Pierwsze kroki z SwiftUI

W poprzednich Częściach utworzyłeś interfejs użytkownika (UI) dla aplikacji Let's Eat za pomocą scenorysów. Proces obejmował przeciąganie obiektów reprezentujących widoki do scenorysu, tworzenie gniazd w plikach kontrolera widoku i łączenie ich ze sobą Tu 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 zachować synchronizację kodu i projektu. Funkcje takie jak typ dynamiczny, tryb ciemny, lokalizacja i dostępność są obsługiwane automatycznie. 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 bardzo 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 listy restauracji. Następnie dodasz obiekty modelu do swojej aplikacji i skonfigurujesz nawigację między ekranami Lista restauracji i Szczegóły restauracji. Następnie dowiesz się, jak używać widoków UIKit i SwiftUI razem, dodając i konfigurując widok mapy dla ekranu szczegółów restauracji. Na koniec utworzysz ekran szczegółów restauracji. Pod koniec dowiesz się, jak zbudować aplikację SwiftUI, która odczytuje obiekty modelu, prezentuje je na liście i umożliwia nawigację do drugiego ekranu zawierającego widok mapy. Możesz to następnie wdrożyć do własnych projektów. Omówione zostaną następujące tematy:

• Tworzenie projektu SwiftUI Xcode

• Tworzenie ekranu listy restauracji

• Dodawanie obiektów modelu i konfigurowanie nawigacji

• Wspólne korzystanie z widoków UIKit i SwiftUI

• Tworzenie ekranu szczegółów restauracji

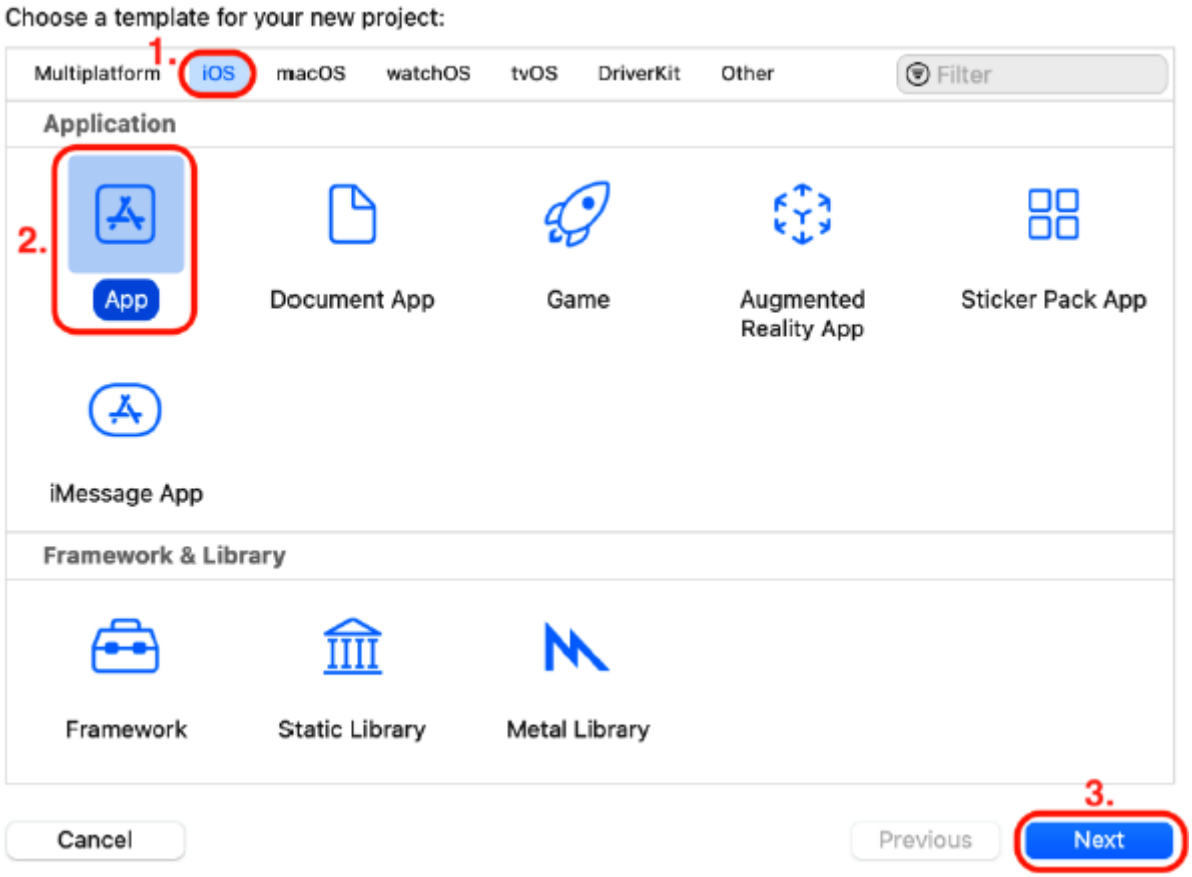
Tworzenie projektu SwiftUI Xcode

Projekt SwiftUI Xcode jest tworzony w taki sam sposób, jak zwykły projekt Xcode, ale konfigurujesz go tak, aby używał SwiftUI zamiast scenorysów. Jak zobaczysz, interfejs użytkownika jest generowany w całości w kodzie i będziesz mógł zobaczyć zmiany w interfejsie użytkownika natychmiast po zmodyfikowaniu kodu.

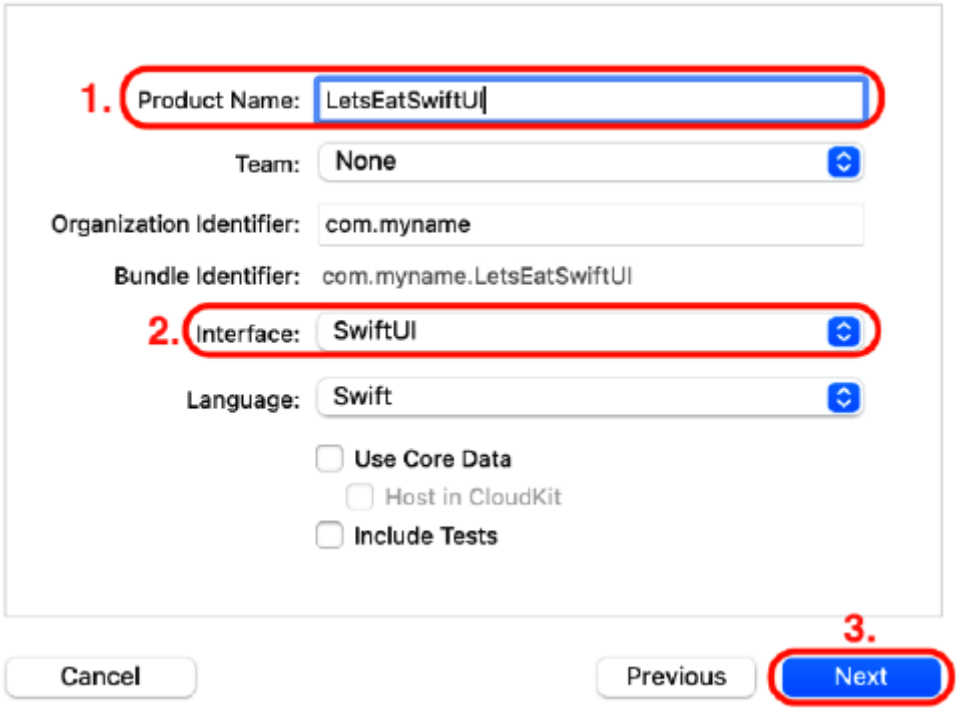
Zacznijmy od stworzenia nowego projektu SwiftUI Xcode. Wykonaj następujące kroki:

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

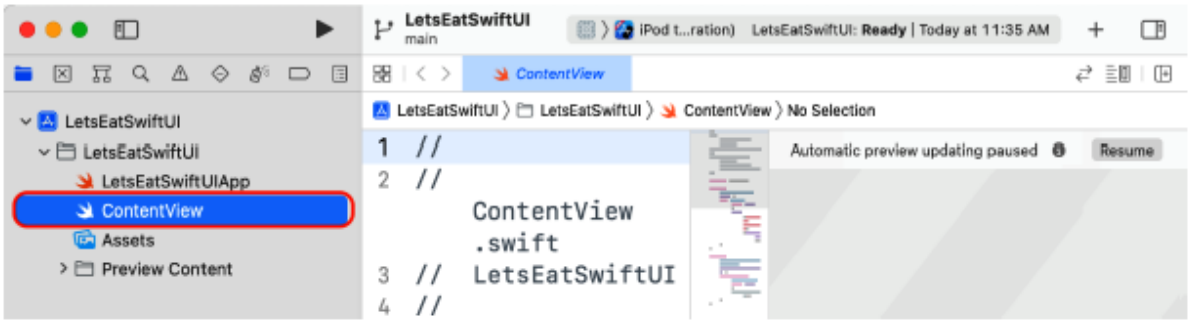
• Interfejs: SwiftUI

Pozostałe ustawienia powinny być już ustawione. Upewnij się, że wszystkie

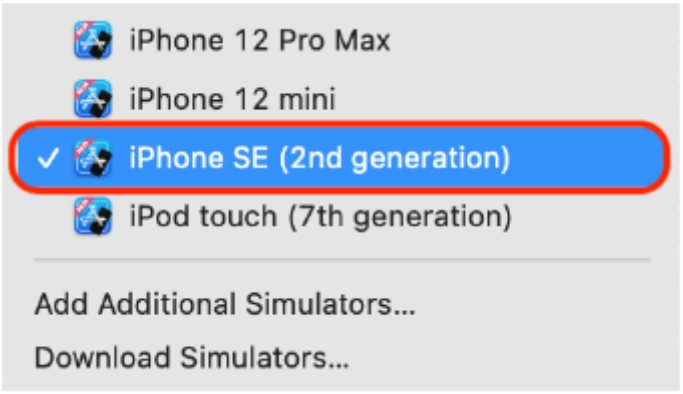
pola wyboru są odznaczone. Po zakończeniu kliknij Dalej.

4. Wybierz lokalizację do zapisania projektu LetsEatSwiftUI i kliknij Utwórz.

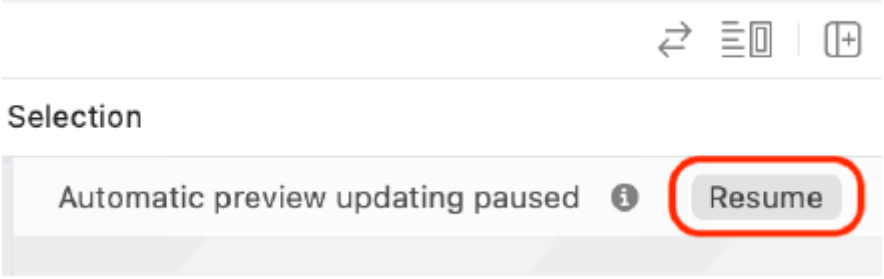
5. Twój projekt pojawi się na ekranie z plikiem ContentView wybranym w nawigatorze projektów. Zobaczysz zawartość tego pliku po lewej stronie obszaru edytora, a obszar roboczy z podglądem po prawej stronie:



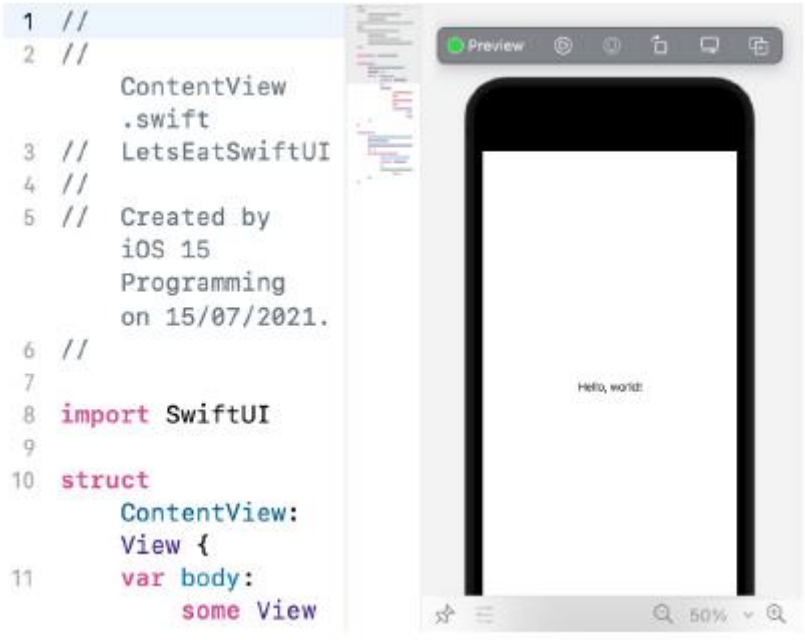
6. Plik ContentView zawiera kod, który wygeneruje początkowy widok Twojej aplikacji. Kliknij menu Schemat i wybierz iPhone SE (2. generacji), aby podgląd widoku był wyświetlany na ekranie iPhone SE (2. generacji):



7. Kliknij przycisk Wznów na płótnie, aby wygenerować podgląd:

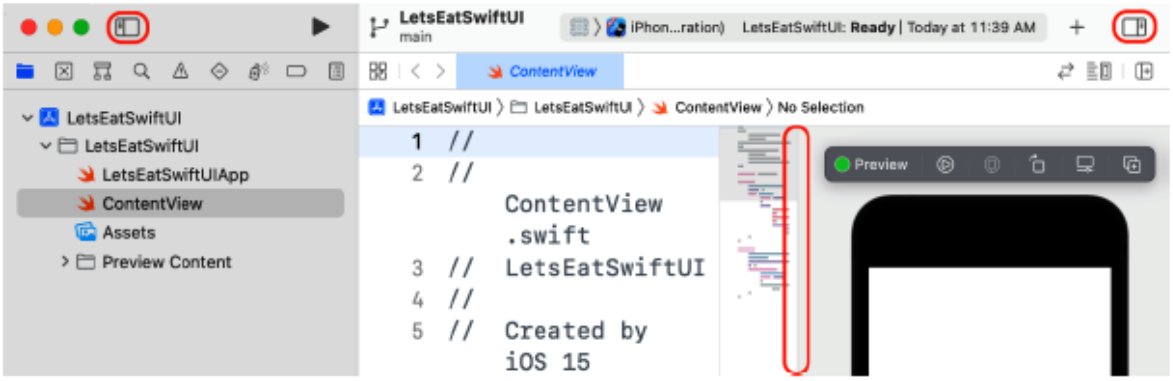


8. Sprawdź, czy podgląd Twojej aplikacji jest wyświetlany na kanwie:



Jeśli płótno nie jest widoczne, wybierz opcję Płótno z menu Dostosuj opcje edytora, aby je wyświetlić. Jeśli używasz MacBooka, możesz użyć gestu uszczypnięcia na gładziku, aby zmienić rozmiar symulowanego obrazu.

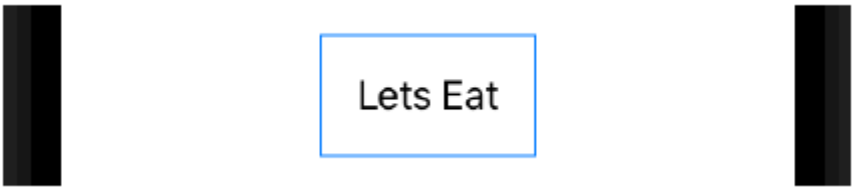
9. Jeśli potrzebujesz więcej miejsca do pracy, kliknij przyciski Nawigator i Edytor, aby ukryć obszary Nawigator i Edytor, a następnie przeciągnij ramkę w obszarze Edytor, aby zmienić rozmiar płótna:



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 jest wyświetlany na kanwie. Aby zobaczyć to w akcji, zmień Hello, World! tekst do Lets Eat, jak pokazano:

```
struct ContentView: View {

var body: some View {

Text("Lets Eat").padding()

}

}
```

Podgląd w kanwie zostanie zaktualizowany, aby odzwierciedlić zmiany:



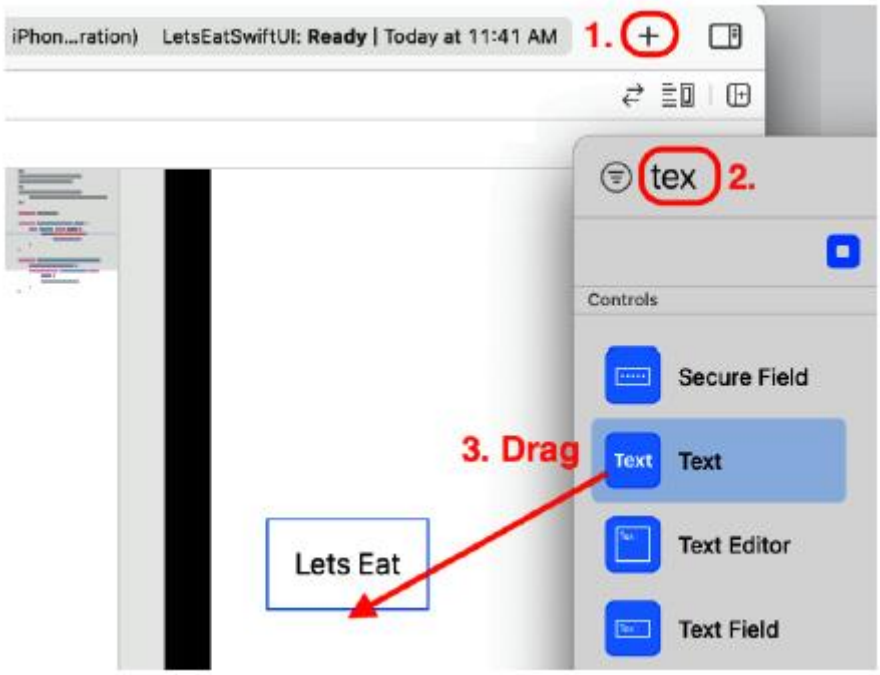
Udało Ci się stworzyć swój pierwszy projekt SwiftUI! Teraz stwórzmy ekran Lista restauracji, zaczynając od widoku, który będzie wyświetlał dane konkretnej restauracji.

Tworzenie ekranu listy restauracji

Używając scenorysów, modyfikujesz atrybuty widoku za pomocą Inspektora atrybutów. W SwiftUI możesz modyfikować swój kod lub podgląd na kanwie. Jak widać, zmiana kodu w pliku ContentView natychmiast zaktualizuje podgląd, a modyfikacja podglądu zaktualizuje kod.

Dostosujmy strukturę ContentView do wyświetlania danych konkretnej restauracji. Wykonaj następujące kroki:

1. Kliknij przycisk Biblioteka. Wpisz tekst w polu filtru i przeciągnij widok Tekst na kanwę i upuść go pod tekstem Lets Eat:



2. Xcode automatycznie dodał kod do pliku ContentView dla tego widoku tekstu. Sprawdź, czy Twój kod wygląda tak:

```
struct ContentView: View {

var body: some View {

VStack {

Text("Lets Eat").padding()
```

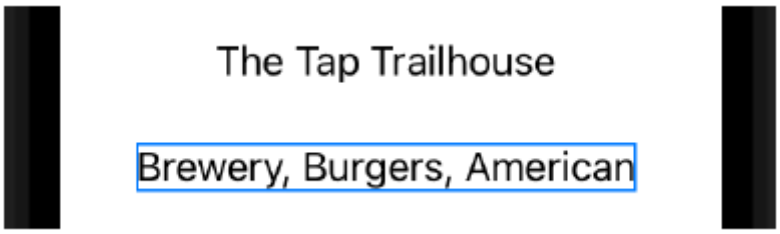
```
Text("Placeholder")

}

}

}
```

Jak widać, drugi widok tekstu został dodany po widoku tekstu zawierającym ciąg „Lets Eat”, a oba widoki tekstu są zawarte w widoku VStack. Widok VStack zawiera podwidoki ułożone pionowo i jest podobny do pionowego widoku stosu.

3. Jako przykładowe dane wykorzystasz dane dotyczące The Tap Trailhouse, restauracji w Bostonie. Zmodyfikuj widoki tekstu w widoku VStack, aby pokazać nazwę i kuchnie oferowane przez restaurację The Tap Trailhouse:

```
struct ContentView: View {

var body: some View {

VStack {

Text("The Tap Trailhouse").padding()

Text("Brewery, Burgers, American")

}

}

}
```

4. Sprawdź, czy zmiany są odzwierciedlone w podglądzie:



5. Użyjesz widoku obrazu SwiftUI, aby wyświetlić zdjęcie restauracji. Zmodyfikuj kod, jak pokazano, aby dodać widok obrazu do widoku VStack:

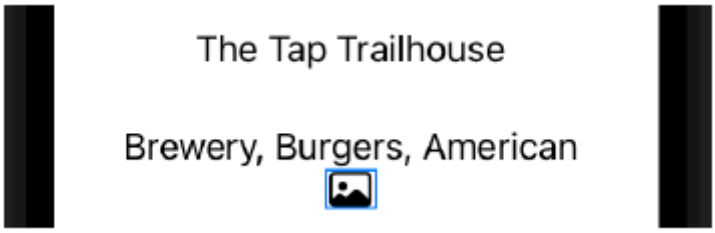
```
struct ContentView: View {

var body: some View {

VStack {

Text("The Tap Trailhouse").padding()

Text("Brewery, Burgers, American")

Image(systemName: "photo")

}
```

}

}

Zauważ, że widok obrazu ma jeden parametr, systemName. Ten parametr pozwala wybrać jeden z obrazów w bibliotece symboli SF firmy Apple. Później zastąpisz ten obraz symboli SF zdjęciem.

6. Sprawdź, czy płótno wyświetla teraz dwa widoki tekstu i jeden widok obrazu, jak pokazano:



7. Aby zmienić wygląd tekstu, użyj modyfikatorów zamiast Inspektora atrybutów. Są to metody, które zmieniają wygląd lub zachowanie obiektów. Zaktualizuj kod, jak pokazano, aby ustawić styl i kolor widoków tekstu:

struct ContentView: View {

var body: some View {

VStack {

Text("The Tap Trailhouse")

.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "photo")

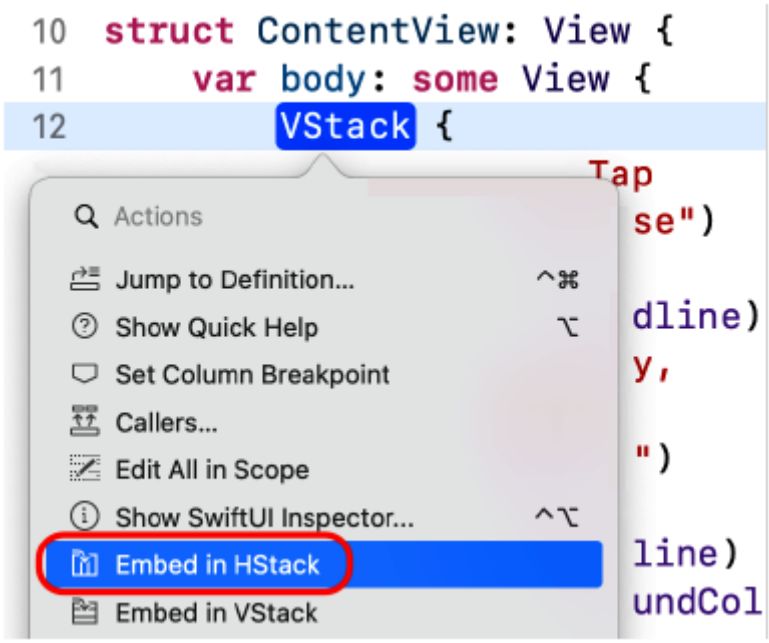
}

}

}

Zwróć uwagę na zmiany w tekście w podglądzie.

8. Aby upewnić się, że widok pozostaje 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 rozszerza się w poziomie w widoku HStack. Command + kliknij widok VStack i wybierz Osadź w HStack z menu podręcznego:



9. Sprawdź, czy Twój kod wygląda tak:

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: "photo")

}

}

}

}

10. Dodaj dwa obiekty Spacer do widoku HStack, jak pokazano, aby wyśrodkować widok poziomo na ekranie:

HStack {

Spacer()

VStack {

Text("The Tap Trailhouse")

.font(.headline)

Text("Brewery, Burgers, American")

.font(.subheadline)

.foregroundColor(.secondary)

Image(systemName: "photo")

}

Spacer()

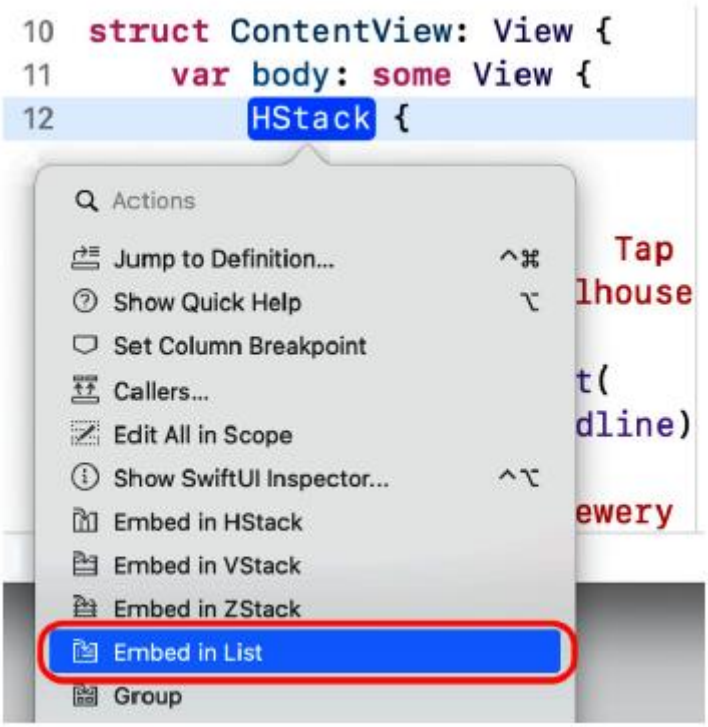
}

Twój widok jest teraz kompletny. Użyjesz tego widoku jako komórki na ekranie Lista restauracji w następnej sekcji.

Dodawanie obiektów modelu i konfigurowanie nawigacji

Masz teraz widok, którego można użyć do wyświetlenia szczegółów restauracji. Użyjesz tego widoku jako komórki na liście SwiftUI, która jest kontenerem prezentującym dane w jednej kolumnie. Skonfigurujesz również obiekty modelu, aby wypełnić tę listę. Wykonaj następujące kroki:.

1. Command + kliknij widok HStack i wybierz Osadź na liście, aby wyświetlić listę zawierającą pięć komórek na kanwie:



2. Sprawdź, czy Twój kod wygląda tak:

struct ContentView: View {

var body: some View {

```
List(0 ..<5) { item in
Spacer()
VStack {
Text("The Tap Trailhouse")
.font(.headline)
Text("Brewery, Burgers, American")
.font(.subheadline)
.foregroundColor(.secondary)
Image(systemName: "photo")
}
Spacer()
}
}
}
```

Jak widać, widok utworzony w poprzedniej sekcji jest teraz zawarty w liście skonfigurowanej do wyświetlania pięciu elementów, a widok HStack nie jest już potrzebny. Należy zauważyć, że do wyświetlania danych na liście nie są wymagane żadne pełnomocniki ani źródła danych.

3. Otwórz folder zasobów zawarty w folderze Chapter23 pakietu kodu pobranego z https://github.com/PacktPublishing/iOS-15-Programming-for-Beginners-Sixth-Edition. Przeciągnij plik RestaurantItem.swift do nawigatora projektów i kliknij przycisk Zakończ, gdy pojawi się monit o dodanie ich do projektu.

4. Kliknij plik RestaurantItem w nawigatorze projektów i powinieneś zobaczyć w nim następujący kod:

```
import Foundation
import MapKit
struct RestaurantItem: Identifiable {
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, której użyłeś w projekcie LetsEat. Aby użyć tej struktury na liście, musisz dostosować ją do protokołu Identyfikowalnego. Ten protokół określa, że element listy musi mieć właściwość id, która może identyfikować określony element. Instancja UUID jest przypisywana do każdej instancji RestaurantItem podczas tworzenia, aby zapewnić, że każdy identyfikator jest unikalny. testData to tablica zawierająca pięć wystąpień RestaurantItem reprezentujących pięć restauracji w rejonie Bostonu. Pełni tę samą funkcję, co pliki JSON, których już używałeś.

5. Kliknij plik ContentView w nawigatorze projektów. Dodaj właściwość restaurantItems do widoku, aby przechowywać dane dla listy po otwierającym nawiasie klamrowym struktury ContentView:

struct ContentView: View {

var restaurantItems: [RestaurantItem] = []

var body: some View {

6. Zmodyfikuj kod, jak pokazano, 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

Spacer()

VStack {

Text(restaurantItem.title)

.font(.headline)

Text(restaurantItem.subtitle)

.font(.subheadline)

.foregroundColor(.secondary)

AsyncImage(url: URL(string:

restaurantItem.imageURLString))

.mask(RoundedRectangle

```
(cornerRadius: 9))

}

Spacer()

}

}

}

struct ContentView_Previews: PreviewProvider {

static var previews: some View {

ContentView(restaurantItems: testData)

}

}
```

Zobaczmy, jak to działa.

Struktura ContentView przechowuje tablicę wystąpień RestaurantItem we właściwości restaurantItems. Ta tablica jest przekazywana do listy. Dla każdego elementu w tablicy restaurantItems tworzony jest widok i przypisywany z danymi 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 systemie iOS 15. Ponieważ w tablicy jest pięć elementów, na kanwie pojawia się pięć widoków. Struktura ContentView_Previews przechodzi w tablicy testData (przechowywanej w pliku RestaurantItem) do struktury ContentView, która jest następnie używana do wypełnienia widoku.

7. Gdy dokonasz większych zmian w kodzie, automatyczna aktualizacja płótna zostanie wstrzymana. Kliknij przycisk Wznów, aby wznowić, jeśli to konieczne. Zwróć uwagę, że rozmiar komórki został zmieniony, aby dopasować go do rozmiaru obrazu restauracji. Następnie zaimplementujesz nawigację, aby po dotknięciu komórki pojawił się drugi ekran, który pokaże szczegóły konkretnej restauracji. Wykonaj następujące kroki:

1. Zmodyfikuj swój kod, jak pokazano, aby zawinąć listę w widoku nawigacji:

```
var body: some View {

NavigationView {

List(restaurantItems) { restaurantItem in

Spacer()

VStack {

Text(restaurantItem.title)

.font(.headline)

Text(restaurantItem.subtitle)

.font(.subheadline)
```

```
.foregroundColor(.secondary)
AsyncImage(url: URL(string:
restaurantItem.imageURLString)
.mask(RoundedRectangle
(cornerRadius: 9))
}
Spacer()
}
}
}
```

Porada : aby ponownie wciąć kod, zaznacz cały kod, wpisując Command + A, a następnie Control + I.

Widok nawigacji jest podobny do klasy UINavigation używanej wcześniej w aplikacji.

2. Dodaj modyfikator, aby ustawić właściwość title listy tak, aby wyświetlała Boston, MA u góry ekranu:

```
.mask(RoundedRectangle(cornerRadius: 9))
}
Spacer()
}.navigationTitle("Boston, MA")
```

3. Zawiń komórkę w widoku łącza nawigacyjnego, jak pokazano:

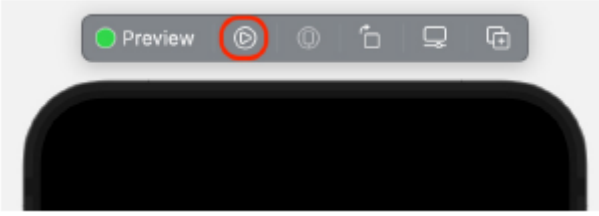
```
List(restaurantItems) { restaurantItem in
NavigationLink(destination:
Text(restaurantItem.title)) {
Spacer()
VStack {
Text(restaurantItem.title)
.font(.headline)
.fixedSize()
Text(restaurantItem.subtitle)
.font(.subheadline)
.foregroundColor(.secondary)
.fixedSize()
AsyncImage(url: URL(string:
```

```
restaurantItem.imageURLString)
.mask(RoundedRectangle
(cornerRadius: 9))
}
Spacer()
}
}.navigationTitle("Boston, MA"
```

Widok łącza nawigacyjnego ma właściwość docelową, która określa widok, który ma być prezentowany po dotknięciu komórki. Obecnie określony widok jest widokiem tekstowym pokazującym nazwę restauracji. Modyfikator .fixedSize() służy do zapewnienia, że tekst nie zostanie obcięty.

4. Zwróć uwagę, że lista na kanwie automatycznie wyświetla strzałki ujawniające:

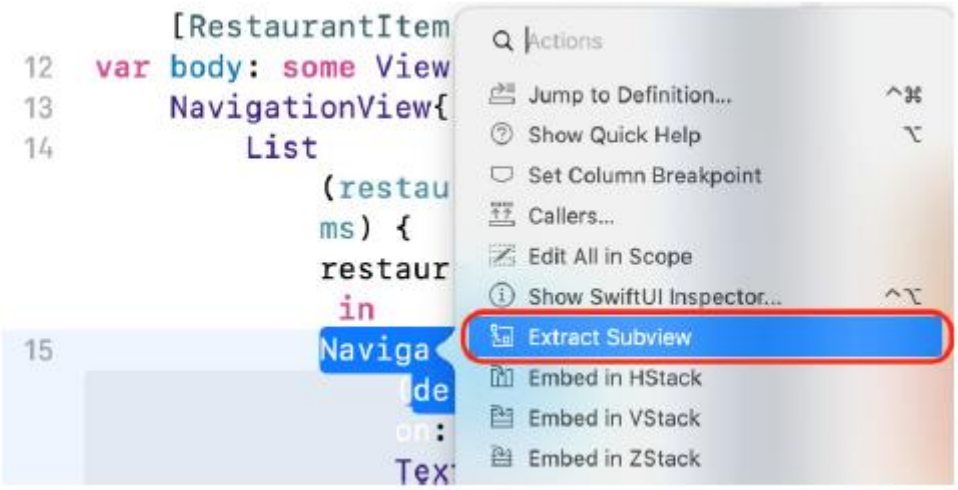
5. Aby zobaczyć, jak działa to tak, jak powinno w aplikacji, kliknij przycisk Podgląd na żywo na płótnie:



6. Kliknij dowolną komórkę w podglądzie, aby wyświetlić tekst zawierający nazwę wybranej restauracji:

To świetny sposób na zapewnienie, że Twoja lista działa zgodnie z oczekiwaniami.

7. Kod widoku zaczyna wyglądać na zaśmiecony, więc wyodrębnisz komórkę do osobnego widoku. Command + kliknij widok NavigationLink i wybierz Wyodrębnij widok podrzędny:



8. Cały kod widoku komórki został przeniesiony do osobnego widoku o nazwie ExtractedView:

```
12      var body: some View {
13          NavigationView{
14              List(restaurantItems) { restaurantItem in
15                  ExtractedView()
16              }.navigationTitle("Boston, MA")
17          }
18      }
19  }
20
21  struct ContentView_Previews: PreviewProvider {
22      static var previews: some View {
23          ContentView(restaurantItems: testData)
24      }
25  }
26
27  struct ExtractedView: View {
28      var body: some View {
29          NavigationLink(destination: Text(restaurantItem.title)){
```

9. Zmień nazwę wywołania metody i wyodrębnionego widoku na RestaurantCell. Twój kod powinien wyglądać tak:

var body: some View {

NavigationView {

List(restaurantItems) { restaurantItem in

RestaurantCell()

}.navigationTitle("Boston, MA")

}

}

}

struct ContentView_Previews: PreviewProvider {

static var previews: some View {

ContentView(restaurantItems: testData)

}

}

struct RestaurantCell: View {

var body: some View {

NavigationLink(destination:

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 widoku RestaurantCell, jak pokazano:

```
struct ContentView: View {

var restaurantItems: [RestaurantItem] = []

var body: some View {

NavigationView {

List(restaurantItems) { restaurantItem in

RestaurantCell(restaurantItem:

restaurantItem)

}.navigationTitle("Boston, MA"

}

}

}
```

12. Sprawdź, czy podgląd nadal działa tak jak wcześniej.

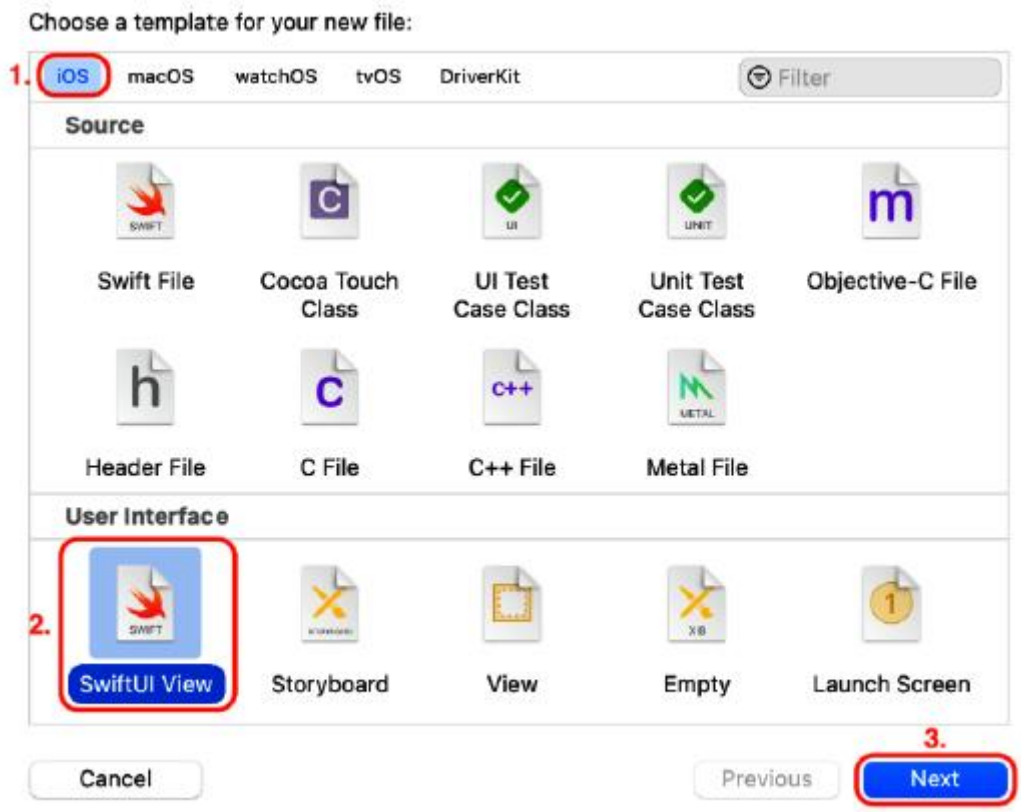
Ukończyłeś implementację ekranu Lista restauracji. Następnie zobaczysz, w jaki sposób można jednocześnie używać widoków UIKit i SwiftUI, aby utworzyć widok mapy, którego będziesz używać na ekranie szczegółów restauracji.

Używanie jednocześnie widoków UIKit i SwiftUI

W tym momencie utworzyłeś ekran listy restauracji, a dotknięcie każdej komórki na tym ekranie wyświetla nazwę restauracji na drugim ekranie. Zmodyfikujesz swoją aplikację tak, aby wyświetlała ekran szczegółów restauracji po dotknięciu komórki na ekranie listy restauracji, ale wcześniej utworzysz widok SwiftUI, który wyświetla 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 można użyć dowolnej podklasy widoku w SwiftUI, pakując je w widok SwiftUI zgodny z protokołem UIViewRepresentable. Utwórzmy teraz niestandardowy widok, który może teraz prezentować widok mapy. Wykonaj następujące kroki:

1. Wybierz Plik | Nowy | Plik, aby otworzyć 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 wyświetlonym błędem, naprawisz go w kilku następnych krokach:

```
import SwiftUI

import MapKit

struct MapView: UIViewRepresentable {

var body: some View {

Text("Hello World")

}

}
```

Protokół UIViewRepresentable to otoka, która umożliwia korzystanie z dowolnego widoku UIKit w hierarchii widoków SwiftUI.

5. Aby zapewnić zgodność z protokołem UIViewRepresentable, potrzebne są dwie metody: metoda makeUIView(context:), która tworzy MKMapView oraz metoda updateUIView(_:context:), która ją konfiguruje i odpowiada 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:) tuż po metodzie makeUIView(context:). Spowoduje to ustawienie regionu widoku mapy, aby wyśrodkować mapę na lokalizacji The Tap Trailhouse:

func updateUIView(_ uiView: MKMapView, context:

Context) {

let coordinate = CLLocationCoordinate2D

(latitude: 42.360847, longitude: -71.056819)

let span = MKCoordinateSpan(latitudeDelta:

0.001, longitudeDelta: 0.001)

let region = MKCoordinateRegion(center:

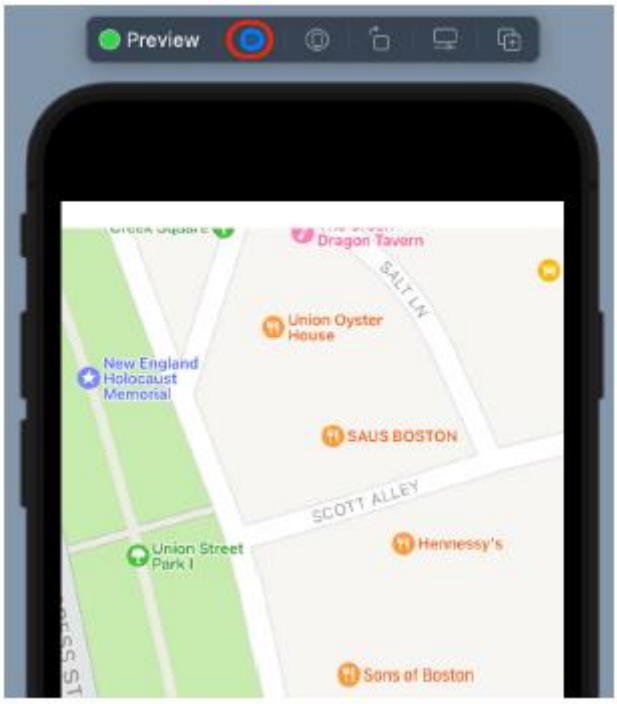
coordinate, span: span)

uiView.setRegion(region, animated: true)

}

Zwróć uwagę, że jest to ta sama metoda, której użyłeś do utworzenia regionu dla ekranu Mapa w aplikacji Let's Eat.

7. Błąd zniknął, a na płótnie pojawia się pusty widok mapy. Dzieje się tak, ponieważ podgląd jest w trybie statycznym i renderuje tylko widoki SwiftUI. Aby zobaczyć mapę, musisz włączyć podgląd na żywo. Kliknij przycisk Podgląd na żywo, a za chwilę powinna pojawić się mapa Bostonu wyśrodkowana na lokalizacji The Tap Trailhouse:



Jeśli to nie zadziała, sprawdź połączenie internetowe i kliknij przycisk Spróbuj ponownie lub Wznów nad podglądem.

8. Wartości szerokości i długości geograficznej są obecnie zakodowane na stałe. Zadeklaruj dwie właściwości do przechowywania 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:), aby używała tych właściwości zamiast wartości zakodowanych na stałe:

func updateUIView(_ view: MKMapView, context: Context) {

let coordinate = CLLocationCoordinate2D(

latitude: lat, longitude: long)

10. Zaktualizuj strukturę MapView_Previews, aby przekazywać 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)

}

}

```
struct MapView_Previews: PreviewProvider {

static var previews: some View {

MapView(lat: 42.360847, long: -71.056819)

}

}
```

Wypełnianie ekranu szczegółów restauracji

Masz teraz widok mapy SwiftUI wyświetlający mapę. Teraz utworzysz nowy widok SwiftUI reprezentujący ekran szczegółów restauracji i dodasz do niego widok mapy. Wykonaj następujące kroki:

1. Wybierz Plik | Nowy | Plik, aby otworzyć selektor szablonów.

2. iOS powinien być już wybrany. W sekcji Interfejs użytkownika kliknij 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:

```
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 obejmuje cztery widoki tekstu. Wyświetlają nazwę restauracji, kuchnie, adres i miasto. Obiekt Spacer przesuwa pierwszy widok Vstack na górę ekranu. Instancja RestaurantItem jest przypisana do właściwości selectedRestaurant, a dane z tej instancji są używane do wypełniania widoku struktury RestaurantDetail. Aby utworzyć podgląd na kanwie, struktura RestaurantDetail_Previews przechodzi do pierwszego wystąpienia RestaurantItem w tablicy testData.

Należy zauważyć, że wystąpienie RestaurantDetail jest zawarte w wystąpieniu NavigationView, aby pasek nawigacyjny był wyświetlany w podglądzie.

5. Podgląd wyświetla widok mapy nad widokami tekstu restauracji, ale nie renderuje mapy. Tak jak poprzednio, kliknij przycisk Podgląd na żywo.

6. Płótno wyświetla teraz ekran szczegółów restauracji z wyrenderowaną mapą:

Ukończyłeś implementację ekranu szczegółów restauracji za pomocą SwiftUI. Teraz zmodyfikujesz listę na ekranie listy restauracji, aby ekran szczegółów restauracji był wyświetlany po dotknięciu komórki.

7. Kliknij plik ContentView w nawigatorze projektów i zmodyfikuj kod struktury RestaurantCell do używania struktury RestaurantDetail jako miejsca docelowego po dotknięciu komórki:

var body: some View {

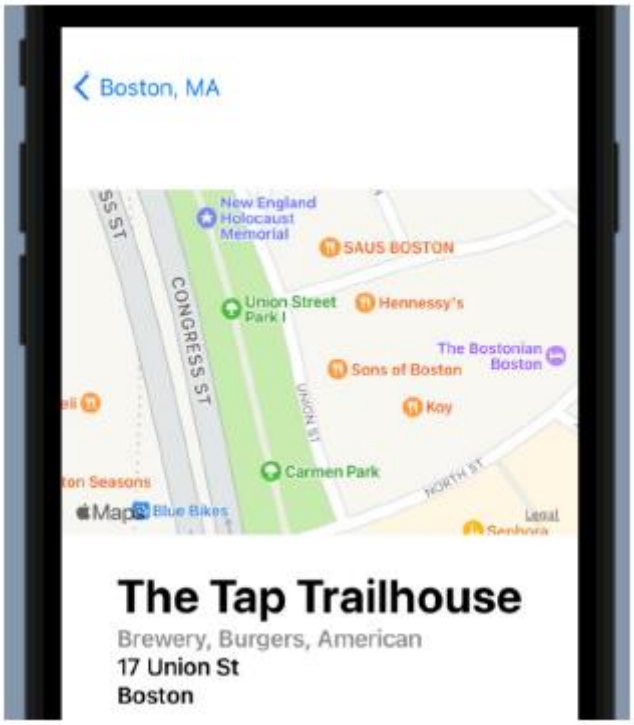
NavigationLink(destination:

RestaurantDetail(selectedRestaurant:

restaurantItem)){

Spacer()

8. Kliknij przycisk Podgląd na żywo na płótnie. Stuknij wiersz na ekranie Lista restauracji. Zobaczysz ekran szczegółów restauracji dla tej restauracji:



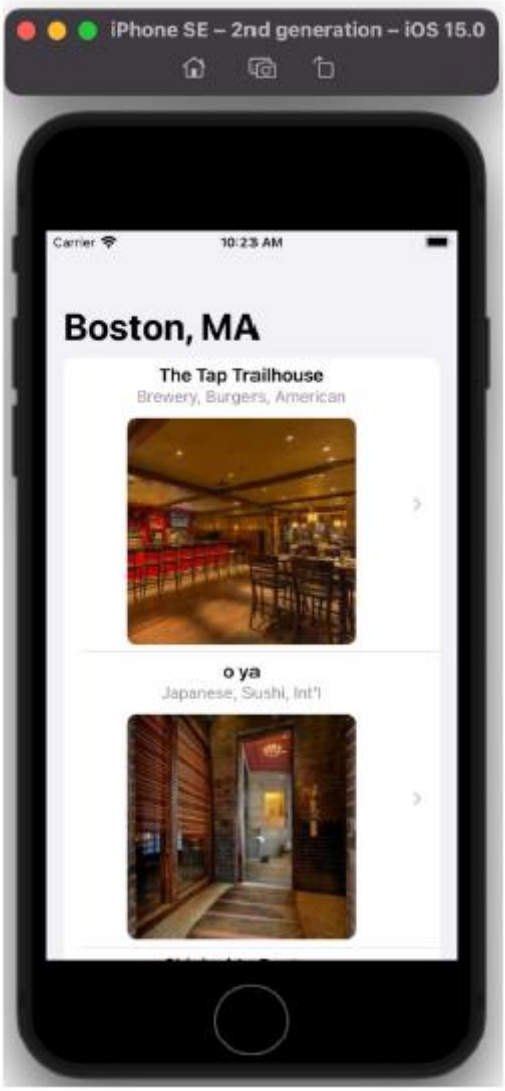
Jak widać, podgląd aplikacji działa dobrze na płótnie. Jeśli chcesz uruchomić w symulatorze, musisz wprowadzić jedną niewielką zmianę w strukturze ContentView. Kliknij plik ContentView w nawigatorze projektów i przypisz tablicę testdata do właściwości restaurantItems, jak pokazano:

struct ContentView: View {

var restaurantItems: [RestaurantItem] = testData

var body: some View {

Zbuduj i uruchom swoją aplikację, a pojawi się ona w symulatorze:



Ukończyłeś budowę prostej aplikacji SwiftUI! Wspaniale!

Podsumowanie

W tym krótkim wprowadzeniu do SwiftUI zobaczyłeś, jak zbudować uproszczoną wersję aplikacji Let's Eat przy użyciu SwiftUI. Zacząłeś od dodania i skonfigurowania widoków SwiftUI, aby utworzyć ekran listy restauracji. Następnie dodano obiekty modelu do aplikacji i skonfigurowano nawigację między 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 wcześniej utworzony widok mapy. Wiesz już, jak używać SwiftUI do tworzenia aplikacji, która odczytuje obiekty modelu, prezentuje je na liście i umożliwia nawigację do drugiego ekranu zawierającego widok mapy. Możesz to następnie wdrożyć do własnych projektów.