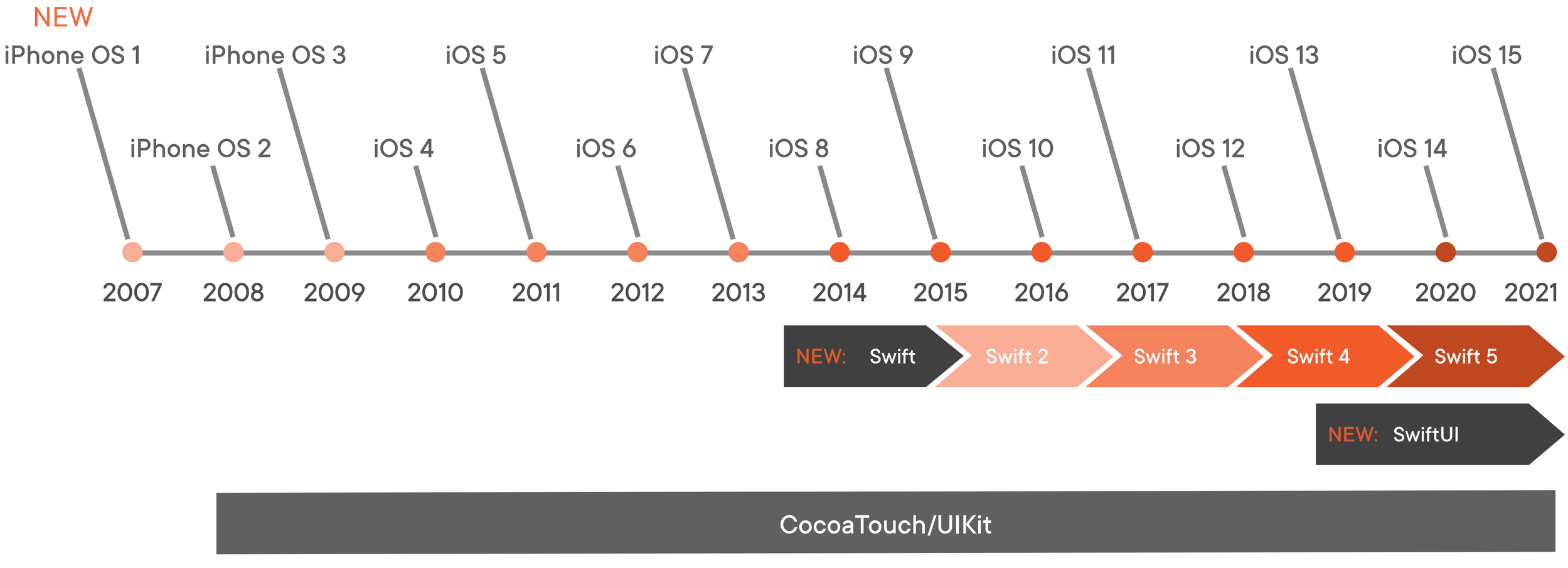
Integrating SwiftUI and UIKit

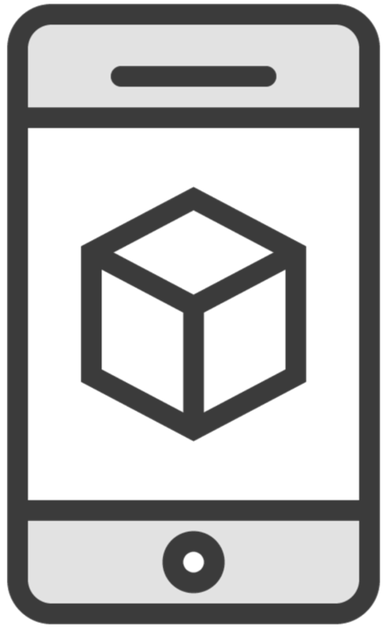
Andrew Bancroft

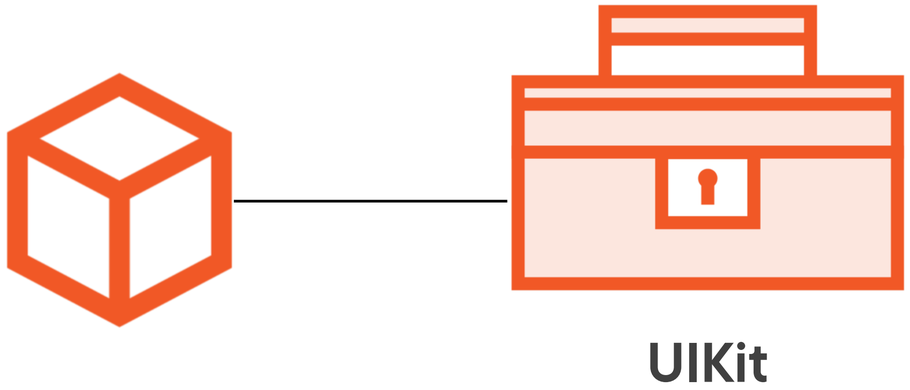
@andrewcbancroft www.andrewcbancroft.com

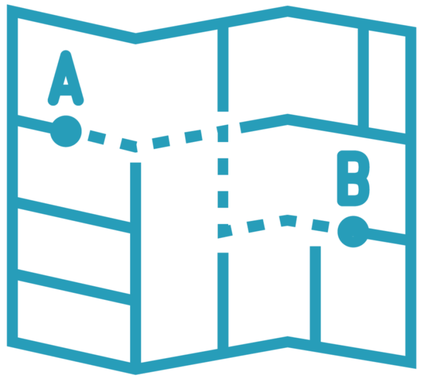


SwiftUI is relatively new and still in the early phases of adoption.

You’re likely to come across UIKit code somewhere along the way in your Apple Developer career.




UIKit



Migrating to SwiftUI

Sometimes the migration path is a little more gradual.

UIKit also has some user interface components that SwiftUI *doesn't* have.


Some of our renovation projects are falling behind...

10:00

Home Edit

Lounge

Work Quality

★ ☆ ☆ ☆ View Inspection Log

Punch List

- ⊛ Remodel fireplace
- ☐ Install new flooring
- ☐ Replace light fixtures
- ☐ Paint walls
- ☐ Hang new artwork

Budget

On Budget

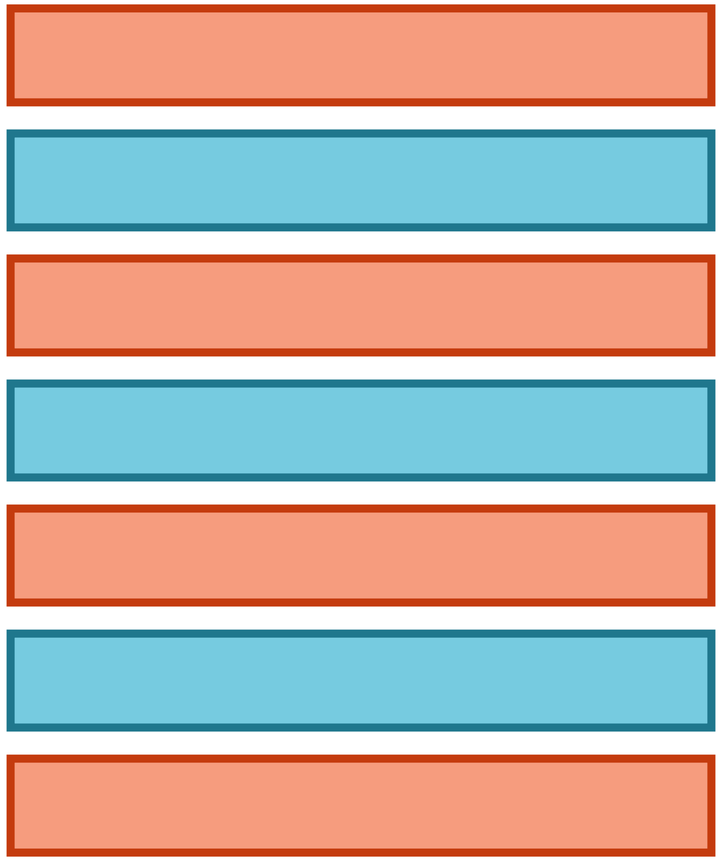
Amount Allocated:	$5,000.00
Spent to-date:	$1,246.00
Amount remaining:	**$3,754.00**

UIKit

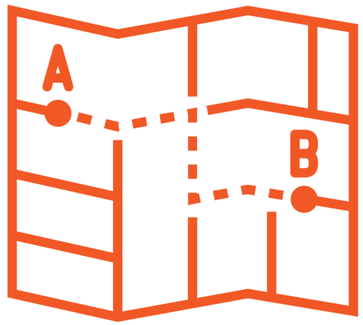
UIKit

Integrating SwiftUI and UIKit

Integrating SwiftUI and UIKit



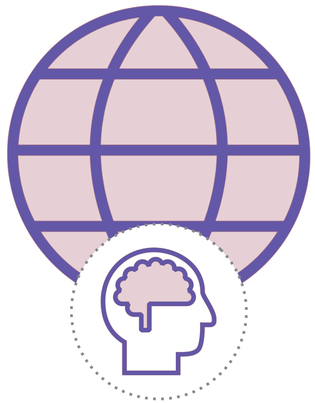
SwiftUI



Being able to integrate SwiftUI and UIKit makes it much easier to develop a migration plan from UIKit to SwiftUI gradually over time.

SwiftUI

UIKit



SwiftUI + UIKit

You do not need to become a UIKit developer before being able to integrate SwiftUI and UIKit.

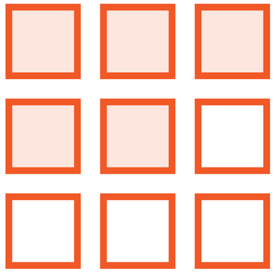
How do we represent a UIKit View Controller in a SwiftUI app?

Representing UIKit ViewControllers in SwiftUI

Creating a View Hierarchy in SwiftUI

Stack

List



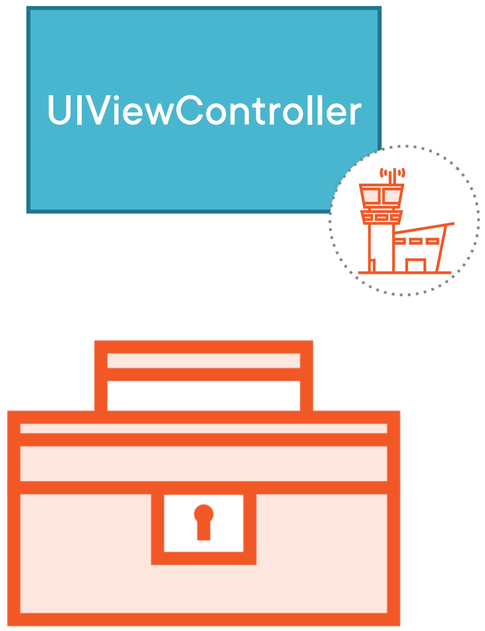
Grid

Form

Creating a View Hierarchy in UIKit

UIKit

Creating a View Hierarchy in UIKit



TVML
TVMLKit
TVUIKit
UIKit
Essentials
About App Development w...
Protecting the User's Priva...
App Structure
App and Environment
Documents, Data, and Past...
Resource Management
App Extensions
Interprocess Communication
Mac Catalyst
User Interface
Views and Controls
View Controllers
Essentials
Managing Content in Yo...
UIKit Catalog: Creating a...
Content View Controllers
Displaying and Managin...
Showing and Hiding Vie...
UIViewController
UITableViewController
UICollectionViewControll...
UIContentContainer
Container View Controllers
Creating a Custom Cont...
UISplitViewController
UINavigationController
UINavigationBar
UINavigationItem
UITabBarController
UITabBar
UITabBarItem
UIPageViewController
Presentation Management
Disabling the Pull-Down...
UIPresentationController
Search Interface
UISearchContainerView...
UISearchController
UISearchBar
UISearchResultsUpdating
Filter

Class

UIViewController

An object that manages a view hierarchy for your UIKit app.

Declaration

```
class UIViewController : UIResponder
```

Overview

The `UIViewController` class defines the shared behavior that is common to all view controllers. You rarely create instances of the `UIViewController` class directly. Instead, you subclass `UIViewController` and add the methods and properties needed to manage the view controller's view hierarchy.

A view controller's main responsibilities include the following:

- Updating the contents of the views, usually in response to changes to the underlying data.
- Responding to user interactions with views.
- Resizing views and managing the layout of the overall interface.
- Coordinating with other objects—including other view controllers—in your app.

A view controller is tightly bound to the views it manages and takes part in handling events in its view hierarchy. Specifically, view controllers are `UIResponder` objects and are inserted into the responder chain between the view controller's root view and that view's superview, which typically belongs to a different view controller. If none of the view controller's views handle an event, the view controller has the option of handling the event or passing it along to the superview.

View controllers are rarely used in isolation. Instead, you often use multiple view controllers, each of which owns a portion of your app's user interface. For example, one view controller might display a table of items while a different view controller displays the selected item from

Language

Swift | Objective-C

Availability

iOS 2.0+

Mac Catalyst 13.0+

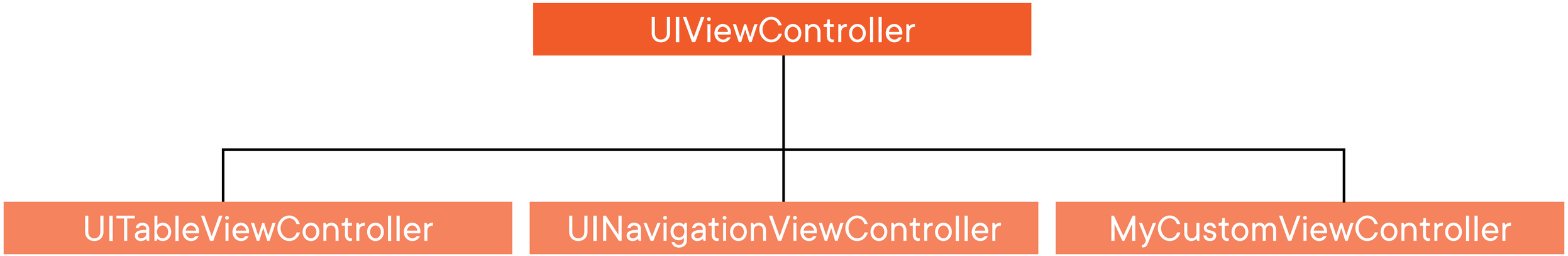
tvOS 9.0+

Framework

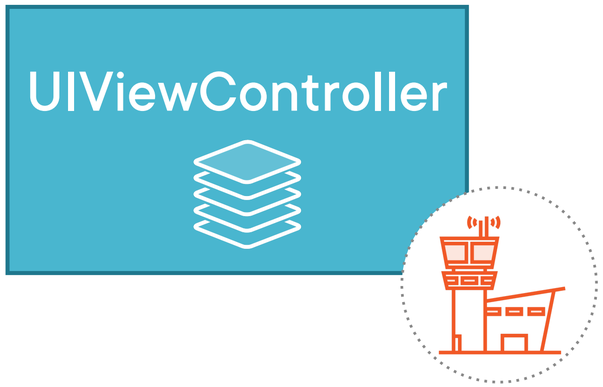
UIKit

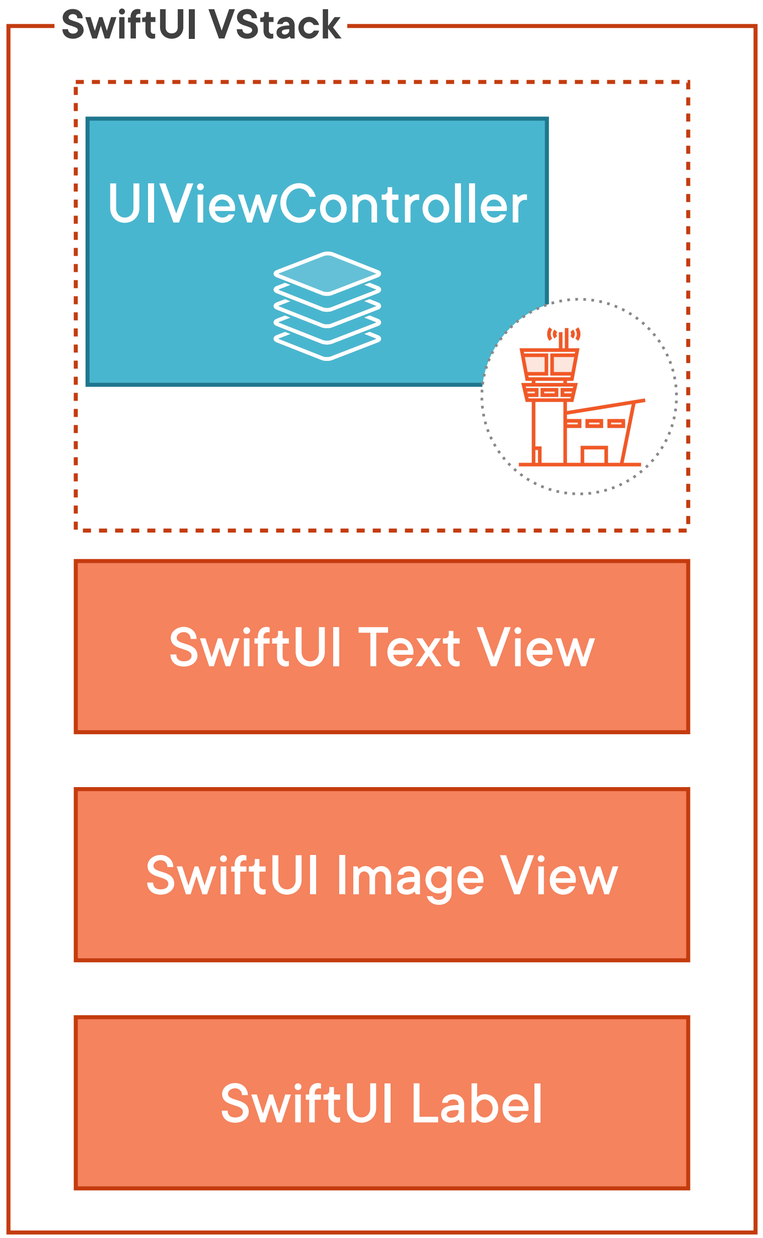
On This Page

Declaration
Overview
Topics
Relationships


UIViewController
UITableViewController
UINavigationViewController
MyCustomViewController

In UIKit, UIViewController and its subclasses are the glue that hold everything together.


UIViewController


SwiftUI VStack
UIViewController
SwiftUI Text View
SwiftUI Image View
SwiftUI Label

Swift
SwiftUI
TVML
TVMLKit
TVUIKit
UIKit
Essentials
About App Development w...
Protecting the User's Priva...
App Structure
App and Environment
Documents, Data, and Past...
Resource Management
App Extensions
Interprocess Communication
Mac Catalyst
User Interface
Views and Controls
View Controllers
View Layout
Appearance Customization
Animation and Haptics
Windows and Screens
User Interactions
Touches, Presses, and Ges...
Drag and Drop
Pointer Interactions
Pencil Interactions
Focus-based Navigation
Menus and Shortcuts
Accessibility for iOS and tv...
Graphics, Drawing, and Prin...
Images and PDF
Drawing
Printing
Text
Text Display and Fonts
Text Storage
Keyboards and Input
Handwriting Recognition
Deprecated
Deprecated Symbols
Classes
UIPrintServiceExtension
UIPrinterDestination
Protocols
Filter

Class

UIPageViewController

A container view controller that manages navigation between pages of content, where a child view controller manages each page.

Declaration

```
class UIPageViewController : UIViewController
```

Overview

Page view controller–navigation can be controlled programmatically by your app or directly by the user using gestures. When navigating from page to page, the page view controller uses the transition that you specify to animate the change.

> **Important**
>
> In tvOS, the `UIPageViewController` class provides only a way to swipe between full-screen content pages. Unlike in iOS, a user cannot interact with or move focus between items on each page.

When defining a page view controller interface, you can provide the content view controllers one at a time (or two at a time, depending upon the spine position and double-sided state) or as-needed using a data source. When providing content view controllers one at a time, you use the `setViewControllers(_:direction:animated:completion:)` method to set the current content view controllers. To support gesture-based navigation, you must provide your view controllers using a data source object.

The data source for a page view controller is responsible for providing the content view controllers on demand and must conform to the `UIPageViewControllerDataSource` protocol. The delegate object—an object that conforms to the `UIPageViewController`

Language

Swift | Objective-C

Availability

iOS 5.0+

Mac Catalyst 13.0+

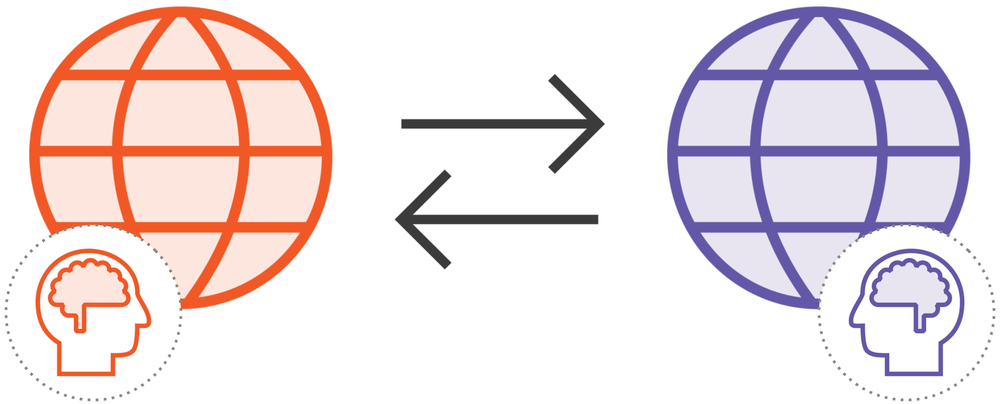
tvOS 9.0+

Framework

UIKit

On This Page

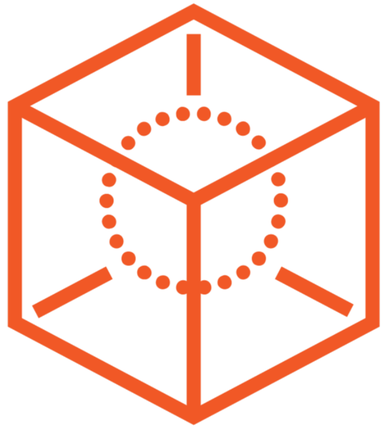
Declaration
Overview
Topics
Relationships



SwiftUI

UIKit

UIViewController



UIViewControllerRepresentable

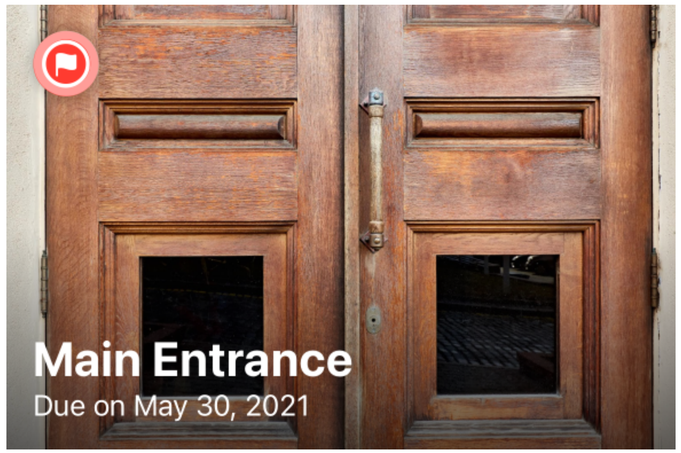
UIViewControllerRepresentable

A protocol that describes how to represent a UIViewController from UIKit as a SwiftUI View

Create a new struct that conforms to the UIViewControllerRepresentable protocol

Implement its required properties and methods

UIPageViewController

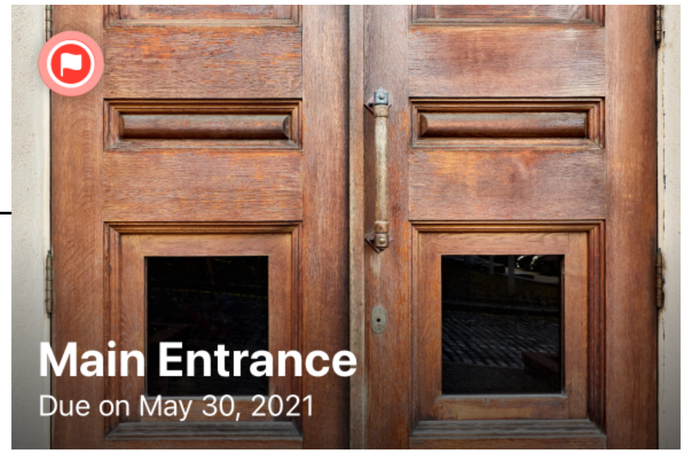


A View has one critical job:

Accurately represent the data of an application
at all times in what gets displayed on the screen.

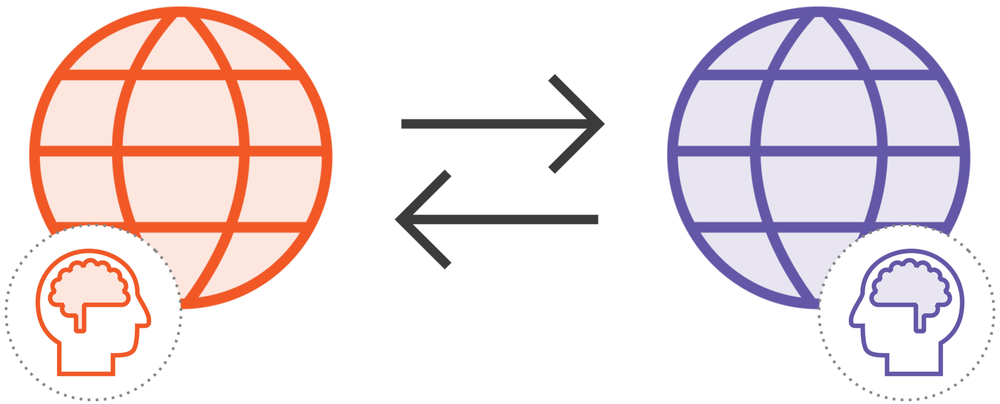
UIPageViewController

SwiftUI View



How do we make SwiftUI Views compatible with UIKit?

Making SwiftUI Views Compatible with UIKit

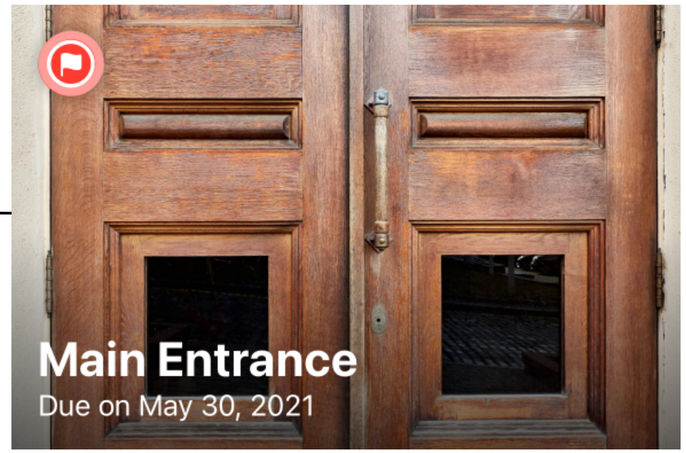


SwiftUI

UIKit

PageViewController

FlaggedProjectCard

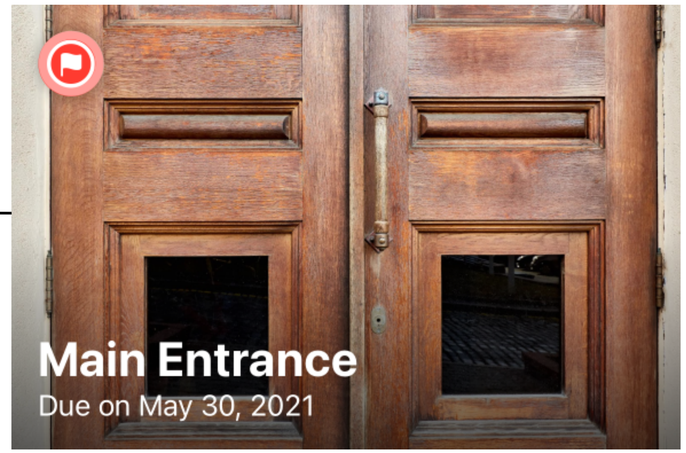


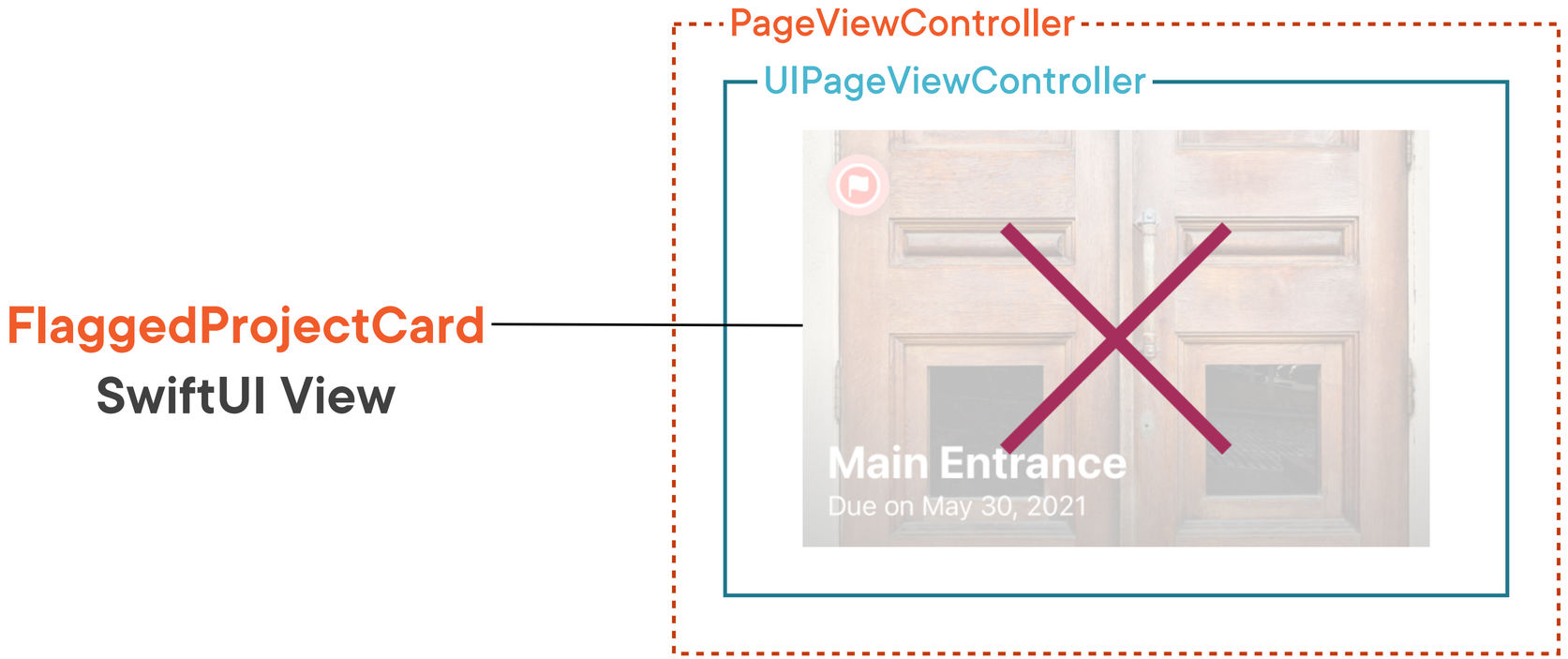
FlaggedProjectCard

SwiftUI View

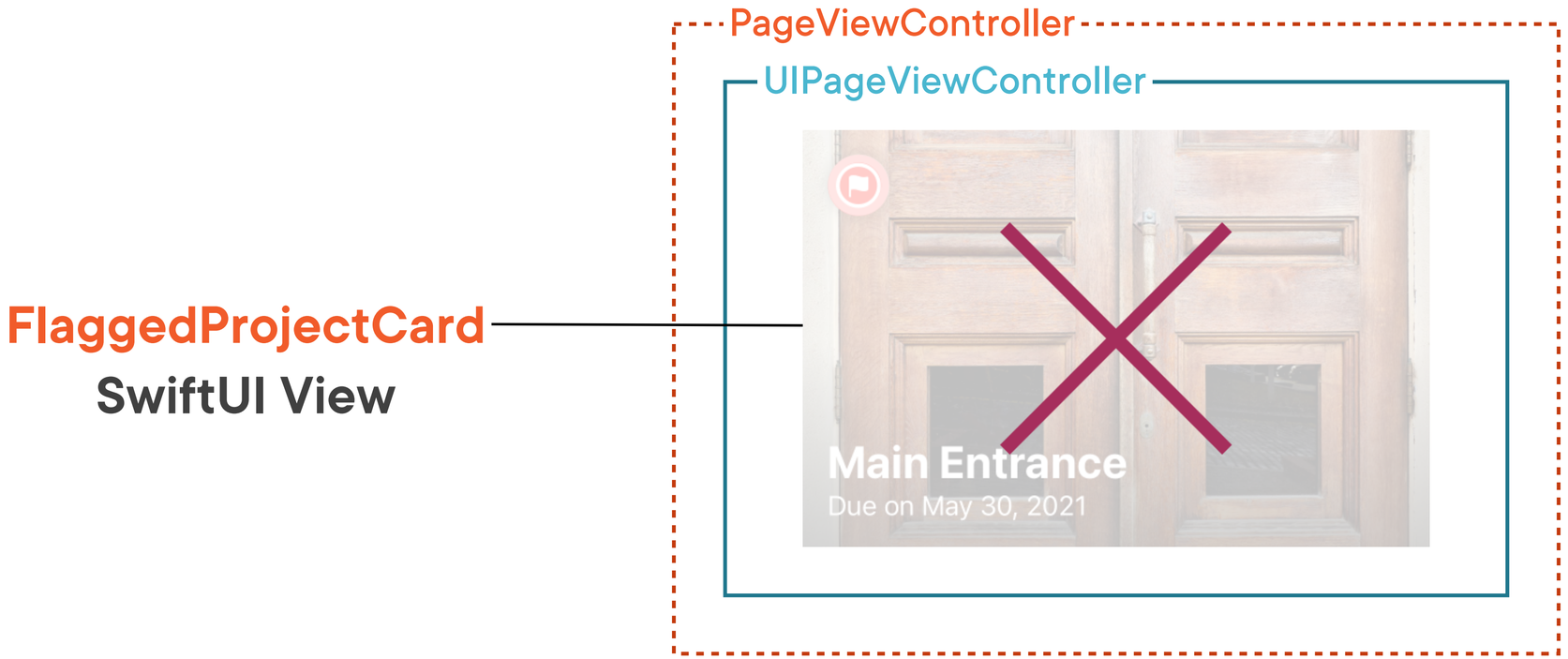
PageViewController

UIPageViewController

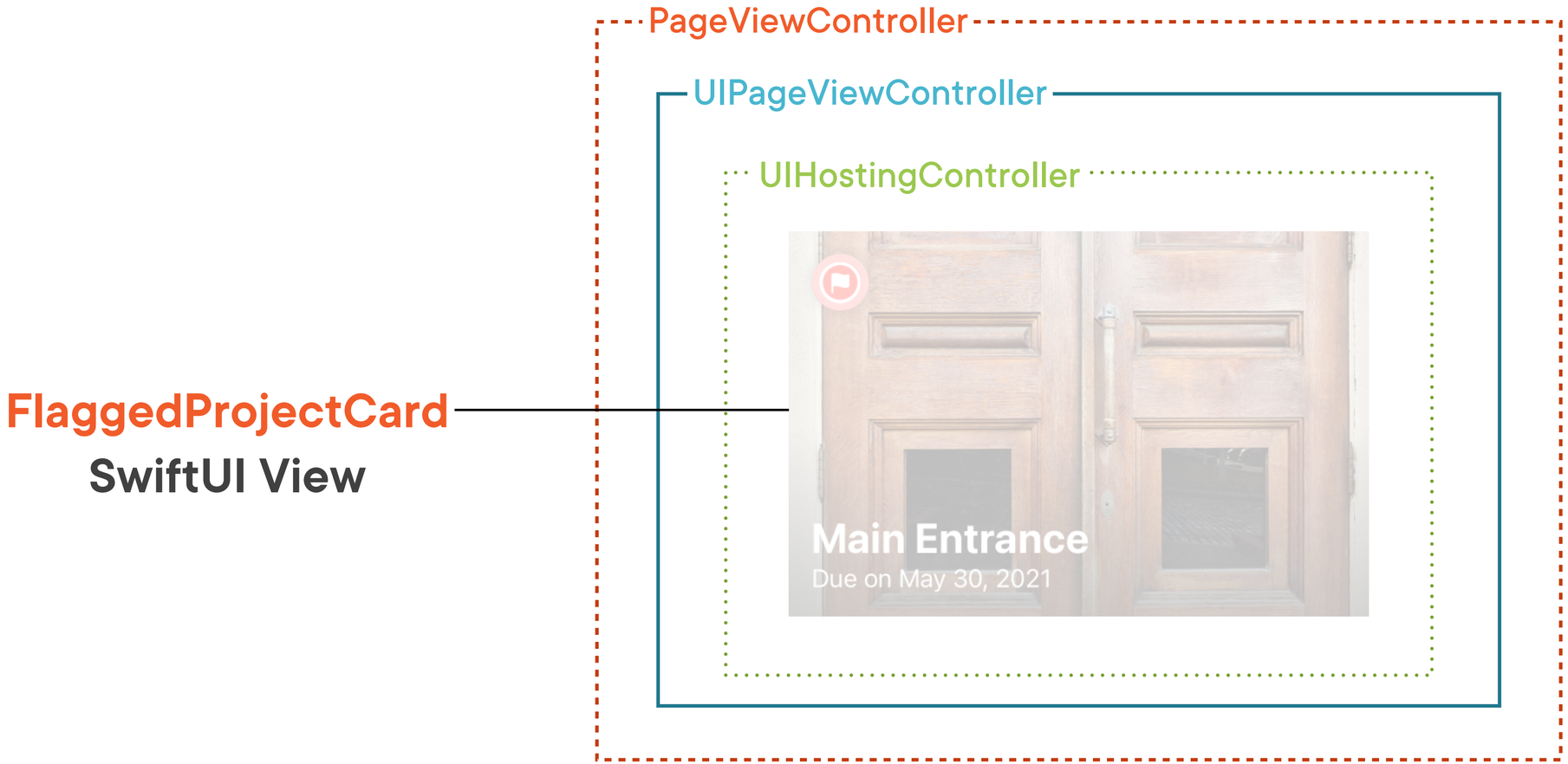




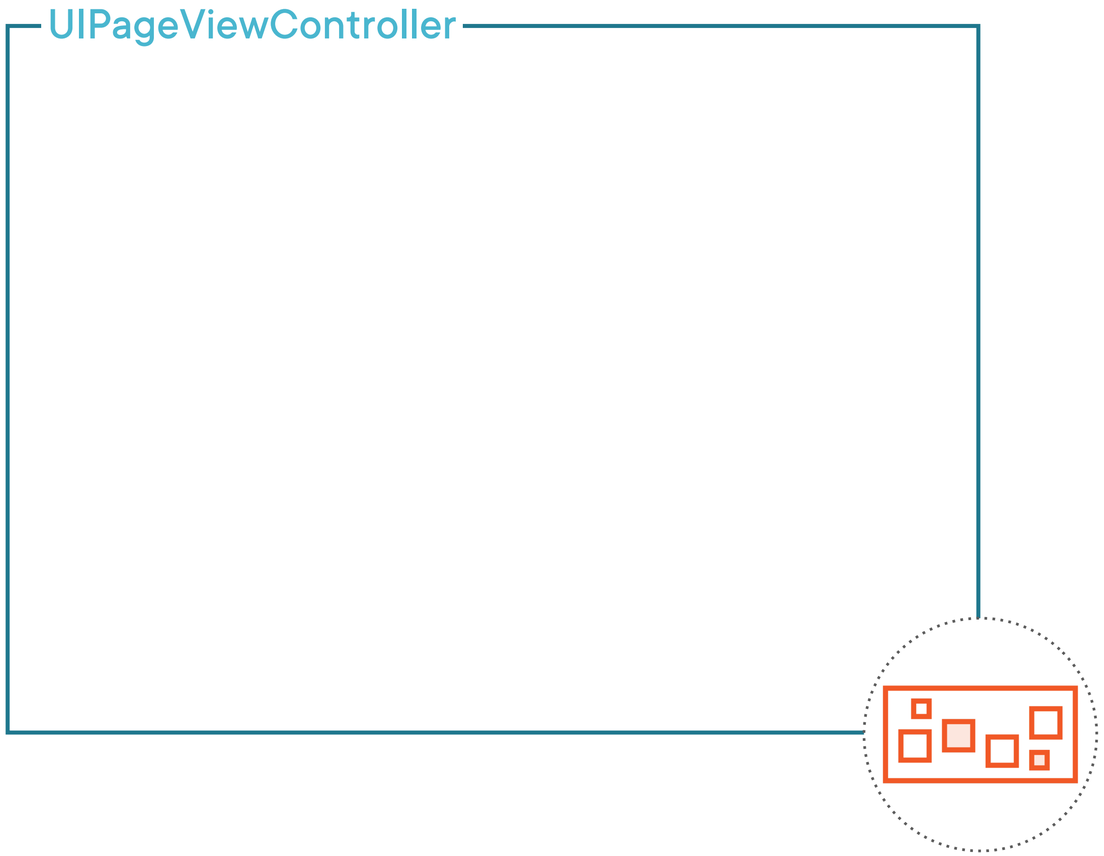
UIPageViewController doesn't manage SwiftUI Views... It manages a UIKit view hierarchy.



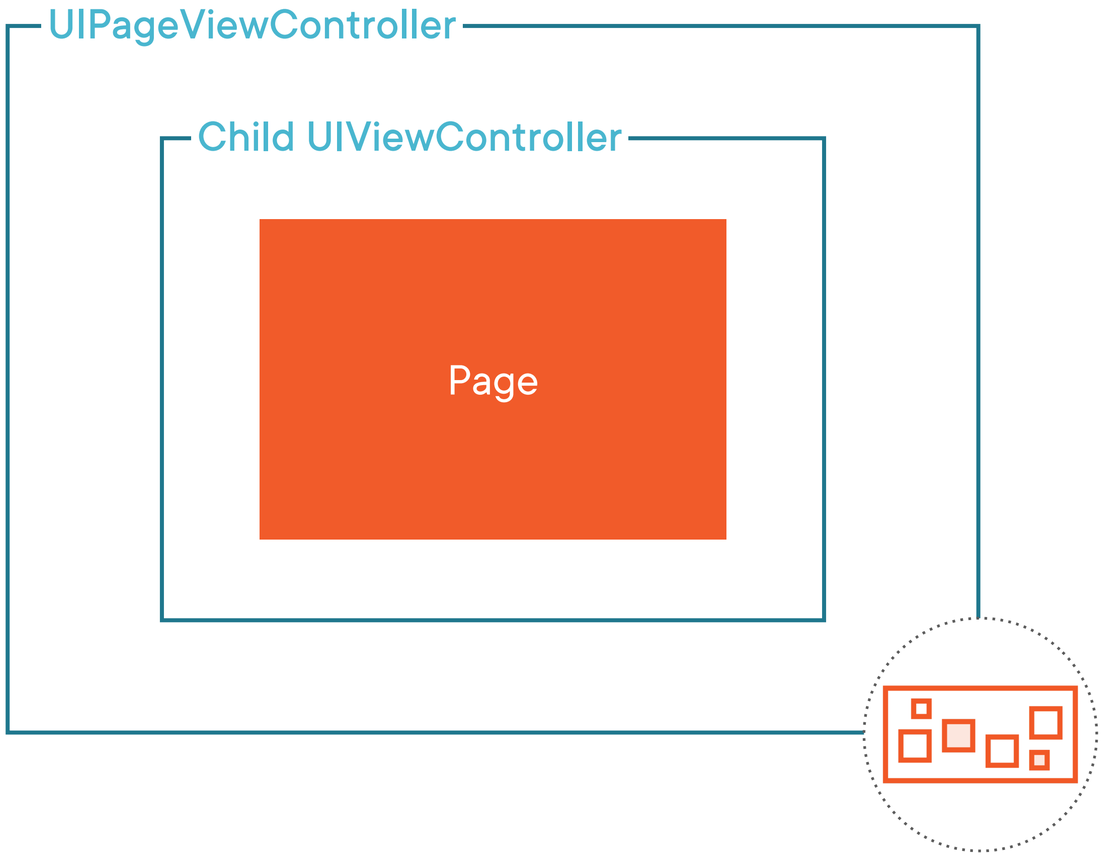
We need a way to integrate FlaggedProjectCard into a component that UIPageViewController can manage.



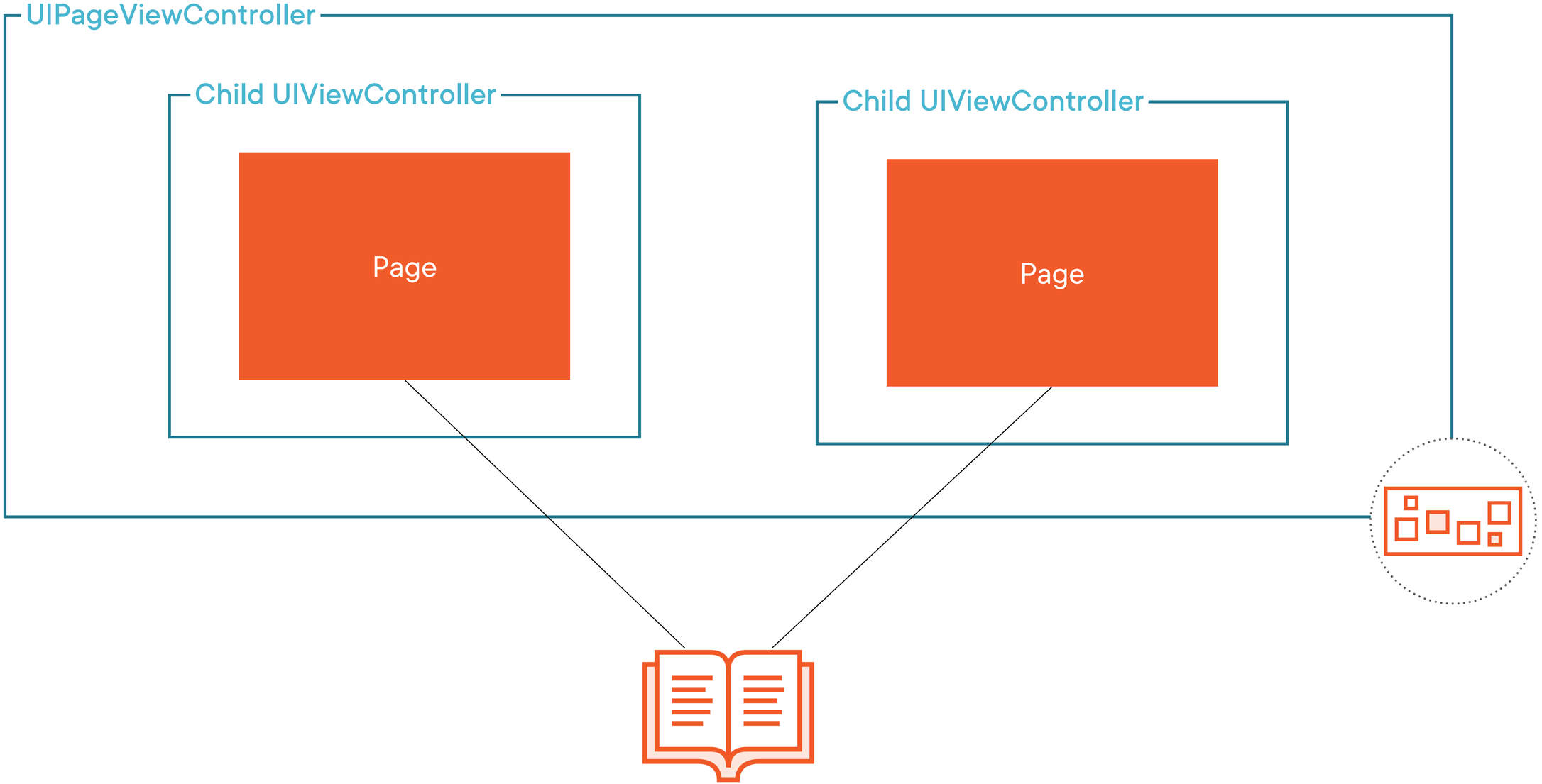
We need a way to integrate FlaggedProjectCard into a component that UIPageViewController can manage.

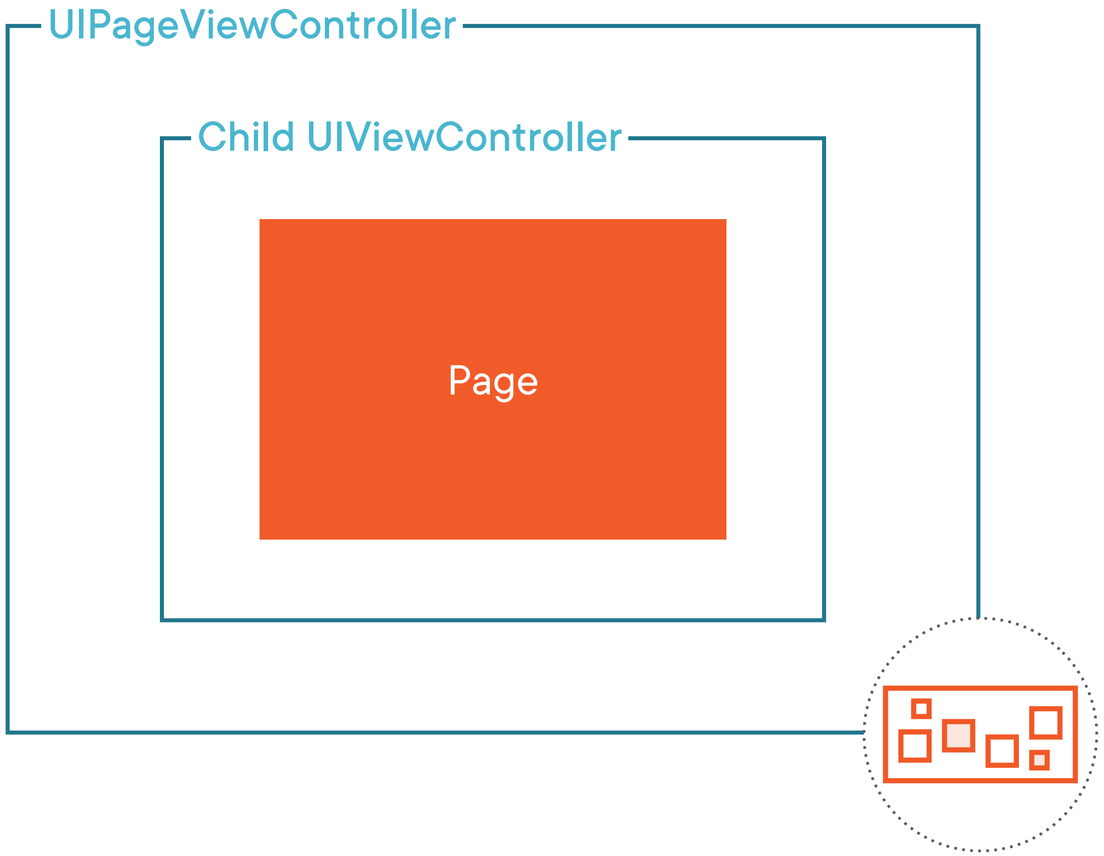


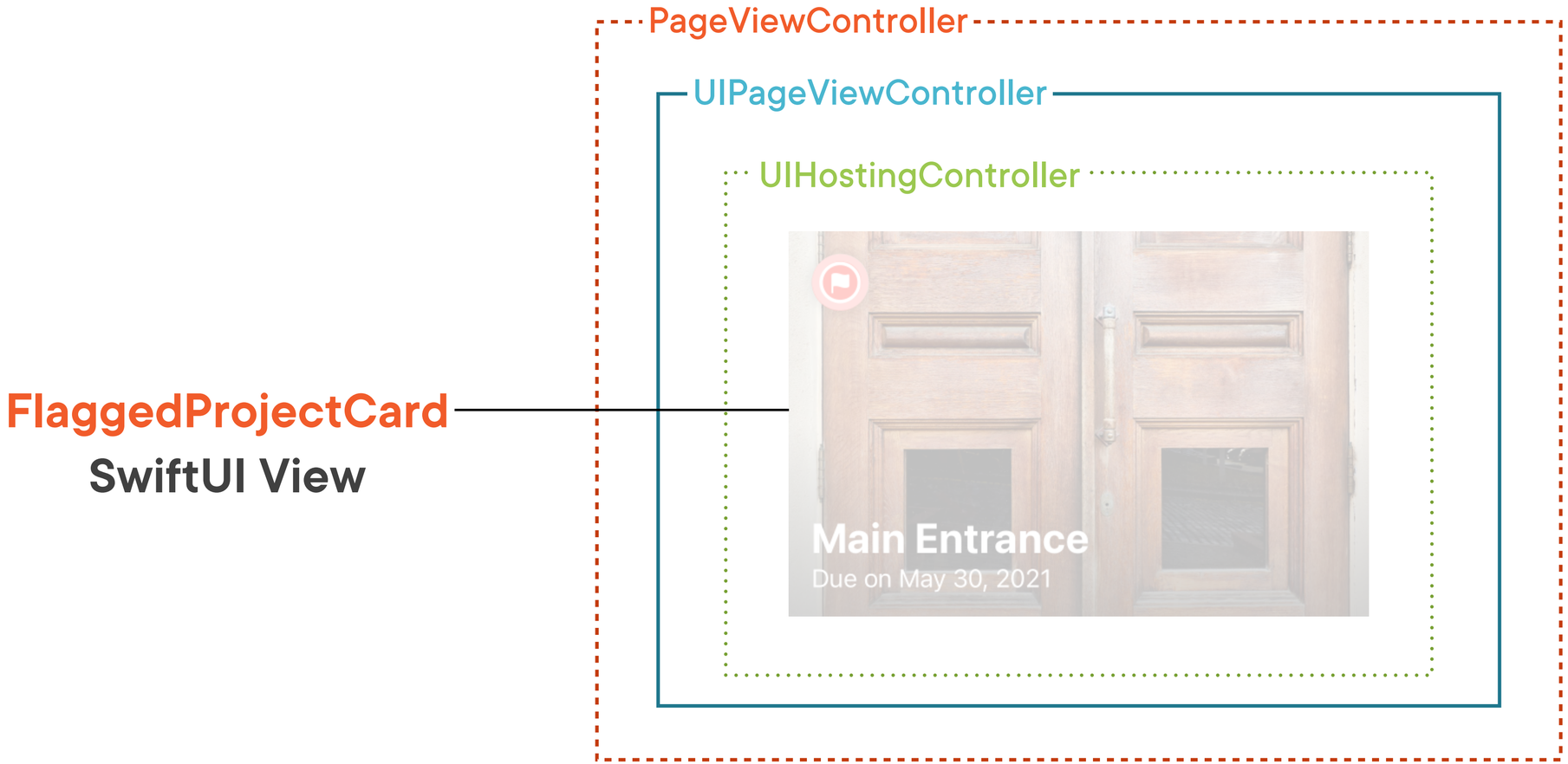
UIPageViewController acts as a container that manages navigation between pages of content.



The actual content... a given page that it displays, is managed by a separate child view controller.


UIPageViewController
Child UIViewController
Page
Child UIViewController
Page

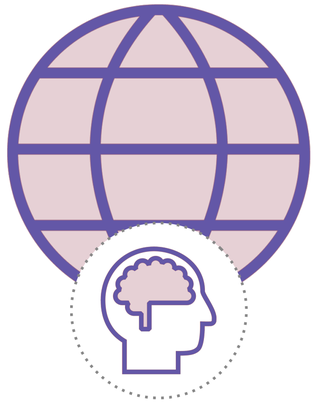

UIPageViewController
Child UIViewController
Page



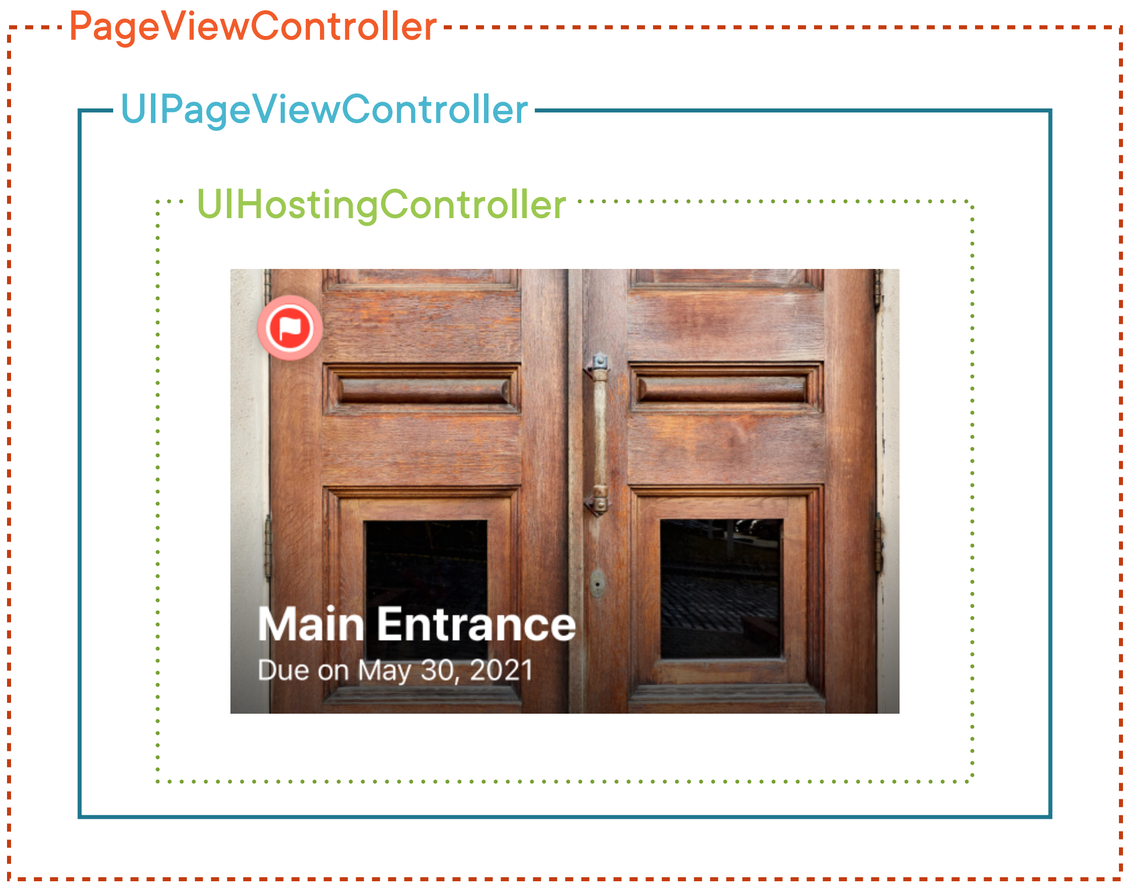
We need a way to integrate FlaggedProjectCard into a component that UIPageViewController can manage.

SwiftUI

UIKit



SwiftUI + UIKit

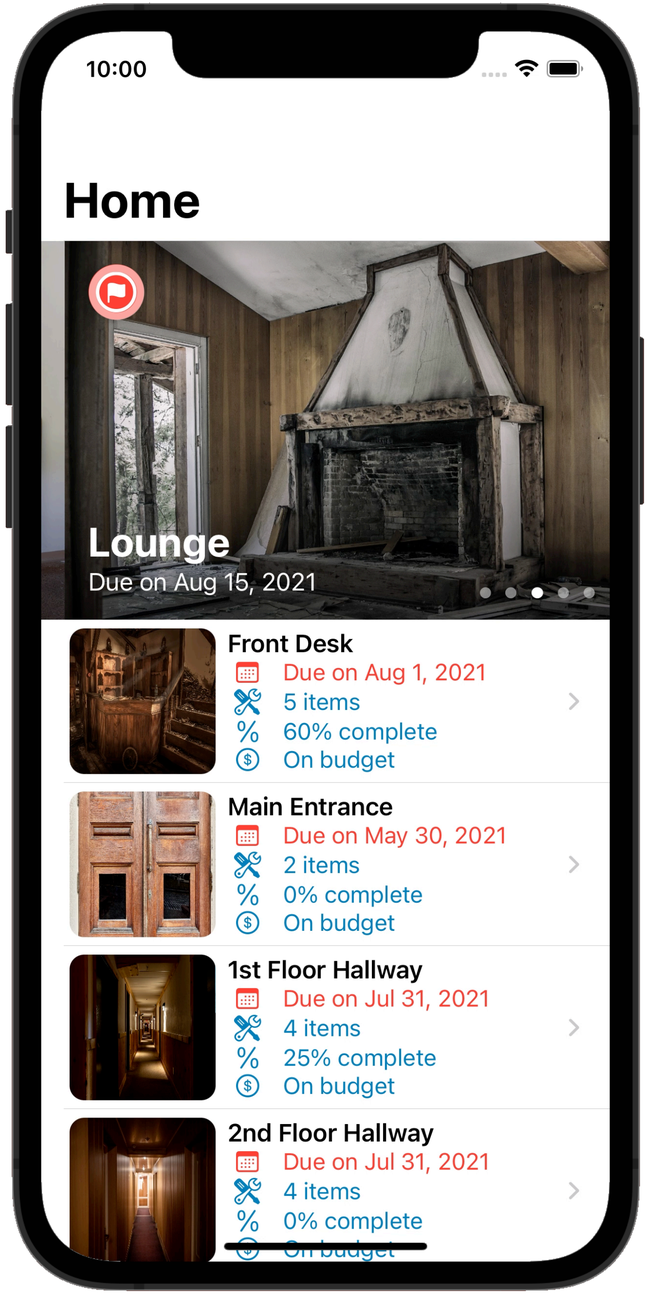


UIViewControllerRepresentable and UIHostingController are only for UIViewControllers.

UIKit

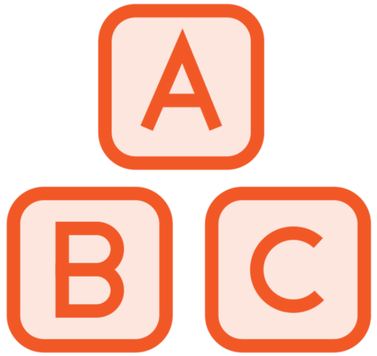
UIView

UIKit


10:00
Home
Lounge
Due on Aug 15, 2021
Front Desk
Due on Aug 1, 2021
5 items
60% complete
On budget
Main Entrance
Due on May 30, 2021
2 items
0% complete
On budget
1st Floor Hallway
Due on Jul 31, 2021
4 items
25% complete
On budget
2nd Floor Hallway
Due on Jul 31, 2021
4 items
0% complete

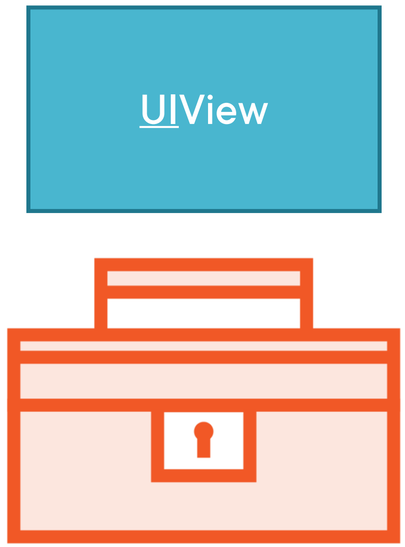
How do we represent a UIView in SwiftUI?

Representing UIKit Views in SwiftUI



SwiftUI View

UIKit



UIKit

UIViewRepresentable

A protocol that describes a way to “wrap around” a UIView so that you can integrate it into your SwiftUI View hierarchy.

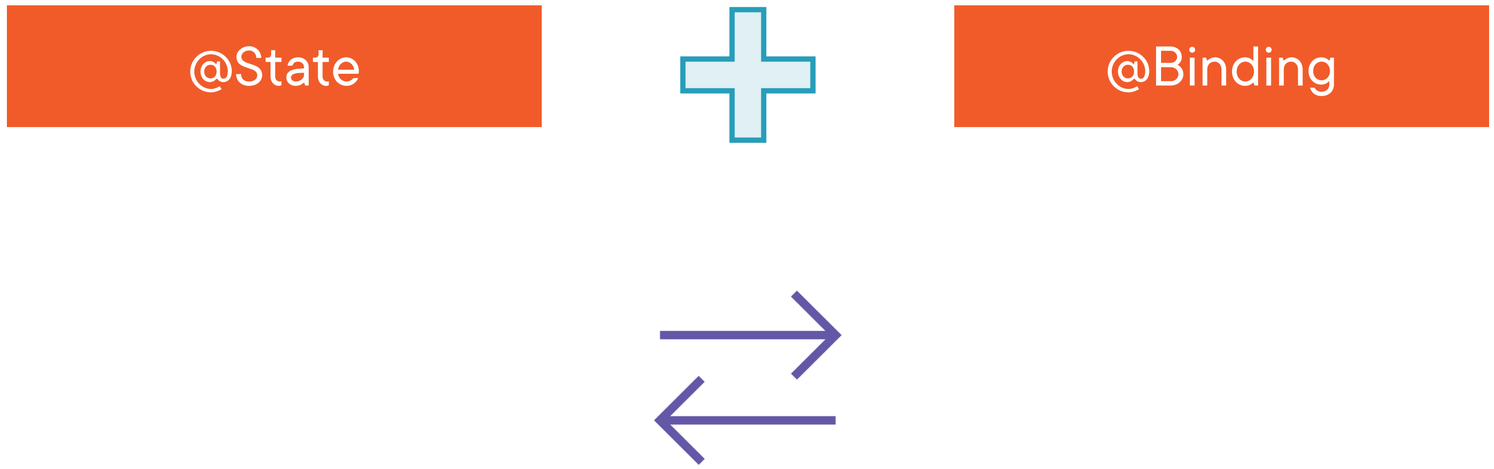
How do we share state between SwiftUI and UIKit?

Sharing State Between SwiftUI and UIKit

How do we coordinate user interaction between UIViewControllers and SwiftUI?

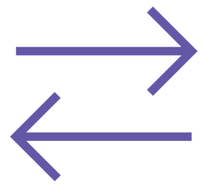
Coordinating User Interaction Between UIViewControllers and SwiftUI

Handling User Interaction with SwiftUI



Handling User Interaction with SwiftUI

@State + @Binding



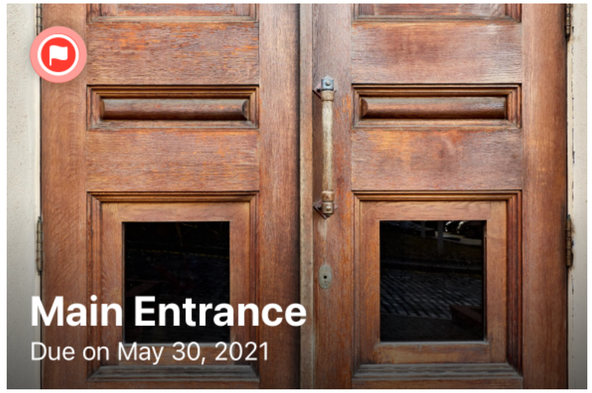
.gesture() View modifier

Child UIViewController

Before

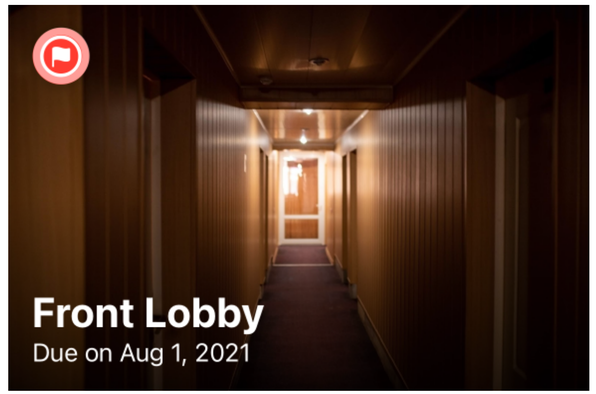
UIPageViewController

Child UIViewController



Current

Child UIViewController



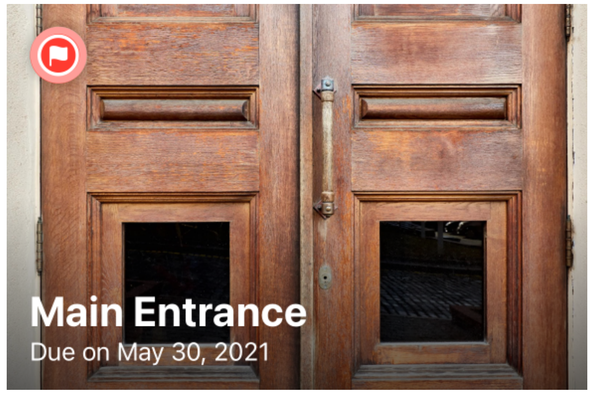
After

UIPageViewController

Child UIViewController

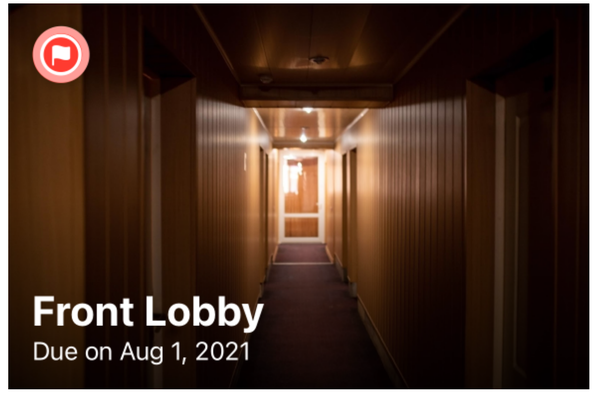
Before

Child UIViewController



Current

Child UIViewController



After

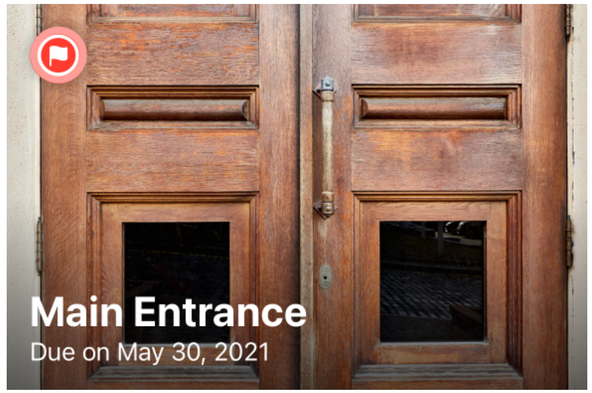
Knows that it needs the before and after pages,
but doesn't take on the responsibility of supplying that data to itself directly...

Child UIViewController

Before

UIPageViewController

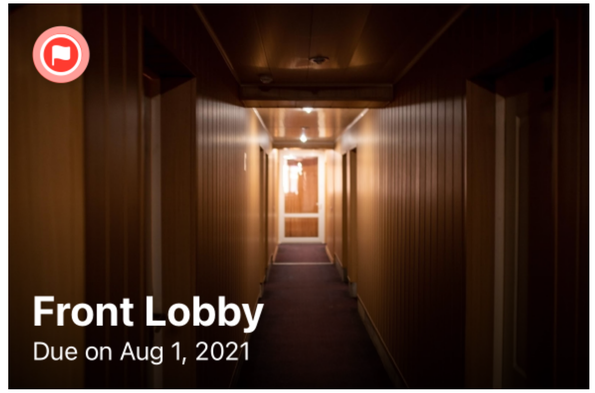
Child UIViewController



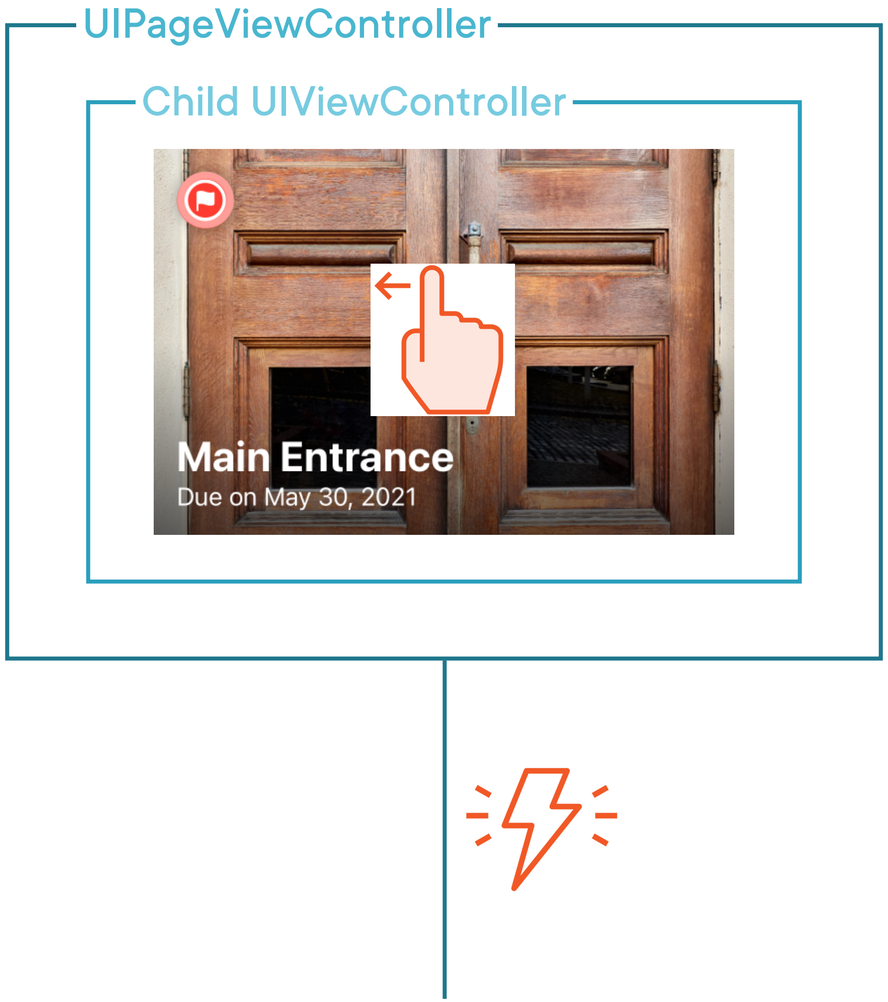
Current

Data Source

Child UIViewController



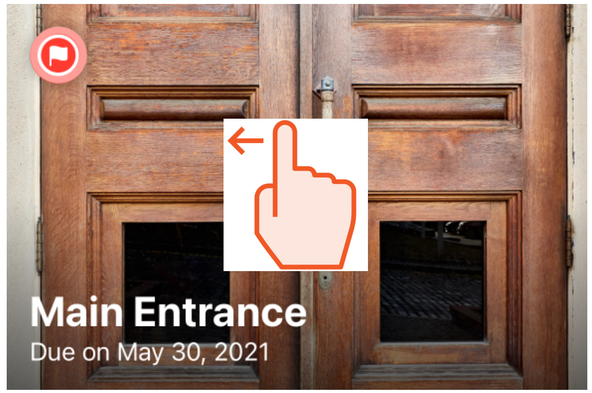
After



UIPageViewController will **detect** gestures and initiate the transition to the appropriate page, but it does not take on the responsibility of defining what should happen next...

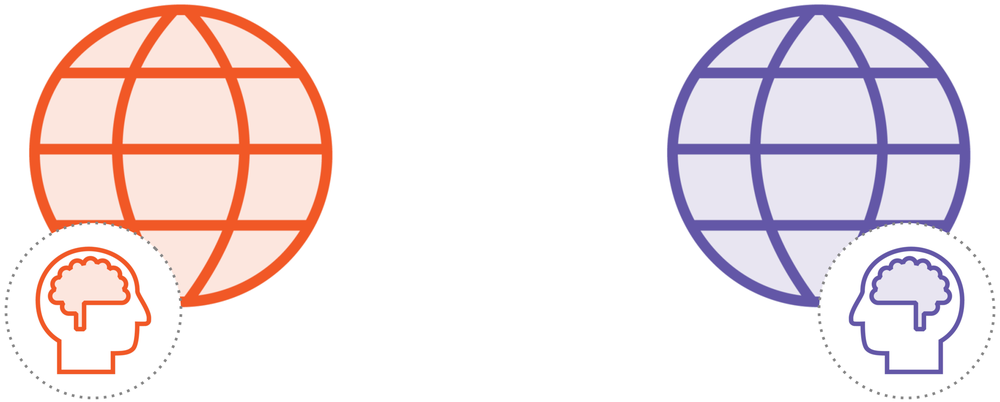
UIPageViewController

Child UIViewController



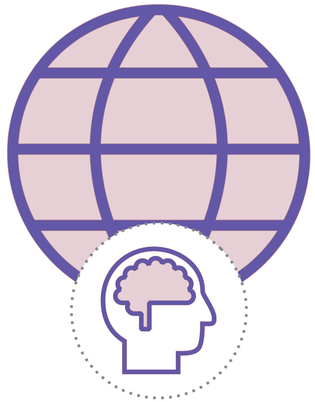
Delegate

Implements logic for “what happens next”



UIViewControllerRepresentable

UIViewRepresentable



UIViewControllerRepresentable

UIViewRepresentable

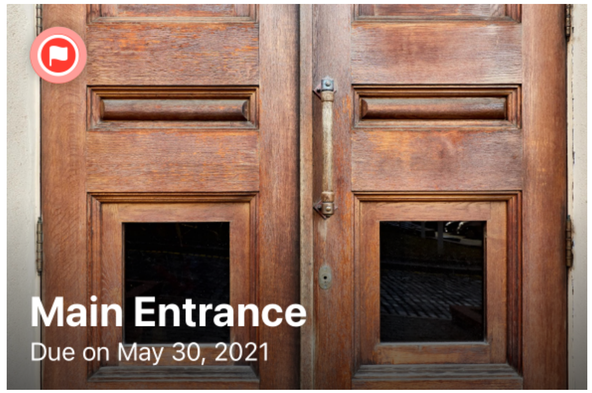
Is there an additional component that can bridge the two worlds when it comes to user interaction?

Child UIViewController

Before

UIPageViewController

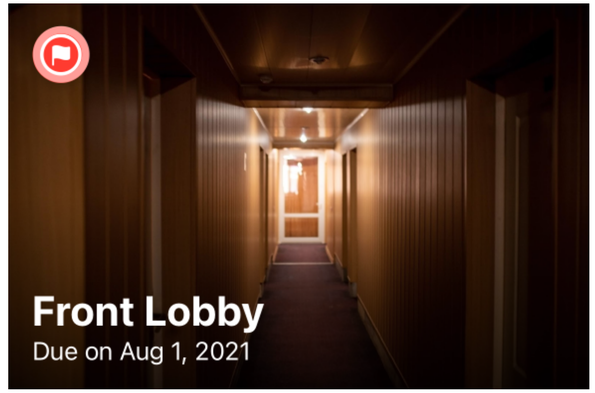
Child UIViewController



Current

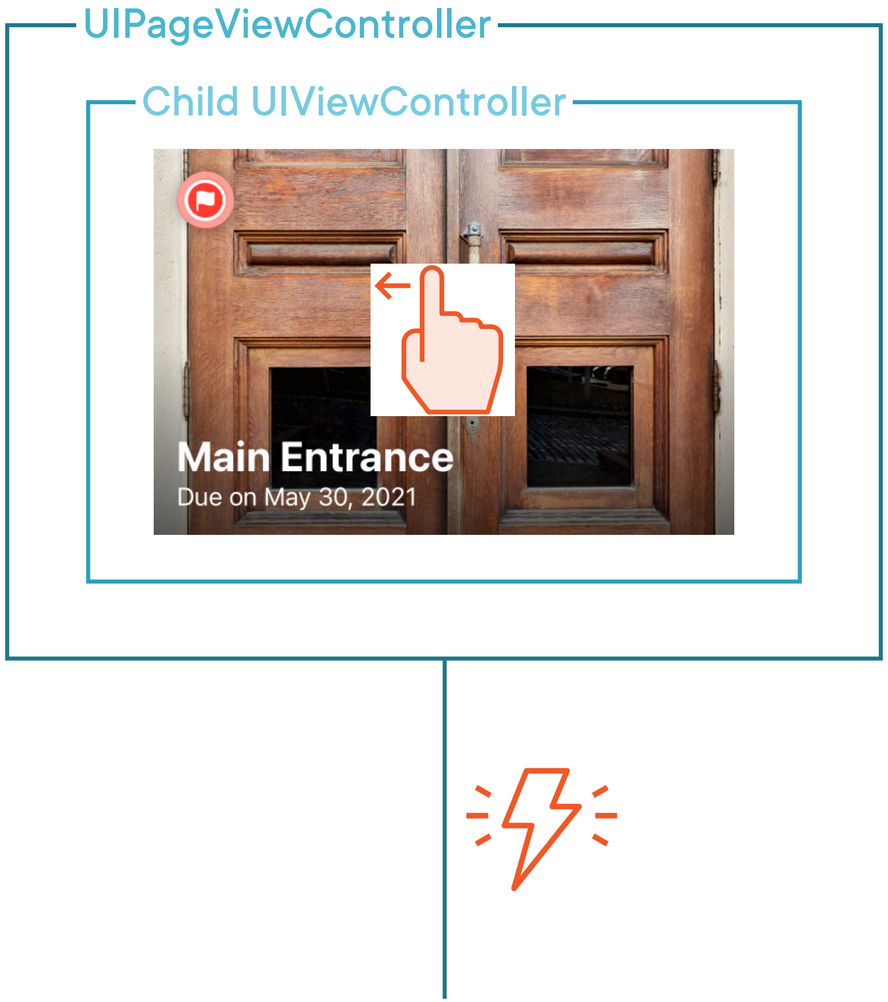
Data Source

Child UIViewController



After

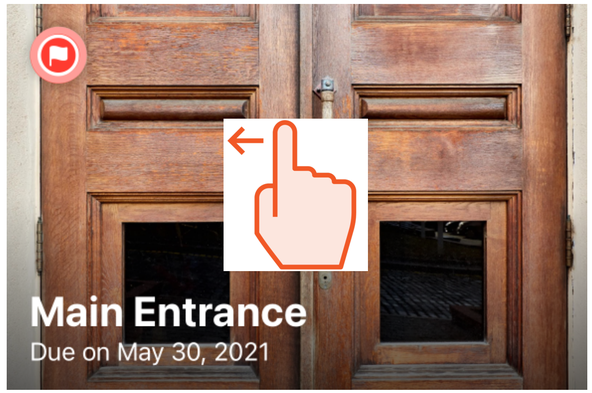
Coordinating User Interaction Between UIViews and SwiftUI



UIPageViewController will **detect** gestures and initiate the transition to the appropriate page, but it does not take on the responsibility of defining what should happen next...

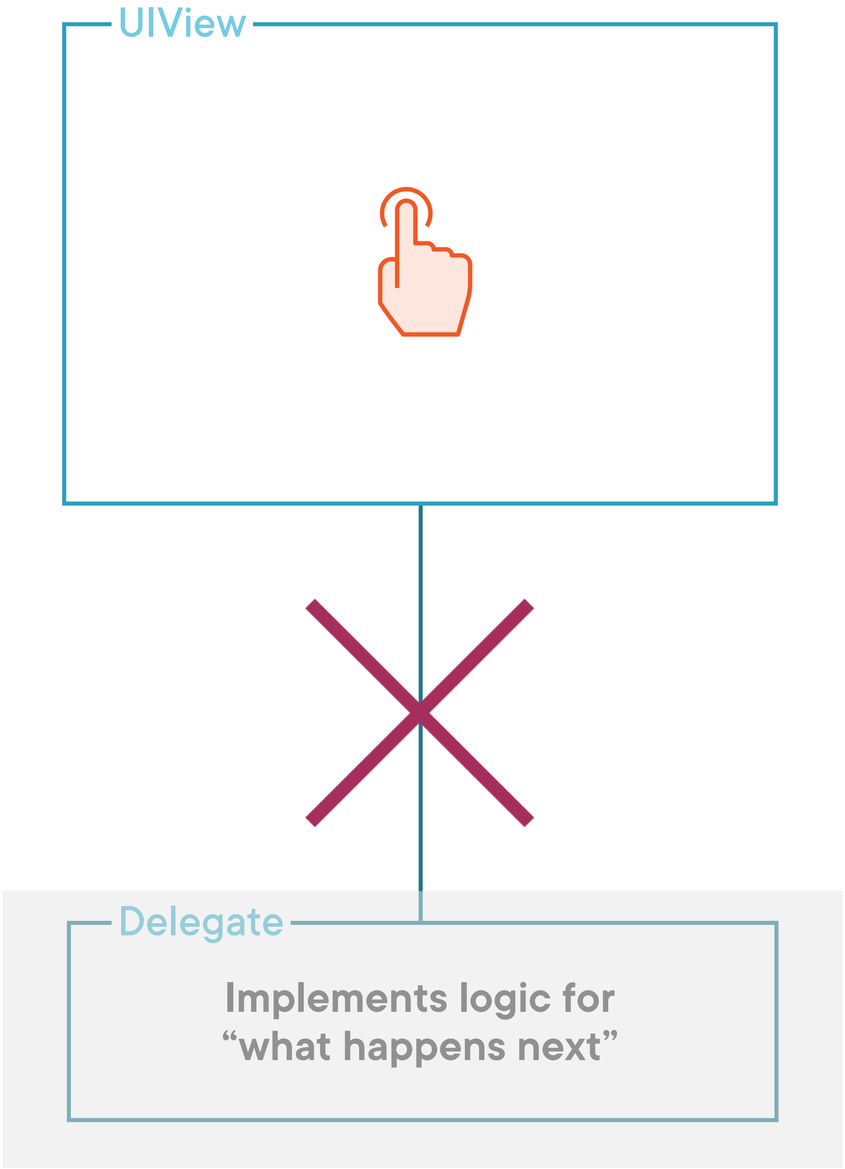
UIPageViewController

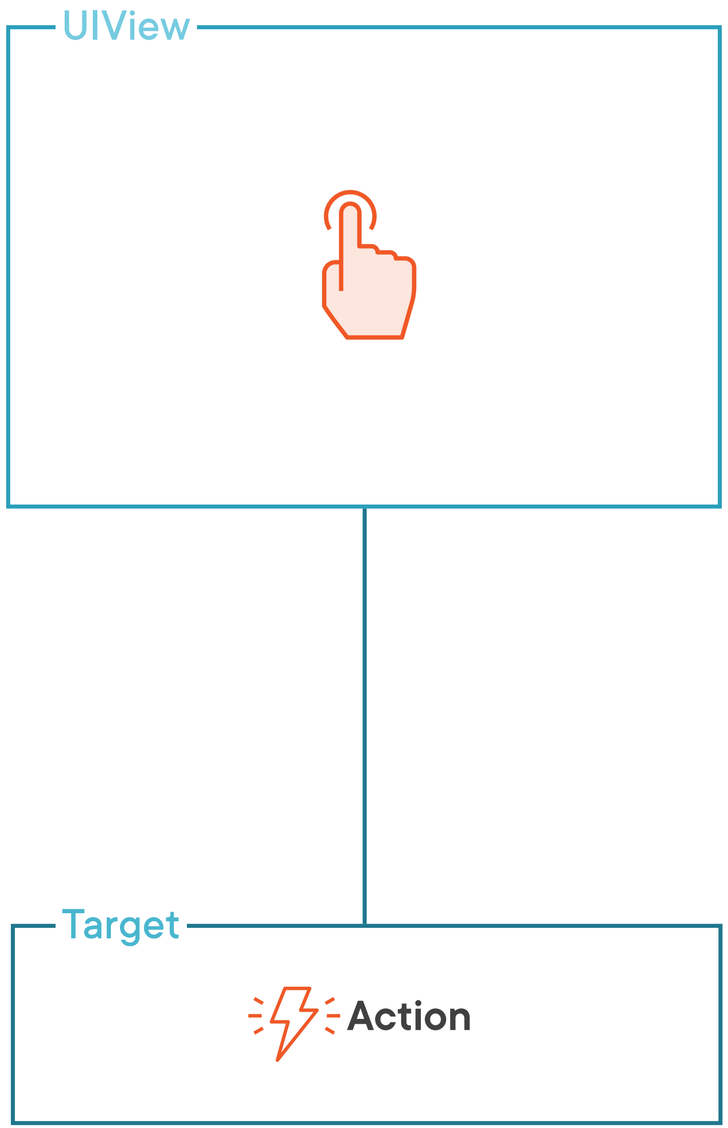
Child UIViewController



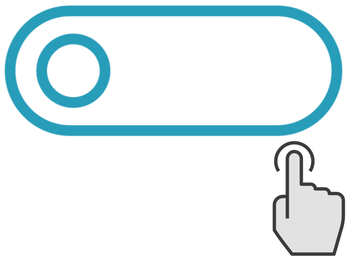
Delegate

Implements logic for “what happens next”

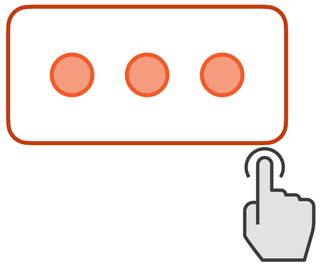

UIView
Delegate
Implements logic for
“what happens next”


UIView
Target
Action

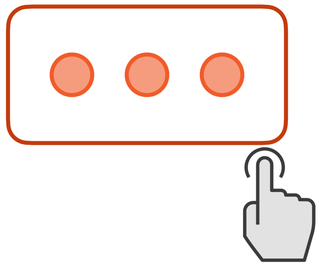
UIButton



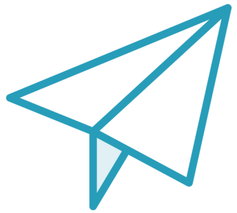
UISwitch



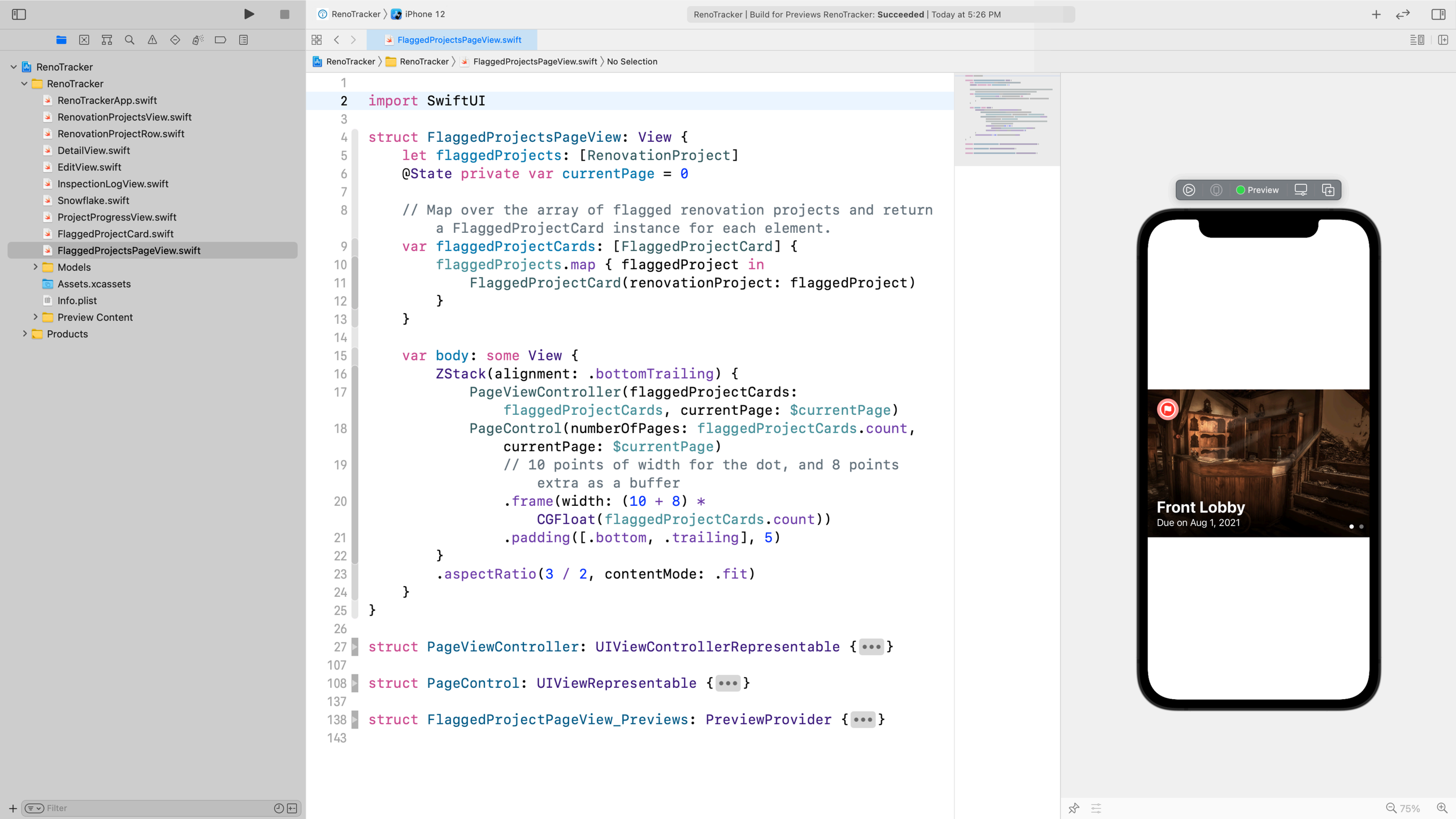
UIPageControl



UIPageControl

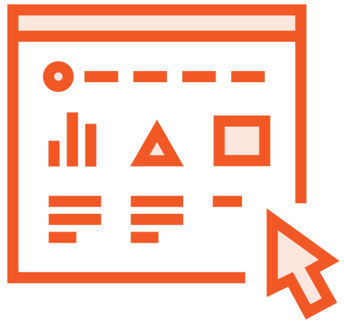


Applying the Finishing Touches

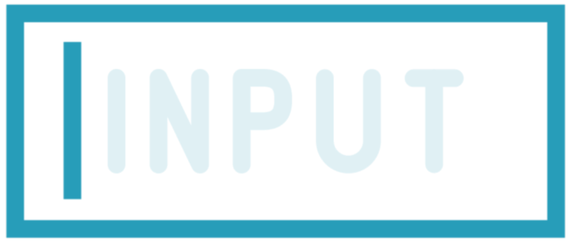

RenoTracker
iPhone 12
RenoTracker | Build for Previews RenoTracker: Succeeded | Today at 5:26 PM
FlaggedProjectsPageView.swift
RenoTracker
RenoTracker
RenoTrackerApp.swift
RenovationProjectsView.swift
RenovationProjectRow.swift
DetailView.swift
EditView.swift
InspectionLogView.swift
Snowflake.swift
ProjectProgressView.swift
FlaggedProjectCard.swift
FlaggedProjectsPageView.swift
Models
Assets.xcassets
Info.plist
Preview Content
Products
RenoTracker
RenoTracker
FlaggedProjectsPageView.swift
No Selection
1 import SwiftUI
2
3 struct FlaggedProjectsPageView: View {
4 let flaggedProjects: [RenovationProject]
5 @State private var currentPage = 0
6
7 // Map over the array of flagged renovation projects and return a FlaggedProjectCard instance for each element.
8 var flaggedProjectCards: [FlaggedProjectCard] {
9 flaggedProjects.map { flaggedProject in
10 FlaggedProjectCard(renovationProject: flaggedProject)
11 }
12 }
13
14 var body: some View {
15 ZStack(alignment: .bottomTrailing) {
16 PageViewController(flaggedProjectCards: flaggedProjectCards, currentPage: $currentPage)
17 PageControl(numberOfPages: flaggedProjectCards.count, currentPage: $currentPage)
18 // 10 points of width for the dot, and 8 points extra as a buffer
19 .frame(width: (10 + 8) * CGFloat(flaggedProjectCards.count))
20 .padding([.bottom, .trailing], 5)
21 }
22 .aspectRatio(3 / 2, contentMode: .fit)
23 }
24 }
25
26 struct PageViewController: UIViewControllerRepresentable {...}
27
107 struct PageControl: UIViewRepresentable {...}
108
137 struct FlaggedProjectPageView_Previews: PreviewProvider {...}
138
143
Preview
Front Lobby
Due on Aug 1, 2021
Filter
75%

Basics

Intermediate

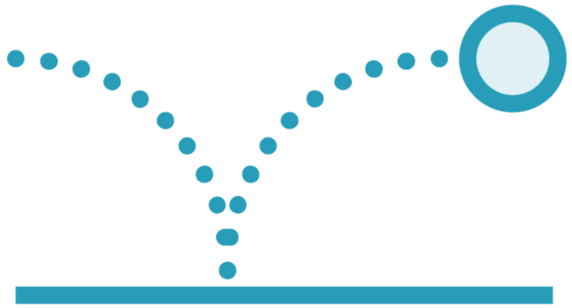


App design and layout



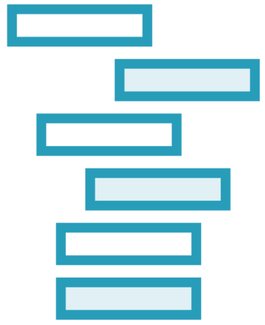
Handle user input

Implement custom drawings



Apply animations

Respond to gestures



Integrate between UIKit and SwiftUI

UIKit Paradigms

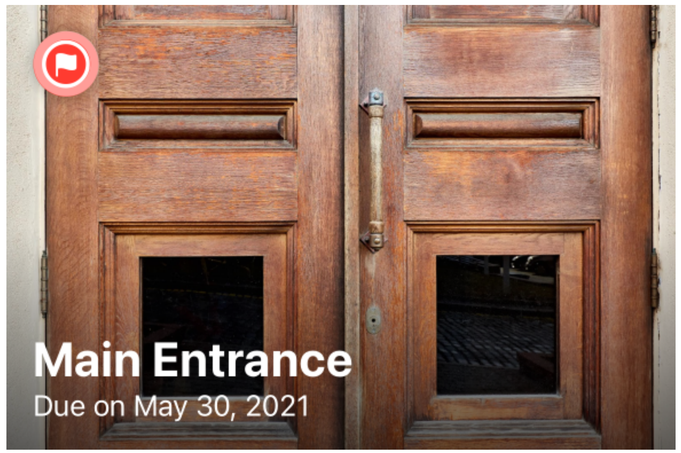
Delegate

Target-action

PageViewController

UIPageViewController

UIHostingController



Thank you!