SwiftUI & async / await: how does it work?

Vincent Pradeilles (@v_pradeilles) – PhotoRoom

I’m Vincent 👋 🇫🇷

Do you remember WWDC 2021?


APPLE
IOS DEVELOPER
WE ADDED
ASYNC/AWAIT TO SWIFT
AND IT WON'T
REQUIRE IOS 15, RIGHT?
IT WON'T
REQUIRE IOS 15, RIGHT?
imgflip.com

But then, in December 2021…

John Sundell
@johnsundell

Amazing news, everyone! The new Swift concurrency system is now backward compatible all the way back to iOS 13, macOS Catalina, watchOS 6, and tvOS 13! 🎉😀🚀

Huge thanks to everyone at Apple and in the Swift open source community who made this happen! 🙌

Traduire le Tweet

Swift

New Features

- You can now use Swift Concurrency in applications that deploy to macOS 10.15, iOS 13, tvOS 13, and watchOS 6 or newer. This support includes async/await, actors, global actors, structured concurrency, and the task APIs. (70738378)

Now is a perfect time to start using async / await with SwiftUI 🔥

Why async / await?

Why async / await?

Asynchronous code is the corner stone of most iOS app

Why async / await?

Asynchronous code is the corner stone of most iOS app

```
let url = URL(string: "http://myapi.com/path")!

URLSession.shared.dataTask(with: url) { data, response, error in
    // use result of network call
}.resume()
```

Why async / await?

Asynchronous code is the corner stone of most iOS app

```
let url = URL(string: "http://myapi.com/path")!

URLSession.shared.dataTask(with: url) { data, response, error in
    // use result of network call
}.resume()
```

This asynchronous code is implemented through a completion handler

Why async / await?

Completion handlers have been very popular in the iOS SDK

Why async / await?

Completion handlers have been very popular in the iOS SDK

They work very well with Swift built-in support of closures 👍

Why async / await?

Completion handlers have been very popular in the iOS SDK

They work very well with Swift built-in support of closures 👍

They are pretty easy to understand for beginners 👍

However!

However!

Composing completion handlers is not a fun experience

However!

Composing completion handlers is not a fun experience

```
getUserId { userId in



}
```

However!

Composing completion handlers is not a fun experience

```
getUserId { userId in
    getUserFirstName(userId: userId) { firstName in



    }
}
```

However!

Composing completion handlers is not a fun experience

```
getUserId { userId in
    getUserFirstName(userId: userId) { firstName in
        getUserLastName(userId: userId) { lastName in
            print("Hello \(firstName) \(lastName)")
        }
    }
}
```

However!

Composing completion handlers is not a fun experience

```
getUserId { userId in
    getUserFirstName(userId: userId) { firstName in
        getUserLastName(userId: userId) { lastName in
            print("Hello \(firstName) \(lastName)")
        }
    }
}
```

It’s so bad that it even has names: callback hell, pyramid of doom, etc.

It’s even worst if we want to add concurrent execution 😰

How do we deal with this?

Option 1
Using specialized libraries

Using specialized libraries

Set of APIs to structure the way we use closures: Combine, RxSwift, etc.

Using specialized libraries

Set of APIs to structure the way we use closures: Combine, RxSwift, etc.

```
getUserId()
.flatMap { userId in
    return getUserFirstName(userId: userId).zip(getUserLastName(userId: userId))
}.sink { (firstName, lastName) in
    print("Hello using Combine \(firstName) \(lastName)")
}
```

Using specialized libraries

Set of APIs to structure the way we use closures: Combine, RxSwift, etc.

```
getUserId()
.flatMap { userId in
    return getUserFirstName(userId: userId).zip(getUserLastName(userId: userId))
}.sink { (firstName, lastName) in
    print("Hello using Combine \(firstName) \(lastName)")
}
```

Closures are still here, but we are forcing them to behave.

Using specialized libraries

Set of APIs to structure the way we use closures: Combine, RxSwift, etc.

```
getUserId()
.flatMap { userId in
    return getUserFirstName(userId: userId).zip(getUserLastName(userId: userId))
}.sink { (firstName, lastName) in
    print("Hello using Combine \(firstName) \(lastName)")
}
```

Closures are still here, but we are forcing them to behave.

It’s definitely better, but we had to introduce a lot of extra syntax…

Option 2
Built-in language support

Built-in language support

Let’s see how “the other guys” are doing it!

Built-in language support

Let’s see how “the other guys” are doing it!

```
static async Task Main(string[] args)
{
    Coffee cup = PourCoffee();
    Console.WriteLine("coffee is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryBaconAsync(3);
    var toastTask = MakeToastWithButterAndJamAsync(2);

    var eggs = await eggsTask;
    Console.WriteLine("eggs are ready");

    var bacon = await baconTask;
    Console.WriteLine("bacon is ready");

    var toast = await toastTask;
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

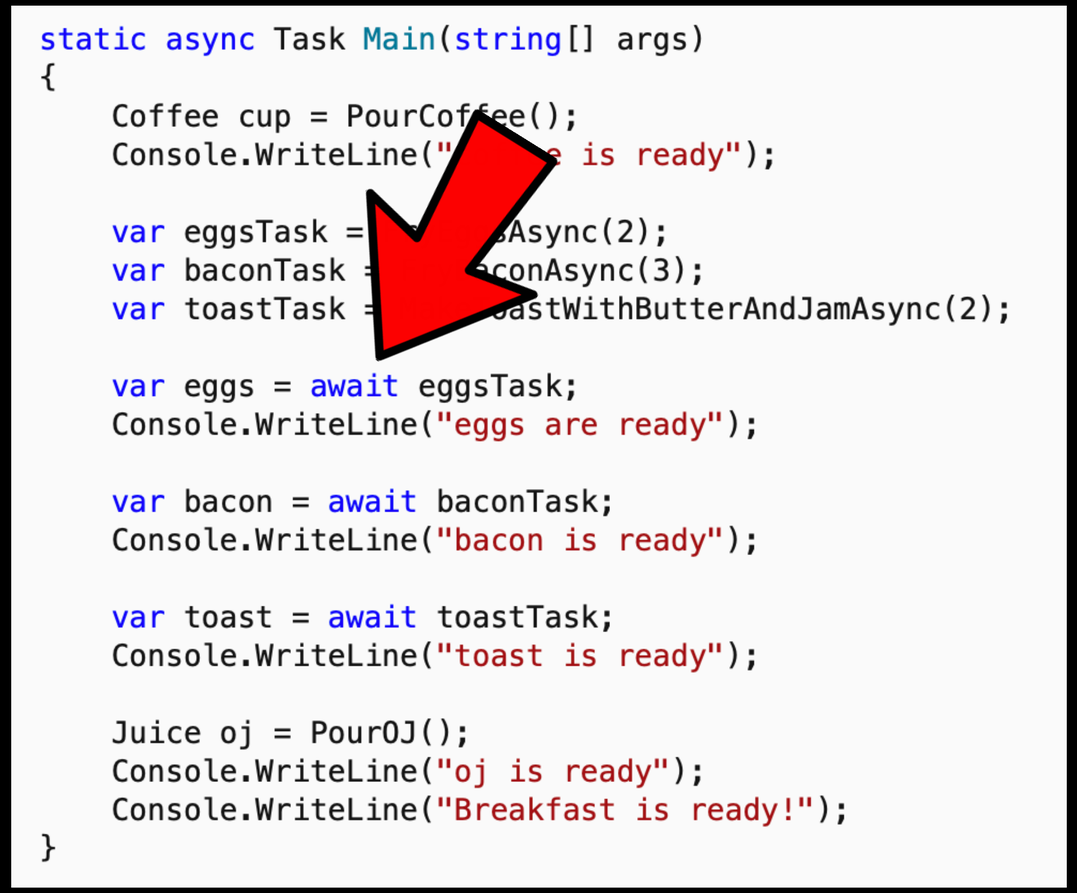
Built-in language support

Let’s see how “the other guys” are doing it!

```
static async Task Main(string[] args)
{
    ffee cup = PourCoffee();
    nsole.WriteLine("coffee is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryBaconAsync(3);
    var toastTask = MakeToastWithButterAndJamAsync(2);

    var eggs = await eggsTask;
    Console.WriteLine("eggs are ready");

    var bacon = await baconTask;
    Console.WriteLine("bacon is ready");

    var toast = await toastTask;
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

Built-in language support

Let’s see how “the other guys” are doing it!



```
static async Task Main(string[] args)
{
    Coffee cup = PourCoffee();
    Console.WriteLine("cup is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryBaconAsync(3);
    var toastTask = MakeToastWithButterAndJamAsync(2);

    var eggs = await eggsTask;
    Console.WriteLine("eggs are ready");

    var bacon = await baconTask;
    Console.WriteLine("bacon is ready");

    var toast = await toastTask;
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

Built-in language support

Let’s see how “the other guys” are doing it!

```
static async Task Main(string[] args)
{
    Coffee cup = PourCoffee();
    Console.WriteLine("coffee is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryB      sync(3);
    var toastTask = Mak       tWithButterAndJamAsync(2);

    var eggs = await          ask;
    Console.WriteLi       gs are ready");

    var bacon = await baconTask;
    Console.WriteLine("bacon is ready");

    var toast = await toastTask;
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

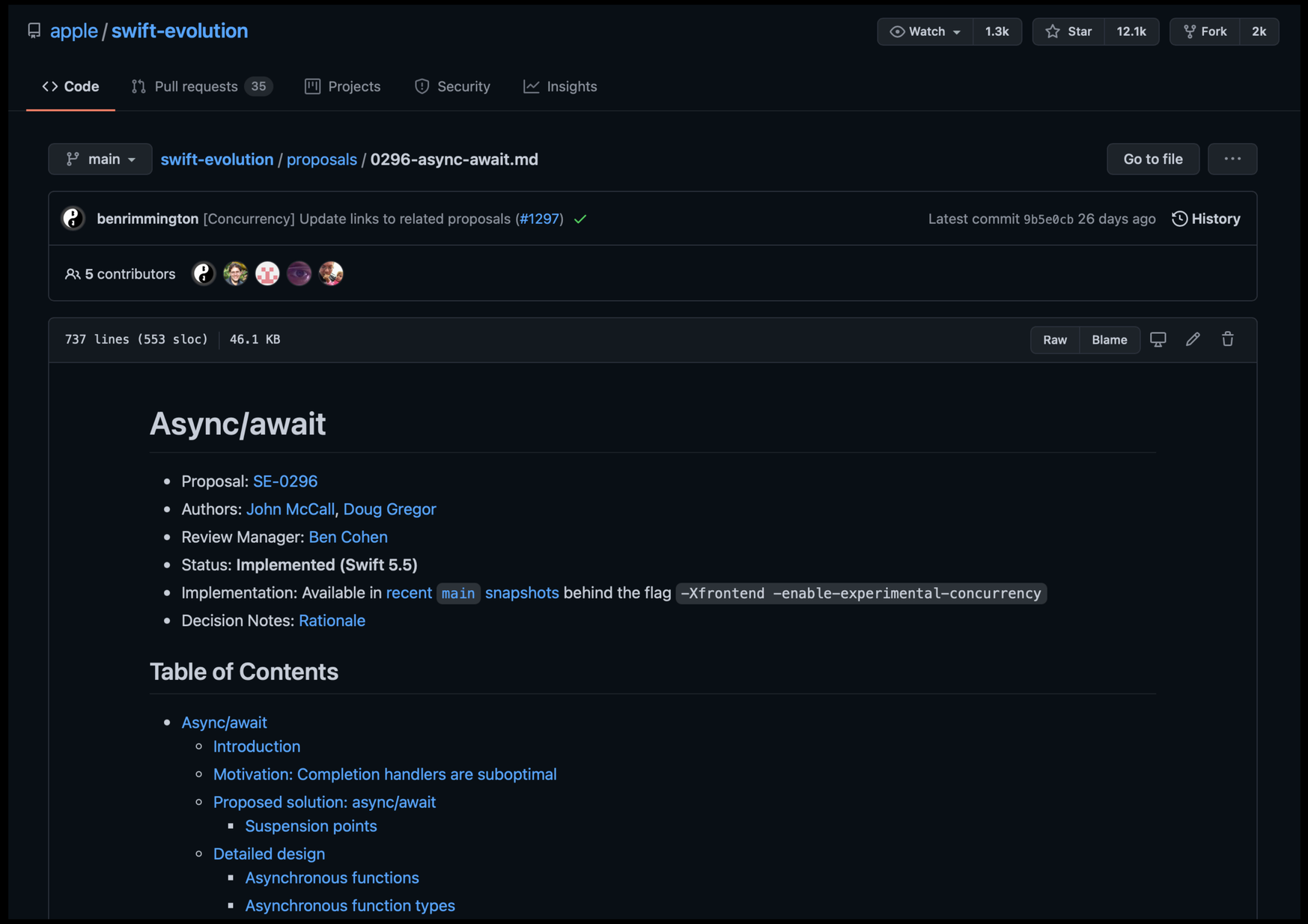
Built-in language support

Let’s see how “the other guys” are doing it!

```
static async Task Main(string[] args)
{
    Coffee cup = PourCoffee();
    Console.WriteLine("coffee is ready");

    var eggsTask = FryEggsAsync(2);
    var baconTask = FryBaconAsync(3);
    var toastTask = MakeToastWithButterAndJamAsync(2);

    var eggs = await egg
    Console.WriteLine("
                        are ready");

    var bacon = awai
                        nTask;
    Console.WriteLi
                        on is ready");

    var toast = await toastTask;
    Console.WriteLine("toast is ready");

    Juice oj = PourOJ();
    Console.WriteLine("oj is ready");
    Console.WriteLine("Breakfast is ready!");
}
```

Pretty cool, isn’t it?

Well it’s now also part of Swift!



https://github.com/apple/swift-evolution/blob/main/proposals/0296-async-await.md

Let’s start using it 🚀

We’ll start by implementing a network call using async / await

network call using async / await

`URLSession` now comes with shiny new **`async`** methods 🙌

network call using async / await

`URLSession` now comes with shiny new **`async`** methods 🙌

```
func data(from url: URL,
          delegate: URLSessionTaskDelegate? = nil) async throws
          -> (Data, URLResponse)
```

network call using async / await

`URLSession` now comes with shiny new **`async`** methods 🙌

```
func data(from url: URL,
          delegate: URLSessionTaskDelegate? = nil) async throws
          -> (Data, URLResponse)
```

Which enables pretty cool call sites:

network call using async / await

`URLSession` now comes with shiny new **`async`** methods 🙌

```
func data(from url: URL,
          delegate: URLSessionTaskDelegate? = nil) async throws
          -> (Data, URLResponse)
```

Which enables pretty cool call sites:

```
let url = URL(string: "https://api.themoviedb.org/3/movie/
upcoming?api_key=\(apiKey)&language=en-US&page=\(page)")!

let (data, _) = try await URLSession.shared.data(from: url)
```

However, there’s a catch!

However, there’s a catch!

While the async / await language feature has been back ported to iOS 13…

…all the APIs in Foundation, UIKit, SwiftUI, etc. haven’t been! 😩

```
let (data, _) = try await URLSession.shared.data(from: url)
```

'data(from:delegate:)' is only available in iOS 15.0 or newer

Add 'if #available' version check — Fix

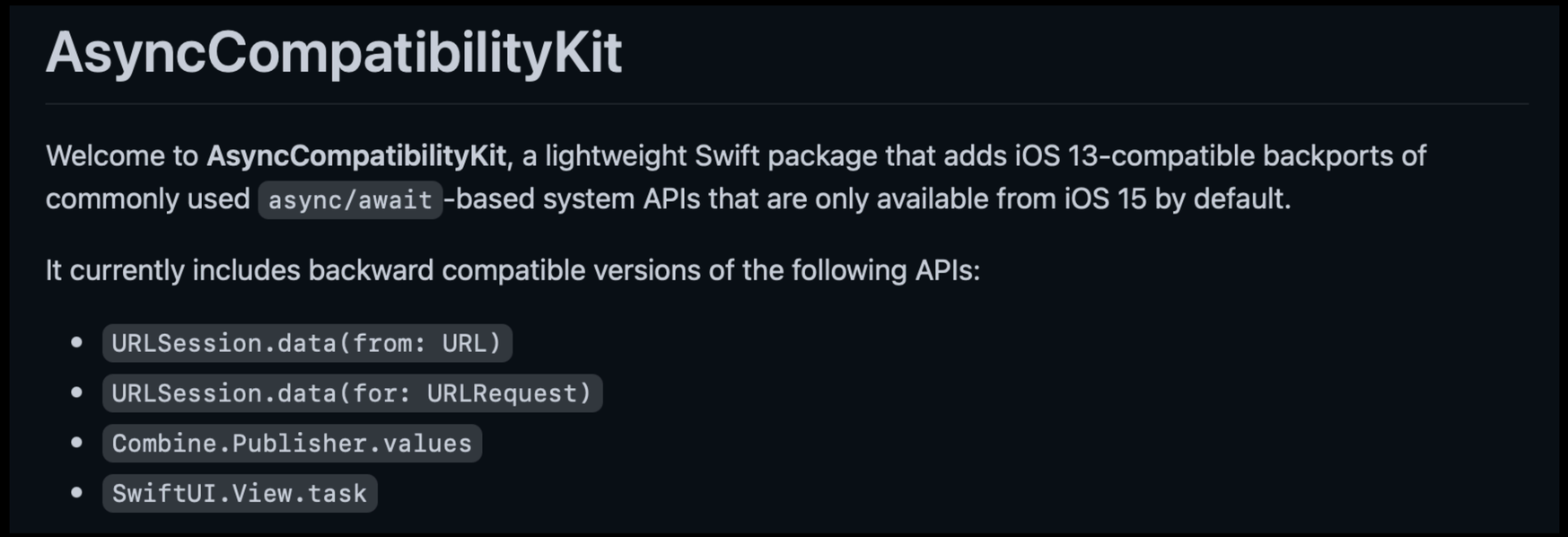
Add @available attribute to enclosing global function — Fix

However, there’s a catch!

It’s a bit annoying, but it’s not a showstopper!

Because we can implement our own wrappers!

There are already some great compatibility wrappers implemented by the iOS community: https://github.com/JohnSundell/AsyncCompatibilityKit



Taking a look at a simplified networking and model layer

simplified networking and model layer

```
struct Movie: Decodable, Equatable, Identifiable {
    let id: Int
    let title: String
    let overview: String
}

struct MovieResponse: Decodable {
    let results: [Movie]
}

let apiKey = "redacted 🙈"
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

simplified networking and model layer

```
func loadMovies(page: Int = 1) async -> [Movie] {
    do {
        let url = URL(string: "https://api.themoviedb.org/3/movie/upcoming?
api_key=\(apiKey)&language=en-US&page=\(page)")!

        let (data, _) = try await URLSession.shared.data(from: url)

        let decoder = JSONDecoder()

        let decoded = try decoder.decode(MovieResponse.self, from: data)

        return decoded.results
    } catch {
        return []
    }
}
```

Now to building the view

building the view

```
struct ContentView: View {

    @State var movies = [Movie]()

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
        }
    }
}
```

How do we call `loadMovies(page:)`?

Building the view

```
struct ContentView: View {

    @State var movies = [Movie]()

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
        }
        .onAppear {  Invalid conversion from 'async' function of type '() async -> ()' to synchronous function type '() -> Void'
            movies = await loadMovies()
        }
    }
}
```

Building the view

```
struct ContentView: View {

    @State var movies = [Movie]()

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
        }
        .task { ✅
            movies = await loadMovies()
        }
    }
}
```

Now let’s implement an infinite scroll

infinite scroll

```
struct ContentView: View {

    @State var movies = [Movie]()
    @State var currentPage = 1

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
            .task {
                if movie == movies.last {
                    currentPage += 1
                    movies += await loadMovies(page: currentPage)
                }
            }
        }
        .task {
            movies = await loadMovies()
        }
    }
}
```

infinite scroll

```
struct ContentView: View {

    @State var movies = [Movie]()
    @State var currentPage = 1

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
            .task {
                if movie == movies.last {
                    currentPage += 1
                    movies += await loadMovies(page: currentPage)
                }
            }
        }
        .task {
            movies = await loadMovies()
        }
    }
}
```

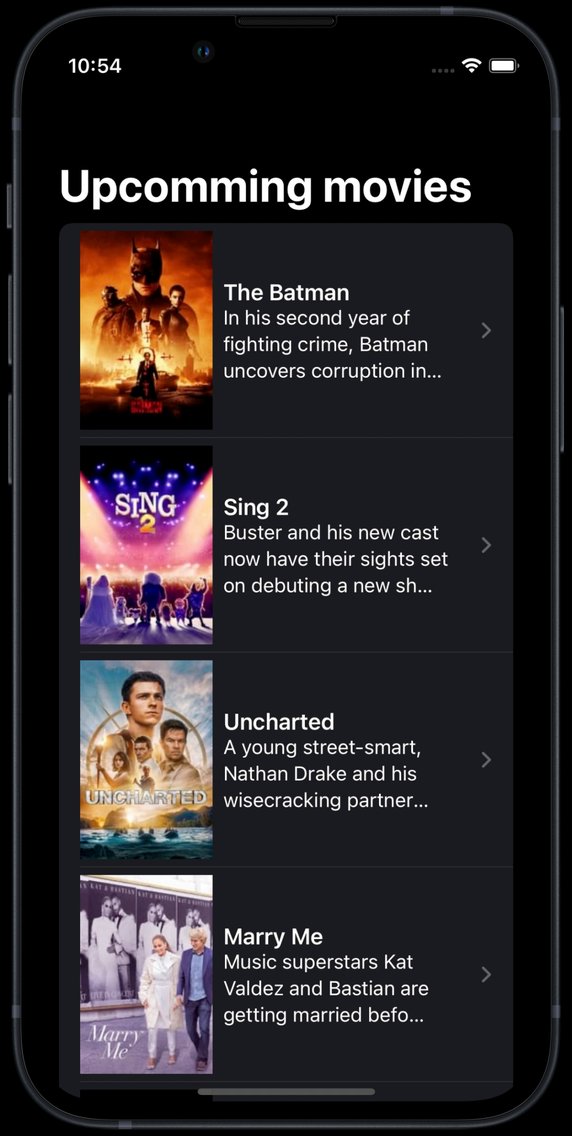
infinite scroll

```
struct ContentView: View {

    @State var movies = [Movie]()
    @State var currentPage = 1

    var body: some View {
        List(movies) { movie in
            HStack(spacing: 10) {
                VStack(alignment: .leading, spacing: 10) {
                    Text(movie.title)
                        .font(.headline)
                    Text(movie.overview)
                        .lineLimit(3)
                        .font(.subheadline)
                }
            }
            .task {
                if movie == movies.last {
                    currentPage += 1
                    movies += await loadMovies(page: currentPage)
                }
            }
        }
        .task {
            movies = await loadMovies()
        }
    }
}
```

Let’s go one step further: Implementing concurrent calls!

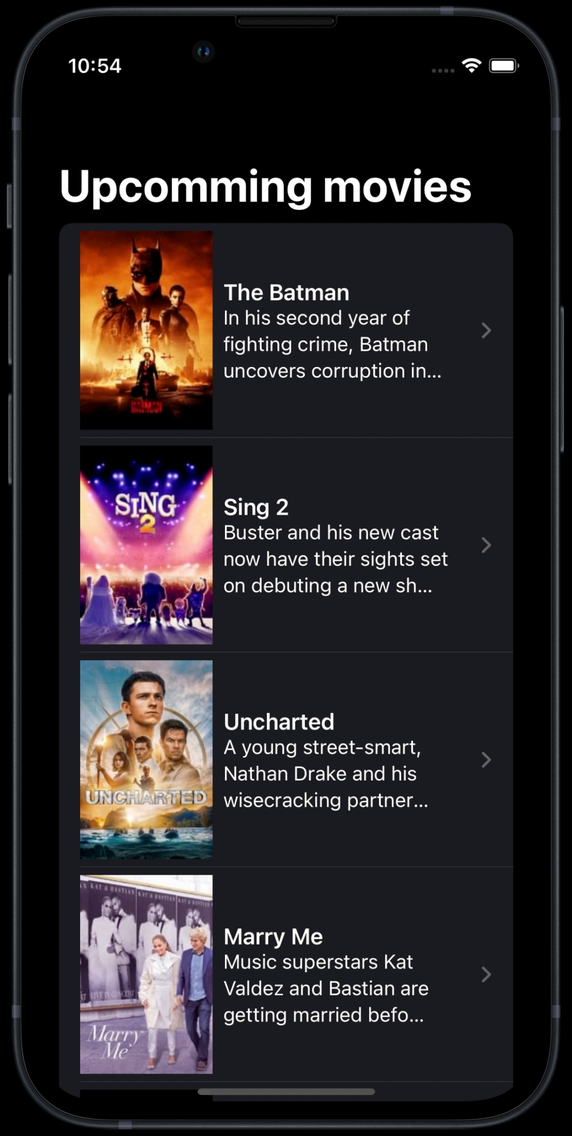
Implementing concurrent calls

I want to implement a screen that displays both the cast and the reviews.



Implementing concurrent calls

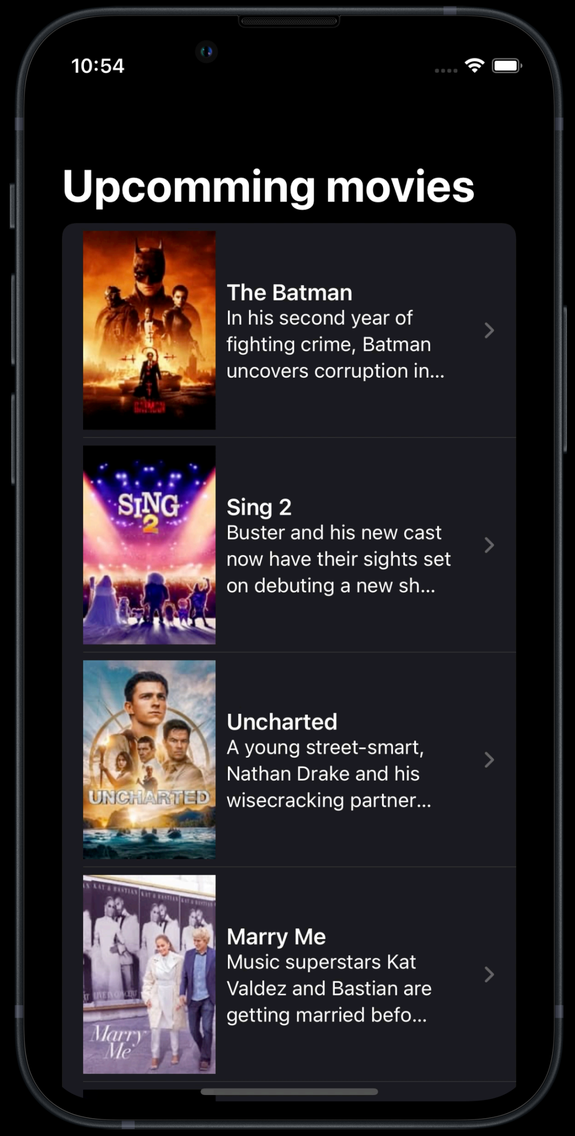
I want to implement a screen that displays both the cast and the reviews.



Implementing concurrent calls

I want to implement a screen that displays both the cast and the reviews.

I'll need to fetch data from two separate endpoints.

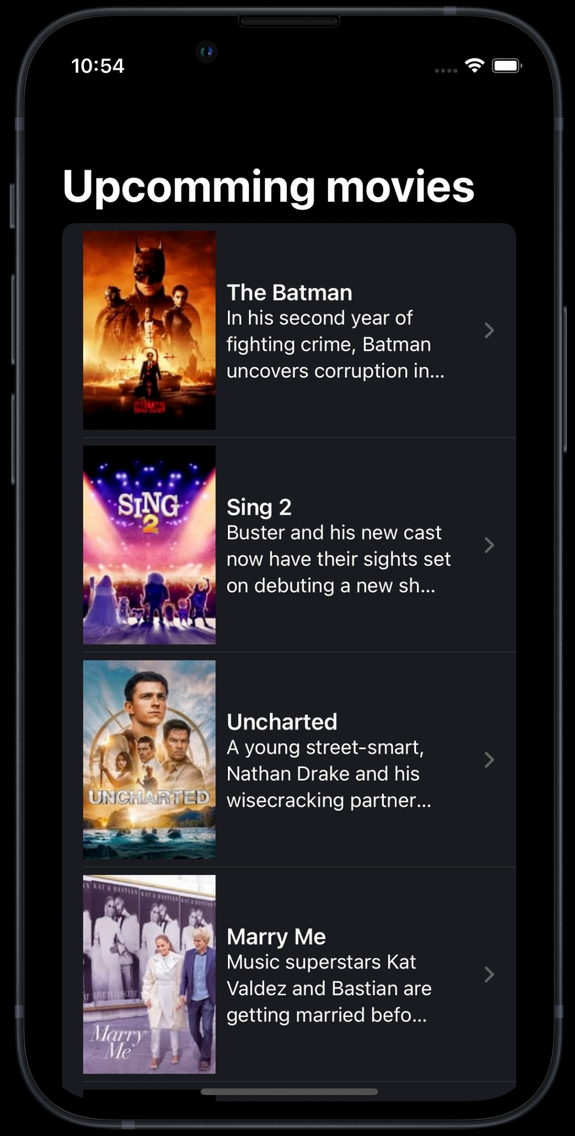


Implementing concurrent calls

I want to implement a screen that displays both the cast and the reviews.

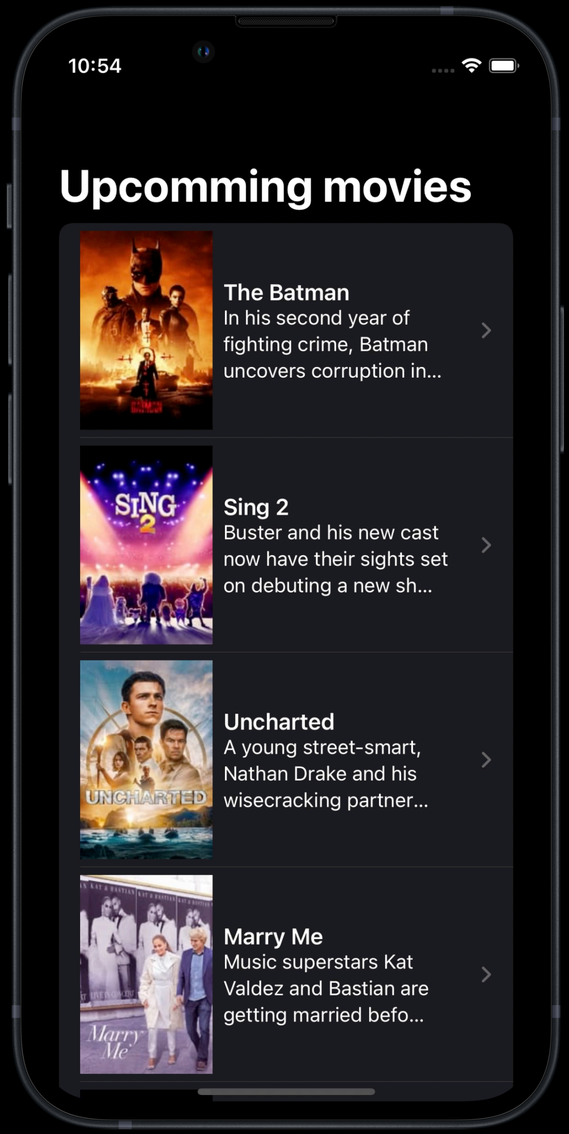
I'll need to fetch data from two separate endpoints.

Let's see how we can make it as efficiently as possible!



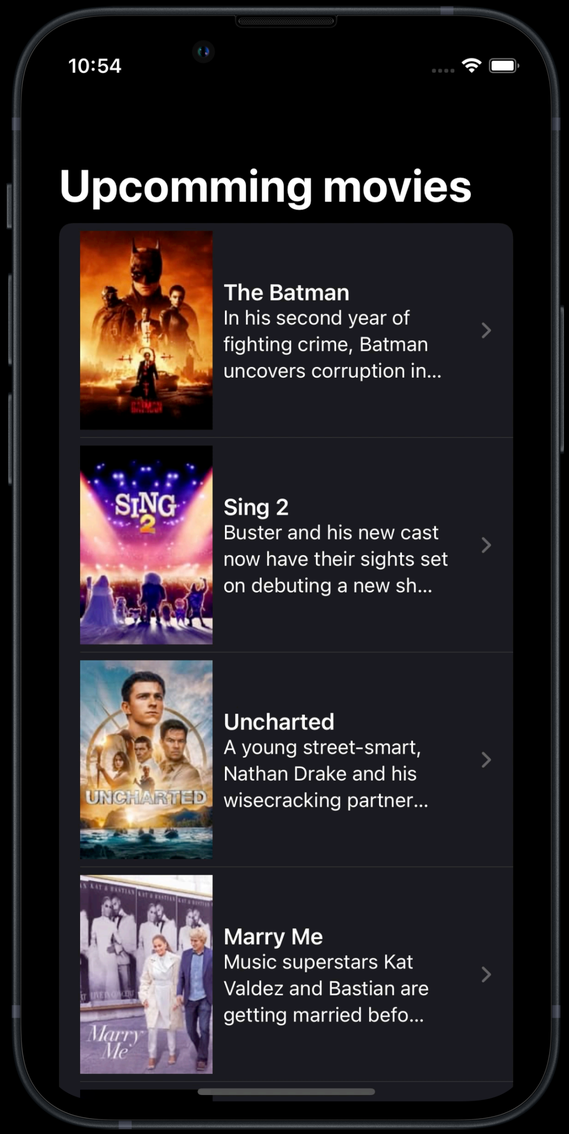
Implementing concurrent calls

```swift
import SwiftUI

struct DetailView: View {

    let movie: Movie

    @State var data: (credits: [MovieCastMember],
                      reviews: [MovieReview]) = ([], [])

    var body: some View {
        List {
            Section(header: Text("Credits")) {
                ForEach(data.credits) { credit in
                    VStack(alignment: .leading) {
                        Text(credit.name)
                            .font(.headline)
                        Text(credit.character)
                            .font(.caption)
                    }
                }
            }

            Section(header: Text("Reviews")) {
                ForEach(data.reviews) { review in
                    VStack(alignment: .leading, spacing: 8) {
                        Text(review.author)
                            .font(.headline)
                        Text(review.content)
                            .font(.body)
                    }
                }
            }
        }
        .navigationBarTitle(movie.title)
        .task {
            let credits = await getCredits(for: movie)
            let reviews = await getReviews(for: movie)

            data = (credits.cast, reviews.results)
        }
    }
}
```



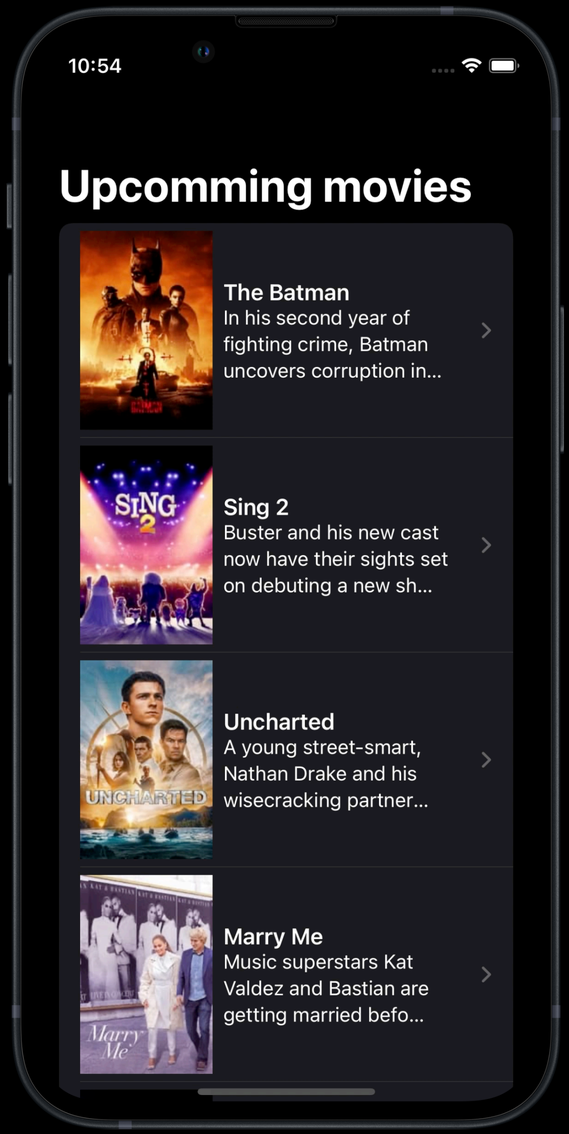
Implementing concurrent calls

```swift
import SwiftUI

struct DetailView: View {

    let movie: Movie

    @State var data: (credits: [MovieCastMember],
                      reviews: [MovieReview]) = ([], [])

    var body: some View {
        List {
            Section(header: Text("Credits")) {
                ForEach(data.credits) { credit in
                    VStack(alignment: .leading) {
                        Text(credit.name)
                            .font(.headline)
                        Text(credit.character)
                            .font(.caption)
                    }
                }
            }

            Section(header: Text("Reviews")) {
                ForEach(data.reviews) { review in
                    VStack(alignment: .leading, spacing: 8) {
                        Text(review.author)
                            .font(.headline)
                        Text(review.content)
                            .font(.body)
                    }
                }
            }
        }
        .navigationBarTitle(movie.title)
        .task {
            let credits = await getCredits(for: movie)
            let reviews = await getReviews(for: movie)

            data = (credits.cast, reviews.results)
        }
    }
}
```



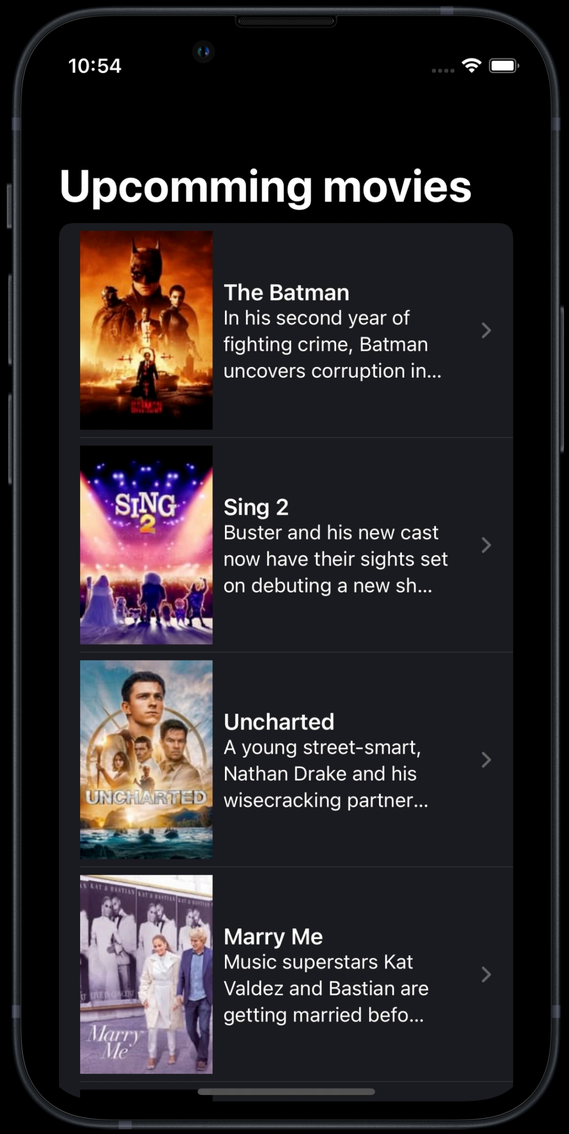
Implementing concurrent calls

```
var body: some View {
        List {
            Section(header: Text("Credits")) {
                ForEach(data.credits) { credit in
                    VStack(alignment: .leading) {
                        Text(credit.name)
                            .font(.headline)
                        Text(credit.character)
                            .font(.caption)
                    }
                }
            }

            Section(header: Text("Reviews")) {
                ForEach(data.reviews) { review in
                    VStack(alignment: .leading, spacing: 8) {
                        Text(review.author)
                            .font(.headline)
                        Text(review.content)
                            .font(.body)
                    }
                }
            }
        }
```



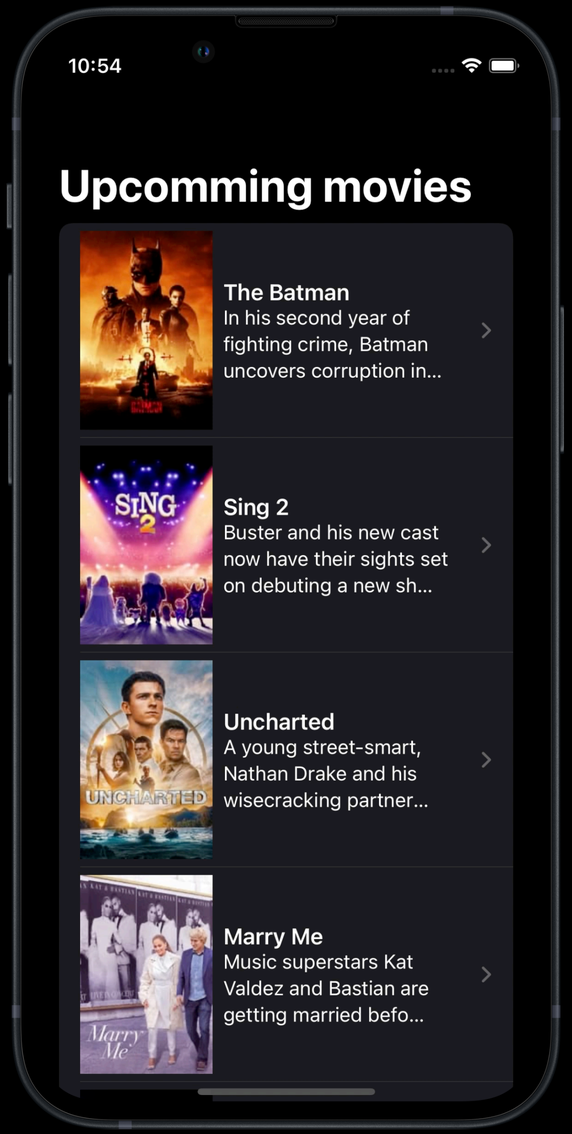
Implementing concurrent calls

```
var body: some View {
        List {
            Section(header: Text("Credits")) {
                ForEach(data.credits) { credit in
                    VStack(alignment: .leading) {
                        Text(credit.name)
                            .font(.headline)
                        Text(credit.character)
                            .font(.caption)
                    }
                }
            }

            Section(header: Text("Reviews")) {
                ForEach(data.reviews) { review in
                    VStack(alignment: .leading, spacing: 8) {
                        Text(review.author)
                            .font(.headline)
                        Text(review.content)
                            .font(.body)
                    }
                }
            }
        }
```



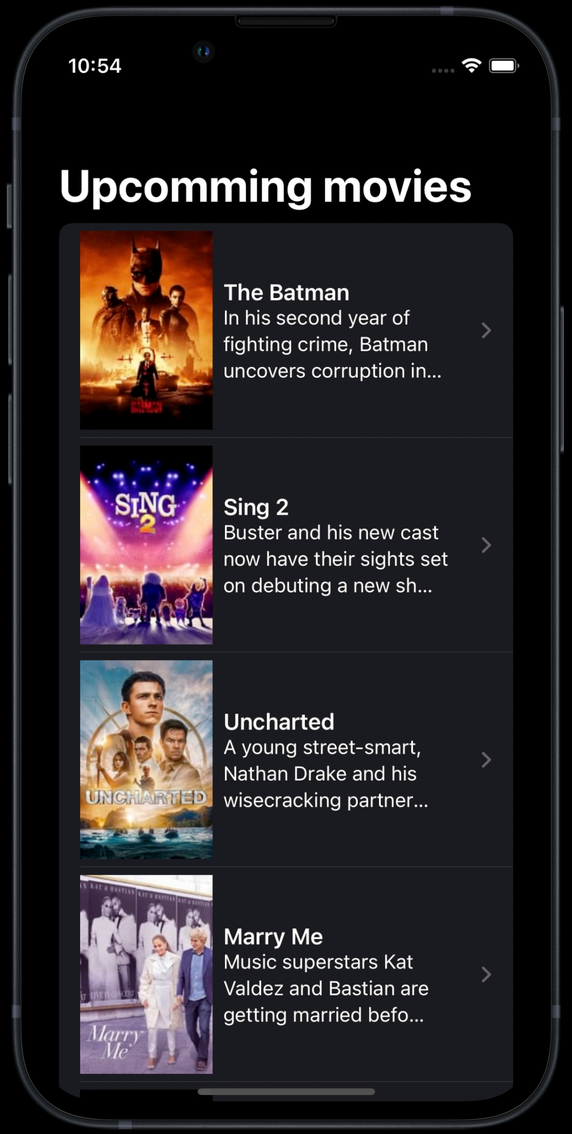
Implementing concurrent calls

```
.task {
    let credits = await getCredits(for: movie)
    let reviews = await getReviews(for: movie)

    data = (credits.cast, reviews.results)
}
```



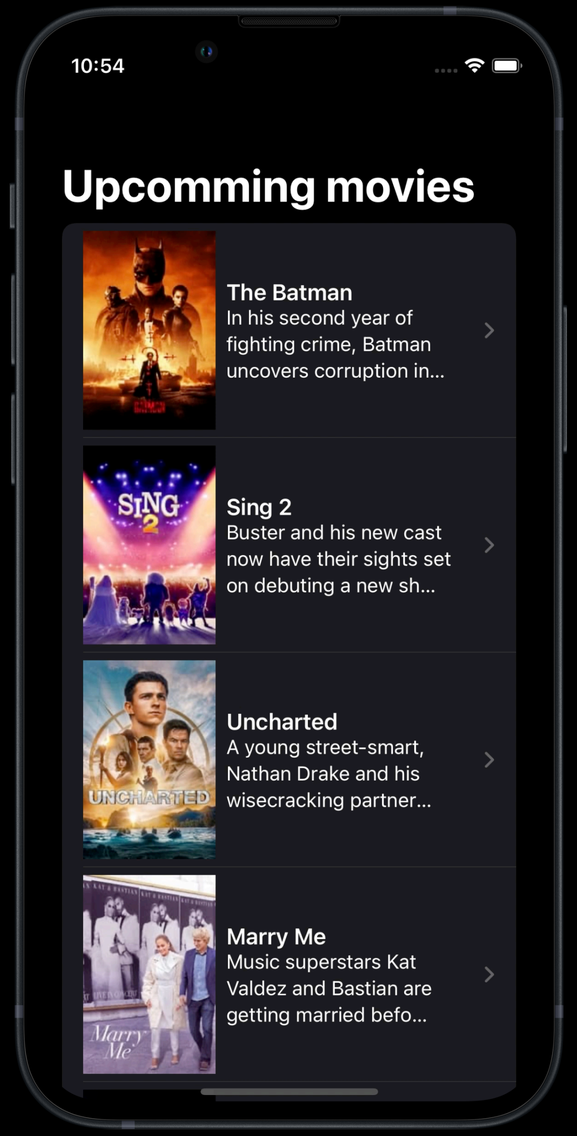
Implementing concurrent calls

```
.task {
    let credits = await getCredits(for: movie)
    let reviews = await getReviews(for: movie)

    data = (credits.cast, reviews.results)
}
```



Implementing concurrent calls

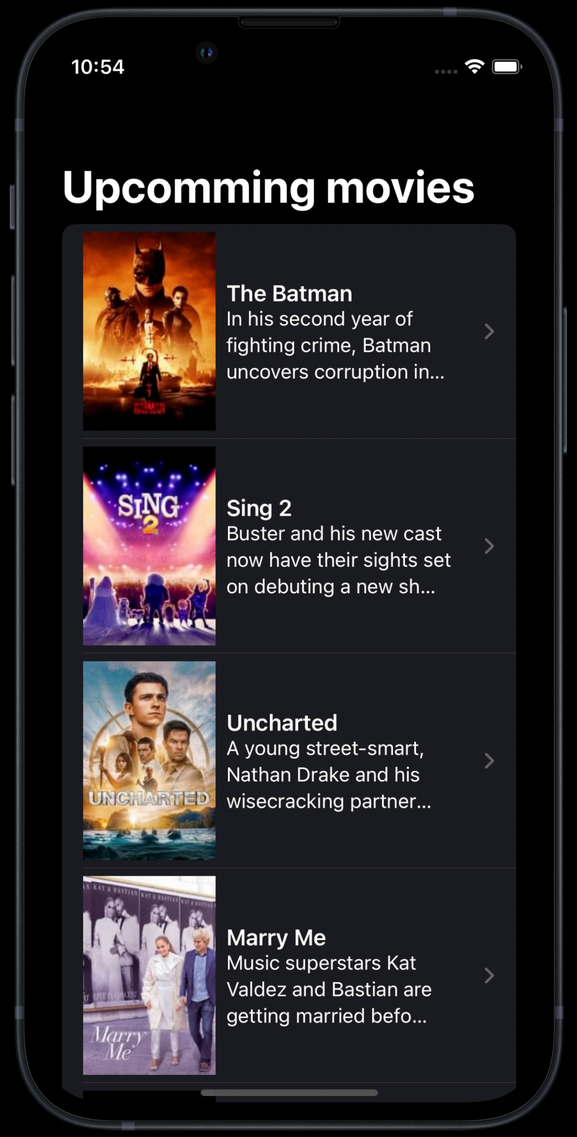
```
.task {
    let credits = await getCredits(for: movie)
    let reviews = await getReviews(for: movie)

    data = (credits.cast, reviews.results)
}
```



It works, but the calls are executed sequentially.

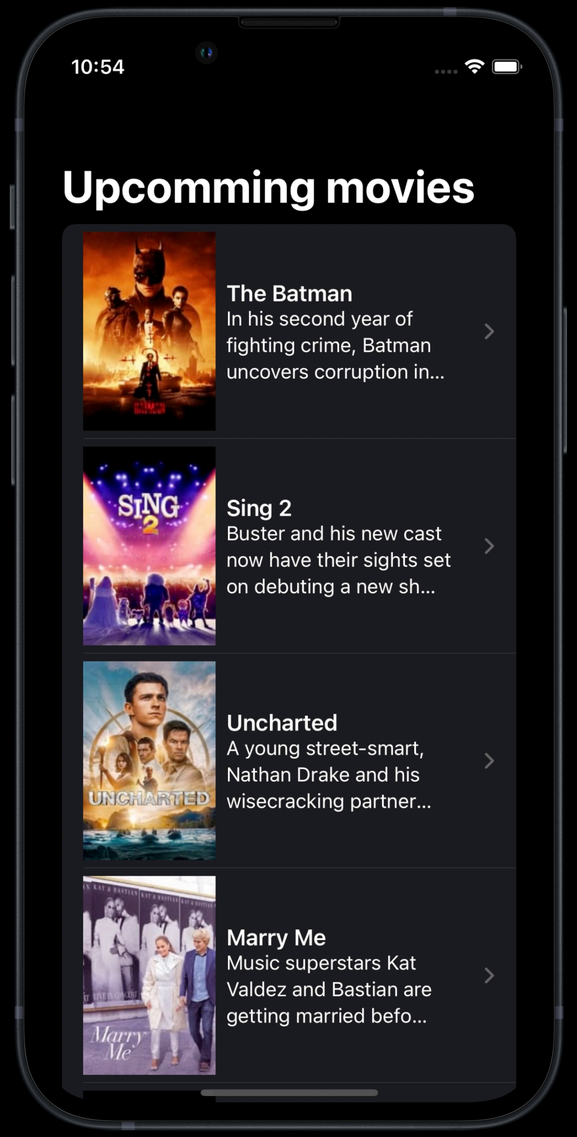
Implementing concurrent calls

```
.task {
    async let credits = getCredits(for: movie)
    async let reviews = getReviews(for: movie)

    data = await (credits.cast, reviews.results)
}
```



Implementing concurrent calls

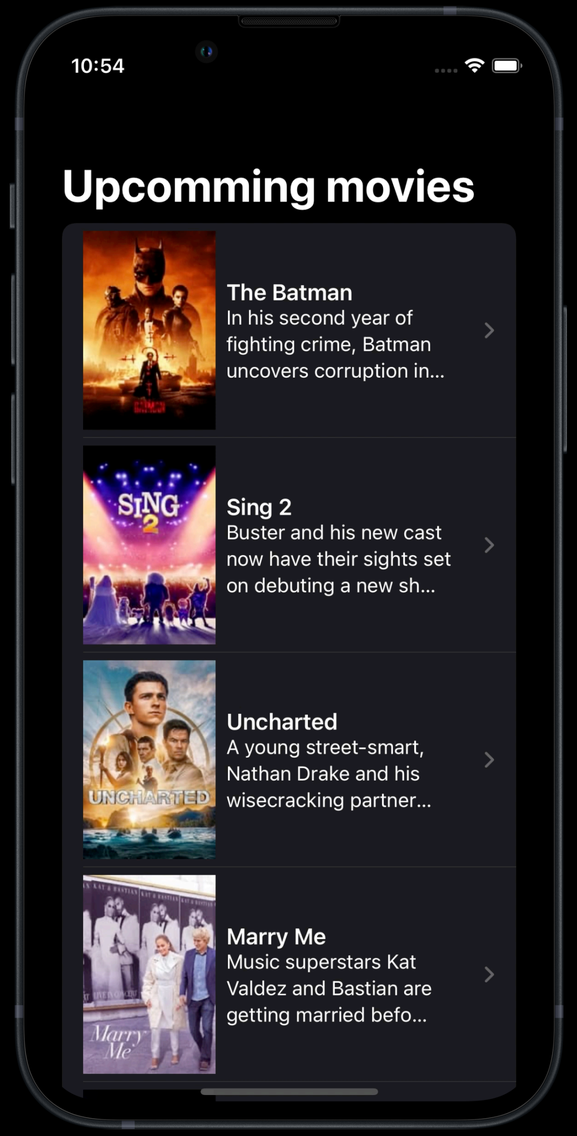
```
.task {
    async let credits = getCredits(for: movie)
    async let reviews = getReviews(for: movie)

    data = await (credits.cast, reviews.results)
}
```



Implementing concurrent calls

```
.task {
    async let credits = getCredits(for: movie)
    async let reviews = getReviews(for: movie)

    data = await (credits.cast, reviews.results)
}
```

Now both calls are happening concurrently 🔥



Where does Combine fit in all this?

Where does Combine fit in all this?

- Combine used to be Apple’s first-party solution to deal with async code

Where does Combine fit in all this?

- Combine used to be Apple's first-party solution to deal with async code
- However, for simple use-cases `async` / `await` is a much better tool

Where does Combine fit in all this?

- Combine used to be Apple's first-party solution to deal with async code
- However, for simple use-cases `async` / `await` is a much better tool
- Still a few weeks ago, I would have argued that Combine was still the right tool for more complex use cases…

Where does Combine fit in all this?

- Combine used to be Apple's first-party solution to deal with async code
- However, for simple use-cases `async` / `await` is a much better tool
- Still a few weeks ago, I would have argued that Combine was still the right tool for more complex use cases…
- …however

Introducing Swift Async Algorithms

MARCH 24, 2022 Tony Parker

> Tony Parker manages teams at Apple working on Foundation and Swift packages.

As part of Swift's move toward safe, simple, and performant asynchronous programming, we are pleased to introduce a new package of algorithms for `AsyncSequence`. It is called **Swift Async Algorithms** and it is available now on GitHub.

This package has three main goals:

- First-class integration with `async/await`
- Provide a home for time-based algorithms
- Be cross-platform and open source

Motivation

AsyncAlgorithms is a package for algorithms that work with *values over time*. That includes those primarily about *time*, like `debounce` and `throttle`, but also algorithms about *order* like `combineLatest` and `merge`. Operations that work with multiple inputs (like `zip` does on `Sequence`) can be surprisingly complex to implement, with subtle behaviors and many edge cases to consider. A shared package can get these details correct, with extensive testing and documentation, for the benefit of all Swift apps.

https://www.swift.org/blog/swift-async-algorithms/

Motivation

AsyncAlgorithms is a package for algorithms that work with *values over time*. That includes those primarily about *time*, like `debounce` and `throttle`, but also algorithms about *order* like `combineLatest` and `merge`. Operations that work with multiple inputs (like `zip` does on `Sequence`) can be surprisingly complex to implement, with subtle behaviors and many edge cases to consider. A shared package can get these details correct, with extensive testing and documentation, for the benefit of all Swift apps.

Motivation

AsyncAlgorithms is a package for algorithms that work with *values over time*. That includes those primarily about *time*, like `debounce` and `throttle`, but also algorithms about *order* like

Combine

Customize handling of asynchronous events by combining event-processing operators.

Overview

The Combine framework provides a declarative Swift API for processing values over time. These values can represent many kinds of asynchronous events. Combine declares *publishers* to expose values that can change over time, and *subscribers* to receive those values from the

Wait and see at WWDC? 🤨


That's all Folks!

Time to recap!

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There’s a good chance that’s it going to progressively replace Combine for most apps

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

- Meet async/await in Swift https://developer.apple.com/videos/play/wwdc2021/10132/

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

- Meet async/await in Swift https://developer.apple.com/videos/play/wwdc2021/10132/
- Use async/await with URLSession https://developer.apple.com/videos/play/wwdc2021/10095/

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

- Meet async/await in Swift https://developer.apple.com/videos/play/wwdc2021/10132/
- Use async/await with URLSession https://developer.apple.com/videos/play/wwdc2021/10095/
- Explore structured concurrency in Swift https://developer.apple.com/videos/play/wwdc2021/10134/

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

- Meet async/await in Swift https://developer.apple.com/videos/play/wwdc2021/10132/
- Use async/await with URLSession https://developer.apple.com/videos/play/wwdc2021/10095/
- Explore structured concurrency in Swift https://developer.apple.com/videos/play/wwdc2021/10134/
- Meet AsyncSequence https://developer.apple.com/videos/play/wwdc2021/10058/

Recap

`async` / `await` is a language feature to intuitively write and manipulate asynchronous code

There's a good chance that's it going to progressively replace Combine for most apps

But this talk was only a short introduction on the topic, there's a lot more to learn!

And of course there are a lot of WWDC sessions on the topic: here's a selection

- Meet async/await in Swift https://developer.apple.com/videos/play/wwdc2021/10132/
- Use async/await with URLSession https://developer.apple.com/videos/play/wwdc2021/10095/
- Explore structured concurrency in Swift https://developer.apple.com/videos/play/wwdc2021/10134/
- Meet AsyncSequence https://developer.apple.com/videos/play/wwdc2021/10058/
- Swift concurrency: Update a sample app https://developer.apple.com/videos/play/wwdc2021/10194/

To go further on the topic

How do you go from completionHandler to asyn...

1 k vues • il y a 3 mois

https://www.youtube.com/watch?v=9CI8O7iufDI

You hate DispatchQueue.main.asyn...

1,3 k vues • il y a 2 mois

https://www.youtube.com/watch?v=6qHgFjbp90U

Thank You! 

Twitter

YouTube