Pierwsze kroki z Swift Concurrency

Firma Apple wprowadziła Swift Concurrency podczas WWDC2021, 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. Tu poznasz podstawowe pojęcia Swift Concurrency. Następnie zbadasz aplikację bez współbieżności i zbadasz jej problemy. Następnie użyjesz async/ await do zaimplementowania współbieżności w aplikacji. Następnie zwiększysz wydajność swojej aplikacji, korzystając z asynchronicznego polecenia. Na koniec zmodyfikujesz klasę RestaurantListViewController w aplikacji Let's Eat, aby używać asynchronicznego/oczekiwania do ładowania obrazów restauracji. Pod koniec tego rozdziału nauczysz się podstaw działania Swift Concurrency oraz aktualizacji własnych aplikacji, aby z niej korzystać.

Omówione zostaną następujące tematy:

• Zrozumienie szybkiej współbieżności

• Badanie aplikacji bez współbieżności

• Aktualizowanie aplikacji za pomocą asynchronicznego/oczekiwania

• Poprawa wydajności za pomocą asynchronicznego-let

• Aktualizowanie kontrolera RestaurantListViewController do korzystania z funkcji asynchronicznej/oczekiwania

Zrozumienie szybkiej współbieżności

W Swift 5.5 firma Apple dodała 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, jednocześnie wykonując operacje, takie jak pobieranie danych z Internetu. Kod równoległy umożliwia aplikacji uruchamianie wielu fragmentów kodu jednocześnie. Aby zorientować się, jak działa Swift Concurrency, wyobraź sobie, że robisz na śniadanie kanapkę z jajkiem w koszulce. Oto jeden ze sposobów na zrobienie tego:

1. Włóż dwie kromki chleba do tostera.

2. Odczekaj dwie minuty, aż chleb się upiecze.

3. Włóż jajko do miski z niewielką ilością wody i włóż miskę do kuchenki mikrofalowej.

4. Poczekaj sześć minut, aż jajko się ugotuje.

5. Zrób kanapkę.

Zajmuje to łącznie osiem minut. Pomyśl teraz o tej sekwencji wydarzeń. Czy spędzasz ten czas tylko wpatrując się w toster i kuchenkę mikrofalową? Prawdopodobnie będziesz używać telefonu, gdy chleb jest w tosterze, a jajko w kuchence mikrofalowej. Innymi słowy, możesz robić inne rzeczy podczas przygotowywania chleba i jajka. Tak więc sekwencja wydarzeń byłaby dokładniej opisana w następujący sposób:

1. Włóż dwie kromki chleba do tostera.

2. Korzystaj z telefonu przez dwie minuty, aż chleb się upiecze.

3. Włóż jajko do miski z niewielką ilością wody i włóż miskę do kuchenki mikrofalowej.

4. Korzystaj z telefonu przez sześć minut, aż jajko się ugotuje.

5. Zrób kanapkę.

Tutaj możesz zobaczyć, że twoja interakcja z tosterem i kuchenką mikrofalową może zostać zawieszona, a następnie wznowiona, co oznacza, że te operacje są asynchroniczne. Operacja nadal trwa osiem minut, ale w tym czasie można było robić inne rzeczy. Jest jeszcze jeden czynnik do rozważenia. 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. Gdy chleb się opieka się, włóż jajko do miski z niewielką ilością wody i włóż miskę do kuchenki mikrofalowej.

3. Korzystaj z telefonu przez sześć minut, aż jajko się ugotuje.

4. Zrób kanapkę.

Opiekanie chleba i gotowanie jajka odbywa się teraz równolegle, co pozwala zaoszczędzić dwie minuty. Świetny! Pamiętaj jednak, że masz więcej rzeczy do śledzenia. Teraz, gdy rozumiesz koncepcje operacji asynchronicznych i równoległych, przyjrzyjmy się problemom związanym z aplikacją, która nie ma współbieżności w następnej akcji.

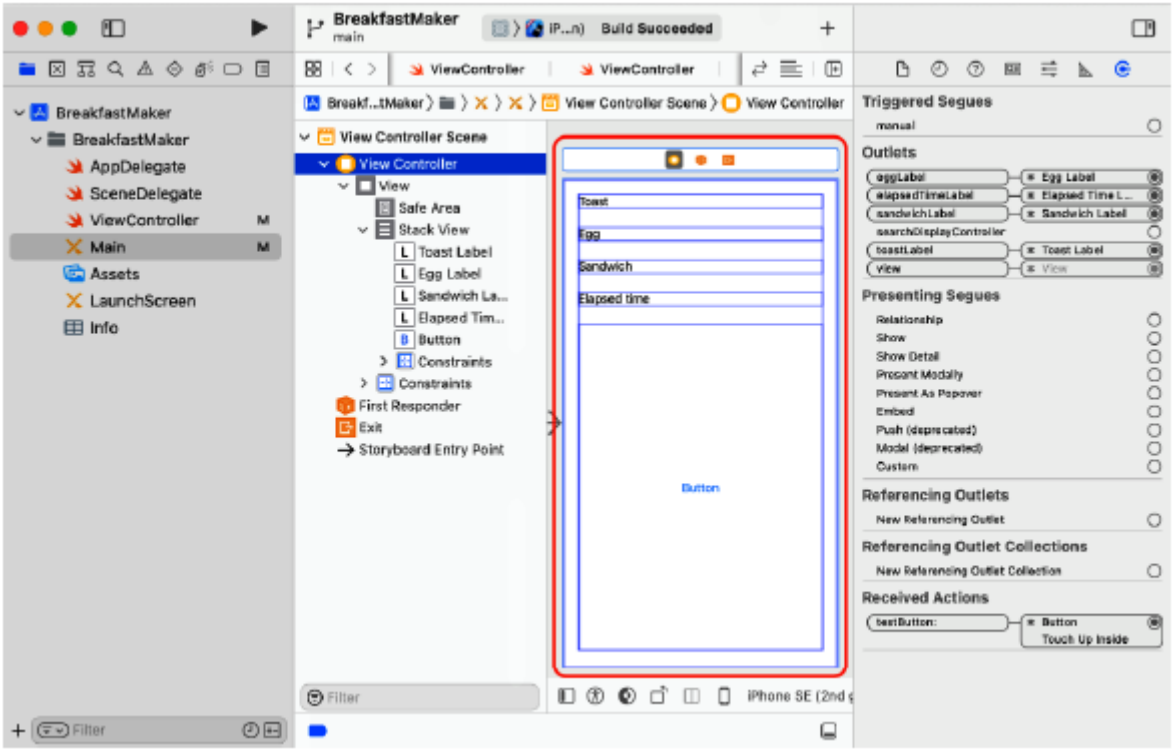
Badanie aplikacji bez współbieżności

Widziałeś, jak operacje asynchroniczne i równoległe mogą pomóc w szybszym przygotowaniu śniadania i umożliwić korzystanie z telefonu podczas jego robienia. Przyjrzyjmy się teraz przykładowej aplikacji, która symuluje 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 Chapter24 pakietu kodu dla tej książki pod tym linkiem: https://github.com/PacktPublishing/iOS-15-Programming-for-Beginners-Sixth-Edition.

2. Otwórz folder zasobów w folderze Chapter24, 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 stan tostów, jajek i kanapek oraz czas przygotowania kanapki. Aplikacja wyświetli również przycisk, za pomocą którego możesz przetestować responsywność interfejsu użytkownika.

4. Kliknij plik ViewController w nawigatorze projektów. 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, ten kod symuluje proces robienia śniadania, który został opisany w poprzedniej sekcji. Podzielmy to:

@IBOutlet var toastLabel: UILabel!

@IBOutlet var eggLabel: UILabel!

@IBOutlet var sandwichLabel: UILabel!

@IBOutlet zmienna elapsedTimeLabel: UILabel!

Te punkty sprzedaży są połączone z czterema etykietami w głównym pliku scenorysu. Po uruchomieniu aplikacji te etykiety będą wyświetlać stan tostów, jajek i kanapek, a także czas potrzebny na ukończenie procesu.

override func viewDidAppear(_ Animation: Bool) {

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

let startTime = Date().timeIntervalSince1970

Ustawia to startTime na bieżący czas, dzięki czemu aplikacja może później obliczyć, ile czasu zajmie zrobienie kanapki.

toastLabel.text = "Making toast..."

Powoduje to, że etykieta toast wyświetla tekst Robienie tostów....

toastLabel.text = makeToast()

Wywołuje to metodę makeToast(), która czeka dwie sekundy, aby zasymulować czas potrzebny na zrobienie tosta, a następnie zwraca tekst Toast done, który zostanie wyświetlony przez toastLabel.

eggLabel.text = "Poaching egg..."

To sprawia, że eggLabel wyświetla tekst Jajko w koszulce....

eggLabel.text = poachEgg()

Wywołuje to metodę poachEgg(), która czeka sześć sekund, aby zasymulować czas potrzebny na przygotowanie jajka, a następnie zwraca tekst Egg done, który zostanie wyświetlony przez eggLabel.

sandwichLabel.text = makeSandwich()

Wywołuje to metodę makeSandwich(), która zwraca tekst Sandwich done, który zostanie wyświetlony przez sandwichLabel.

let endTime = Date().timeIntervalSince1970

To ustawia endTime na bieżący czas.

elapsedTimeLabel.text = "Elapsed time is

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

/ 100) seconds"

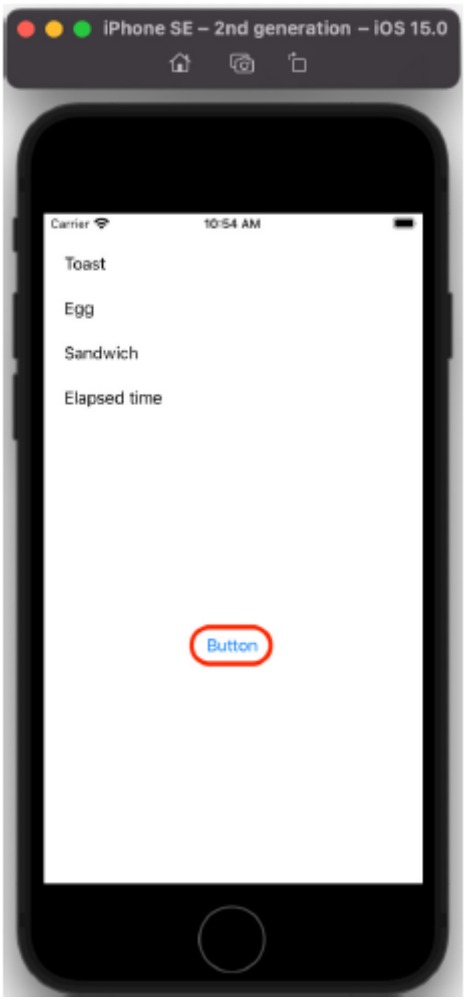
Oblicza to czas, który upłynął (około osiem sekund), który zostanie wyświetlony przez elapsedTimeLabel.

@IBAction func testButton(_ sender: UIButton) {

print("Button tapped")

}

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



Powinieneś zauważyć następujące problemy:

• Naciśnięcie przycisku początkowo nie ma żadnego efektu, a po około ośmiu sekundach zobaczysz tylko przycisk Naciśnięty przycisk w obszarze Debug.

• Robienie tostów... i Jajko w koszulce... nigdy nie są wyświetlane, a Tost gotowy i Jajko gotowe pojawiają się dopiero po około ośmiu sekundach.

Powodem tego jest to, że kod aplikacji nie aktualizował interfejsu użytkownika podczas działania metod makeToast() i poachEgg(). Twoja aplikacja zarejestrowała naciśnięcia przycisków, ale była w stanie je przetworzyć i zaktualizować etykiety dopiero po zakończeniu wykonywania 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 async/await, aby mogła aktualizować interfejs użytkownika podczas działania metod makeToast() i poachEgg().

Aktualizowanie aplikacji za pomocą async/await

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

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

Zapisanie słowa kluczowego await przed wywołaniem metody oznacza punkt, w którym wykonanie może zostać zawieszone, co umożliwia uruchamianie innych operacji. Tak to wygląda:

await methodName()

Zmodyfikujesz swoją aplikację tak, aby używała asynchronicznego/oczekiwania. Umożliwi to zawieszenie metod makeToast() i poachEgg() w celu przetwarzania naciśnięć przycisków i aktualizacji interfejsu użytkownika, a następnie wznowienia wykonywania obu metod. Wykonaj następujące kroki:

1. Zmodyfikuj metody makeToast() i poachEgg() tak, jak pokazano, aby kod w ich ciałach 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"

}

Task reprezentuje jednostkę pracy asynchronicznej. Zadanie posiada metodę statyczną sleep(nanoseconds:), która wstrzymuje wykonanie na określony czas, mierzony w nanosekundach. Pomnożenie przez 1 000 000 000 zamienia czas trwania na sekundy. Słowo kluczowe await 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:

```
31
32      func makeToast() -> String {
33          try! await Task.sleep(nanoseconds: 2 *
                1_000_000_000)
34          return "Toast done"
35      }
36
37      func poachEgg() -> String {
38          try! await Task.sleep(nanoseconds: 6 *
                1_000_000_000)
39          return "Egg done"
40      }
41
```

Błąd jest wyświetlany, ponieważ wywołujesz metodę asynchroniczną wewnątrz metody, która nie obsługuje współbieżności. Musisz dodać słowo kluczowe async do deklaracji metody, aby wskazać, że jest 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 błędu, aby wyświetlić komunikat o błędzie, który będzie taki sam jak komunikat, który widziałeś wcześniej. Dzieje się tak, ponieważ wywołujesz metodę asynchroniczną wewnątrz metody, która nie obsługuje współbieżności.

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

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

Ten komunikat o błędzie jest wyświetlany, ponieważ await nie został użyty podczas wywoływania funkcji asynchronicznej.

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

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

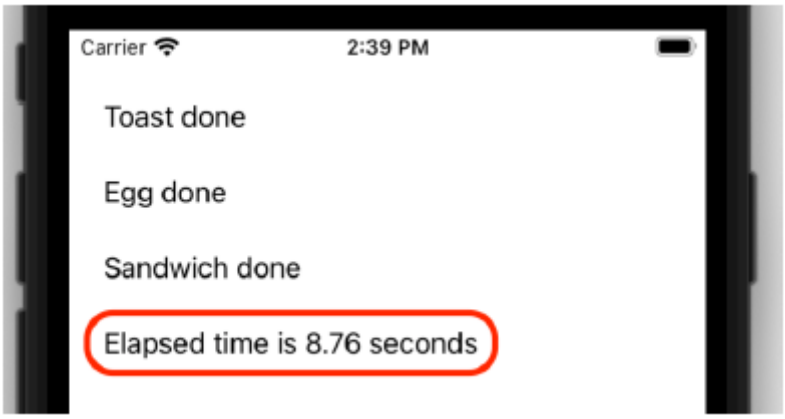
10. Kliknij ikonę błędu w deklaracji metody viewDidLoad(), aby wyświetlić komunikat o błędzie:

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

11. Aby rozwiązać ten problem, usuniesz słowo kluczowe async i zamkniesz cały kod po super.viewDidAppear() w bloku Task, co pozwoli na jego asynchroniczne wykonanie 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 = await makeToast()

eggLabel.text = "Poaching egg..."

eggLabel.text = await poachEgg()

sandwichLabel.text = makeSandwich()

let endTime = Date().timeIntervalSince1970

elapsedTimeLabel.text = "Elapsed time is

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

/ 100) seconds"

}

}
```

Zbuduj i uruchom aplikację, a następnie naciśnij przycisk, gdy tylko zobaczysz interfejs użytkownika. Zauważ, że naciśnięty przycisk pojawia się teraz natychmiast w obszarze debugowania, a etykiety są aktualizowane tak, jak powinny. Dzieje się tak, ponieważ aplikacja może teraz zawiesić metody makeToast() i poachEgg(), aby reagować na stuknięcia i aktualizować interfejs użytkownika, a następnie wznawiać je później. Wspaniały! Jeśli jednak spojrzysz na czas, jaki upłynął, zobaczysz, że aplikacja potrzebuje nieco więcej czasu na przygotowanie śniadania niż wcześniej:



Wynika to częściowo z narzutu wymaganego do zawieszania i wznawiania metod, ale jest to związane z innym czynnikiem. Mimo że metody makeToast() i poachEgg() są teraz asynchroniczne, metoda poachEgg() rozpoczyna wykonywanie dopiero po zakończeniu wykonywania metody makeToast(). W następnej sekcji zobaczysz, jak używać asynchronicznego-let do równoległego uruchamiania metod makeToast() i poachEgg().

Poprawa wydajności za pomocą async-let

Mimo że Twoja aplikacja reaguje teraz na naciśnięcia przycisków i może aktualizować interfejs użytkownika podczas działania metod makeToast() i poachEgg(), obie metody nadal są wykonywane sekwencyjnie. Rozwiązaniem tutaj jest użycie asynchronicznego let. Pisanie async przed instrukcją let, gdy definiujesz stałą, a następnie pisanie await, gdy uzyskujesz dostęp do stałej, umożliwia równoległe wykonywanie metod asynchronicznych:

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 używała asynchronicznego let w celu umożliwienia równoległego działania 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

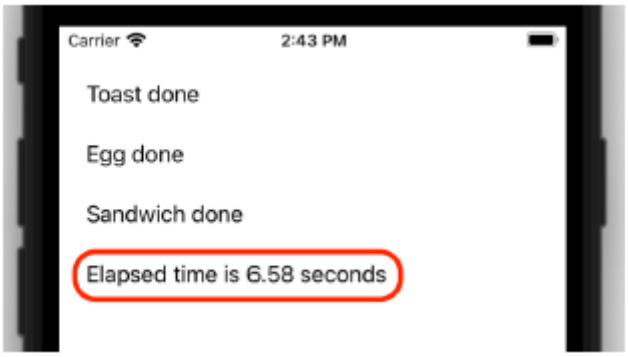
elapsedTimeLabel.text = "Elapsed time is

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

/ 100) seconds"

}

Zbuduj i uruchom aplikację. Zobaczysz, że upływający czas jest teraz krótszy niż wcześniej:



Dzieje się tak, ponieważ użycie async-let umożliwia równoległe działanie metod makeToast() i poachEgg(), a metoda poachEgg() nie czeka już na zakończenie metody makeToast() przed rozpoczęciem wykonywania. Gruchać! W następnej sekcji zaktualizujesz klasę RestaurantListViewController w aplikacji Let's Eat, aby używać asynchronicznego/oczekiwania podczas pobierania obrazów restauracji.

Aktualizowanie kontrolera RestaurantListViewController w celu użycia asynchronicznego/oczekiwania

Po uruchomieniu aplikacji Let's Eat możesz zauważyć opóźnienie, gdy na ekranie Lista restauracji wyświetla się lista restauracji. Dzieje się tak, ponieważ kod używany do pobierania obrazów restauracji nie jest asynchroniczny, a aplikacja nie może wykonywać innych czynności podczas pobierania obrazów restauracji. Kod, który pobiera dane obrazu restauracji i konwertuje je na obraz, znajduje się w metodzie collectionView(_:cellForItemAt:) w definicji klasy RestaurantListViewController. Zmodyfikujesz ten kod tak, aby był wykonywany asynchronicznie. Otwórz projekt LetsEat i otwórz plik RestaurantListViewController (w folderze Restauracje) w nawigatorze projektów. Zaktualizuj metodę collectionView(_:cellForItemAt:), jak pokazano poniżej:

```
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:

Task {

Tworzy to jednostkę pracy asynchronicznej.

guard let url = URL(string: imageURL)

else {

return

}

Ta instrukcja Guard tworzy adres URL z właściwości imageURL instancji RestaurantItem i przypisuje go do adresu URL, a następnie zwraca, jeśli nie jest w stanie tego zrobić.

let (imageData, response) = try await

URLSession.shared.data(from: url)

To asynchronicznie pobiera dane z adresu URL przechowywanego w url i przypisuje je do imageData. Odpowiedź z serwera jest przypisywana do odpowiedzi.

guard let httpResponse = response as? HTTPURLResponse,

httpResponse.statusCode == 200 else {

return

}

Ta instrukcja strażnika sprawdza, czy kod odpowiedzi serwera to 200 (co oznacza, że pobieranie się powiodło) i zwraca, jeśli tak nie jest.

guard let cellImage = UIImage(data: imageData) else {

return

}

Ta instrukcja guard tworzy wystąpienie UIImage na podstawie danych przechowywanych w imageData i przypisuje do cellImage i zwraca, jeśli nie jest w stanie tego zrobić.

cell.restaurantImageView.image = cellImage

Spowoduje to przypisanie UIImage przechowywanego w cellImage do właściwości restaurantImageView instancji restaurantCell, która będzie wyświetlana w widoku kolekcji ekranu listy restauracji. W przeciwnym razie zostanie wyświetlony domyślny obraz ustawiony dla właściwości restaurantImageView.

return cel

Zwraca to wystąpienie restaurantCell.

Zbuduj i uruchom swoją aplikację. Zauważysz, że ekran listy restauracji będzie bardziej responsywny i przewijał się płynniej niż wcześniej:

Jeśli wyłączysz połączenie internetowe, ekran Lista restauracji będzie nadal działał, ale zamiast tego będą wyświetlane domyślne obrazy zastępcze:

Pomyślnie zaimplementowano kod asynchroniczny w klasie RestaurantListViewController aplikacji. Fantastyczny! Jest jeszcze wiele rzeczy do nauczenia się o Swift Concurrency, takich jak ustrukturyzowana współbieżność i aktorzy, ale to wykracza poza zakres tego rozdziału.

Podsumowanie

Dowiedziałeś się o Swift Concurrency oraz o tym, jak wdrożyć ją zarówno w aplikacjach BreakfastMaker, jak i Let's Eat. Zacząłeś od poznania podstawowych pojęć Swift Concurrency. Następnie sprawdziłeś aplikację bez współbieżności i zbadałeś jej problemy. Następnie zaimplementowałeś współbieżność w aplikacji za pomocą async/await. Następnie ulepszyłeś swoją aplikację, używając async-let. Na koniec zaktualizowano klasę RestaurantListViewController w aplikacji Let's Eat, aby używać asynchronicznego/oczekiwania do ładowania obrazów restauracji. Rozumiesz teraz podstawy Swift Concurrency i możesz teraz używać async/await i async-let we własnych aplikacjach.