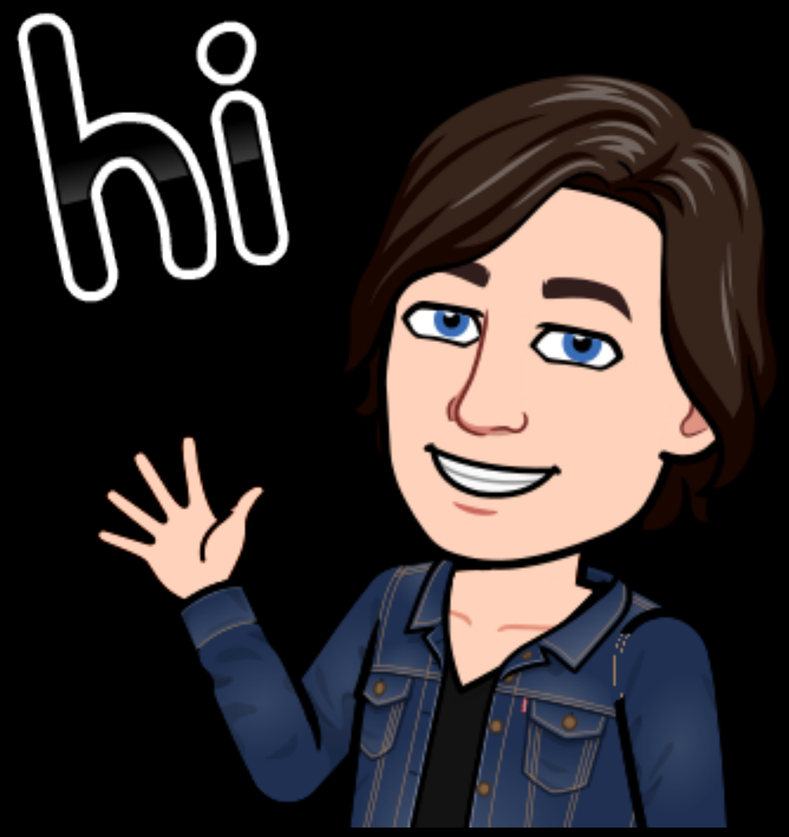
Your first steps with async / await

Vincent Pradeilles (@v_pradeilles) – Worldline 🇫🇷

I’m Vincent 👋 🇫🇷



⚠️

This talk is about a Swift feature that is still in beta

All code examples are subject to deprecations!

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("[illegible]e is ready");

    var eggsTask = [illegible]Async(2);
    var baconTask = [illegible]conAsync(3);
    var toastTask = [illegible]astWithButterAndJamAsync(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 = FryBacon[illegible]sync(3);
    var toastTask = Mak[illegible]tWithButterAndJamAsync(2);

    var eggs = awai[illegible]task;
    Console.WriteLi[illegible] 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.WriteLi  are ready");

    var bacon = awai  nTask;
    Console.WriteLi  n 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!

apple / swift-evolution

Watch 1.3k | Star 12.1k | Fork 2k

Code | Pull requests 35 | Projects | Security | Insights

main | swift-evolution / proposals / 0296-async-await.md | Go to file | ···

benrimmington [Concurrency] Update links to related proposals (#1297) ✓ Latest commit 9b5e0cb 26 days ago History

5 contributors

737 lines (553 sloc) | 46.1 KB | Raw | Blame

Async/await

- Proposal: SE-0296
- Authors: John McCall, Doug Gregor
- Review Manager: Ben Cohen
- Status: **Implemented (Swift 5.5)**
- Implementation: Available in recent `main` snapshots behind the flag `-Xfrontend -enable-experimental-concurrency`
- Decision Notes: Rationale

Table of Contents

- Async/await
  - Introduction
  - Motivation: Completion handlers are suboptimal
  - Proposed solution: async/await
    - Suspension points
  - Detailed design
    - Asynchronous functions
    - Asynchronous function types

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

How do we start using it?

Step 1
Download the latest Xcode beta

Developer Discover Design Develop Distribute Support Account

Downloads

Beta Release Profiles and Logs More

Vincent Pradeilles | Sign Out

Beta Software Downloads

Xcode 13 beta

Download

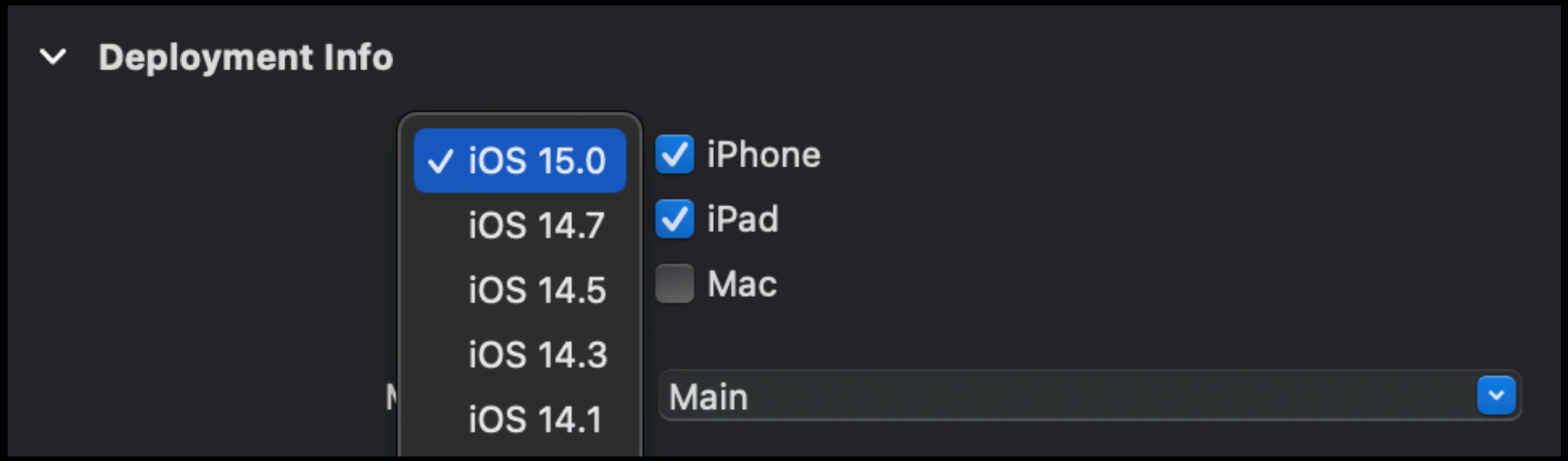
Release Notes ›

Xcode 13 includes everything you need to create amazing apps for all Apple platforms. Includes the latest SDKs for macOS, iOS, watchOS, and tvOS.

Released	Build
June 7, 2021	13A5154h

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 go back to the previous code

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

How about we update it to use async / await?

```swift
typealias CompletionHandler<Output> = (Output) -> Void

func getUserId(_ completion: @escaping CompletionHandler<Int>)
```

```swift
typealias CompletionHandler<Output> = (Output) -> Void

func getUserId(_ completion: @escaping CompletionHandler<Int>)

func getUserId() async -> Int {
    return await withCheckedContinuation { continuation in
        getUserId { userId in
            continuation.resume(returning: userId)
        }
    }
}
```

`await` creates a suspend point: it suspends the execution until the `async` function returns

`await` creates a suspend point: it suspends the execution until the `async` function returns

This little trick allows us to manipulate an asynchronous function just as if it were synchronous!

`await` creates a suspend point: it suspends the execution until the `async` function returns

This little trick allows us to manipulate an asynchronous function just as if it were synchronous!

But there's a tradeoff: `await` can only be used in a function that is itself also `async`

Now let’s wrap the other 2 functions

```swift
func getUserFirstName(userId: Int) async -> String {
    return await withCheckedContinuation { continuation in
        getUserFirstName(userId: userId) { firstName in
            continuation.resume(returning: firstName)
        }
    }
}

func getUserLastName(userId: Int) async -> String {
    return await withCheckedContinuation { continuation in
        getUserLastName(userId: userId) { lastName in
            continuation.resume(returning: lastName)
        }
    }
}
```

And actually call them!

```swift
func greetUser() async {
    let userId = await getUserId()
    let firstName = await getUserFirstName(userId: userId)
    let lastName = await getUserLastName(userId: userId)

    print("Hello \(firstName) \(lastName)")
}
```

But wait, how do we actually call `greetUser()`?

Remember, an `async` function can only be called from another `async` function.

Remember, an `async` function can only be called from another `async` function.

But typical entry points, like `application(_:didFinishLaunchingWithOptions:)` or `viewDidLoad()`, are not `async`!

```
override func viewDidLoad() {
    super.viewDidLoad()

    await greetUser() ❌
}
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    async {
        await greetUser() ✅
    }
}
```

In this snippet of code, `async` is a global function

In this snippet of code, `async` is a global function

It allows us to create a context in which we are allowed to call `async` functions.

In this snippet of code, `async` is a global function

It allows us to create a context in which we are allowed to call `async` functions.

You can think of it as dispatching code on a background thread.

In this snippet of code, `async` is a global function

It allows us to create a context in which we are allowed to call `async` functions.

You can think of it as dispatching code on a background thread.

As a consequence of that, this `async` context won’t be able to return any value, it can only perform side effects.

```swift
func greetUser() async {
    let userId = await getUserId()
    let firstName = await getUserFirstName(userId: userId)
    let lastName = await getUserLastName(userId: userId)

    print("Hello \(firstName) \(lastName)")
}
```

Can we sill improve? Yes!

```swift
func greetUser() async {
    let userId = await getUserId()
    let firstName = await getUserFirstName(userId: userId)
    let lastName = await getUserLastName(userId: userId)

    print("Hello \(firstName) \(lastName)")
}
```

```swift
func greetUser() async {
    let userId = await getUserId()
    async let firstName = getUserFirstName(userId: userId)
    async let lastName = getUserLastName(userId: userId)

    await print("Hello \(firstName) \(lastName)")
}
```

Using `async let`, we implicitly create a task hierarchy.

Using `async let`, we implicitly create a task hierarchy.

Then, using `await` we execute this hierarchy and then use the result the tasks yielded.

Let’s compare with Combine

```swift
func greetUser() async {
    let userId = await getUserId()
    async let firstName = getUserFirstName(userId: userId)
    async let lastName = getUserLastName(userId: userId)

    await print("Hello \(firstName) \(lastName)")
}
```

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

Is async / await better than Combine?

No!

Combine is a much more general purpose tool

On the other hand, async / await is very focused on a single use case

Let’s dig a bit deeper…

…to understand what happens behind the magic 🪄✨

What if we need to cancel a task?

Then we're going to have to explicitly create our task!

```
func greetUser() async {
    let userId = await getUserId()
    async let firstName = getUserFirstName(userId: userId)
    async let lastName = getUserLastName(userId: userId)

    await print("Hello \(firstName) \(lastName)")
}
```

```swift
func greetUser() async {
    let userId = await getUserId()
    async let firstName = getUserFirstName(userId: userId)
    async let lastName = getUserLastName(userId: userId)

    await print("Hello \(firstName) \(lastName)")
}
```

```swift
func greetUser() async {
    let userId = await getUserId()

    async let firstName = getUserFirstName(userId: userId)

    let lastNameHandle: Task.Handle<String, Error> =
    asyncDetached(priority: .userInitiated) {
        return await getUserLastName(userId: userId)
    }

    await print("Hello \(firstName) \(lastName)")
}
```

```swift
func greetUser() async {
    let userId = await getUserId()

    async let firstName = getUserFirstName(userId: userId)

    let lastNameHandle: Task.Handle<String, Error> =
    asyncDetached(priority: .userInitiated) {
        return await getUserLastName(userId: userId)
    }

    do {
        await print("Hello again \(firstName) \(try lastNameHandle.get())")
    } catch {
        print(error)
    }
}
```

```swift
func greetUser() async {
    let userId = await getUserId()

    async let firstName = getUserFirstName(userId: userId)

    let lastNameHandle: Task.Handle<String, Error> =
    asyncDetached(priority: .userInitiated) {
        return await getUserLastName(userId: userId)
    }

    if Bool.random() { lastNameHandle.cancel() }

    do {
        await print("Hello again \(firstName) \(try lastNameHandle.get())")
    } catch {
        print(error)
    }
}
```

```swift
func greetUser() async {
    let userId = await getUserId()

    async let firstName = getUserFirstName(userId: userId)

    let lastNameHandle: Task.Handle<String, Error> =
    asyncDetached(priority: .userInitiated) {
        try Task.checkCancellation()

        return await getUserLastName(userId: userId)
    }

    if Bool.random() { lastNameHandle.cancel() }

    do {
        await print("Hello again \(firstName) \(try lastNameHandle.get())")
    } catch {
        print(error)
    }
}
```

Now we see a little bit clearer!

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/

Thank You! 😊

Twitter

YouTube