async / await + SwiftUI : comment ça marche ?

Vincent Pradeilles (@v_pradeilles) – Worldline 🇫🇷

I’m Vincent 👋 🇫🇷


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

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)

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!");
}
```

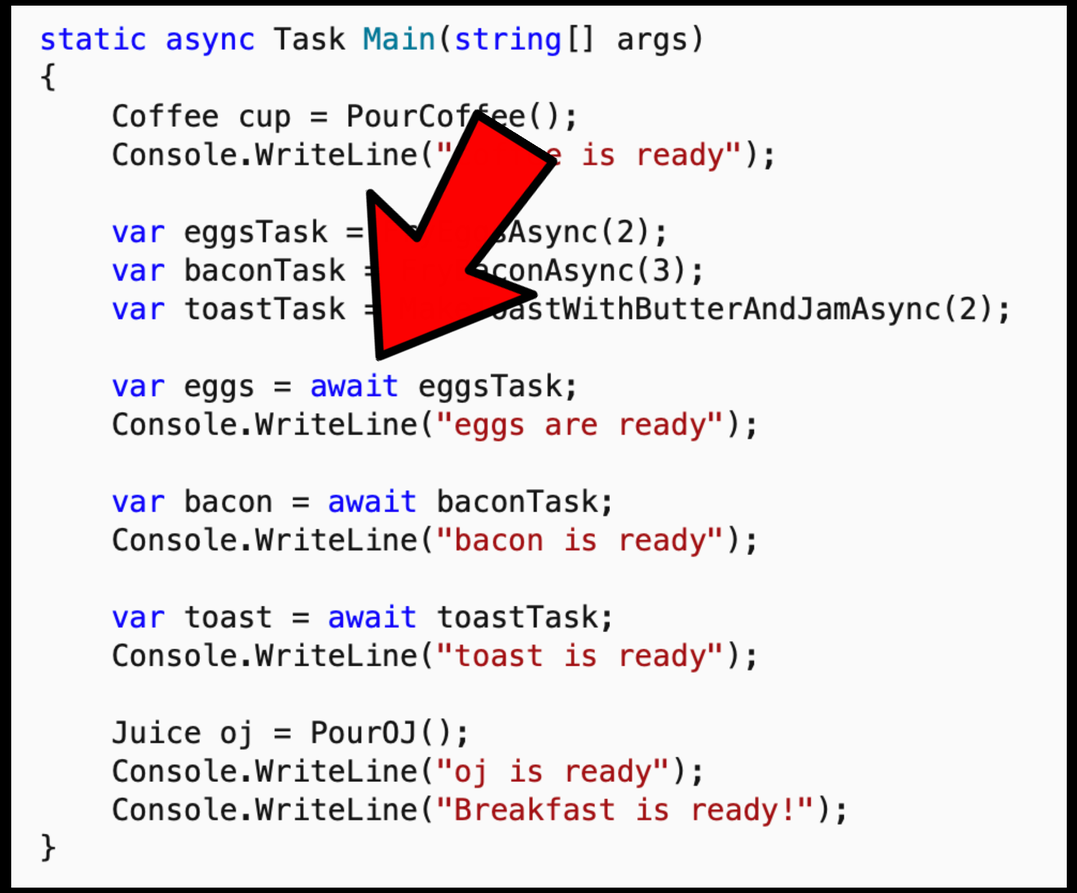
Built-in language support

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!");
}
```

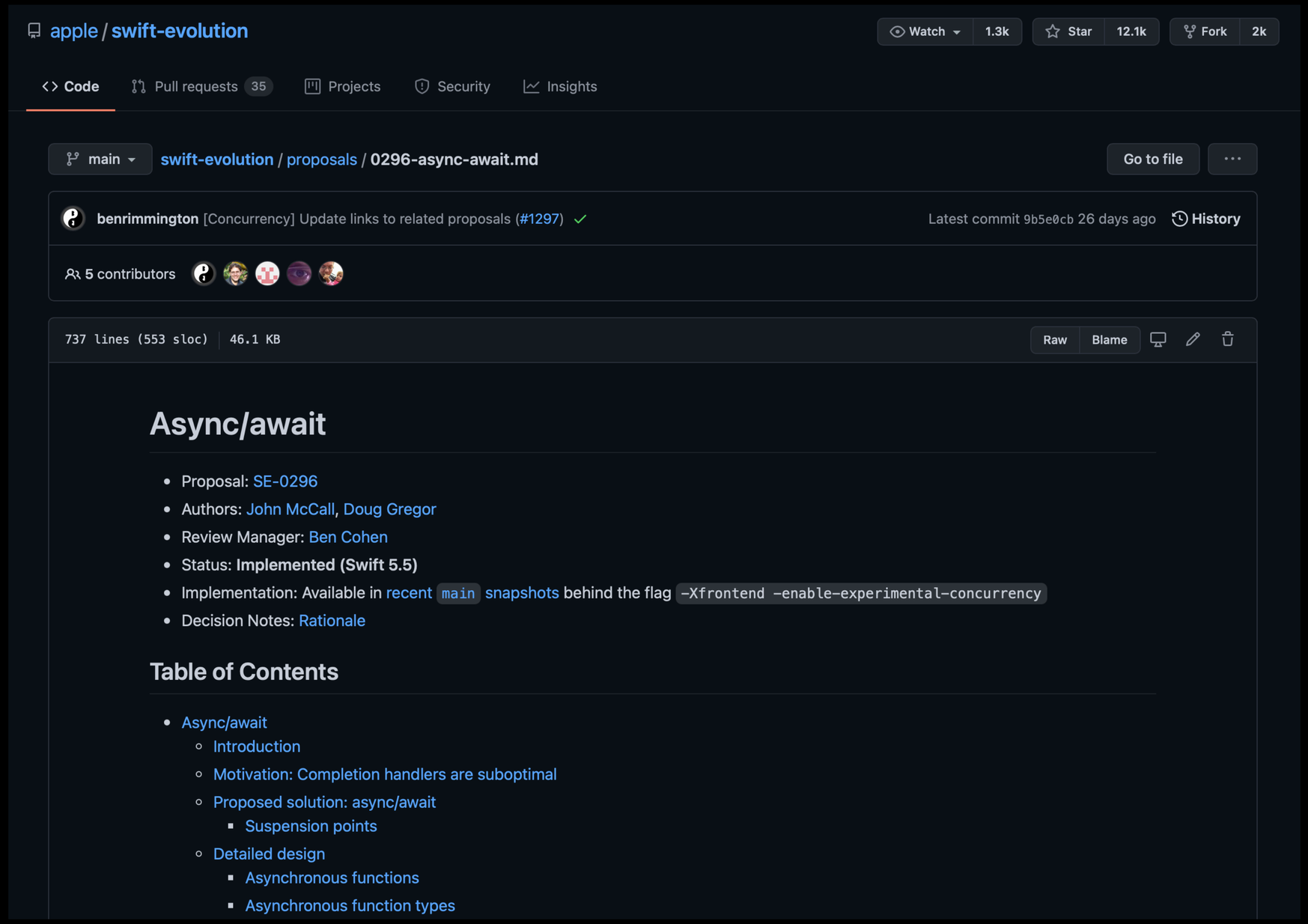
Built-in language support

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

How do we start using it?

Step 1
Download Xcode 13

Xcode

Développeurs

1,1 K NOTES	ÂGE	CLASSEMENT	DÉVELOPPEUR	LANGUE	TAILLE
3,0 ★★★☆☆	4+ Ans	N° 1 Développeurs	Apple	EN Anglais	12,4 Go

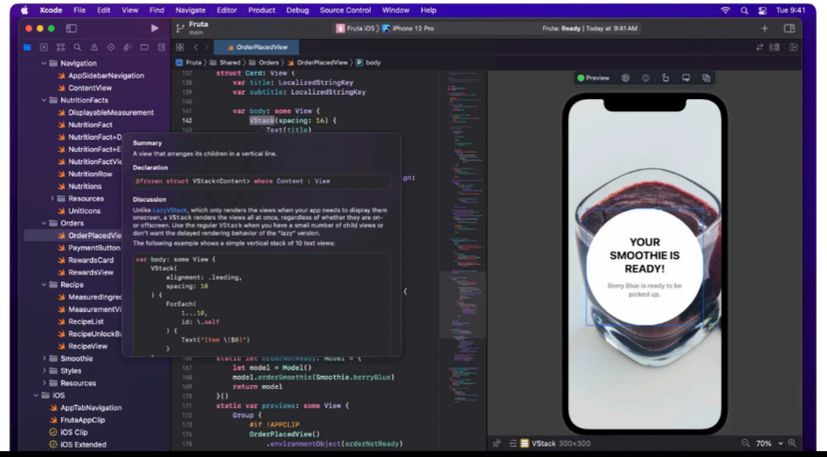
Nouveautés

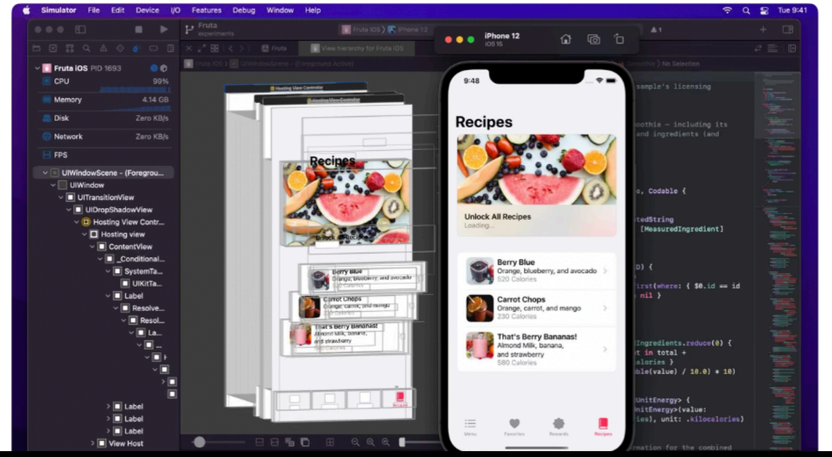
Historique

Xcode 13.1 includes Swift 5.5 and SDKs for iOS 15, iPad OS 15, tvOS 15, watchOS 8, and macOS Monterey. suite

Il y a 1 j
Version 13.1

Aperçu





https://developer.apple.com/download/

Step 2
Set the deployment target to iOS 15

(Yeah, I know that sucks)

Step 3
Time to experiment!

Let’s implement 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)
```

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

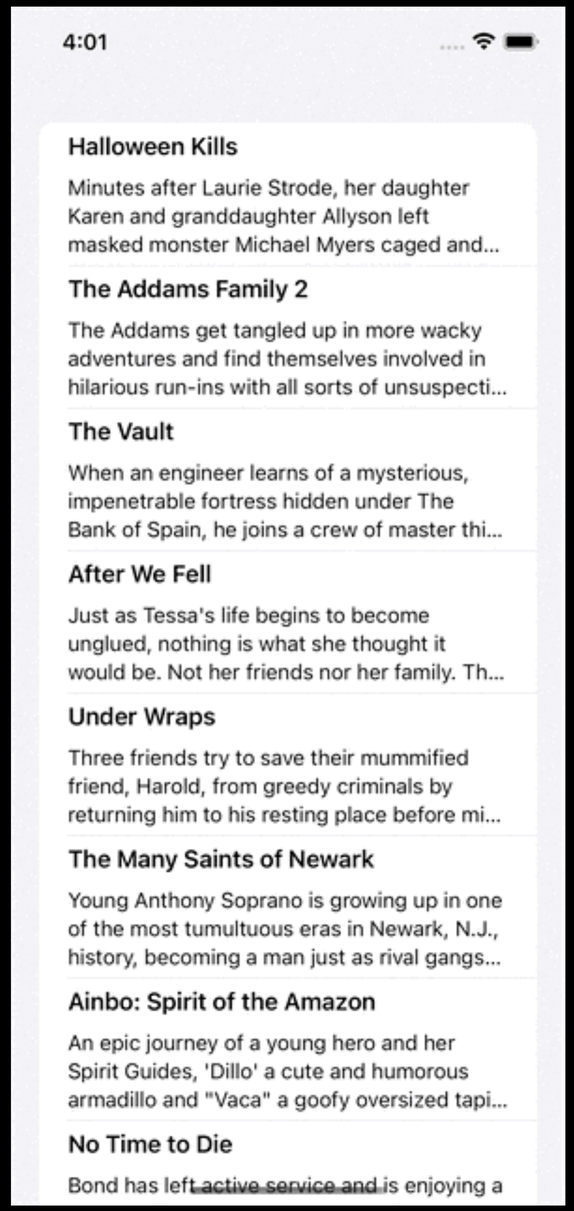
```
struct ContentView: View {

    @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()
        }
    }
}
```




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

But this talk was only an 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

But this talk was only an 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

But this talk was only an 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

But this talk was only an 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

But this talk was only an 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

But this talk was only an 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

But this talk was only an 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 with me

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