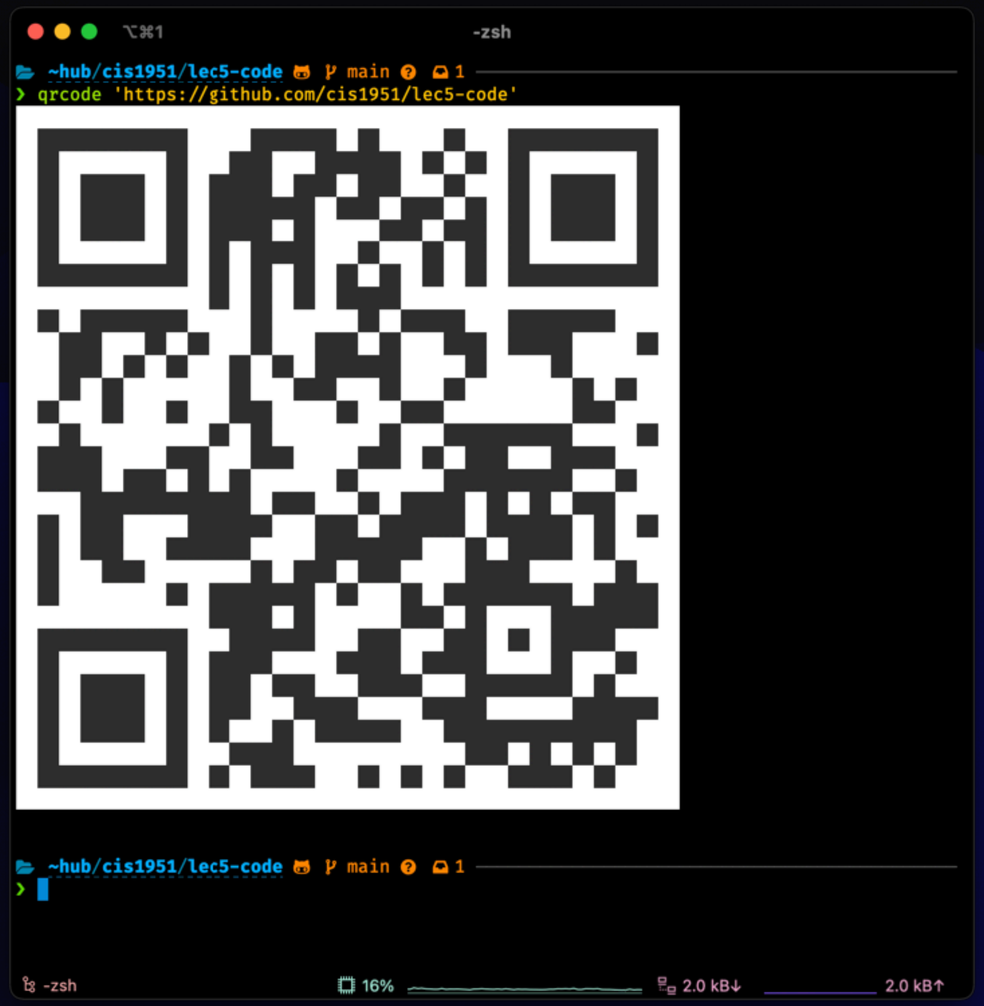
App Structure

Lecture 5

CIS 1951

https://github.com/cis1951/lec5-code



Previously, on CIS 1951...

SwiftUI State Management

@Observable

@Binding

@State

.onDisappear

.onAppear

.onChange

So far, we've only made simple, single-screen apps.

That changes today.

This week

The tools you need to create larger apps

Navigation & modal presentations

MVVM

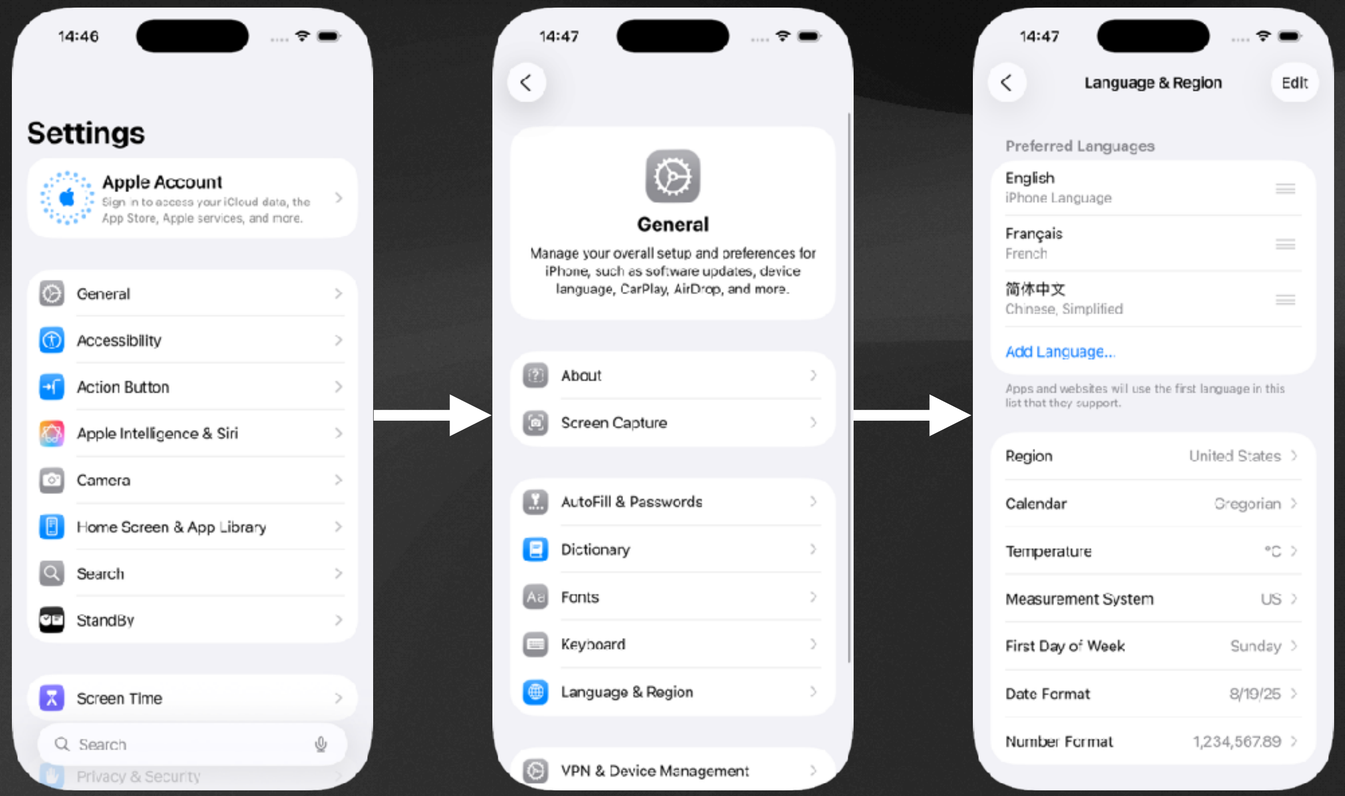
@Observable, @Bindable, @Environment

Navigation & Modal Presentations

How do we organize multiple screens?

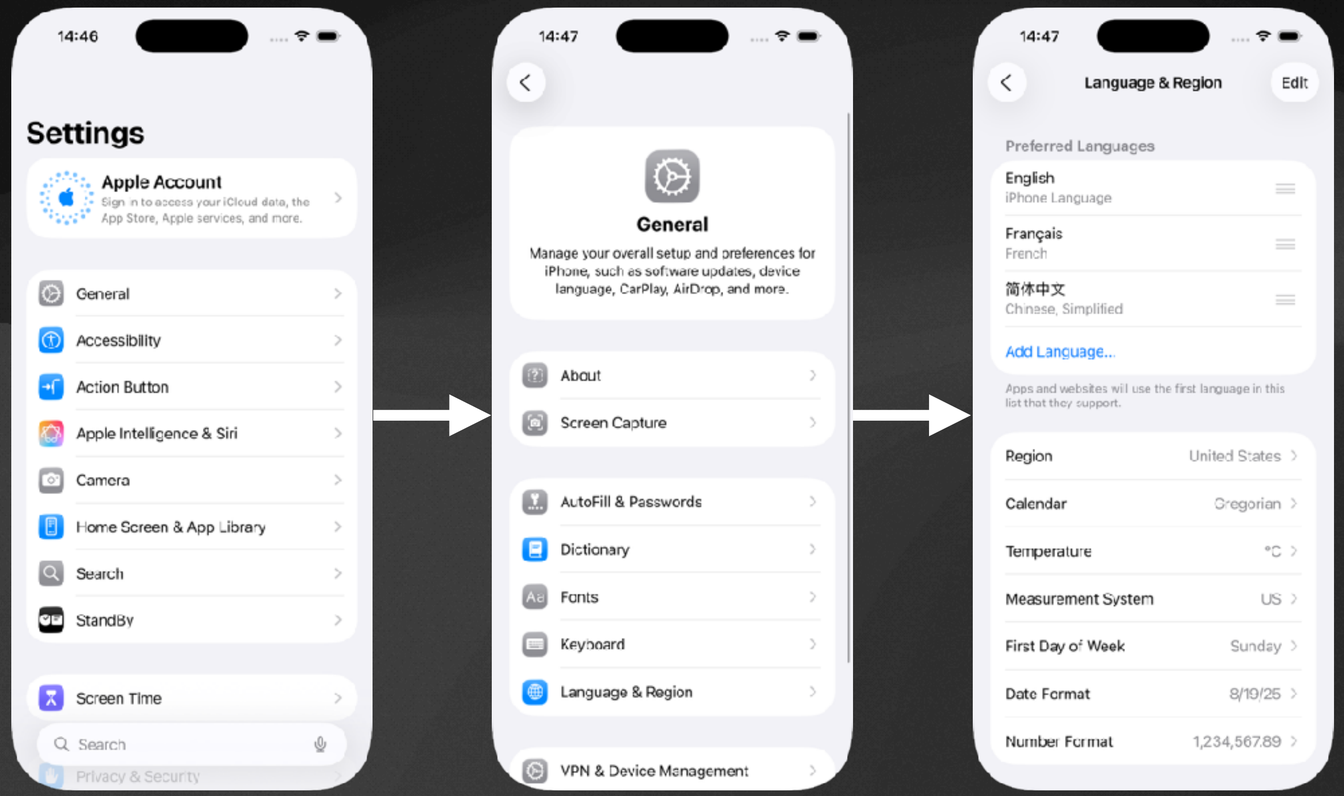
Hierarchical Navigation

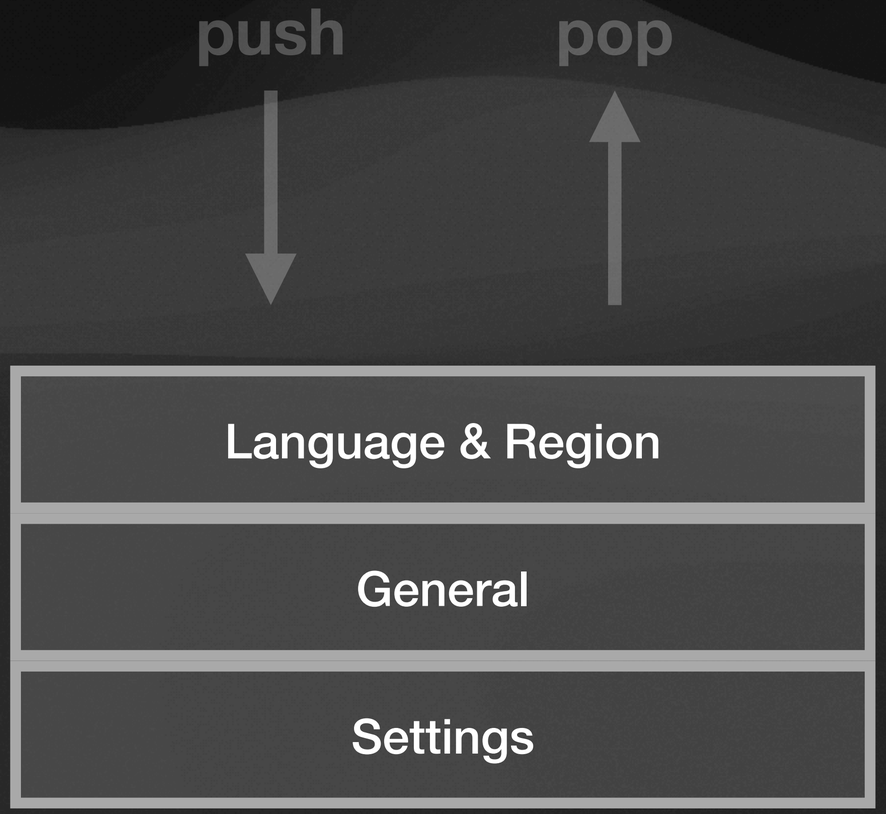
Example: Settings App



Hierarchical Navigation

Example: Settings App



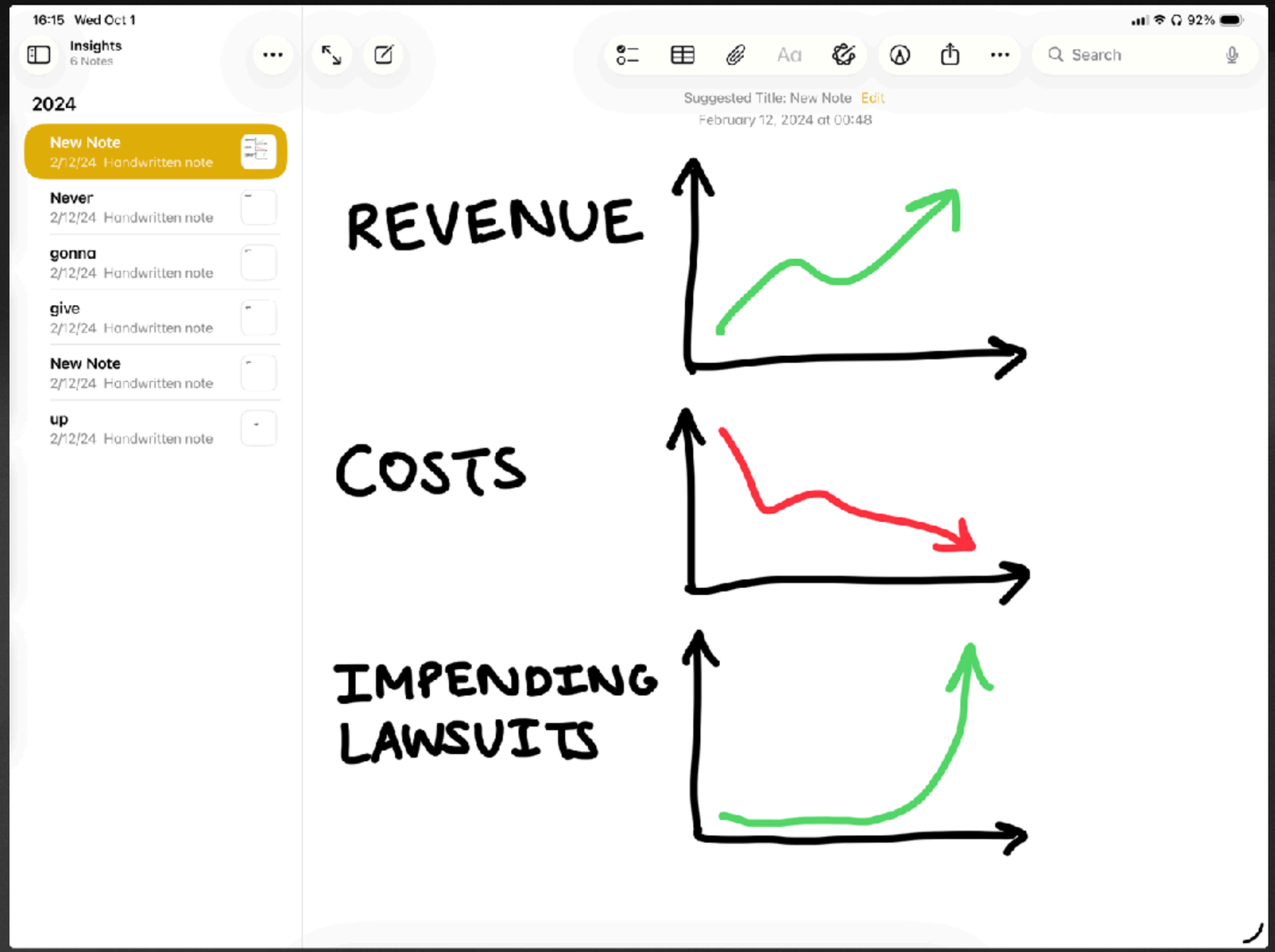


Master-Detail Navigation

Example: Notes App

Master

List of notes



Detail

Single note

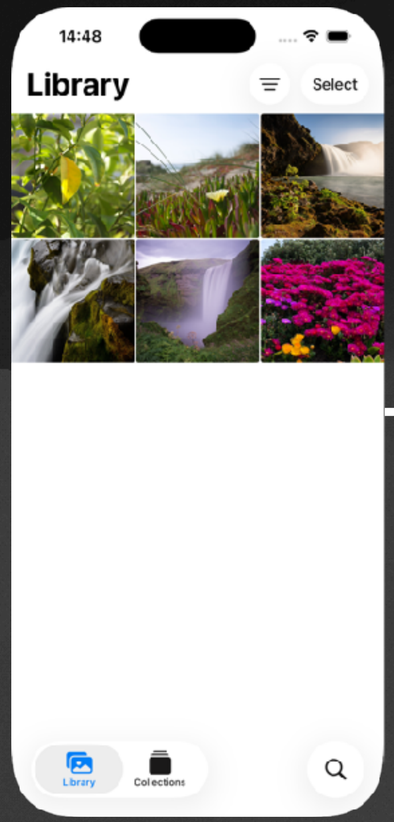
Tab Bar Navigation

Example: Clock App

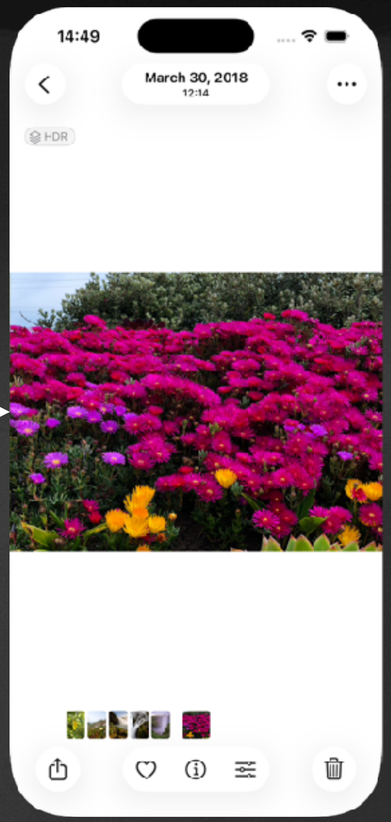
Bottom/top bar for **quick navigation**

Hybrid Navigation

Example: Photos App

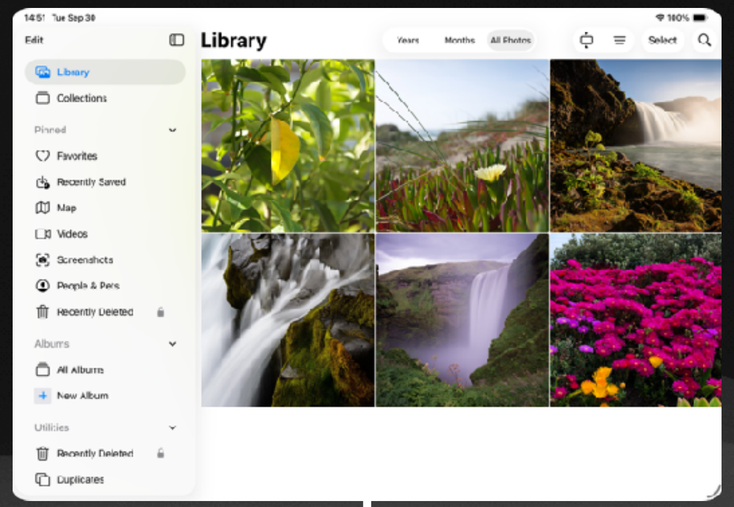


Tab bar



Hierarchical

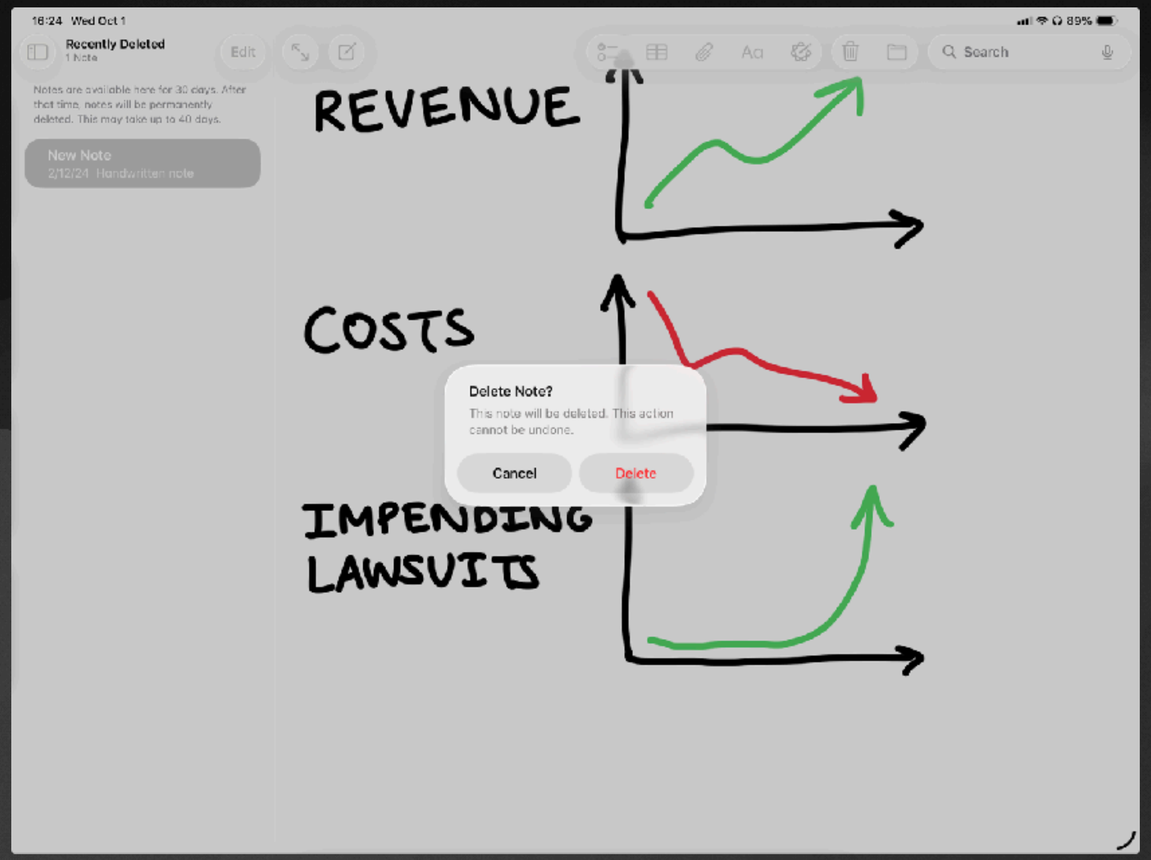
Master-detail

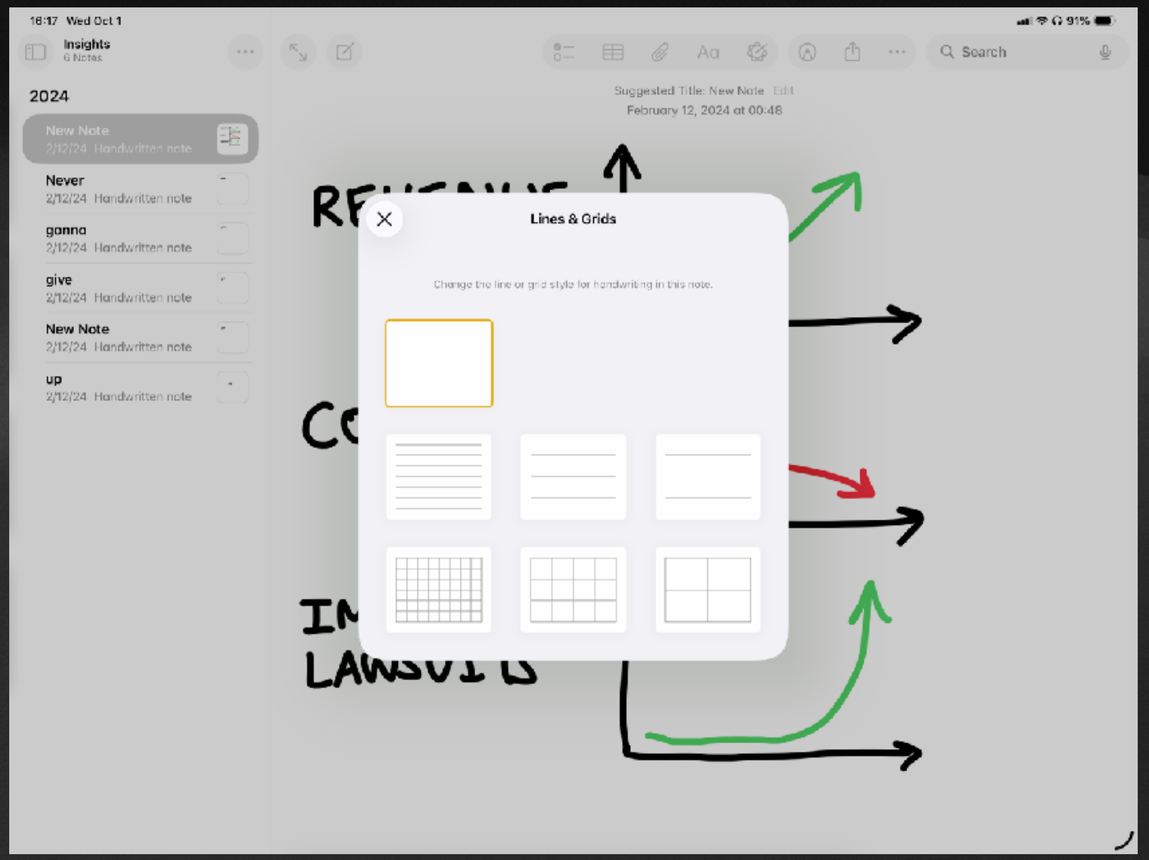


Hierarchical

Modals

Example: Notes App





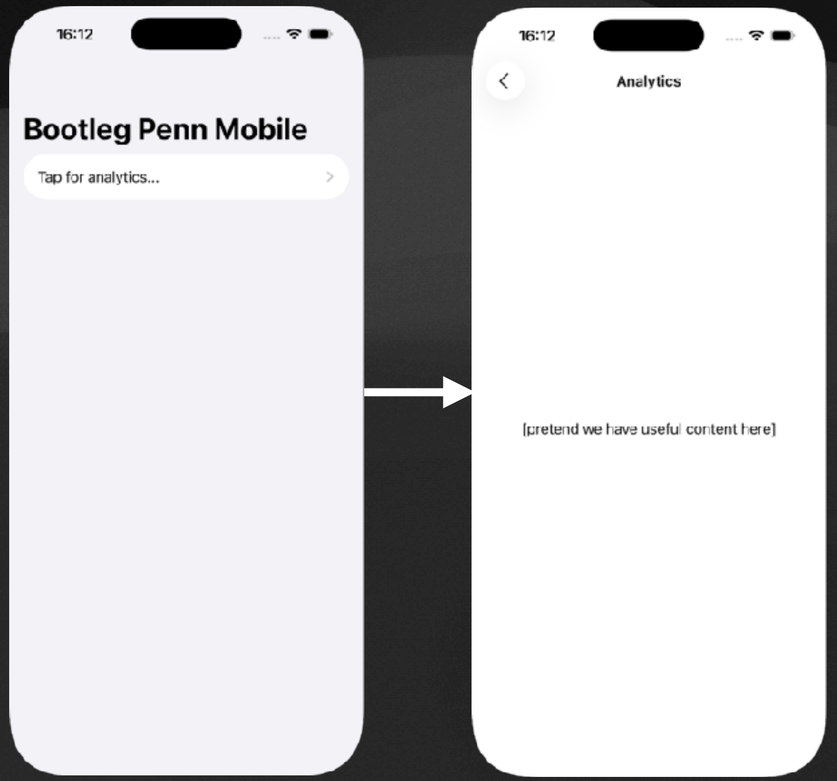
Lightweight and **focused** interactions

Implementation

NavigationStack

Directly linking to views

```
NavigationStack {
    List {
        NavigationLink("Tap for analytics...") {
            Text("[pretend we have useful content here]")
                .navigationTitle("Analytics")
                .navigationBarTitleDisplayMode(.inline)
        }
    }
    .navigationTitle("Bootleg Penn Mobile")
}
```



NavigationStack

Presenting based on data

```
NavigationStack {
    List {
        NavigationLink("Tap for analytics...", value: "Analytics")
    }
    .navigationTitle("Bootleg Penn Mobile")
    .navigationDestination(for: String.self) { value in
        Text("[pretend we have useful content here]")
            .navigationTitle(value)
            .navigationBarTitleDisplayMode(.inline)
    }
}
```


Allows you to modify path programmatically

`value` is passed into **`.navigationDestination`**

Can help make code cleaner

NavigationStack

Programmatically changing the path

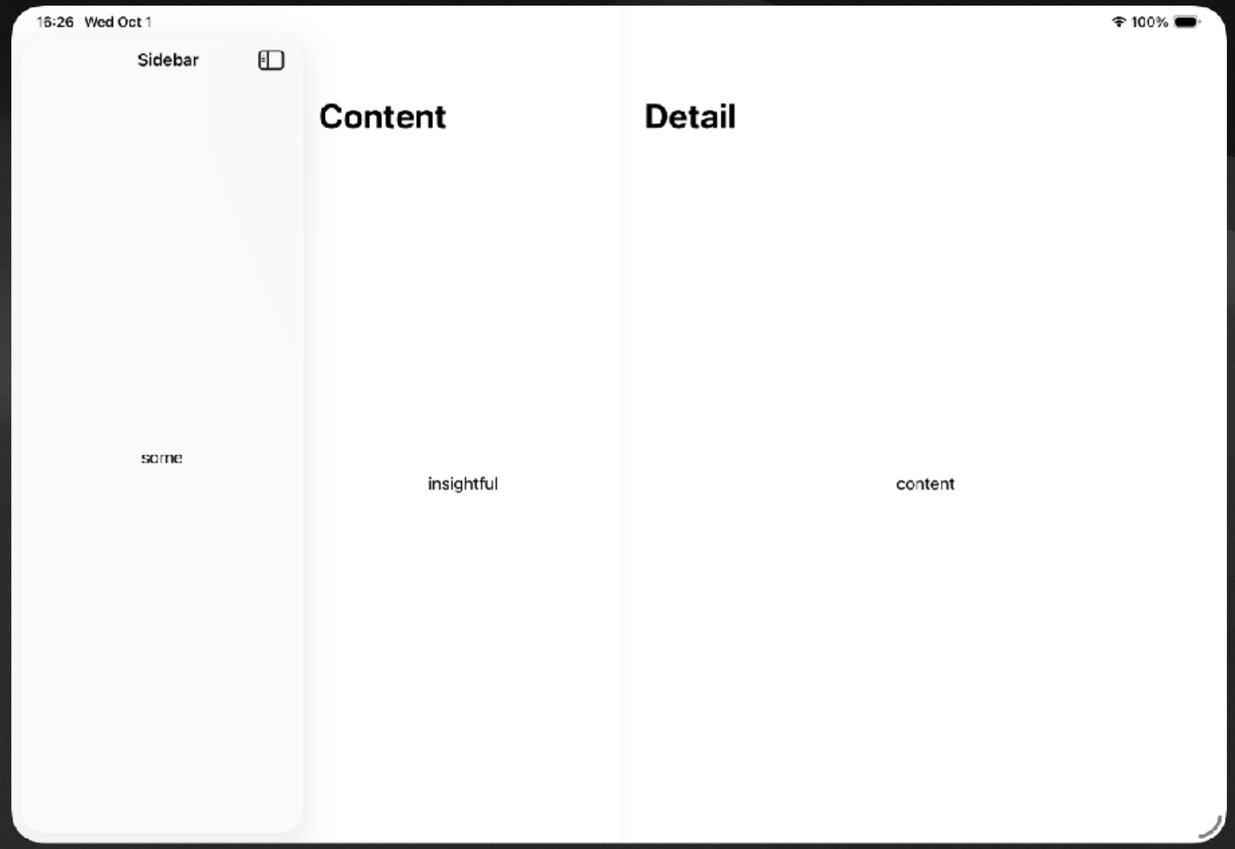
```
@State var path = NavigationPath()
...

NavigationStack(path: $path) {
    List {
        Button(action: {
            path.append("Analytics")
        }) {
            Text("Tap for analytics...")
        }
    }
    .navigationTitle("Bootleg Penn Mobile")
    .navigationDestination(for: String.self) { value in
        Text("[pretend we have useful content here]")
            .navigationTitle(value)
            .navigationBarTitleDisplayMode(.inline)
    }
}
```

Allows you to modify path programmatically

NavigationSplitView

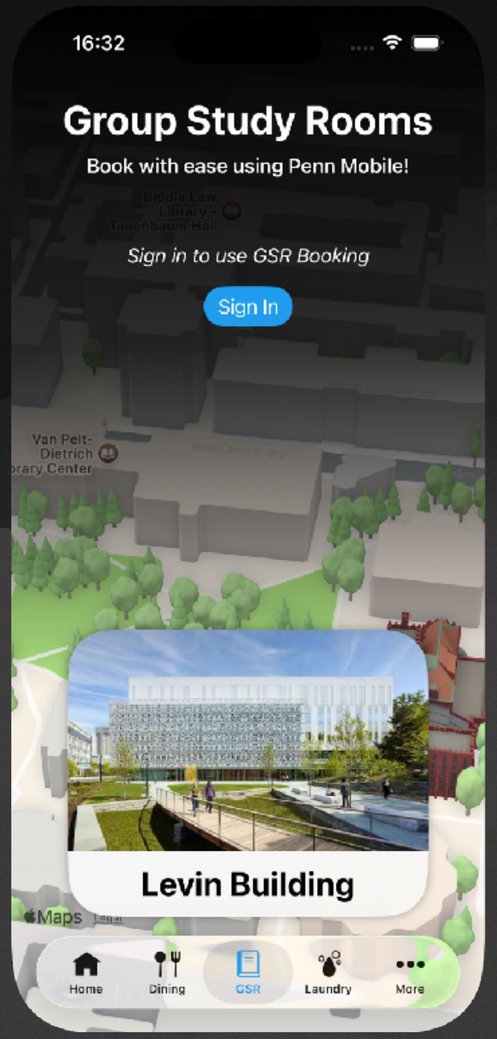
Multi-column layouts

```
NavigationSplitView {
    Text("Sidebar")
} content: {
    Text("Content")
} detail: {
    Text("Detail")
}
```



Appears as a NavigationStack on iPhone

TabView



CIS 1951 developer.apple.com/documentation/SwiftUI/TabView

Developer News Discover Design Develop Distribute Support Account

Documentation Language: Swift

All Technologies

SwiftUI

- TabView
  - Creating a tab view
  - init(content:)
  - init(selection:content:)
  - Configuring search activation
  - TabSearchActivation
  - Tab
  - TabRole
  - TabSection
  - func tabViewStyle<S>(S) -> some ...
  - Configuring a tab bar
  - func tabViewSidebarHeader<Cont...
  - func tabViewSidebarFooter<Conte...
  - func tabViewSidebarBottomBar<C...

Filter /

SwiftUI / TabView

Structure

TabView

A view that switches between multiple child views using interactive user interface elements.

iOS 13.0+ | iPadOS 13.0+ | Mac Catalyst 13.0+ | macOS 10.15+ | tvOS 13.0+ | visionOS 1.0+ | watchOS 7.0+

```
struct TabView<SelectionValue, Content> where SelectionValue : Hashable,
```

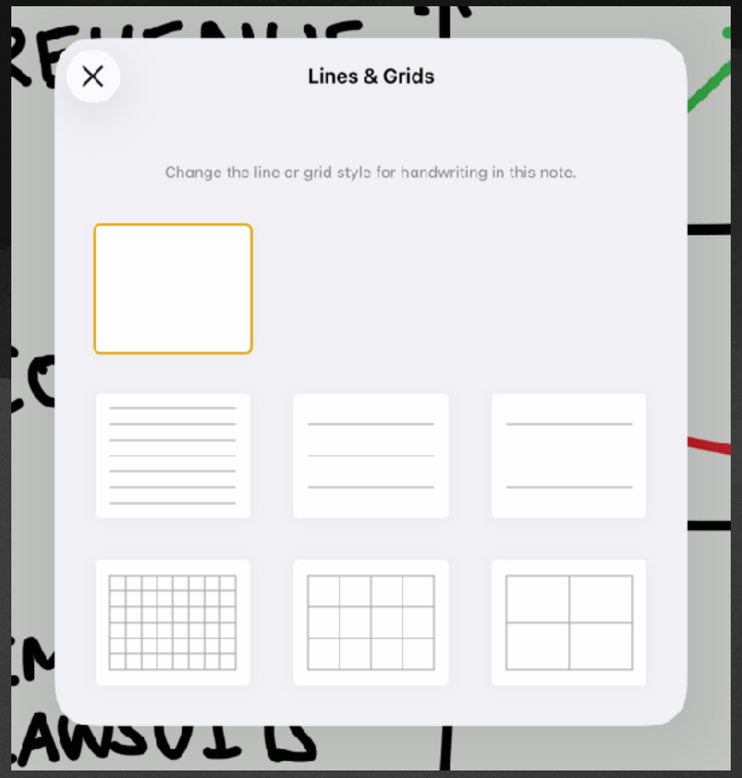
Overview

To create a user interface with tabs, place Tabs in a `TabView`. On iOS, you can also use one of the badge modifiers, like `badge(_:)`, to assign a badge to each of the tabs.

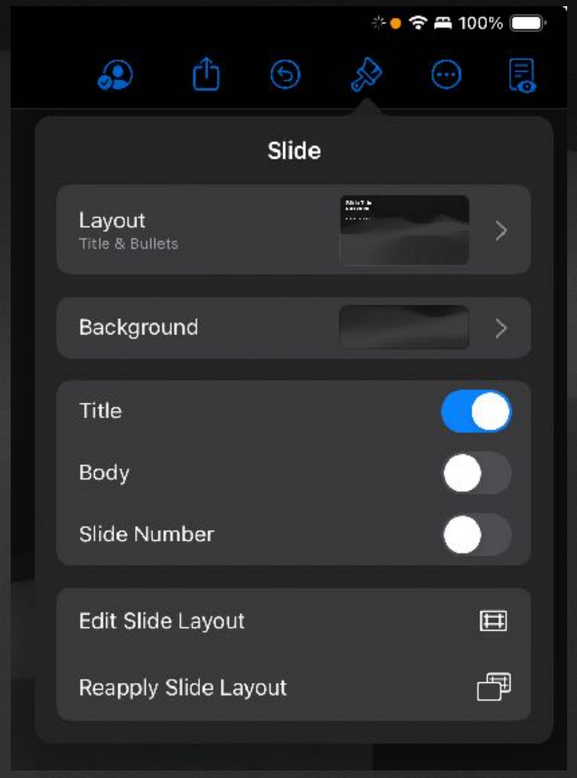
The following example creates a tab view with three tabs, each presenting a custom child view. The first tab has a numeric badge and the third has a string badge.

Modal presentations

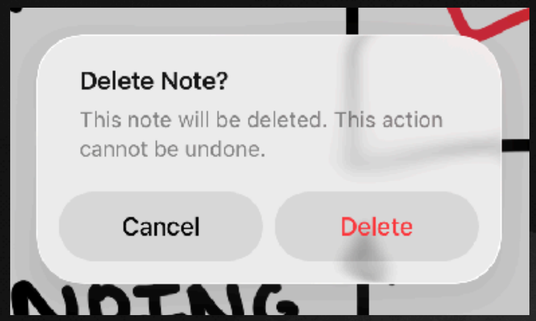
For lightweight, focused interactions



.sheet

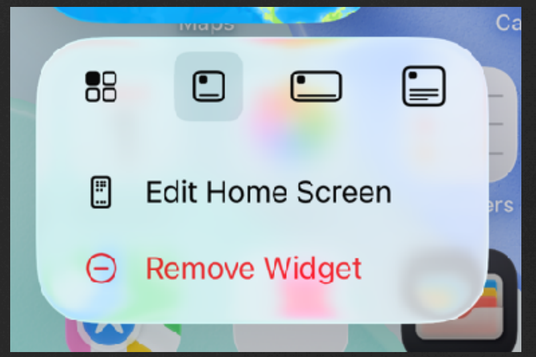


.popover



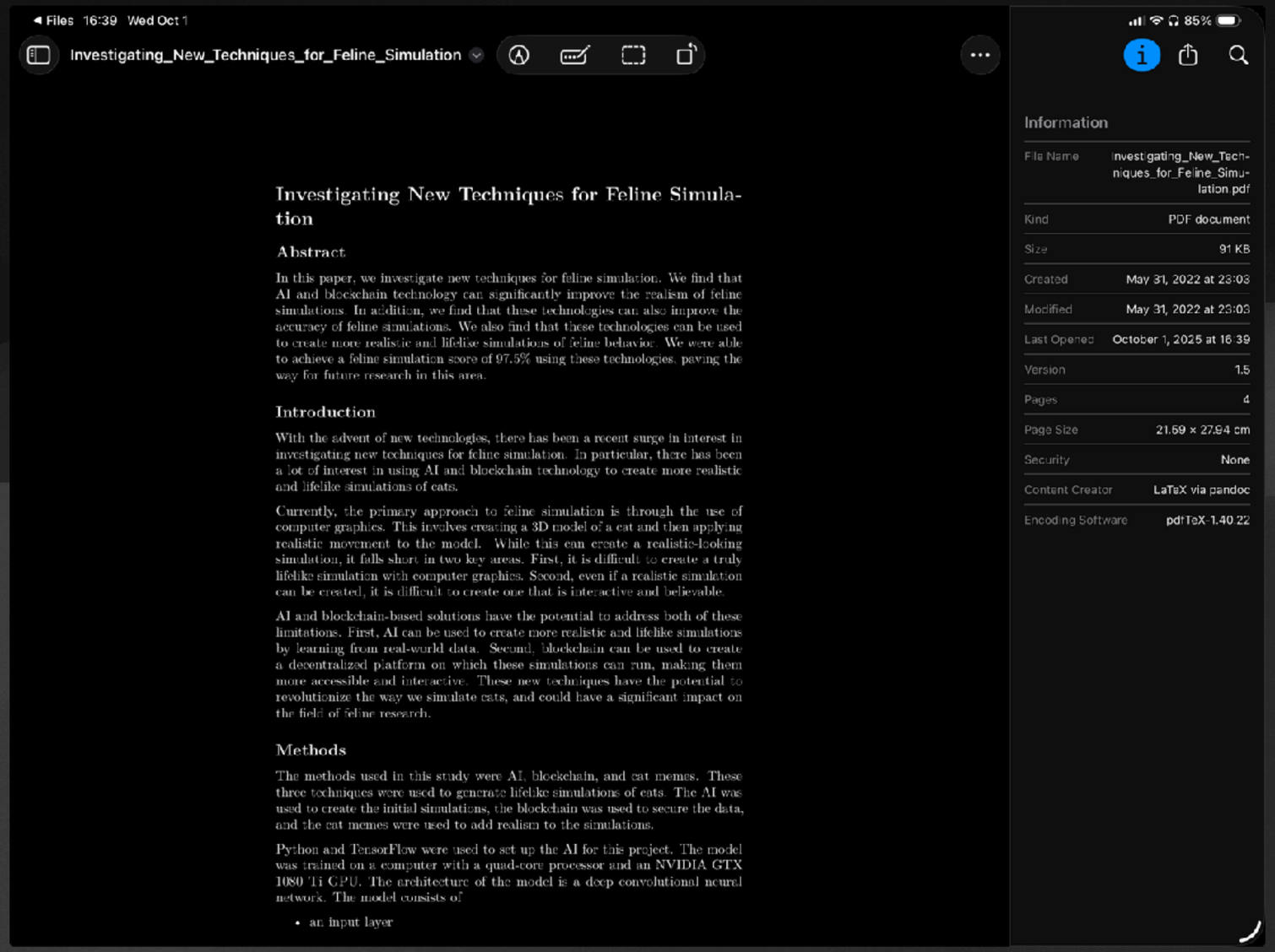
.alert

.confirmationDialog

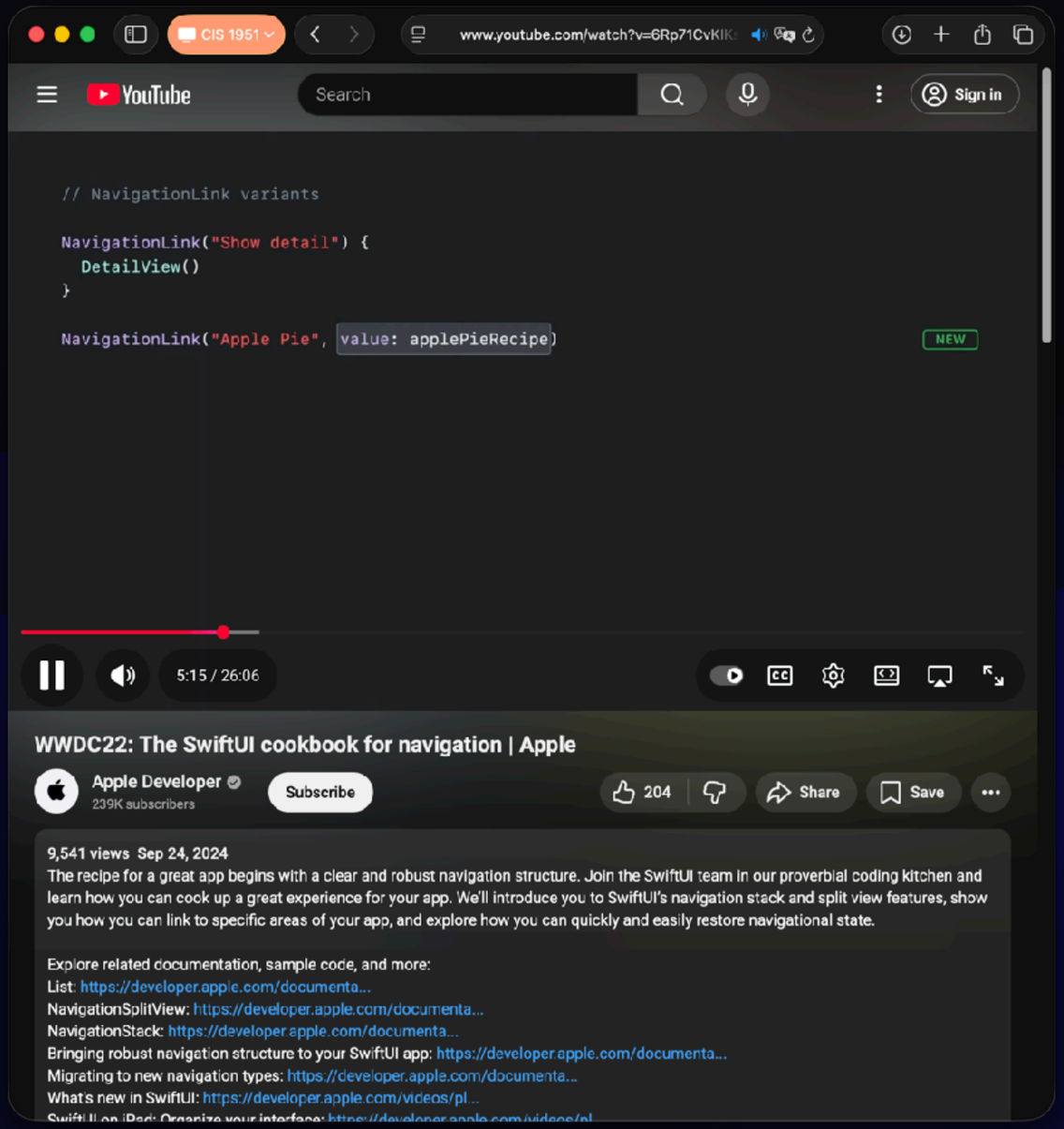


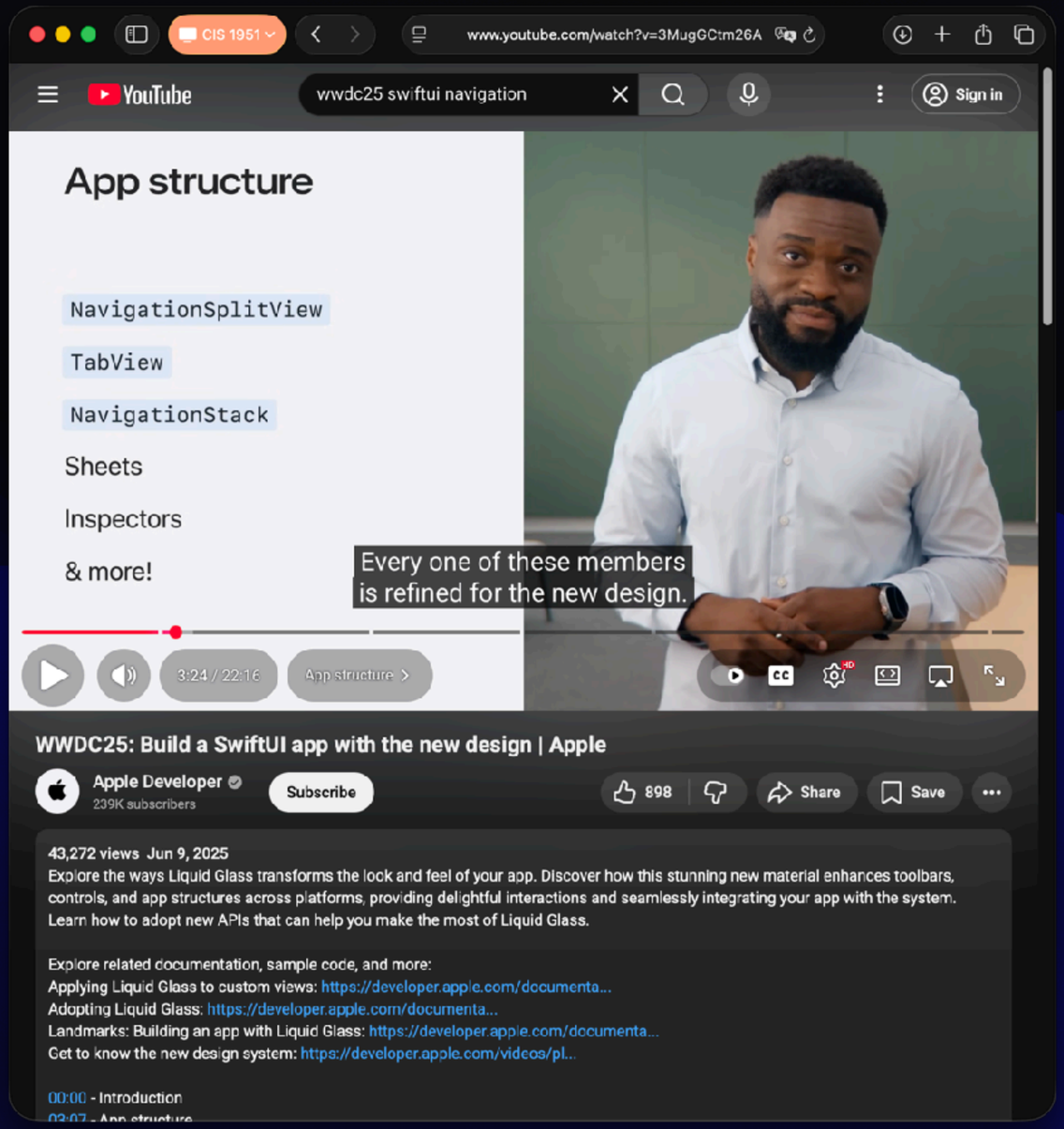
Menu
– OR –
.contextMenu

.inspector



Ideal for showing auxiliary content


www.youtube.com/watch?v=6Rp71CvKIK
YouTube
Search
Sign in
// NavigationLink variants
NavigationLink("Show detail") {
DetailView()
}
NavigationLink("Apple Pie", value: applePieRecipe)
NEW
5:15 / 26:06
WWDC22: The SwiftUI cookbook for navigation | Apple
Apple Developer
239K subscribers
Subscribe
204
Share
Save
9,541 views Sep 24, 2024
The recipe for a great app begins with a clear and robust navigation structure. Join the SwiftUI team in our proverbial coding kitchen and learn how you can cook up a great experience for your app. We'll introduce you to SwiftUI's navigation stack and split view features, show you how you can link to specific areas of your app, and explore how you can quickly and easily restore navigational state.
Explore related documentation, sample code, and more:
List: https://developer.apple.com/documenta...
NavigationSplitView: https://developer.apple.com/documenta...
NavigationStack: https://developer.apple.com/documenta...
Bringing robust navigation structure to your SwiftUI app: https://developer.apple.com/documenta...
Migrating to new navigation types: https://developer.apple.com/documenta...
What's new in SwiftUI: https://developer.apple.com/videos/pl...


www.youtube.com/watch?v=3MugGCtm26A
YouTube
wwdc25 swiftui navigation
Sign in
App structure
NavigationSplitView
TabView
NavigationStack
Sheets
Inspectors
& more!
Every one of these members is refined for the new design.
3:24 / 22:16
App structure
WWDC25: Build a SwiftUI app with the new design | Apple
Apple Developer
239K subscribers
Subscribe
898
Share
Save
43,272 views Jun 9, 2025
Explore the ways Liquid Glass transforms the look and feel of your app. Discover how this stunning new material enhances toolbars, controls, and app structures across platforms, providing delightful interactions and seamlessly integrating your app with the system. Learn how to adopt new APIs that can help you make the most of Liquid Glass.
Explore related documentation, sample code, and more:
Applying Liquid Glass to custom views: https://developer.apple.com/documenta...
Adopting Liquid Glass: https://developer.apple.com/documenta...
Landmarks: Building an app with Liquid Glass: https://developer.apple.com/documenta...
Get to know the new design system: https://developer.apple.com/videos/pl...
00:00 - Introduction

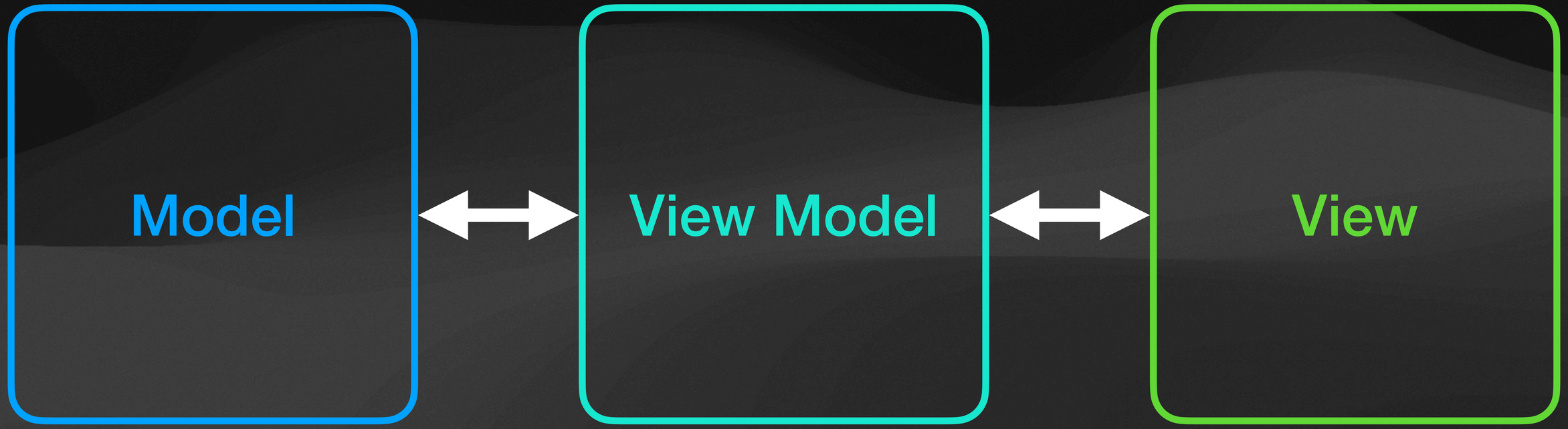
MVVM

Separation of Concerns

- Split code into **modular components**
- Each component only **handles one thing** (a "concern")
- Why? More testable, reusable, maintainable code

MVVM

Model-View-View Model



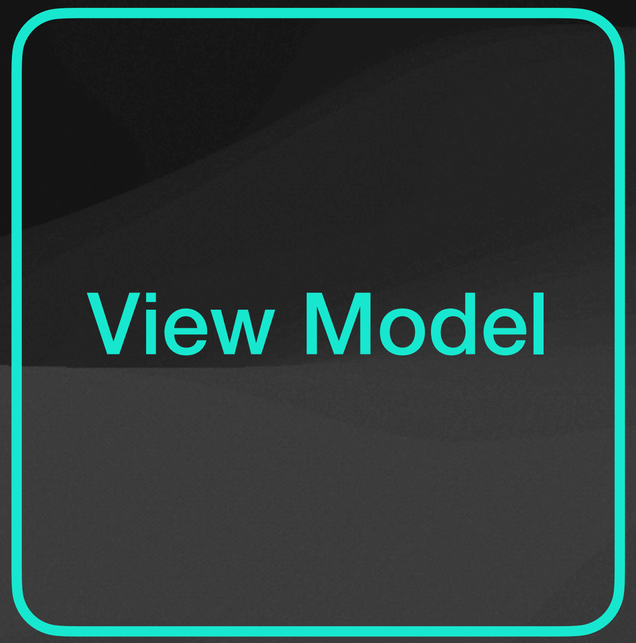
Represents **the app's data**

Coordinates between the two

Defines **what the user sees**

MVVM

The View Model



Coordinates
between the two

Lets the view **bind to data** and **send commands**

Notifies the view of any changes

Converts data to and from what the view wants

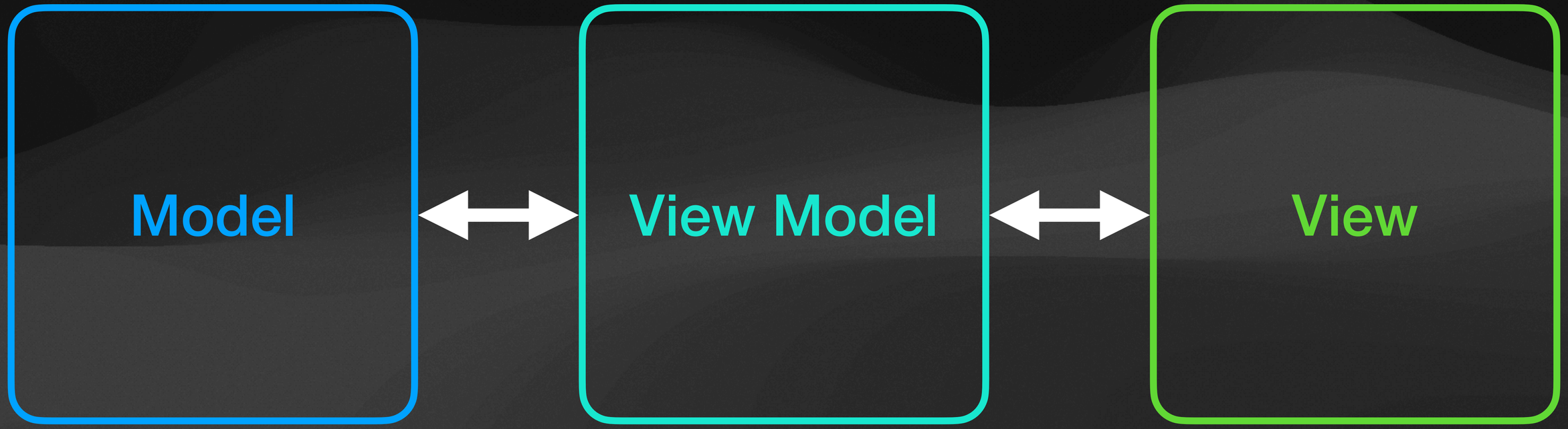
Isolates the view from its underlying data

Usually **a class**

https://learn.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)

MVVM

How we used it last week



TodoItem

Represents an actual todo item

Todo(View)Model

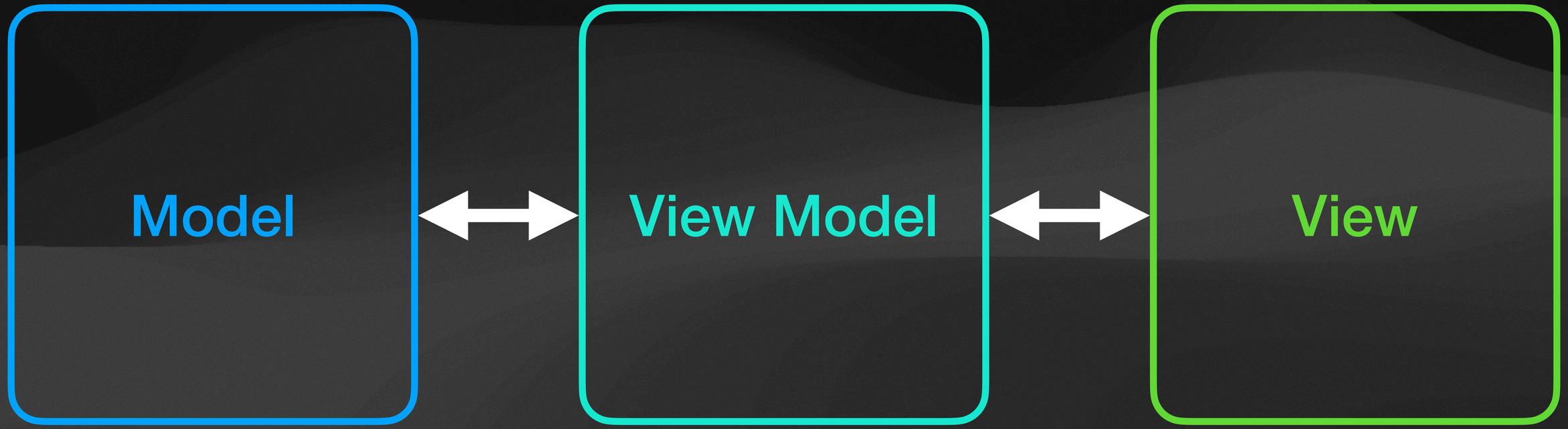
Manages the list of todos and handles modifications

TodoView

Displays the list of todos to the user

MVVM

How we'll use it



Course

Contains the actual course data

CoursesViewModel

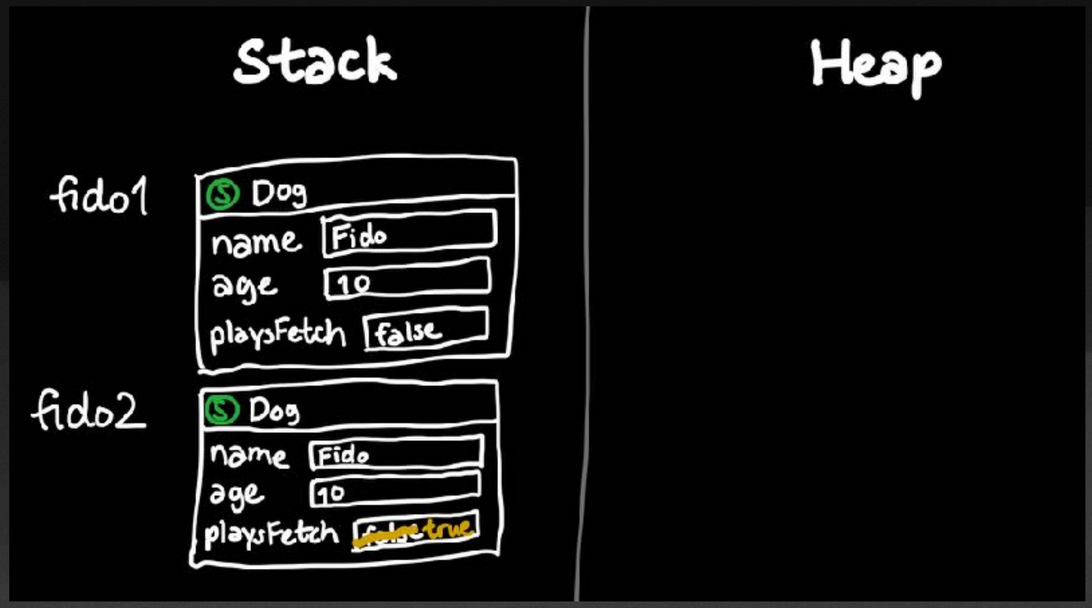
Manages favorited courses and filters them for the view

CoursesView

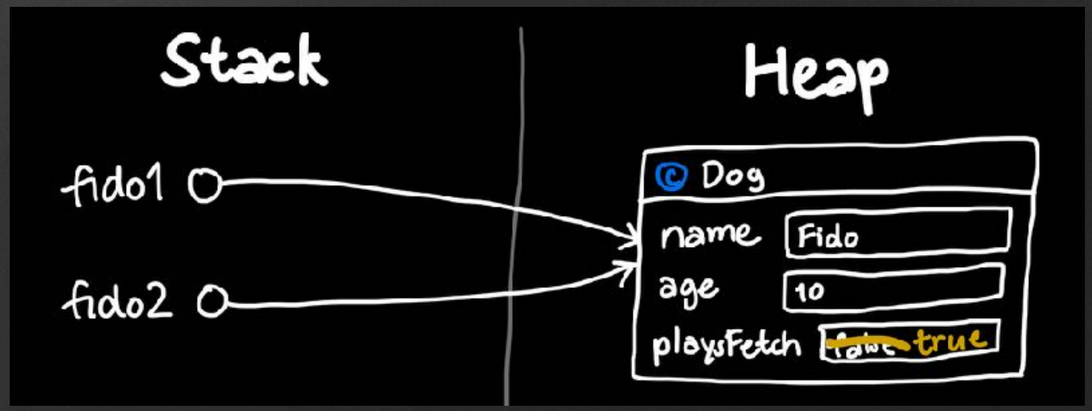
Displays the list of courses to the user

Refresher

Structs



Classes



@Observable

When we want to use a class's property as state, we need to tag the class as **@Observable**

```
@Observable class MyViewModel
```

@Bindable

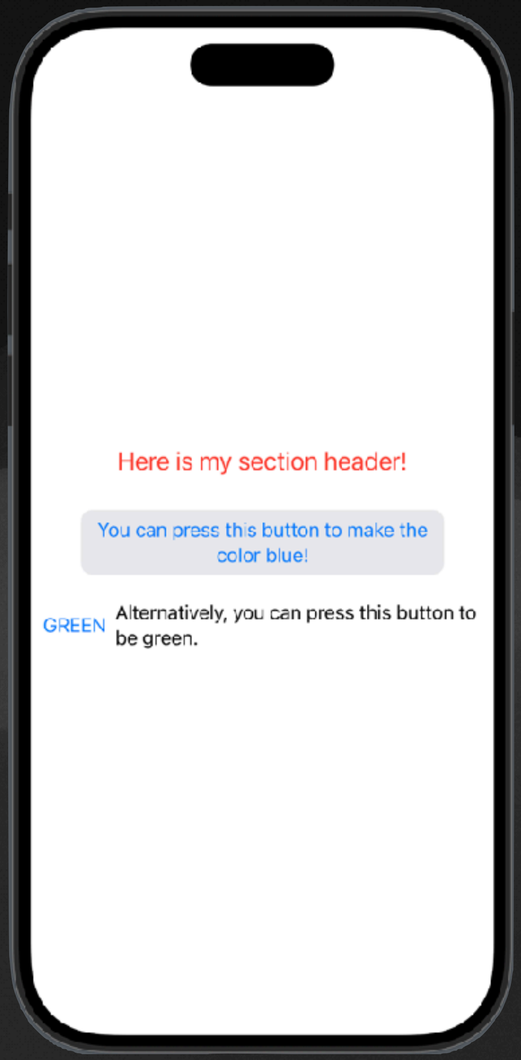
Bindings for Observables

- Similar to @Binding, we can pass a class marked with @Observable to a child view.
- Remember the relationship between @State (owns) and @Binding (allowed to change). The same relationship exists here.

ew

```
@Observable
class MySettings: Identifiable {
    var color: Color = .red
    let internships = 0
}


struct PropDrilling: View {
    @State var settings = MySettings()


    var body: some View {
        SectionView(settings: settings)
    }
}
```

```
struct SectionView: View {
    @Bindable var settings: MySettings

    var body: some View {
        VStack {
            Text("Here is my section header!")
                .font(.title2)
                .foregroundStyle(settings.color)
            Button {
                settings.color = .blue
            } label: {
                Text("You can press this button to make the color blue!")
            }
            .buttonStyle(.bordered)
            .padding()
            SwitchView(settings: settings)
        }
    }
}

struct SwitchView: View {
    @Bindable var settings: MySettings

    var body: some View {
        HStack {
            Button {
                settings.color = .green
            } label: {
                Text("GREEN")
            }
            Text("Alternatively, you can press this button to be green.")
        }
    }
}
```



"prop drilling"

@Environment

Pass an object/property to **all** subviews

```
@Observable
class MySettings: Identifiable {
    var color: Color = .red
    let internships = 0
}


struct PropDrilling: View {
    @State var settings = MySettings()


    var body: some View {
        SectionView()
            .environment(settings)
    }
}
```

```
struct SectionView: View {
    // Will crash if not present
    @Environment(MySettings.self) var settings

    var body: some View {
        VStack {
            Text("Here is my section header!")
                .font(.title2)
                .foregroundStyle(settings.color)
            Button {
                settings.color = .blue
            } label: {
                Text("You can press this button to make the color blue!")
            }
                .buttonStyle(.bordered)
                .padding()
            SwitchView()
        }
    }
}
```

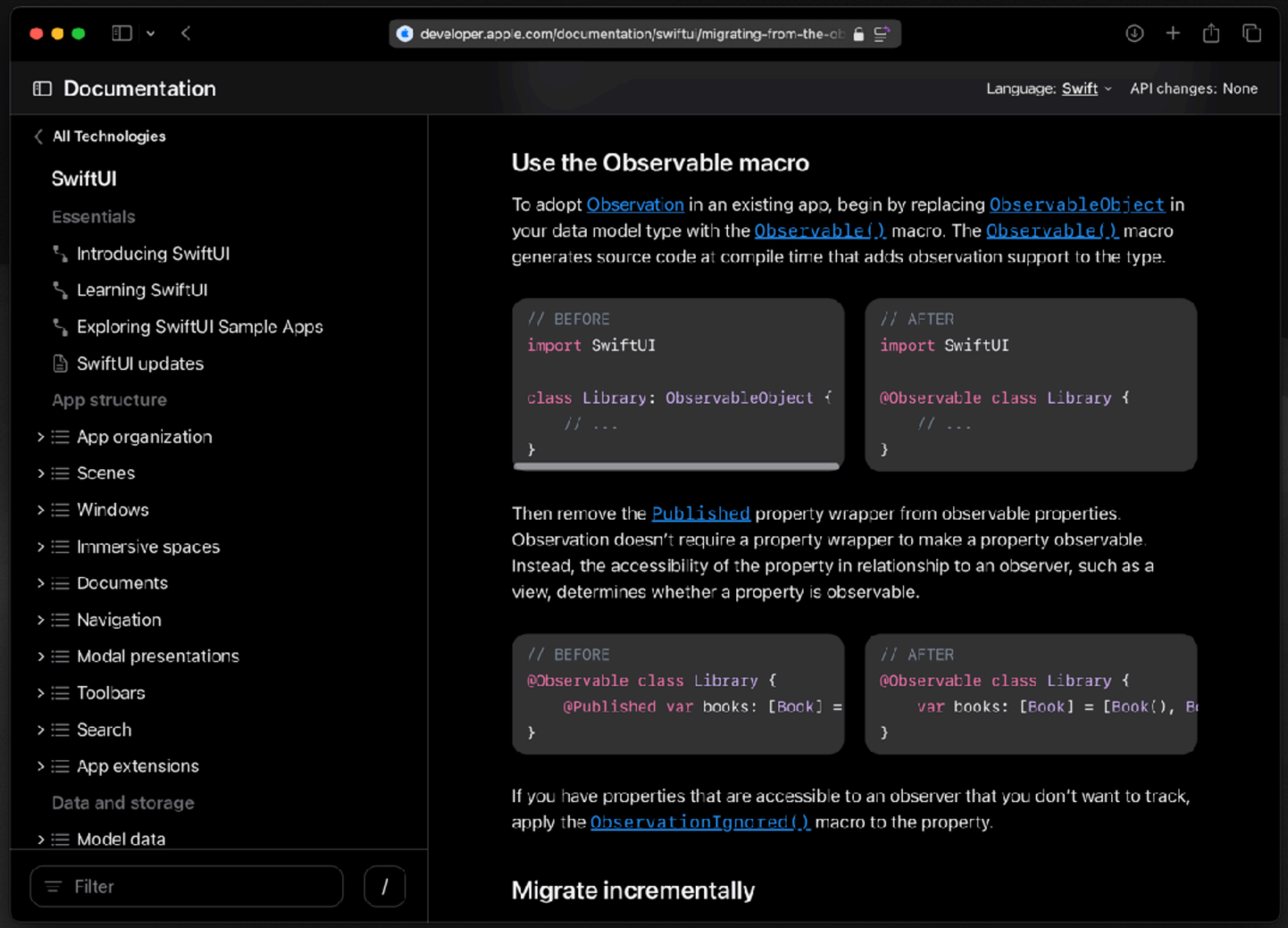
```
struct SwitchView: View {
    @Environment(MySettings.self) var settings

    var body: some View {
        HStack {
            Button {
                settings.color = .green
            } label: {
                Text("GREEN")
            }
            Text("Alternatively, you can press this button to be green.")
        }
    }
}
```

Note: using @Environment is a design decision. If you find yourself passing the same property/object to 2+ views (to be modified), consider using environment.

Brief aside

@ObservableObject

iOS 13-16



Recap

- **Navigation and modal presentation views** let us organize multiple screens
- **Model-view-view model** enables separation of concerns
- **@Observable** lets us manage state in classes

Homework 2

Trivia Game

- Will be released **Wednesday, 2/19**
- Due on **Wednesday, 3/19**
- Focuses on **lectures 3-5**
- [details pending]