wyświetlała ekran szczegółów restauracji po dotknięciu komórki na ekranie listy restauracji, ale wcześniej utworzysz widok SwiftUI wyświetlający mapę. Podczas korzystania ze scenorysów wystarczyło przeciągnąć widok mapy z Biblioteki do widoku w scenorysie. SwiftUI nie ma natywnego widoku mapy, ale możesz użyć tego samego widoku mapy, którego użyłeś w scenorysie do renderowania mapy. W rzeczywistości możesz użyć dowolnej podklasy widoku w SwiftUI, zawijając je w widok SwiftUI zgodny z protokołem UIViewRepresentable. Stwórzmy teraz niestandardowy widok, który może teraz przedstawiać widok mapy. Wykonaj następujące kroki:

1. Wybierz Plik | Nowy | Plik otwierający selektor szablonów.

2. iOS powinien być już wybrany. W sekcji Interfejs użytkownika kliknij Widok SwiftUI i kliknij Dalej:



3. Nazwij nowy plik MapView i kliknij Utwórz. Plik MapView pojawi się w nawigatorze projektu.

4. W pliku MapView zaimportuj MapKit i dostosuj strukturę MapView do protokołu UIViewRepresentable, jak pokazano. Nie przejmuj się pojawiającym się błędem. Poprawisz go w kilku kolejnych krokach:

```
import SwiftUI

import MapKit

struct MapView: UIViewRepresentable {

var body: some View {

Text("Hello World")

}

}
```

Protokół UIViewRepresentable to opakowanie umożliwiające użycie dowolnego widoku UIKit w hierarchii widoków SwiftUI.

5. Potrzebujesz dwóch metod, aby zachować zgodność z protokołem UIViewRepresentable; metodę makeUIView(context:) która tworzy MKMapView oraz metodę updateUIView(_:context:) która ją konfiguruje i reaguje na wszelkie zmiany. Zmodyfikuj swój kod, jak pokazano, aby zastąpić właściwość body metodą makeUIView(context:) która tworzy i zwraca pustą instancję MKMapView:

```
struct MapView: UIViewRepresentable {

func makeUIView(context: Context) -> MKMapView {

MKMapView(frame: .zero)

}

}
```

6. Zmodyfikuj swój kod, jak pokazano, aby dodać metodę updateUIView(_:context:) zaraz po metodzie makeUIView(context:). Spowoduje to ustawienie regionu widoku mapy tak, aby wyśrodkował mapę na lokalizacji The Tap Trailhouse:

```
func updateUIView(_ uiView: MKMapView, kontekst:

Context) {

let coordinate = CLLocationCoordinate2D

(latitude: 42.360847, longitude: -71.056819)

let span = MKCoordinateSpan(latitudeDelta:

0.001, longitudeDelta: 0.001)

let region = MKCoordinateRegion(center:

coordinate, span: span)

uiView.setRegion(region, animated: true)

}
```

7. Błąd zniknął i za chwilę powinieneś zobaczyć mapę Bostonu wyśrodkowaną na lokalizacji The Tap Trailhouse:

Jeśli to nie zadziała, sprawdź swoje połączenie internetowe.

8. Wartości szerokości i długości geograficznej są obecnie zakodowane na stałe. Zadeklaruj dwie właściwości przechowujące wartości szerokości i długości geograficznej, jak pokazano po deklaracji struktury MapView:

struct MapView: UIViewRepresentable {

var lat: CLLocationDegrees

var long: CLLocationDegrees

9. Zmodyfikuj metodę updateUI(_:context:) tak, aby używała tych właściwości zamiast zakodowanych na stałe wartości:

func updateUIView(_ view: MKMapView, context: Context) {

let coordinate = CLLocationCoordinate2D(

latitude: lat, longitude: long)

10. Zaktualizuj strukturę MapView_Previews, aby przekazać przykładowe wartości szerokości i długości geograficznej, jak pokazano. Spowoduje to wygenerowanie tej samej mapy, którą widziałeś wcześniej w podglądzie:

struct MapView_Previews: PreviewProvider {

static var previews: some View {

```
MapView(lat: 42.360847, long: -71.056819)

}

}
```

11. Sprawdź w obszarze roboczym, czy mapa jest nadal wyświetlana (może być konieczne kliknięcie Wznów). Utworzyłeś widok mapy SwiftUI, który pokazuje lokalizację restauracji. Zobaczmy teraz, jak utworzyć pełny ekran szczegółów restauracji w następnej sekcji.

Zamykanie ekranu szczegółów restauracji

Masz teraz widok mapy SwiftUI wyświetlający mapę. Teraz utworzysz nowy widok SwiftUI, który będzie reprezentował ekran szczegółów restauracji i dodasz do niego widok mapy. Wykonaj następujące kroki:

1. Wybierz Plik | Nowy | Plik otwierający selektor szablonów.

2. iOS powinien być już wybrany. W sekcji Interfejs użytkownika kliknij opcję Widok SwiftUI i kliknij Dalej.

3. Nazwij nowy plik RestaurantDetail i kliknij Utwórz. Plik RestaurantDetail pojawi się w nawigatorze projektu.

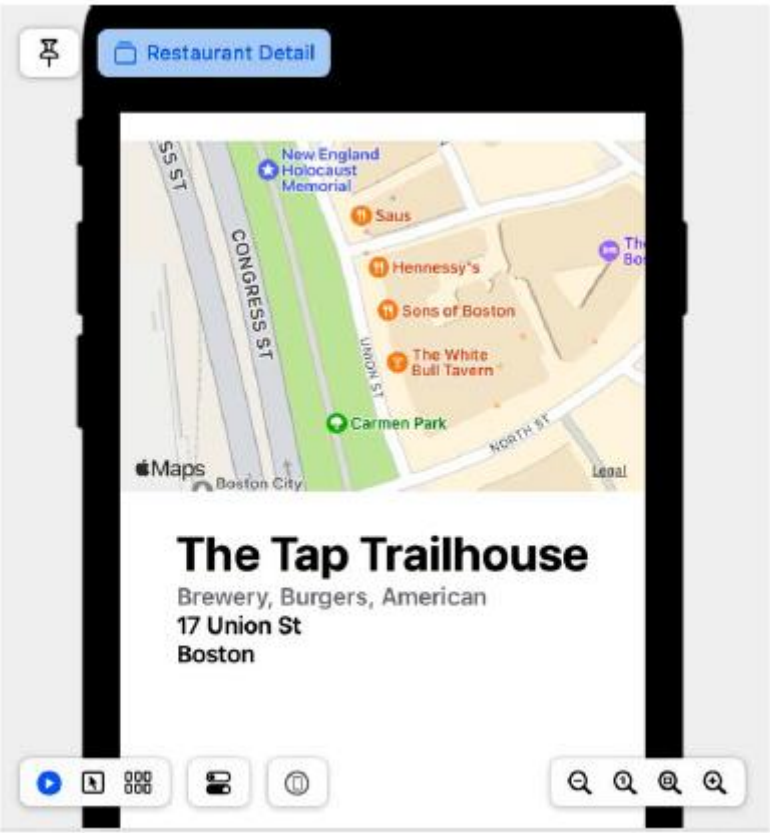
4. Zadeklaruj i zdefiniuj struktury RestaurantDetail i RestaurantDetail_Previews jak pokazano na rysunku:

```
import SwiftUI

struct RestaurantDetail: View {

var selectedRestaurant: RestaurantItem

var body: some View {

VStack {

MapView(lat: selectedRestaurant.lat,

long: selectedRestaurant.long)

.frame(height: 250)

VStack(alignment: .leading) {

Text(selectedRestaurant.title)

.font(.largeTitle)

.fontWeight(.bold)

Text(selectedRestaurant.subtitle)

.font(.headline)

.foregroundColor(.secondary)

Text(selectedRestaurant.address)

.font(.headline)
```

```
Text(selectedRestaurant.city)
.font(.headline)
}.padding()
Spacer()
}
}
}
struct RestaurantDetail_Previews: PreviewProvider {
static var previews: some View {
NavigationView {
RestaurantDetail(selectedRestaurant:
testData[0])
}
}
}
```

Struktura RestaurantDetail zawiera widok Vstack obejmujący widok mapy i drugi widok Vstack. Widok mapy wyświetla mapę pokazującą lokalizację restauracji. Drugi widok Vstack zawiera cztery widoki tekstowe. Wyświetlają one nazwę restauracji, kuchnię, adres i miasto. Obiekt Spacer wypycha pierwszy widok Vstack na górę ekranu. Do właściwości wybranej restauracji przypisana jest instancja RestaurantItem, a dane z tej instancji służą do zapełnienia widoków struktury RestaurantDetail. Aby utworzyć podgląd na kanwie, struktura RestaurantDetail_Previews przechodzi w pierwszej instancji RestaurantItem w tablicy testData. Należy pamiętać, że instancja RestaurantDetail jest zawarta w instancji NawigacjiView, aby pasek nawigacji pojawił się w podglądzie.

5. W obszarze roboczym wyświetlany jest teraz ekran szczegółów restauracji z wyrenderowaną mapą:



Zakończyłeś wdrażanie ekranu szczegółów restauracji za pomocą SwiftUI. Teraz zmodyfikujesz listę na ekranie Lista restauracji, tak aby po dotknięciu komórki wyświetlany był ekran Szczegóły restauracji.

6. Kliknij plik ContentView w nawigatorze projektu i zmodyfikuj modyfikator nawigacjiDestination(for:destination:) tak, aby po dotknięciu komórki jako miejsca docelowego używana była struktura RestaurantDetail:

.navigationDestination(for:

RestaurantItem.self) {

restaurantItem in

RestaurantDetail(selectedRestaurant:

restaurantItem)

}

7. Przycisk Podgląd na żywo w obszarze roboczym powinien być już wybrany. Kliknij wiersz na ekranie Lista restauracji. Zobaczysz ekran szczegółów restauracji dla tej restauracji:

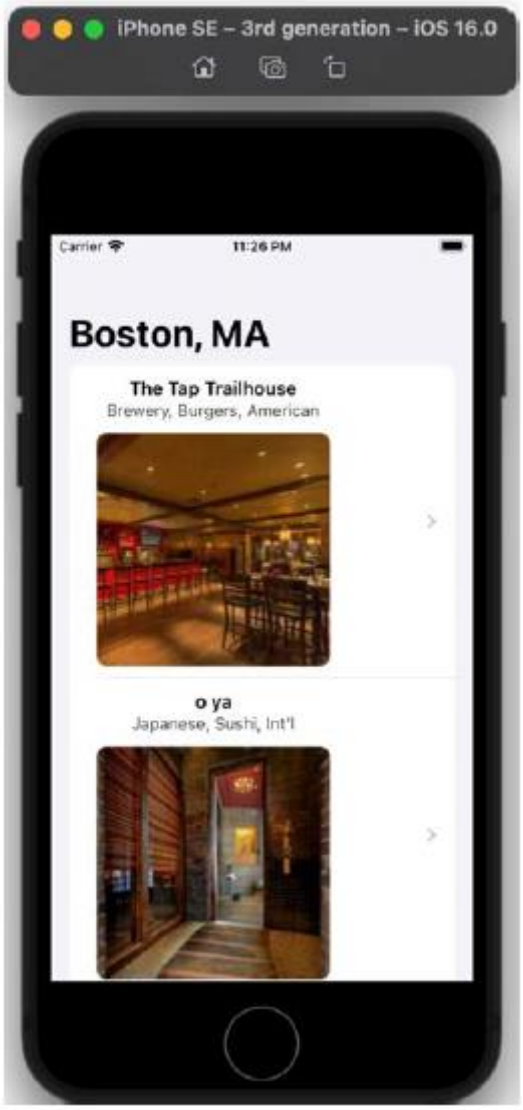
Jak widać, podgląd aplikacji działa dobrze w obszarze roboczym. Jeśli chcesz uruchomić w symulatorze, musisz wprowadzić jedną małą zmianę w strukturze ContentView. Kliknij plik ContentView w nawigatorze projektu i przypisz tablicę testdata do właściwości RestaurantItems, jak pokazano:

struct ContentView: View {

var restaurantItems: [RestaurantItem] = testData

var body: some View {

Zbuduj i uruchom swoją aplikację, a pojawi się ona w symulatorze:



Ukończyłeś tworzenie prostej aplikacji SwiftUI! Wspaniały!

Streszczenie

W tym krótkim wprowadzeniu do SwiftUI pokazałeś, jak zbudować uproszczoną wersję aplikacji Let’s Eat przy użyciu SwiftUI. Rozpocząłeś od dodania i skonfigurowania widoków SwiftUI w celu utworzenia ekranu Lista restauracji. Następnie dodałeś obiekty modelu do swojej aplikacji i skonfigurowałeś nawigację pomiędzy ekranami Lista restauracji i Szczegóły restauracji. Następnie użyłeś razem widoków UIKit i SwiftUI, dodając i konfigurując widok mapy dla ekranu szczegółów restauracji. Na koniec utworzyłeś ekran szczegółów restauracji i dodałeś do niego utworzony wcześniej widok mapy. Teraz wiesz, jak używać SwiftUI do tworzenia aplikacji, która odczytuje obiekty modelu, prezentuje je na liście i umożliwia nawigację do drugiego ekranu zawierającego widok mapy. Następnie możesz wdrożyć to w swoich własnych projektach. Poznasz WidgetKit, który pozwala na tworzenie widżetów na ekranie blokady Twojego iPhone'a.

Pierwsze kroki z widżetami ekranu blokady

Firma Apple wprowadziła WidgetKit podczas WWDC 2020, który umożliwia wyświetlanie odpowiednich i czytelnych treści z aplikacji na ekranie głównym systemu iOS. Podczas WWDC 2022 firma Apple zaktualizowała WidgetKit, aby obsługiwał komplikacje związane z Apple Watch i widżetami na ekranie blokady iPhone'a. W tym rozdziale zaimplementujesz widżet ekranu blokady dla swojego projektu LetsEat, który pokaże Ci restauracje z określonej lokalizacji. Zaczniesz od dodania rozszerzenia widżetu do swojej aplikacji. Skonfigurujesz widżet tak, aby wyświetlał restauracje z określonej lokalizacji i dostosujesz wygląd widżetu. Na koniec dodasz widżet do ekranu blokady. Pod koniec tego rozdziału dowiesz się, jak utworzyć i skonfigurować widżet ekranu blokady, a także będziesz mógł go zaimplementować we własnych aplikacjach.

Przedstawiamy widżety

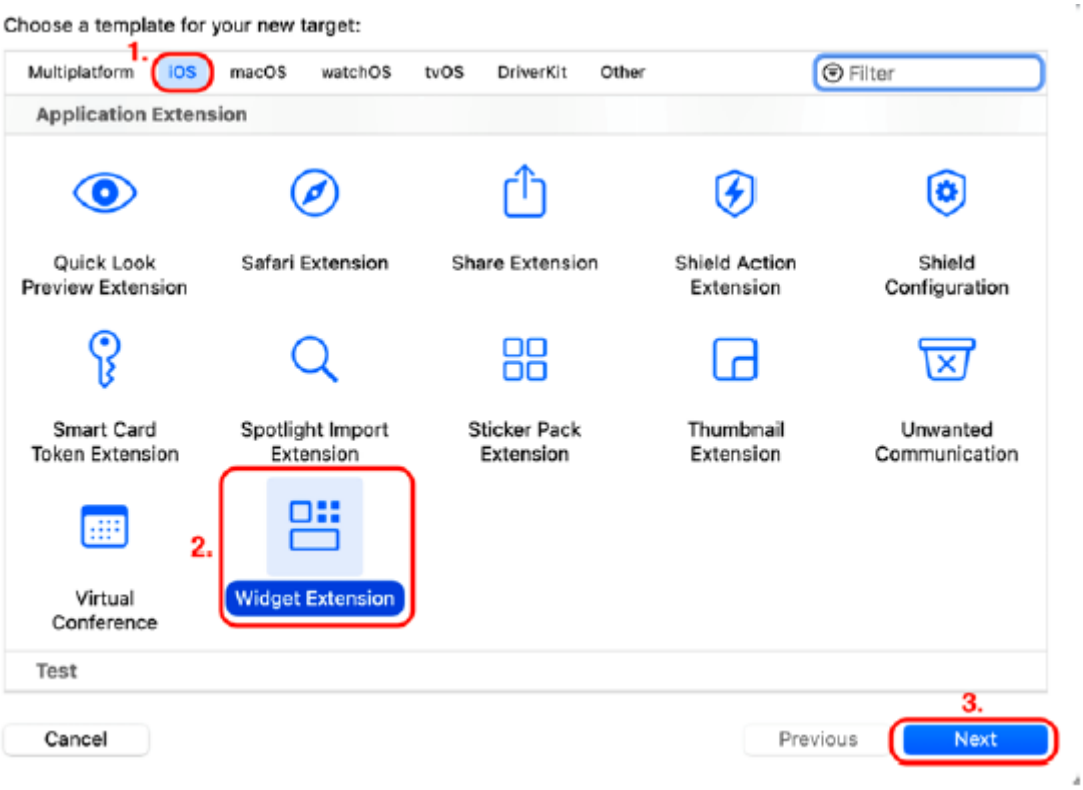
Widżet pojawia się w postaci małego panelu informacji na ekranie głównym. Ma na celu dostarczanie odpowiednich i czytelnych treści. Na przykład, jeśli jesteś w określonej lokalizacji, widżet może wyświetlić restauracje w tej lokalizacji. Dotknięcie widżetu powoduje uruchomienie aplikacji. W systemie iOS 16 widżety można także wyświetlać na ekranie blokady oraz na ekranie głównym. Apple udostępnia szablony, które bardzo ułatwiają dodanie widżetu do aplikacji. Zacznijmy od dodania celu widżetu do Twojego projektu LetsEat w następnej sekcji.

Dodanie celu widgetu do Twojej aplikacji

Aby dodać widżet do projektu LetsEat, użyjesz szablonu rozszerzenia widżetu. Ten szablon zawiera cały kod niezbędny do wyświetlenia widżetu na ekranie głównym, skonfigurowania początkowego widoku tego widżetu i uruchomienia aplikacji po dotknięciu widżetu. Wykonaj następujące kroki:

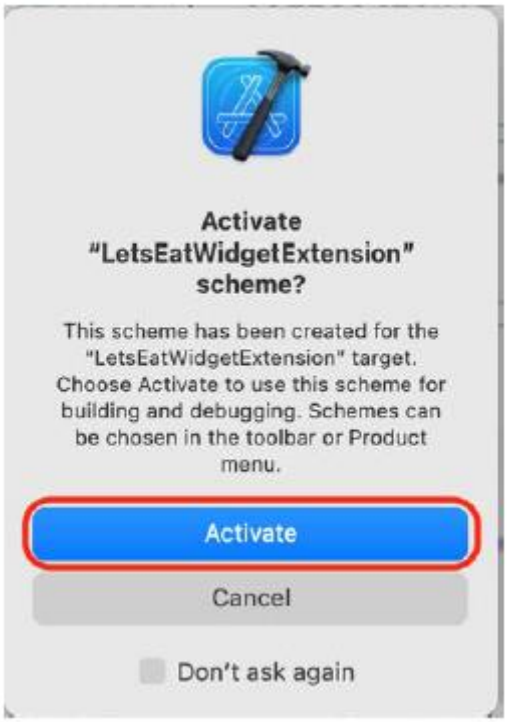
1. Wybierz Plik | Nowy | Cel, aby otworzyć selektor szablonów.

2. iOS powinien być już wybrany. W sekcji Rozszerzenie aplikacji kliknij Rozszerzenie widgetu i kliknij Dalej:

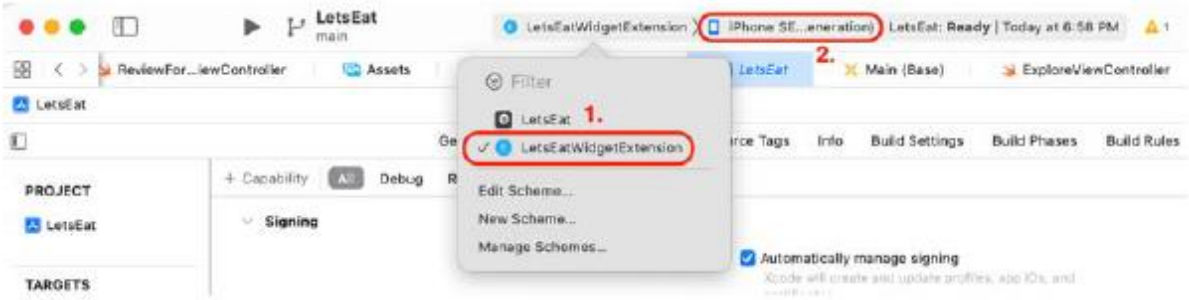


3. Nazwij rozszerzenie LetsEatWidget i upewnij się, że pole wyboru Uwzględnij intencję konfiguracji nie jest zaznaczone. Dzieje się tak dlatego, że użytkownik nie może konfigurować tego widżetu. Kliknij Zakończ.

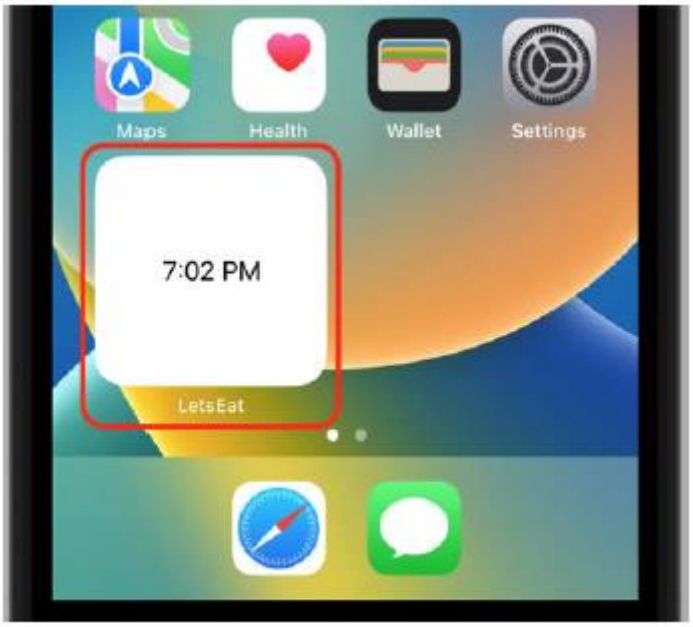
4. Kliknij Aktywuj w wyświetlonym oknie dialogowym. Spowoduje to utworzenie schematu dla Twojego widżetu, którego możesz użyć do uruchomienia go w symulatorze iOS:



5. Sprawdź, czy w menu Schemat wybrano schemat LetsEatWidgetExtension i iPhone SE (3. generacji):



Kompiluj i uruchamiaj swoją aplikację. Na ekranie głównym zobaczysz widżet wyświetlający godzinę:



Kliknij swój widget, a uruchomi się aplikacja Let's Eat. Pomyślnie dodałeś widżet do swojej aplikacji! W następnej sekcji skonfigurujesz widget tak, aby wyświetlał informacje o restauracji z określonej lokalizacji.

Dostarczanie wpisów osi czasu do swojego widżetu

Aby wyświetlić informacje w widżecie, skonfiguruj dostawcę osi czasu widżetu. Spowoduje to wygenerowanie osi czasu składającej się z wpisów na osi czasu. Każdy wpis określa datę i godzinę aktualizacji zawartości widżetu. Użyjesz klasy MapDataManager, aby uzyskać szczegółowe informacje o restauracji, utworzyć wpisy na osi czasu dla każdej restauracji i skonfigurować je tak, aby były wyświetlane co pięć minut. Wykonaj następujące kroki:

1. Zatrzymaj aplikację, jeśli jest uruchomiona. Kliknij plik LetsEatWidget w folderze LetsEatWidget w nawigatorze projektu. Podobnie jak w Rozdziale 24, Pierwsze kroki z SwiftUI, po lewej stronie edytora zobaczysz kod swojego widżetu, a po prawej stronie płótno zawierające podgląd.

2. Kliknij przycisk Wznów, aby wyświetlić podgląd widżetu na kanwie. Wyświetla tylko aktualny czas.

3. Twój widget jest zadeklarowany i zdefiniowany w strukturze LetsEatWidget. Zmodyfikuj go w następujący sposób:

```
@main

struct LetsEatWidget: Widget {
```

```
let kind: String = "LetsEatWidget"

var body: some WidgetConfiguration {

StaticConfiguration(kind: kind,

provider: Provider()) { entry in

LetsEatWidgetEntryView(entry: entry)

}

.configurationDisplayName("LetsEatWidget")

.description("This widget shows you

restaurants in your area.")

.supportedFamilies([.accessoryRectangular,

.systemSmall])

}

}
```

Rozbijmy to:

let kind: String = "LetsEatWidget"

Właściwość kind zawiera ciąg znaków używany do identyfikacji widgetu.

var body: some WidgetConfiguration {

Protokół WidgetConfiguration opisuje zawartość widżetu. Widżety mogą być statyczne lub konfigurowalne przez użytkownika.

StaticConfiguration(kind: kind, provider: Provider())

Używany w widżecie bez właściwości konfigurowalnych przez użytkownika. Provider() to obiekt zgodny z protokołem TimelineProvider, który generuje oś czasu dla Twojego widgetu. Ta oś czasu służy do informowania WidgetKit o konieczności aktualizacji widgetu. Zawiera instancje niestandardowego typu TimelineEntry. Każdy wpis zawiera datę używaną przez WidgetKit do aktualizacji zawartości widżetu.

{ entry in

LetsEatWidgetEntryView(entry: entry)

}

Spowoduje to pobranie wpisu z osi czasu i wyświetlenie go przy użyciu widoku LetsEatWidgetEntryView.

configurationDisplayName(_:)

Ustawia nazwę wyświetlaną dla widżetu w selektorze widżetów.

description(_:)

Ustawia opis wyświetlany dla widżetu w selektorze widżetów.

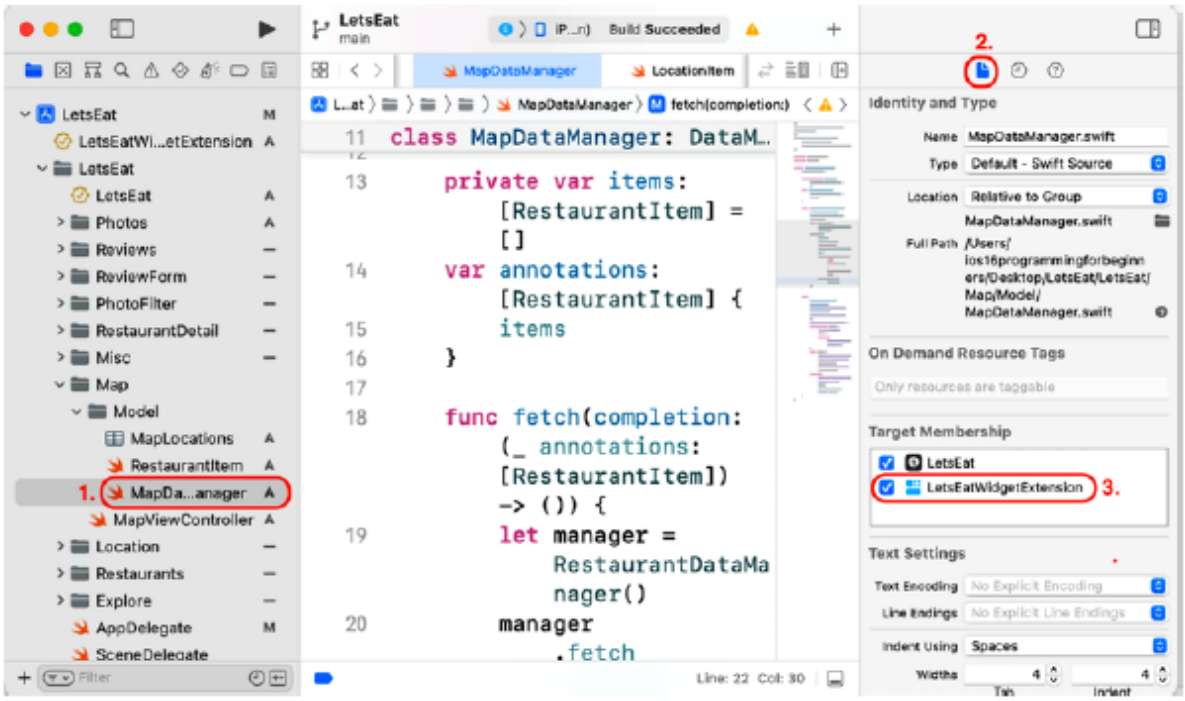
.supportedFamilies(_:)

Ustawia rozmiary obsługiwane przez widget. W przypadku ekranu głównego obsługiwane rozmiary to .systemSmall, .systemMedium, .systemLarge i .systemExtraLarge. W przypadku ekranu blokady obsługiwane rozmiary to .accessoryCircular, .accessoryCorner, .accessoryRectangular i .accessoryInline. Będziesz używać rozmiaru .systemSmall dla widżetu ekranu głównego i .accessoryRectangular dla widżetu ekranu blokady.

4. Za podgląd widgetu wyświetlany na canvasie odpowiada struktura LetsEatWidget_Previews. Zmodyfikuj kod, jak pokazano, aby wyświetlić podgląd widżetu ekranu blokady na kanwie:

.previewContext(WidgetPreviewContext(family: .accessoryRectangular))

5. Użyjesz klasy MapDataManager (stworzonej w rozdziale 17, Pierwsze kroki z MapKit), aby dostarczyć do swojego widżetu listę restauracji w określonej lokalizacji. Aby udostępnić tę klasę swojemu widżetowi, kliknij MapDataManager (znajdujący się w folderze Model w folderze Mapa) w nawigatorze projektu i kliknij przycisk Inspektora plików. W obszarze Członkostwo docelowe zaznacz LetsEatWidgetExtension:



6. Klasa MapDataManager wymaga udostępnienia widgetowi innych klas. Powtórz krok 4 dla następujących plików:

RestaurantItem (w folderze Model w folderze Mapa)

DataManager (w folderze Różne)

Boston.json (w folderze JSON)

RestaurantDataManager (w folderze Model w folderze Restauracje)

7. Kliknij LetsEatWidget w nawigatorze projektu. Metoda getTimeline(in:completion:) w strukturze Provider aktualnie generuje oś czasu składającą się z pięciu wpisów w odstępie godziny,

począwszy od aktualnej daty. Zmodyfikuj go w następujący sposób, aby załadować dane restauracji z pliku Boston.json i zapisać je w tablicy RestaurantItems:

```
func getTimeline(w kontekście: Kontekst, zakończenie:

@escaping (Timeline<Entry>) -> ()) {

var entries: [SimpleEntry] = []

let currentDate = Date()

var restaurantItems: [RestaurantItem] = []

let manager = MapDataManager()

manager.fetch { (annotations) in

restaurantItems = annotations

}
```

8. Struktura SimpleEntry jest obiektem modelu dostarczającym dane dla każdego wpisu na osi czasu. Obecnie ma właściwość date. Zmodyfikuj go w następujący sposób, aby dodać właściwości przechowujące tytuł i podtytuł restauracji:

```
struct SimpleEntry: TimelineEntry {

let date: Date

var restaurantTitle: String

var restaurantSubtitle: String

}
```

Nie martw się błędem, ponieważ wkrótce go naprawisz.

9. Struktura LetsEatWidgetEntryView określa wygląd widgetu. Obecnie ma pojedynczą właściwość widoku tekstowego wyświetlającą datę. Zmodyfikuj go w następujący sposób, aby wyświetlał tytuł i podtytuł restauracji:

```
struct LetsEatWidgetEntryView : View {

var entry: Provider.Entry

var body: some View {

VStack {

Text(entry.restaurantTitle)

Text(entry.restaurantSubtitle)

}

}
```

}

10. Zmodyfikuj pętlę for w metodzie getTimeline(in:completion:) tak, aby wygenerować oś czasu składającą się z wpisu dla każdej restauracji w Boston.json, przy czym każdy wpis będzie wyświetlany pięć minut po poprzednim wpisie, zaczynając od bieżącej daty:

```
manager.fetch { (annotations) in

restaurantItems = annotations

}

for minuteOffset in 0 ..< restaurantItems.count {

let entryDate = Calendar.current.date(byAdding:

.minute, value: minuteOffset * 5,

to: currentDate)!

let entry = SimpleEntry(date: entryDate,

restaurantTitle:

restaurantItems[minuteOffset].title!,

restaurantSubtitle:

restaurantItems[minuteOffset].subtitle!)

entries.append(entry)

}

let timeline = Timeline(entries: entries,

policy: .atEnd)
```

Ustawienie opóźnienia czasowego na pięć minut służy wyłącznie celom testowym, dzięki czemu można szybko zobaczyć wyniki. W przypadku własnych aplikacji powinieneś ładować osie czasu tak oszczędnie, jak to możliwe.

11. WidgetKit wysyła żądanie migawki podczas wyświetlania widgetu w sytuacjach przejściowych. Musisz podać przykładowe dane, jeśli widget potrzebuje więcej niż kilka sekund na pobranie lub obliczenie swojego aktualnego stanu. Zmodyfikuj metodę getSnapshot(in:) w następujący sposób, aby wyświetlała ciąg „LetsEat” w tytule restauracji i pusty ciąg w podtytule restauracji:

```
func getSnapshot(in context: Context, completion:

@escaping (SimpleEntry) -> ()) {

let entry = SimpleEntry(date: Date(),

restaurantTitle: "LetsEat",

restaurantSubtitle: "")

completion(entry)

}
```

12. Metoda placeholder(in:) udostępnia widok zastępczy podczas konfigurowania widżetu do wyświetlania. Zmodyfikuj go tak, aby widżet wyświetlał ciąg „LetsEat” jako tytuł restauracji i pusty ciąg znaków jako podtytuł restauracji:

```
func placeholder(in context: Context) -> SimpleEntry

{

SimpleEntry(date: Date(),

restaurantTitle: "LetsEat",

restaurantSubtitle: "")

}
```

13. Zmodyfikuj strukturę LetsEatWidget_Previews tak, aby widżet wyświetlał w podglądzie przykładowy tytuł i podtytuł restauracji:

```
struct LetsEatWidget_Previews: PreviewProvider {

static var previews: some View {

LetsEatWidgetEntryView(entry: SimpleEntry(

date: Date(),

restaurantTitle: "The Tap Trailhouse",

restaurantSubtitle: "Brewery, Burgers,

American"))

.previewContext(WidgetPreviewContext(

family: .accessoryRectangular))

}

}
```

14. Sprawdź w Canvas, czy podgląd widżetu wyświetla nazwę restauracji oraz kuchnię:

Skonfigurowałeś widżet tak, aby co pięć minut wyświetlał nazwę restauracji i kuchnię. W następnej sekcji zmodyfikujesz widok widżetu, aby dopasować go do projektu aplikacji.

Dostosowywanie widoku widżetu

Obecnie Twój widget wyświetla nazwę restauracji i kuchnię białym tekstem na ekranie blokady. Skonfigurujesz widżet, aby poprawić jego wygląd. Kliknij LetsEatWidget.swift w nawigatorze projektu i zmodyfikuj strukturę LetsEatWidgetEntryView w następujący sposób:

```
struct LetsEatWidgetEntryView : View {

var entry: Provider.Entry

var body: some View {

ZStack {

AccessoryWidgetBackground()

VStack {

Text(entry.restaurantTitle)

.font(.headline)

Text(entry.restaurantSubtitle)

.font(.body)

}

}
```

}

}

Rozbijmy to:

Zstack {

AccessoryWidgetBackground()

Widok Zstack nakłada i wyrównuje wszystkie swoje widoki podrzędne. Widok podrzędny AccessoryWidgetBackground() zapewnia widgetowi tło.

Text(entry.restaurantTitle)

.font(.headline)

Ustawia właściwości tekstu dla widoku tekstu tytułu restauracji. .font ma ustawiony styl .headline, który jest zalecany w przypadku widżetów na ekranie blokady.

Text(entry.restaurantSubtitle)

.font(.body)

Ustawia właściwości tekstu dla widoku tekstu napisów restauracji. Plik .font ma ustawiony styl .body, który jest zalecany w przypadku widżetów na ekranie blokady. Jeśli to konieczne, kliknij przycisk Wznów na obszarze roboczym, a Twój widżet powinien wyglądać następująco:

Nazwa restauracji jest teraz widoczna, a za tekstem znajduje się subtelne tło. W następnej sekcji zobaczysz, jak dodać widżety do ekranu blokady.

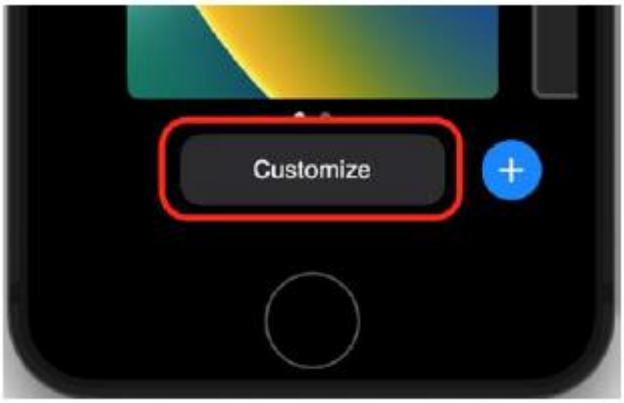
Dodawanie widżetu do ekranu blokady

Do tej pory testowałeś swój widget wykorzystując schemat LetsEatWidgetExtension. Będziesz teraz korzystać ze schematu LetsEat i zobaczyć, jak dodawać widżety do ekranu blokady. Wykonaj następujące kroki:

1. Z menu Schemat wybierz schemat LetsEat i iPhone SE (3. generacji):

2. Zbuduj i uruchom swoją aplikację. Wybierz opcję Zablokuj z menu Urządzenie.

3. Naciśnij przycisk Początek, aby wybudzić symulator. Stuknij i przytrzymaj dowolne miejsce na ekranie blokady, aby wyświetlić przycisk Dostosuj:

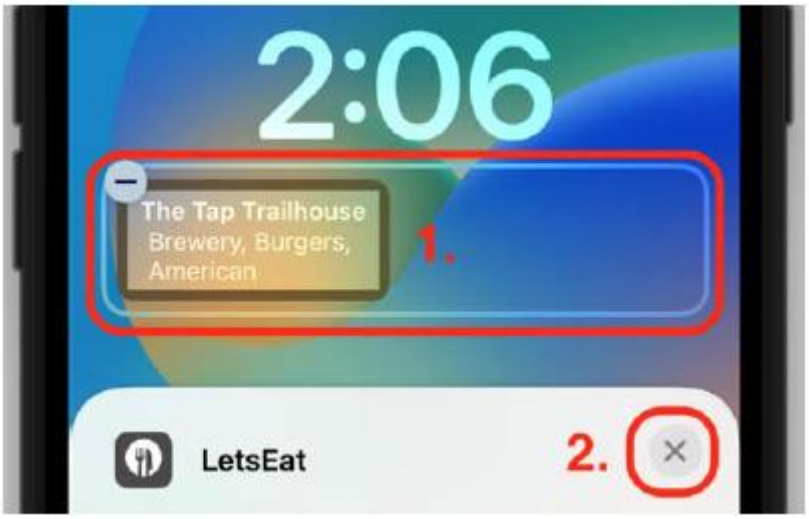


4. Stuknij przycisk Dostosuj. Twój ekran powinien wyglądać jak na poniższym zrzucie ekranu, pokazującym obszary, w których można zainstalować widżety ekranu blokady:

5. Stuknij obszar pod wyświetlaczem czasu, aby wyświetlić selektor widżetów. Kliknij pozycję menu LetsEat:

6. Wyświetlona zostanie nazwa i opis widżetu. Stuknij widżet, aby dodać go do ekranu blokady:

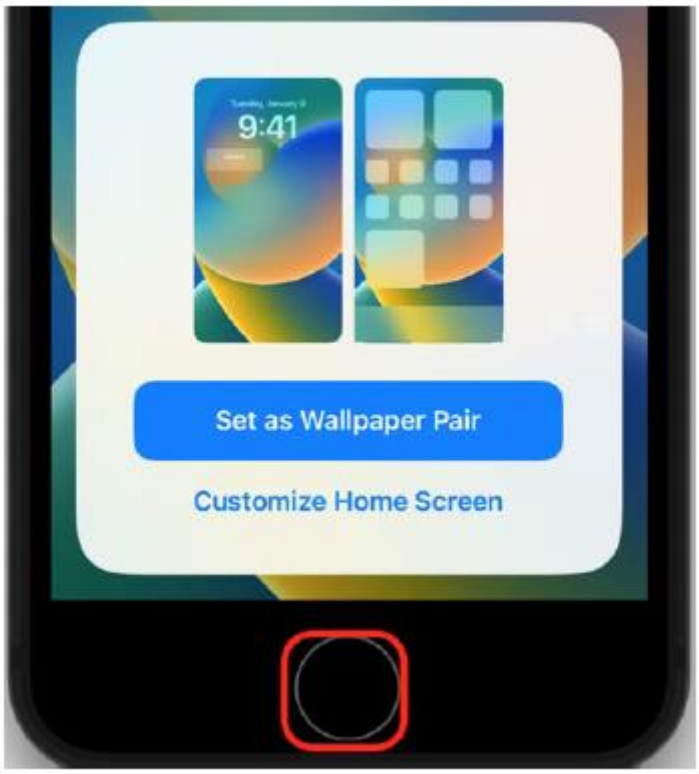
7. Sprawdź, czy widżet LetsEat został dodany do ekranu blokady i dotknij przycisku x:



8. Naciśnij przycisk x, aby zamknąć selektor widżetów:

9. Stuknij Gotowe:

10. Jeśli pojawi się okno dialogowe z dwoma przyciskami, Ustaw jako parę tapet i Dostosuj ekran główny, dotknij przycisku Początek, aby zamknąć to okno dialogowe:



11. Sprawdź, czy Twój widżet znajduje się teraz na ekranie blokady i czy nazwa restauracji zmienia się co pięć minut:

Pomyślnie utworzyłeś i dodałeś widżet ekranu blokady do swojej aplikacji! Wspaniały!

Streszczenie

W tej części zaimplementowałeś widżet ekranu blokady dla swojego projektu LetsEat, który pokazuje restauracje z określonej lokalizacji. Najpierw dodałeś rozszerzenie widżetu do swojej aplikacji. Następnie skonfigurowałeś widżet tak, aby pobierał informacje o restauracji z klasy MapDataManager i skonfigurowałeś jego wygląd za pomocą SwiftUI. Wreszcie dodałeś widżet do ekranu blokady. Wiesz już, jak utworzyć i skonfigurować widżet ekranu blokady i będziesz mógł go zaimplementować we własnych aplikacjach. W następnym rozdziale poznasz WeatherKit, świetny sposób na uzyskiwanie dostępu do informacji o pogodzie i wyświetlanie ich w aplikacjach.

Pierwsze kroki z WeatherKit

Podczas WWDC 2022 firma Apple wprowadziła WeatherKit, który oferuje dane pogodowe dla aplikacji i usług obsługiwanych przez zupełnie nową usługę Apple Weather Service. Wykorzystuje modele pogody o wysokiej rozdzielczości, uczenie maszynowe i algorytmy predykcyjne, aby zapewnić hiperlokalne prognozy pogody na całym świecie bez narażania prywatności użytkowników. W tym rozdziale zmodyfikujesz aplikację Let's Eat, aby używać WeatherKit do uzyskiwania informacji o pogodzie dla konkretnej restauracji i wyświetlania ich na ekranie szczegółów restauracji. Zaczniesz od nauczenia się, jak działa WeatherKit. Następnie utworzysz identyfikator aplikacji i dodasz funkcję WeatherKit do aplikacji Let's Eat. Następnie utworzysz nową klasę menedżera danych, WeatherDataManager, aby uzyskać informacje o pogodzie z lokalizacji określonej restauracji. Na koniec zaktualizujesz klasę RestaurantDetailViewController, aby wyświetlać pogodę w lokalizacji restauracji na ekranie Szczegóły restauracji. Pod koniec tej części dowiesz się, jak działa WeatherKit i jak dodać go do własnych aplikacji.

Zrozumienie WeatherKita

WeatherKit został wprowadzony przez firmę Apple podczas WWDC 2022. Zapewnia hiperlokalne prognozy pogody na całym świecie, obsługiwane przez zupełnie nową usługę Apple Weather Service, i robi to przy jednoczesnym zachowaniu bezpieczeństwa użytkownika

Prywatność. Dzięki WeatherKit będziesz mieć dostęp do wielu danych, takich jak aktualna pogoda, prognoza minutowa, prognoza godzinowa, prognoza dzienna, alerty pogodowe i pogoda historyczna. Dane te są dostępne zarówno poprzez natywną platformę iOS, jak i zestaw interfejsów API REST. Zanim użyjesz WeatherKit w swojej aplikacji, musisz utworzyć niestandardowy identyfikator aplikacji dla swojej aplikacji i dodać do niej funkcję WeatherKit. Zobaczmy, jak to zrobić w następnej sekcji.

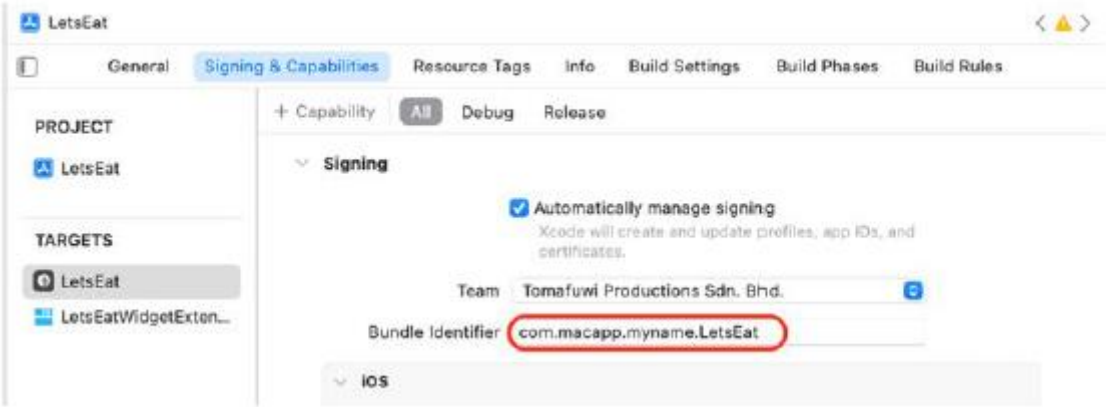
Przygotowanie aplikacji do korzystania z WeatherKit

Aby używać WeatherKit w swojej aplikacji, musisz zarejestrować dla niej niestandardowy identyfikator aplikacji, co wymaga płatnego konta programisty. Następnie dodasz funkcję WeatherKit do projektu Xcode swojej aplikacji. Zobaczmy, jak to zrobić za pomocą aplikacji Let's Eat. Wykonaj następujące kroki:

1. Sprawdź, czy dodałeś płatne konto programisty do Xcode i czy Twoja aplikacja Let's Eat może działać na komputerze Mac lub urządzeniu z systemem iOS.

2. W Xcode, w nawigatorze projektu wybierz projekt LetsEat i kliknij zakładkę Podpisywanie i możliwości.

3. Skopiuj identyfikator pakietu w polu Identyfikator pakietu w celu LetsEat. Powinno to wyglądać mniej więcej tak: com.macapp.myname.LetsEat:



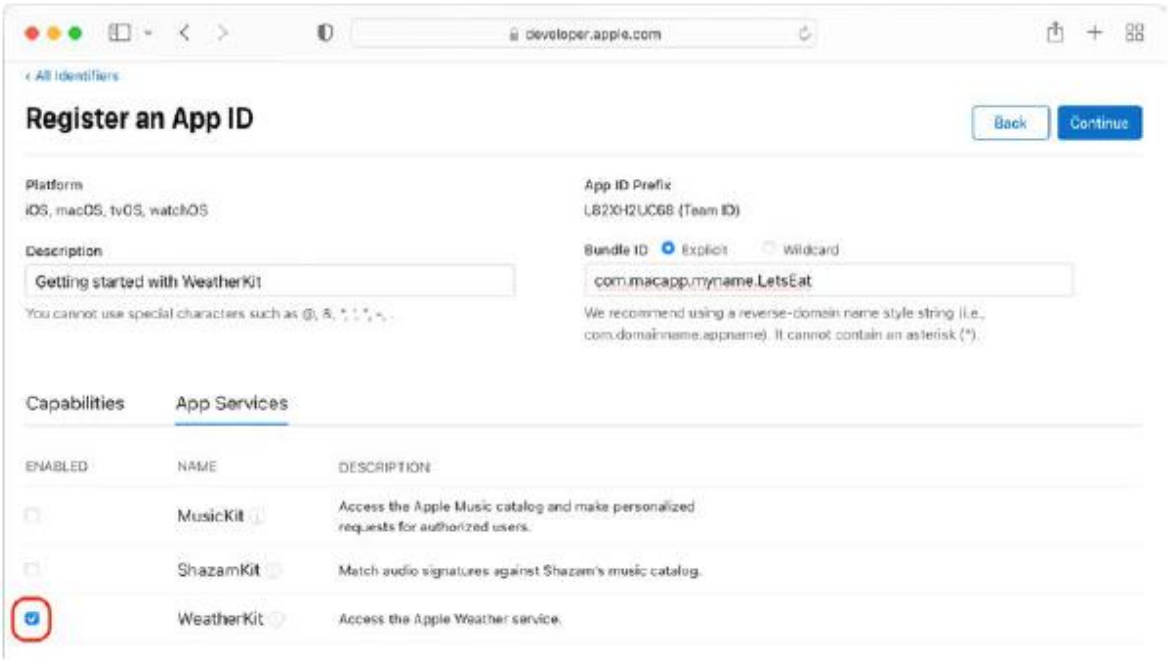
4. W przeglądarce Safari przejdź do https://developer.apple.com i zaloguj się na swoje płatne konto programisty.

5. Przejdź do sekcji Certyfikaty, identyfikatory i profile w witrynie programisty.

6. Wybierz Identyfikatory i kliknij przycisk Dodaj, aby utworzyć nowy identyfikator aplikacji dla aplikacji Let's Eat. Postępuj zgodnie z instrukcjami, aż dojdziesz do strony Zarejestruj identyfikator aplikacji.

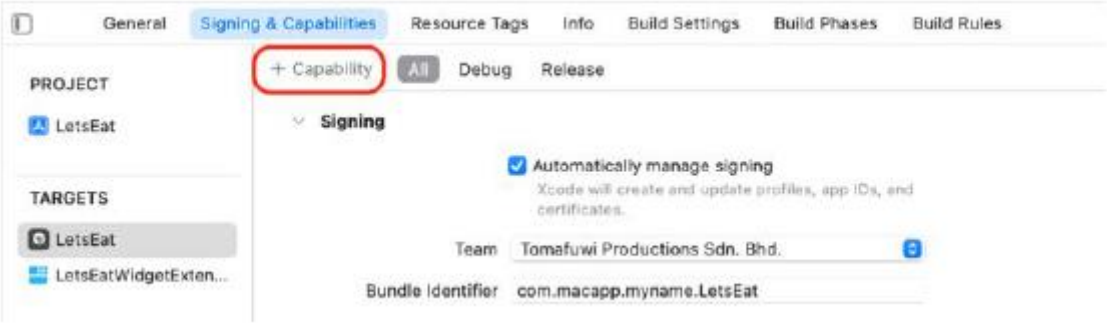
7. Jako Identyfikator pakietu wybierz Jawny i wprowadź identyfikator pakietu skopiowany w kroku 3.

8. Kliknij zakładkę App Services i zaznacz pole wyboru WeatherKit:

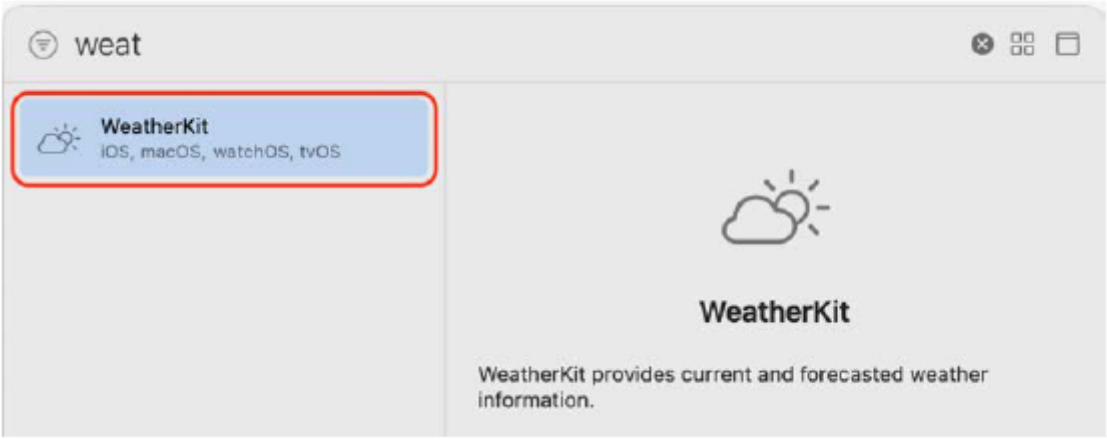


9. Zakończ proces tworzenia identyfikatora aplikacji i poczekaj 30 minut, aż usługa zarejestruje identyfikator pakietu Twojej aplikacji.

10. Podczas oczekiwania przejdź do panelu Podpisywanie i możliwości w Xcode i kliknij przycisk +, aby dodać funkcję do aplikacji Let's Eat:



11. W wyświetlonym oknie wyszukaj i kliknij dwukrotnie opcję WeatherKit, aby dodać ją do aplikacji Let's Eat.



Utworzyłeś identyfikator aplikacji dla swojej aplikacji i dodałeś do niej funkcję WeatherKit. W następnej sekcji utworzysz nową klasę menedżera danych WeatherDataManager, która będzie pobierać informacje WeatherKit z usługi Apple Weather Service.

Tworzenie klasy WeatherDataManager

Twoja aplikacja ma teraz identyfikator aplikacji zarejestrowany w usłudze Apple Weather Service i dodałeś do niej funkcję WeatherKit. Teraz utworzysz nową klasę WeatherDataManager, która będzie pobierać aktualną pogodę w lokalizacji konkretnej restauracji. Wykonaj następujące kroki:

1. Kliknij prawym przyciskiem myszy folder Misc i wybierz Nowy plik.

2. iOS powinien być już wybrany. Wybierz opcję Plik Swift, a następnie kliknij Dalej.

3. Nadaj plikowi nazwę WeatherDataManager i kliknij Utwórz, aby wyświetlić jego zawartość w obszarze Edytora.

4. Dodaj następujący kod po instrukcji import Foundation:

```
import CoreLocation

struct WeatherDataManager {

private let weatherService = WeatherService()

func fetchWeather(at restaurantLocation:

CLLocation) async -> (String, String) {
```

```
let weather = try? await weatherService.weather(

for: restaurantLocation)

if let weather = weather {

let temperature = weather.currentWeather

.temperature

let symbol = weather.currentWeather

.symbolName

return (symbol, temperature.formatted())

}

return ("square.dashed","--°F")

}

}
```

Rozbijmy to:

```
import WeatherKit

import CoreLocation
```

Spowoduje to zaimportowanie środowiska WeatherKit wymaganego do uzyskania dostępu do usługi Apple Weather Service oraz środowiska CoreLocation wymaganego do pracy z lokalizacjami geograficznymi.

```
struct WeatherDataManager {
```

To deklaruje strukturę WeatherDataManager.

```
private let weatherService = WeatherService()
```

Spowoduje to inicjowanie obiektu WeatherService(), który służy jako punkt wejścia do usługi Apple Weather Service, i przypisanie go do stałej WeatherService.

```
func fetchWeather(at restaurantLocation:

CLLocation) async -> (String, String) {
```

Deklaruje metodę pobierającą pogodę z położenia geograficznego restauracji.

```
let weather = try? await weatherService.weather(

for: restaurantLocation)
```

Spowoduje to zapytanie usługi Apple Weather Service o pogodę w podanej lokalizacji.

```
if let weather = weather {

let temperature = weather.currentWeather

.temperature
```

let symbol = weather.currentWeather

.symbolName

return (symbol, temperature.formatted())

}

Jeśli zapytanie zakończyło się pomyślnie i zwrócony zostanie obiekt pogody, pobierz z niego wartości temperatury i symbolName i zwróć je jako krotkę zawierającą dwa ciągi znaków.

return ("square.dashed","--°F")

Jeśli zapytanie nie powiodło się, zwróć krotkę („square.dashed”, „--°F”).

Utworzyłeś nową klasę WeatherDataManager zawierającą metodę, która zwróci krotkę opisującą warunki pogodowe w lokalizacji restauracji. W następnej sekcji zmodyfikujesz klasę RestaurantDetailViewController, tak aby mogła wyświetlać pogodę dla restauracji na ekranie Szczegóły restauracji.

Wyświetlanie danych o pogodzie na ekranie szczegółów restauracji

Skonfigurowałeś aplikację Let’s Eat, rejestrując jej identyfikator aplikacji i dodając do niej funkcję WeatherKit, a także utworzyłeś klasę WeatherDataManager do pobierania pogody z określonej lokalizacji. Teraz zmodyfikujesz klasę RestaurantDetailViewController, aby wyświetlać informacje o pogodzie dla restauracji na ekranie Szczegóły restauracji. Wykonaj następujące kroki:

1. Kliknij plik RestaurantDetailViewController w nawigatorze projektu. Dodaj nową właściwość, która będzie przechowywać instancję WeatherDataManager() po właściwości wybranej restauracji:

var selectedRestaurant: RestaurantItem?

let weatherDataManager = WeatherDataManager()

2. Dodaj następujący kod do rozszerzenia prywatnego po metodzie inicjalizacji(), aby zaimplementować metodę displayWeather:

func displayWeather() {

Task {

if let lat = selectedRestaurant?.lat, let

long = selectedRestaurant?.long {

let restLocation = CLLocation(latitude:

lat, longitude: long)

let restWeather = await weatherDataManager

.fetchWeather(at: restLocation)

let largeTitle = UIImage.SymbolConfiguration

(textStyle: .largeTitle)

let bold = UIImage.SymbolConfiguration

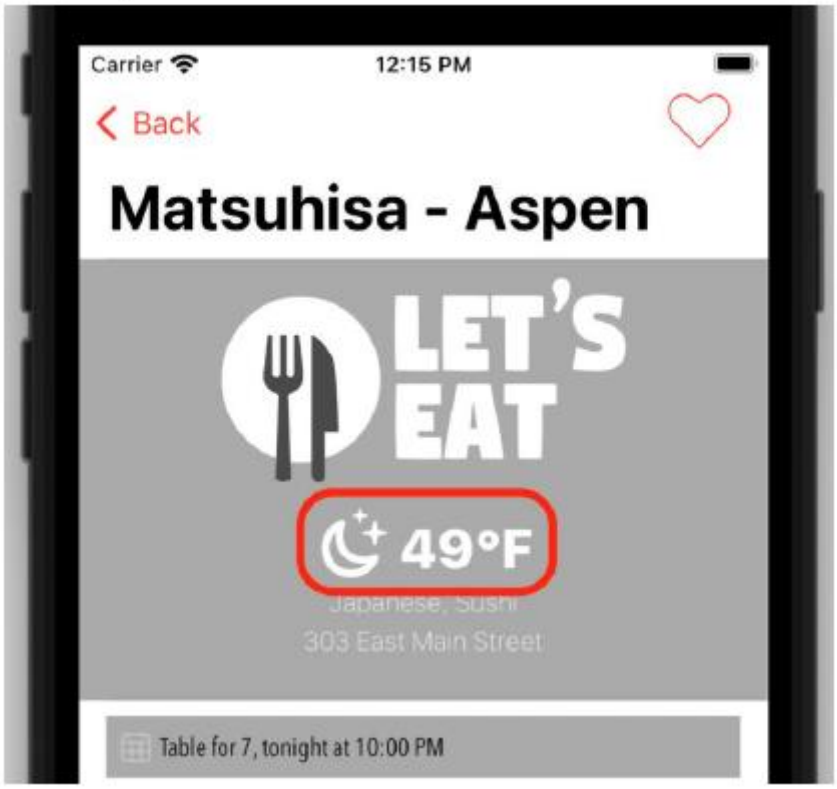
```
(weight: .bold)

let combined = largeTitle.applying(bold)

let weatherImage = UIImage(systemName:

restWeather.0, withConfiguration:

combined)?.withTintColor(.white)

let attachment = NSTextAttachment()

attachment.image = weatherImage

let imageString = NSMutableAttributedString

(attachment: attachment)

let textString = NSAttributedString(string:

" " + restWeather.1)

imageString.append(textString)

nameLabel.attributedText = imageString

nameLabel.sizeToFit()

}

}

}
```

Ta metoda pobiera szerokość i długość geograficzną restauracji oraz tworzy instancję CLLocation, która jest następnie przekazywana do metody fetchWeather(at:) instancji WeatherDataManager. Zwrócona krotka służy do utworzenia instancji NSAttributedString, która jest przypisana do właściwości atrybuttext nameLabel i będzie widoczna na ekranie Szczegóły restauracji.

3. W metodzie inicjalizacji() dodaj wywołanie metody displayWeather() przed ostatnim nawiasem klamrowym:

```
func initialize() {

setupLabels()

createMap()

createRating()

displayWeather()

}
```

Kompiluj i uruchamiaj swoją aplikację. Informacje o pogodzie dla restauracji będziesz mógł zobaczyć na ekranie szczegółów restauracji:



Gratulacje! Właśnie wdrożyłeś WeatherKit dla aplikacji Let’s Eat!

Streszczenie

Zmodyfikowałeś aplikację Let's Eat, aby używać WeatherKit do uzyskiwania informacji o pogodzie dla konkretnej restauracji i wyświetlania ich na ekranie szczegółów restauracji. Zacząłeś od nauczenia się, jak działa WeatherKit. Następnie utworzyłeś identyfikator aplikacji i dodałeś funkcję WeatherKit do aplikacji Let’s Eat. Następnie utworzyłeś nową klasę menedżera danych, WeatherDataManager, aby uzyskać pogodę z lokalizacji określonej restauracji. Na koniec zaktualizowano klasę RestaurantDetailViewController, aby wyświetlała pogodę w lokalizacji restauracji na ekranie Szczegóły restauracji. Znasz teraz podstawy WeatherKit i będziesz mógł dodawać niestandardowe działania grupowe do swoich własnych aplikacji.

Testowanie i przesyłanie aplikacji do App Store

Gratulacje! Dotarłeś do ostatniego rozdziału tej książki! Z tej książki dowiedziałeś się o języku programowania Swift i o tym, jak zbudować całą aplikację przy użyciu Xcode. Jednak uruchamiałeś aplikację tylko w symulatorze iOS lub na własnym urządzeniu, korzystając z bezpłatnego konta Apple Developer. W tym rozdziale zaczniesz od nauczenia się, jak uzyskać płatne konto Apple Developer. Następnie dowiesz się o certyfikatach, identyfikatorach, rejestracji urządzeń testowych i profilach udostępniania. Następnie dowiesz się, jak utworzyć wpis w App Store i przesłać aplikację do App Store. Na koniec dowiesz się, jak przeprowadzać testy aplikacji przy użyciu testerów wewnętrznych i zewnętrznych. Pod koniec tego rozdziału dowiesz się, jak tworzyć i przesyłać aplikacje do App Store oraz jak przeprowadzać wewnętrzne i zewnętrzne testy swojej aplikacji.

Zakładanie konta programisty Apple

Jak widzieliście we wcześniejszych rozdziałach, wszystko, czego potrzebujesz, aby przetestować aplikację na urządzeniu, to bezpłatny Apple ID. Aplikacje będą jednak działać tylko przez kilka dni i nie będzie można dodawać zaawansowanych funkcji, takich jak Zaloguj się za pomocą konta Apple, ani przesyłać aplikacji do App Store. Do tego potrzebne jest płatne konto Apple Developer. Wykonaj poniższe kroki, aby kupić indywidualne/samodzielne konto programisty Apple:

1. Przejdź do https://developer.apple.com/programs/ i kliknij przycisk Zarejestruj.

2. Przewiń w dół ekranu i kliknij opcję Rozpocznij rejestrację.

3. Po wyświetleniu monitu wprowadź swój Apple ID i hasło.

4. Na zaufaj tej przeglądarce? kliknij opcję Zaufaj tylko, jeśli jesteś jedyną osobą korzystającą z tej przeglądarki. W przeciwnym razie kliknij opcję Nie teraz. Ma to na celu ochronę informacji o Twoim koncie.

5. Kliknij Kontynuuj rejestrację w internecie >.

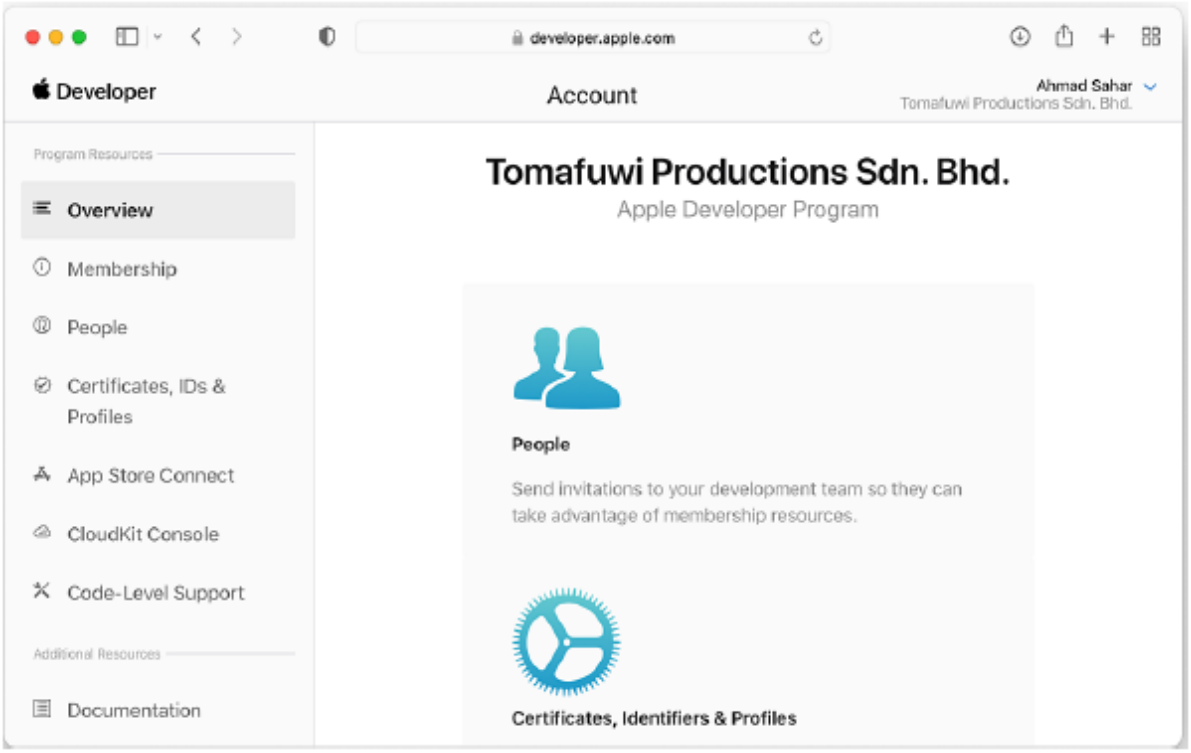
6. Na ekranie Potwierdź swoje dane osobowe wprowadź swoje dane osobowe i po zakończeniu kliknij przycisk Kontynuuj.

7. Na ekranie Wybierz typ podmiotu wybierz opcję Osoba fizyczna/samodzielny właściciel. Kliknij Kontynuuj.

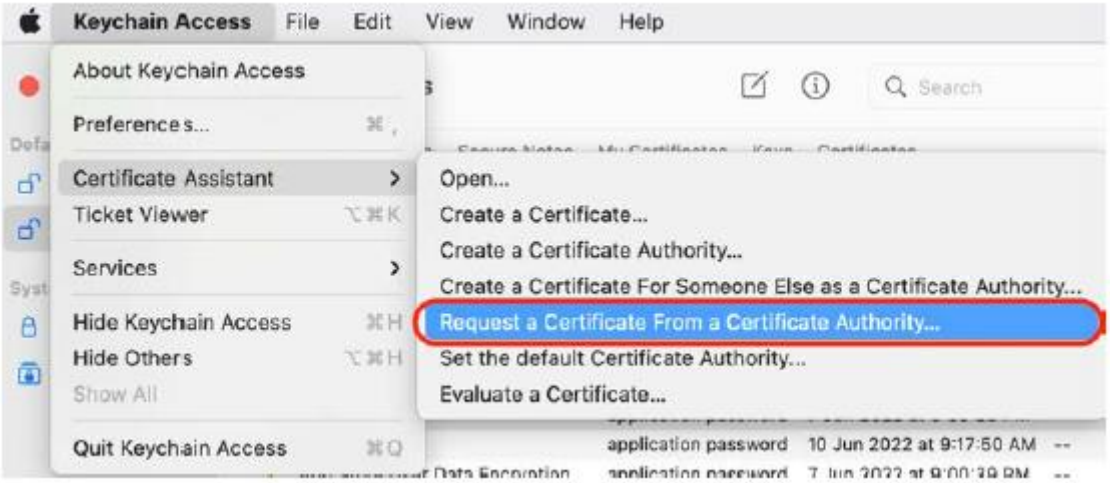
8. Na ekranie Przejrzyj i zaakceptuj zaznacz pole wyboru u dołu strony i kliknij Kontynuuj.

9. Na ekranie Zakończ zakup kliknij Kup.

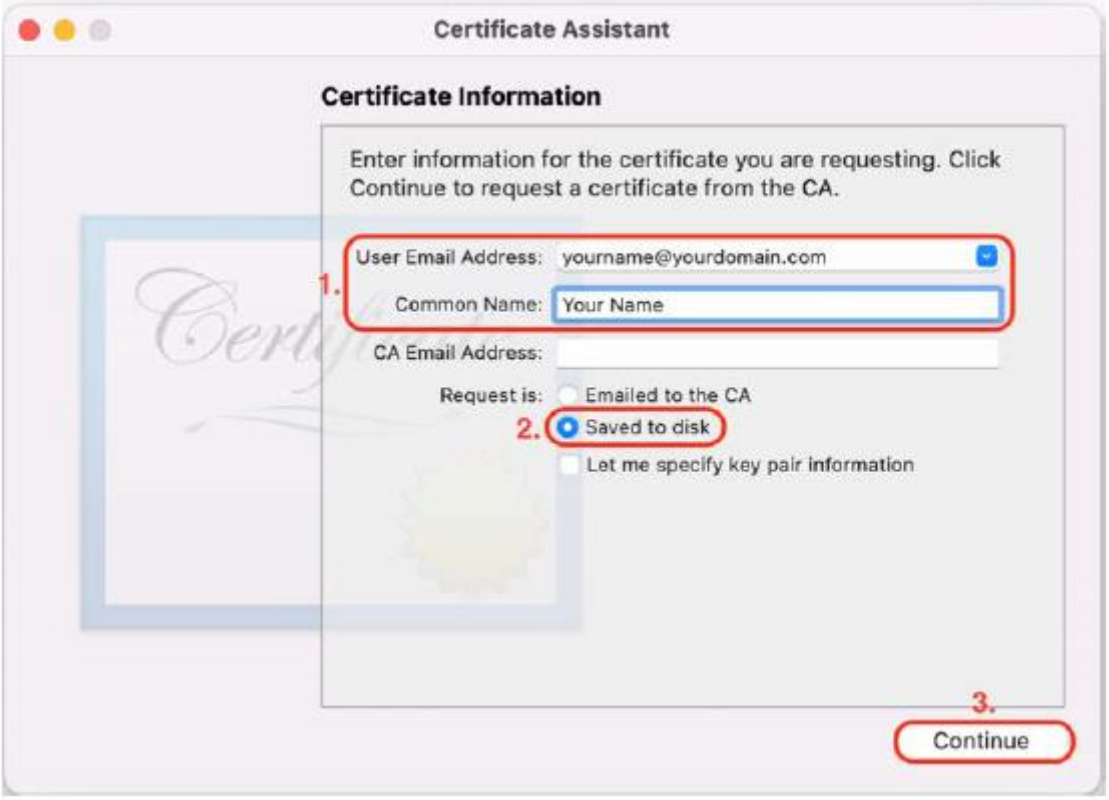
10. Postępuj zgodnie ze wskazówkami wyświetlanymi na ekranie, aby sfinalizować zakup. Po zakupie konta przejdź do https://developer.apple.com/account/ i zaloguj się przy użyciu tego samego Apple ID, którego użyłeś do zakupu konta programisty. Powinieneś zobaczyć coś podobnego do poniższego:



2. Wybierz Asystenta certyfikatu | Poproś o certyfikat od urzędu certyfikacji... z menu Dostęp do pęku kluczy:



3. W polu Adres e-mail użytkownika wprowadź adres e-mail Apple ID użyty do rejestracji konta Apple Developer. W polu Nazwa zwyczajowa wpisz swoje imię i nazwisko. Wybierz opcję Zapisano na dysku w obszarze Żądanie to: i kliknij Kontynuuj:



4. Zapisz CSR na dysku twardym.

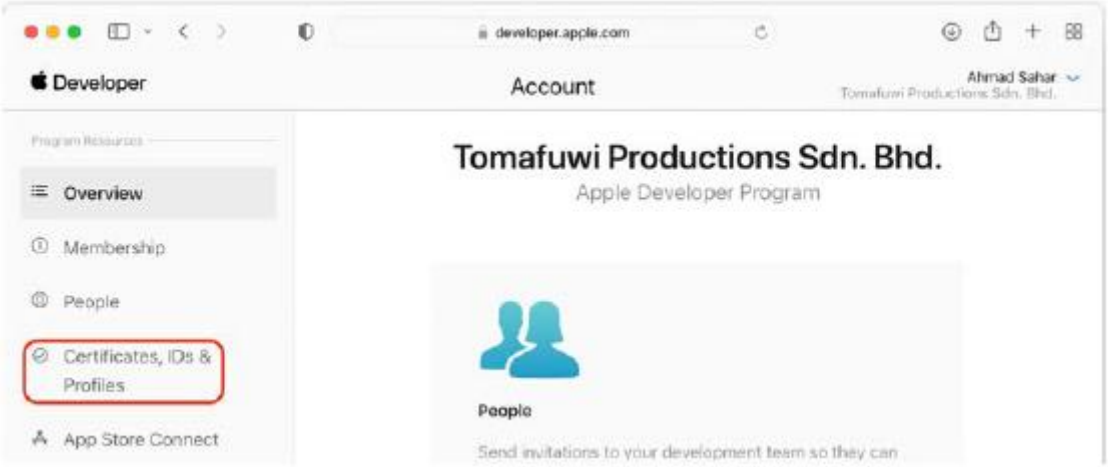
5. Kliknij Gotowe.

Teraz, gdy masz już CSR, przyjrzyjmy się, jak go użyjesz, aby uzyskać certyfikaty programistyczne (do testowania na własnym urządzeniu) i certyfikaty dystrybucyjne (do przesłania do App Store) w następnej sekcji.

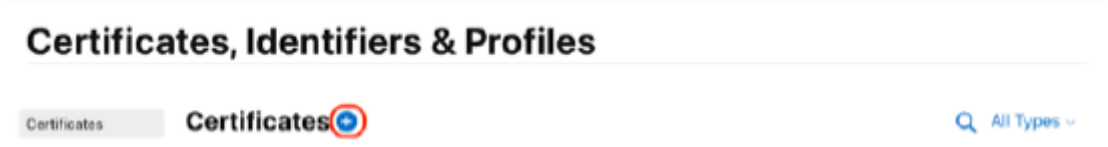
Tworzenie certyfikatów deweloperskich i dystrybucyjnych

Po otrzymaniu żądania podpisania certyfikatu możesz go użyć do utworzenia certyfikatów programistycznych i dystrybucyjnych. Certyfikaty programistyczne są używane, gdy chcesz przetestować aplikację na urządzeniach testowych, a certyfikaty dystrybucyjne są używane, gdy chcesz przesłać aplikację do App Store. Oto jak utworzyć certyfikaty rozwojowe i dystrybucyjne:

1. Zaloguj się na swoje konto Apple Developer i kliknij Certyfikaty, identyfikatory i profile:

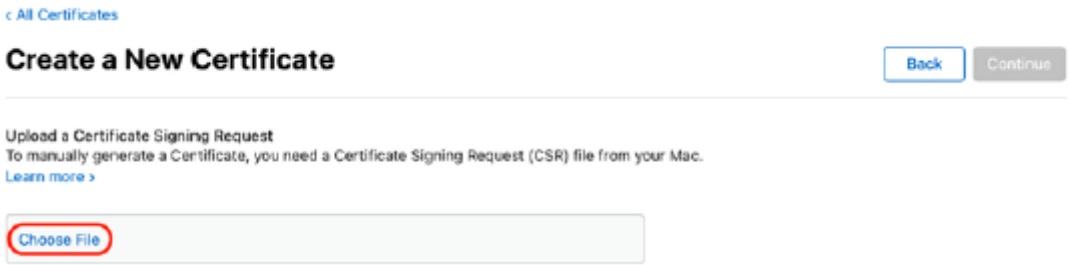


2. Zobaczysz ekran Certyfikaty. Kliknij przycisk +:



3. Kliknij przycisk opcji Apple Development i kliknij Kontynuuj:

4. Kliknij Wybierz plik:



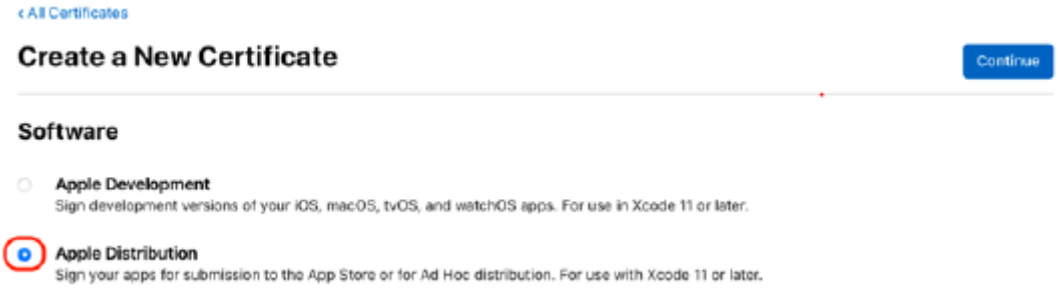
5. Prześlij swój CSR, wybierając opcję Wybierz plik w obszarze Prześlij plik CSR, wybierając plik CSR zapisany wcześniej na dysku twardym i klikając Wybierz.

6. Kliknij Kontynuuj:

7. Twój certyfikat zostanie wygenerowany automatycznie. Kliknij Pobierz, aby pobrać wygenerowany certyfikat na komputer Mac:7. Twój certyfikat zostanie wygenerowany automatycznie. Kliknij Pobierz, aby pobrać wygenerowany certyfikat na komputer Mac:

8. Kliknij dwukrotnie pobrany certyfikat, aby zainstalować go na komputerze Mac.

9. Powtórz kroki 3-8 ponownie, ale tym razem wybierz opcję Dystrybucja Apple w kroku 3:



Świetnie! Masz teraz certyfikaty rozwojowe i dystrybucyjne. Następnym krokiem jest zarejestrowanie identyfikatora aplikacji w celu identyfikacji jej w App Store. Dowiesz się, jak to zrobić w następnej sekcji.

Rejestrowanie identyfikatora aplikacji

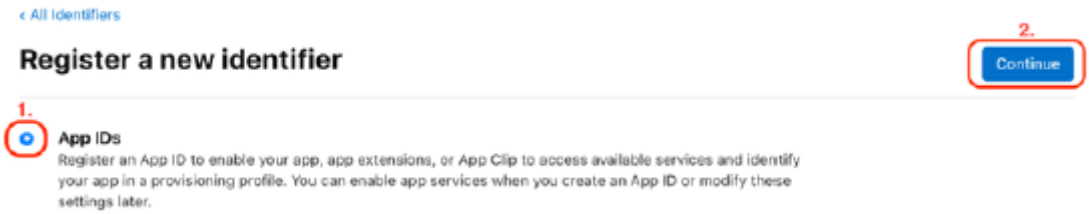
Kiedy tworzyłeś projekt w Rozdziale 1, Zapoznanie się z Xcode, utworzyłeś dla niego identyfikator pakietu (znany również jako identyfikator aplikacji). Identyfikator aplikacji służy do identyfikowania Twojej aplikacji w sklepie App Store. Przed przesłaniem aplikacji do App Store musisz zarejestrować ten identyfikator aplikacji na swoim koncie programisty. Oto jak zarejestrować swój identyfikator aplikacji:

1. Zaloguj się na swoje konto Apple Developer i kliknij Certyfikaty, identyfikatory i profile.

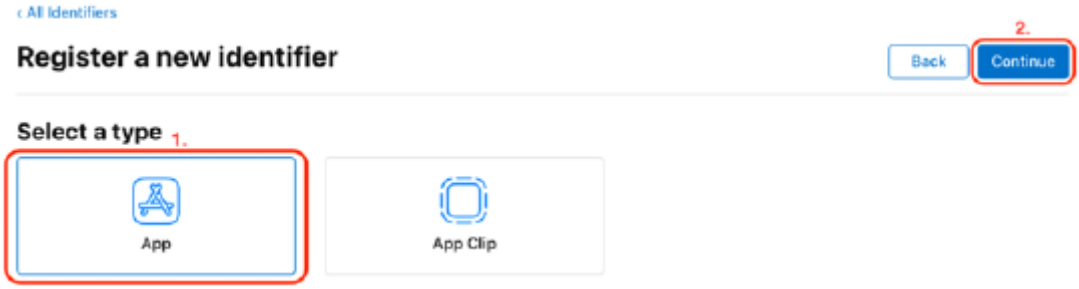
2. Kliknij Identyfikatory.

3. Kliknij przycisk +:

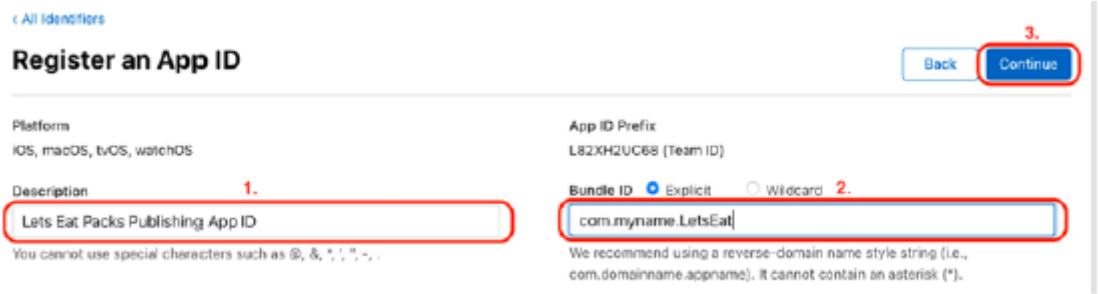
4. Kliknij Identyfikatory aplikacji i kliknij Kontynuuj:



5. Kliknij opcję Aplikacja i kliknij Kontynuuj:



6. iOS powinien być już wybrany. Wprowadź opis tego identyfikatora aplikacji, na przykład identyfikator aplikacji Lets Eat Packt Publishing. Zaznacz przycisk Jasny i wpisz w polu identyfikator pakietu aplikacji. Upewnij się, że ta wartość jest taka sama jak identyfikator pakietu, którego użyłeś podczas tworzenia projektu. Gdy skończysz, kliknij przycisk Kontynuuj:



7. Kliknij Zarejestruj się:

Twój identyfikator aplikacji został teraz zarejestrowany. Fajny! Następnie w następnej sekcji zarejestrujesz urządzenia, na których będziesz testować aplikację.

Rejestracja urządzeń

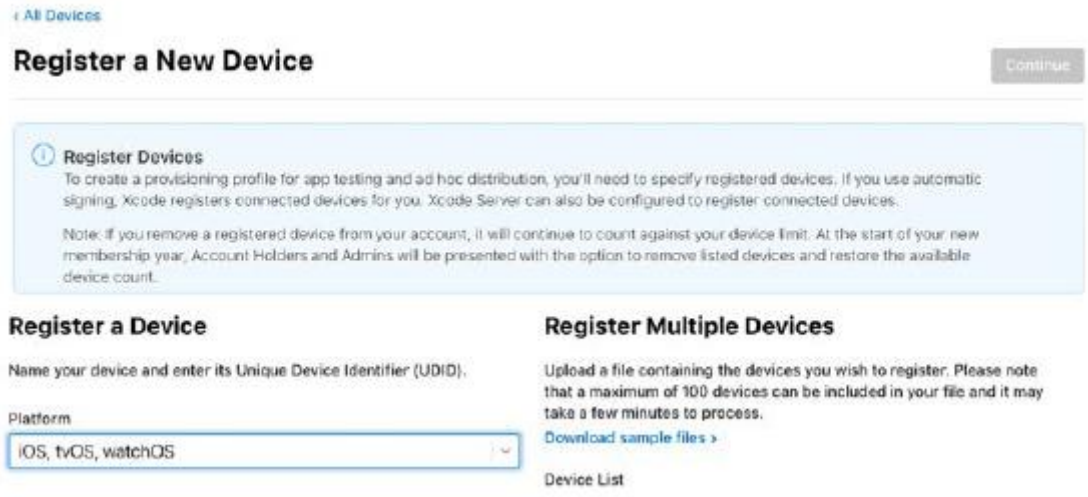
Aby uruchomić aplikacje na urządzeniach osobistych w celu przetestowania, musisz je zarejestrować na swoim koncie programisty. Oto jak zarejestrować swoje urządzenia:

1. Zaloguj się na swoje konto Apple Developer i kliknij Certyfikaty, identyfikatory i profile.

2. Kliknij Urządzenia.

3. Kliknij przycisk +:

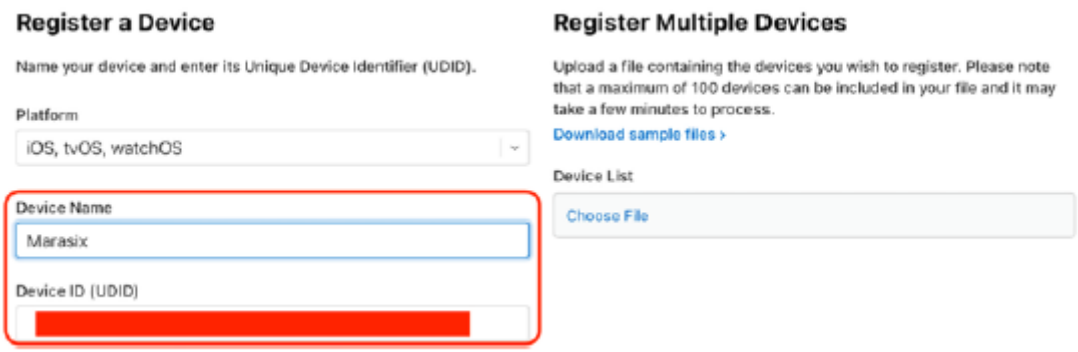
4. Pojawi się ekran Zarejestruj nowe urządzenie:



Aby zarejestrować urządzenie, będziesz potrzebować nazwy urządzenia i identyfikatora urządzenia.

5. Podłącz urządzenie do komputera Mac. Uruchom Xcode i wybierz Urządzenia i symulatory z menu Okno. Wybierz urządzenie w lewym panelu i skopiuj wartość Identyfikatora:

6. Wpisz nazwę urządzenia w polu Nazwa urządzenia i wklej wartość identyfikatora w polu Identyfikator urządzenia (UDID). Kliknij Kontynuuj:



Pomyślnie zarejestrowałeś swoje urządzenia testowe. Świetnie! Następnym krokiem jest utworzenie profili udostępniania. Aby aplikacje mogły działać w teście, wymagany jest profil tworzenia aplikacji na iOS

urządzeń, a w przypadku aplikacji, które zostaną przesłane do App Store, wymagany jest profil dystrybucji iOS App Store. W następnej sekcji utworzysz profile rozwoju i dystrybucji.

Tworzenie profili udostępniania

Konieczne będzie utworzenie dwóch profili udostępniania. Aby aplikacje mogły działać na urządzeniach testowych, wymagany jest profil programistyczny aplikacji na iOS. Profil dystrybucyjny iOS App Store służy do przesyłania aplikacji do App Store. Oto jak utworzyć profil programisty:

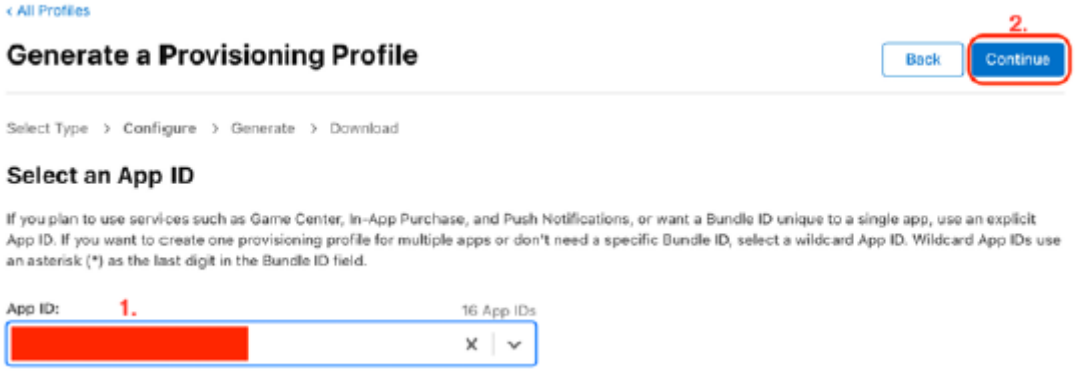
1. Zaloguj się na swoje konto Apple Developer i kliknij Certyfikaty, identyfikatory i profile.

2. Kliknij opcję Profile.

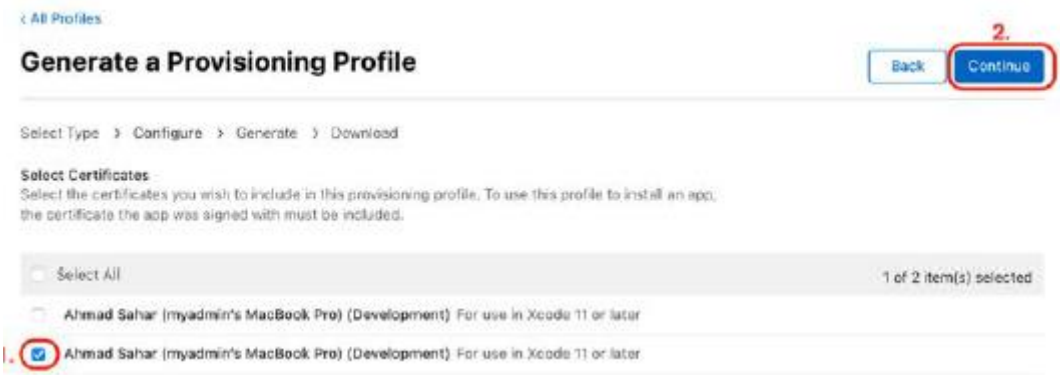
3. Kliknij przycisk +:

4. Kliknij opcję Tworzenie aplikacji na iOS i kliknij Kontynuuj:

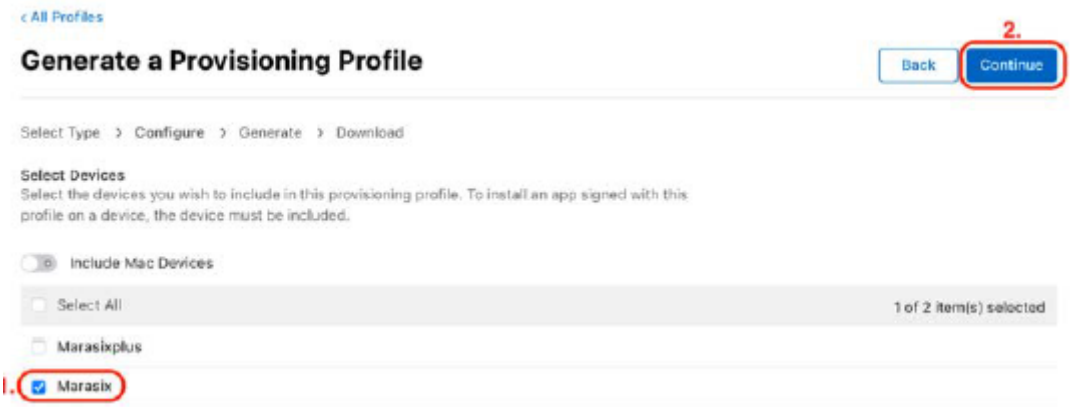
5. Wybierz identyfikator aplikacji, którą chcesz przetestować, i kliknij Kontynuuj:



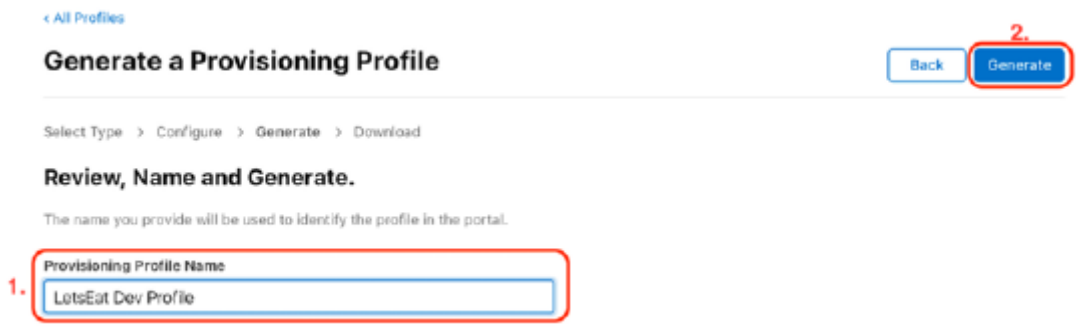
6. Wybierz certyfikat programistyczny i kliknij Kontynuuj:



7. Zaznacz wszystkie urządzenia, na których będziesz testować tę aplikację, i kliknij Kontynuuj:



8. Wpisz nazwę profilu i kliknij Generuj:



9. Kliknij przycisk Pobierz, aby pobrać profil.

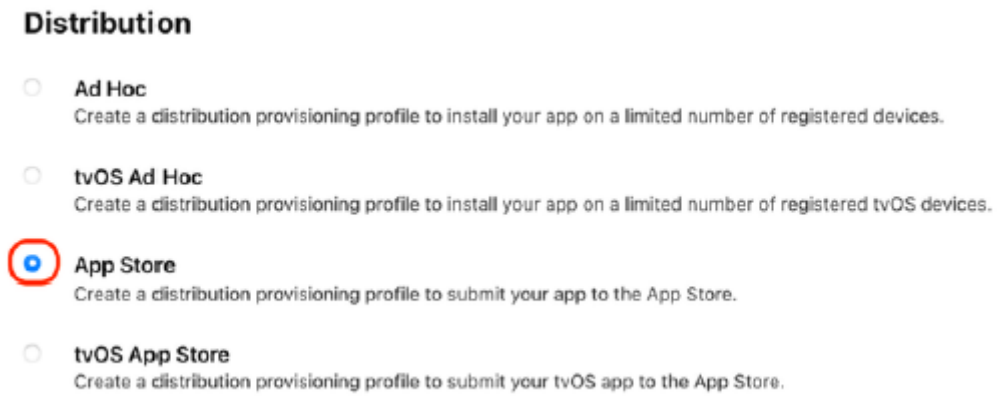
10. Kliknij dwukrotnie profil, aby go zainstalować.

Następnie utworzysz profil dystrybucyjny.

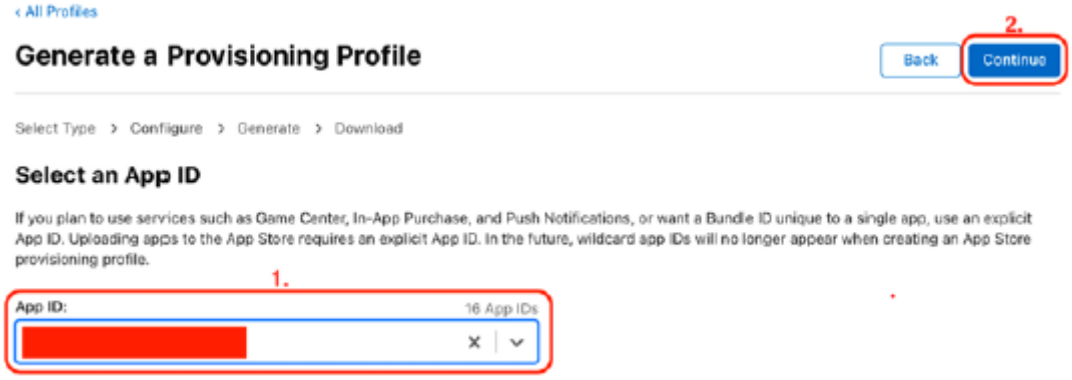
11. Kliknij łącze Wszystkie profile, aby wrócić do poprzedniej strony:

12. Kliknij przycisk +:

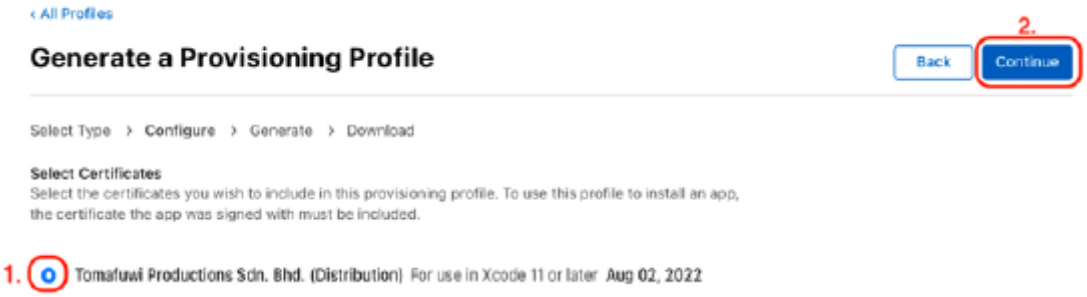
13. Kliknij App Store i kliknij Kontynuuj:



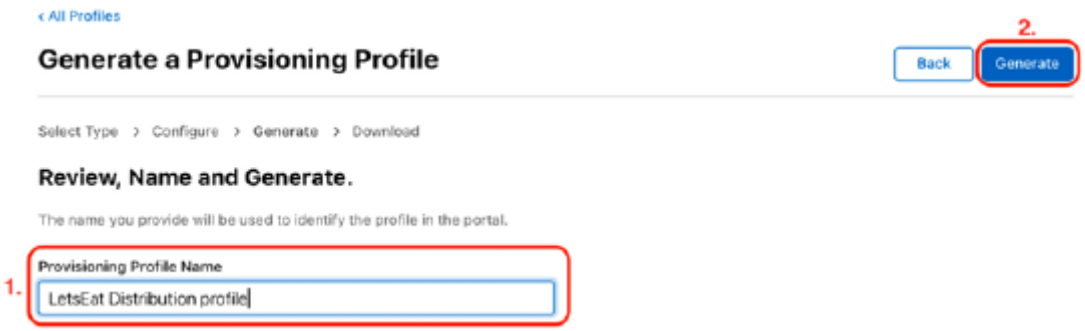
14. Wybierz identyfikator aplikacji, którą chcesz opublikować w App Store i kliknij Kontynuuj:



15. Wybierz certyfikat dystrybucyjny i kliknij Kontynuuj:



16. Wpisz nazwę profilu i kliknij Generuj:



17. Kliknij przycisk Pobierz, aby pobrać profil.

18. Kliknij dwukrotnie profil, aby go zainstalować.

Wykonałeś wszystkie niezbędne kroki przed przesłaniem aplikacji do App Store. Dowiedzmy się więcej o procesie składania zamówienia w następnej sekcji, na przykładzie aplikacji ShareOrder.

Przesyłanie aplikacji do App Store

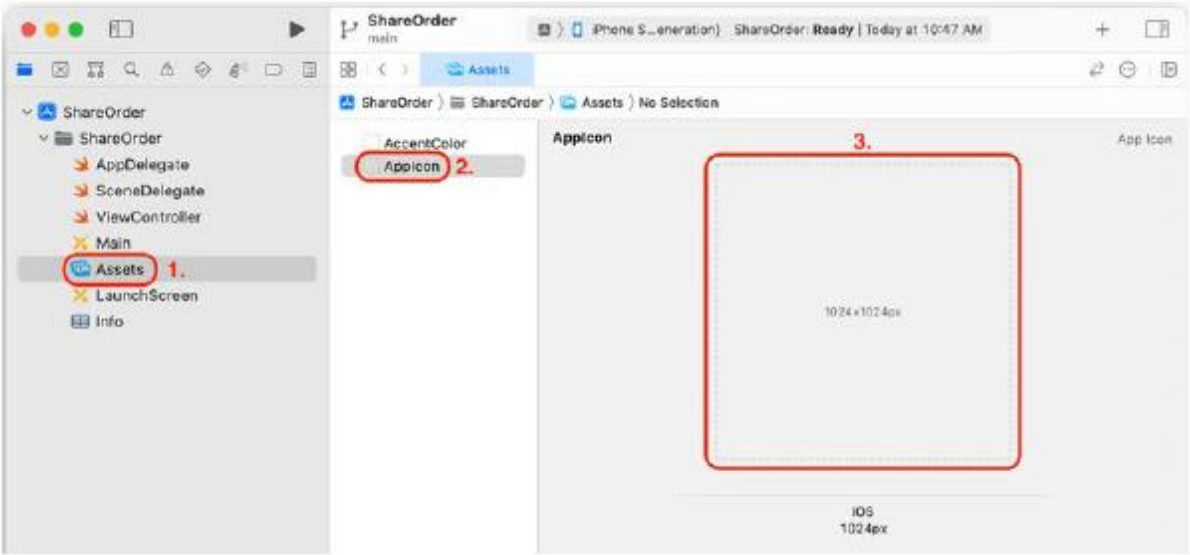
Możesz teraz przesłać swoją aplikację do App Store! W tej sekcji jako przykład zostanie użyta aplikacja ShareOrder. Podsumujmy, co zrobiłeś do tego momentu. Utworzyłeś certyfikaty programistyczne i dystrybucyjne, zarejestrowałeś swój identyfikator aplikacji i urządzenia testowe oraz wygenerowałeś profile programistyczne i dystrybucyjne. Aby przetestować aplikację na urządzeniach testowych, użyjesz certyfikatu programistycznego, identyfikatora aplikacji, zarejestrowanych urządzeń testowych i profilu programistycznego. Aby przesłać aplikację do App Store, użyj certyfikatu dystrybucyjnego, identyfikatora aplikacji i profilu dystrybucyjnego. Skonfigurujesz Xcode tak, aby zarządzał tym automatycznie. Zanim prześlesz aplikację, musisz utworzyć ikony aplikacji i uzyskać zrzuty ekranu przedstawiające aplikację. Następnie możesz utworzyć listę App Store, wygenerować kompilację archiwum do przesłania i uzupełnić informacje w App Store Connect. Następnie Apple sprawdzi Twoją aplikację i, jeśli wszystko pójdzie dobrze, pojawi się ona w App Store. W następnej sekcji zobaczmy, jak utworzyć ikony aplikacji, które pojawią się na ekranie urządzenia po zainstalowaniu aplikacji.

Tworzenie ikon dla Twojej aplikacji

Zanim prześlesz aplikację do App Store, musisz utworzyć dla niej zestaw ikon. Oto jak utworzyć zestaw ikon dla swojej aplikacji:

1. Utwórz ikonę swojej aplikacji o wymiarach 1024 x 1024 pikseli.

2. Kliknij Assets.xcassets w nawigatorze projektu i przeciągnij utworzoną ikonę do przestrzeni pokazanej na poniższym zrzucie ekranu:



Po uruchomieniu aplikacji w symulatorze lub urządzeniu i zamknięciu aplikacji ikona aplikacji powinna być widoczna na ekranie głównym. Schludny!

Przyjrzyjmy się dalej, jak tworzyć zrzuty ekranu. Będziesz ich potrzebować do przesłania aplikacji do App Store, aby klienci mogli zobaczyć, jak wygląda Twoja aplikacja. Zrobisz to w następnej sekcji.

Tworzenie zrzutów ekranu dla Twojej aplikacji

Będziesz potrzebować zrzutów ekranu swojej aplikacji, które zostaną użyte na Twojej liście App Store. Aby je utworzyć, uruchom aplikację w symulatorze i kliknij przycisk zrzutu ekranu. Zostanie zapisany na pulpicie:

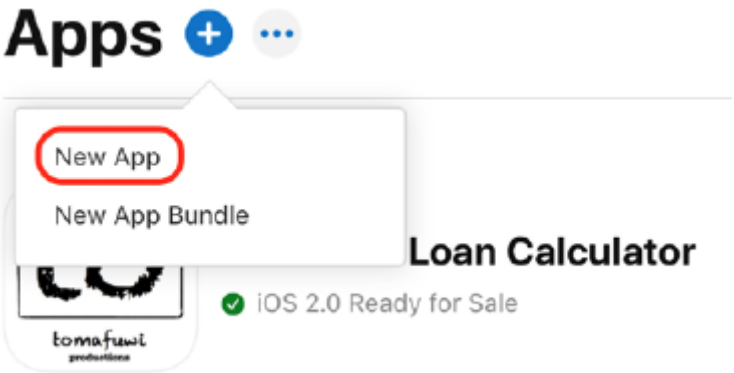
Skorzystaj z symulatorów iPhone'a 14 Pro Max, iPhone'a 8 Plus, iPada Pro (12,9 cala) (3. generacji) i iPada Pro (12,9 cala) (2. generacji) i wykonaj na każdym z nich kilka zrzutów ekranu przedstawiających różne funkcje Twojej aplikacji. Powodem, dla którego musisz używać tych wszystkich symulatorów, jest to, że będziesz potrzebować zrzutów ekranu aplikacji działającej na ekranach o różnych rozmiarach, co zostanie omówione bardziej szczegółowo w następnej sekcji, w której dowiesz się, jak utworzyć wpis w App Store. Lista App Store zawiera wszystkie informacje o Twojej aplikacji, które będą wyświetlane w App Store, dzięki czemu klienci mogą podjąć świadomą decyzję o pobraniu lub zakupie Twojej aplikacji.

Tworzenie wpisu w App Store

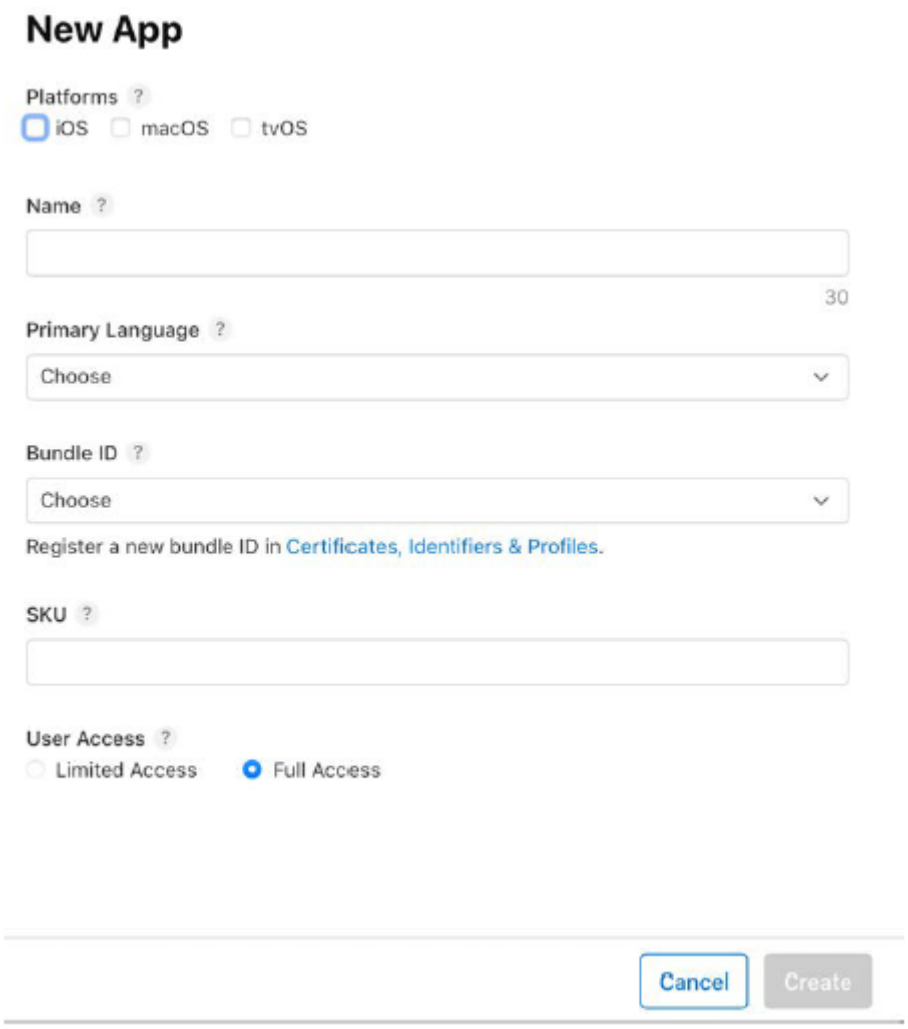
Teraz, gdy masz już ikony i zrzuty ekranu swojej aplikacji, utworzysz listę App Store. Dzięki temu klienci mogą zobaczyć informacje o Twojej aplikacji przed jej pobraniem. Wykonaj następujące kroki:

1. Przejdź do http://appstoreconnect.apple.com i wybierz Moje aplikacje:

2. Kliknij przycisk + w lewym górnym rogu ekranu i wybierz opcję Nowa aplikacja:



3. Pojawi się ekran Nowa aplikacja zawierający listę pól:



Wprowadź dane swojej aplikacji:

Platformy: wszystkie platformy obsługiwane przez Twoją aplikację (iOS, macOS i/lub tvOS).

Nazwa: nazwa Twojej aplikacji.

Język podstawowy: język używany przez Twoją aplikację.

Identyfikator pakietu: Identyfikator pakietu, który utworzyłeś wcześniej.

SKU: dowolny numer referencyjny lub ciąg znaków używany w odniesieniu do Twojej aplikacji.

Dostęp użytkownika: zarządza tym, kto w Twoim zespole ds. kont programistów może zobaczyć tę aplikację w App Store

Łączyć. Jeśli jesteś jedyną osobą w swoim zespole, po prostu ustaw opcję Pełny dostęp.

Kiedy skończysz, kliknij Utwórz.

Aplikacja będzie teraz widoczna na Twoim koncie, ale nadal musisz przesłać aplikację i wszystkie informacje na jej temat. Aby przesłać aplikację, musisz utworzyć kompilację archiwum. Dowiesz się, jak to zrobić w następnej sekcji.

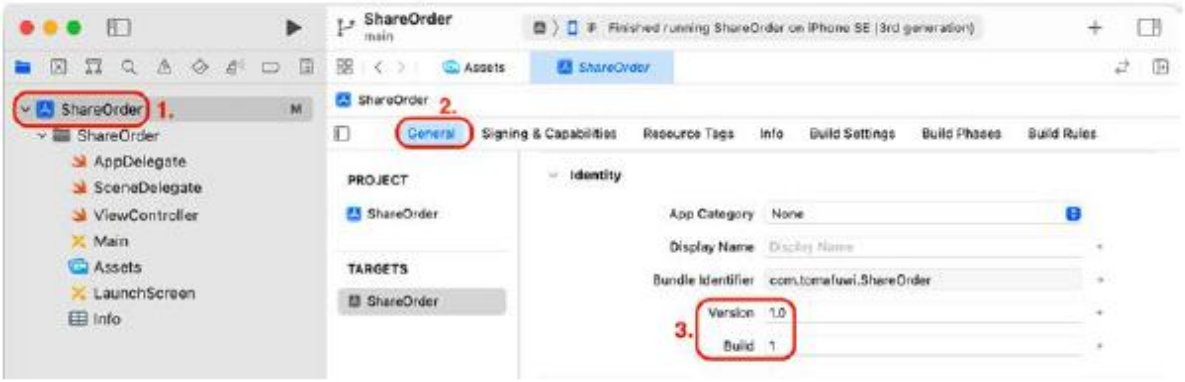
Kiedy skończysz, kliknij Utwórz.

Aplikacja będzie teraz widoczna na Twoim koncie, ale nadal musisz przesłać aplikację i wszystkie informacje na jej temat. Aby przesłać aplikację, musisz utworzyć kompilację archiwum. Dowiesz się, jak to zrobić w następnej sekcji.

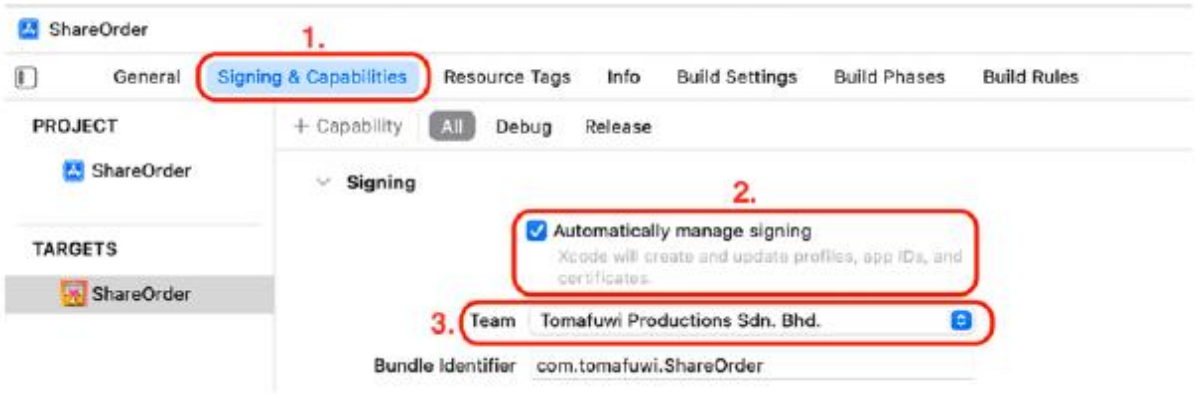
Tworzenie kompilacji archiwum

Utworzysz kompilację archiwum, która zostanie przesłana do Apple w celu umieszczenia w App Store. Będzie to również wykorzystywane do testów wewnętrznych i zewnętrznych. Oto kroki tworzenia kompilacji archiwum:

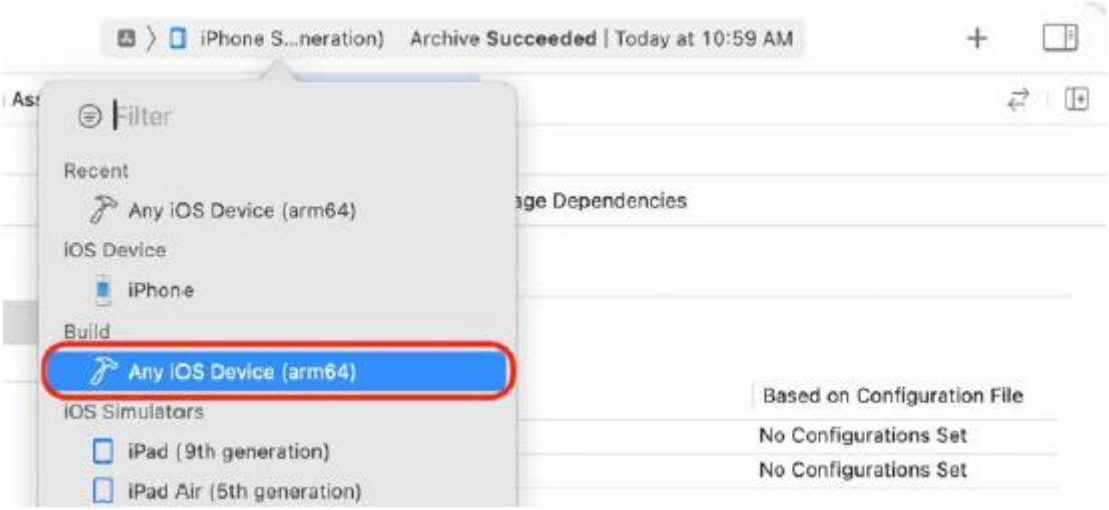
1. Otwórz Xcode, wybierz nazwę projektu w nawigatorze projektu i wybierz panel Ogólne. W sekcji Tożsamość możesz zmienić numer wersji i kompilacji według własnego uznania. Na przykład, jeśli jest to pierwsza wersja Twojej aplikacji i po raz pierwszy ją zbudowałeś, możesz ustawić wersję na 1.0 i kompilację na 1:



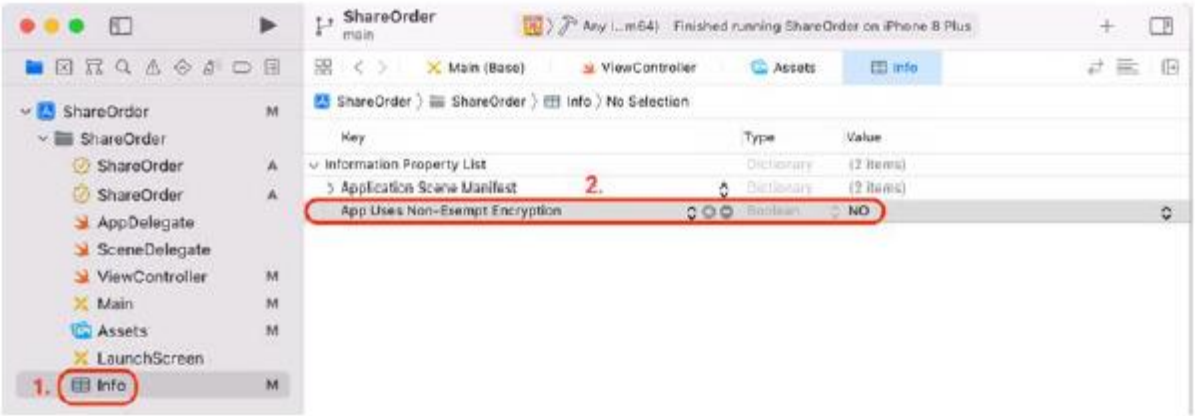
2. Wybierz panel Podpisywanie i możliwości. Upewnij się, że opcja Automatycznie zarządzaj podpisywaniem jest zaznaczona. Umożliwi to Xcode tworzenie certyfikatów, identyfikatorów aplikacji i profili, a także rejestrowanie urządzeń podłączonych do Twojego komputera Mac. Wybierz swoje płatne konto programisty w menu Zespół:



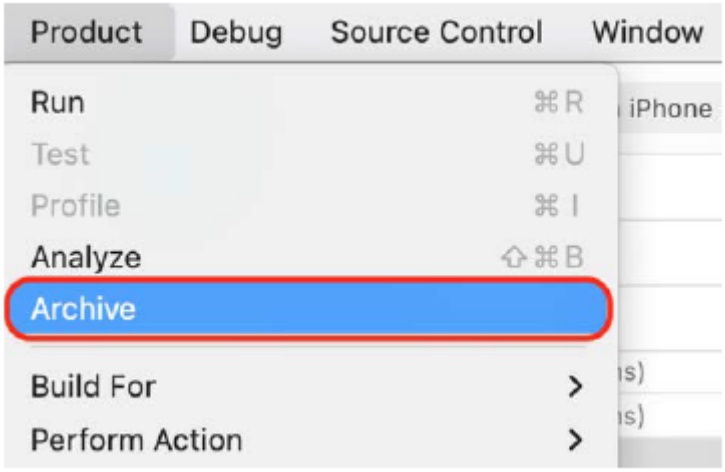
3. Wybierz dowolne urządzenie iOS jako miejsce docelowe kompilacji:



4. Jeśli Twoja aplikacja nie korzysta z szyfrowania, zaktualizuj plik Info.plist, dodając ITSAppUsesNonExemptEncryption, ustawiając jego typ na Boolean i ustawiając jego wartość na NIE:

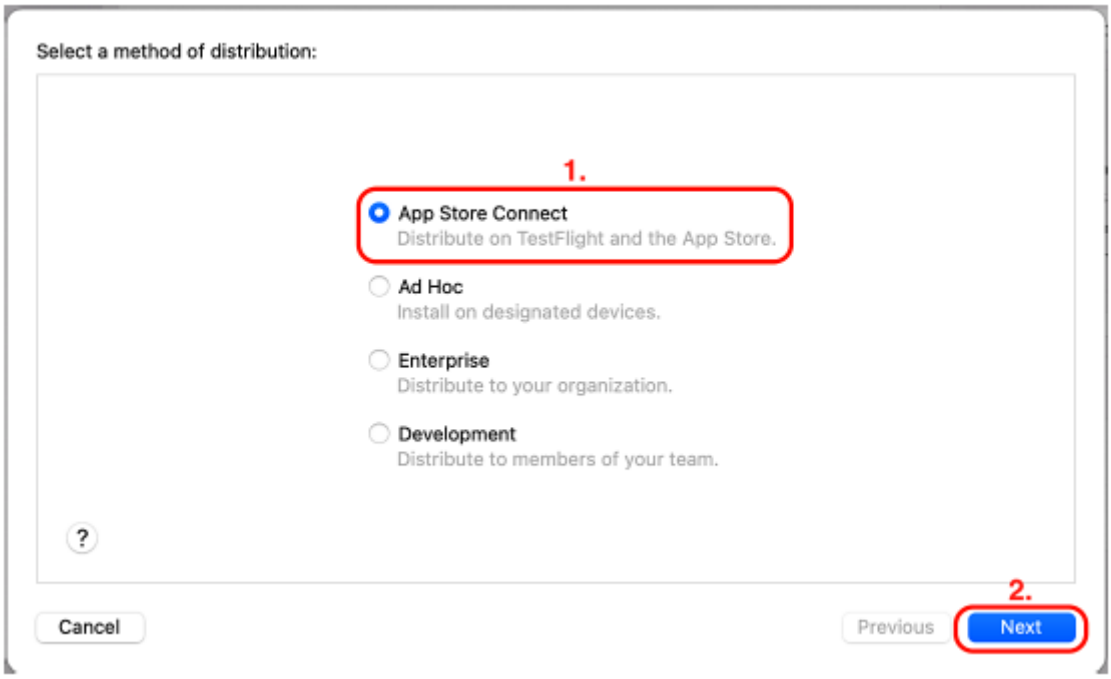


5. Wybierz Archiwum z menu Produkt:

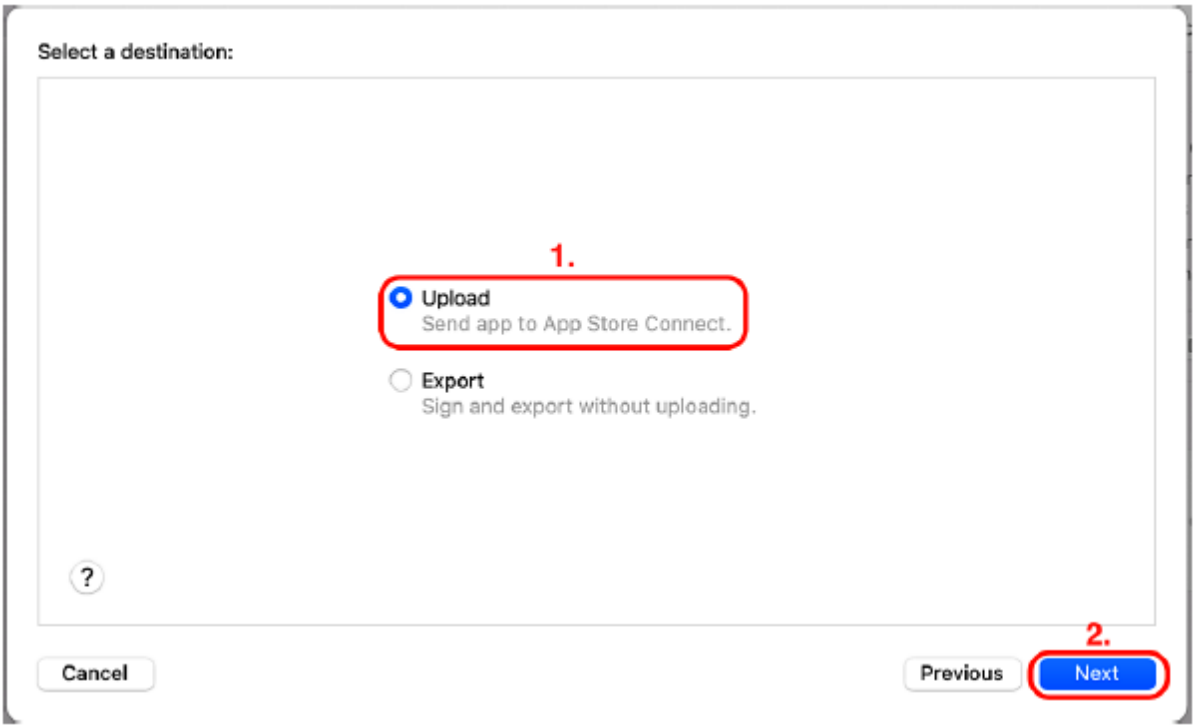


6. Pojawi się okno Organizatora z wybraną zakładką Archiwa. Twoja aplikacja pojawi się na tym ekranie. Wybierz ją i kliknij przycisk Rozpowszechniaj aplikację:

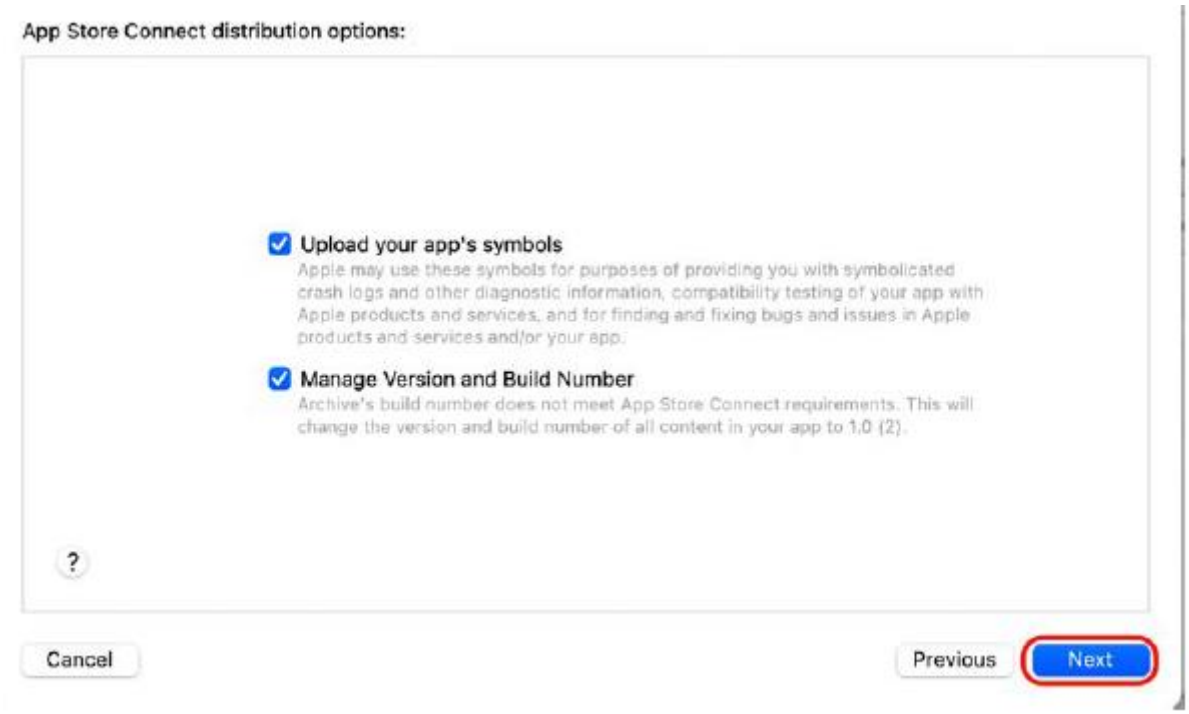
7. Wybierz App Store Connect i kliknij Dalej:



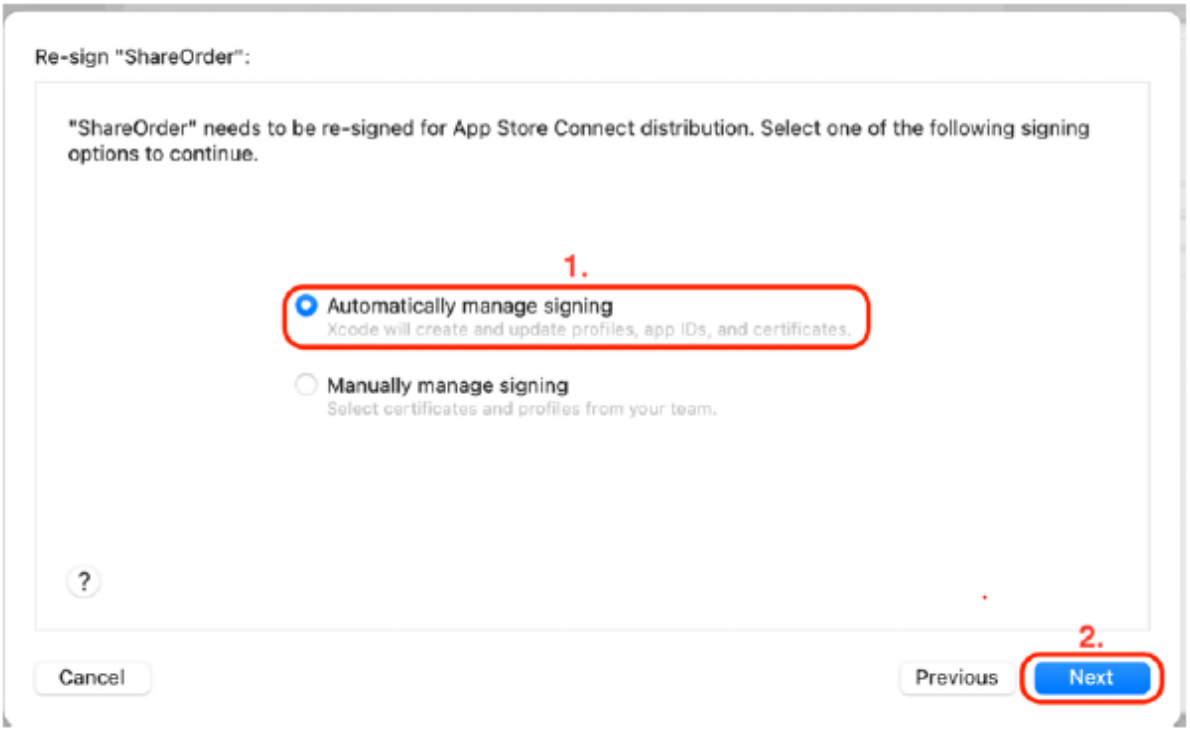
8. Wybierz opcję Prześlij i kliknij Dalej:



9. Pozostaw ustawienia domyślne bez zmian i kliknij Dalej:

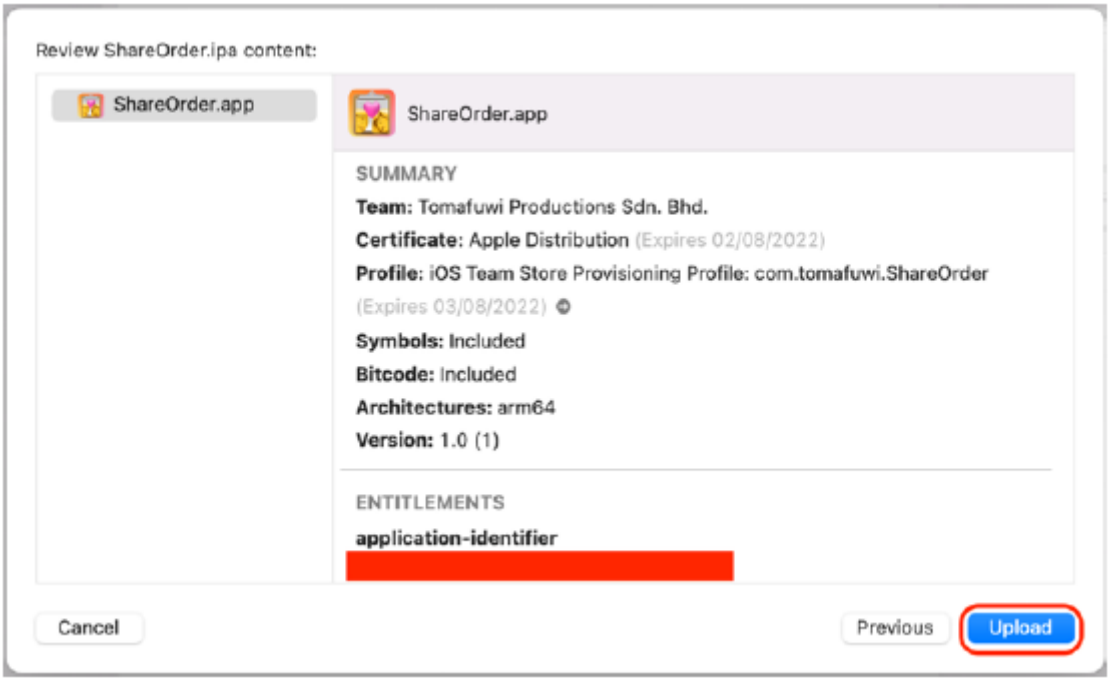


10. Wybierz opcję Automatycznie zarządzaj podpisywaniem i kliknij Dalej:



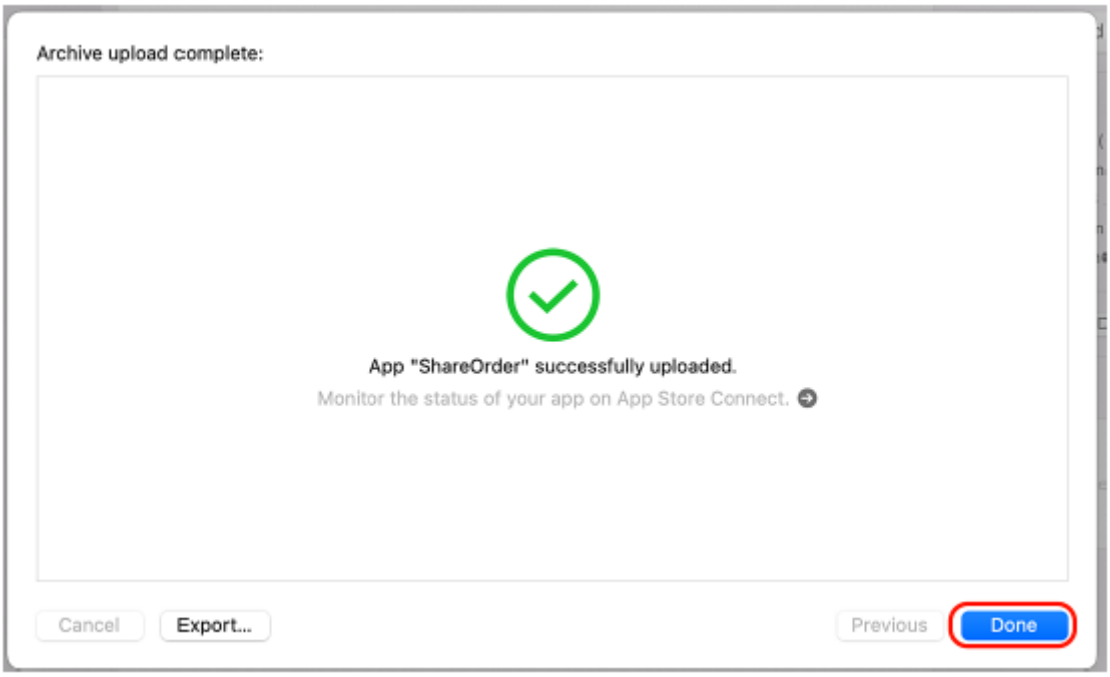
11. Jeśli zostaniesz poproszony o podanie hasła, wprowadź hasło do konta Mac i kliknij Zawsze zezwalaj.

12. Kliknij Prześlij:



13. Poczekaj na zakończenie przesyłania.

14. Kliknij Gotowe:



W tym momencie została przesłana kompilacja aplikacji, która będzie dystrybuowana przez App Store. W następnej sekcji dowiesz się, jak przesyłać zrzuty ekranu i uzupełniać informacje o swojej aplikacji

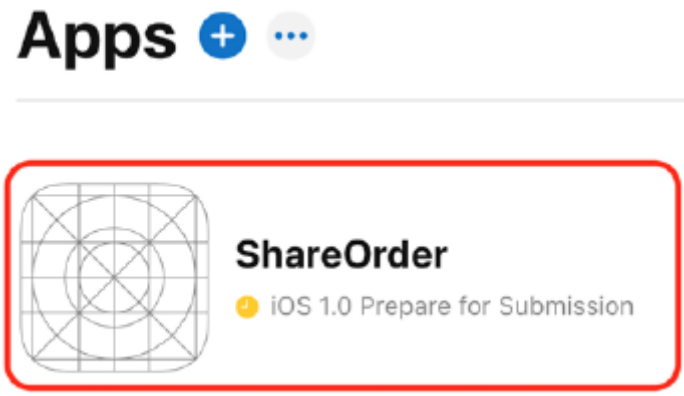
które pojawią się w App Store wraz z aplikacją.

Uzupełnianie informacji w App Store Connect

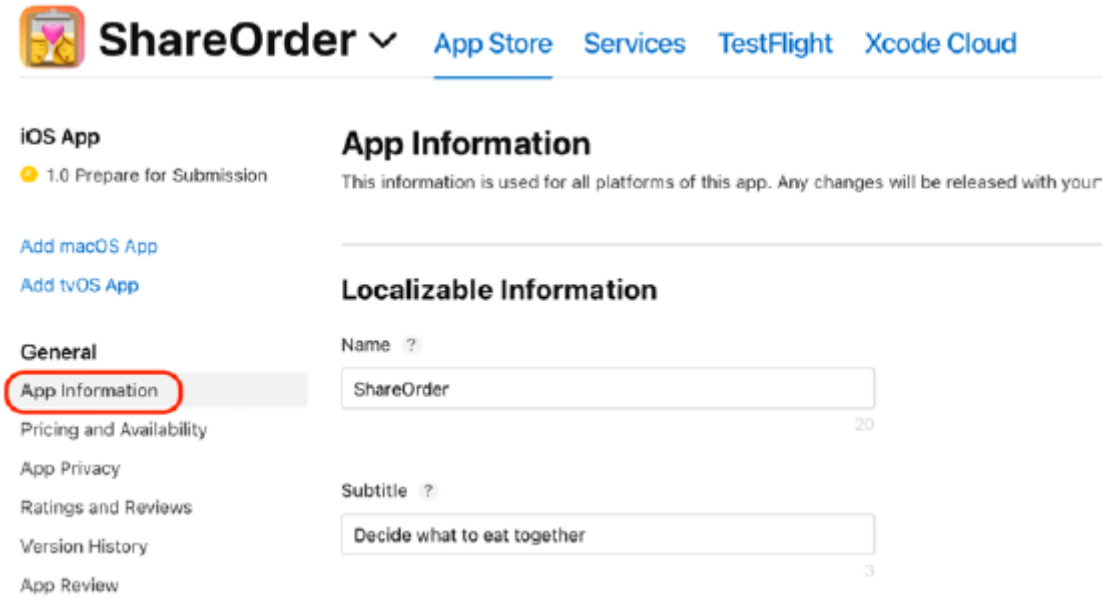
Twoja aplikacja została przesłana, ale nadal musisz podać informacje o niej w App Store Connect. Oto kroki:

1. Przejdź do http://appstoreconnect.apple.com i wybierz Moje aplikacje.

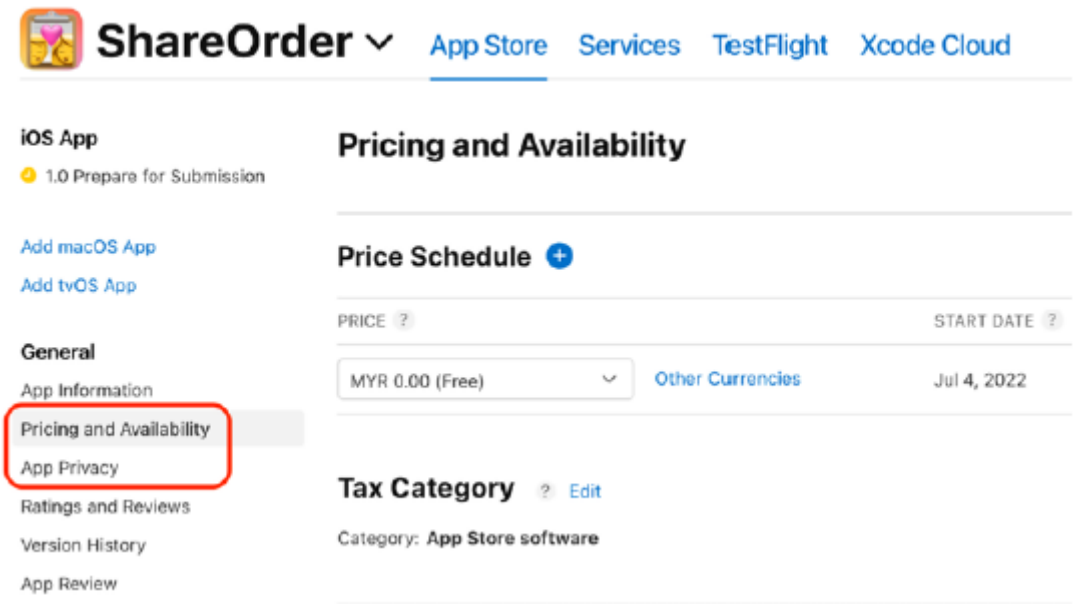
2. Wybierz właśnie utworzoną aplikację:



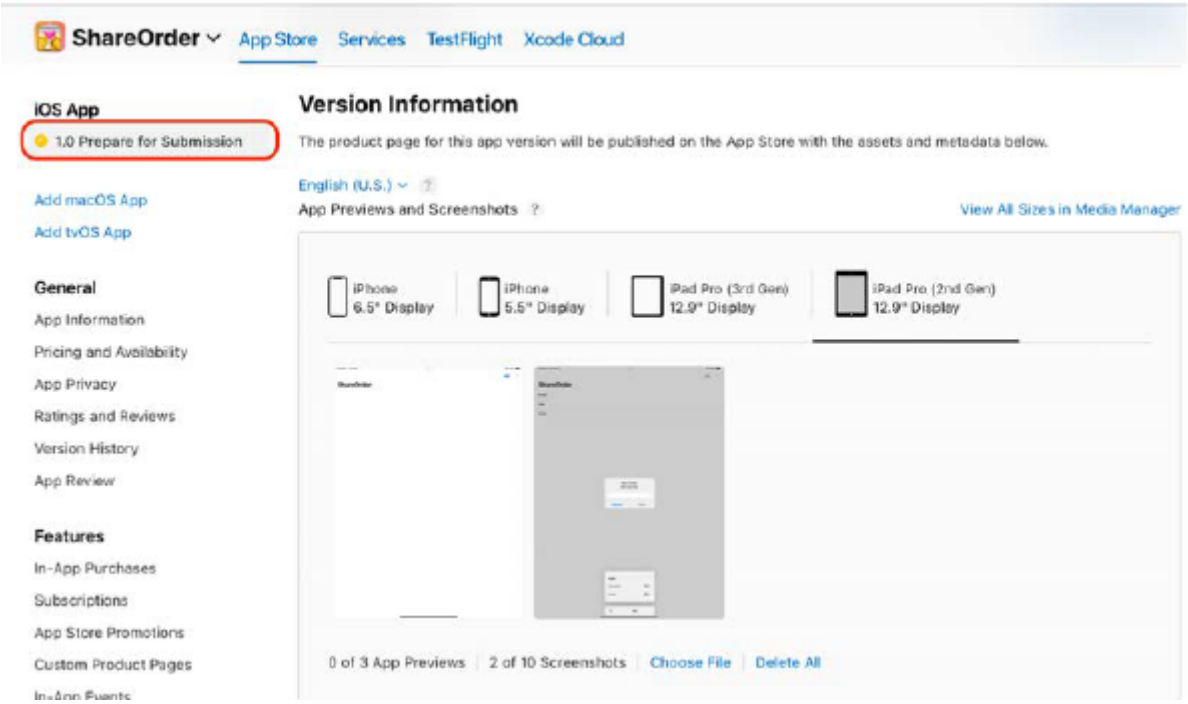
3. Wybierz Informacje o aplikacji po lewej stronie ekranu i upewnij się, że wszystkie informacje są prawidłowe:



4. Zrób to samo w sekcjach Ceny i dostępność oraz Prywatność aplikacji:



5. Wybierz opcję Przygotuj do przesłania po lewej stronie ekranu. W sekcji Podgląd aplikacji i zrzuty ekranu przeciągnij wykonane wcześniej zrzuty ekranu. Użyj zrzutów ekranu iPhone'a 13 Pro Max w sekcji Wyświetlacz iPhone'a 6,5", zrzutów ekranu iPhone'a 8 Plus w sekcji Wyświetlacz iPhone'a 5,5", a zrzutów ekranu iPada w odpowiednich sekcjach:



6. Przewiń w dół i wypełnij pola Tekst promocyjny, Opis, Słowa kluczowe, Adres URL pomocy technicznej i Adres URL marketingu:

Promotional Text ?

The easiest way to decide what everyone would like to eat.

112

Description ?

You're buying food for a group of people. Just FaceTime them and launch this app! Everyone gets to decide what they want to eat. Couldn't be simpler.

3,851

Keywords ?

#SharePlay

90

Support URL ?

https://tomafuwi.tumblr.com

Marketing URL ?

https://tomafuwi.tumblr.com

7. Przewiń w dół do sekcji Kompilacja, a zobaczysz wcześniej przesłaną kompilację archiwum. Jeśli go nie widzisz, kliknij przycisk +, wybierz kompilację i kliknij Gotowe:

Build

Upload your builds using one of several tools. See Upload Tools

Select a build before you submit your app

8. Sprawdź, czy kompilacja znajduje się w sekcji Kompilacja:

Build

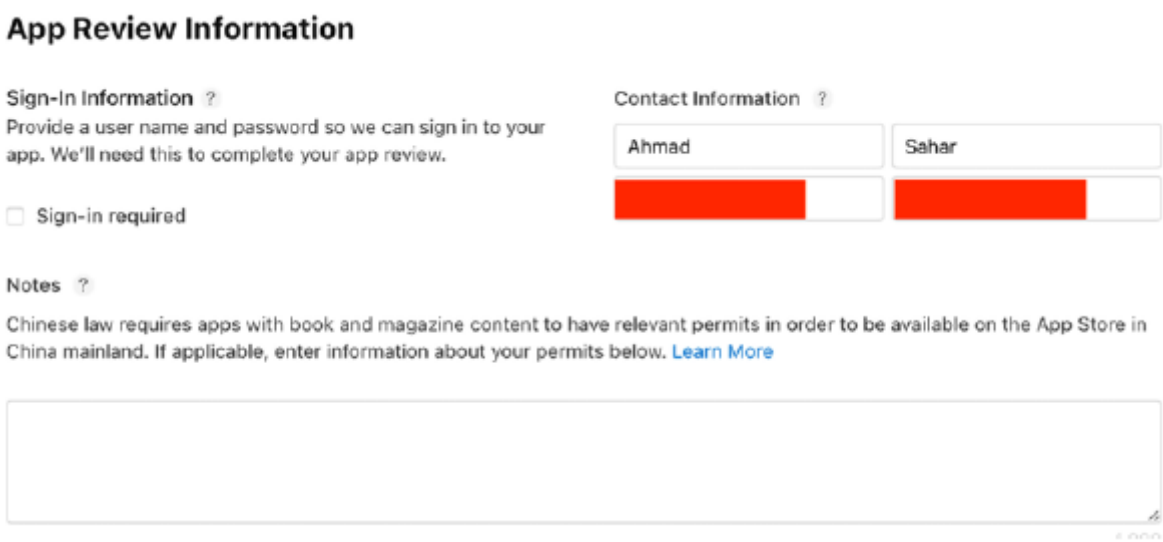
BUILD	VERSION	HAS APP CLIP
1	1.0	NO

Included Assets

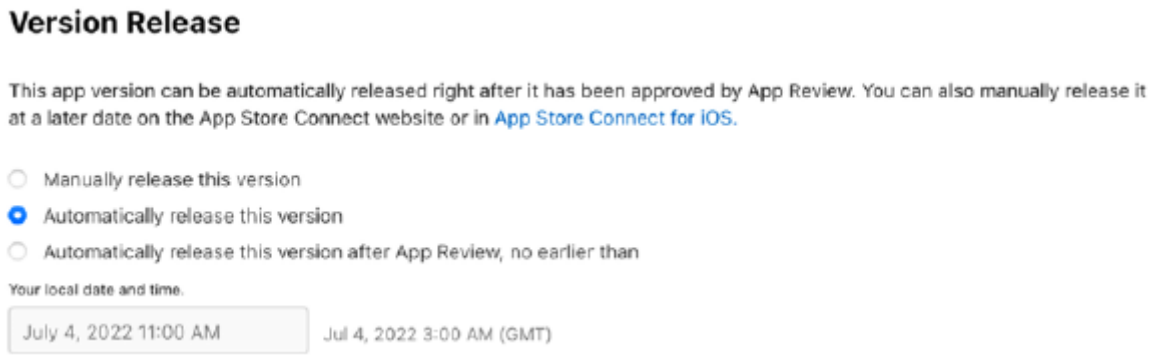
App Icon

9. Przewiń do sekcji ogólnych informacji o aplikacji i podaj wszystkie wymagane szczegóły:

10. Przewiń w dół do sekcji Informacje o recenzji aplikacji. Jeśli chcesz przekazać recenzentowi aplikacji dodatkowe informacje, umieść je tutaj:

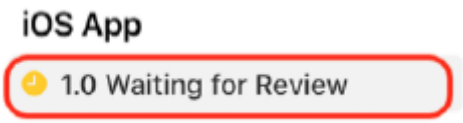


11. Przewiń w dół do sekcji Wersja wersji i zachowaj ustawienia domyślne:



12. Przewiń z powrotem do góry ekranu i kliknij przycisk Dodaj do recenzji:

13. Sprawdź, czy status aplikacji zmienił się na Oczekuje na sprawdzenie:



Będziesz musiał poczekać, aż Apple sprawdzi aplikację i otrzymasz wiadomość e-mail z informacją, czy Twoja aplikacja została zatwierdzona, czy odrzucona. Jeśli Twoja aplikacja zostanie odrzucona, pojawi się łącze prowadzące do strony Centrum rozstrzygania firmy Apple, która zawiera opis przyczyn odrzucenia Twojej aplikacji. Po rozwiązaniu problemów możesz zaktualizować archiwum i przesłać je ponownie. Teraz wiesz, jak przesłać aplikację do App Store! Wspaniały! W następnej sekcji dowiesz się, jak przeprowadzać wewnętrzne i zewnętrzne testy swojej aplikacji, co jest ważne dla zapewnienia jej wysokiej jakości i wolnej od błędów.

Testowanie aplikacji

Firma Apple udostępnia funkcję o nazwie TestFlight, która umożliwia dystrybucję aplikacji wśród testerów przed udostępnieniem ich w sklepie App Store. Musisz pobrać aplikację TestFlight, dostępną na stronie https://developer. apple.com/testflight/, aby przetestować aplikację. Twoi testerzy mogą być członkami Twojego wewnętrznego zespołu (testerzy wewnętrzni) lub ogółem społeczeństwa (testerzy zewnętrzni). Zobaczmy, jak zezwolić członkom wewnętrznego zespołu na pierwsze testowanie aplikacji w następnej sekcji.

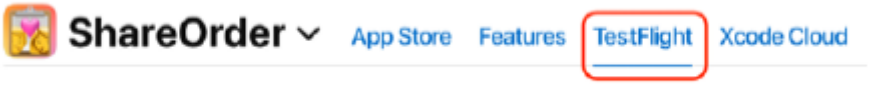
Wewnętrzne testowanie aplikacji

Testowanie wewnętrzne jest dobre, gdy aplikacja jest na wczesnym etapie rozwoju. Dotyczy to wyłącznie członków Twojego wewnętrznego zespołu. Apple nie recenzuje aplikacji dla wewnętrznych testerów. Kompilacje możesz wysłać do maksymalnie 100 testerów w celu przeprowadzenia testów wewnętrznych. Wykonaj następujące kroki:

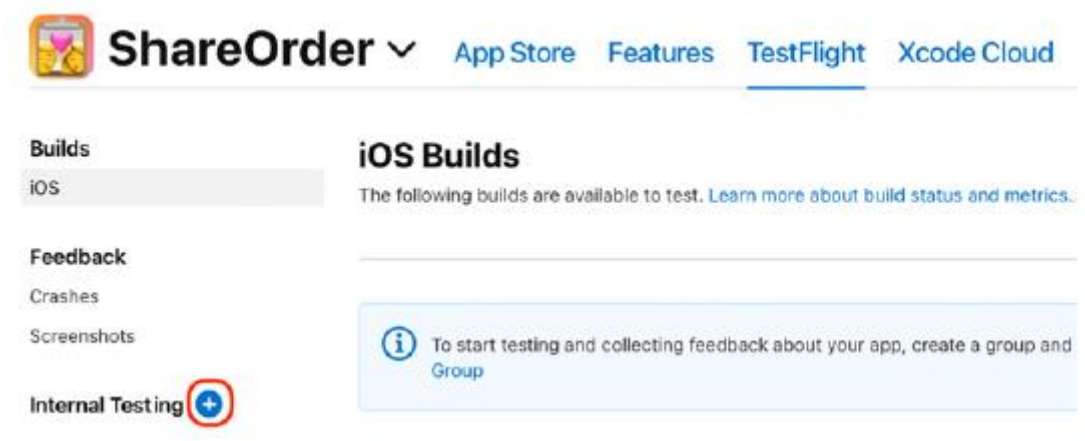
1. Przejdź do http://appstoreconnect.apple.com i wybierz Moje aplikacje.

2. Wybierz aplikację, którą chcesz przetestować.

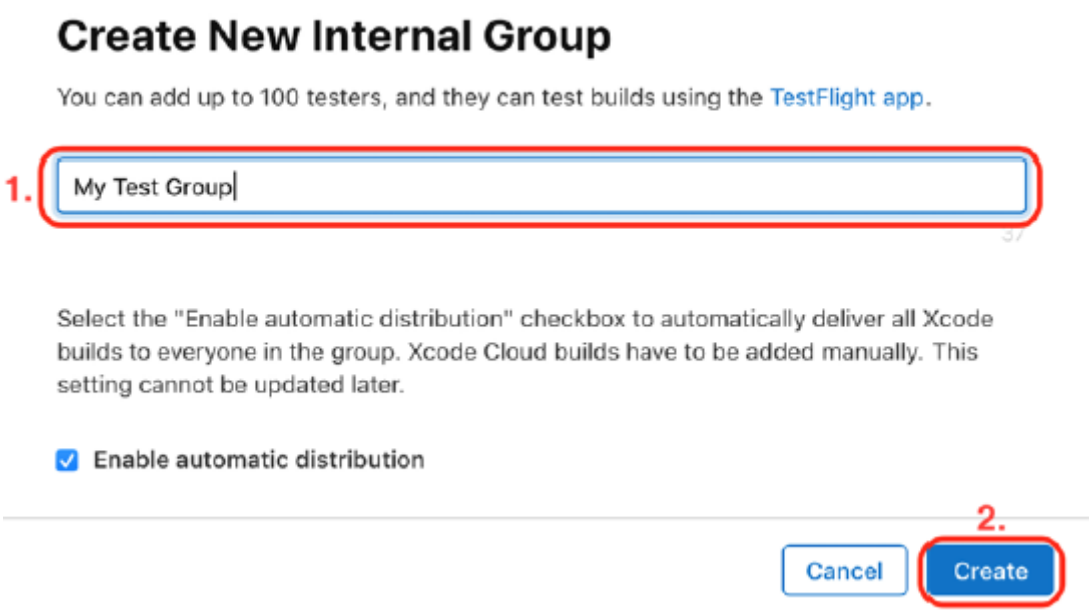
3. Kliknij zakładkę TestFlight:



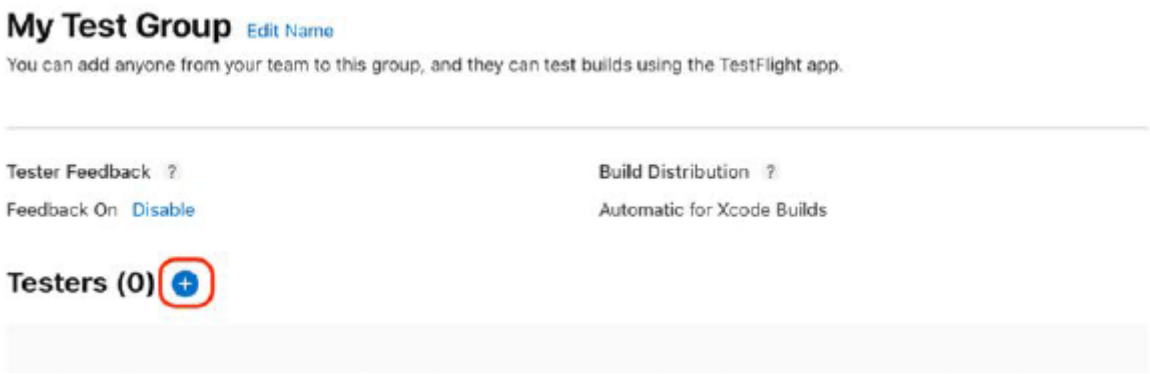
4. Kliknij przycisk + obok opcji Testowanie wewnętrzne, aby utworzyć nową grupę testów wewnętrznych:



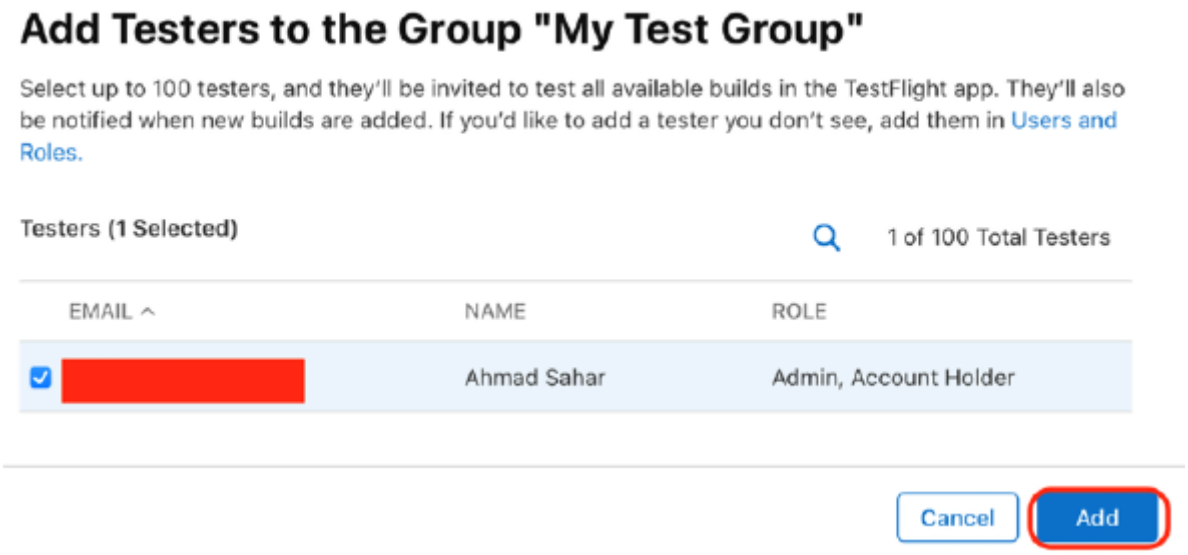
5. Pojawi się okno dialogowe Utwórz nową grupę wewnętrzną. Nazwij swoją wewnętrzną grupę testową i kliknij Utwórz:



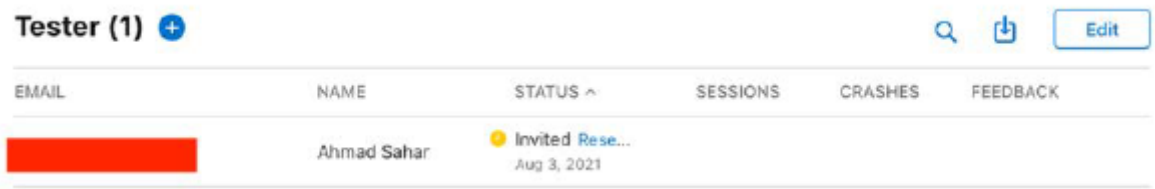
6. Po utworzeniu grupy testowej kliknij przycisk +, aby dodać użytkowników do swojej grupy:



7. Zaznacz wszystkich użytkowników, do których chcesz wysłać kompilacje testowe i kliknij Dodaj. Zostaną zaproszeni do przetestowania wszystkich dostępnych kompilacji:



8. Sprawdź, czy Twoi testerzy zostali dodani:



W testach wewnętrznych biorą udział wyłącznie członkowie Twojego zespołu. Jeśli chcesz przeprowadzić testy z dużą liczbą testerów, będziesz musiał przeprowadzić testy zewnętrzne, co opisano w następnej sekcji.

Testowanie aplikacji na zewnątrz

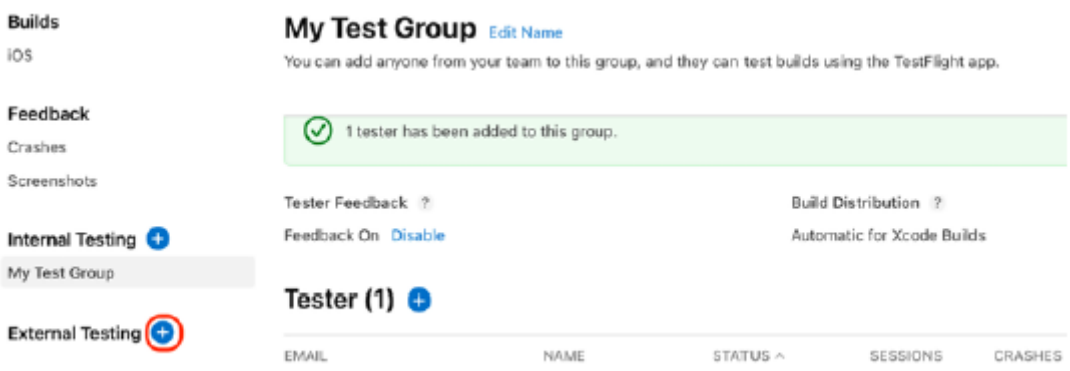
Testowanie zewnętrzne jest dobre, gdy aplikacja jest w końcowej fazie rozwoju. Możesz wybrać dowolną osobę na zewnętrznego testera i wysłać kompilacje nawet do 10 000 testerów. Apple może przeglądać aplikacje dla zewnętrznych testerów. Oto kroki:

1. Przejdź do http://appstoreconnect.apple.com i wybierz Moje aplikacje.

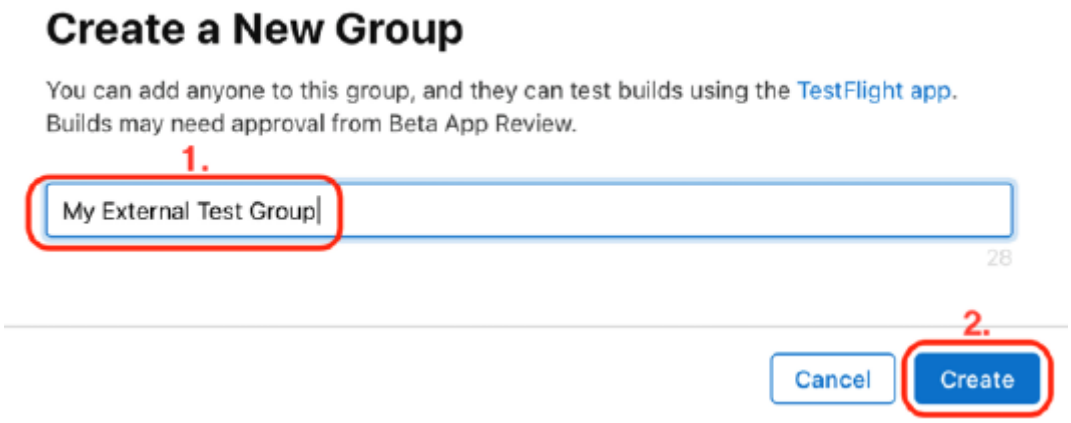
2. Wybierz aplikację, którą chcesz przetestować.

3. Kliknij zakładkę TestFlight.

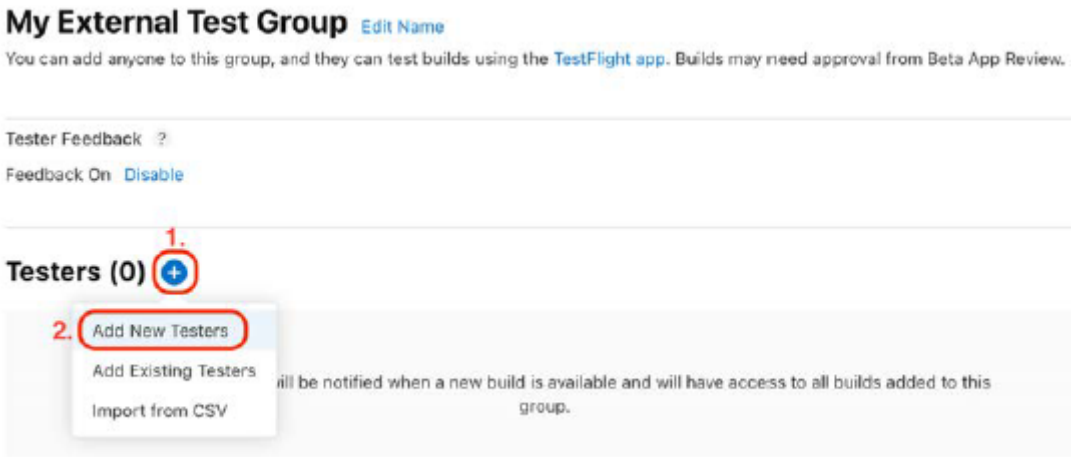
4. Kliknij przycisk + obok Testowanie zewnętrzne:



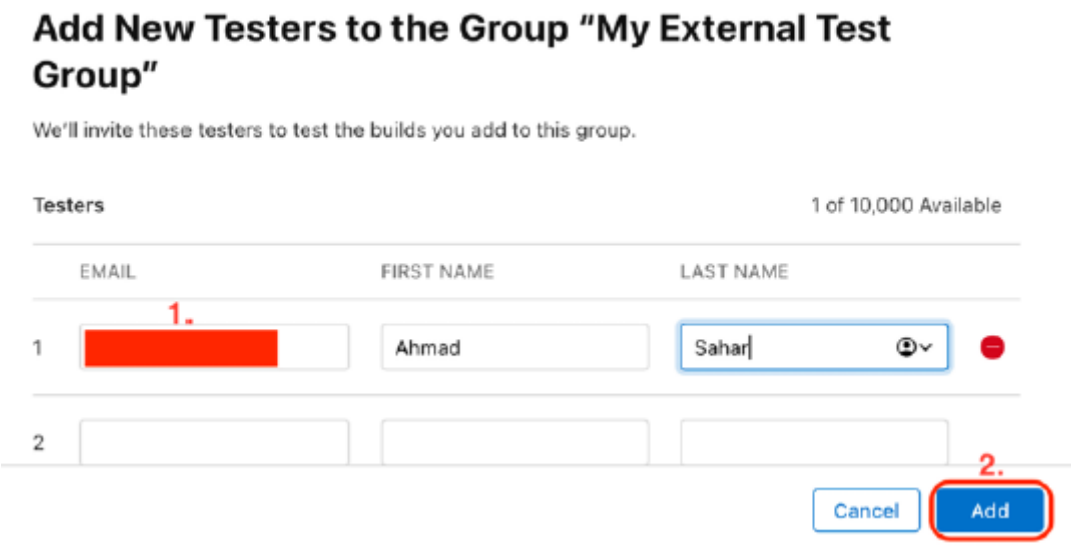
5. Wpisz nazwę grupy testowej i kliknij Utwórz:



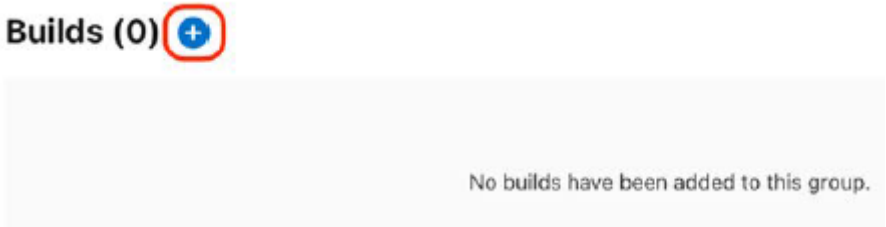
6. Kliknij przycisk + obok Testerów i wybierz Dodaj nowych testerów:



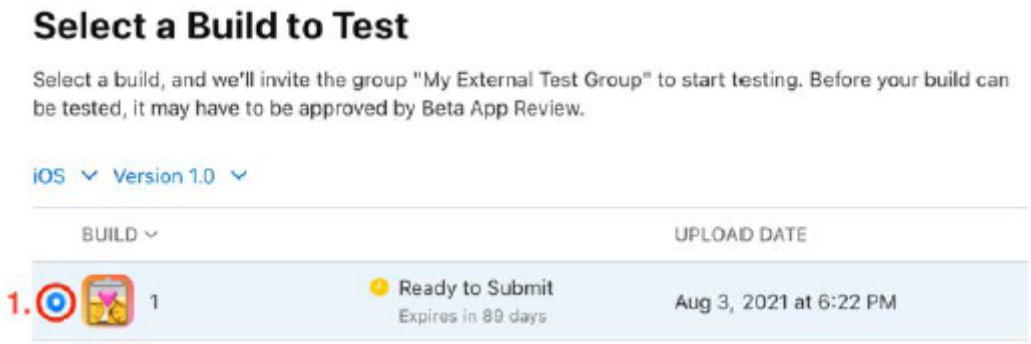
7. Wpisz nazwiska i adresy e-mail swoich testerów, a Apple powiadomi ich automatycznie, gdy kompilacja będzie gotowa do testowania:



8. W sekcji Kompilacje kliknij +:

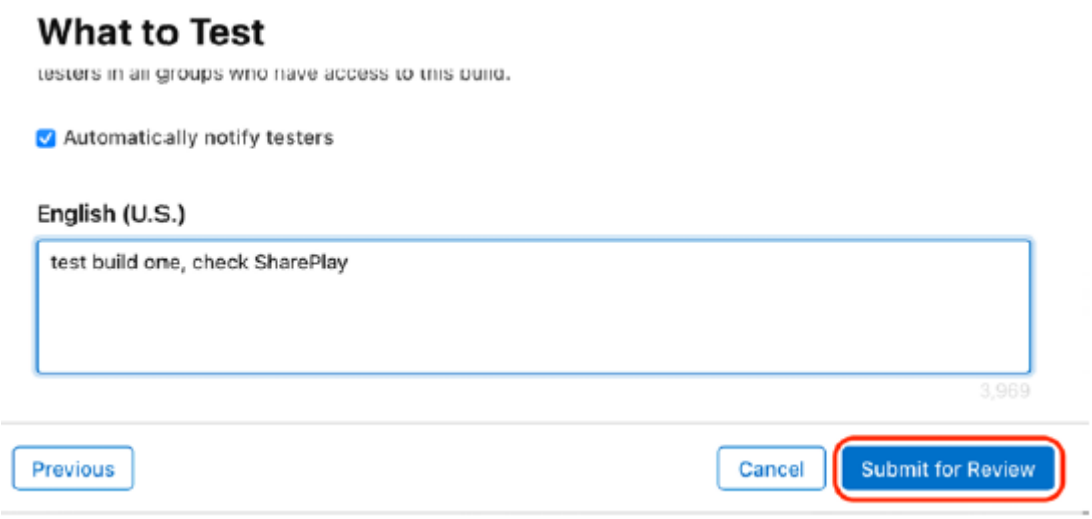


9. Wybierz jedną ze swoich kompilacji i kliknij Dalej:



10. Wprowadź informacje o teście i kliknij Dalej:

11. Wpisz Co testować i kliknij Prześlij do recenzji:



Podobnie jak w przypadku przesyłania aplikacji do sklepu App Store, musisz poczekać, aż firma Apple sprawdzi aplikację, a jeśli aplikacja zostanie odrzucona, na stronie Centrum rozstrzygania firmy Apple będą znajdować się szczegółowe informacje o przyczynach odrzucenia aplikacji. Następnie możesz rozwiązać problemy i przesłać ponownie. Świetnie! Wiesz już, jak testować aplikacje wewnętrznie i zewnętrznie, i dotarłeś do końca tej książki!

Streszczenie

Zakończyłeś cały proces tworzenia aplikacji i przesyłania jej do App Store. Gratulacje! Zacząłeś od nauczenia się, jak uzyskać konto programisty Apple. Następnie nauczyłeś się, jak wygenerować żądanie podpisania certyfikatu, aby utworzyć certyfikaty umożliwiające testowanie aplikacji na własnych urządzeniach i publikowanie ich w App Store. Dowiedziałeś się, jak utworzyć identyfikator pakietu, aby jednoznacznie identyfikować aplikację w App Store, i zarejestrować urządzenia testowe. Następnie nauczyłeś się tworzyć profile udostępniania programistycznego i produkcyjnego, aby umożliwić uruchamianie aplikacji na urządzeniach testowych i przesyłanie ich do App Store. Następnie nauczyłeś się, jak utworzyć listę App Store i przesłać kompilację wersji do App Store. Wreszcie nauczyłeś się, jak przeprowadzać testy swojej aplikacji przy użyciu testerów wewnętrznych i zewnętrznych. Teraz wiesz,

jak tworzyć i przesyłać aplikacje do App Store oraz przeprowadzać działania wewnętrzne i zewnętrzne testowanie Twojej aplikacji. Gdy aplikacja zostanie przesłana do sprawdzenia, jedyne, co możesz zrobić, to poczekać, aż Apple sprawdzi Twoją aplikację. Nie martw się, jeśli aplikacja zostanie odrzucona — zdarza się to wszystkim programistom. Współpracuj z firmą Apple, aby rozwiązać problemy za pośrednictwem Centrum rozstrzygania i przeprowadź badania, aby dowiedzieć się, co jest, a co nie, akceptowane przez firmę Apple. Gdy Twoje aplikacje znajdą się w App Store, skontaktuj się ze mną (@shah_apple) i Craigiem Claytonem (@thedevme) na Twitterze, aby nas o tym powiadomić — chętnie zobaczymy, co stworzyłeś.