SwiftUI in UIKit: what's a pragmatic approach?

Vincent Pradeilles 🇫🇷 (@v_pradeilles) – PhotoRoom

Disclaimer

Disclaimer

Disclaimer

- There’s no one-size-fits-all solution: every app is different!

Disclaimer

- There’s no one-size-fits-all solution: every app is different!
- My goal with this talk is to share what worked and didn’t work for me

Disclaimer

- There’s no one-size-fits-all solution: every app is different!
- My goal with this talk is to share what worked and didn’t work for me
- Best-case scenario: you can reuse directly

Disclaimer

- There’s no one-size-fits-all solution: every app is different!
- My goal with this talk is to share what worked and didn’t work for me
- Best-case scenario: you can reuse directly
- Worst-case scenario: it’s food for thought

Why use SwiftUI in a UIKit app?

Why use SwiftUI in a UIKit app?

Why use SwiftUI in a UIKit app?

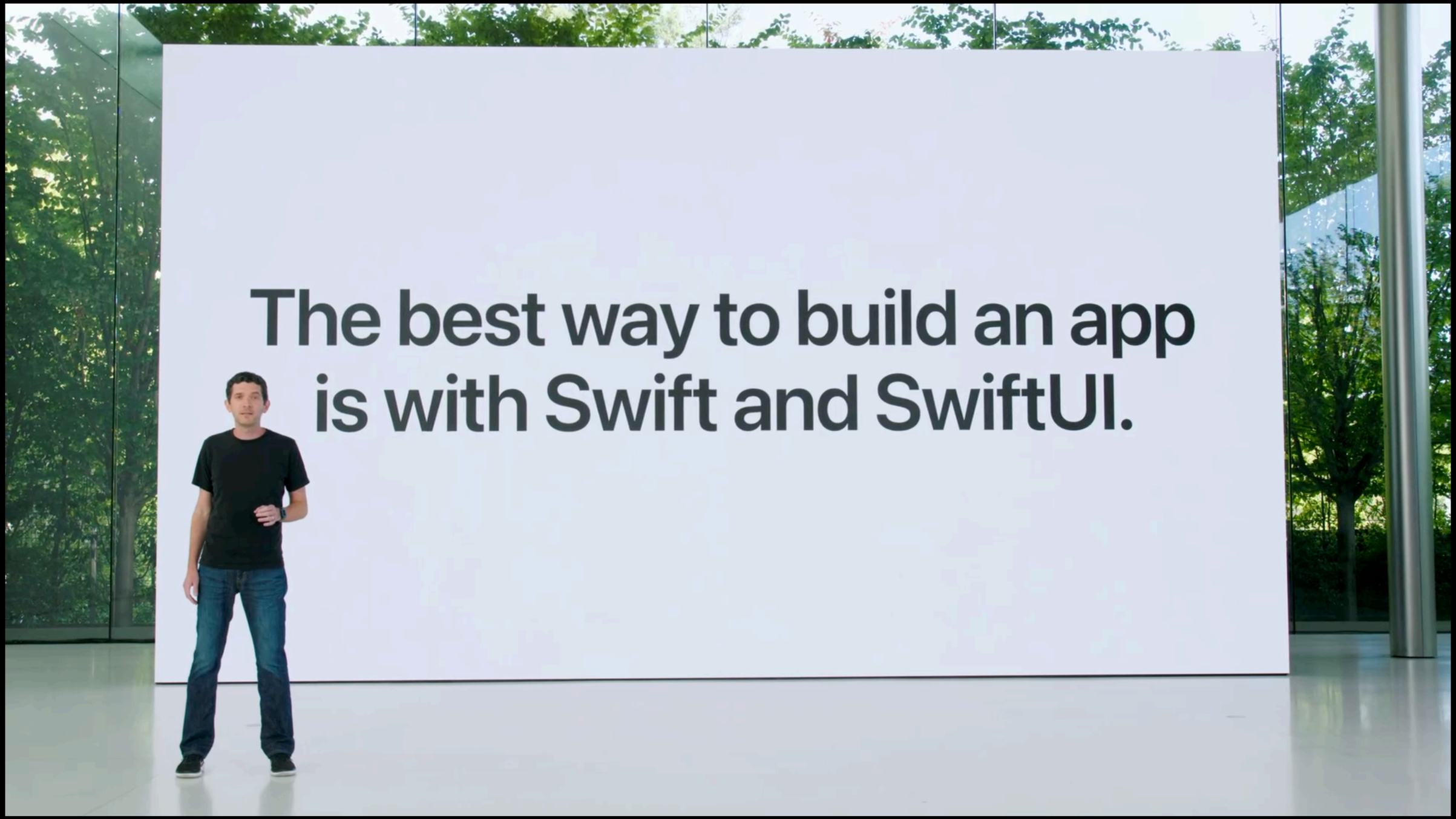
- It’s the future

Why use SwiftUI in a UIKit app?

- It’s the future
- It’s an exciting piece of tech

Why use SwiftUI in a UIKit app?

- It’s the future
- It’s an exciting piece of tech
- Everyone else is doing it


The best way to build an app
is with Swift and SwiftUI.

Why use SwiftUI in a UIKit app?

- It’s the future
- It’s an exciting piece of tech
- Everyone else is doing it

Why use SwiftUI in a UIKit app?

- It’s the future
- It’s an exciting piece of tech
- Everyone else is doing it
- To implement features faster

Why use SwiftUI in a UIKit app?

- It’s the future
- It’s an exciting piece of tech
- Everyone else is doing it
- To implement features faster

“To implement features faster”

“To implement features faster”

- Using SwiftUI in a UIKit app is all about making the right tradeoffs

“To implement features faster”

- Using SwiftUI in a UIKit app is all about making the right tradeoffs
- You need a north star that will guide you through these decisions

“To implement features faster”

- Using SwiftUI in a UIKit app is all about making the right tradeoffs
- You need a north star that will guide you through these decisions
- (especially when you’re working in a team!)

“To implement features faster”

- Using SwiftUI in a UIKit app is all about making the right tradeoffs
- You need a north star that will guide you through these decisions
- (especially when you’re working in a team!)
- I’ve found that this is a pretty good north star to follow!

When SwiftUI really shines

When SwiftUI really shines

When SwiftUI really shines

- I want to start by sharing the first time I was truly impressed by SwiftUI

When SwiftUI really shines

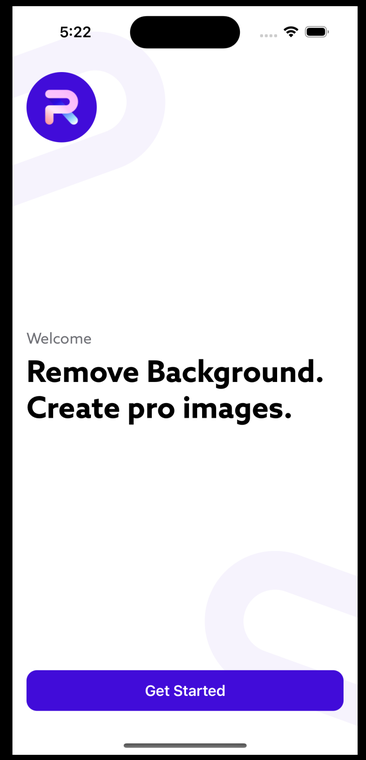
- I want to start by sharing the first time I was truly impressed by SwiftUI
- The problem to solve: quickly iterating over new onboarding screens

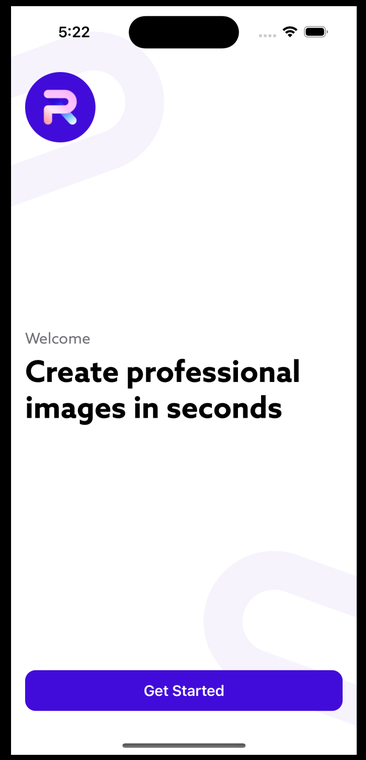
When SwiftUI really shines

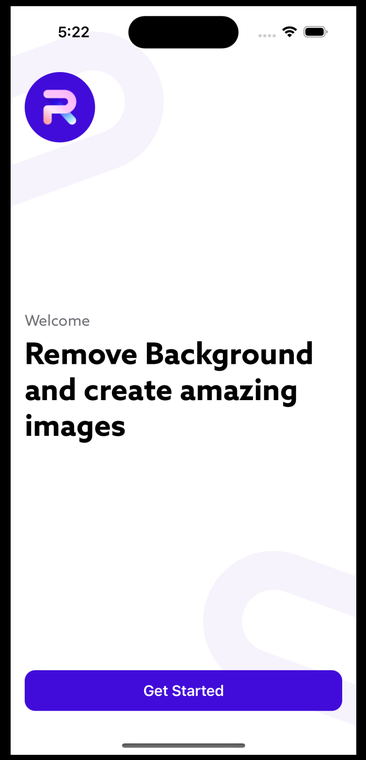
- I want to start by sharing the first time I was truly impressed by SwiftUI
- The problem to solve: quickly iterating over new onboarding screens
- If you like to A / B test features, SwiftUI can make it very easy to test several variants of a same view

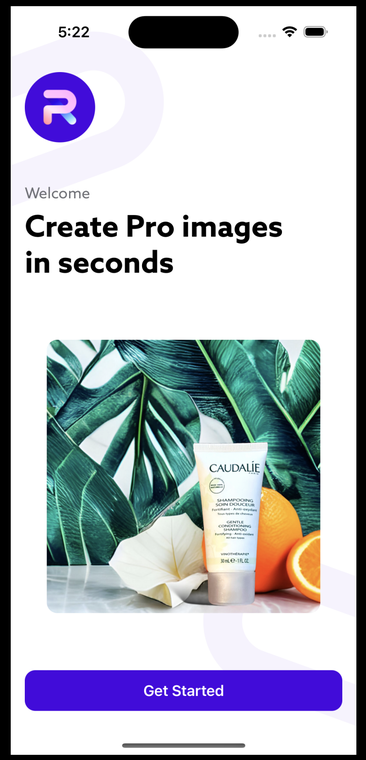
When SwiftUI really shines

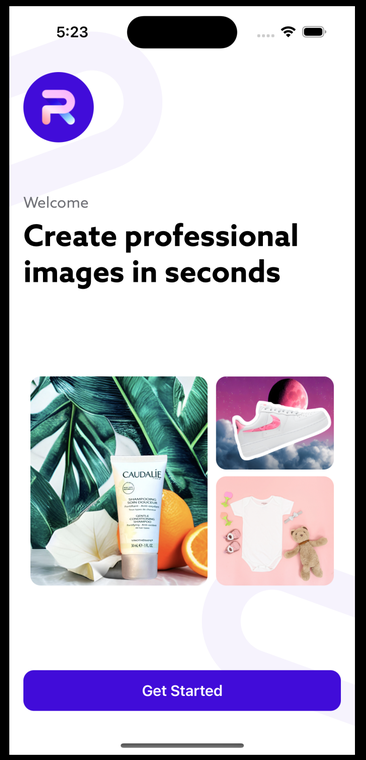
- I want to start by sharing the first time I was truly impressed by SwiftUI
- The problem to solve: quickly iterating over new onboarding screens
- If you like to A / B test features, SwiftUI can make it very easy to test several variants of a same view
- Let me show how!

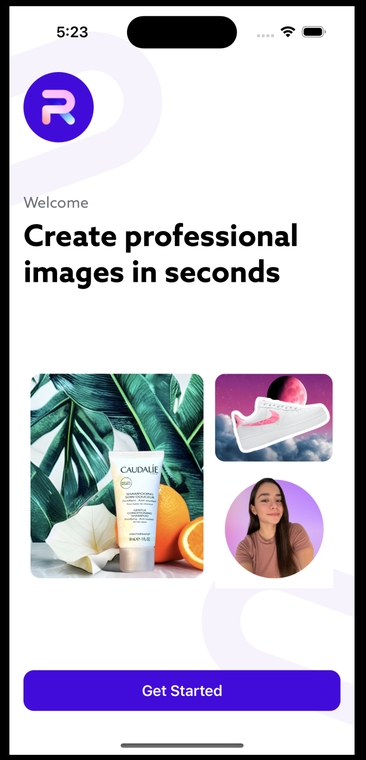












This is the first screen of our app: we wanted to test 6 variants over a week

```
struct OnboardingStartView: View {

}
```

```swift
struct OnboardingStartView: View {

}
```

```swift
enum Variant: String, CaseIterable {
    case optionA
    case optionB
    case optionC
    case optionD
    case optionE
    case optionF
}
```

```swift
struct OnboardingStartView: View {

    let variant: Variant

}
```

```swift
enum Variant: String, CaseIterable {
    case optionA
    case optionB
    case optionC
    case optionD
    case optionE
    case optionF
}
```

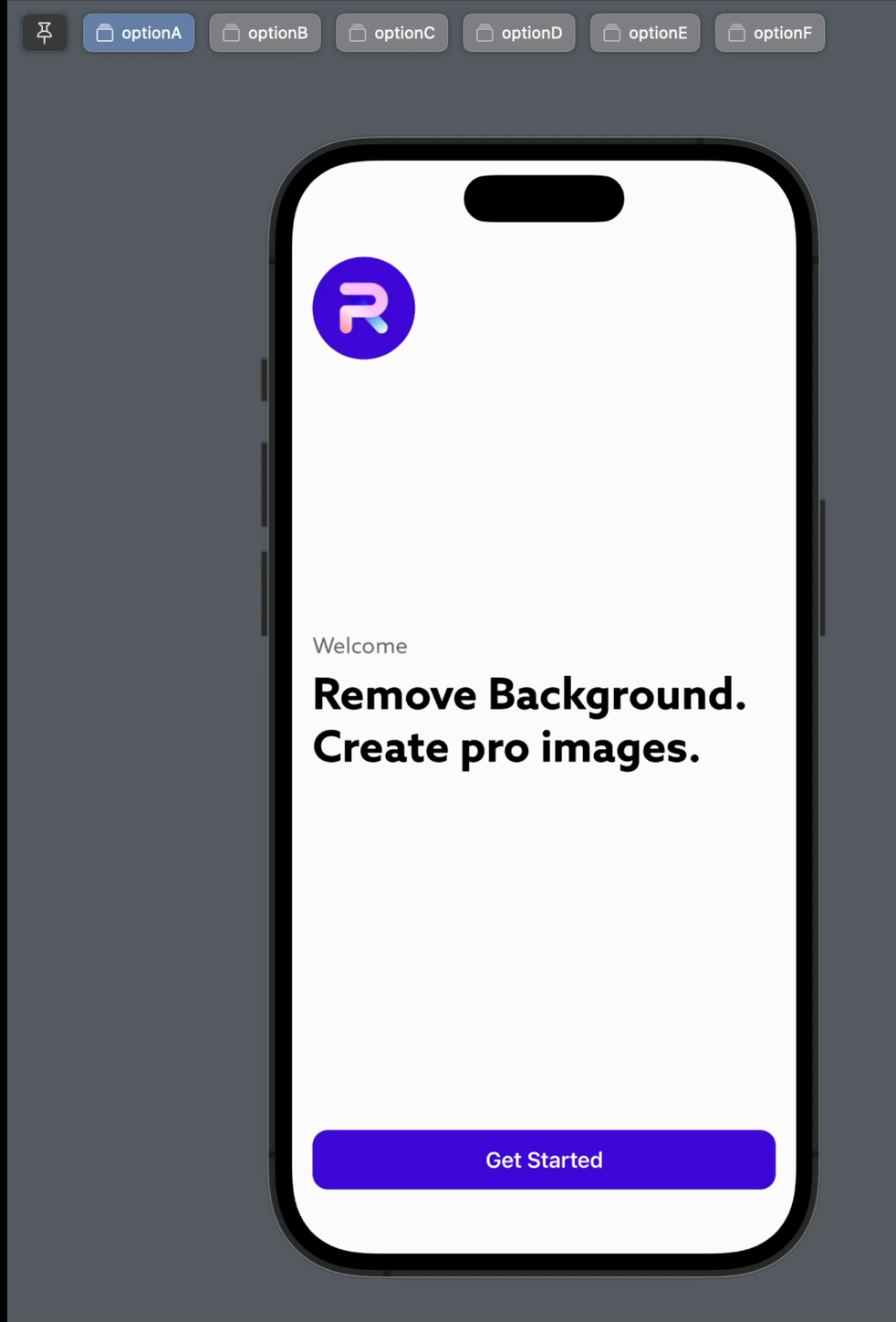
```swift
struct OnboardingStartView: View {

    let variant: Variant

    var body: some View {
        switch variant {
        case .optionA:
            bodyForOptionA
        case .optionB:
            bodyForOptionB
        case .optionC:
            bodyForOptionC
        case .optionD:
            bodyForOptionD
        case .optionE:
            bodyForOptionE
        case .optionF:
            bodyForOptionF
        }
    }
}
```

```swift
enum Variant: String, CaseIterable {
    case optionA
    case optionB
    case optionC
    case optionD
    case optionE
    case optionF
}
```

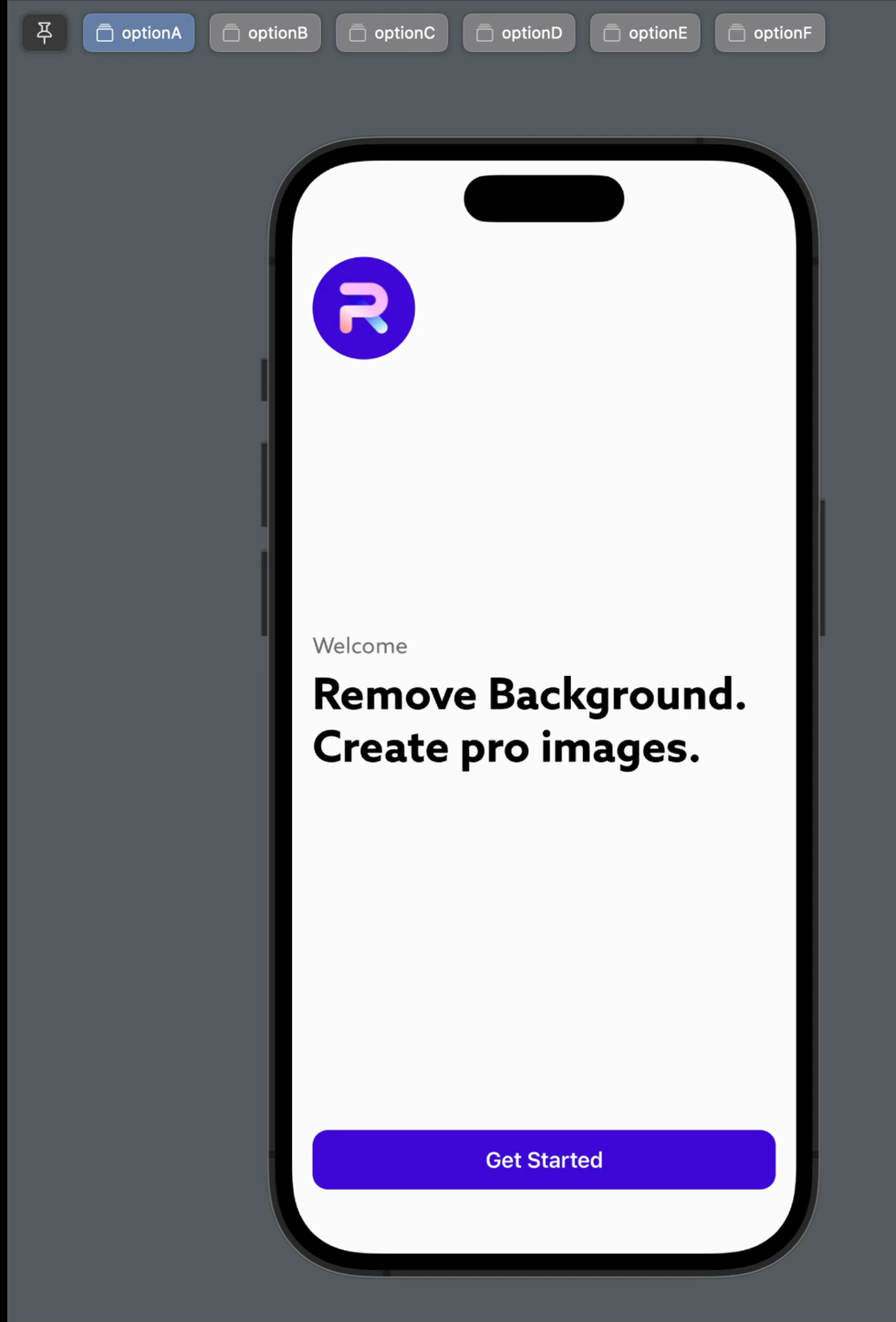
```swift
extension OnboardingStartView {
    var bodyForOptionA: some View {
        /* ... */
    }
    var bodyForOptionB: some View {
        /* ... */
    }
    var bodyForOptionC: some View {
        /* ... */
    }
    var bodyForOptionD: some View {
        /* ... */
    }
    var bodyForOptionE: some View {
        /* ... */
    }
    var bodyForOptionF: some View {
        /* ... */
    }
}
```

```swift
struct OnboardingStartView_Previews: PreviewProvider {
    static var previews: some View {
        Group {
            ForEach(OnboardingStartView.Variant.allCases, id: \.self) { variant in
                OnboardingStartView(variant: variant)
                    .previewDisplayName(variant.rawValue)
            }
        }
    }
}
```

```swift
struct OnboardingStartView_Previews: PreviewProvider {
    static var previews: some View {
        Group {
            ForEach(OnboardingStartView.Variant.allCases, id: \.self) { variant in
                OnboardingStartView(variant: variant)
                    .previewDisplayName(variant.rawValue)
            }
        }
    }
}
```



```swift
struct OnboardingStartView_Previews: PreviewProvider {
    static var previews: some View {
        Group {
            ForEach(OnboardingStartView.Variant.allCases, id: \.self) { variant in
                OnboardingStartView(variant: variant)
                    .previewDisplayName(variant.rawValue)
            }
        }
    }
}
```



When SwiftUI really shines

When SwiftUI really shines

- Of course, we could have achieved the same goal using UIKit…

When SwiftUI really shines

- Of course, we could have achieved the same goal using UIKit…
- …but the speed at which it can be done using SwiftUI is pretty impressive!

When can you (realistically) use SwiftUI in a UIKit app?

When can you (realistically) use SwiftUI in a UIKit app?

When can you (realistically) use SwiftUI in a UIKit app?

- SwiftUI requires iOS 13 or higher

When can you (realistically) use SwiftUI in a UIKit app?

- SwiftUI requires iOS 13 or higher

When can you (realistically) use SwiftUI in a UIKit app?

- SwiftUI requires iOS 13 or higher

When can you (realistically) use SwiftUI in a UIKit app?

- SwiftUI requires iOS 13 or higher
- But I would argue it becomes really usable with iOS 14

When can you (realistically) use SwiftUI in a UIKit app?

- SwiftUI requires iOS 13 or higher
- But I would argue it becomes really usable with iOS 14
- However iOS 14 is not without issues, iOS 15+ makes it really confortable

When can you (realistically) use SwiftUI in a UIKit app?

When can you (realistically) use SwiftUI in a UIKit app?

- In short:

When can you (realistically) use SwiftUI in a UIKit app?

- In short:
  - SwiftUI on iOS 13+ can help for simple views, but can be risky at scale: be cautious

When can you (realistically) use SwiftUI in a UIKit app?

- In short:
  - SwiftUI on iOS 13+ can help for simple views, but can be risky at scale: be cautious
  - SwiftUI on iOS 14+ works pretty well, but expect some bugs specific to iOS 14

When can you (realistically) use SwiftUI in a UIKit app?

- In short:
  - SwiftUI on iOS 13+ can help for simple views, but can be risky at scale: be cautious
  - SwiftUI on iOS 14+ works pretty well, but expect some bugs specific to iOS 14
  - SwiftUI on iOS 15+ should be pretty seamless

What are the blockers to start using SwiftUI in a UIKit app?

What are the blockers to start using SwiftUI in a UIKit app?

What are the blockers to start using SwiftUI in a UIKit app?

- SwiftUI still doesn’t offer all the built-in views you could need

What are the blockers to start using SwiftUI in a UIKit app?

- SwiftUI still doesn’t offer all the built-in views you could need
- What if later I need a subview that is UIKit only?

What are the blockers to start using SwiftUI in a UIKit app?

- SwiftUI still doesn’t offer all the built-in views you could need
- What if later I need a subview that is UIKit only?
- Like a web view, for instance!

```
struct WebView {

}
```

```
struct WebView: UIViewRepresentable {

}
```

```
struct WebView: UIViewRepresentable {
    func makeUIView(context: Context) -> WKWebView {

    }

    func updateUIView(_ webView: WKWebView, context: Context) {

    }
}
```

```
struct WebView: UIViewRepresentable {
    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {

    }
}
```

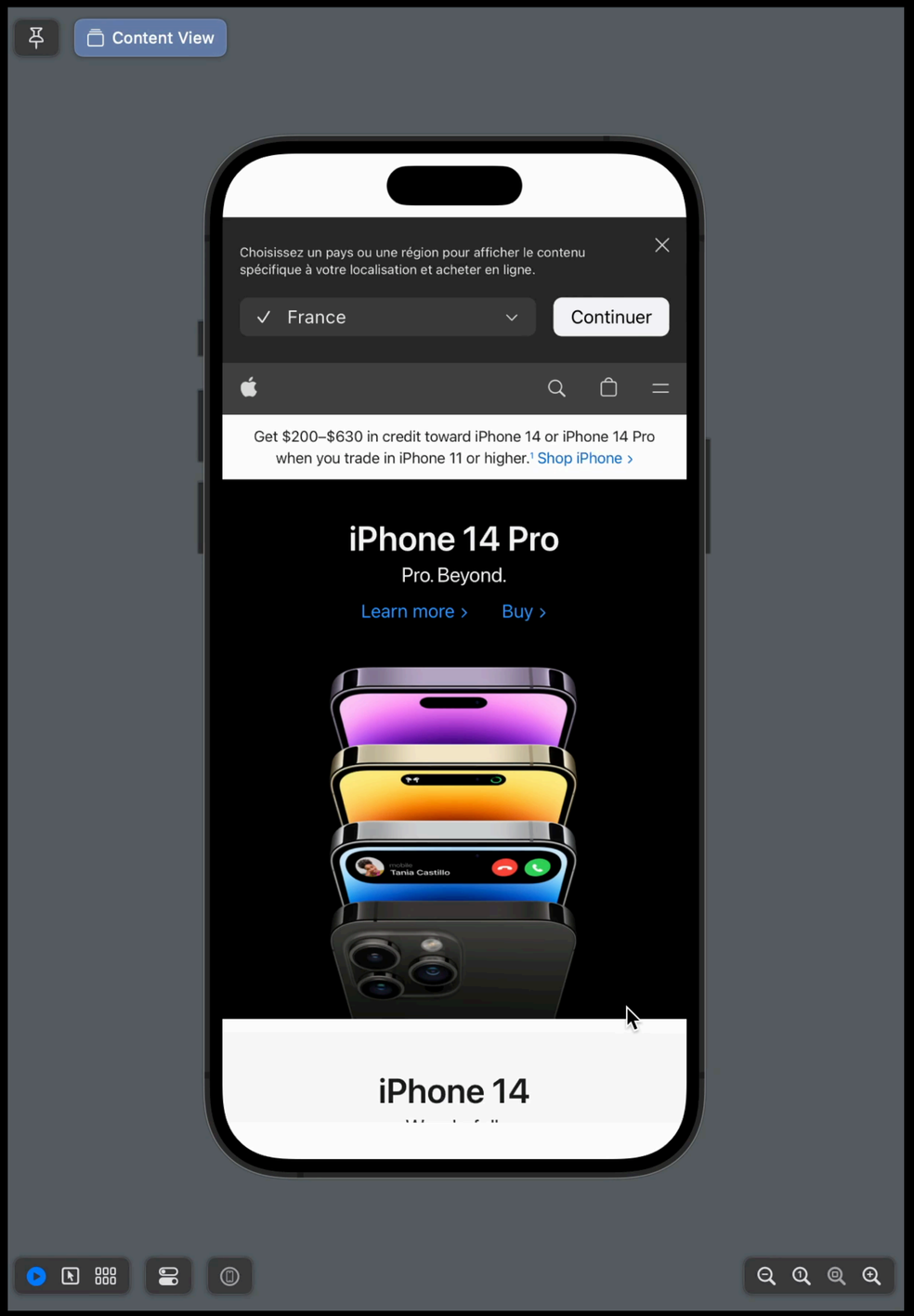
```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {

    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }
}
```

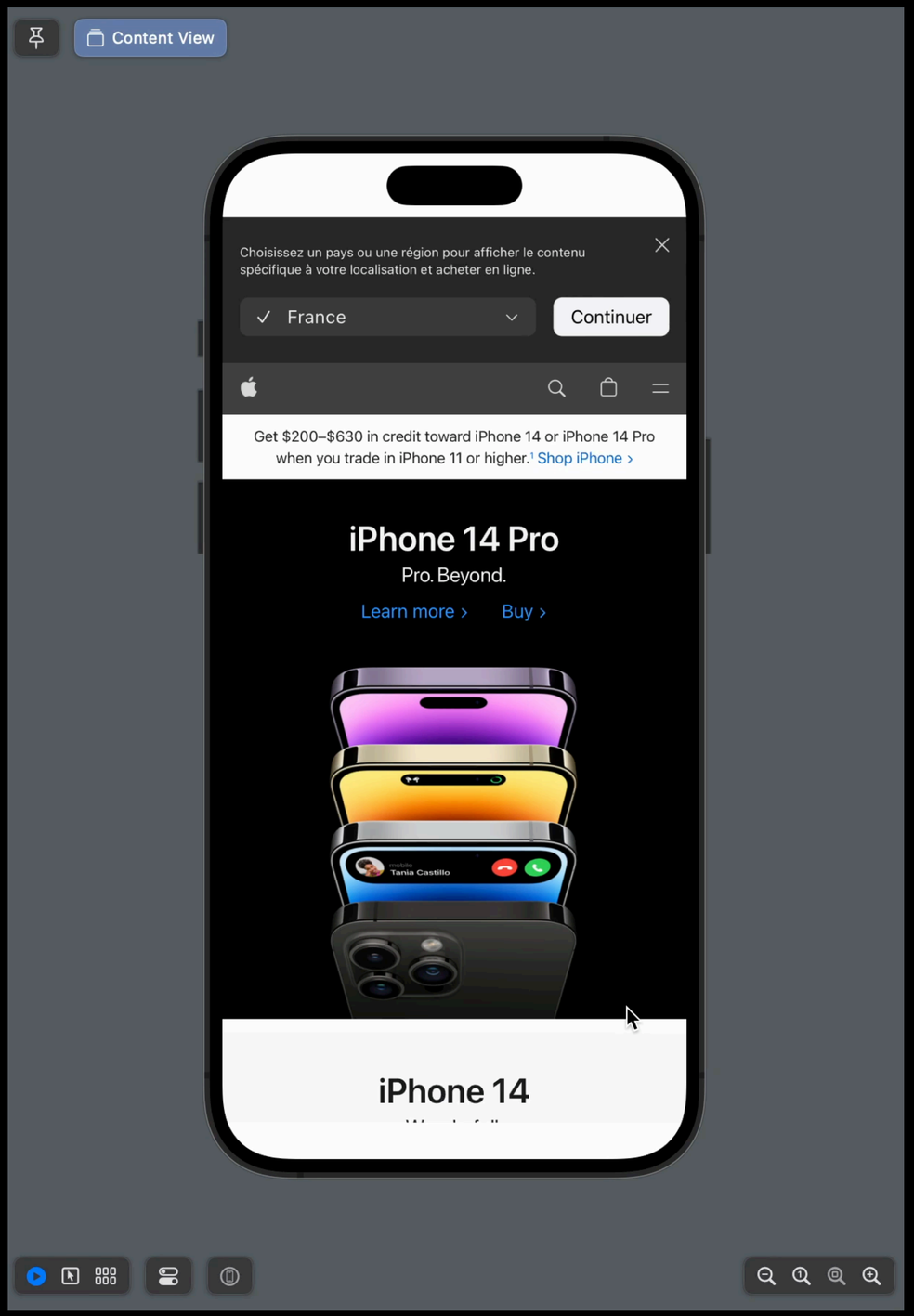
```
struct ContentView: View {
    var body: some View {

    }
}
```

```
struct ContentView: View {
    var body: some View {
        WebView(url: URL(string: "https://www.apple.com")!)
    }
}
```

```swift
struct ContentView: View {
    var body: some View {
        WebView(url: URL(string: "https://www.apple.com")!)
    }
}
```



```swift
struct ContentView: View {
    var body: some View {
        WebView(url: URL(string: "https://www.apple.com")!)
    }
}
```



```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject {

    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject, WKNavigationDelegate {

    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {

        }
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {

        }
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        var parent: WebView

        init(_ parent: WebView) {
            self.parent = parent
        }

        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {

        }
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        var parent: WebView

        init(_ parent: WebView) {
            self.parent = parent
        }

        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {
            decisionHandler(parent.navigationPolicy?(navigationAction) ?? .allow)
        }
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    func makeUIView(context: Context) -> WKWebView {
        return WKWebView()
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        var parent: WebView

        init(_ parent: WebView) {
            self.parent = parent
        }

        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {
            decisionHandler(parent.navigationPolicy?(navigationAction) ?? .allow)
        }
    }
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    func makeUIView(context: Context) -> WKWebView {
        let webView = WKWebView()
        webView.navigationDelegate = context.coordinator
        return webView
    }

    func updateUIView(_ webView: WKWebView, context: Context) {
        webView.load(URLRequest(url: url))
    }

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    class Coordinator: NSObject, WKNavigationDelegate {
        var parent: WebView

        init(_ parent: WebView) {
            self.parent = parent
        }

        func webView(
            _ webView: WKWebView,
            decidePolicyFor navigationAction: WKNavigationAction,
            decisionHandler: @escaping (WKNavigationActionPolicy) -> Void
        ) {
            decisionHandler(parent.navigationPolicy?(navigationAction) ?? .allow)
        }
    }
}
```

```
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    /* ... */
}
```

```swift
struct WebView: UIViewRepresentable {

    let url: URL
    var navigationPolicy: ((WKNavigationAction) -> WKNavigationActionPolicy)? = nil

    /* ... */
}

extension WebView {
    func set(navigationPolicy: @escaping (WKNavigationAction) -> WKNavigationActionPolicy) -> WebView {
        var copy = self
        copy.navigationPolicy = navigationPolicy
        return copy
    }
}
```

```
struct ContentView: View {
    var body: some View {
        WebView(url: URL(string: "https://www.apple.com")!)
    }
}
```

```swift
struct ContentView: View {
    var body: some View {
        WebView(url: URL(string: "https://www.apple.com")!)
            .set(navigationPolicy: { navigationAction in
                navigationAction.navigationType == .linkActivated ? .cancel : .allow
            })
    }
}
```

What are the blockers to start using SwiftUI in a UIKit app?

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app
- Everything is implemented in the Views…

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app
- Everything is implemented in the Views…
- …it’s Massive View Controllers all over again

“Let's quickly add a bit of SwiftUI to our UIKit app, what could go wrong?”

“Let's quickly add a bit of SwiftUI to our UIKit app, what could go wrong?”

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app
- Everything is implemented in the Views…
- …it’s Massive View Controllers all over again

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app
- Everything is implemented in the Views…
- …it’s Massive View Controllers all over again
- That doesn’t have to be true!

What are the blockers to start using SwiftUI in a UIKit app?

- Out-of-the box SwiftUI can feel like a regression in terms of architecture compared to a mature UIKit app
- Everything is implemented in the Views…
- …it’s Massive View Controllers all over again
- That doesn’t have to be true!
- Let’s see how to implement a popular pattern like the Coordinator in SwiftUI

Implementing the Coordinator pattern in SwiftUI

Implementing the Coordinator pattern in SwiftUI

Implementing the Coordinator pattern in SwiftUI

- In UIKit, coordinators are relatively straightforward to implement

Implementing the Coordinator pattern in SwiftUI

- In UIKit, coordinators are relatively straightforward to implement
  - You store references to view controllers…

Implementing the Coordinator pattern in SwiftUI

- In UIKit, coordinators are relatively straightforward to implement
  - You store references to view controllers…
  - …and then you call methods on these references

```swift
import UIKit

class Coordinator {

    var navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let vc = ViewControllerProvider.moviesViewController
        vc.coordinator = self
        navigationController.pushViewController(vc, animated: false)
    }

    func displayDetails(of movie: Movie) {
        let detailsVC = ViewControllerProvider.movieDetailsController(for: movie)
        detailsVC.coordinator = self
        navigationController.pushViewController(detailsVC, animated: true)
    }
}
```

```swift
import UIKit

class Coordinator {

    var navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let vc = ViewControllerProvider.moviesViewController
        vc.coordinator = self
        navigationController.pushViewController(vc, animated: false)
    }

    func displayDetails(of movie: Movie) {
        let detailsVC = ViewControllerProvider.movieDetailsController(for: movie)
        detailsVC.coordinator = self
        navigationController.pushViewController(detailsVC, animated: true)
    }
}
```

```swift
import UIKit

class Coordinator {

    var navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let vc = ViewControllerProvider.moviesViewController
        vc.coordinator = self
        navigationController.pushViewController(vc, animated: false)
    }

    func displayDetails(of movie: Movie) {
        let detailsVC = ViewControllerProvider.movieDetailsController(for: movie)
        detailsVC.coordinator = self
        navigationController.pushViewController(detailsVC, animated: true)
    }
}
```

```swift
import UIKit

class Coordinator {

    var navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let vc = ViewControllerProvider.moviesViewController
        vc.coordinator = self
        navigationController.pushViewController(vc, animated: false)
    }

    func displayDetails(of movie: Movie) {
        let detailsVC = ViewControllerProvider.movieDetailsController(for: movie)
        detailsVC.coordinator = self
        navigationController.pushViewController(detailsVC, animated: true)
    }
}
```

Implementing the Coordinator pattern in SwiftUI

- In UIKit, coordinators are relatively straightforward to implement
  - You store references to view controllers…
  - …and you call methods on these references

Implementing the Coordinator pattern in SwiftUI

- In UIKit, coordinators are relatively straightforward to implement
  - You store references to view controllers…
  - …and you call methods on these references
- In SwiftUI, you basically need to take the opposite approach!

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    var body: some View {
        TabView {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    var body: some View {
        TabView {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {

}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    var body: some View {
        TabView {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

We've implemented a simple Coordinator!

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

```swift
import SwiftUI

@main
struct SwiftUI_CoordinatorApp: App {

    let coordinatorViewModel = CoordinatorViewModel()

    var body: some Scene {
        WindowGroup {
            CoordinatorView(viewModel: coordinatorViewModel)
                .onAppear {
                    DispatchQueue.main.asyncAfter(deadline: .now() + 2) {
                        coordinatorViewModel.currentRoute = .second
                    }
                }
        }
    }
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
        .fullScreenCover(isPresented: $viewModel.showAuthent) {
            AuthenticationView()
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
        .fullScreenCover(isPresented: $viewModel.showAuthent) {
            AuthenticationView()
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
}
```

```swift
enum TabRoutes {
    case first
    case second
}

struct CoordinatorView: View {

    @ObservedObject var viewModel: CoordinatorViewModel

    var body: some View {
        TabView(selection: $viewModel.currentRoute) {
           FirstFeatureView()
           .tabItem {
                Image(systemName: "1.circle.fill")
                Text("First Feature")
            }
            .tag(TabRoutes.first)

            SecondFeatureView()
                .tabItem {
                    Image(systemName: "2.circle.fill")
                    Text("Second Feature")
                }
                .tag(TabRoutes.second)
        }
        .sheet(item: $viewModel.modalContent) { modalContent in
            ModalView(content: modalContent)
        }
        .fullScreenCover(isPresented: $viewModel.showAuthent) {
            AuthenticationView()
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
}
```

Implementing the Coordinator pattern in SwiftUI

Implementing the Coordinator pattern in SwiftUI

- We implemented a Coordinator for tab and modal…

Implementing the Coordinator pattern in SwiftUI

- We implemented a Coordinator for tab and modal…
- …but these were the easy ones!

Implementing the Coordinator pattern in SwiftUI

- We implemented a Coordinator for tab and modal…
- …but these were the easy ones!
- We “only” had to wrap existing SwiftUI APIs

Implementing the Coordinator pattern in SwiftUI

- We implemented a Coordinator for tab and modal…
- …but these were the easy ones!
- We “only” had to wrap existing SwiftUI APIs
- How about the navigation?

Implementing the Coordinator pattern in SwiftUI

- We implemented a Coordinator for tab and modal…
- …but these were the easy ones!
- We “only” had to wrap existing SwiftUI APIs
- How about the navigation?
- (and we can’t use `NavigationStack`, because it’s iOS 16+)

```swift
struct FirstFeatureView: View {

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
struct FirstFeatureView: View {

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```
struct NavigationModifier: ViewModifier {

    func body(content: Content) -> some View {
        content
    }
}
```

```
struct NavigationModifier: ViewModifier {

    @Binding var isActive: Bool

    func body(content: Content) -> some View {
        content
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {

            }
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: ,
                               destination: ,
                               label: )
            }
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: ,
                               label: )
            }
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: )
            }
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}
```

```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}
```

```swift
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}

extension View {
    func onNavigation<Destination: View>(
        isActive: Binding<Bool>,
        _ destination: @escaping () -> Destination)
    -> some View {
        self.modifier(
            NavigationModifier(
                isActive: isActive,
                destination: destination
            )
        )
    }
}
```

```swift
struct FirstFeatureView: View {

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
struct FirstFeatureView: View {

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
struct FirstFeatureView: View {

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {
                FirstFeatureDetailView()
            } label: {
                Text("Navigate towards Detail View")
            }
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            NavigationLink {

            } label: {
                Text("Navigate towards Detail View")
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {

            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

We’ve implemented a navigation Coordinator!

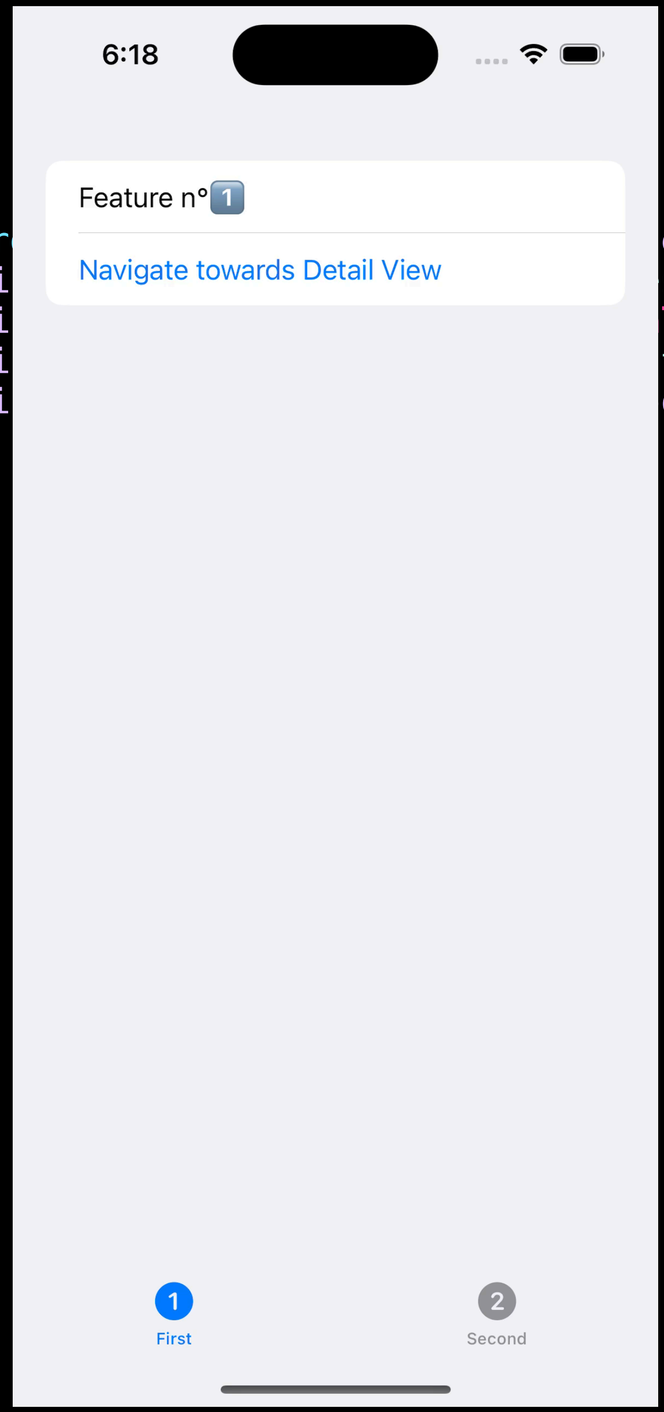
```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class Coor
    @Publi
    @Publi
    @Publi
    @Publi
}
```

```swift
ect {
s = .first
lse
tent?
ol = false
```



```
struct NavigationModifier<Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var isActive: Bool

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: $isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }

    var isActive: Binding<Bool> {
        Binding<Bool>(get: {
            return item != nil
        }, set: { newValue in
            if newValue == false {
                item = nil
            }
        })
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: () -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }

    var isActive: Binding<Bool> {
        Binding<Bool>(get: {
            return item != nil
        }, set: { newValue in
            if newValue == false {
                item = nil
            }
        })
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: (Item) -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: isActive,
                               destination: destination,
                               label: { EmptyView() })
            }
    }

    var isActive: Binding<Bool> {
        Binding<Bool>(get: {
            return item != nil
        }, set: { newValue in
            if newValue == false {
                item = nil
            }
        })
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: (Item) -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: isActive,
                               destination: { item.map { destination($0) } },
                               label: { EmptyView() })
            }
    }

    var isActive: Binding<Bool> {
        Binding<Bool>(get: {
            return item != nil
        }, set: { newValue in
            if newValue == false {
                item = nil
            }
        })
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {

    @Binding var item: Item?

    @ViewBuilder var destination: (Item) -> Destination

    func body(content: Content) -> some View {
        content
            .overlay {
                NavigationLink(isActive: isActive,
                               destination: { item.map { destination($0) } },
                               label: { EmptyView() })
            }
    }

    var isActive: Binding<Bool> {
        Binding<Bool>(get: {
            return item != nil
        }, set: { newValue in
            if newValue == false {
                item = nil
            }
        })
    }
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {
    /* ... */
}
```

```
struct NavigationModifier<Item, Destination: View>: ViewModifier {
    /* ... */
}

extension View {
    func onNavigation<Item, Destination: View>(
        item: Binding<Item?>,
        _ destination: @escaping (Item) -> Destination
    ) -> some View {
        self.modifier(
            NavigationModifierWithItem(
                item: item,
                destination: destination
            ))

    }
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View") {
                viewModel.navigationWithBool = true
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")


        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View with Argument A") {
                viewModel.navigationWithArgument = "Argument A"
            }

            Button("Navigate towards Detail View with Argument B") {
                viewModel.navigationWithArgument = "Argument B"
            }
        }
        .onNavigation(
            isActive: $viewModel.navigationWithBool,
            destination: { FirstFeatureDetailView() }
        )
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View with Argument A") {
                viewModel.navigationWithArgument = "Argument A"
            }

            Button("Navigate towards Detail View with Argument B") {
                viewModel.navigationWithArgument = "Argument B"
            }
        }
        .onNavigation(item: $viewModel.navigationWithArgument) { detailContent in
            FirstFeatureDetailView(additionalInfo: detailContent)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View with Argument A") {
                viewModel.navigationWithArgument = "Argument A"
            }

            Button("Navigate towards Detail View with Argument B") {
                viewModel.navigationWithArgument = "Argument B"
            }
        }
        .onNavigation(item: $viewModel.navigationWithArgument) { detailContent in
            FirstFeatureDetailView(additionalInfo: detailContent)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

We can now pass data as we navigate!

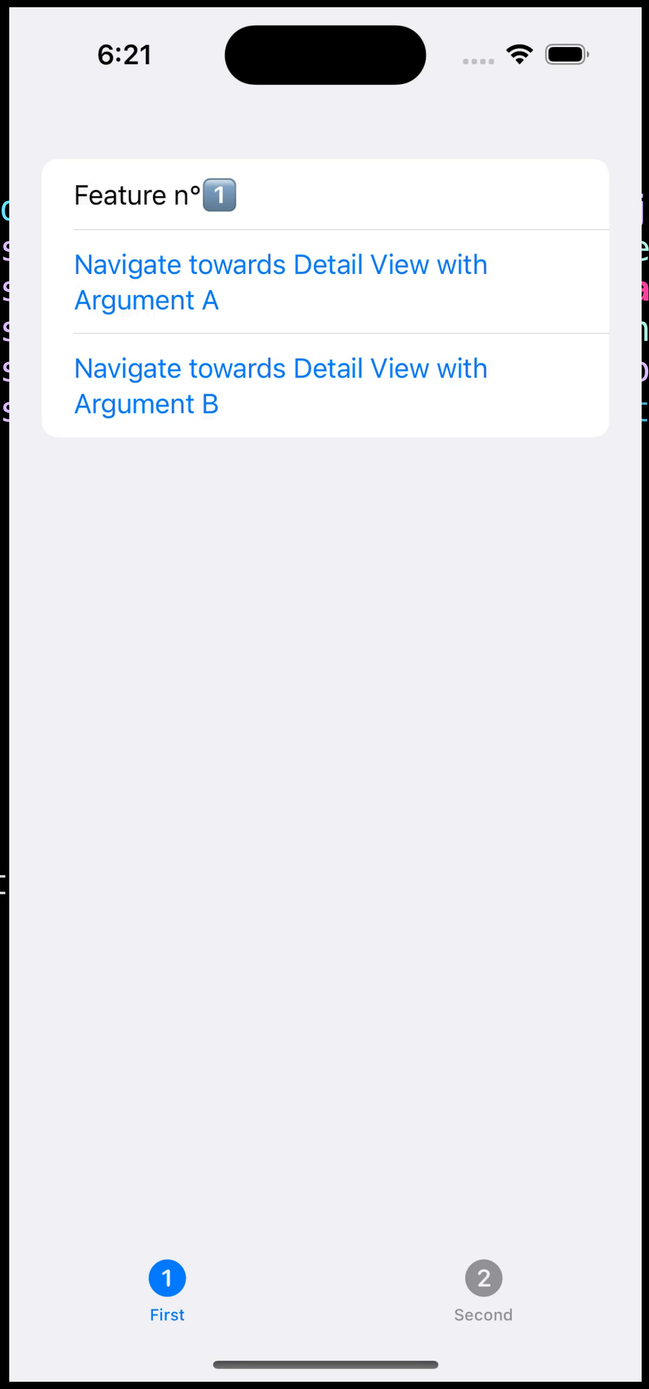
```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View with Argument A") {
                viewModel.navigationWithArgument = "Argument A"
            }

            Button("Navigate towards Detail View with Argument B") {
                viewModel.navigationWithArgument = "Argument B"
            }
        }
        .onNavigation(item: $viewModel.navigationWithArgument) { detailContent in
            FirstFeatureDetailView(additionalInfo: detailContent)
        }
    }
}
```

```swift
class CoordinatorViewModel: ObservableObject {
    @Published var currentRoute: TabRoutes = .first
    @Published var showAuthent: Bool = false
    @Published var modalContent: ModalContent?
    @Published var navigationWithBool: Bool = false
    @Published var navigationWithArgument: String?
}
```

```swift
struct FirstFeatureView: View {

    @EnvironmentObject var viewModel: CoordinatorViewModel

    var body: some View {
        Form {
            Text("Feature n°1️⃣")

            Button("Navigate towards Detail View with Argument A") {
                viewModel.navigationWithArgument = "Argument A"
            }

            Button("Navigate towards Detail View with Argument B") {
                viewModel.navigationWithArgument = "Argument B"
            }
        }
        .onNavigation(item: $viewModel.navigationWithArgument) { detailCont
            FirstFeatureDetailView(additionalInfo: detailContent)
        }
    }
}
```

```swift
class Coord
    @Publis
    @Publis
    @Publis
    @Publis
    @Publis
}
```

```swift
ject {
es = .first
alse
ntent?
ool = false
t: String?
```



What can still go wrong?

What can still go wrong?

What can still go wrong?

- Navigation can be tricky when it becomes complex

What can still go wrong?

- Navigation can be tricky when it becomes complex
- Typically, when you need to dynamically decide which view to push

What can still go wrong?

- Navigation can be tricky when it becomes complex
- Typically, when you need to dynamically decide which view to push
- In these situations, it can be simpler to handle the navigation in UIKit

What can still go wrong?

- Navigation can be tricky when it becomes complex
- Typically, when you need to dynamically decide which view to push
- In these situations, it can be simpler to handle the navigation in UIKit
- Let’s see how we can do it!

How to use SwiftUI with a UIKit navigation

How to use SwiftUI with a UIKit navigation

How to use SwiftUI with a UIKit navigation

- Using a Swift `View` inside a `UINavigationController` is actually pretty easy

```
class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }
}
```

```
class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }
}
```

How to use SwiftUI with a UIKit navigation

- Using a Swift `View` inside a `UINavigationController` is actually pretty easy:

How to use SwiftUI with a UIKit navigation

- Using a Swift `View` inside a `UINavigationController` is actually pretty easy:
- You embed the `View` inside a `UIHostingController`, and that's it!

How to use SwiftUI with a UIKit navigation

- Using a Swift `View` inside a `UINavigationController` is actually pretty easy:
- You embed the `View` inside a `UIHostingController`, and that's it!
- But then comes the tricky question: how do you trigger the navigation from one SwiftUI `View` to another?

```
struct FirstView: View {

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {

            }
        }
    }
}
```

```swift
struct FirstView: View {

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {

            }
        }
    }
}


struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

```swift
struct FirstView: View {

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                // How to trigger the navigation?
            }
        }
    }
}


struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

```
class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }
}
```

```
enum Destination {
    case secondView(text: String)
}

class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }
}
```

```swift
enum Destination {
    case secondView(text: String)
}

class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }

    func navigate(to destination: Destination) {
        guard case let .secondView(text) = destination else {
            return
        }

        let secondViewController = UIHostingController(
            rootView: SecondView(text: text)
        )

        navigationController.pushViewController(
            secondViewController,
            animated: true
        )
    }
}
```

```
struct NavigationActionKey: EnvironmentKey {

}
```

```
struct NavigationActionKey: EnvironmentKey {
    static var defaultValue: (Destination) -> Void = { _ in }
}
```

```
struct NavigationActionKey: EnvironmentKey {
    static var defaultValue: (Destination) -> Void = { _ in }
}

extension EnvironmentValues {
    var navigationAction: (Destination) -> Void {
        get {
            self[NavigationActionKey.self]
        }
        set {
            self[NavigationActionKey.self] = newValue
        }
    }
}
```

```swift
enum Destination {
    case secondView(text: String)
}

class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }

    func navigate(to destination: Destination) {
        guard case let .secondView(text) = destination else {
            return
        }

        let secondViewController = UIHostingController(
            rootView: SecondView(text: text)
        )

        navigationController.pushViewController(
            secondViewController,
            animated: true
        )
    }
}
```

```
func start() {
    let firstViewController = UIHostingController(
        rootView: FirstView()
    )

    navigationController.pushViewController(
        firstViewController,
        animated: false
    )
}
```

```swift
func start() {
    let firstViewController = UIHostingController(
        rootView: FirstView()
            .environment(\.navigationAction, { [weak self] destination in
                self?.navigate(to: destination)
            })
    )

    navigationController.pushViewController(
        firstViewController,
        animated: false
    )
}
```

```swift
enum Destination {
    case secondView(text: String)
}

class Coordinator {

    let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        let firstViewController = UIHostingController(
            rootView: FirstView()
                .environment(\.navigationAction, { [weak self] destination in
                    self?.navigate(to: destination)
                })
        )

        navigationController.pushViewController(
            firstViewController,
            animated: false
        )
    }

    func navigate(to destination: Destination) {
        guard case let .secondView(text) = destination else {
            return
        }

        let secondViewController = UIHostingController(
            rootView: SecondView(text: text)
        )

        navigationController.pushViewController(
            secondViewController,
            animated: true
        )
    }
}
```

```swift
struct FirstView: View {

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                // How to trigger the navigation?
            }
        }
    }
}


struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

```
struct FirstView: View {

    @Environment(\.navigationAction) var navigationAction

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                // How to trigger the navigation?
            }
        }
    }
}

struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

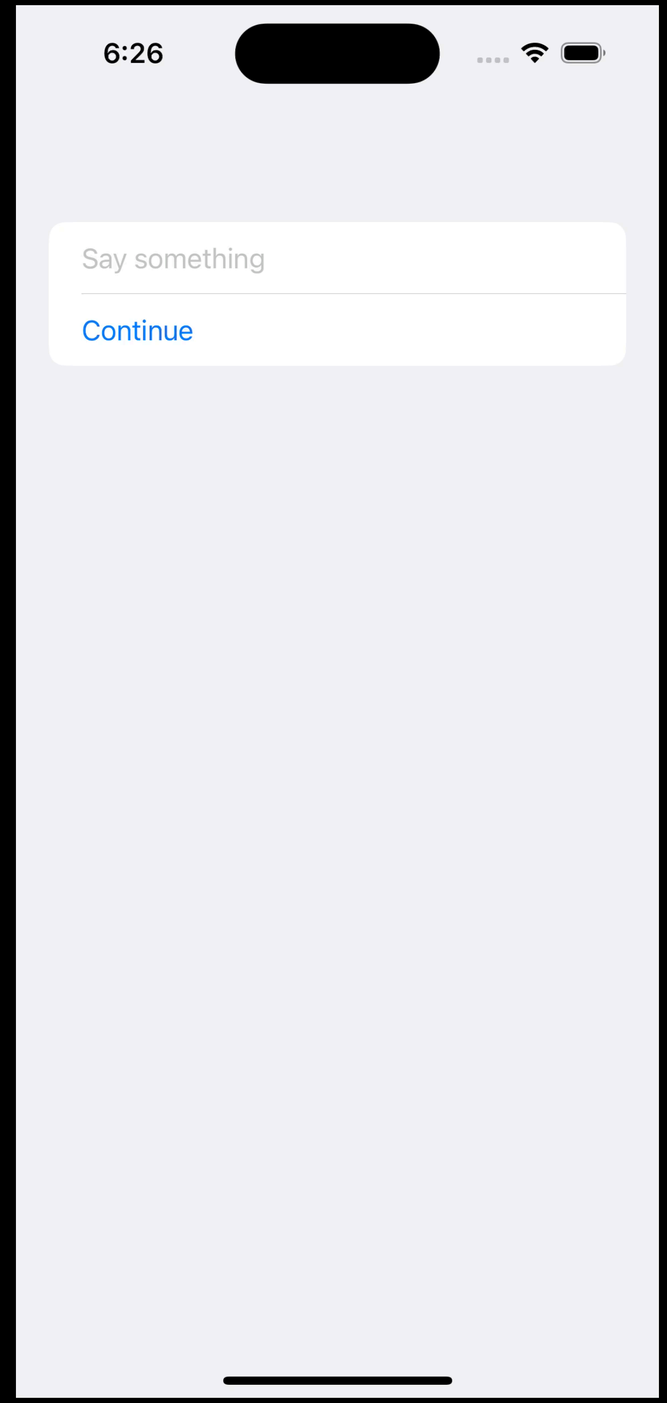
```swift
struct FirstView: View {

    @Environment(\.navigationAction) var navigationAction

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                navigationAction(.secondView(text: text))
            }
        }
    }
}

struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

```
struct FirstView: View {

    @Environment(\.navigationAction) var navigationAction

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                navigationAction(.secondView(text: text))
            }
        }
    }
}

struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

And we’re done 👌

```swift
struct FirstView: View {

    @Environment(\.navigationAction) var navigationAction

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                navigationAction(.secondView(text: text))
            }
        }
    }
}

struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```

```swift
struct FirstView: View {

    @Environment(\.navigationAction) var navigationAction

    @State var text: String = ""

    var body: some View {
        Form {
            TextField("Say something", text: $text)

            Button("Continue") {
                navigationAction(.secondView(text: text))
            }
        }
    }
}


struct SecondView: View {
    let text: String

    var body: some View {
        Text("Here's what you said: \(text)")
    }
}
```



Recap

Recap

Recap

- The main challenge when we want to use SwiftUI in a UIKit app is to make it fit an already existing architecture

Recap

- The main challenge when we want to use SwiftUI in a UIKit app is to make it fit an already existing architecture
- We’ve seen that with a few efforts, it’s possible to make a SwiftUI view fit within a UIKit-oriented architecture

Recap

- The main challenge when we want to use SwiftUI in a UIKit app is to make it fit an already existing architecture
- We've seen that with a few efforts, it's possible to make a SwiftUI view fit within a UIKit-oriented architecture
- But don't forget: SwiftUI is a tool that can help us save time in a lot of situations…

Recap

- The main challenge when we want to use SwiftUI in a UIKit app is to make it fit an already existing architecture
- We've seen that with a few efforts, it's possible to make a SwiftUI view fit within a UIKit-oriented architecture
- But don't forget: SwiftUI is a tool that can help us save time in a lot of situations…
- …however, if it becomes clear it's not the right tool, we can always quickly switch back to UIKit!

If you haven’t yet had time to learn SwiftUI, I have something for you!

https://vpradeilles.gumroad.com/l/learning-swiftui-when-you-already-know-uikit/plswift-swiftui

https://vpradeilles.gumroad.com/l/learning-swiftui-when-you-already-know-uikit/plswift-swiftui


That's all Folks!

Thank You! 

Thank You! ☺️

Thank you to the PhotoRoom iOS team:
Jan, Marc & Franck 🤝



Twitter

YouTube

https://www.photoroom.com/company/