

iOS 14: Getting Started

PREPARING TO BUILD IOS APPLICATIONS



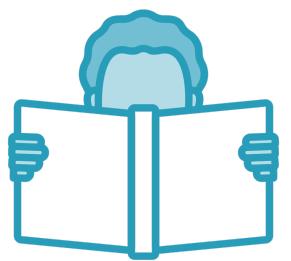
Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

In order to be successful...



What do I need to **have?**

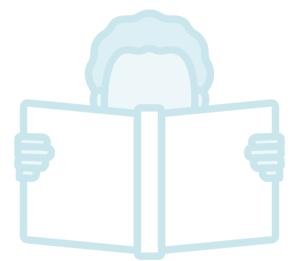


What do I need to **know?**

In order to be successful...



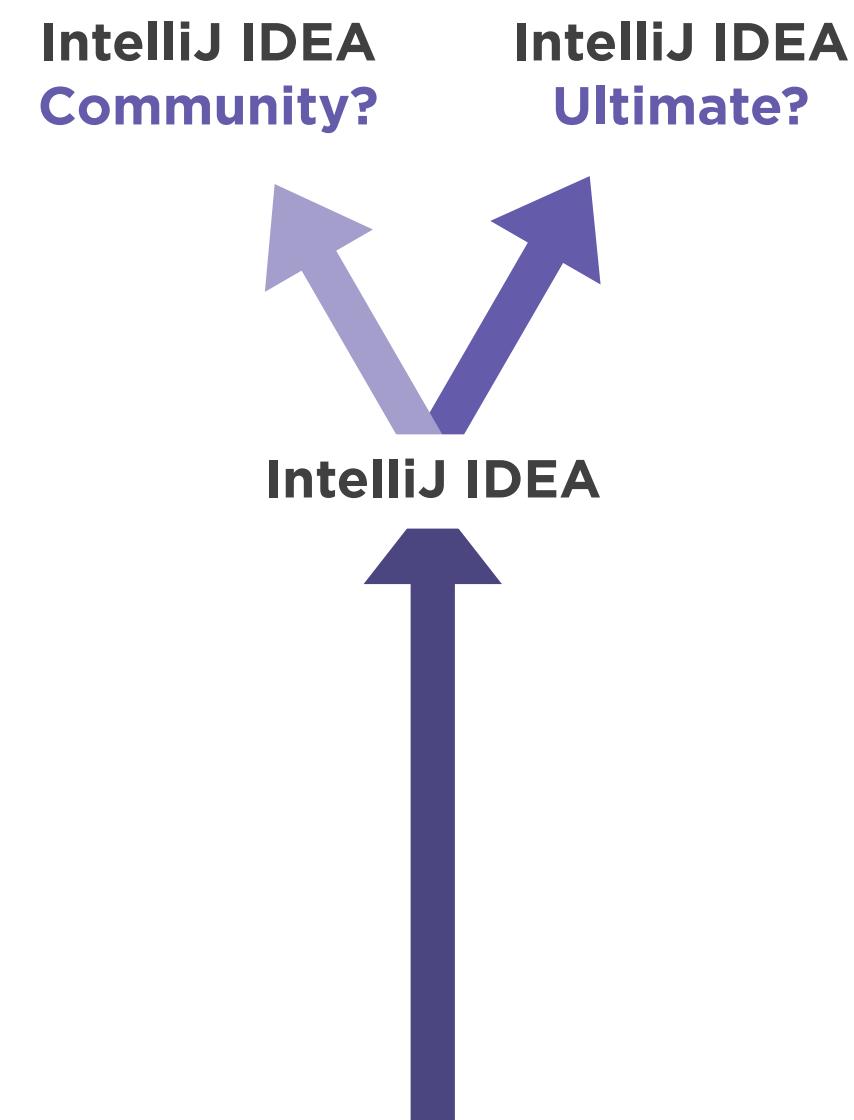
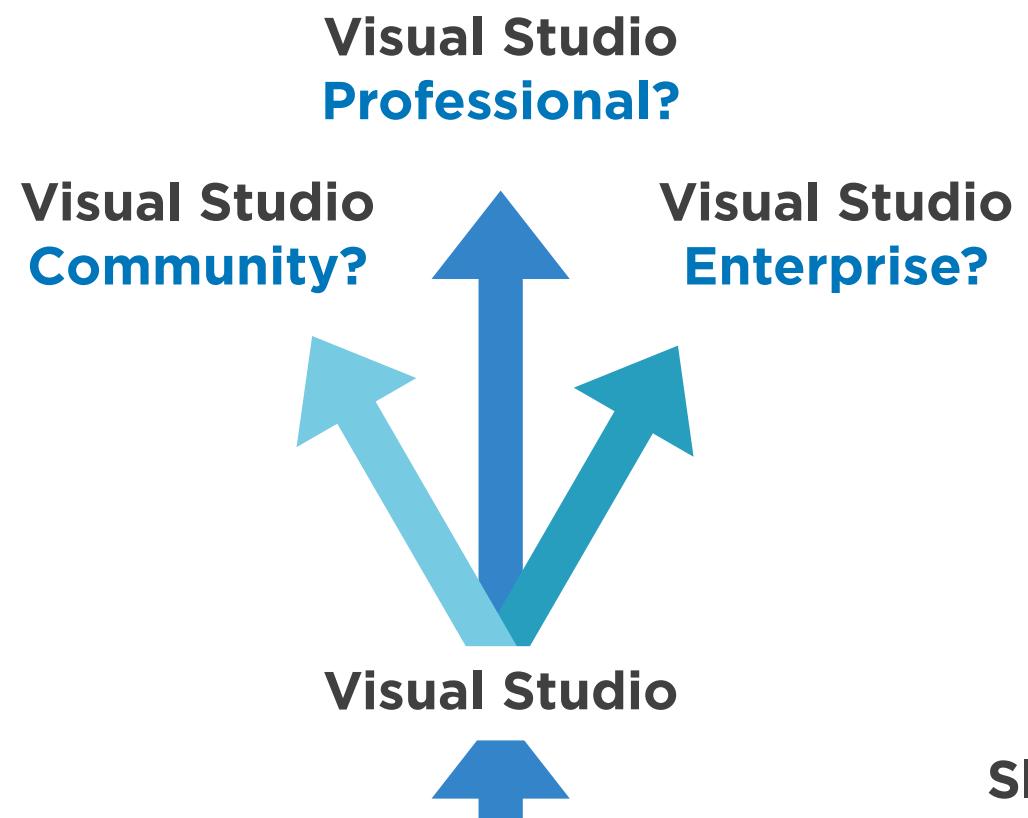
What do I need to **have**?

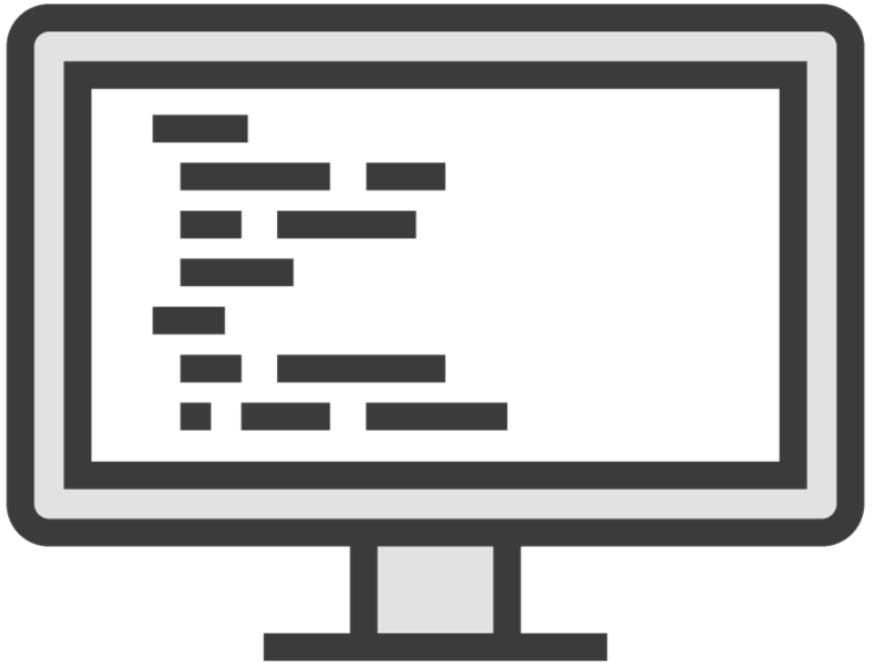


What do I need to **know**?

Side Note: Xcode “Editions”







Everybody uses Xcode

No “lite” edition

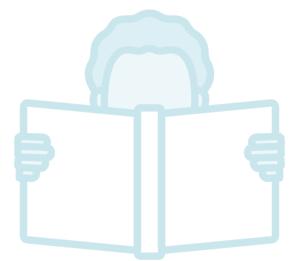
No “pro” edition

No “enterprise” edition

In order to be successful...



What do I need to **have**?

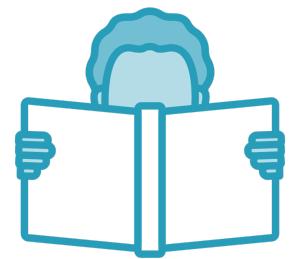


What do I need to **know**?

In order to be successful...

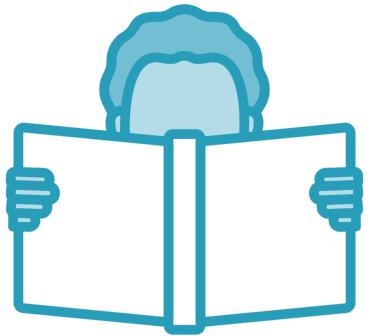


What do I need to **have**?



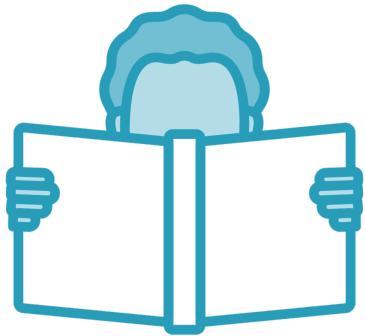
What do I need to **know**?

What do I need to know?



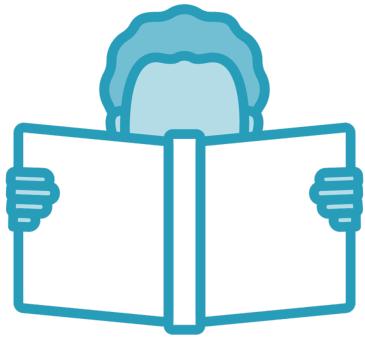
History or background with iOS development

What do I need to know?



✗ ~~History or background with iOS development~~

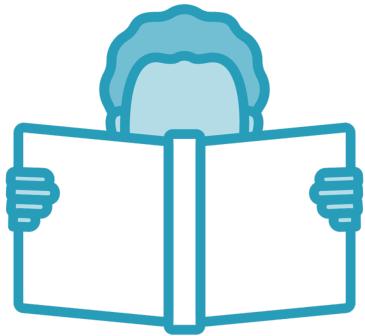
What do I need to know?



✗ ~~History or background with iOS development~~

✓ **Swift**

What do I need to know?

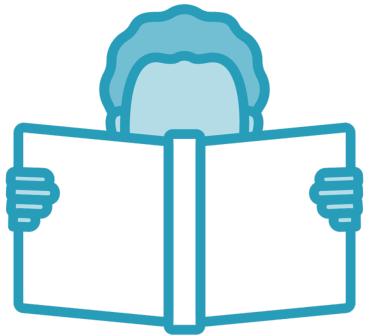


✗ ~~History or background with iOS development~~

✓ **Swift**

Do *not* need to be an expert

What do I need to know?

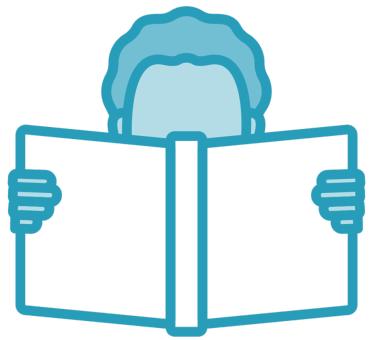


✗ ~~History or background with iOS development~~

✓ **Swift**

Do not need to be an expert

Know at least the basics



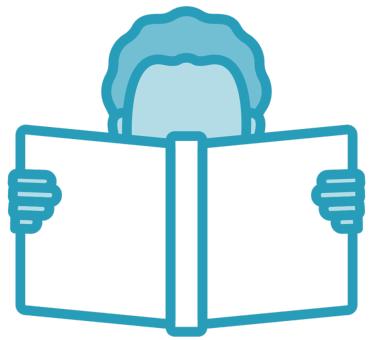
Swift



Swift



“How do you make a String constant?”



Swift



“How do you make a String constant?”



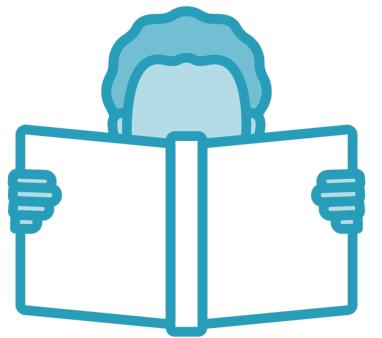
Swift



“How do you make a String constant?”



“What is type inference?”



Swift



“How do you make a String constant?”



“What is type inference?”



Swift



“How do you make a String constant?”



“What is type inference?”



“What’s the difference between a class and a struct?”

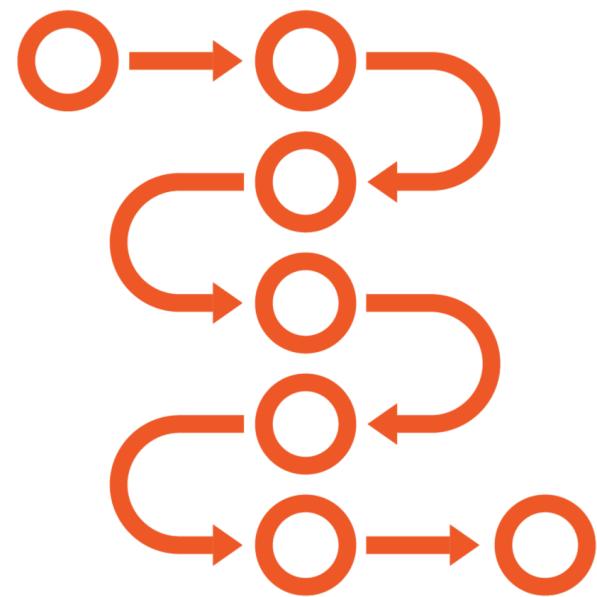
No Swift experience?

No Swift experience?

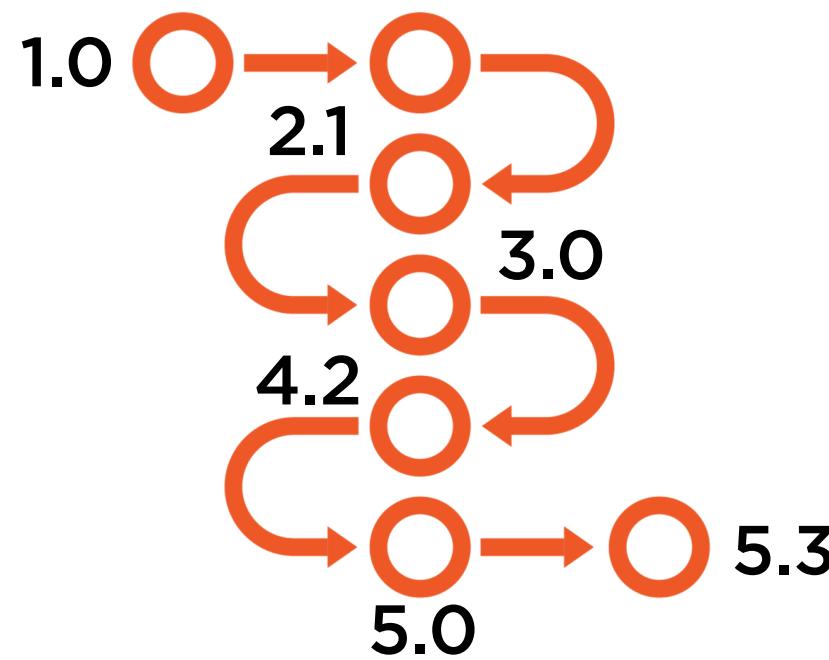
No Problem.

Side Note: Swift Versions

Side Note: Swift Versions



Side Note: Swift Versions



Side Note: Swift Versions

Swift 5.3

Side Note: Swift Versions

Swift 5.3

Side Note: Swift Versions

Swift 5.3

4.x

Side Note: Swift Versions

Swift 5.3

4.x

Side Note: Swift Versions

Swift 5.3

4.x

3.x

2.x

1.x

Side Note: Swift Versions

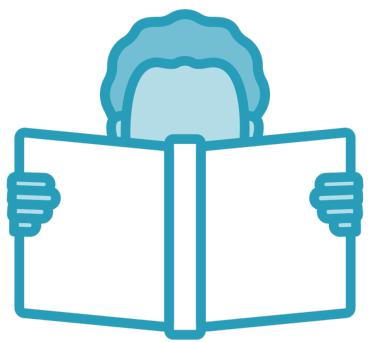
Swift 5.3

4.x

3.x

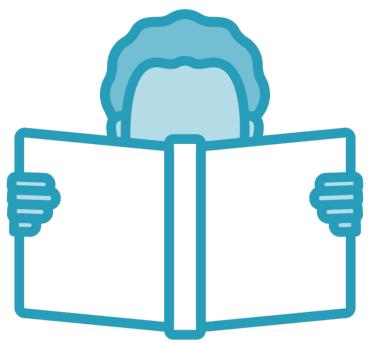
✗ 2.x

✗ 1.x



Swift Fundamentals

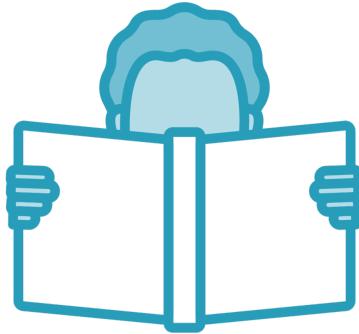
- Optionals
- Closures
- Protocols



Swift Fundamentals

- Optionals
- Closures
- Protocols

What do I need to know?



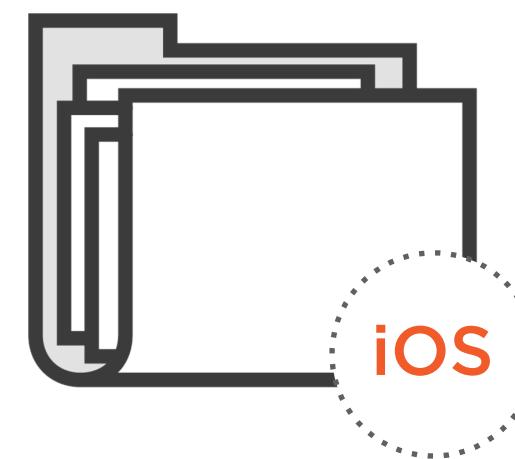
~~✗ History or background with iOS development~~

✓ **Swift**

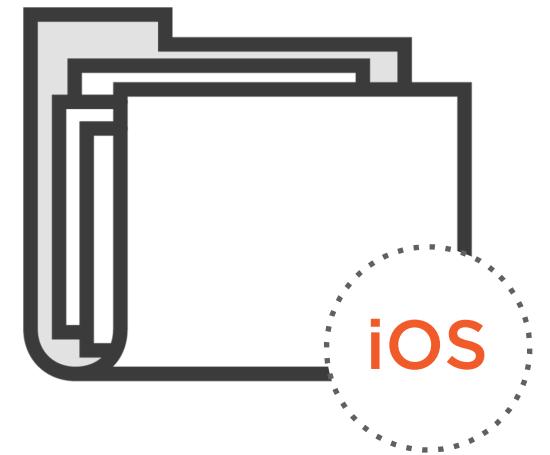
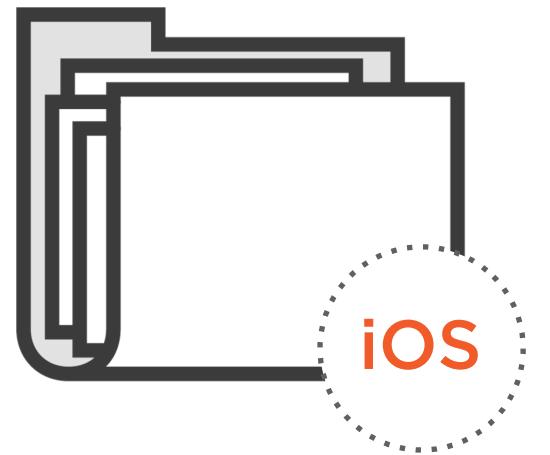
✓ **Object-oriented software development**

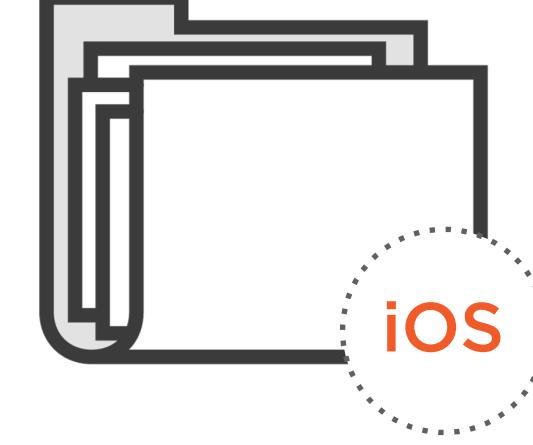
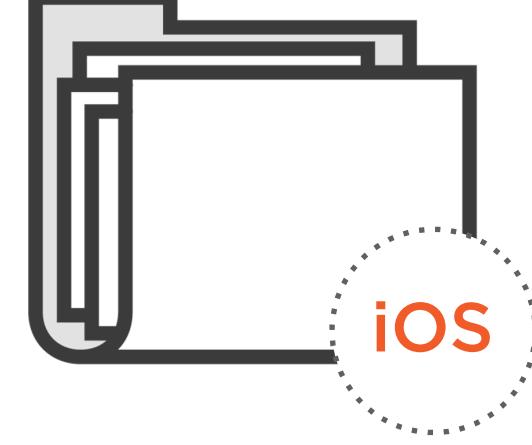
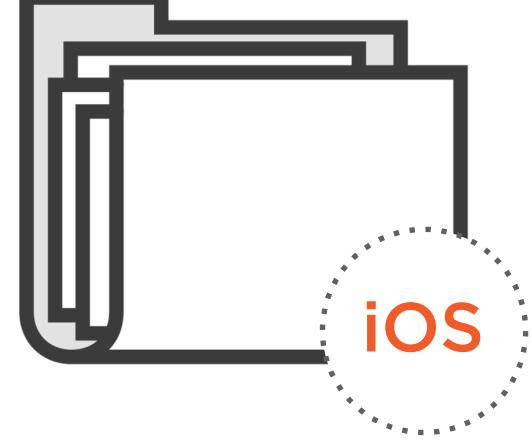
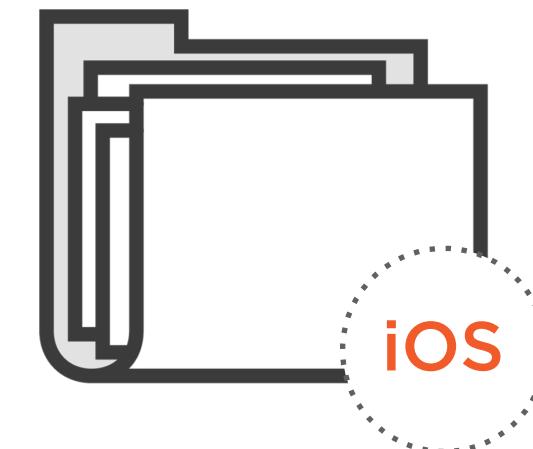
✓ **Accustomed to Mac**

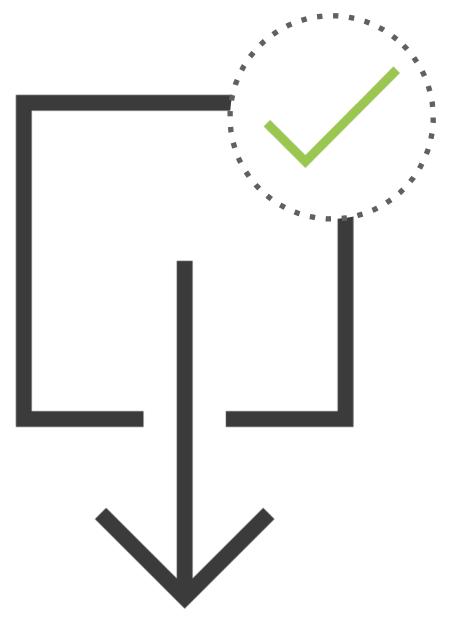
Creating an iOS Project with Xcode





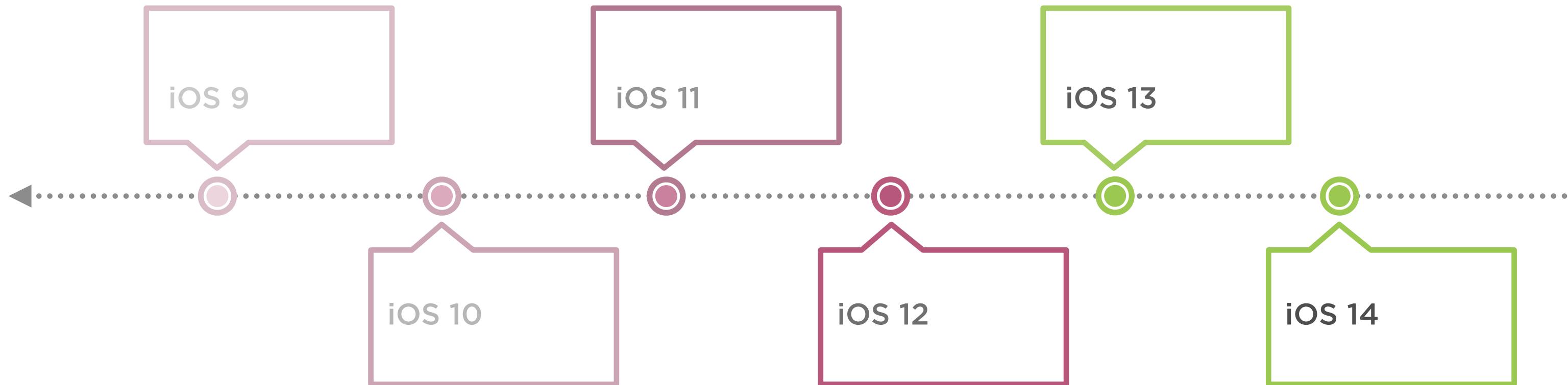




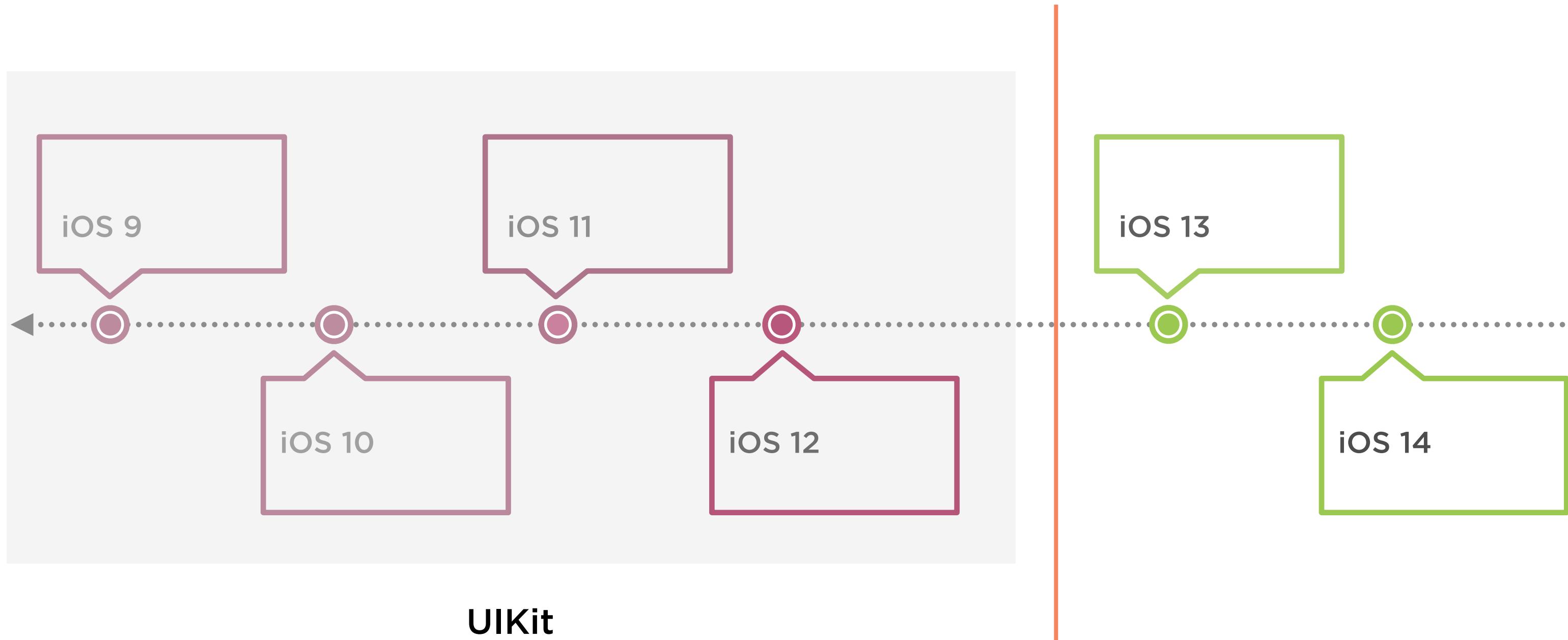


Xcode install complete

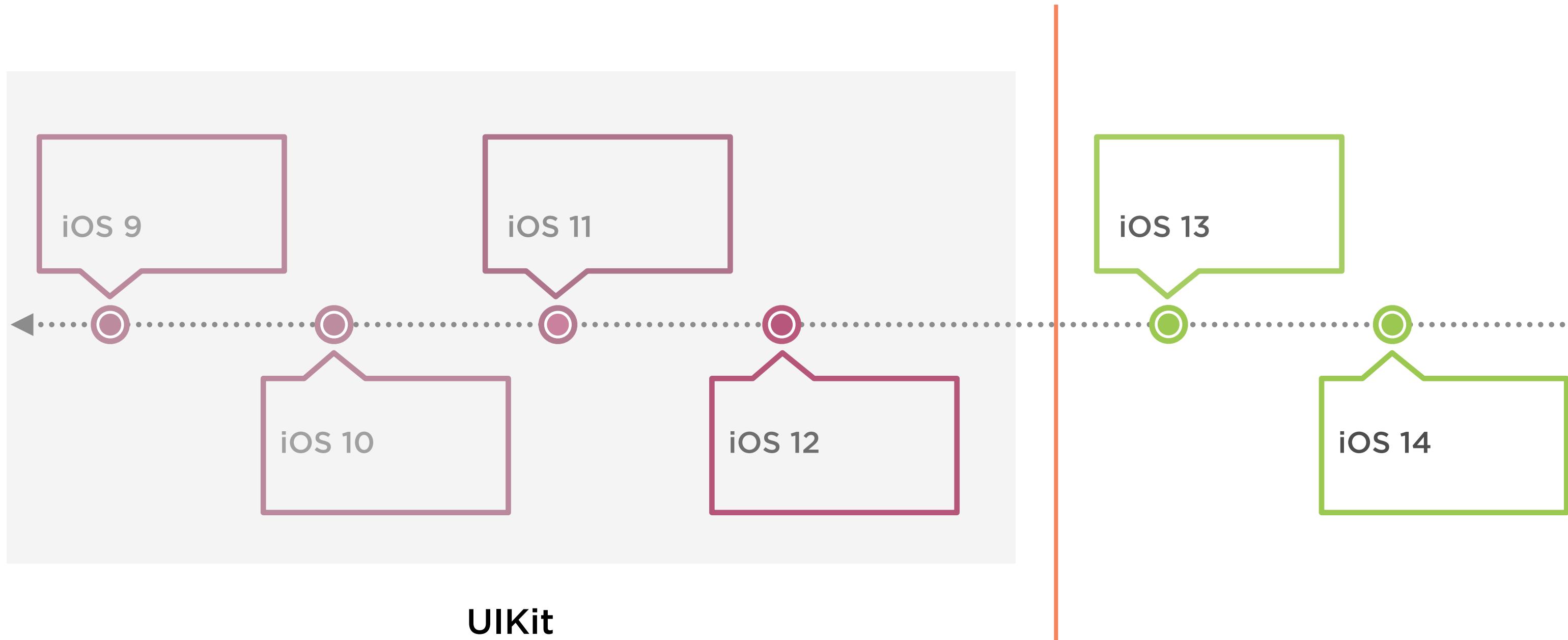
User Interface Framework Options



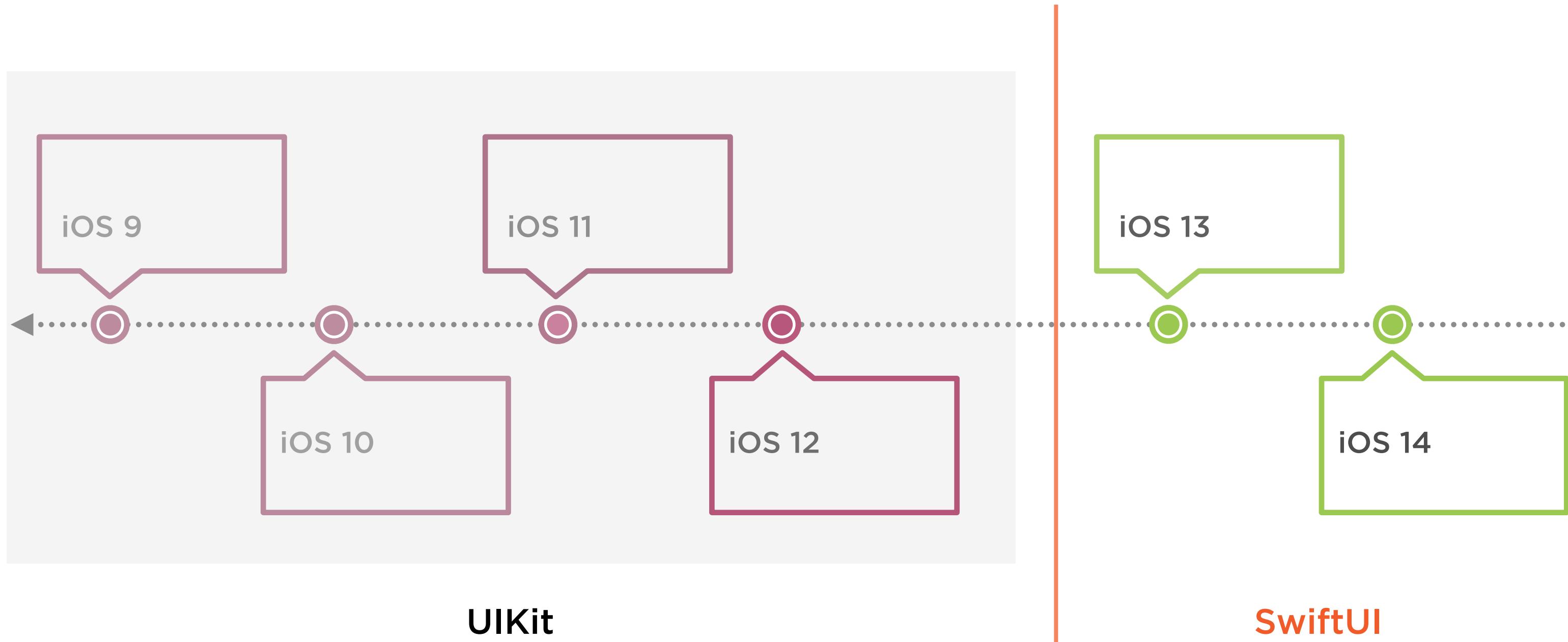
User Interface Framework Options



User Interface Framework Options

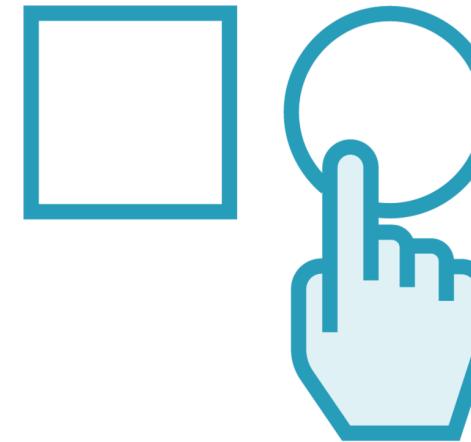


User Interface Framework Options

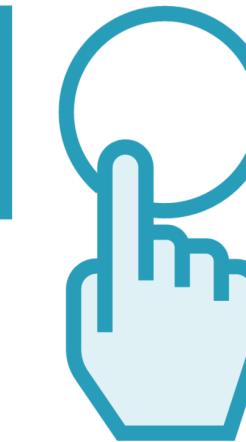


User Interface Framework Options

UIKit



SwiftUI



UIAccelerometer	UIColor	UILabel
UIAccessibility	UIControl	UILayoutGuide
UIAccessibilityAdditions	UIDataDetectors	UILexicon
UIAccessibilityConstants	UIDatePicker	UILocalNotification
UIAccessibilityCustomAction	UIDevice	UILocalizedIndexedCollation
UIAccessibilityElement	UIDocument	UILongPressGestureRecognizer
UIAccessibilityIdentification	UIDocumentInteractionController	UIManagedDocument
UIAccessibilityZoom	UIDocumentMenuViewController	UIMenuController
UIActionsheet	UIDocumentPickerController	UITabBarController
UIActiv	UITouch	UIApplication
UIActiv	UIPress	UIApplicationState
UIActiv	UISwipeGestureRecognizer	UIEvent
UIActiv	UIPinchGestureRecognizer	UIPasteboard
UIAlert	UIFieldBehavior	UIPrinter
UIAlert	UIFocus	UIPanGestureRecognizer
UIAlert	UIFocusAnimationCoordinator	UIPasteboard
UIAlert	UIFont	UIPickerView
UIAppea	UIFontDescriptor	UIPinchGestureRecognizer
UIAppli	UIGeometry	UIPopoverBackgroundView
UIApplicationShortcutItem	UIGestureRecognizer	UIPopoverController
UIAttachmentBehavior	UIGestureRecognizerSubclass	UIPopoverPresentationController
UIBarButtonItem	UIGraphics	UIPopoverSupport
UIBarButtonItemGroup	UIGravityBehavior	UIPresentationController
UIBarCommon	UIGuidedAccessRestrictions	UIPress
UIBarItem	UIImage	UIPressesEvent
UIBezierPath	UITImageAsset	UIPrintError
UIButton		
UICollectionView		
UICollectionViewCell		

App Lifecycle



- Launch app
- Present on-screen
- Send to background
- Other “plumbing” and infrastructure

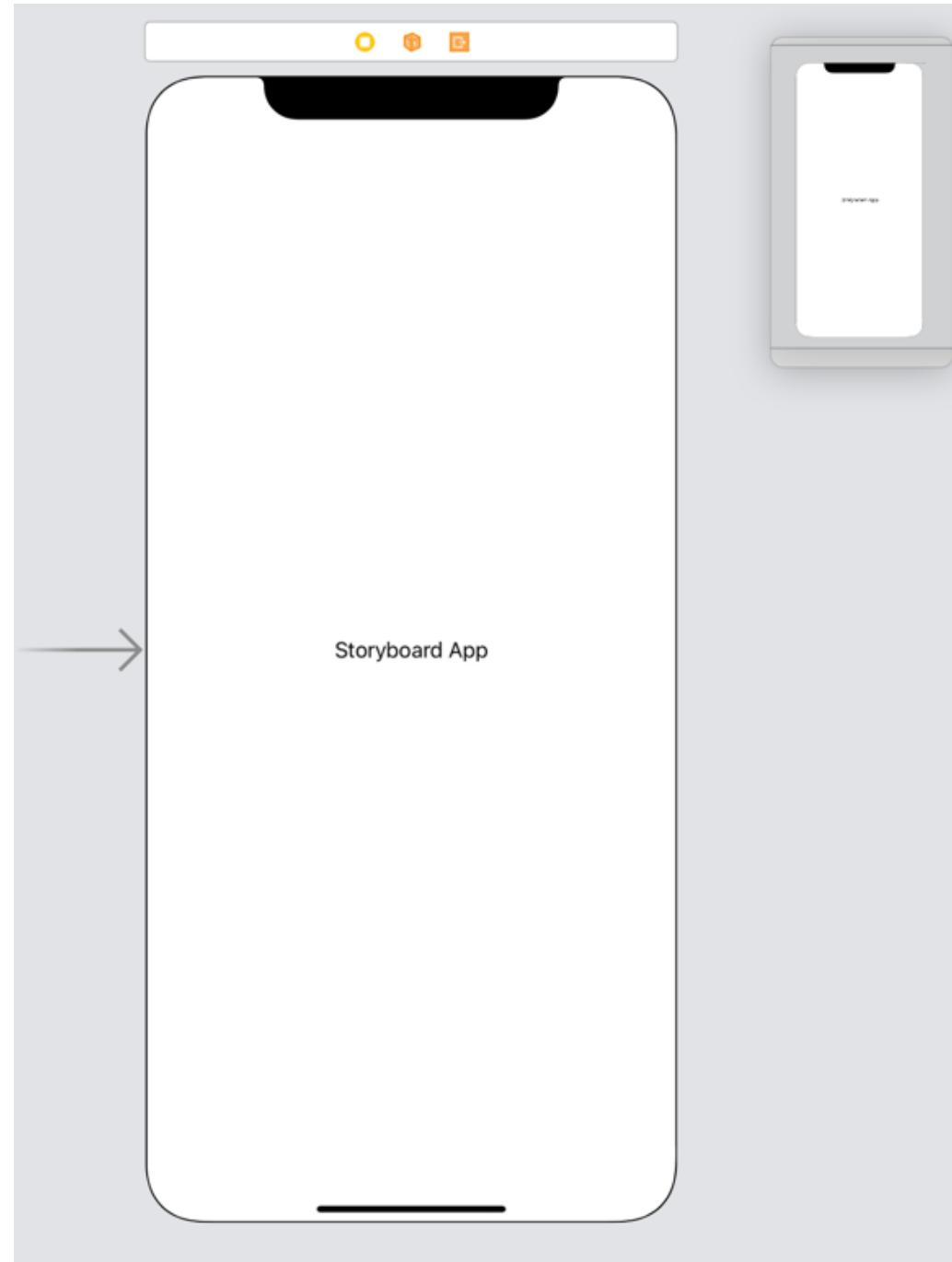
App Lifecycle



UIKit App Delegate

- Launch app
- Present on-screen
- Send to background
- Other “plumbing” and infrastructure

App Lifecycle

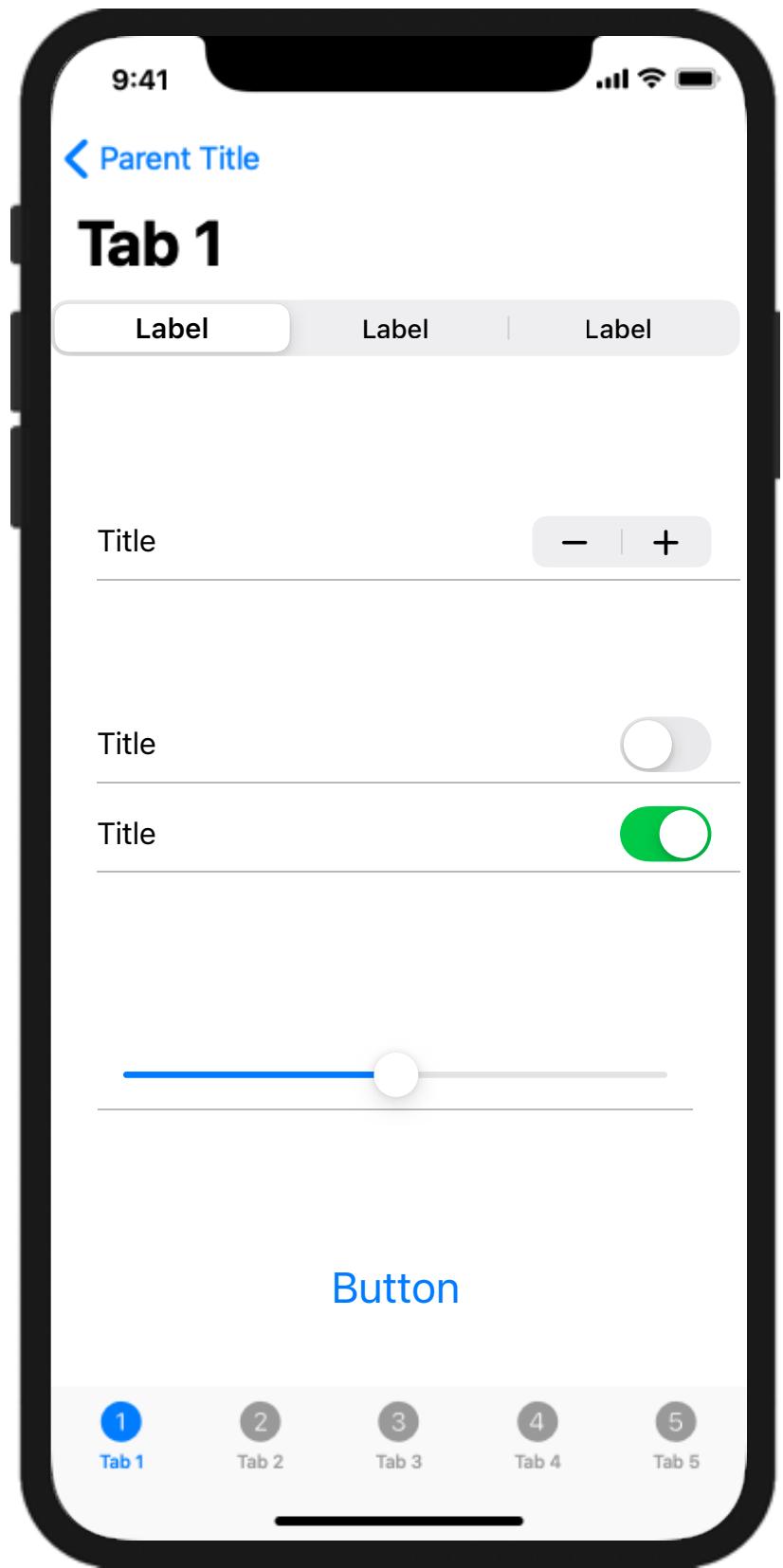


UIKit App Delegate

- Launch app
- Present on-screen
- Send to background
- Other “plumbing” and infrastructure

Storyboard

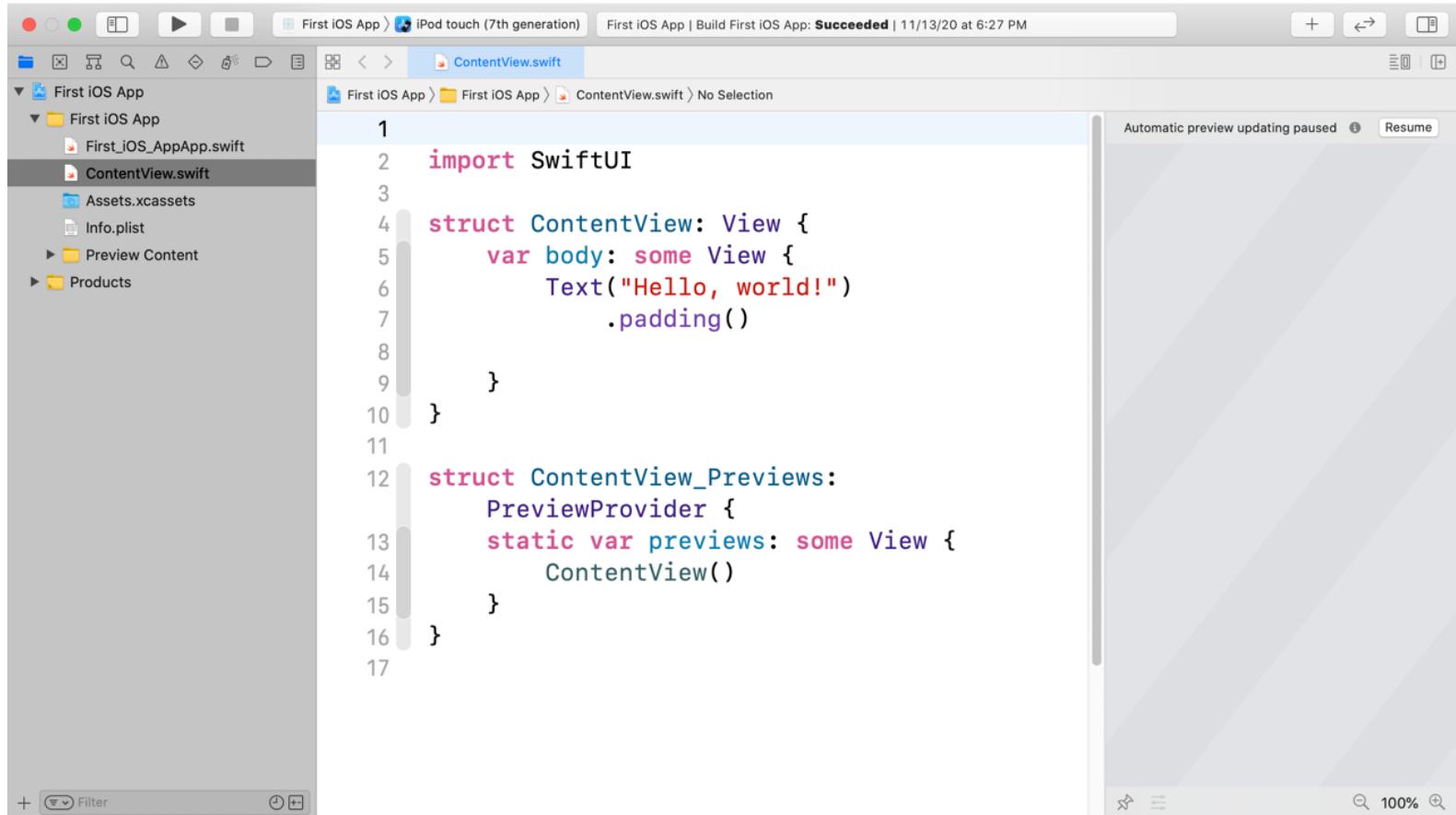
Exploring an iOS Project



A screenshot of the Xcode IDE interface. The title bar shows "First iOS App" and "iPod touch (7th generation)". The status bar indicates "Build First iOS App: Succeeded | 11/13/20 at 6:27 PM". The main window displays the file "ContentView.swift" under the "First iOS App" project. The code editor contains the following Swift code:

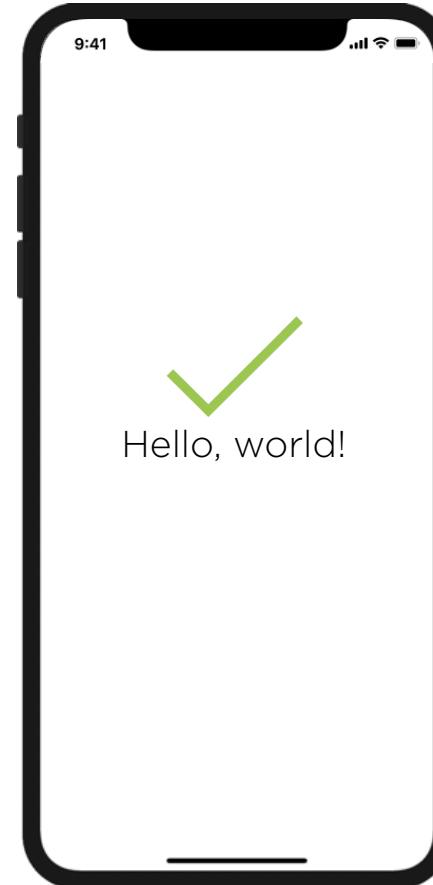
```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         Text("Hello, world!")
6             .padding()
7     }
8 }
9
10 struct ContentView_Previews: PreviewProvider {
11     static var previews: some View {
12         ContentView()
13     }
14 }
15
16
17
```

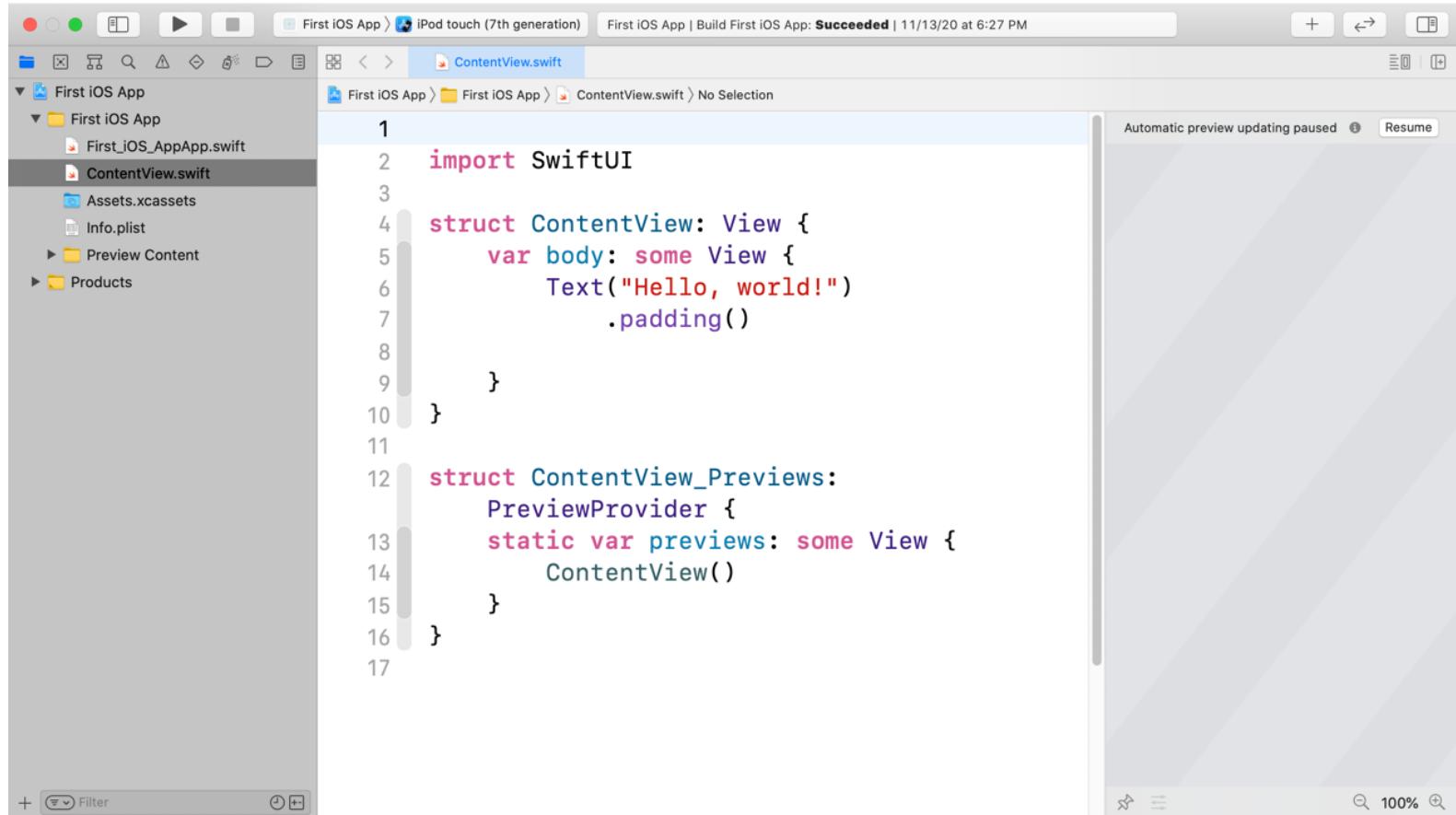
The code defines a SwiftUI view named `ContentView` which displays the text "Hello, world!" with padding. It also includes a preview provider for the view.



A screenshot of the Xcode IDE showing the ContentView.swift file. The code defines a struct ContentView that returns a Text view with the string "Hello, world!". It also includes a PreviewProvider for generating previews of the view.

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         Text("Hello, world!")
6             .padding()
7     }
8 }
9
10 struct ContentView_Previews: PreviewProvider {
11     static var previews: some View {
12         ContentView()
13     }
14 }
15
16
17
```





A screenshot of the Xcode IDE showing the `ContentView.swift` file in the editor. The file contains the following Swift code:

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         Text("Hello, world!")
6             .padding()
7     }
8 }
9
10 struct ContentView_Previews: PreviewProvider {
11     static var previews: some View {
12         ContentView()
13     }
14 }
15
16
17
```

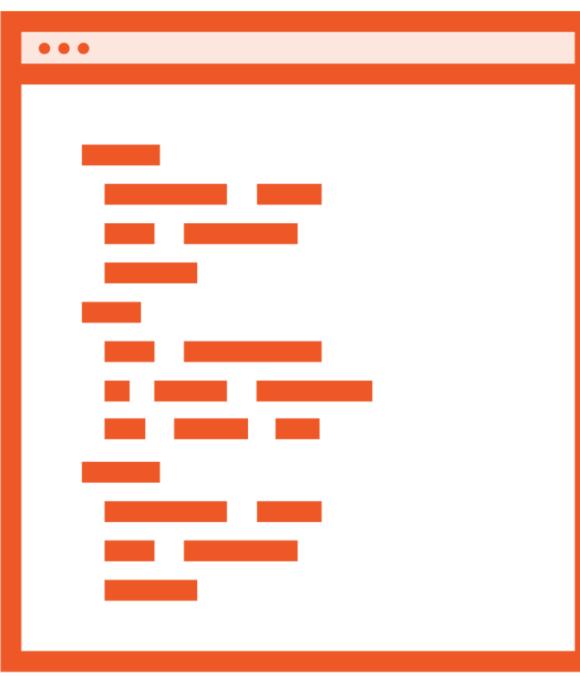
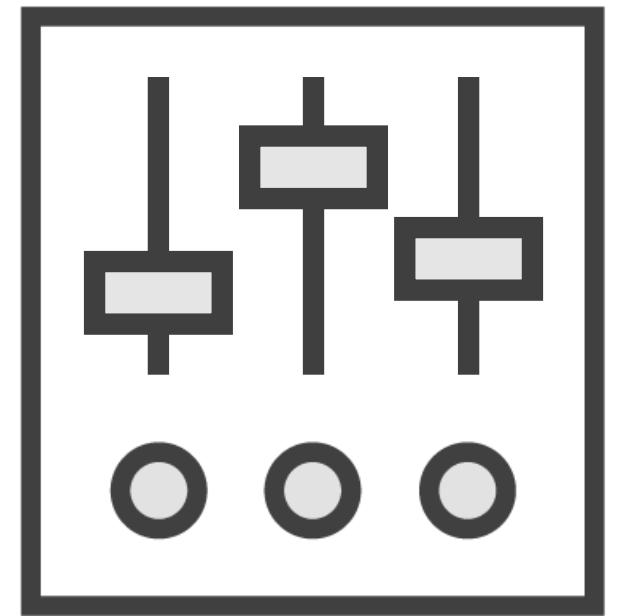
The code defines a `ContentView` struct that returns a `Text` view with the string "Hello, world!" and a padding of 12 pixels. It also defines a `ContentView_Previews` struct with a static `previews` variable that returns an instance of `ContentView`. The Xcode interface shows the project structure on the left, the file path at the top center, and a preview area on the right.

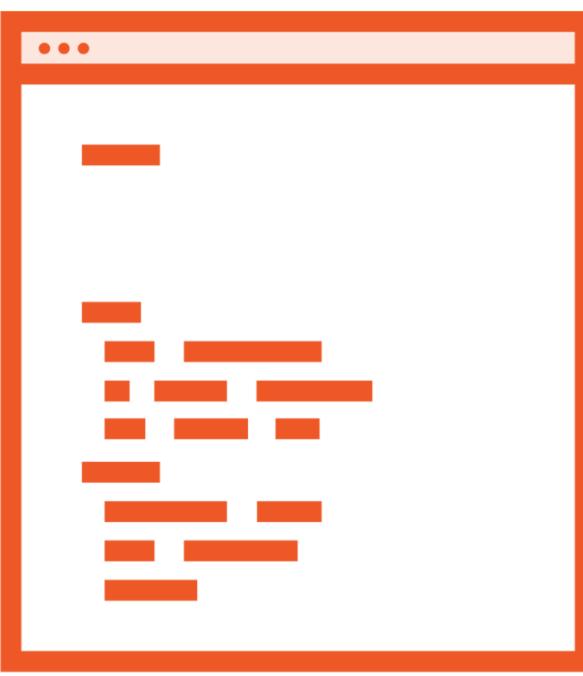
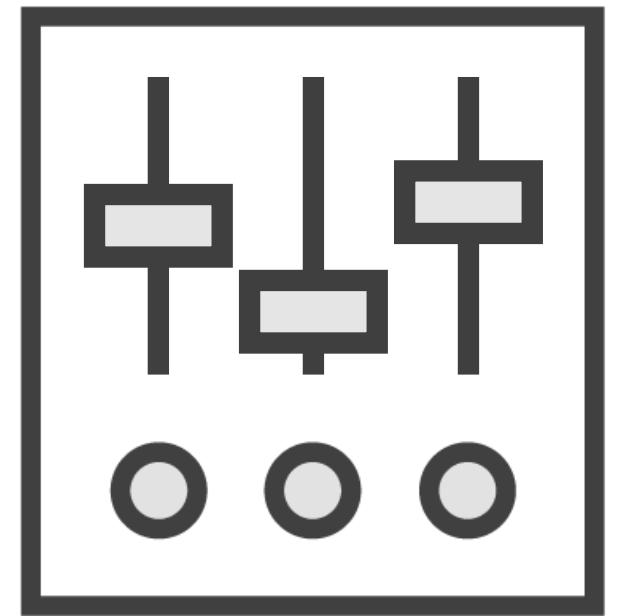
How to Learn iOS

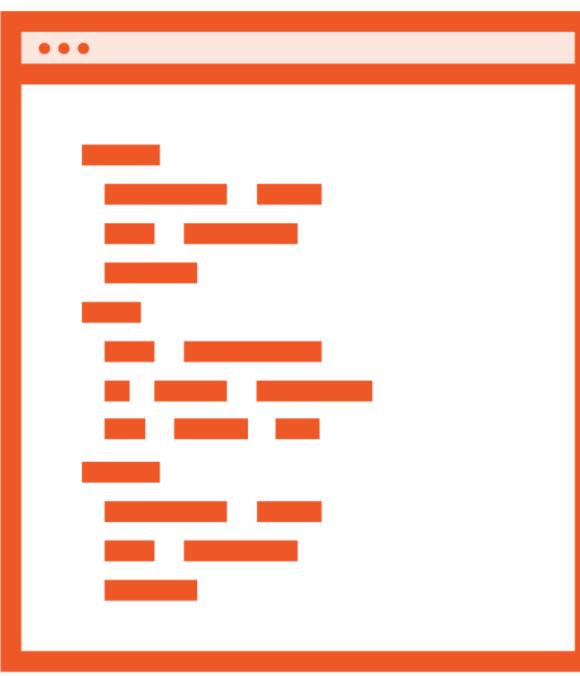
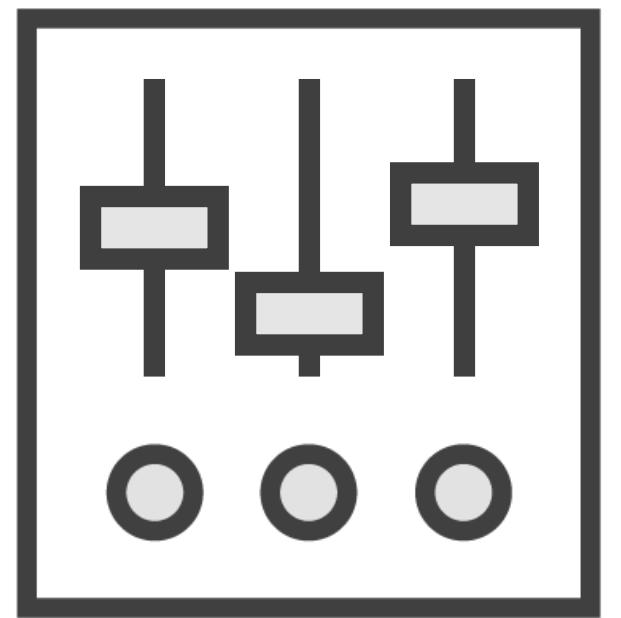


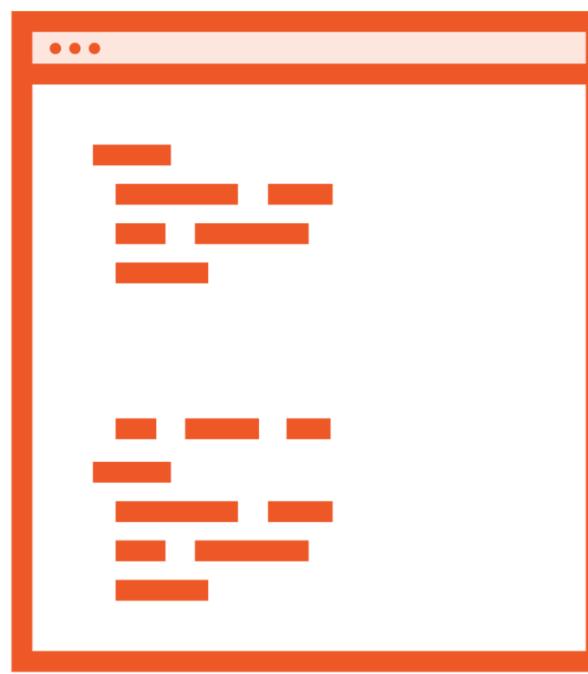
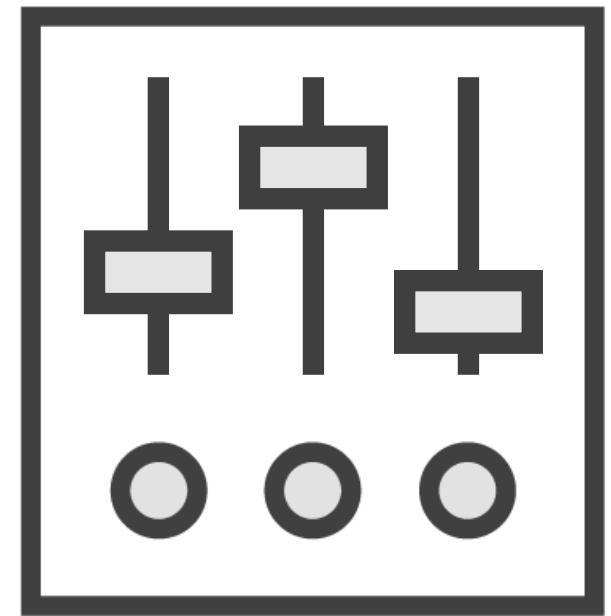
1101011010
0101101011
0010100101
1101011010
0101101011
0010100101
1101011010
0101101011
0010100101
1101011010
0101101011
0010100101

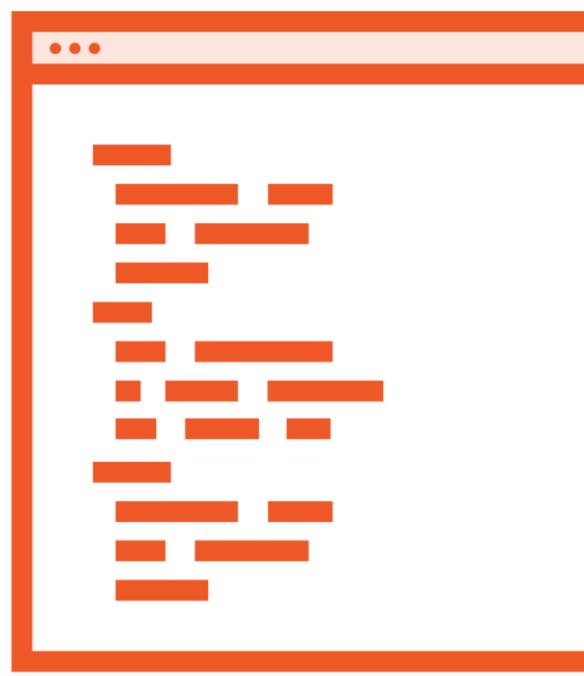
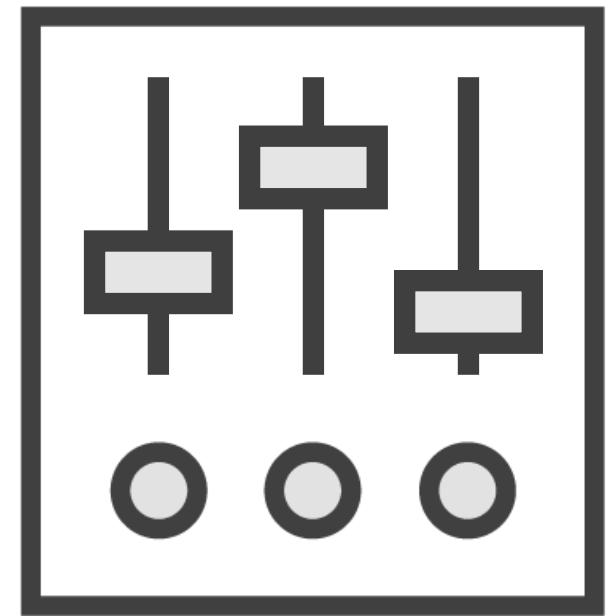


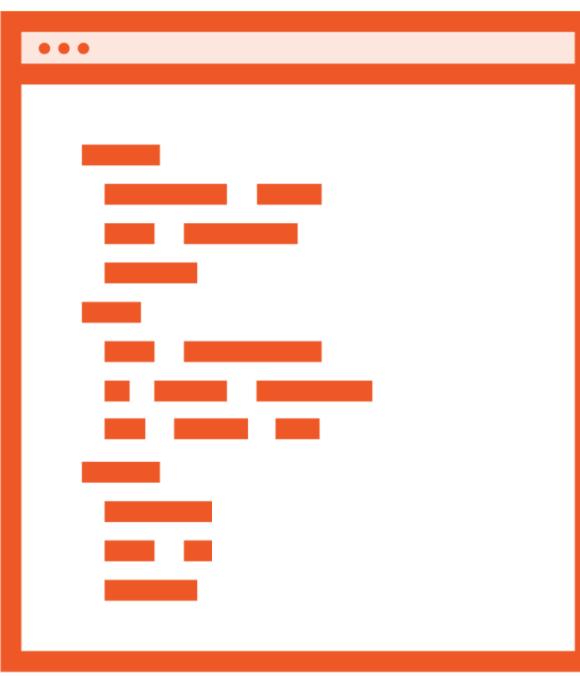
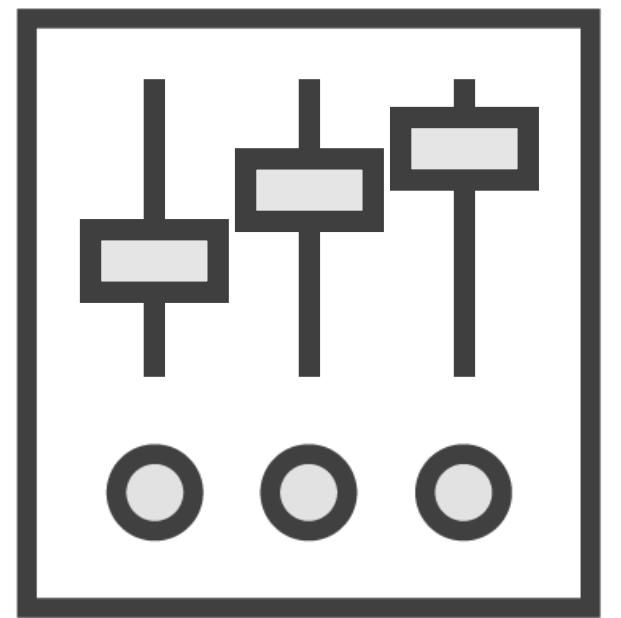


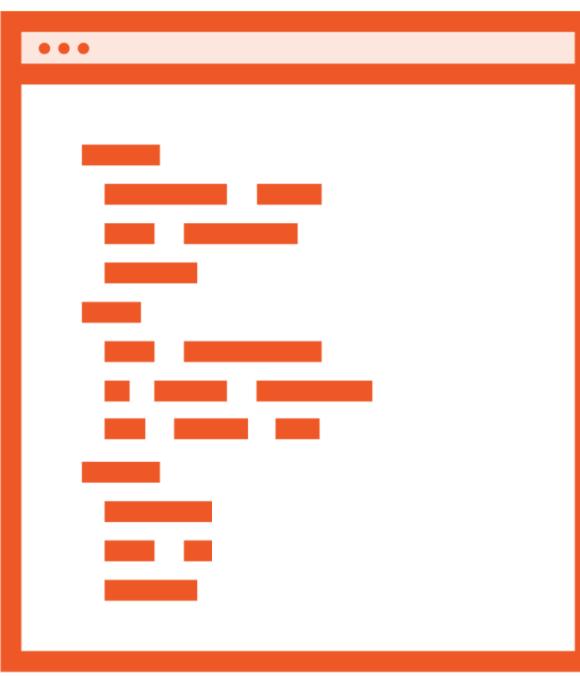
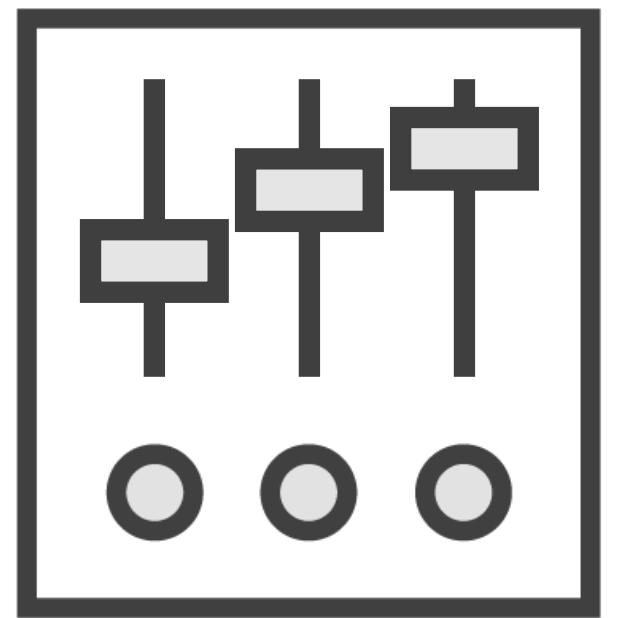


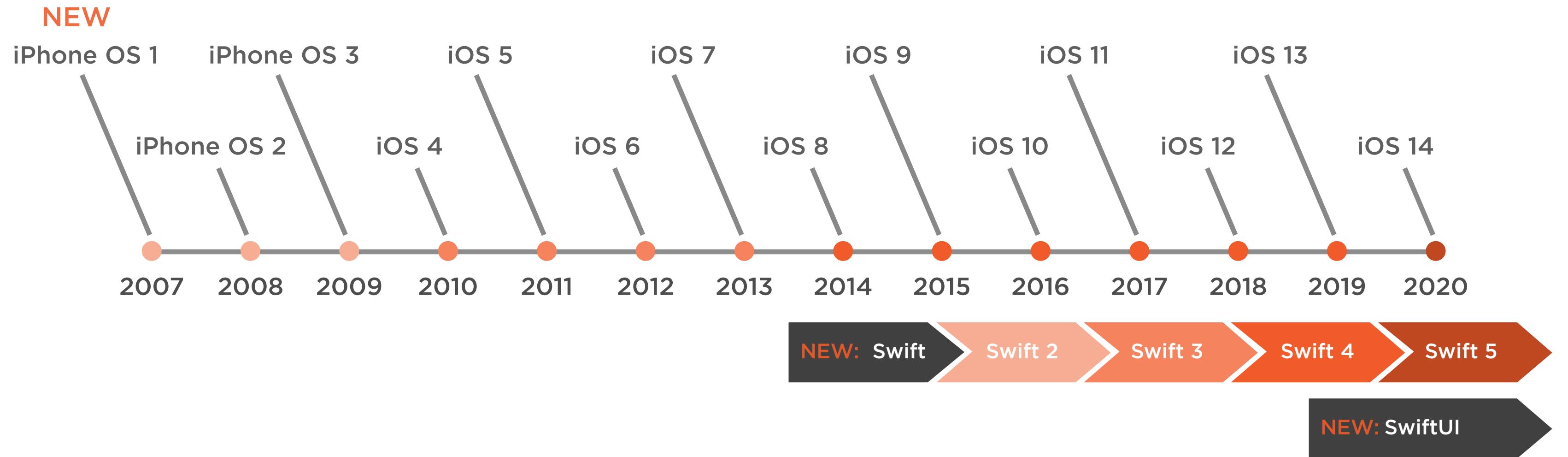


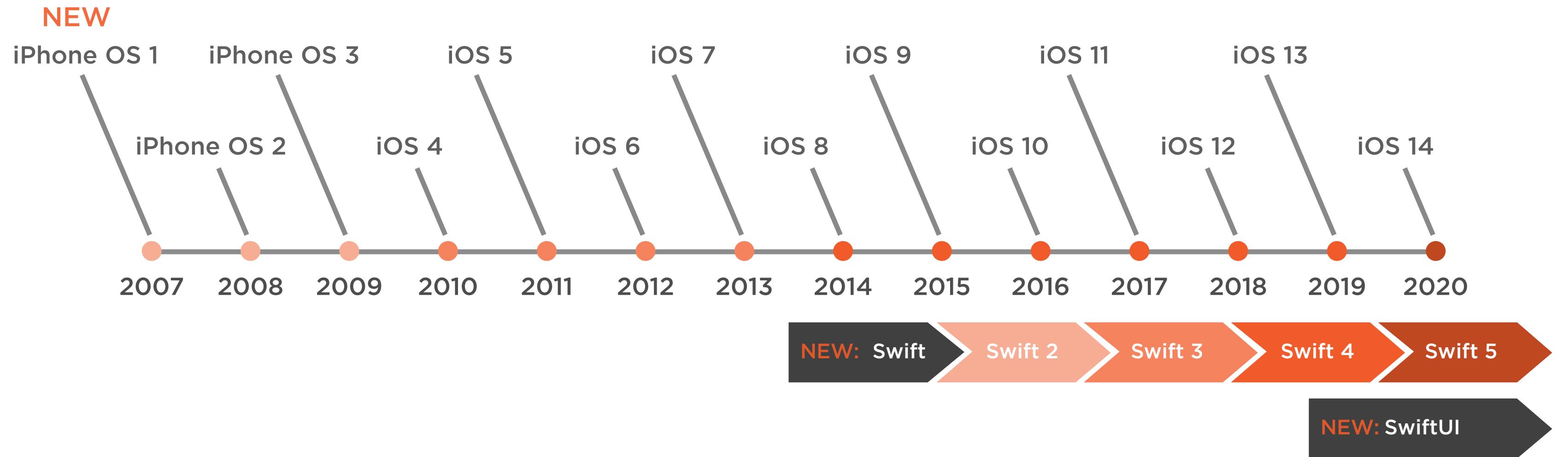


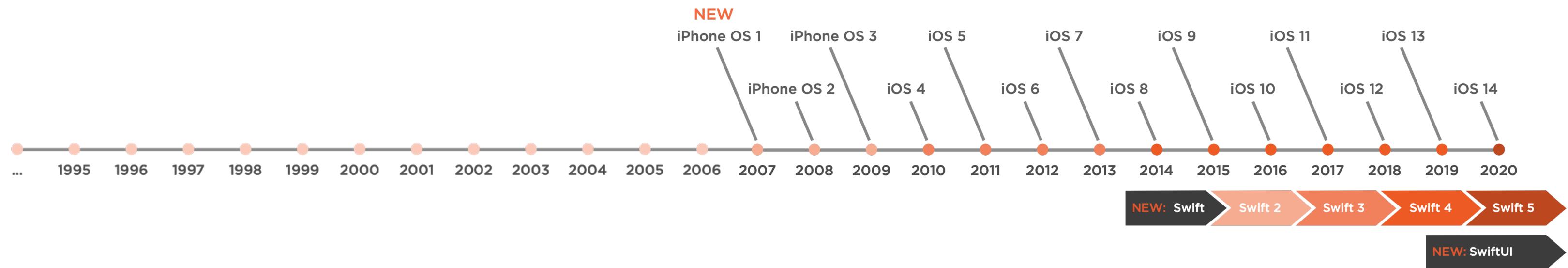












NeXTSTEP

Mac OS X / macOS

Project Builder

Xcode

Objective-C

Cocoa / Foundation / AppKit (Mac) / UIKit (iOS)



Button





inheritance!

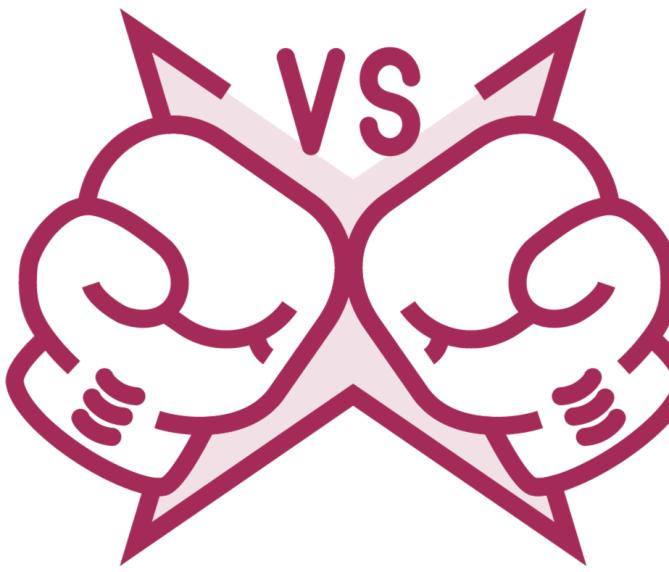
protocol!



Not academic or theoretical nitpicking



Work with iOS



Fight iOS

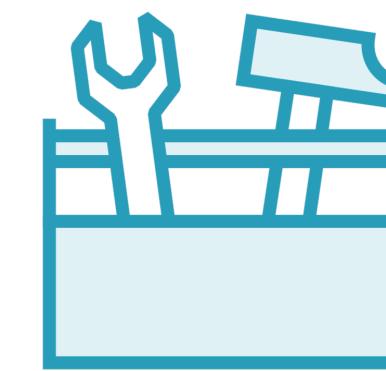


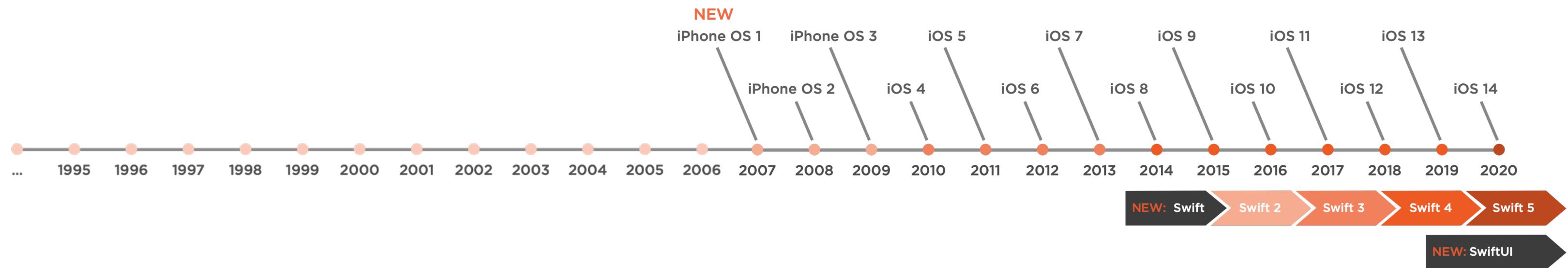


Visual Studio
Eclipse

...

Xcode





NeXTSTEP

Mac OS X / macOS

Project Builder

Xcode

Objective-C

Cocoa / Foundation / AppKit (Mac) / UIKit (iOS)

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14 Developer Skillset

Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14 Developer Skillset

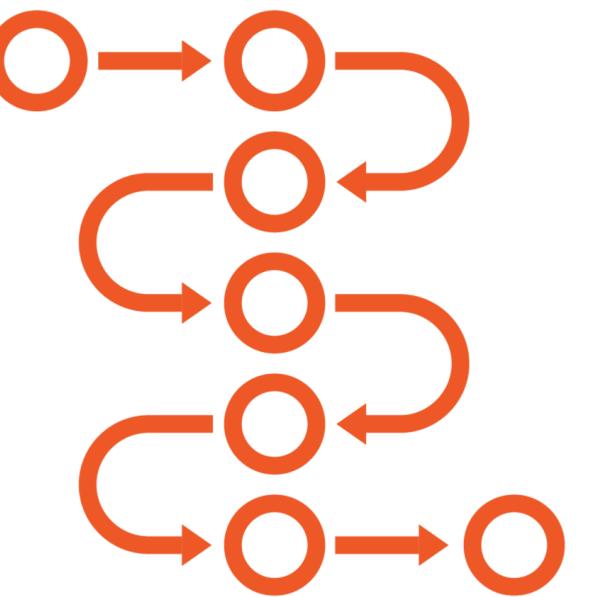
Software Patterns
Human Interface Guidelines
Deployment

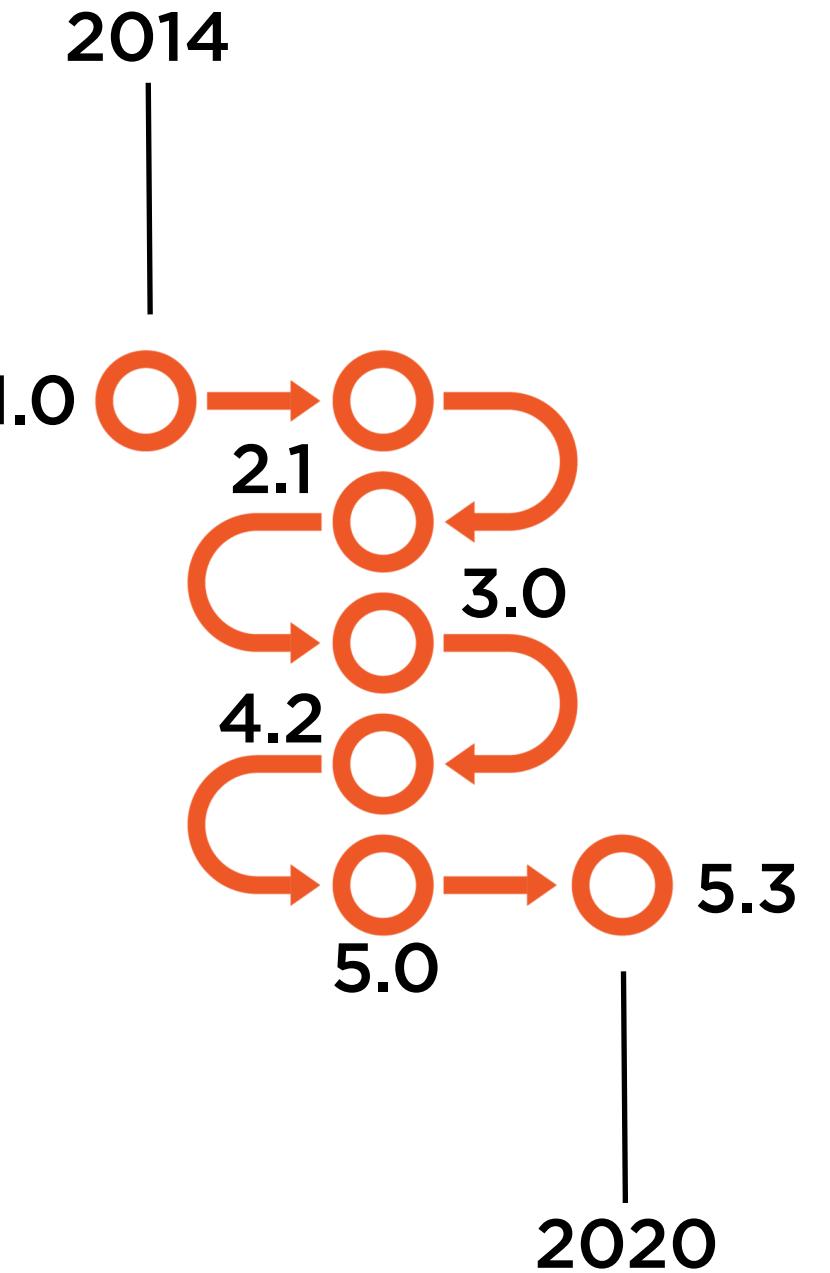
iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

2014





Swift 5.3

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14 Developer Skillset

Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14 Developer Skillset

Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14 Developer Skillset

Software Patterns
Human Interface Guidelines
Deployment

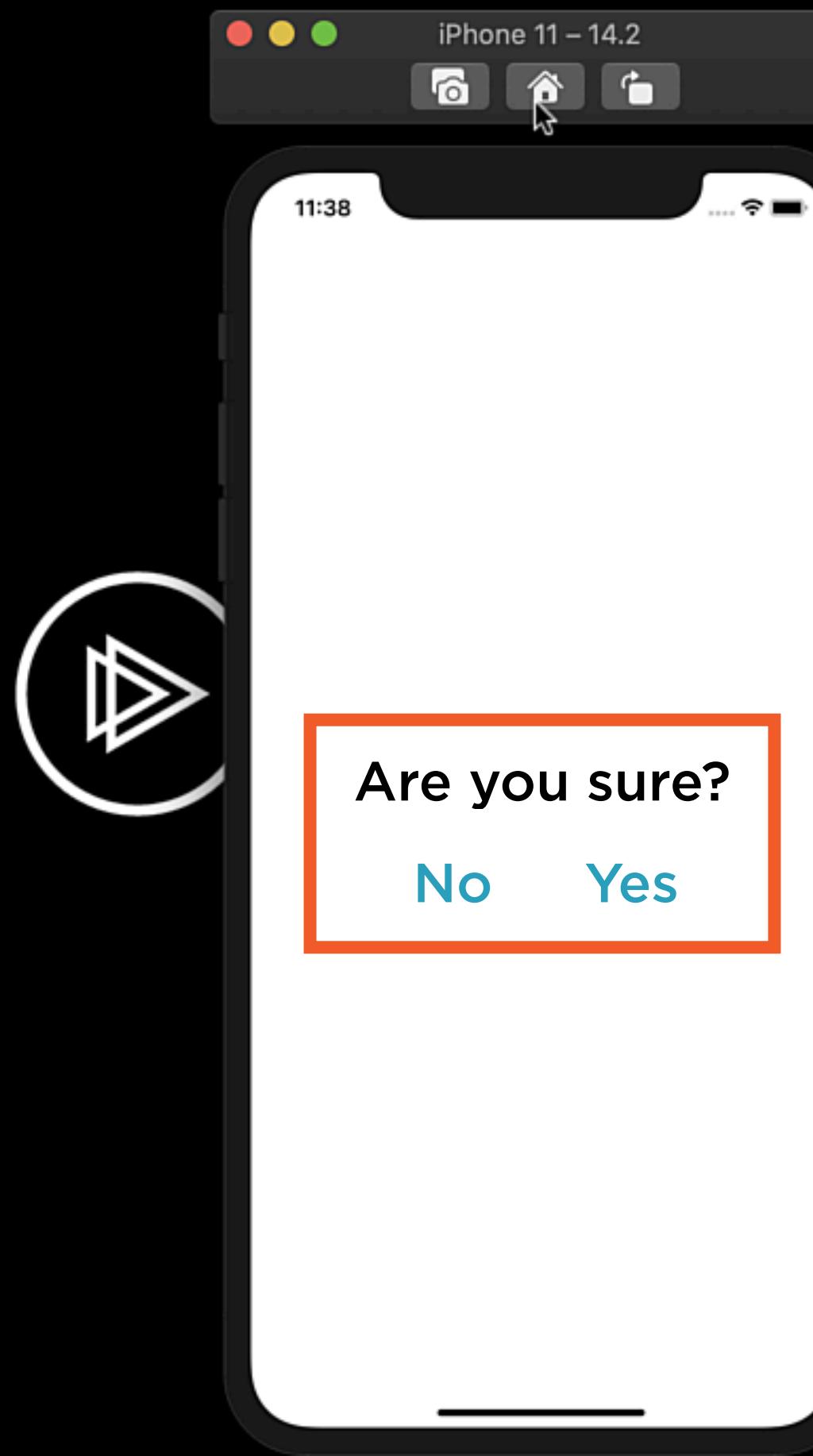
iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture



GHT



Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14

Developer
Skillset

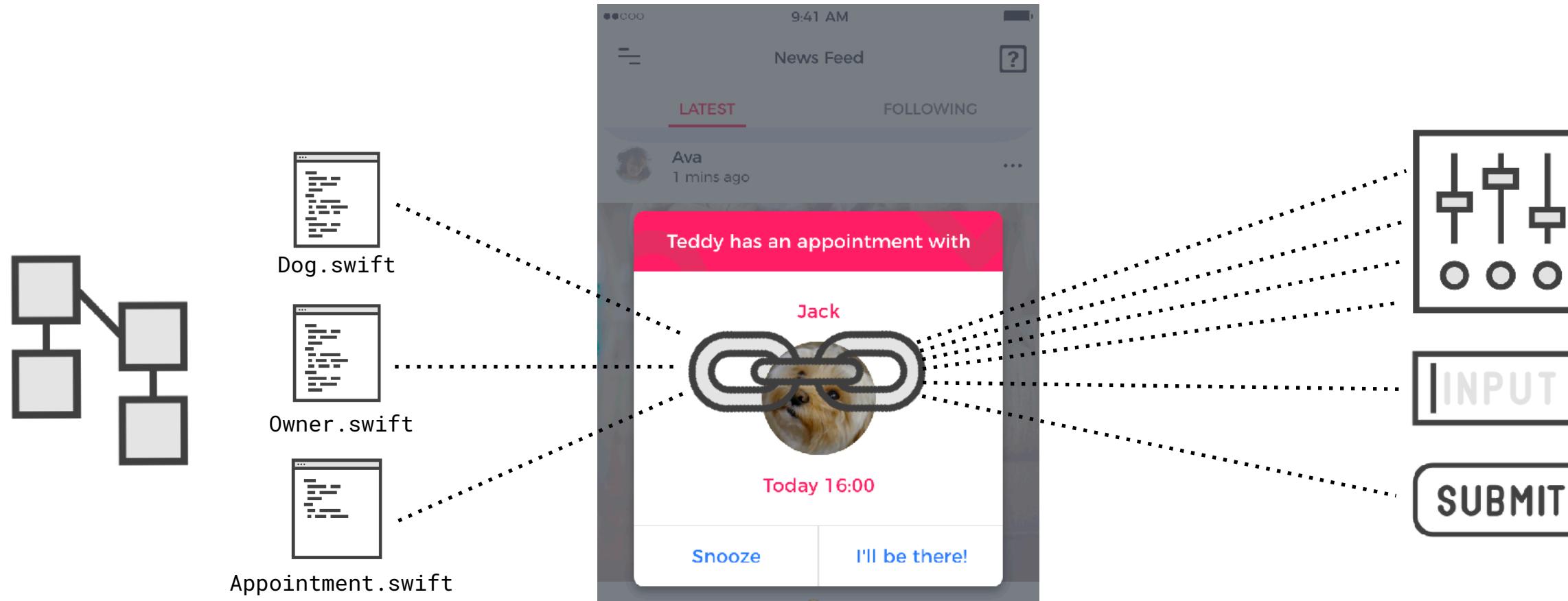
Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

Model-View-Controller in iOS



Model

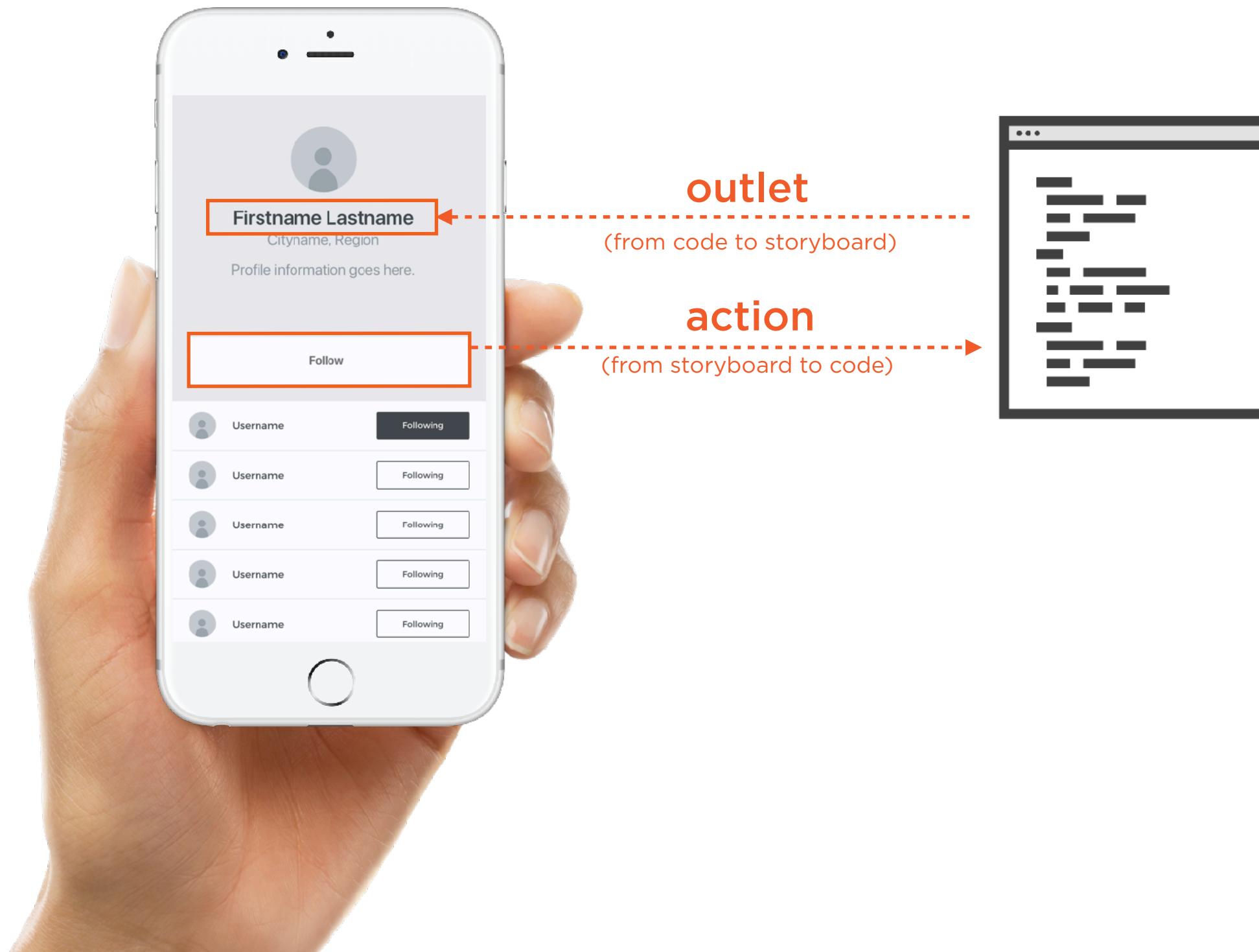
Defines data / business logic
—but not the presentation

To connect the model to the view, and the view to the model

View

Presentation / user interaction—
no business logic, and no storage
or calculation of data

Actions and Outlets



Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14

Developer
Skillset

Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture

Swift

Syntax
Best Practices
Working with pre-Swift Technologies

Developer Tools

Xcode
Simulator
Instruments

iOS 14

Developer
Skillset

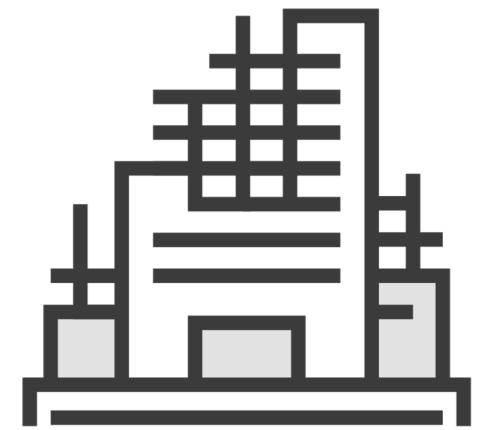
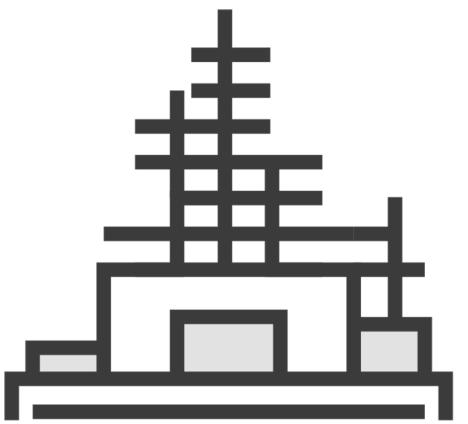
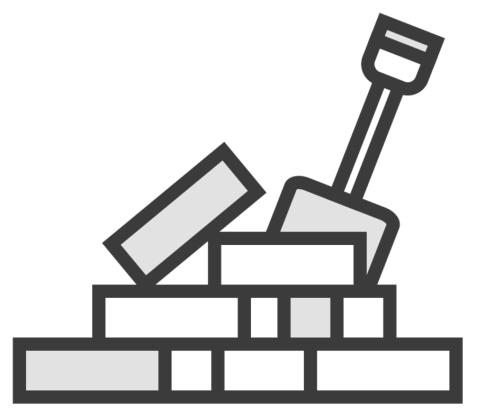
Software Patterns
Human Interface Guidelines
Deployment

iOS Application Lifecycle
Using Existing Frameworks
Device Capabilities

Patterns and Practices

iOS Architecture





The Night Watch Scenario







Next Billion Dollar App





Mountain View
Motel



Patrol the building

Check all windows

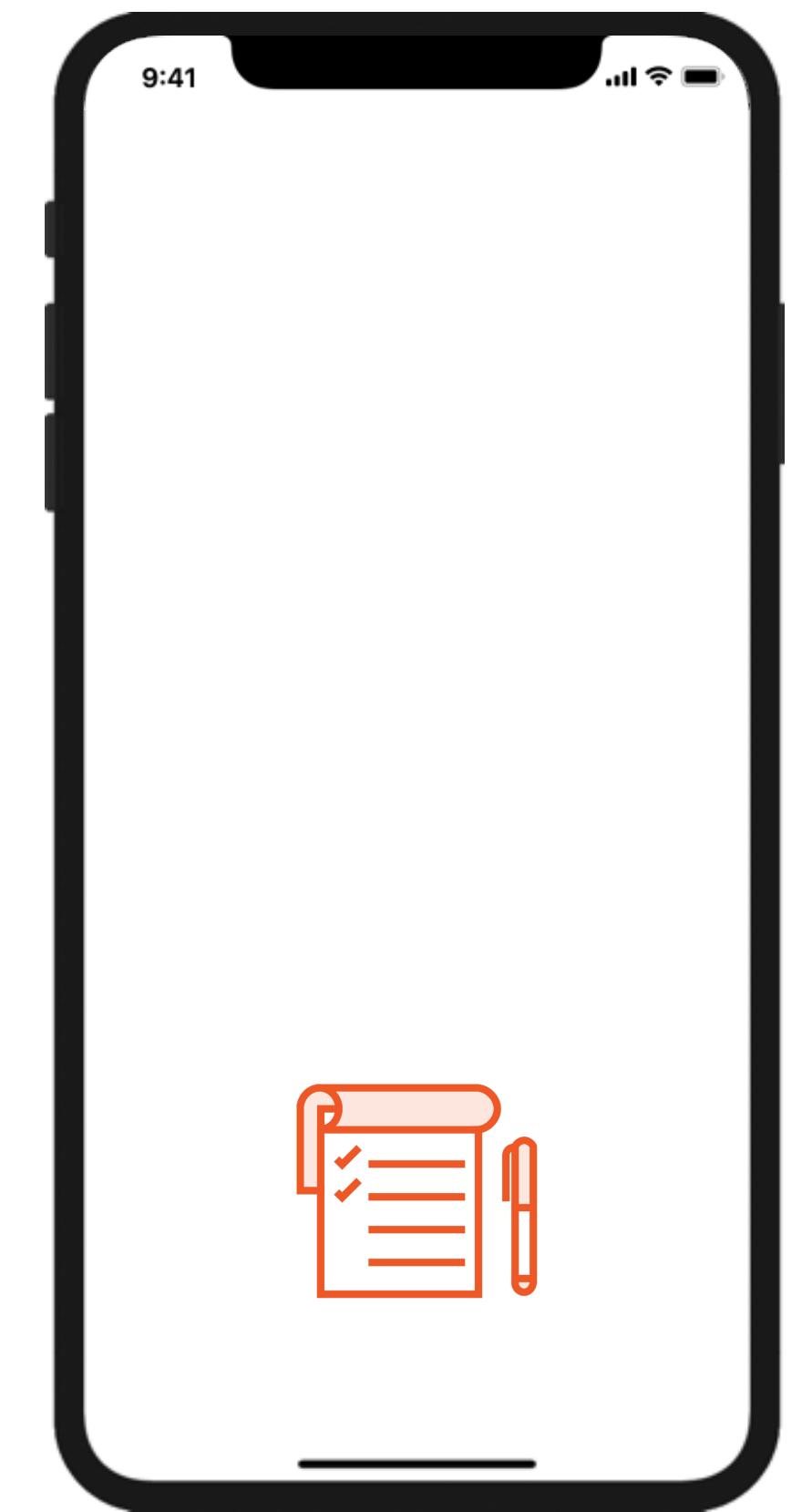
Check all doors

Walk all the guest floors

Check the storage areas

Go down to the basement to see if the pipes are freezing

Watch out for creepy children in the corridors







Night Watch







9:41

 **NIGHTLY TASKS**

Check all windows
Check all doors
Check that the safe is locked
Check the mailbox
Inspect security cameras
Clear ice from sidewalks
Document "strange and unusual" occurrences

WEEKLY TASKS**MONTHLY TASKS**



9:41

NIGHTLY TASKS

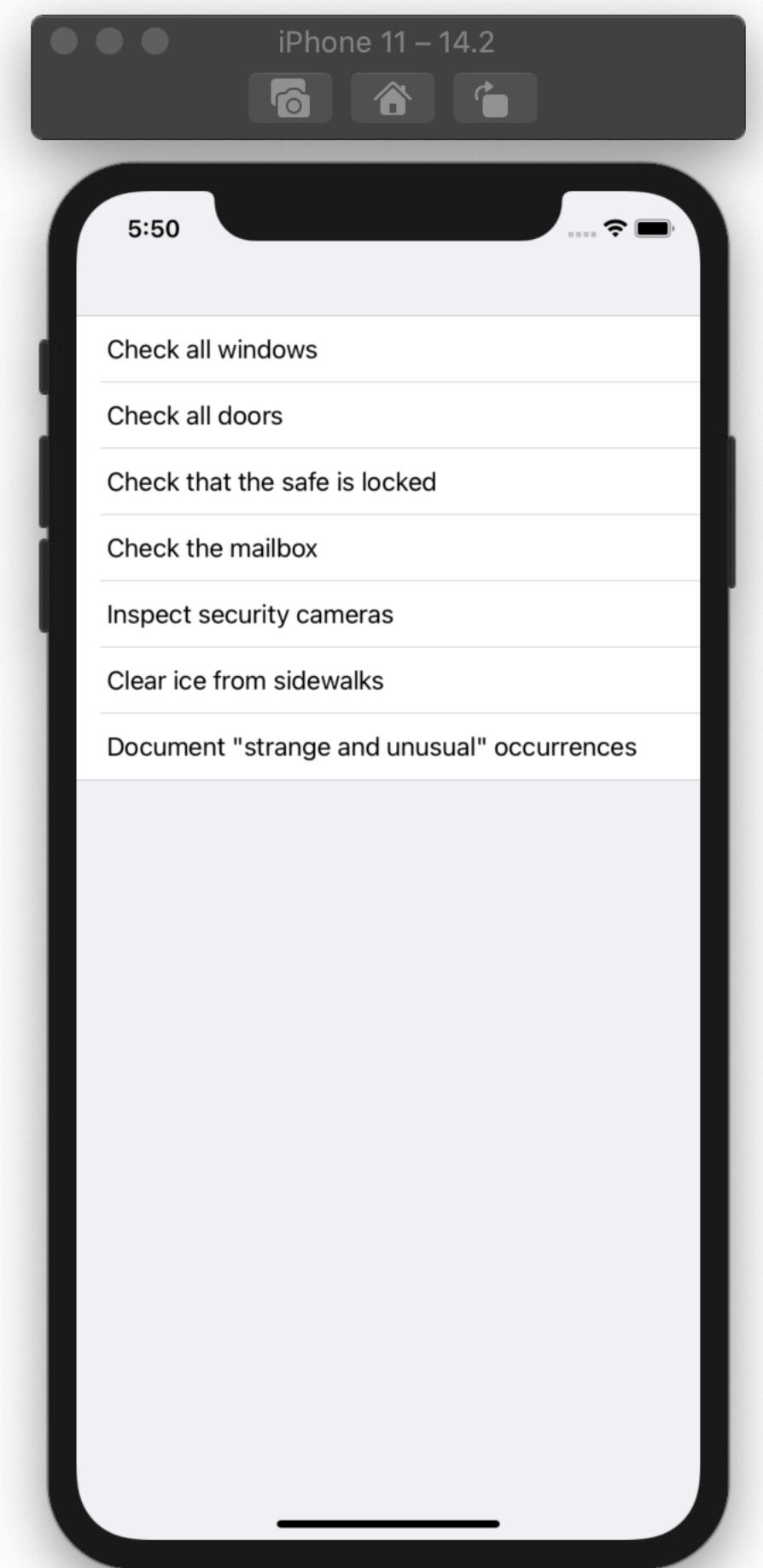
- Check all windows
- Check all doors
- Check that the safe is locked
- Check the mailbox
- Inspect security cameras
- Clear ice from sidewalks
- Document "strange and unusual" occurrences

WEEKLY TASKS

- Check inside all vacant rooms
- Walk the perimeter of property

MONTHLY TASKS

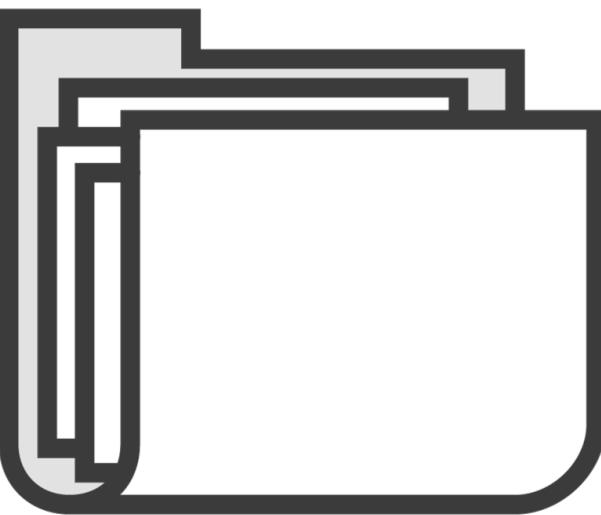
- Test security alarm
- Test motion detectors
- Test smoke alarms



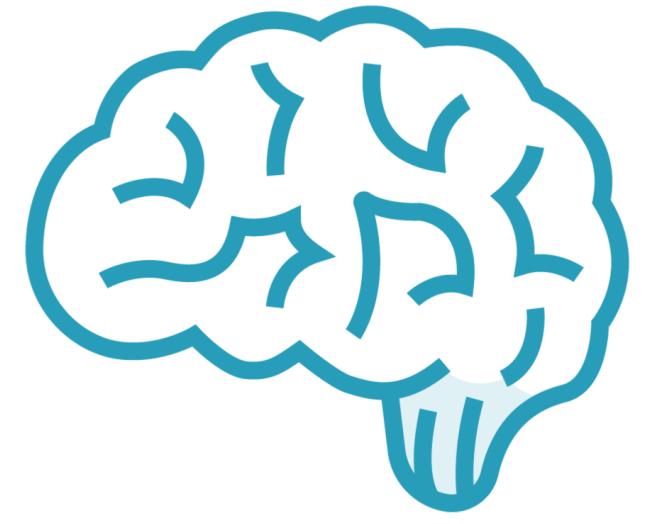
Working with Xcode Projects



Conceptual

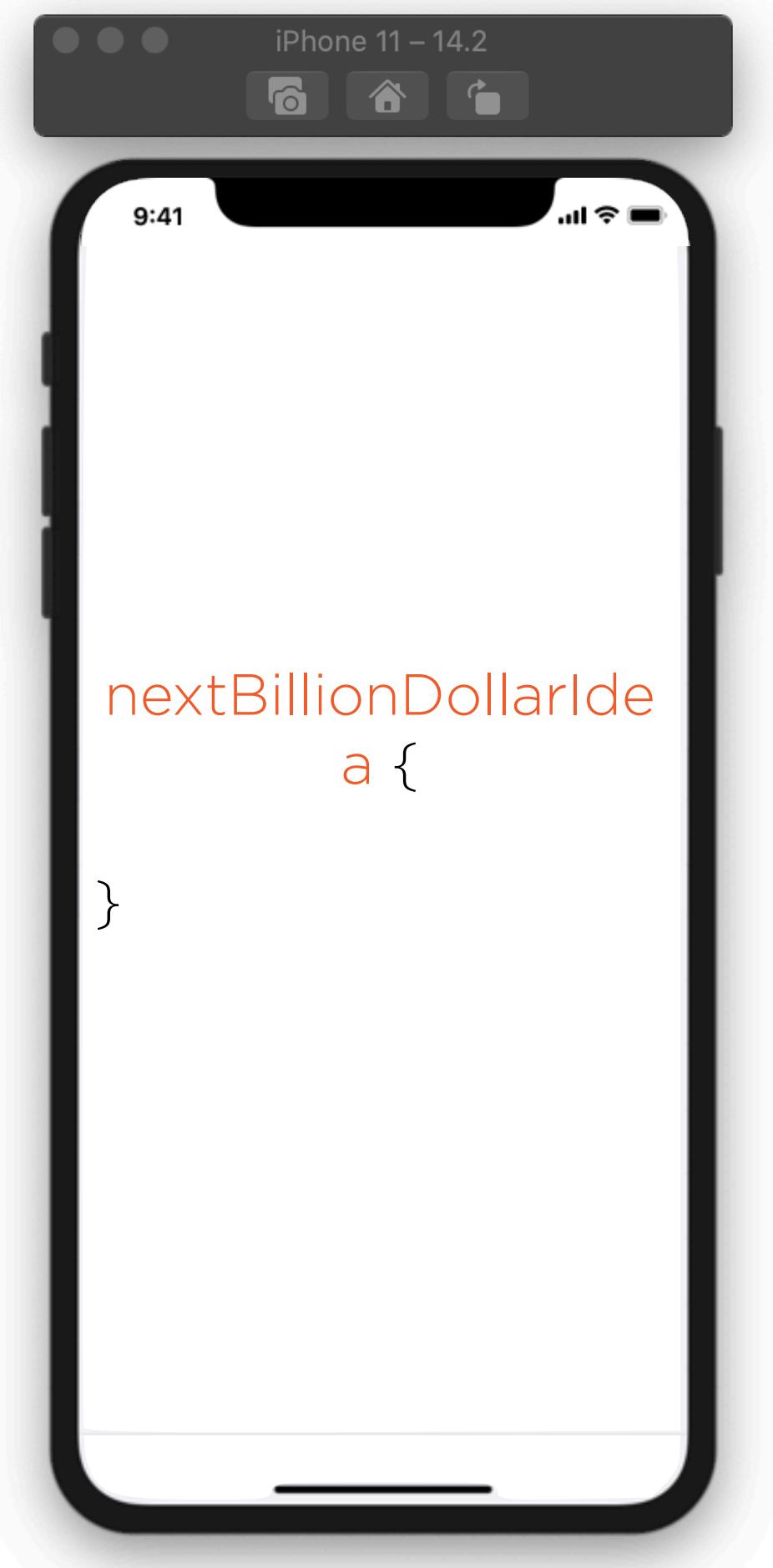


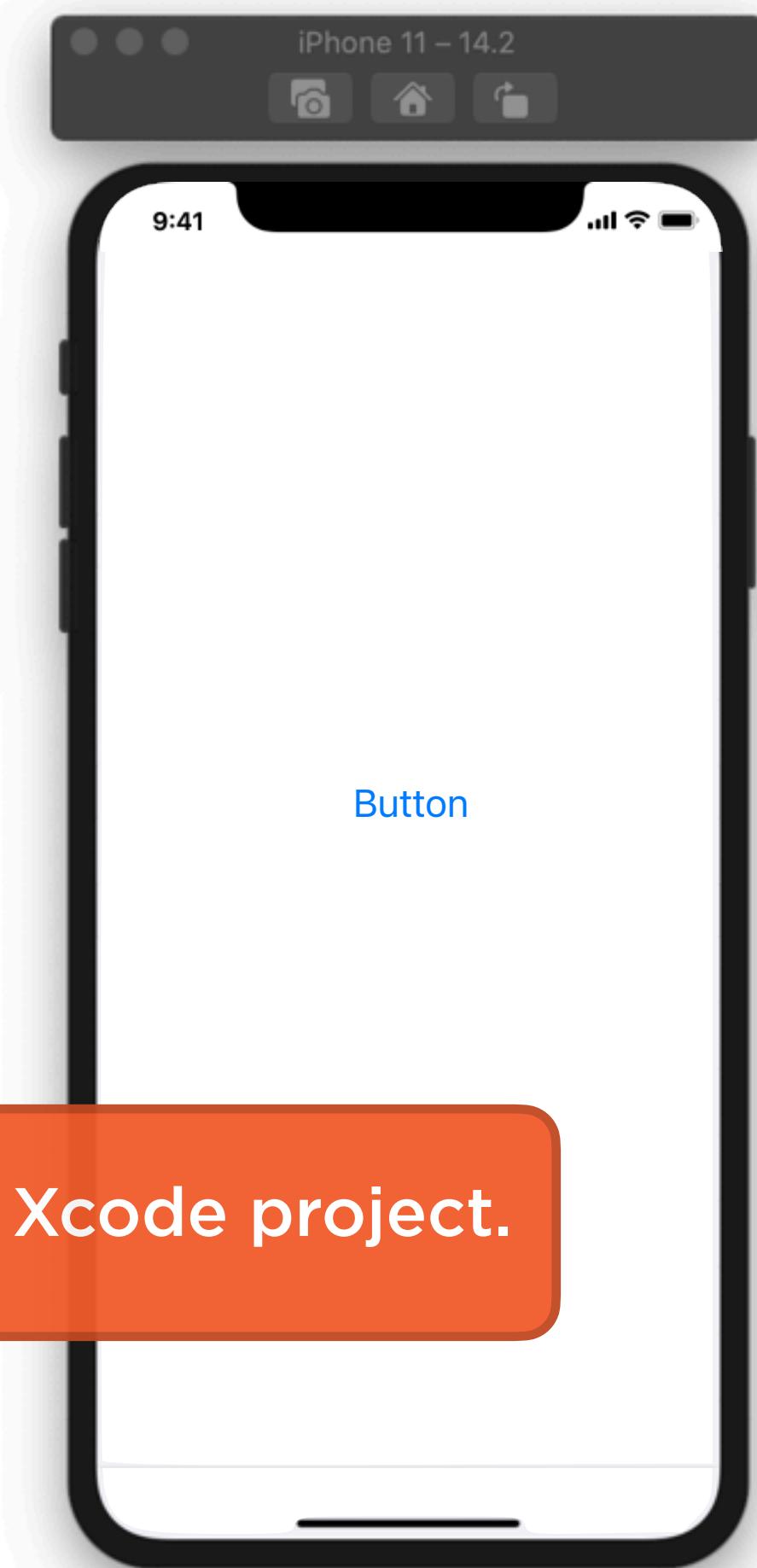
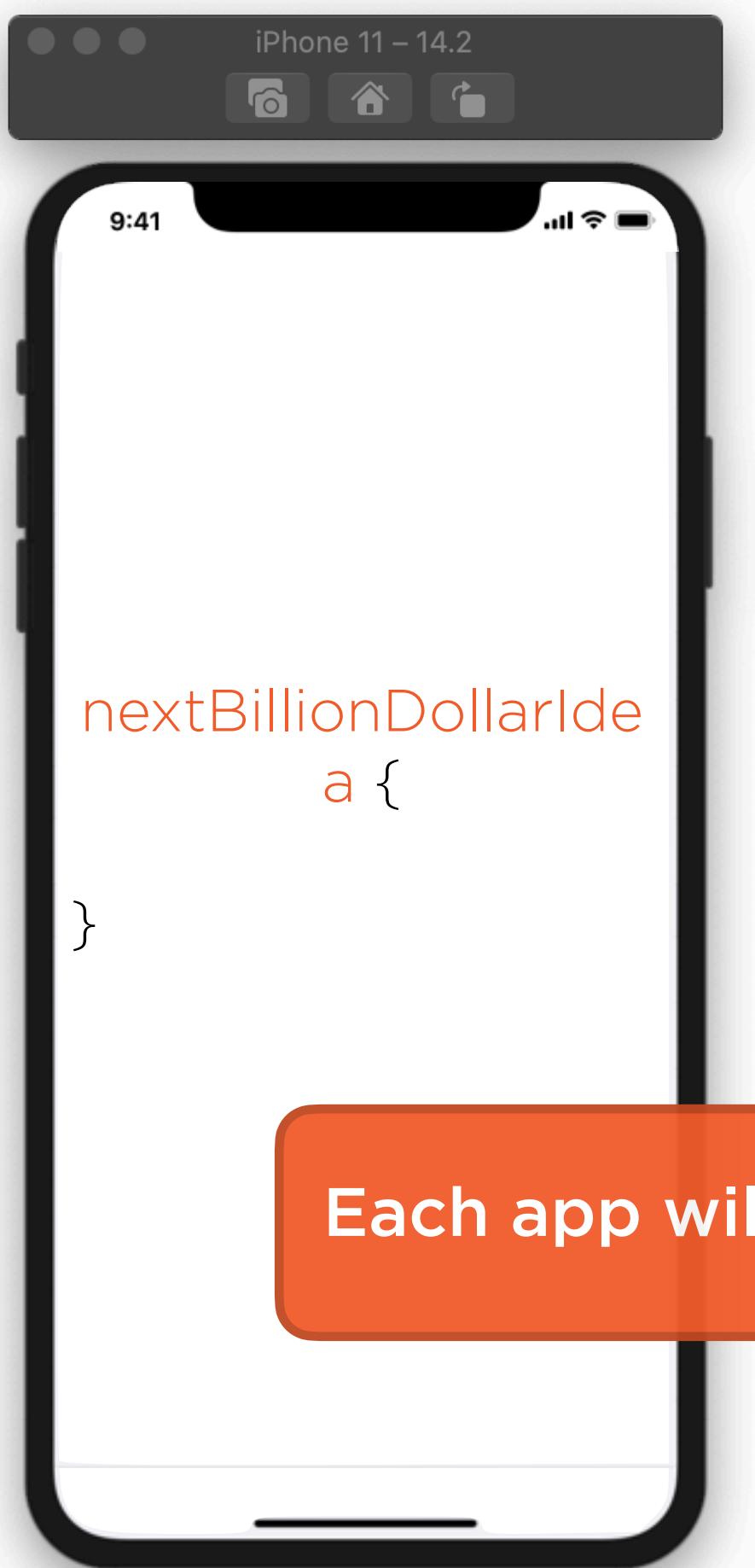
Physical

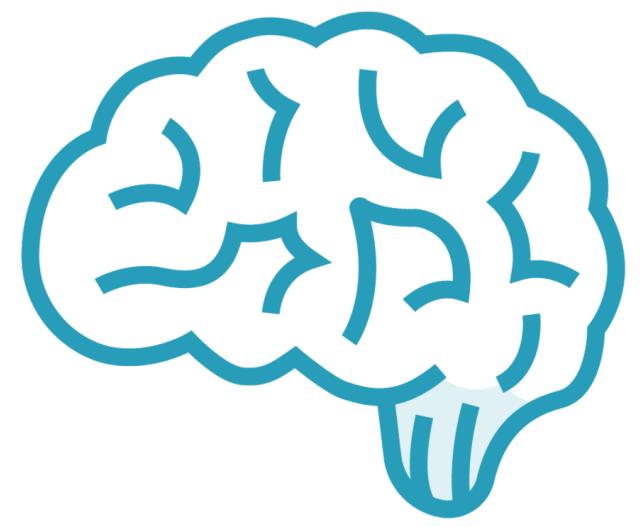


Conceptual

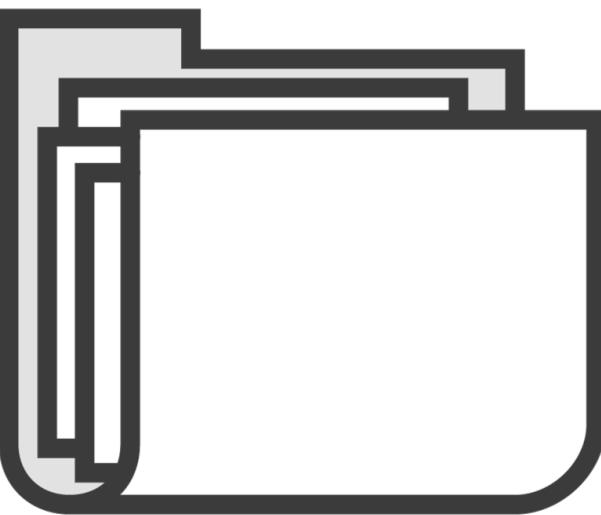
App = Project



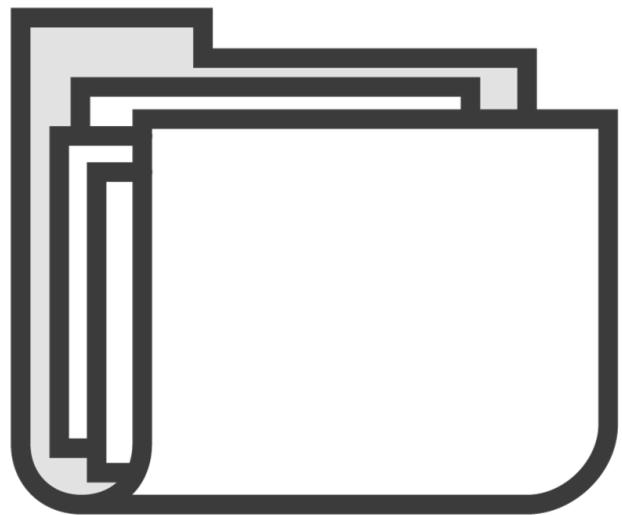




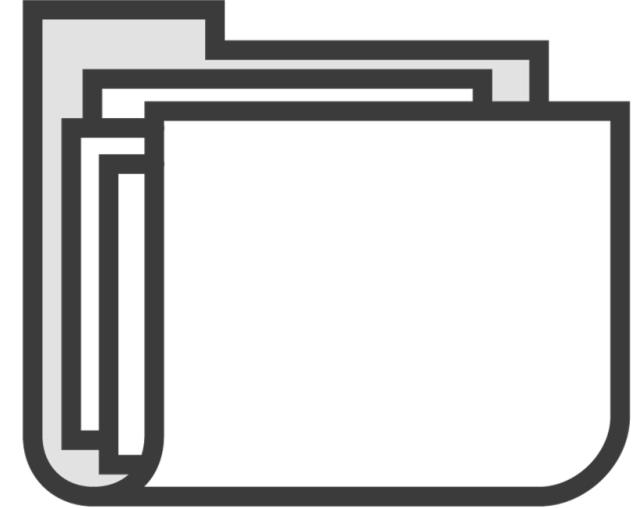
Conceptual



Physical

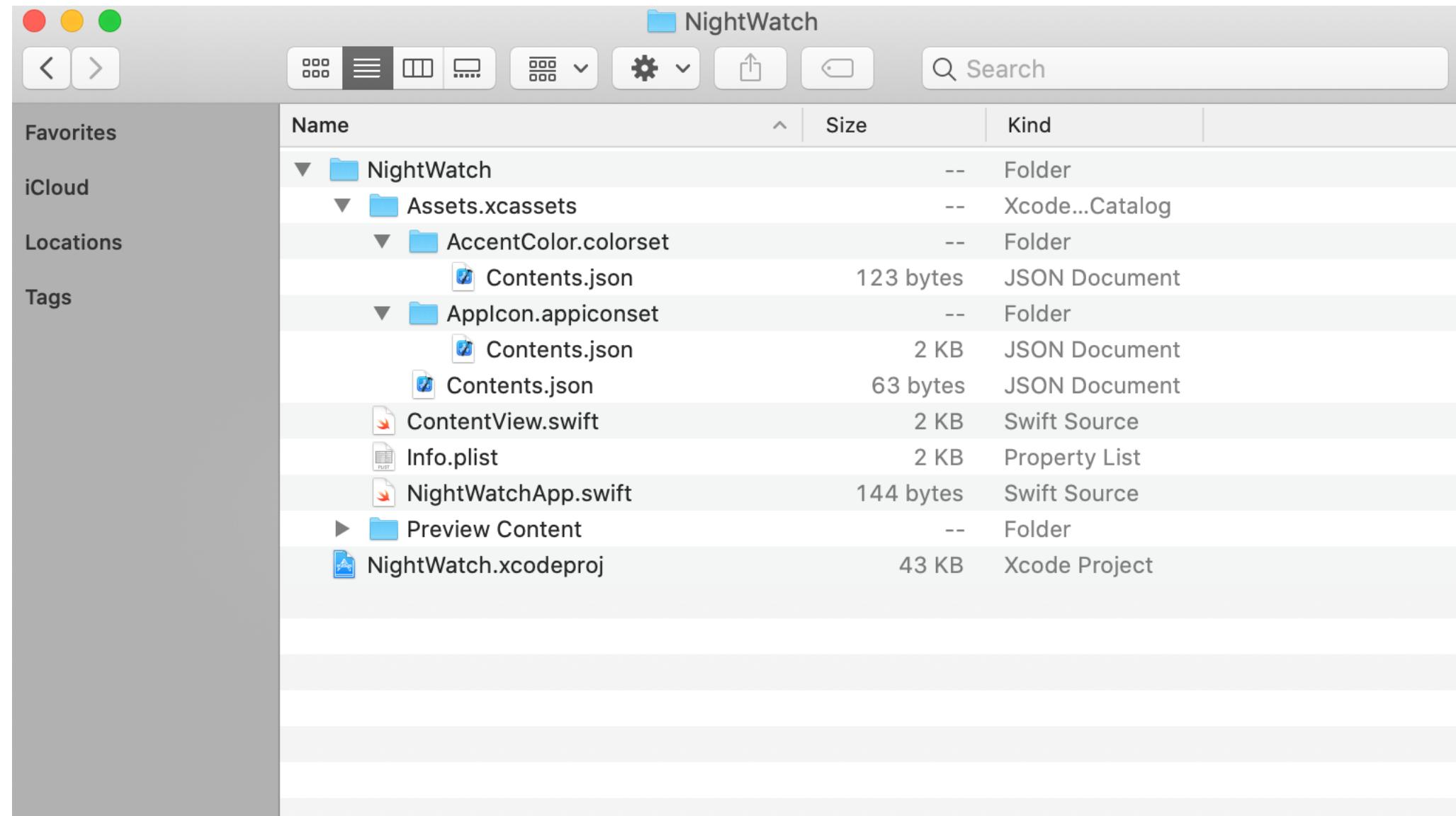


Physical



Xcode Project

Add new files
Delete files
Rename files
Move files
Organize files



Not in a Finder window

Grasping the Basics of the Xcode Development Environment

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure with files: NightWatchApp.swift, ContentView.swift, Assets.xcassets, Info.plist, Preview Content, and Products.
- Editor:** The ContentView.swift file is open. The code defines a struct ContentView that returns a Text view with the content "Hello, world!" and padding. It also defines a struct ContentView_Previews with a PreviewProvider.
- Preview Area:** A preview of the ContentView is displayed, showing the text "Hello, world!" in a black font with a bounding box of approximately [180, 360, 300, 640].
- Text Inspector:** On the right, the Text inspector is open, showing settings for the selected text:
 - Text:** Hello, world!
 - Modifiers:** None
 - Font:** Inherited
 - Weight:** Inherited
 - Color:** Inherited
 - Alignment:** Center
 - Line Limit:** Inherited
- Padding Inspector:** Shows a padding value of Default.
- Frame Inspector:** Shows size values of Inher for both width and height.
- Add Modifier:** A button to add more modifiers.

Text Labels:

- everything** (in blue) is located in the bottom left corner of the Xcode interface.
- the one thing being edited** (in red) is located in the bottom center of the Xcode interface.
- one piece of the thing being edited** (in green) is located in the bottom right corner of the Xcode interface.

Summary

Building Single View Applications



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

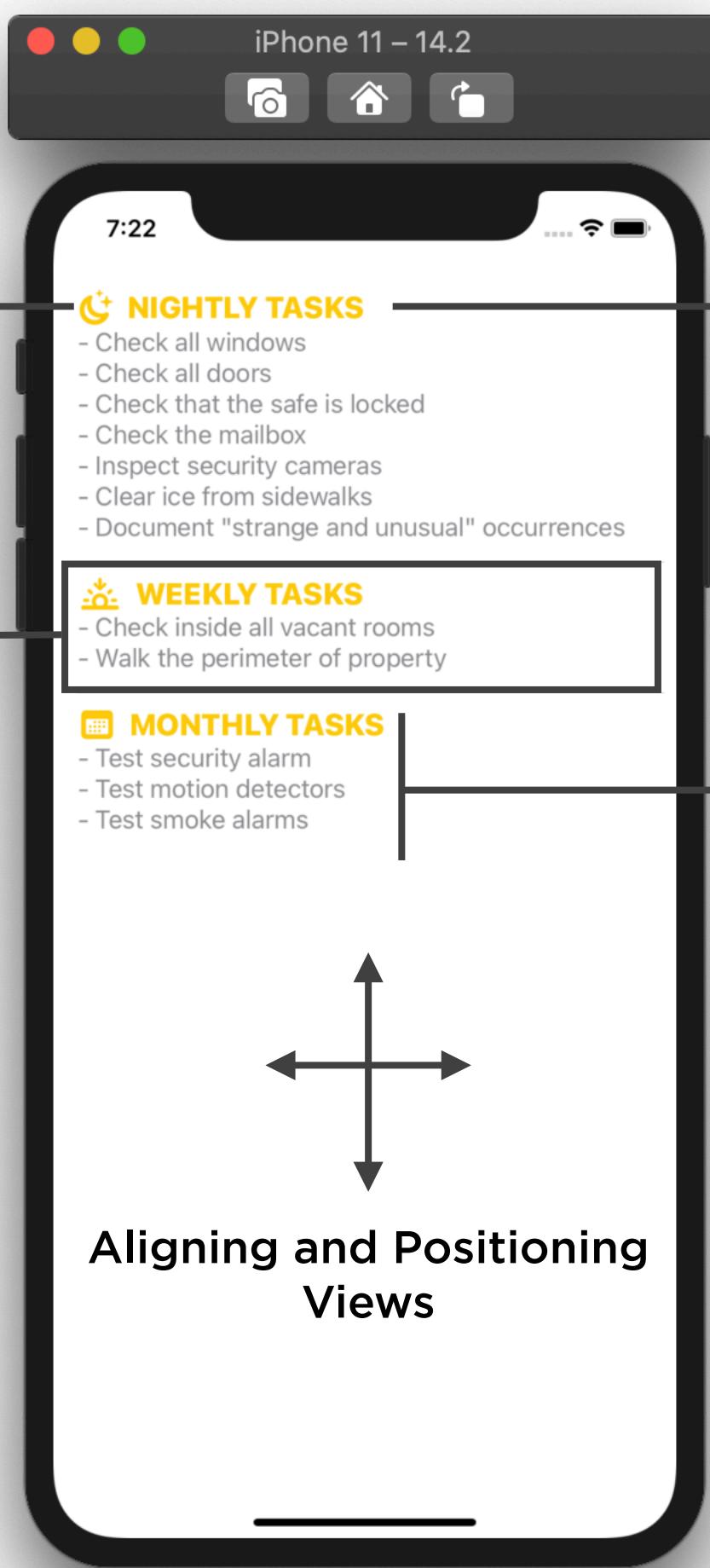


Image Views

Text Views

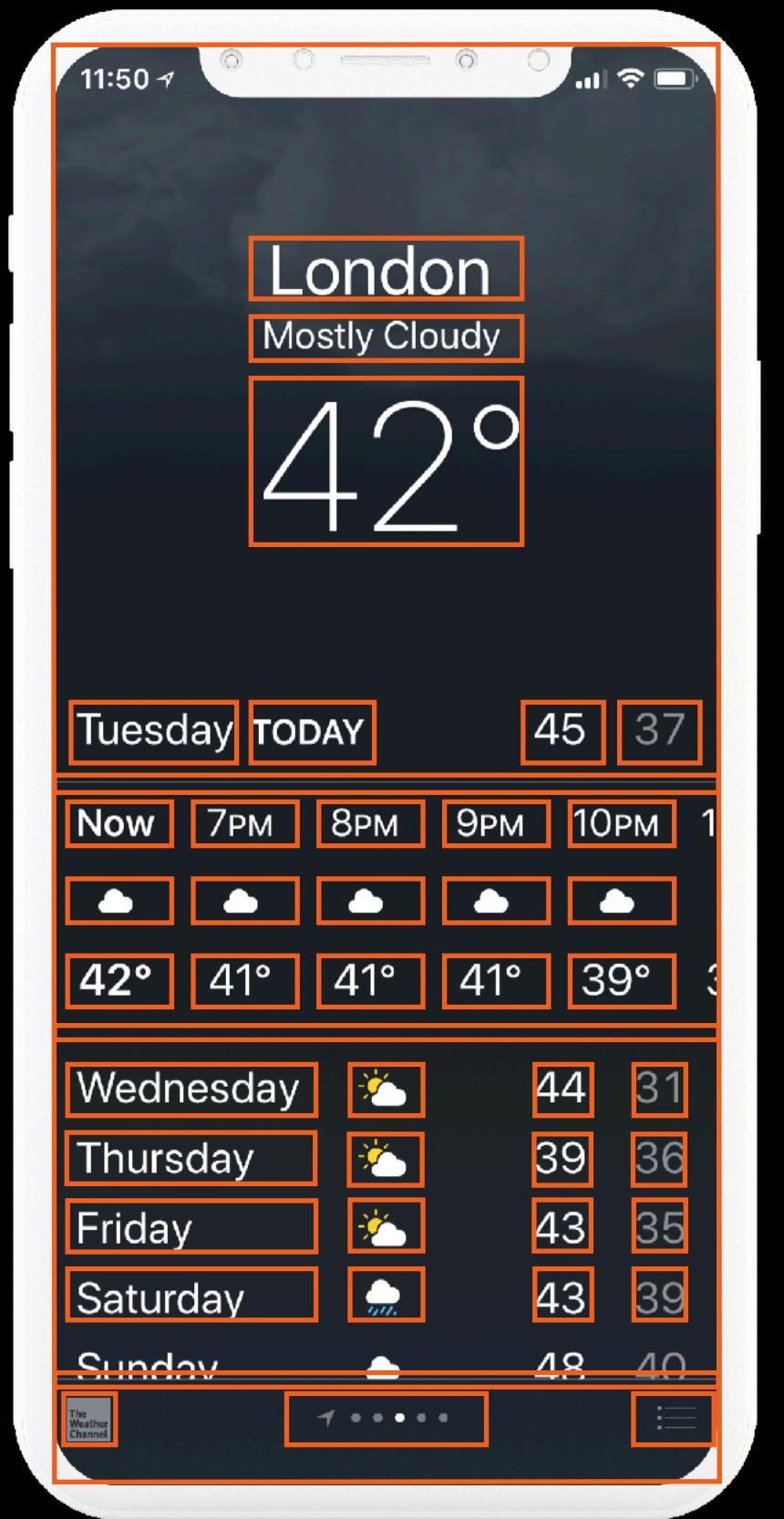
Combining Views

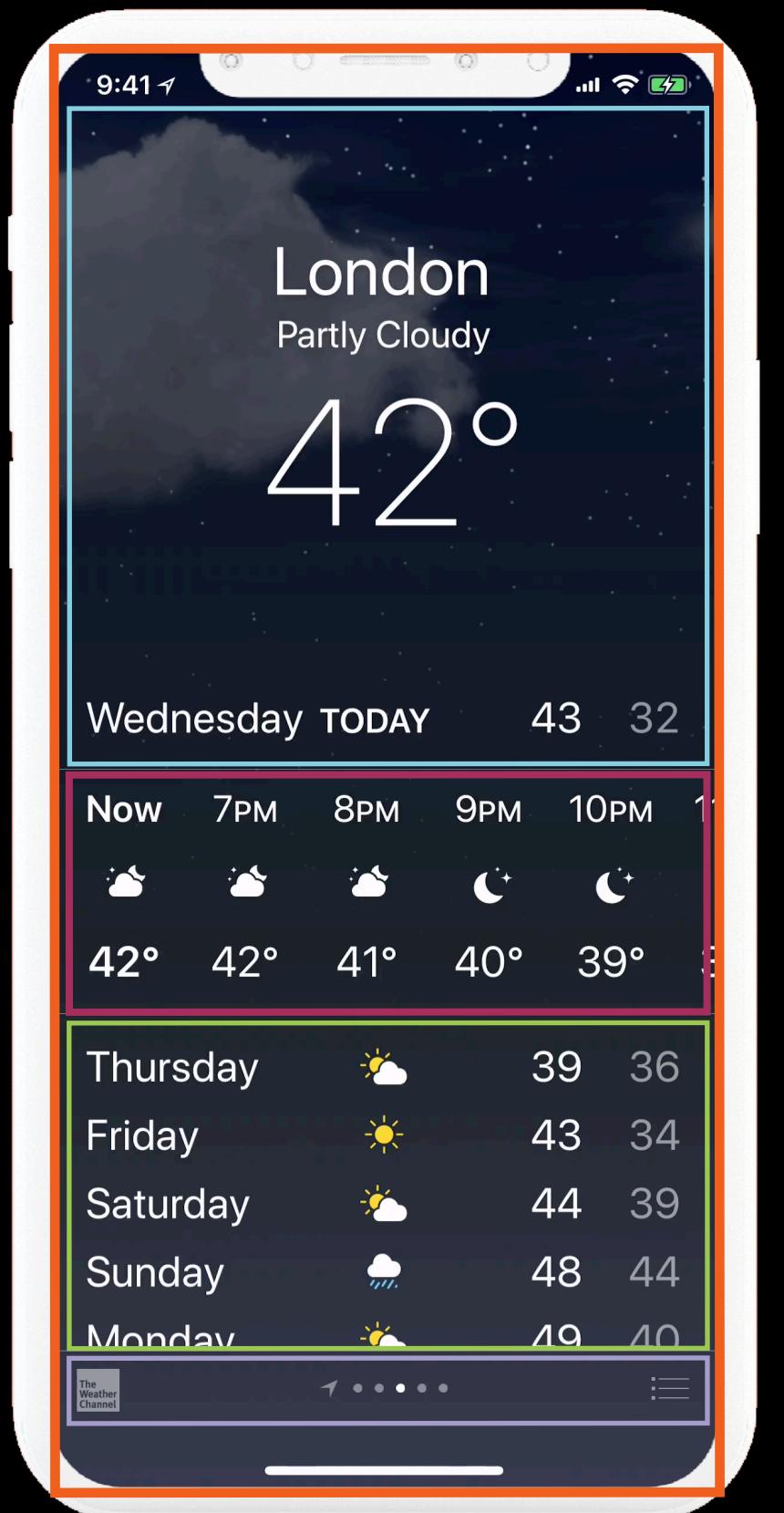
Modifying View Appearance

Aligning and Positioning Views



View







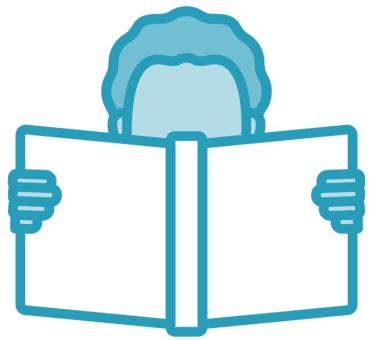
Swift**UI**



Swift**UI**



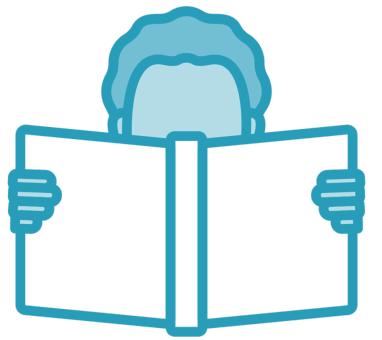
“What is the SwiftUI framework? What does it provide?”



Swift**UI**



“What is the SwiftUI framework? What does it provide?”



Swift**UI**



“What is the SwiftUI framework? What does it provide?”



“What does the word **View really mean in iOS?”**



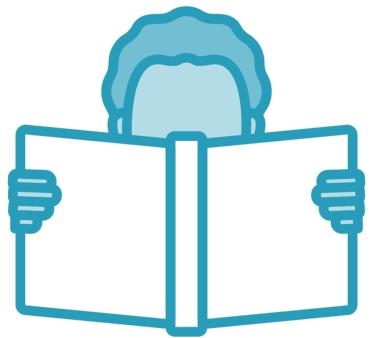
Swift**UI**



“What is the SwiftUI framework? What does it provide?”



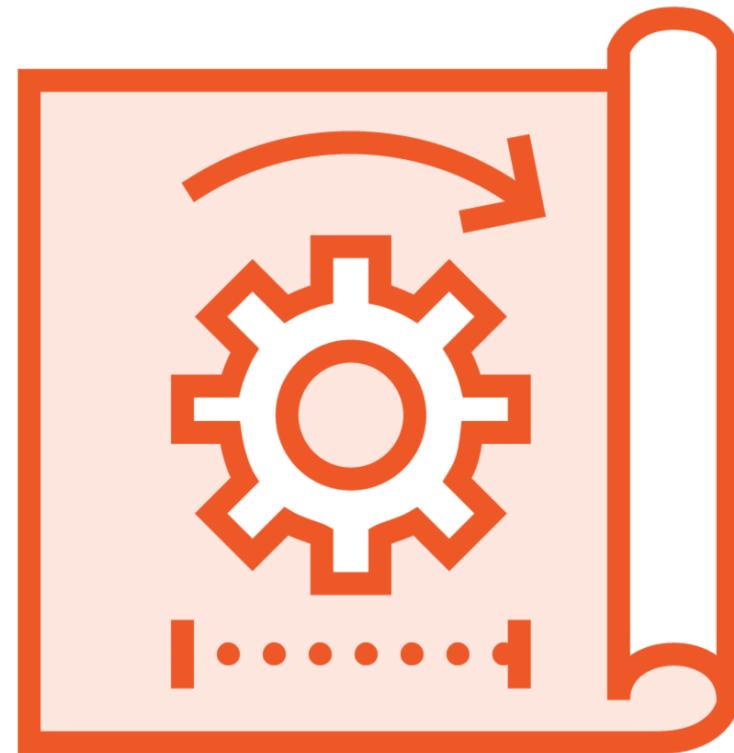
“What does the word **View really mean in iOS?”**

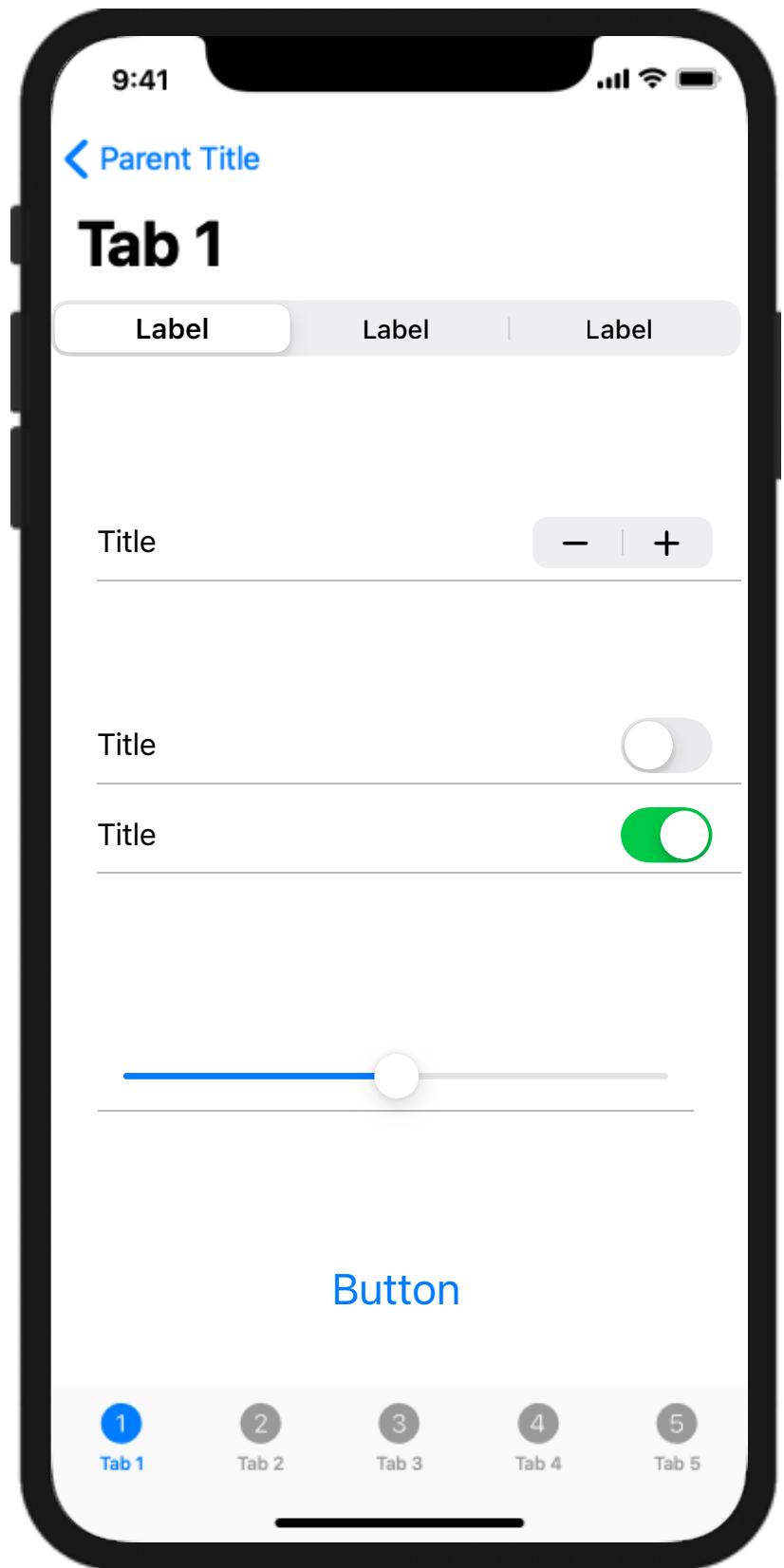


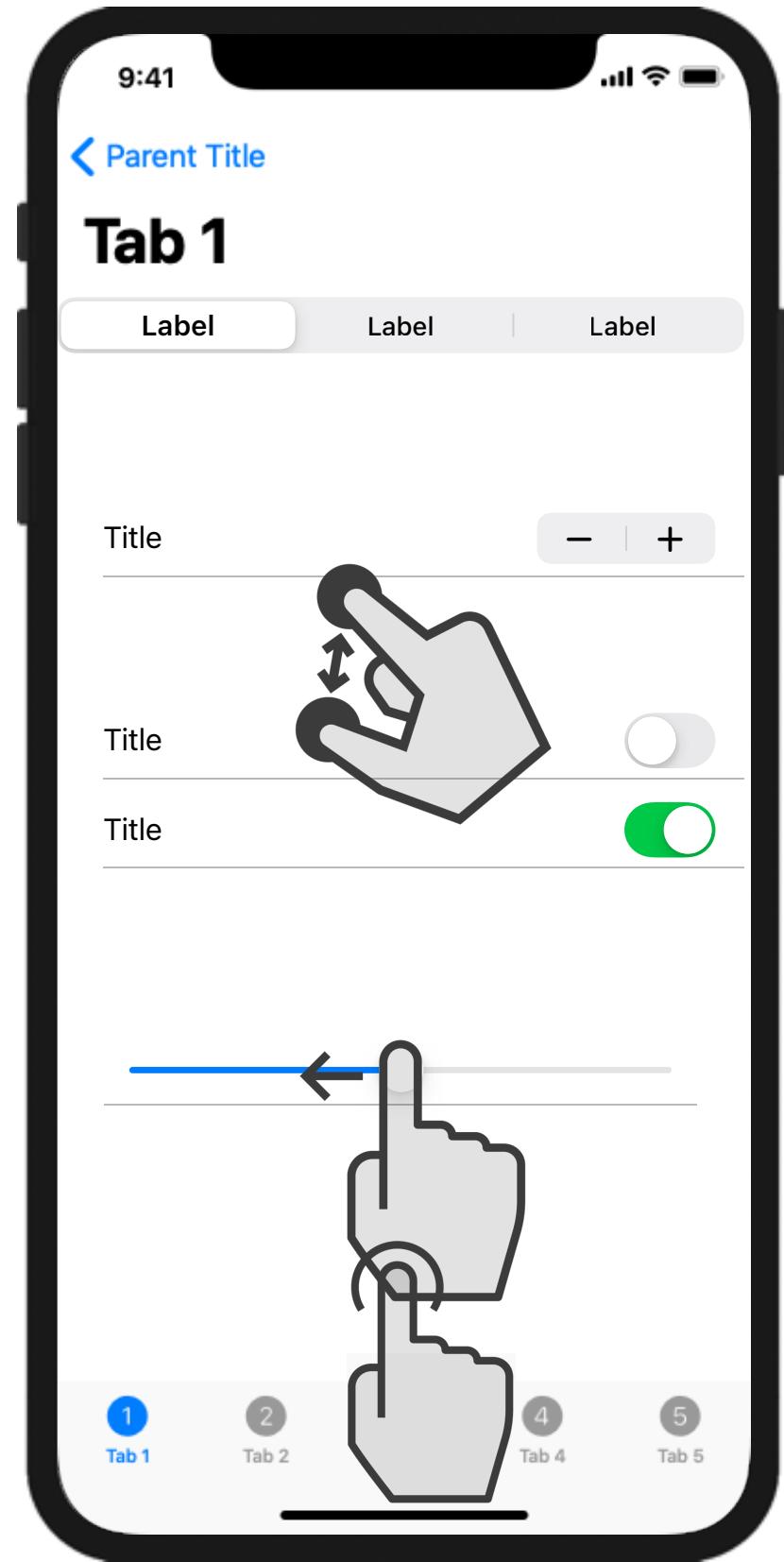
Swift**UI**

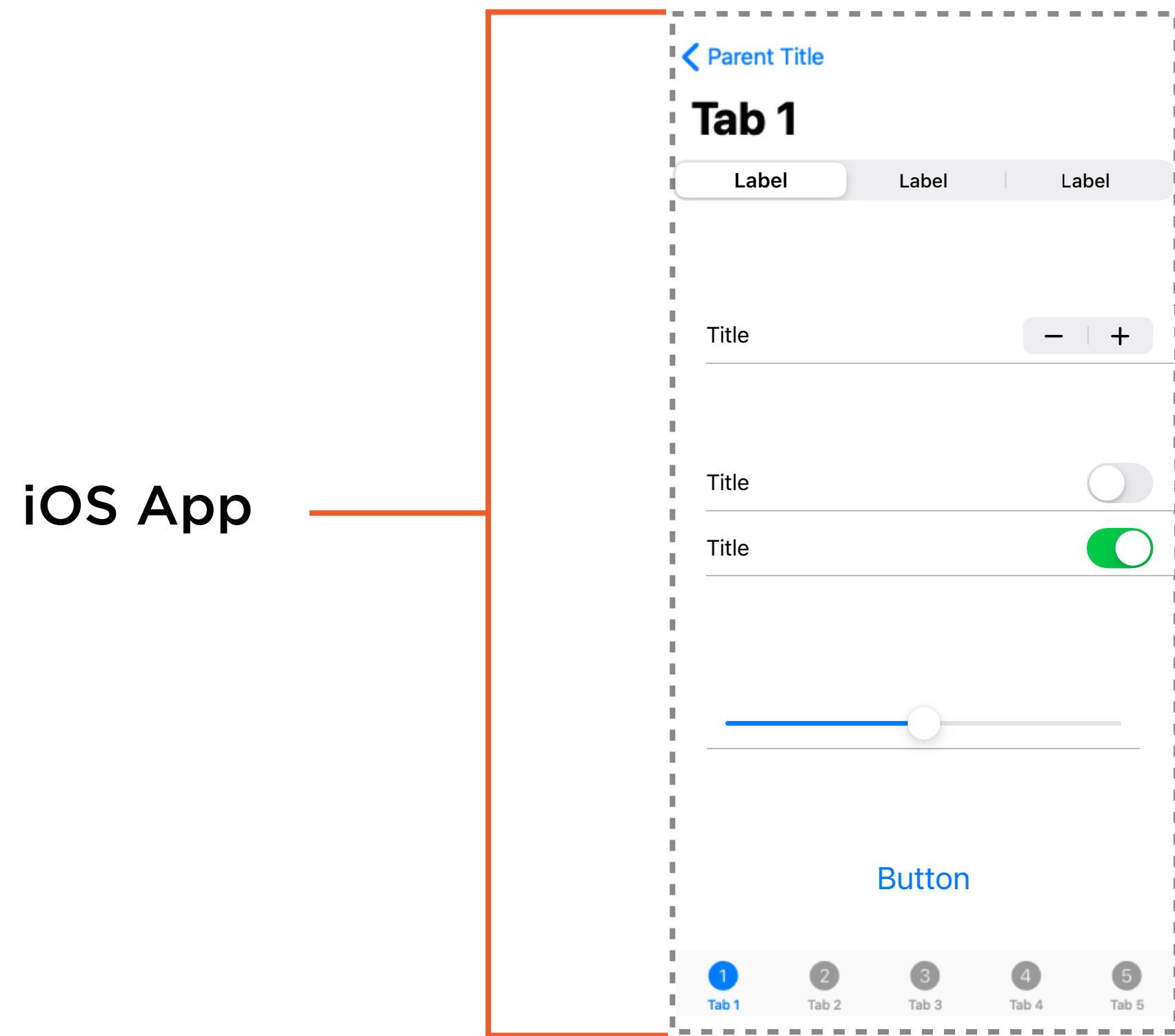
- ? **“What is the SwiftUI framework? What does it provide?”**
- ? **“What does the word **View** really mean in iOS?”**
- ? **“How do we combine views, lay them out, and customize them?”**

Understanding the SwiftUI Framework



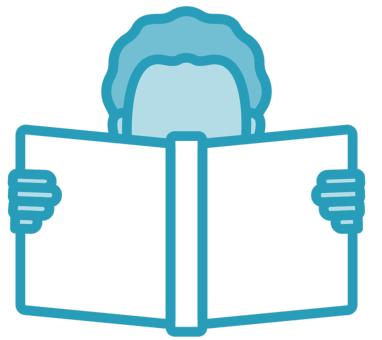




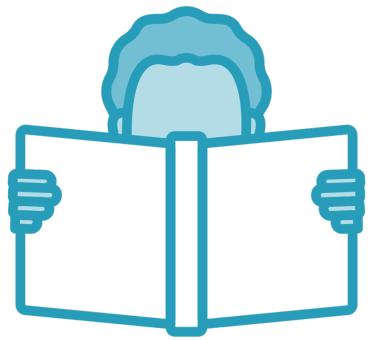


Application Lifecycle

Event Handling



Swift**UI**



Swift**UI**



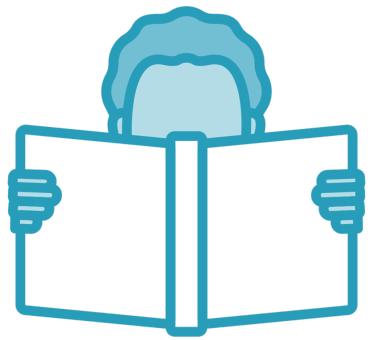
“How do we **tap in to all of that ‘stuff’?”**



Swift**UI**



“How do we **tap in to all of that ‘stuff’?”**



SwiftUI



“How do we **tap in to all of that ‘stuff’?”**



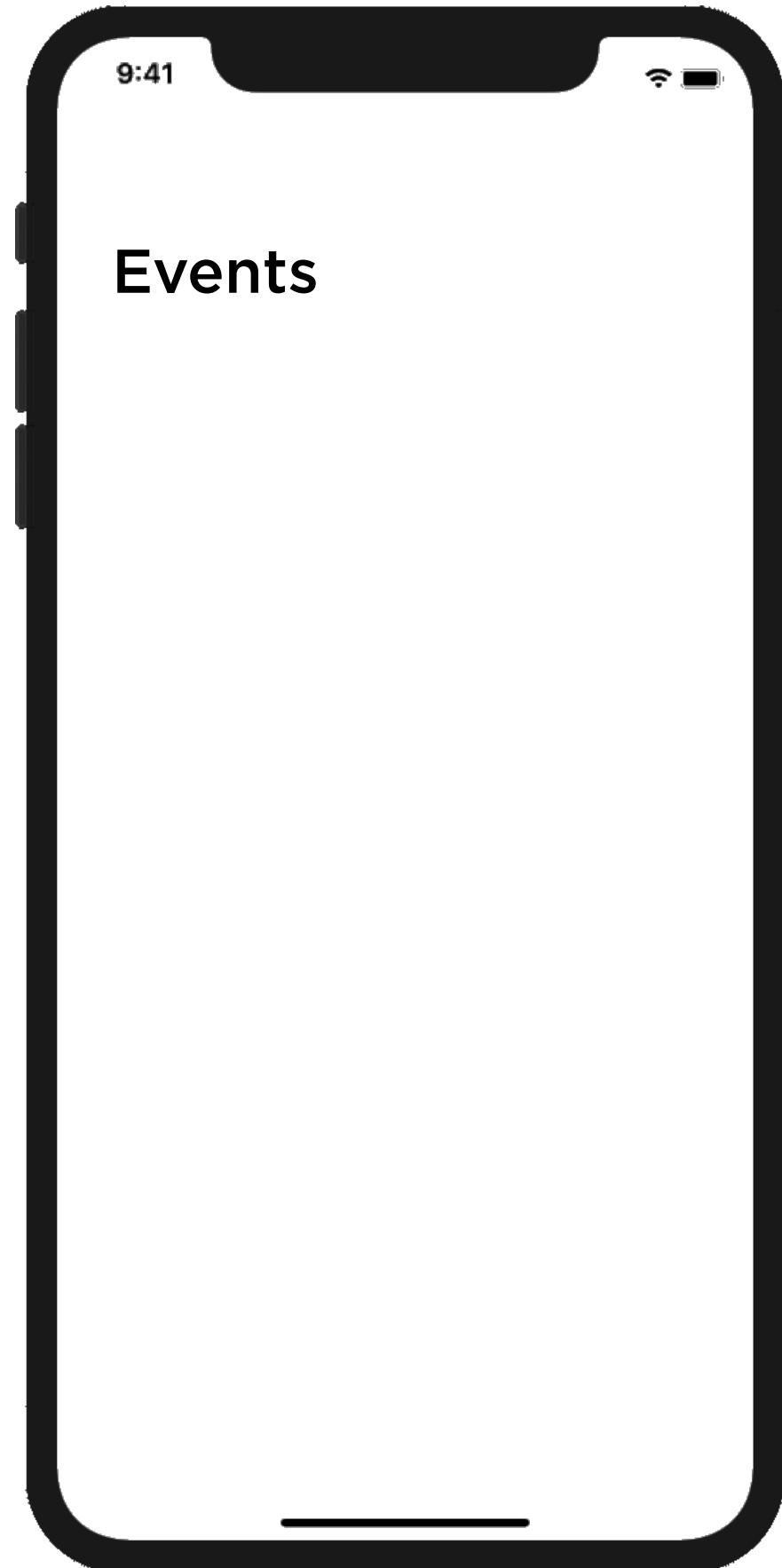
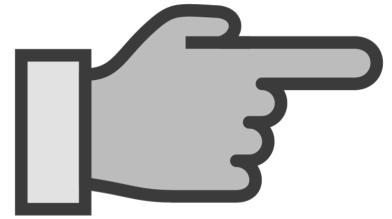
“How do we **use the Apple-provided infrastructure components?”**

“Here, link to all the stuff Apple wrote to support making an iOS 14 application. Not just user interface, but all the infrastructure stuff as well.”

SwiftUI is the **core framework**
we need in this course for
building apps.

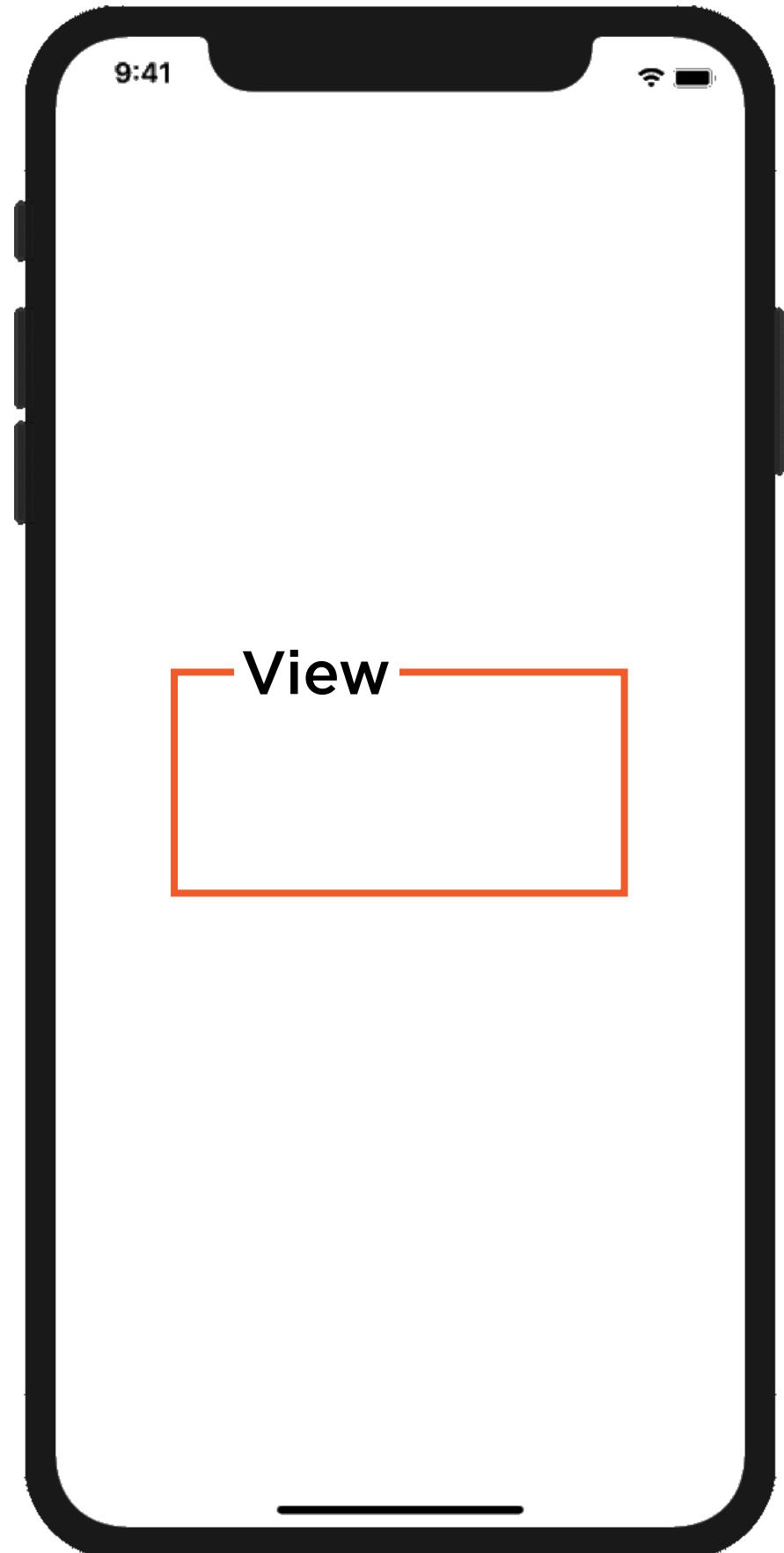
Understanding Views

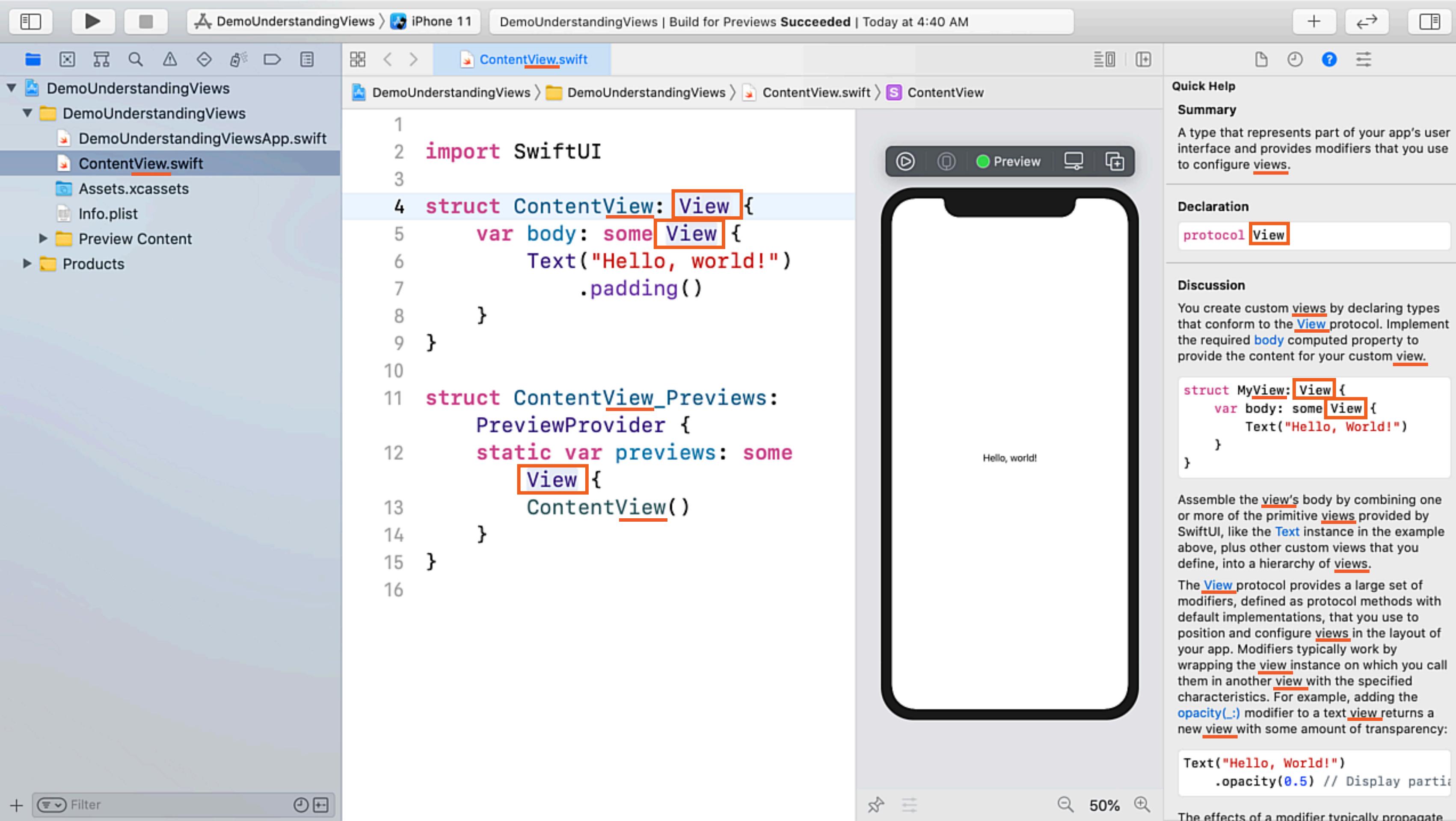
“Okay, so on this screen the user will see a **view of all the upcoming events...”**



SwiftUI View

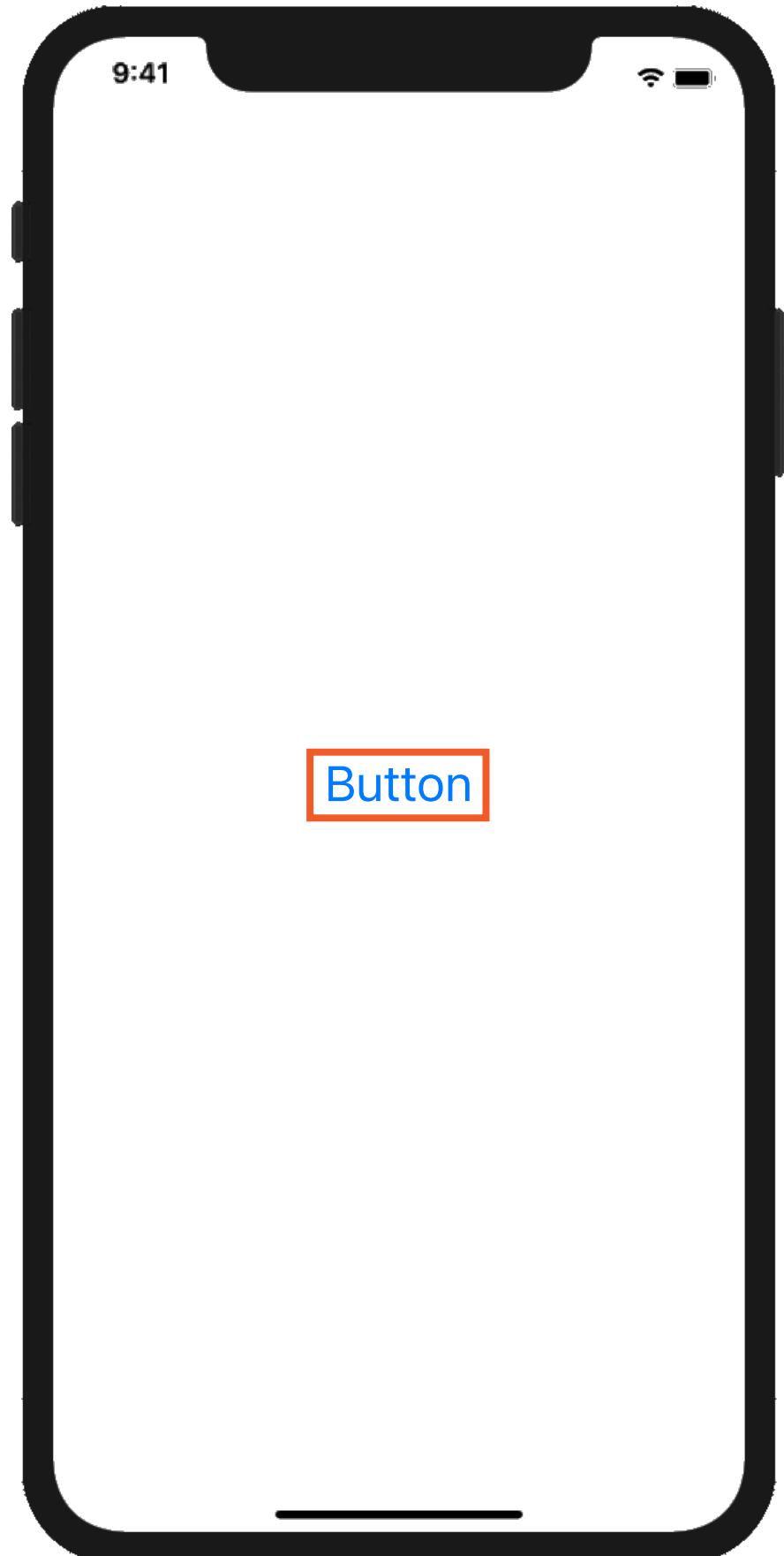
A description of anything that is capable of being drawn in a rectangular region on an iOS screen.





SwiftUI View

A description of anything that is capable of being drawn in a rectangular region on an iOS screen.



SwiftUI View

A description of anything that is capable of being drawn in a rectangular region on an iOS screen.



DemoUnderstandingViews > iPhone 11 DemoUnderstandingViews | Build for Previews Succeeded | Today at 4:40 AM + ↻

File Xcode ContentView.swift

DemoUnderstandingViews > DemoUnderstandingViews > ContentView.swift ContentView

```
1 import SwiftUI
2
3
4 struct ContentView: View {
5     var body: some View {
6         Text("Hello, world!")
7             .padding()
8     }
9 }
10
11 struct ContentView_Previews: PreviewProvider {
12     static var previews: some View {
13         ContentView()
14     }
15 }
16
```

Preview

Hello, world!

Quick Help

Summary

A type that represents part of your app's user interface and provides modifiers that you use to configure views.

Declaration

protocol View

Discussion

You create custom views by declaring types that conform to the `View` protocol. Implement the required `body` computed property to provide the content for your custom view.

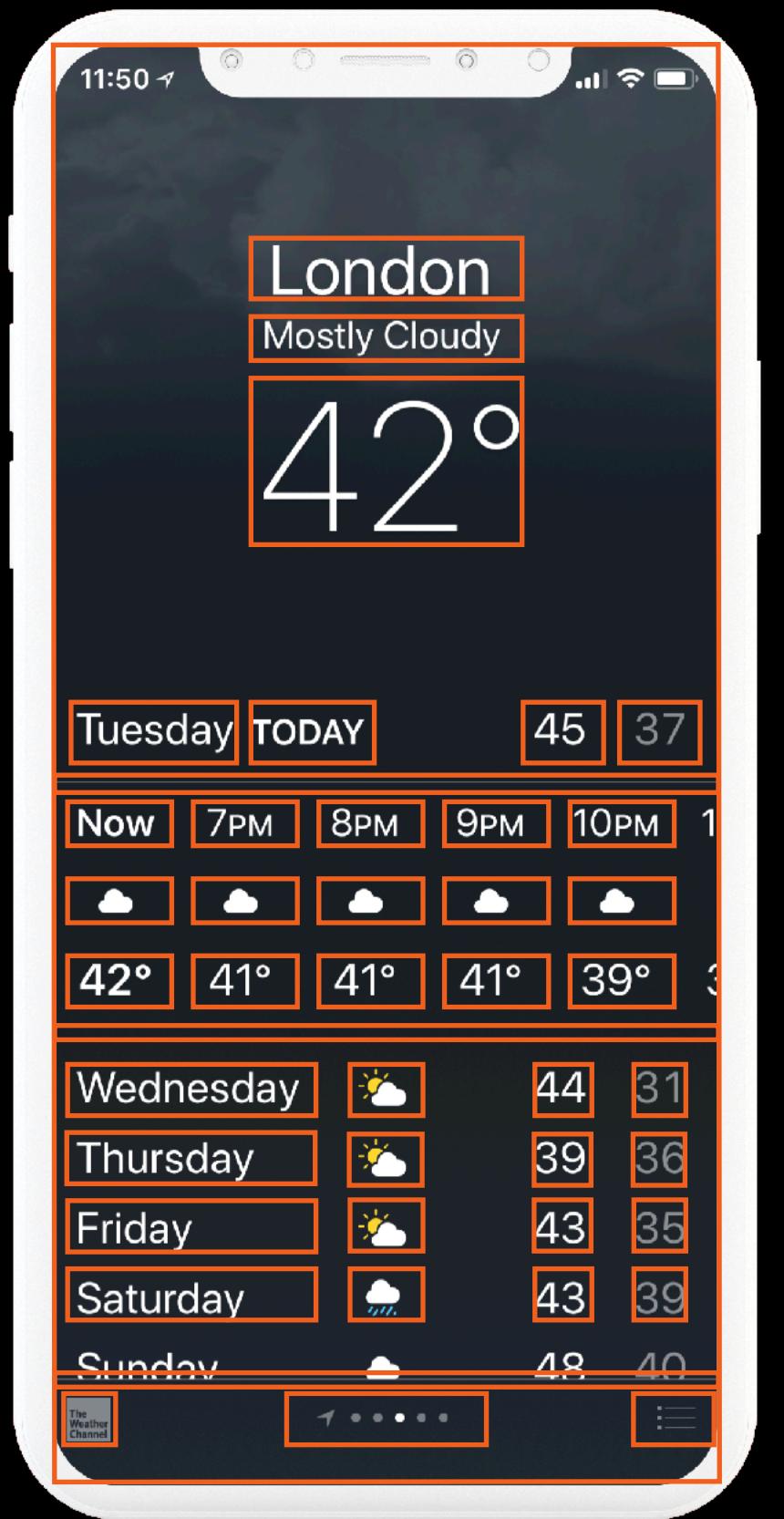
```
struct MyView: View {
    var body: some View {
        Text("Hello, World!")
    }
}
```

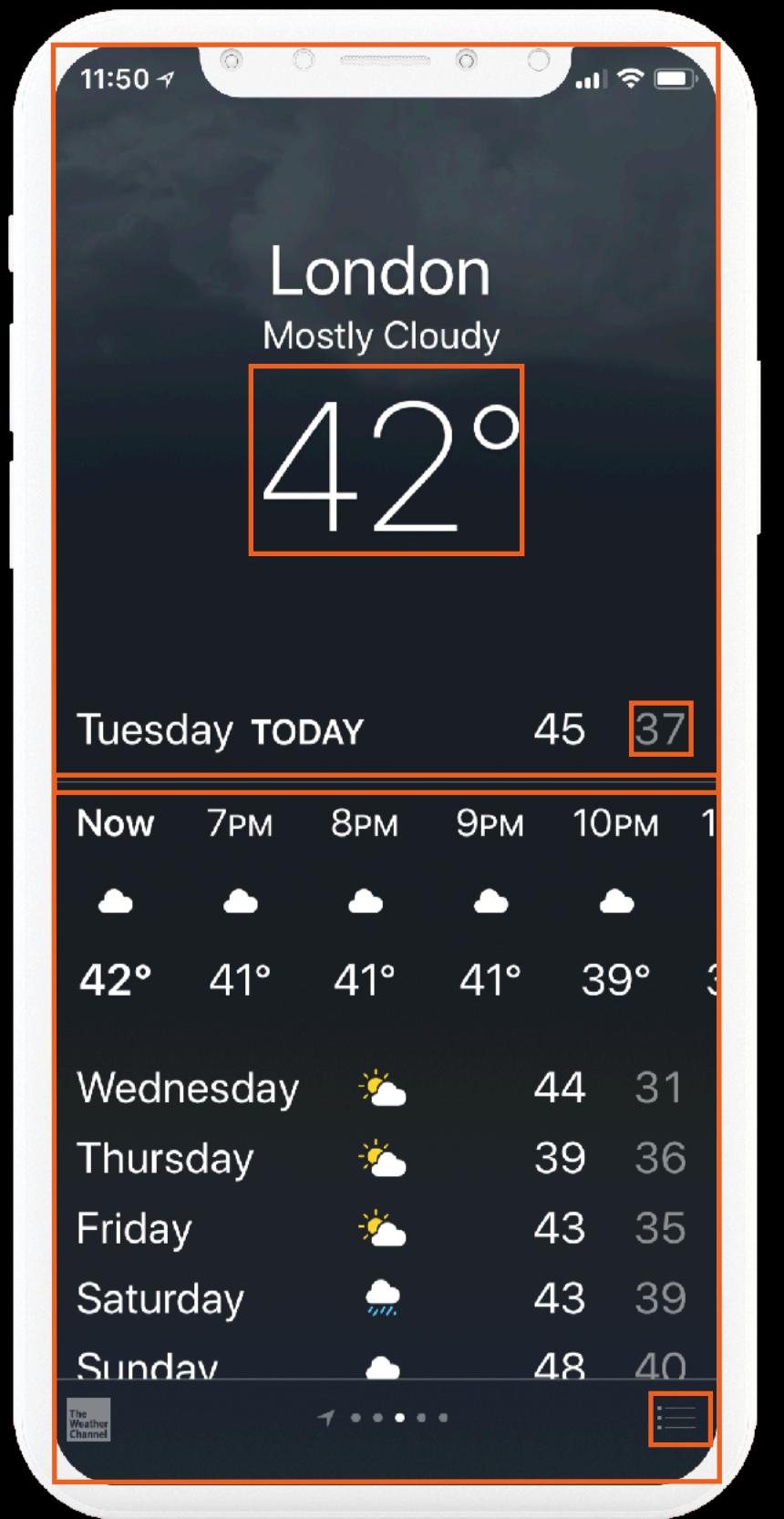
Assemble the view's body by combining one or more of the primitive views provided by SwiftUI, like the `Text` instance in the example above, plus other custom views that you define, into a hierarchy of views.

The `View` protocol provides a large set of modifiers, defined as protocol methods with default implementations, that you use to position and configure views in the layout of your app. Modifiers typically work by wrapping the view instance on which you call them in another view with the specified characteristics. For example, adding the `opacity(_)` modifier to a text view returns a new view with some amount of transparency:

```
Text("Hello, World!")
    .opacity(0.5) // Display partially
```

The effects of a modifier typically propagate







Hone your intuition about how
SwiftUI works.



Hone your intuition about how
SwiftUI works.



Hone your intuition about how
SwiftUI works.



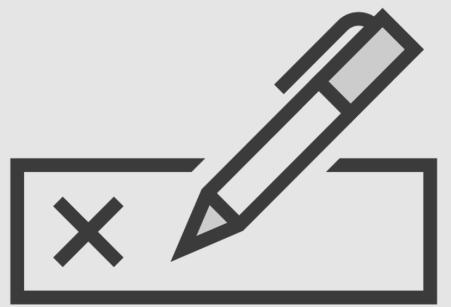
Hone your intuition about how Swift**UI** works.



“What makes a view... a **View?**”



Not merely a scholarly question that's only useful during your iOS Developer trivia nights...



- ✓ A **computed property**...
- ✓ **named body**...
- ✓ that returns **some View**



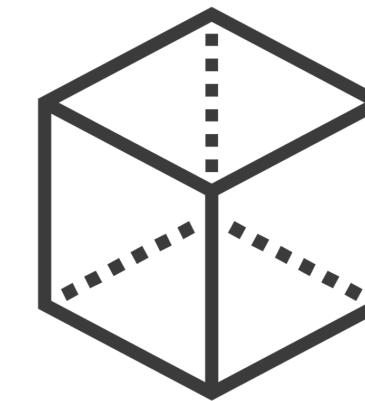
View Protocol



View Protocol

- ✓ A **computed property...**
- ✓ **named body...**
- ✓ **that returns some View**

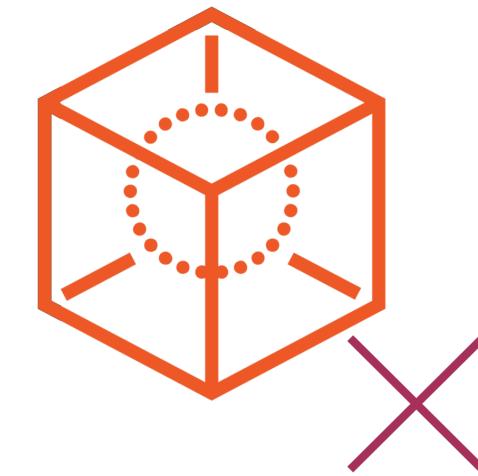
Swift Computed Properties



Value

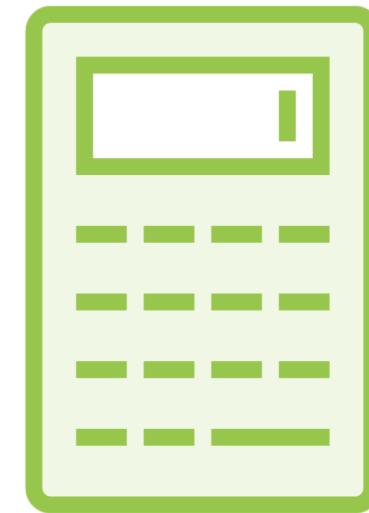
Swift Computed Properties

Do not store any values

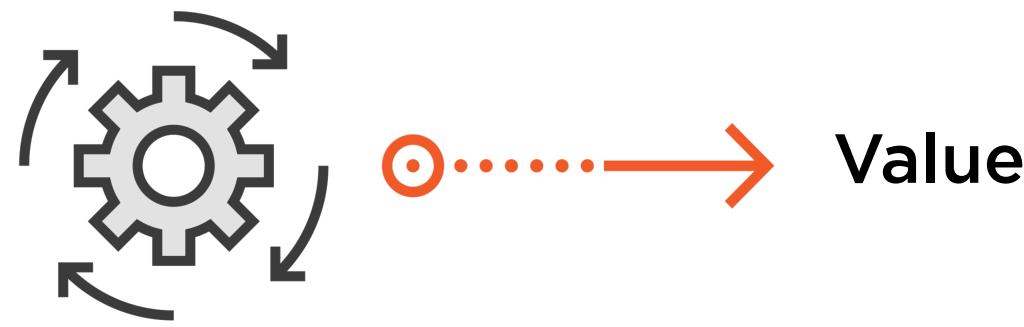


Swift Computed Properties

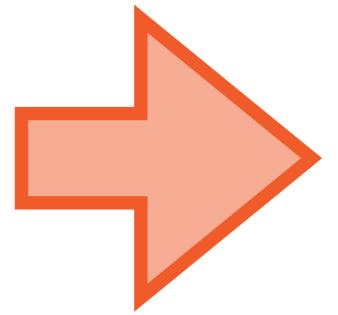
Only **calculate** values



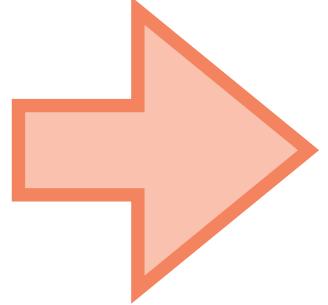
Swift Computed Properties



Combining and Laying out Views



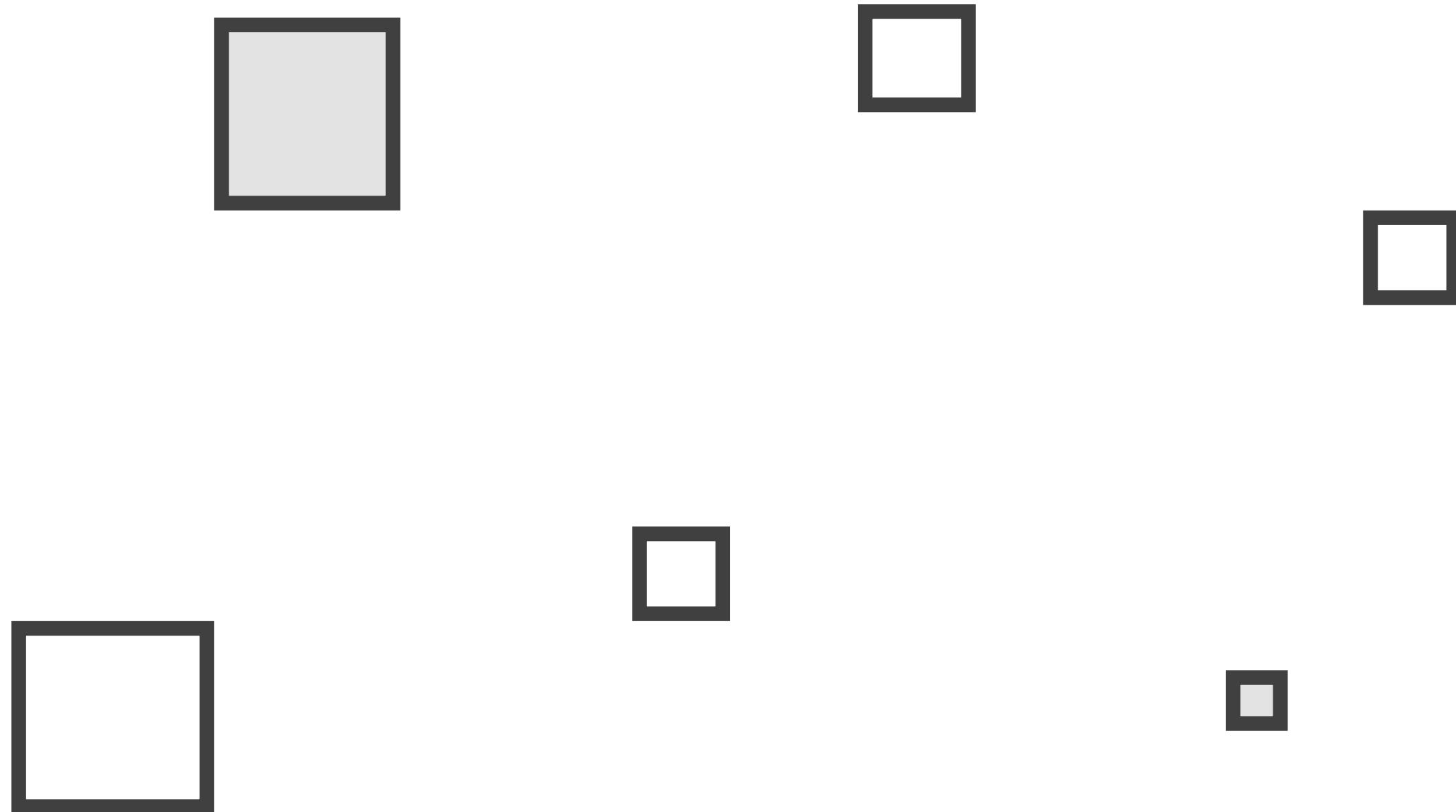
If **body** can only return one instance of
a specific kind of View...



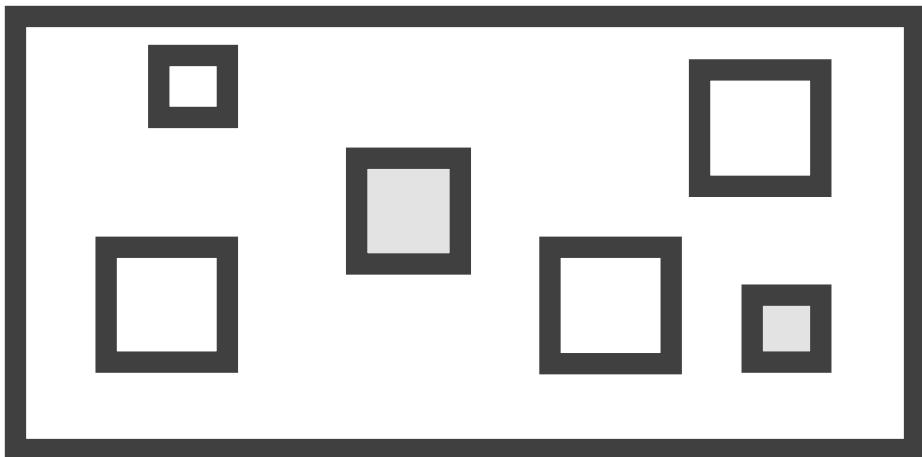
If **body** can only return **one instance** of
a specific kind of View...



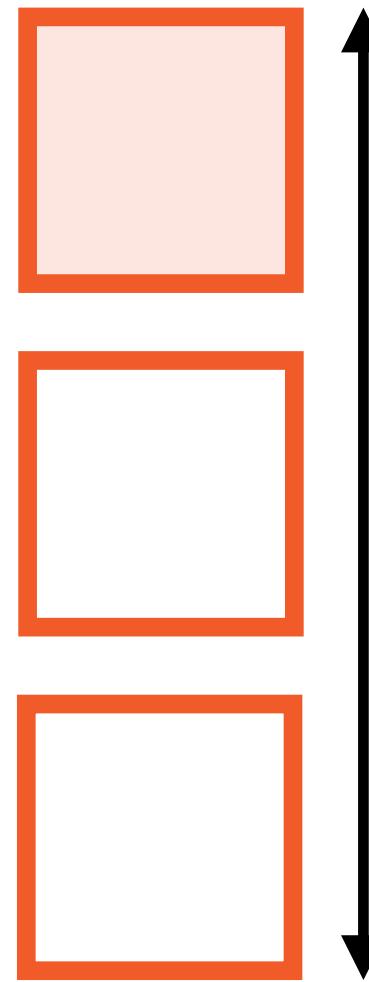
How do we put **multiple Views** together
onto the screen?



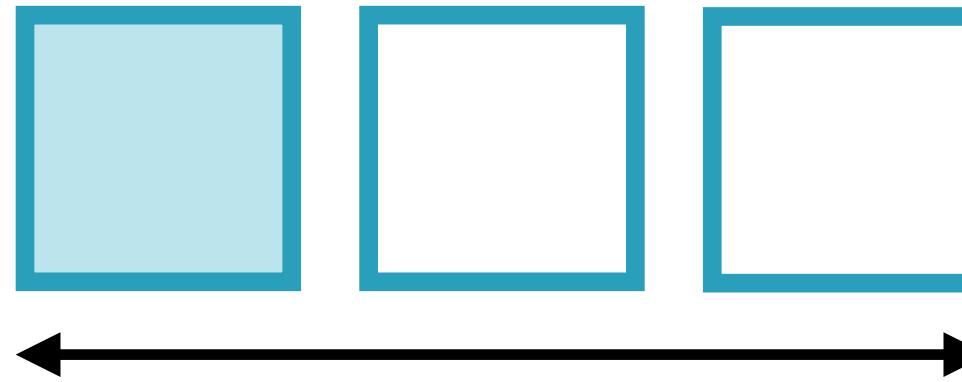
Layout container



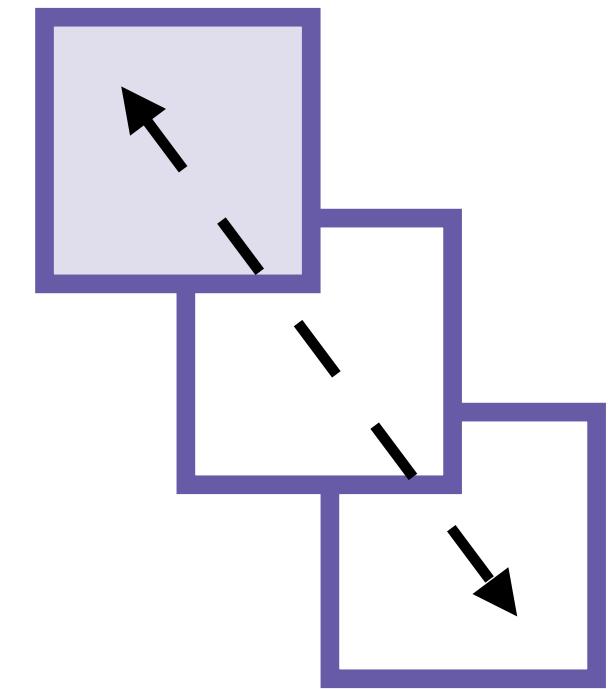
Layout Directions



Vertical

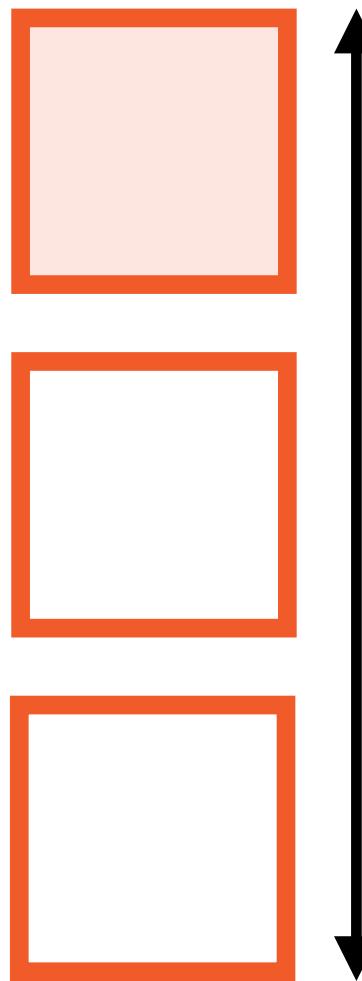


Horizontal

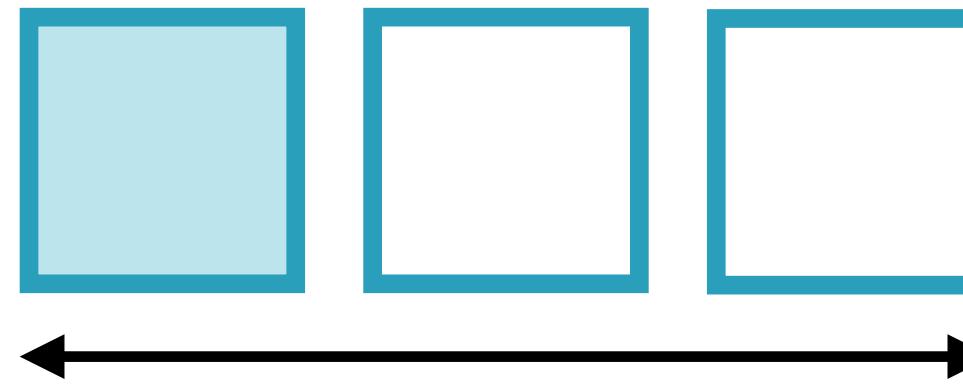


Depth

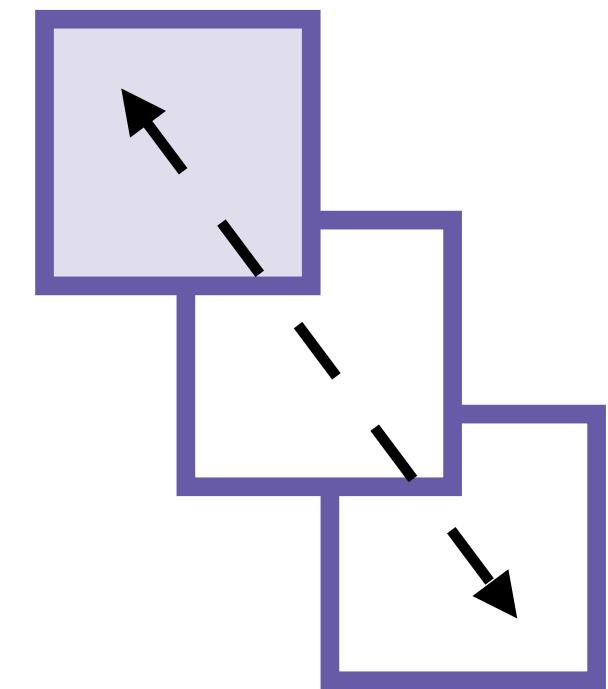
Layout Directions



Vertical



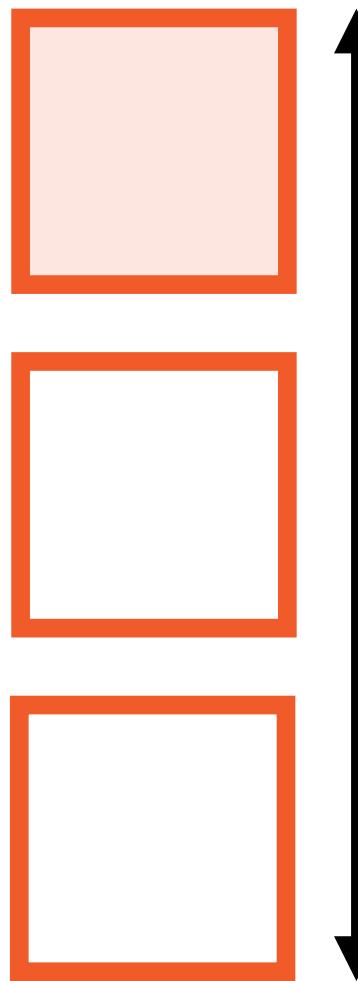
Horizontal



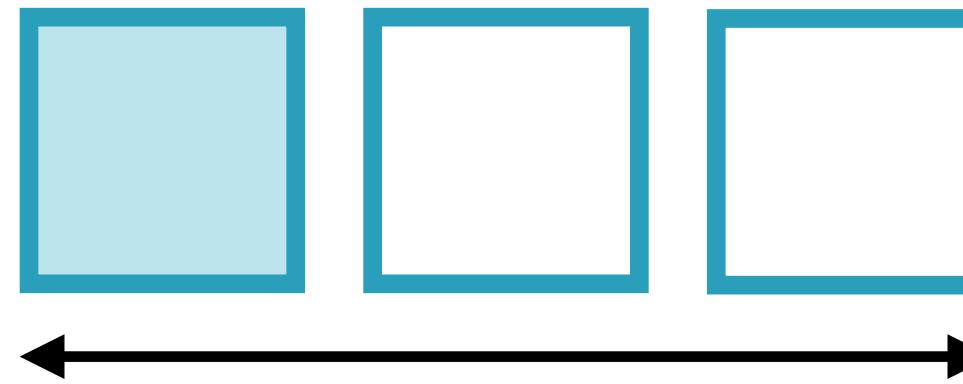
Depth

Stack

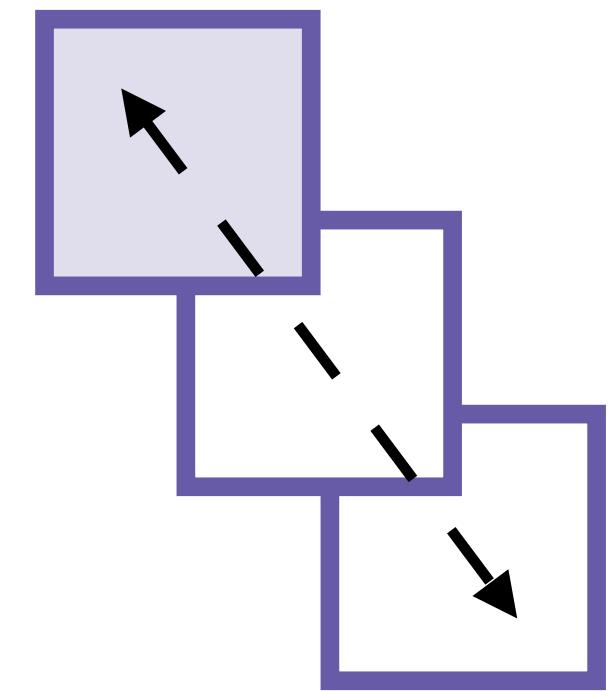
Layout Directions



Vertical Stack

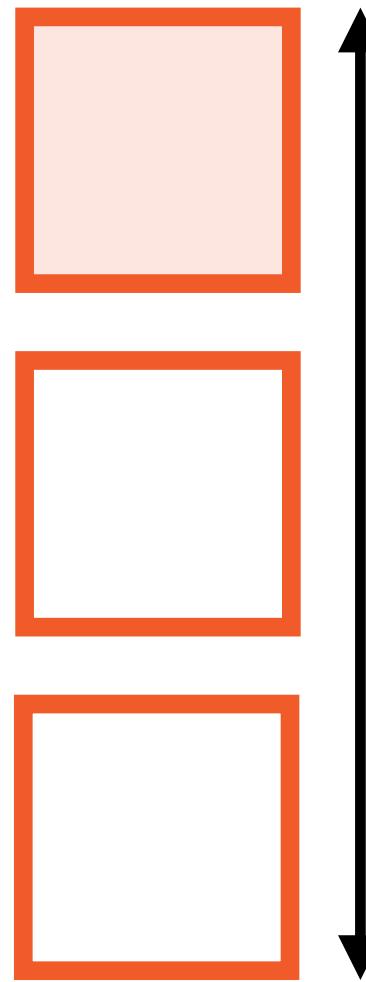


Horizontal Stack

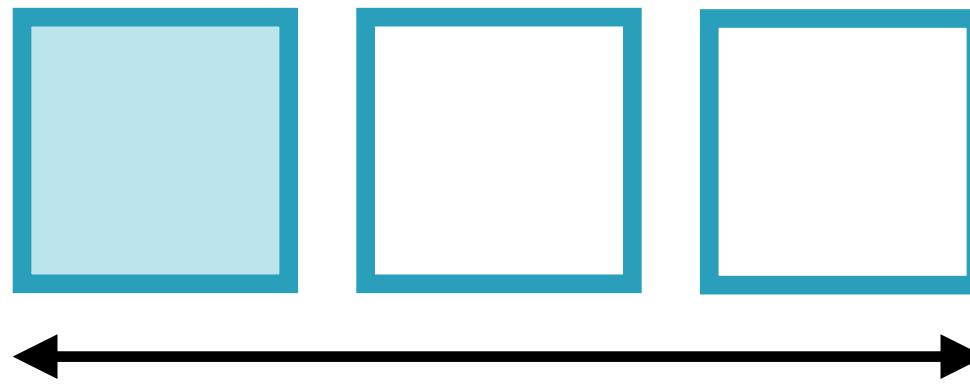


Depth Stack

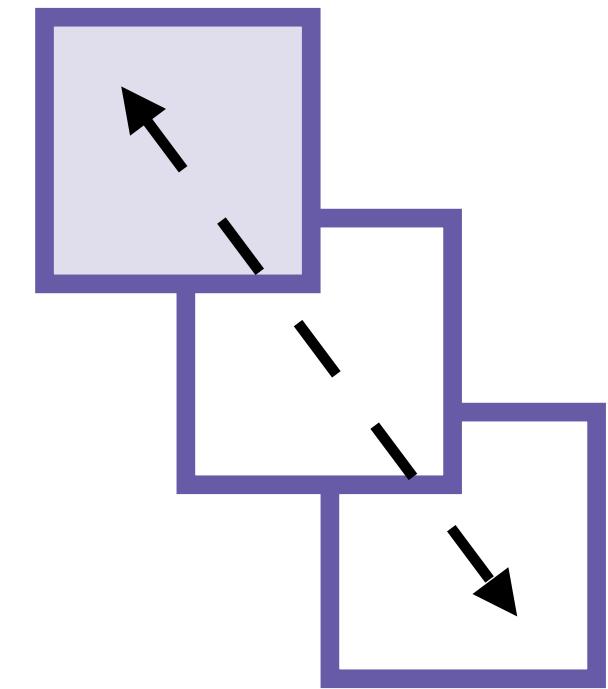
Layout Directions



V Stack

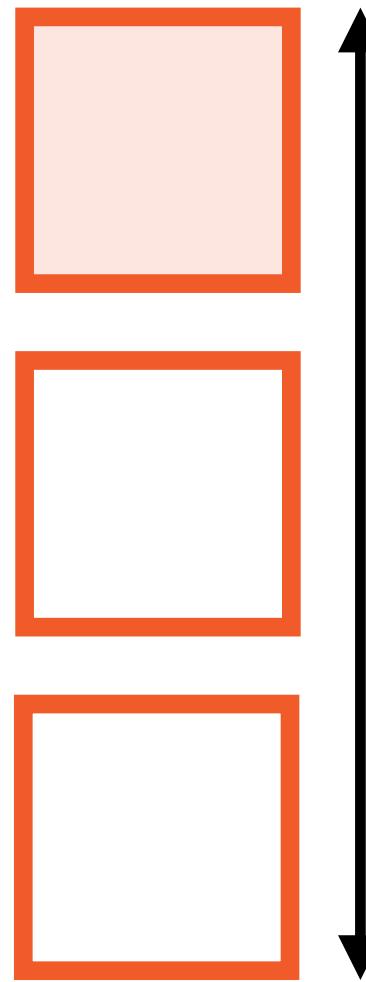


Horizontal Stack

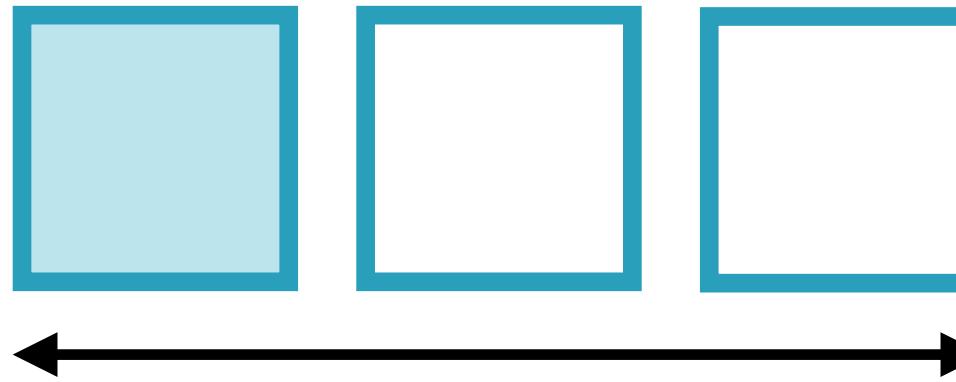


Depth Stack

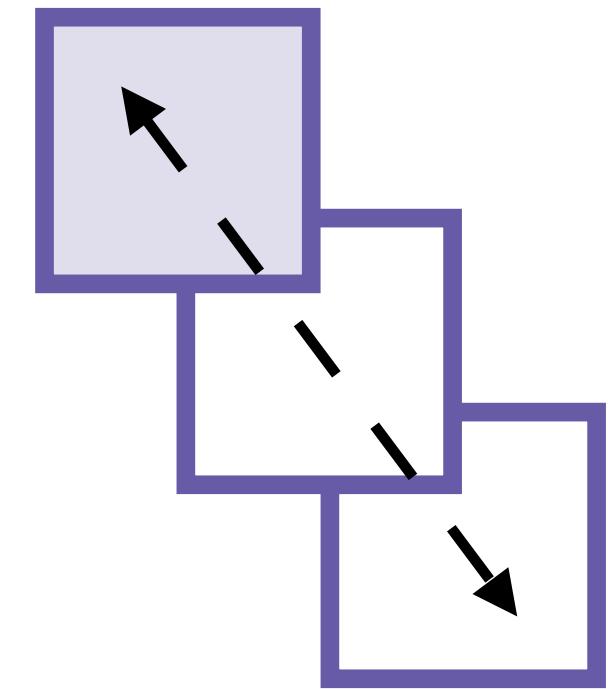
Layout Directions



VStack

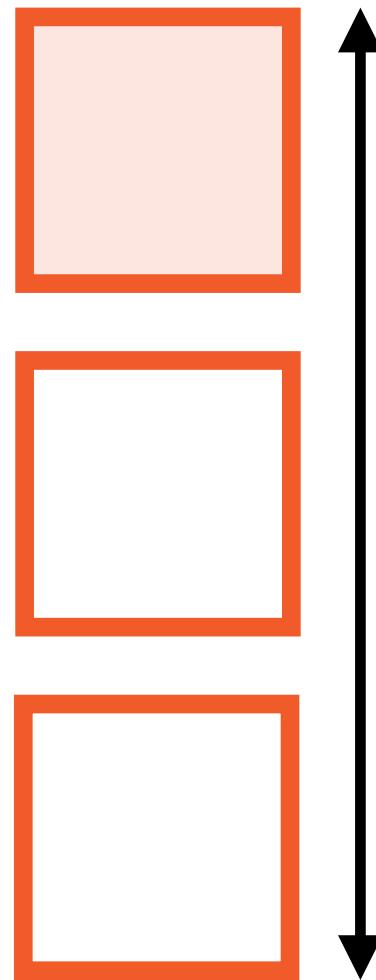


HStack

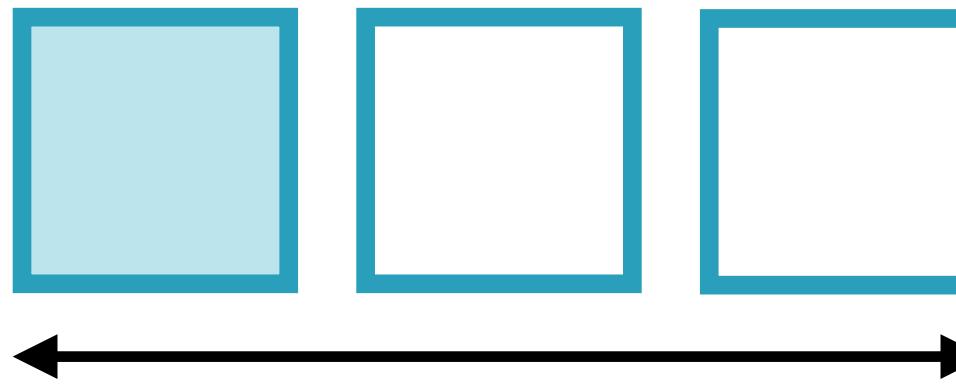


Depth Stack

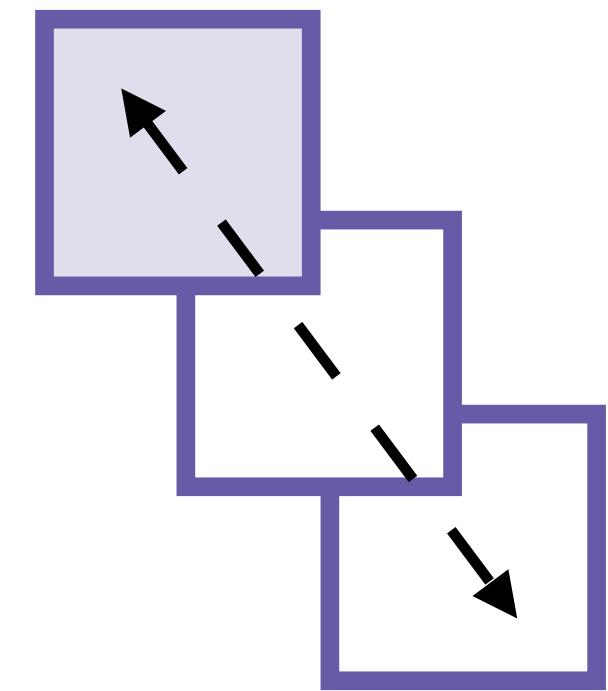
Layout Directions



VStack



HStack



~~**X**~~**D**Stack

Three dimensions [edit]

Further information: [Three-dimensional space](#)

A Cartesian coordinate system for a three-dimensional space consists of an ordered triplet of lines (the axes) that go through a common point (the *origin*), and are pair-wise perpendicular; an orientation for each axis; and a single unit of length for all three axes. As in the two-dimensional case, each axis becomes a number line. For any point P of space, one considers a hyperplane through P perpendicular to each coordinate axis, and interprets the point where that hyperplane cuts the axis as a number. The Cartesian coordinates of P are those three numbers, in the chosen order. The reverse construction determines the point P given its three coordinates.

Alternatively, each coordinate of a point P can be taken as the distance from P to the hyperplane defined by the other two axes, with the sign determined by the orientation of the corresponding axis.

Each pair of axes defines a *coordinate hyperplane*. These hyperplanes divide space into eight [trihedra](#), called *octants*.

The octants are: | $(+x, +y, +z)$ | $(-x, +y, +z)$ | $(+x, +y, -z)$ | $(-x, +y, -z)$ | $(+x, -y, +z)$ | $(-x, -y, +z)$ | $(+x, -y, -z)$ | $(-x, -y, -z)$ |

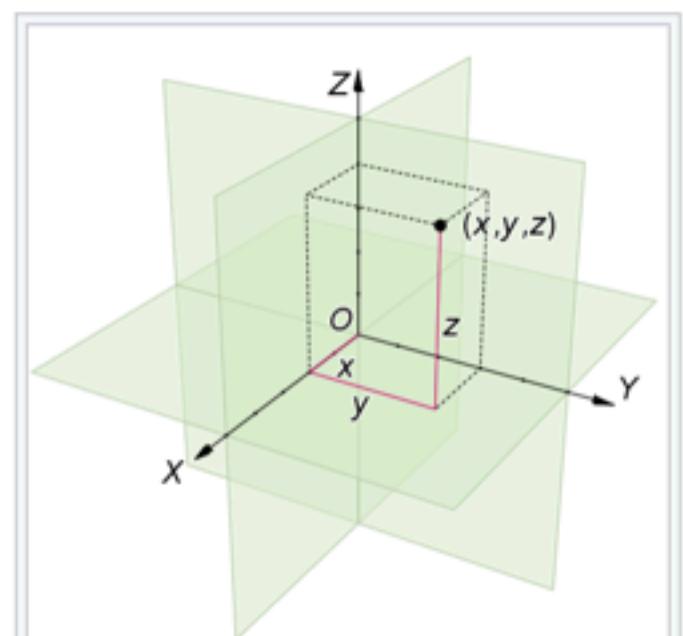
The coordinates are usually written as three numbers (or algebraic formulas) surrounded by parentheses and separated by commas, as in $(3, -2.5, 1)$ or $(t, u + v, \pi/2)$. Thus, the origin has coordinates $(0, 0, 0)$, and the unit points on the three axes are $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.

There are no standard names for the coordinates in the three axes (however, the terms *abscissa*, *ordinate* and *applicate* are sometimes used). The coordinates are often denoted by the letters X , Y , and Z , or x , y , and z . The axes may then be referred to as the X -axis, Y -axis, and Z -axis, respectively. Then the coordinate hyperplanes can be referred to as the XY -plane, YZ -plane, and XZ -plane.

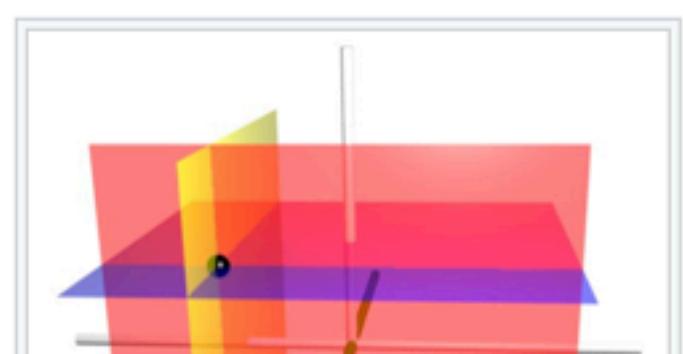
In mathematics, physics, and engineering contexts, the first two axes are often defined or depicted as horizontal, with the third axis pointing up. In that case the third coordinate may be called *height* or *altitude*. The orientation is usually chosen so that the 90 degree angle from the first axis to the second axis looks counter-clockwise when seen from the point $(0, 0, 1)$; a convention that is commonly called the [right hand rule](#).

Higher dimensions [edit]

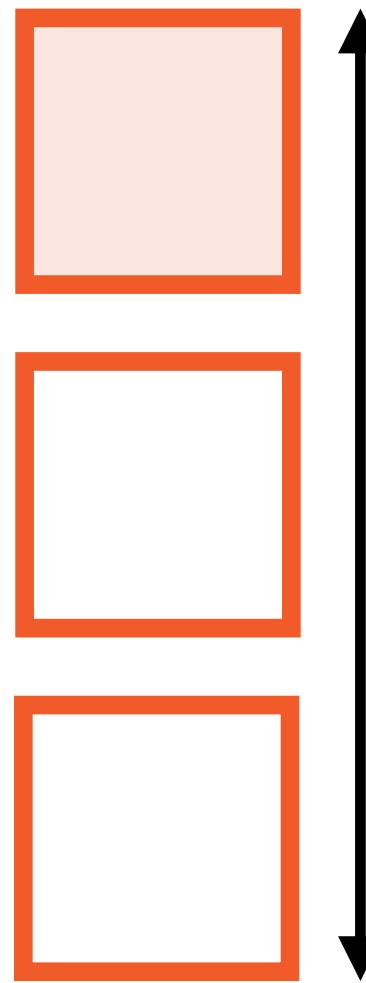
Since Cartesian coordinates are unique and non-ambiguous, the points of a Cartesian plane can be identified with pairs of [real numbers](#); that is with the [Cartesian product](#) $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$, where \mathbb{R} is the set of all real numbers. In the same way, the points in any [Euclidean space](#) of dimension n be identified with the [tuples](#) (lists) of n real numbers, that is, with the Cartesian product \mathbb{R}^n .



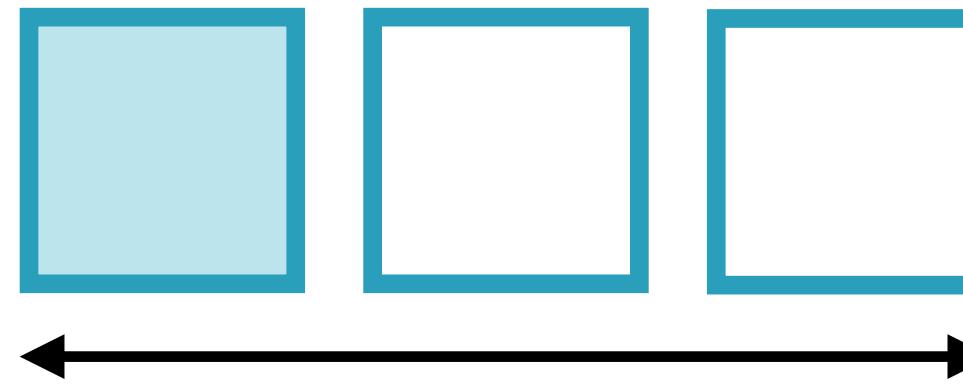
A three dimensional Cartesian coordinate system, with origin O and axis lines X , Y and Z , oriented as shown by the arrows. The tick marks on the axes are one length unit apart. The black dot shows the point with coordinates $x = 2$, $y = 3$, and $z = 4$, or $(2, 3, 4)$.



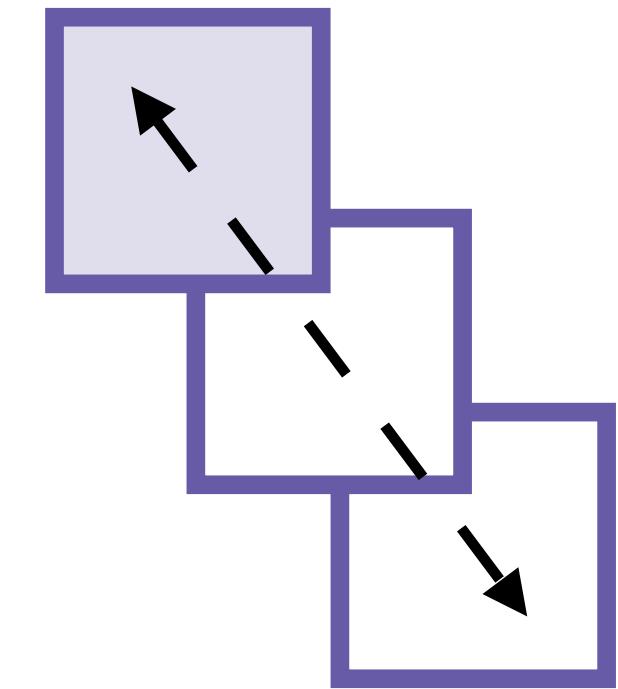
Layout Directions



V Stack

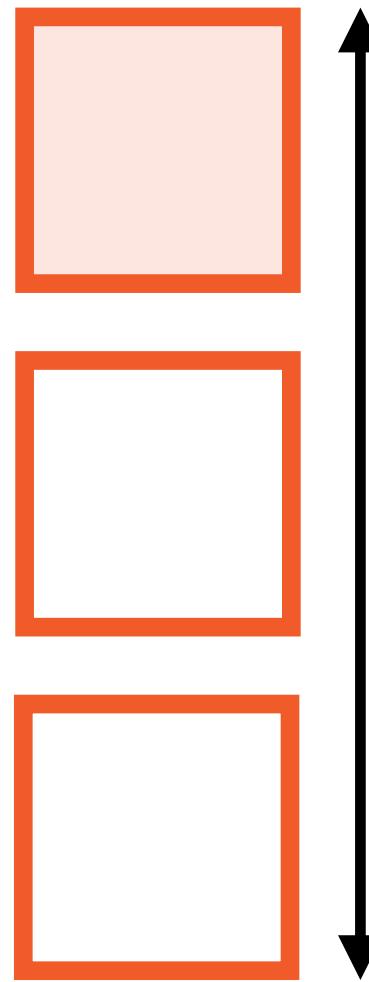


H Stack

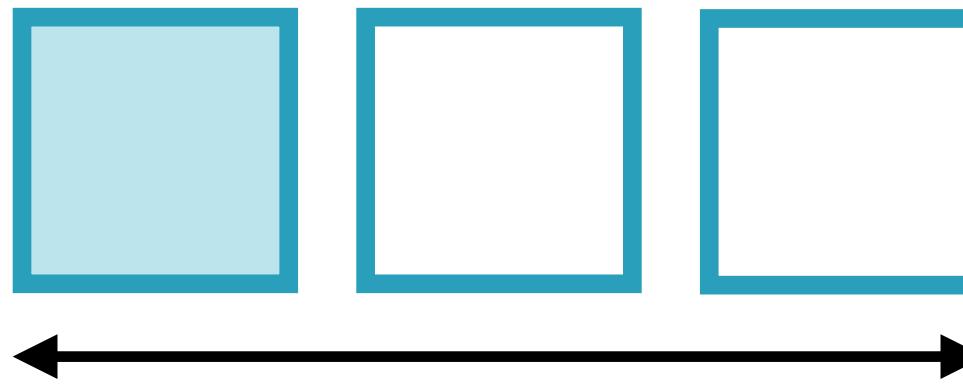


Z Stack

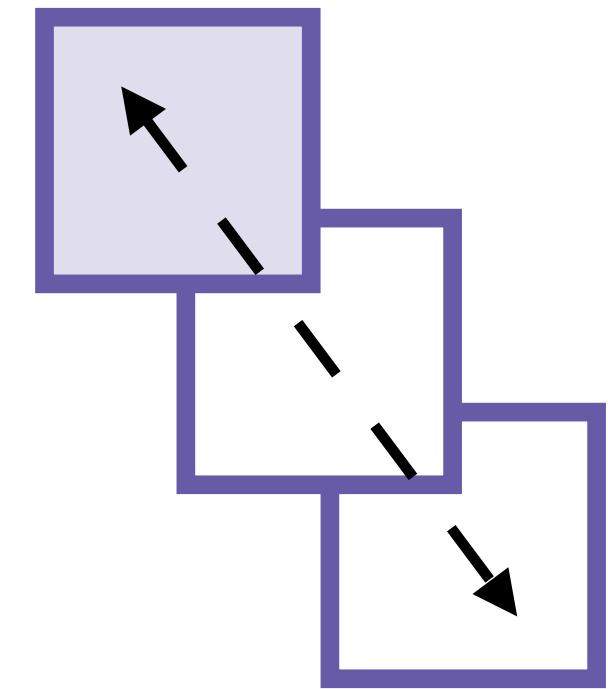
Layout Directions



VStack

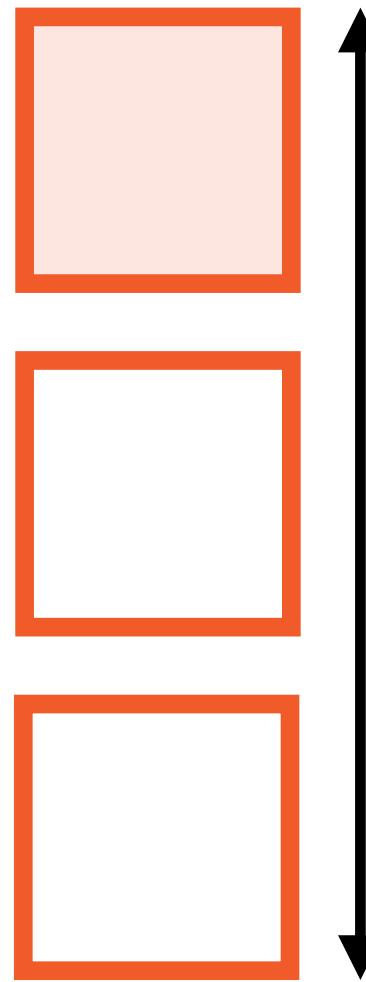


HStack

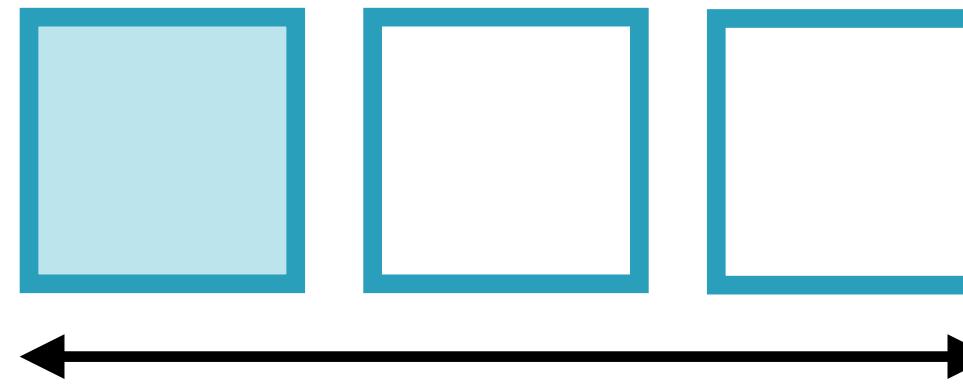


ZStack

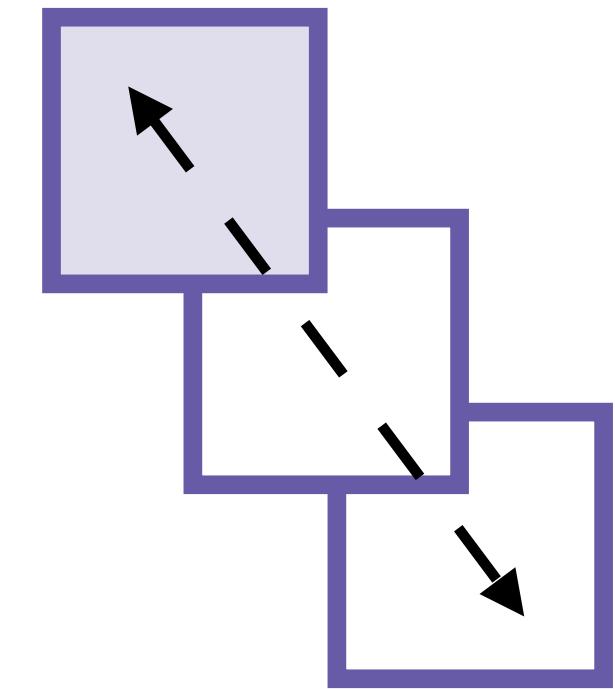
Layout Directions



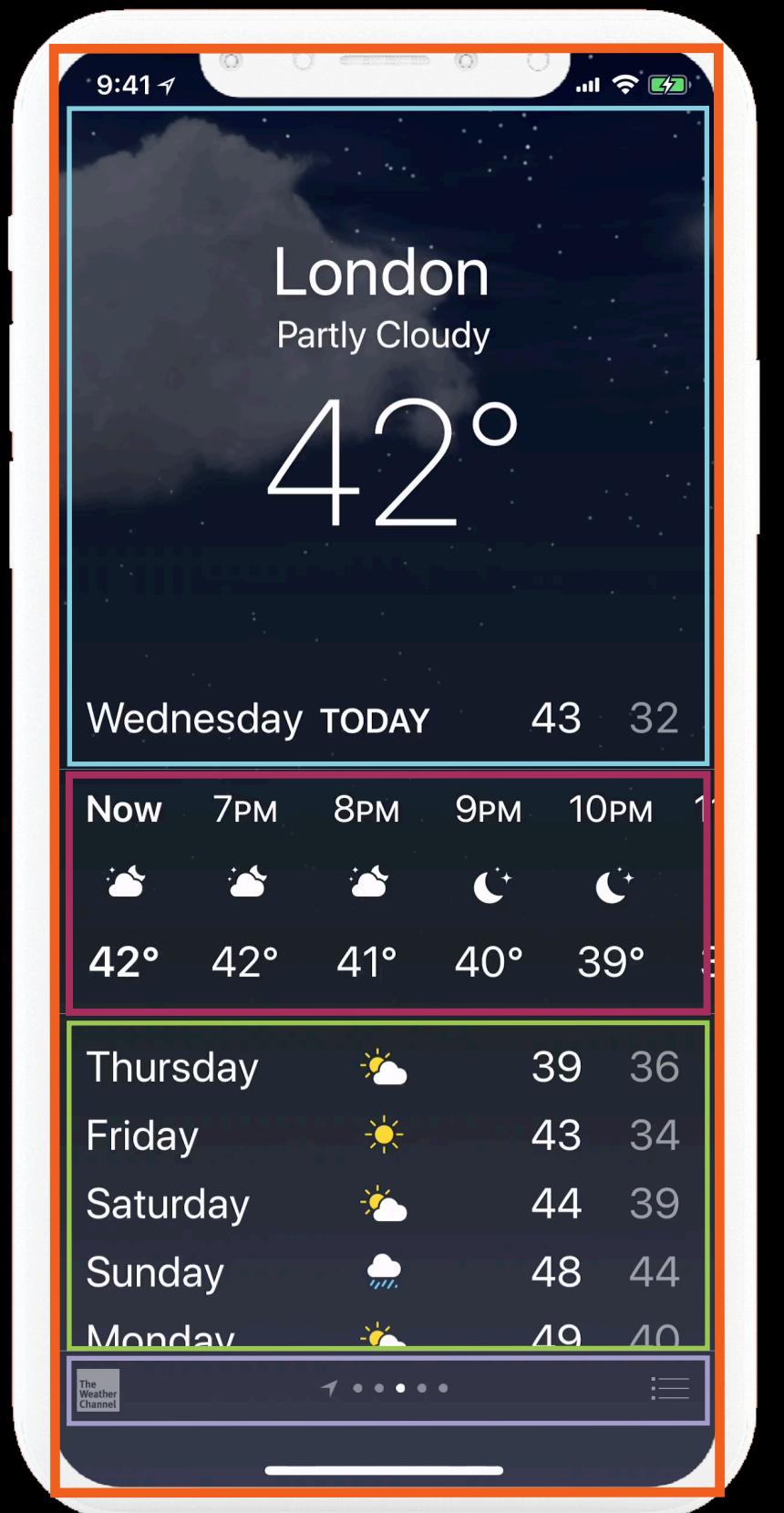
V Stack

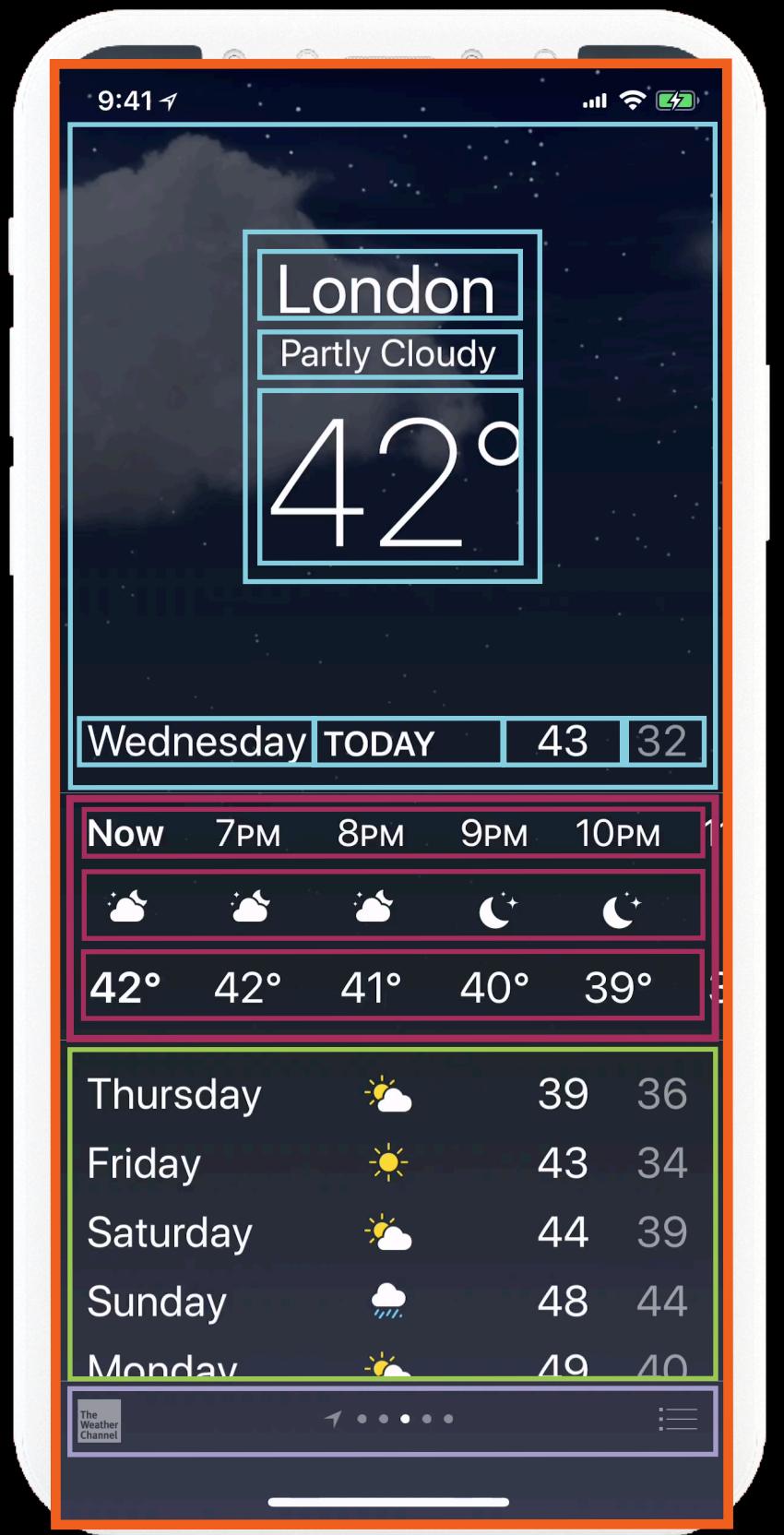


H Stack



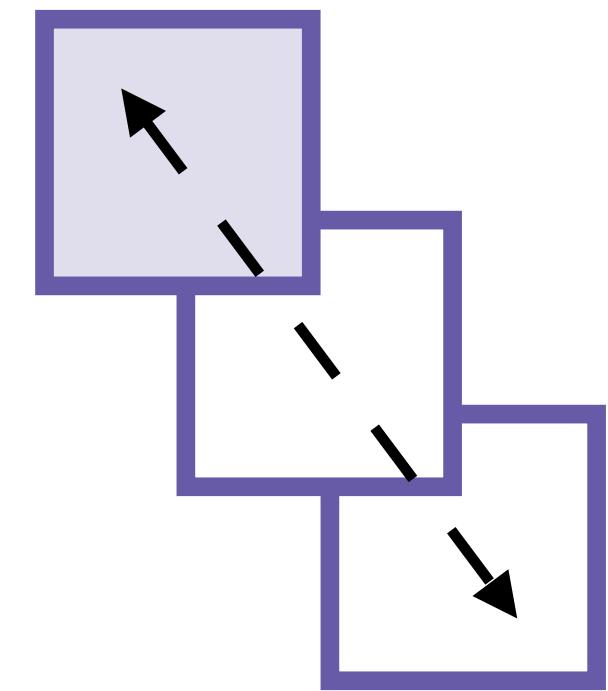
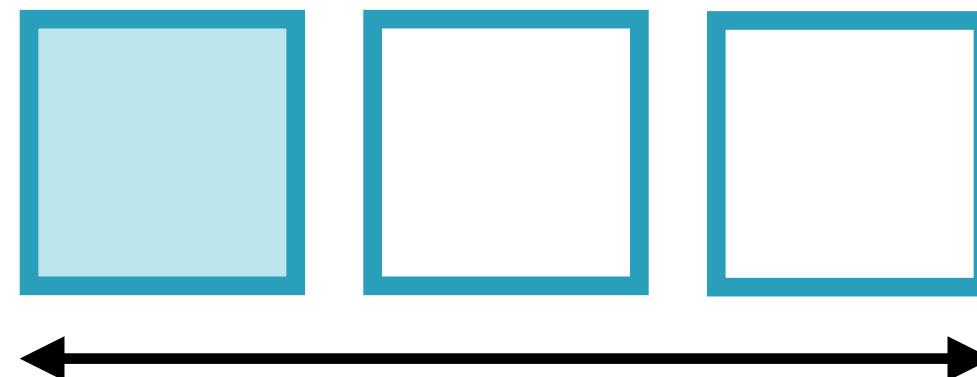
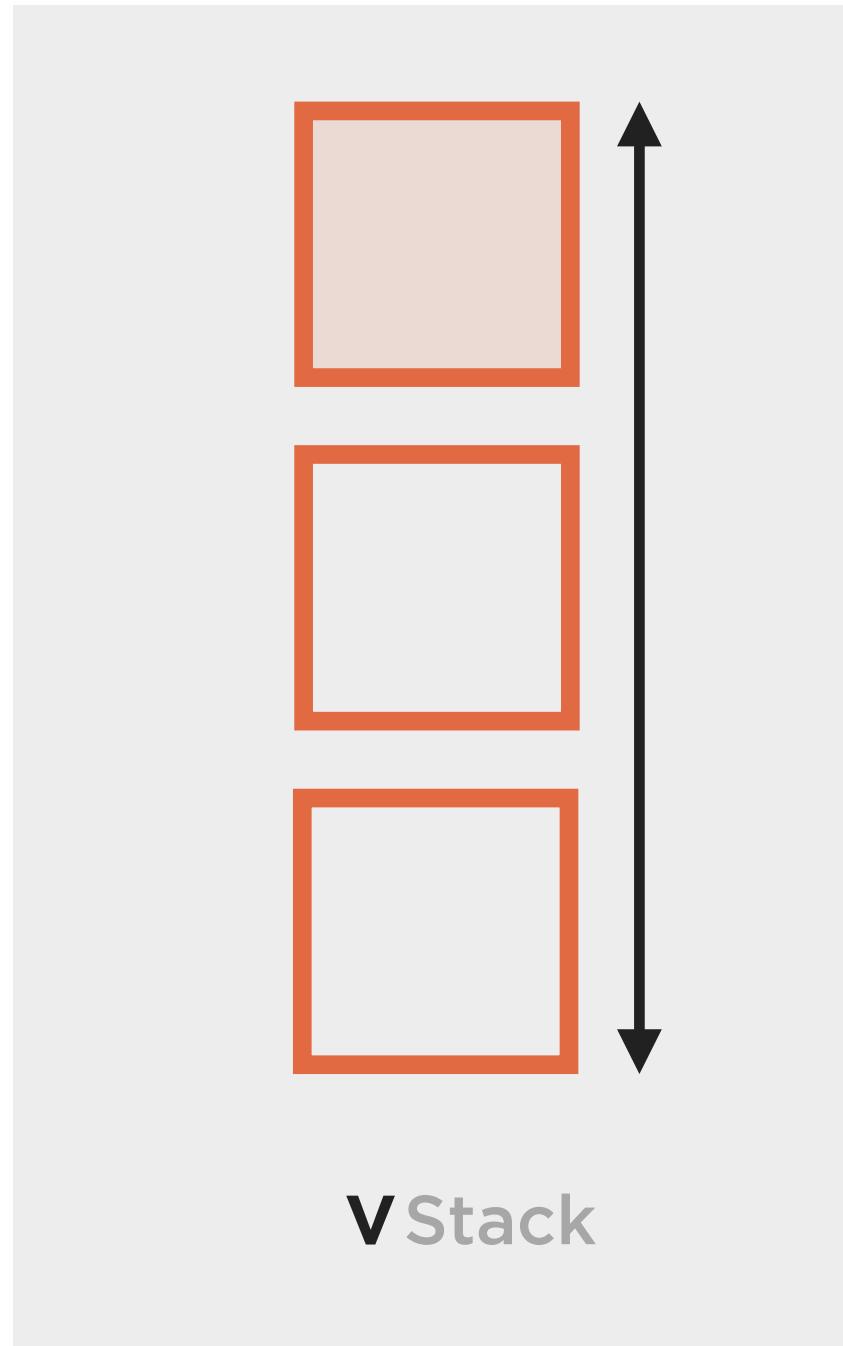
Z Stack



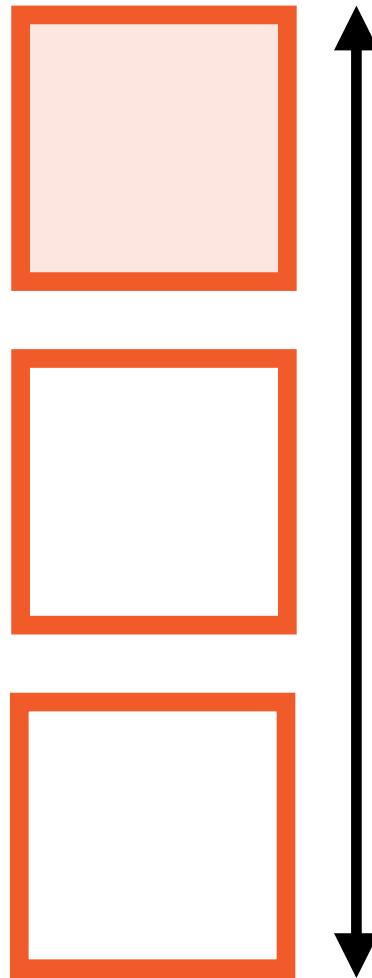


Understanding the SwiftUI Layout System

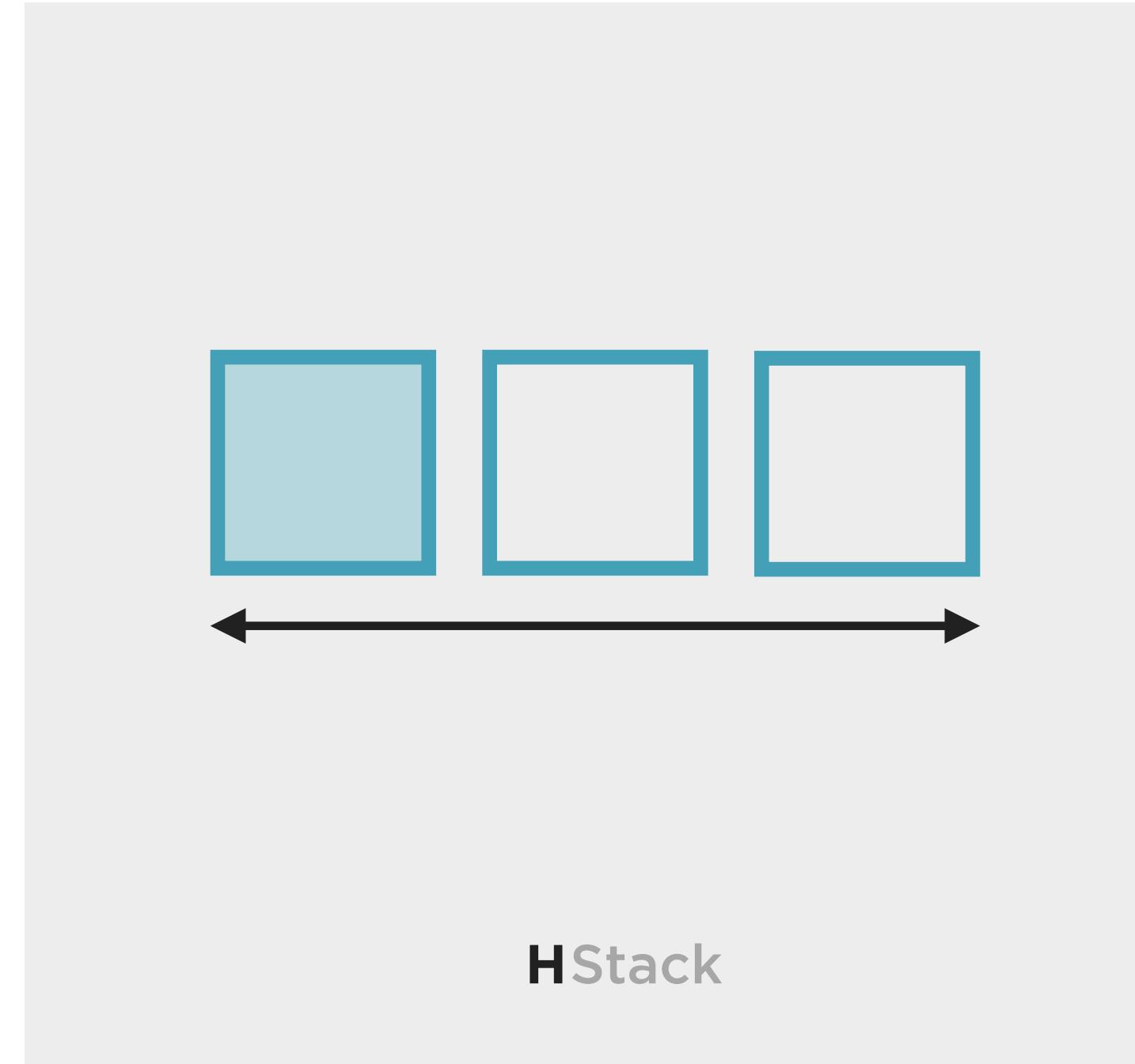
Layout Directions



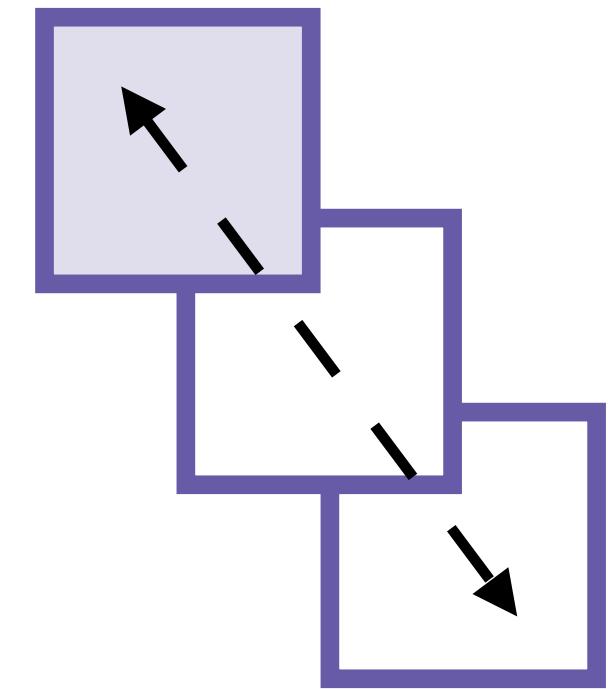
Layout Directions



V Stack

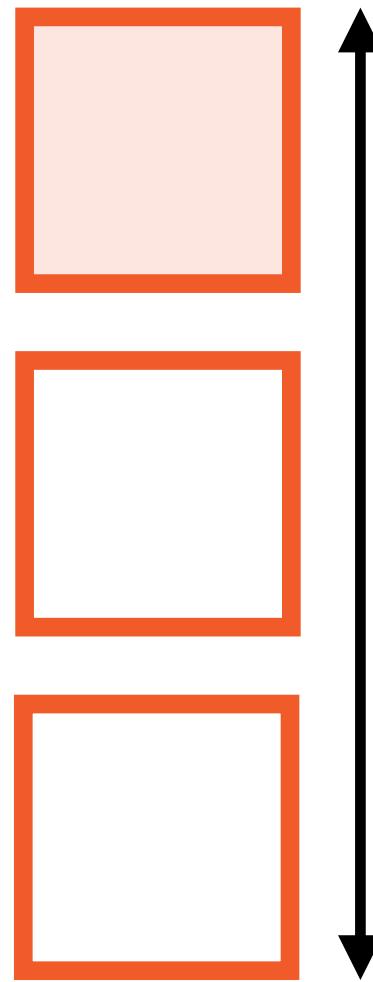


H Stack

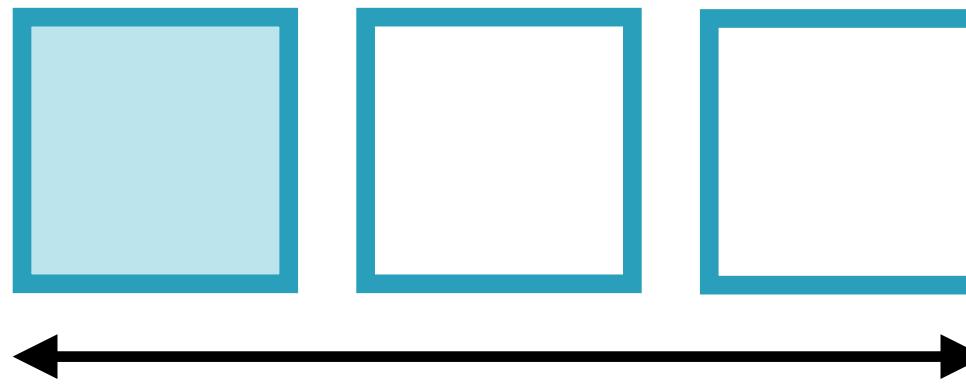


Z Stack

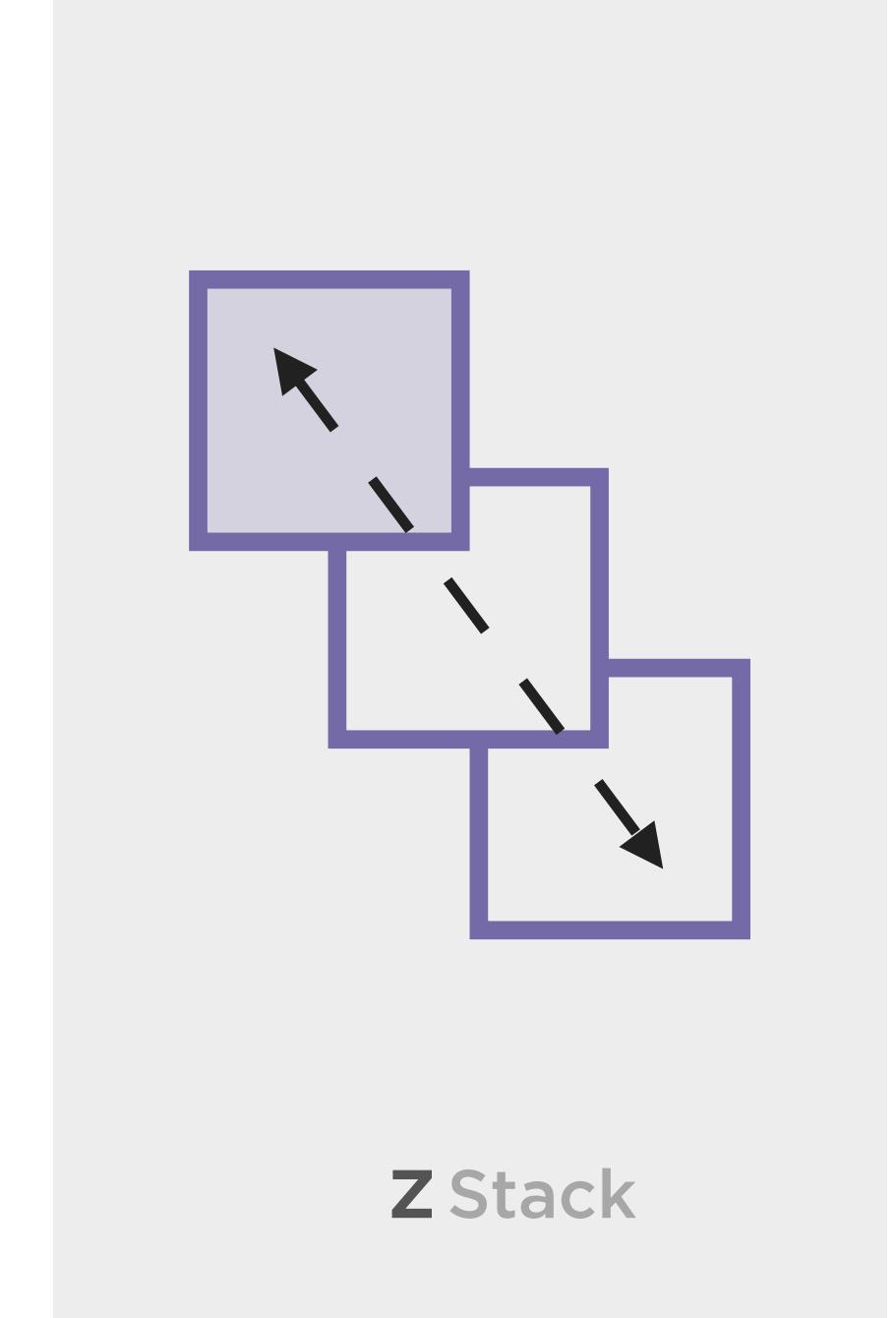
Layout Directions



VStack



HStack



ZStack

First iOS App > iPhone 11 First iOS App | Build for Previews First iOS App: Succeeded | Today at 4:23 PM + ↗

ContentView.swift

No Selection

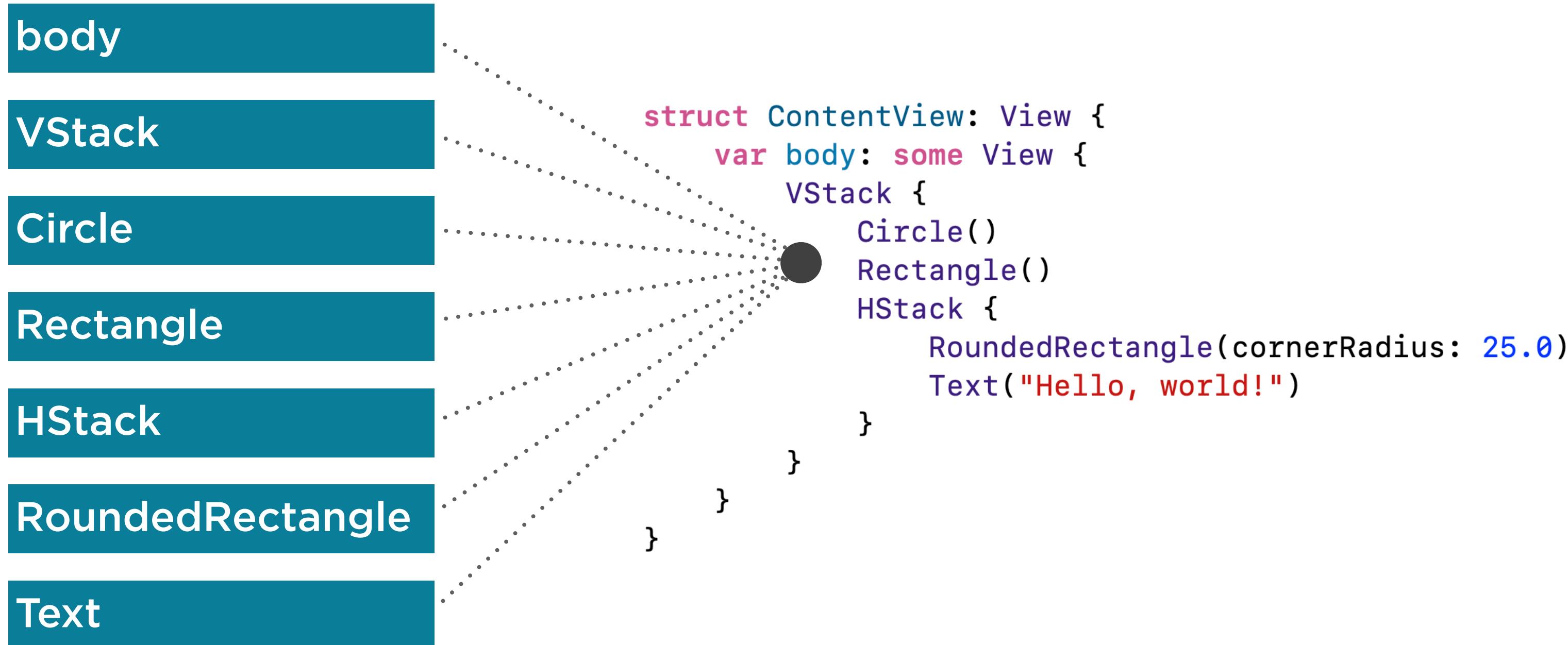
```
1 import SwiftUI
2
3
4 struct ContentView: View {
5     var body: some View {
6         VStack {
7             Circle()
8             Rectangle()
9             HStack {
10                 RoundedRectangle(cornerRadius: 25.0)
11                 Text("Hello, world!")
12             }
13         }
14     }
15 }
16
17 struct ContentView_Previews: PreviewProvider {
18     static var previews: some View {
19         ContentView()
20     }
21 }
```

Preview

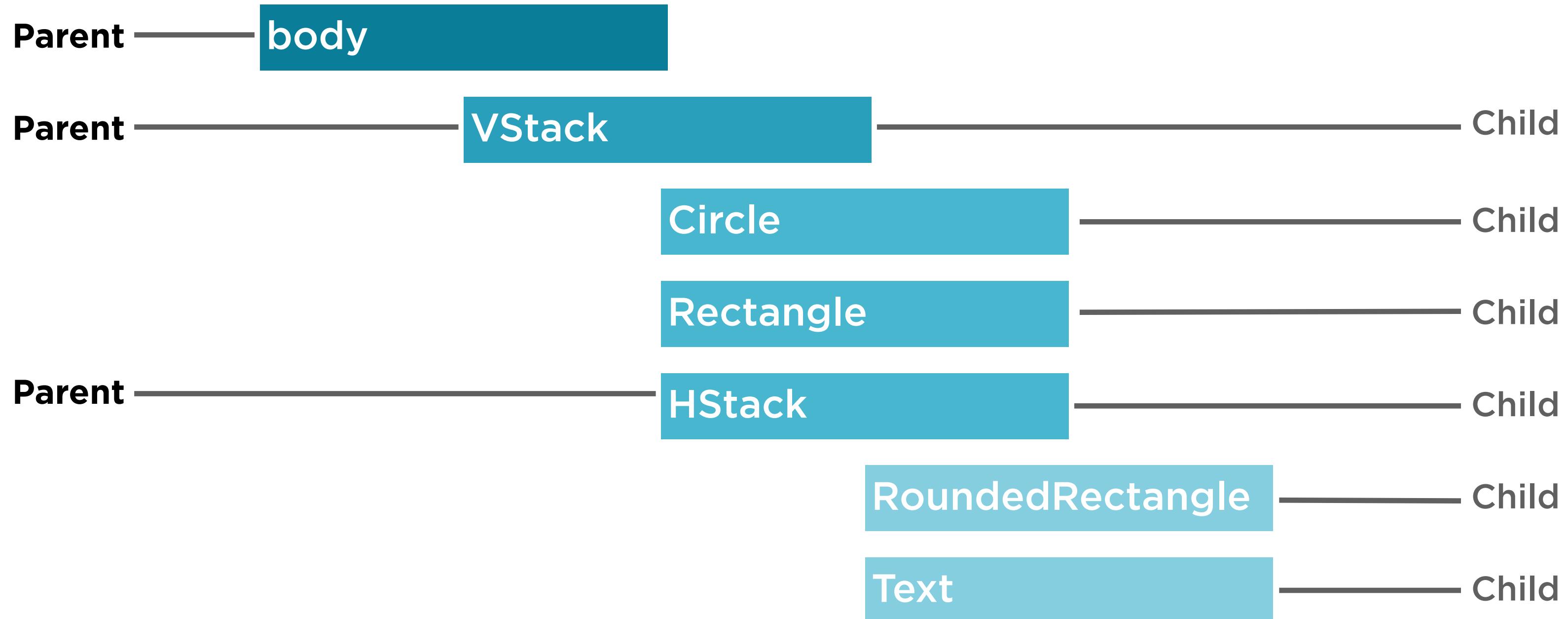
Hello, world!

```
struct ContentView: View {
    var body: some View {
        VStack {
            Circle()
            Rectangle()
            HStack {
                RoundedRectangle(cornerRadius: 25.0)
                Text("Hello, world!")
            }
        }
    }
}
```

View Hierarchy



View Hierarchy

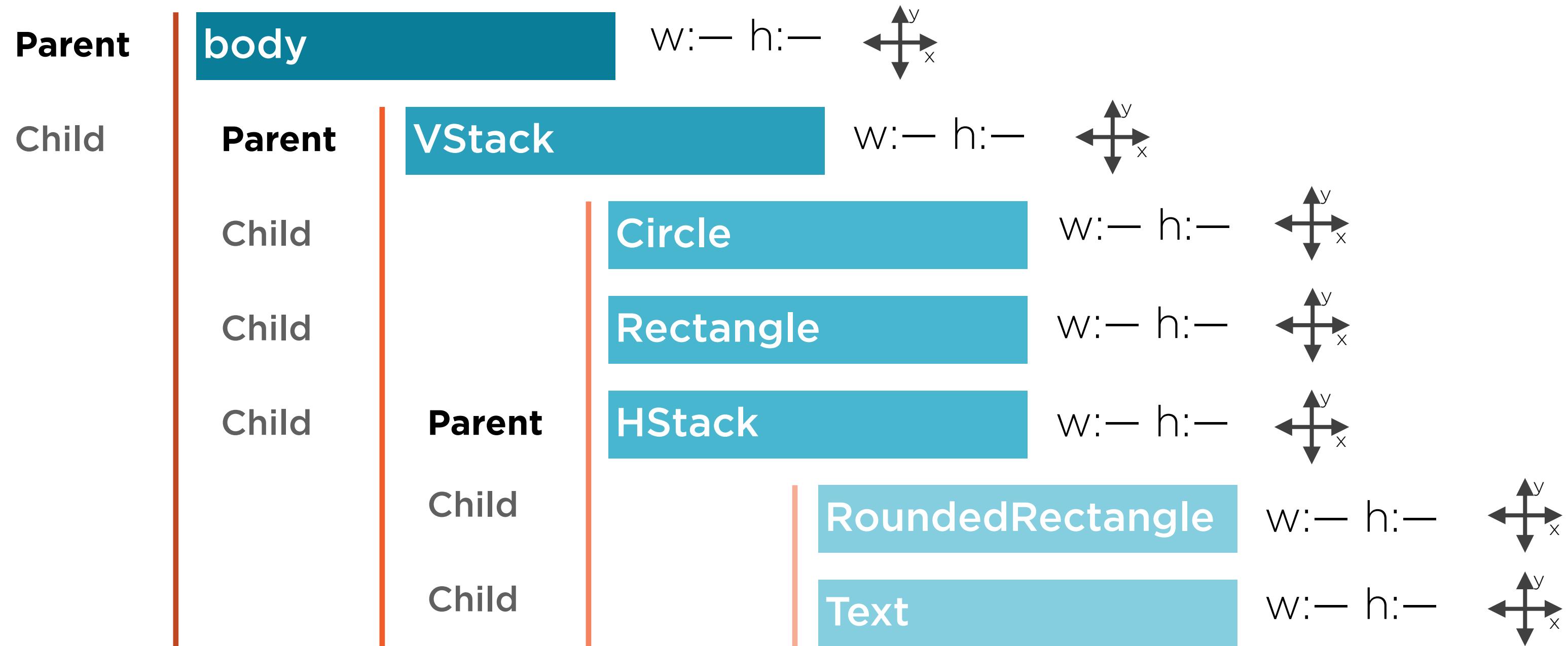


“Child View”



“Subview”

View Hierarchy

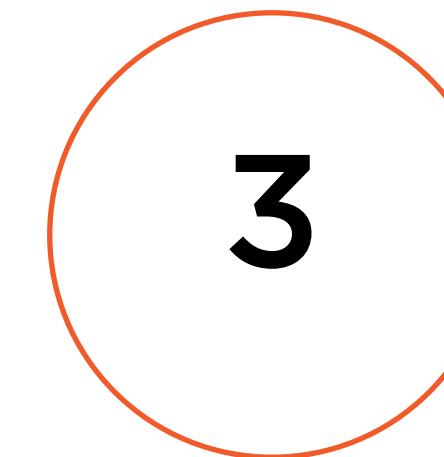
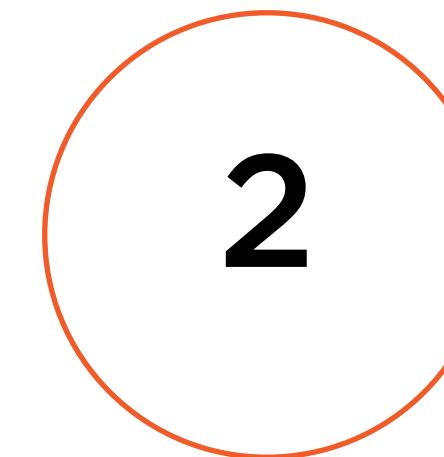
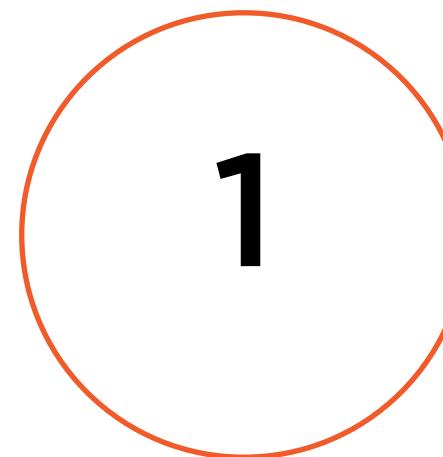


“Child View”

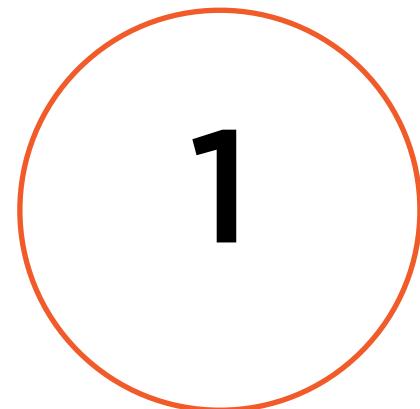


“Subview”

SwiftUI Layout Process



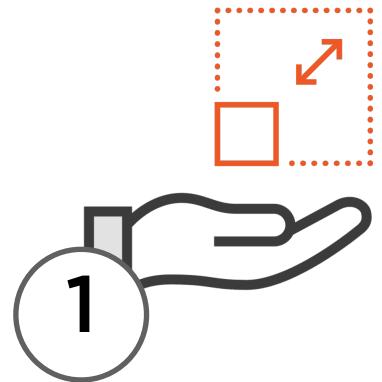
SwiftUI Layout Process



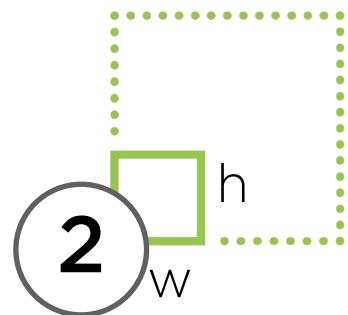
How much space a particular View takes up

Where a particular View gets positioned onto an iOS screen

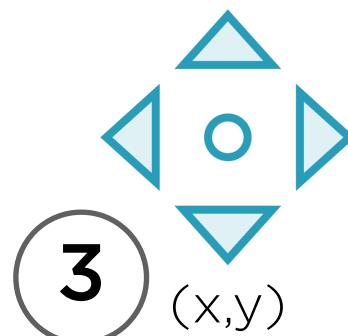
SwiftUI Layout Process



A **parent View** **offers** a child View some **space**

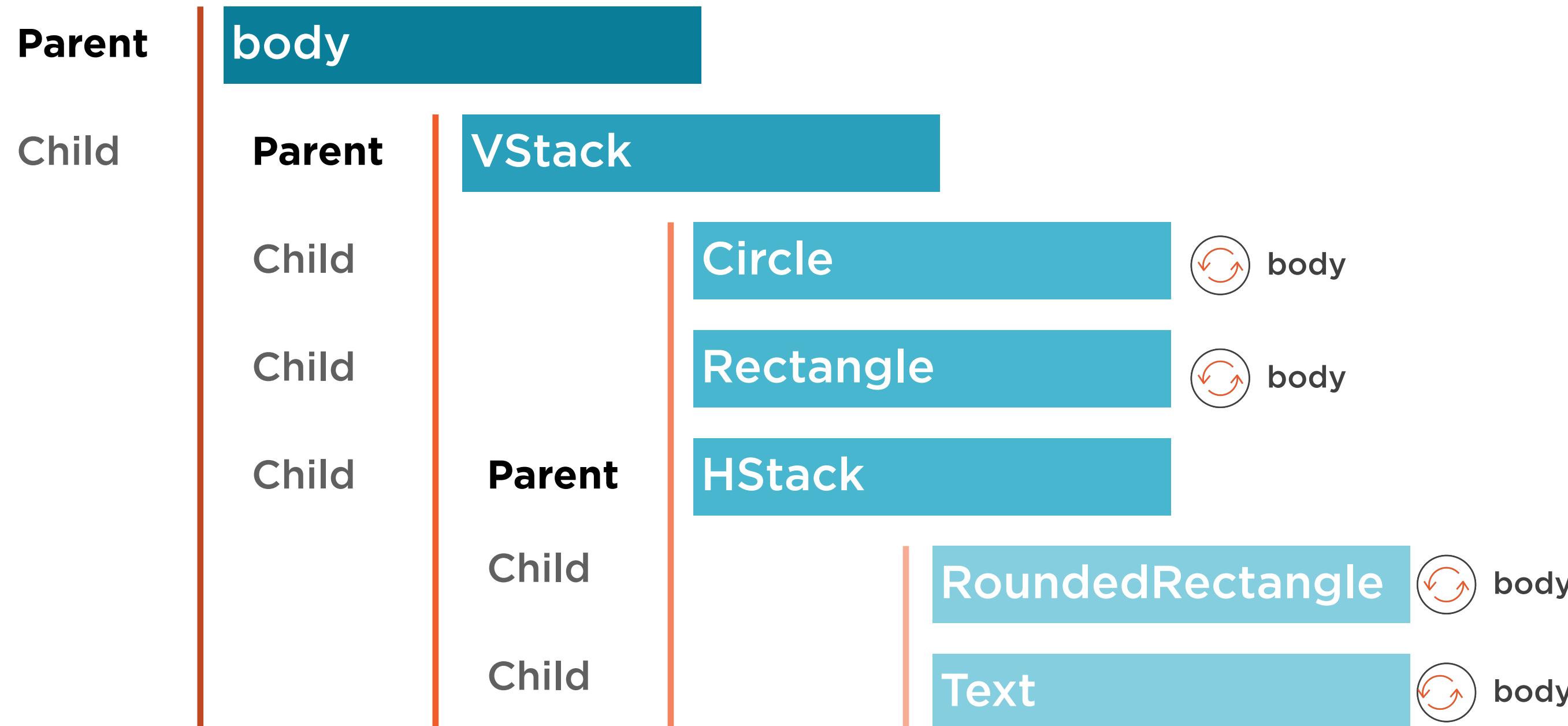


The **child** considers the offer and **chooses a size for itself** that is up to and including the total amount offered to it, then reports back to the layout system

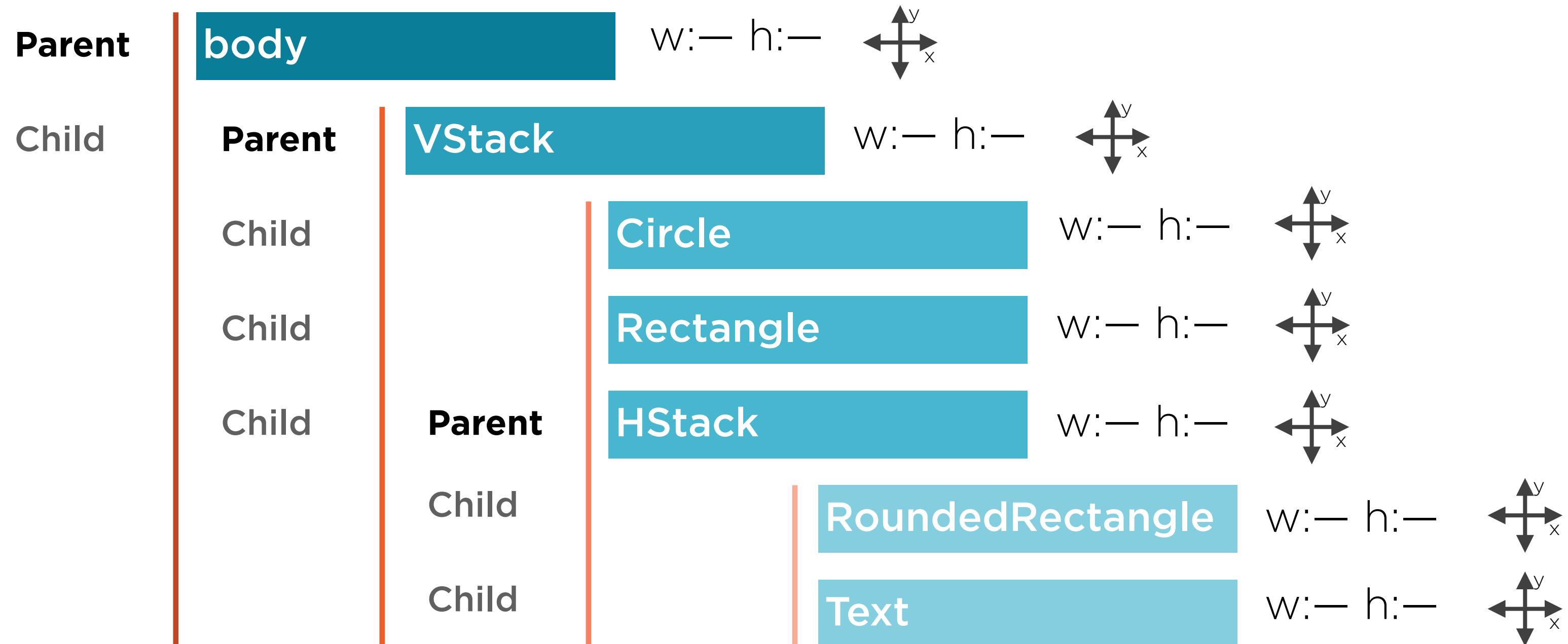


The **parent View** uses the height and width that its **child** chose, and **positions** that child View directly **in the center** of its own available coordinate space

View Hierarchy



View Hierarchy



Aligning and Positioning Views

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:41 AM

ContentView.swift NightWatchApp.swift

NightWatch

NightWatch

NightWatchApp.swift ContentView.swift Assets.xcassets Info.plist Preview Content Products

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         VStack(alignment: .leading) {
6             Text("Nightly Tasks")
7             Text("Weekly Tasks")
8             Text("Monthly Tasks")
9         }
10    }
11 }
12
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         ContentView()
17     }
18 }
19
```

Vertical Stack

Spacing: - Inherited +

Alignment:   

Modifiers

Padding

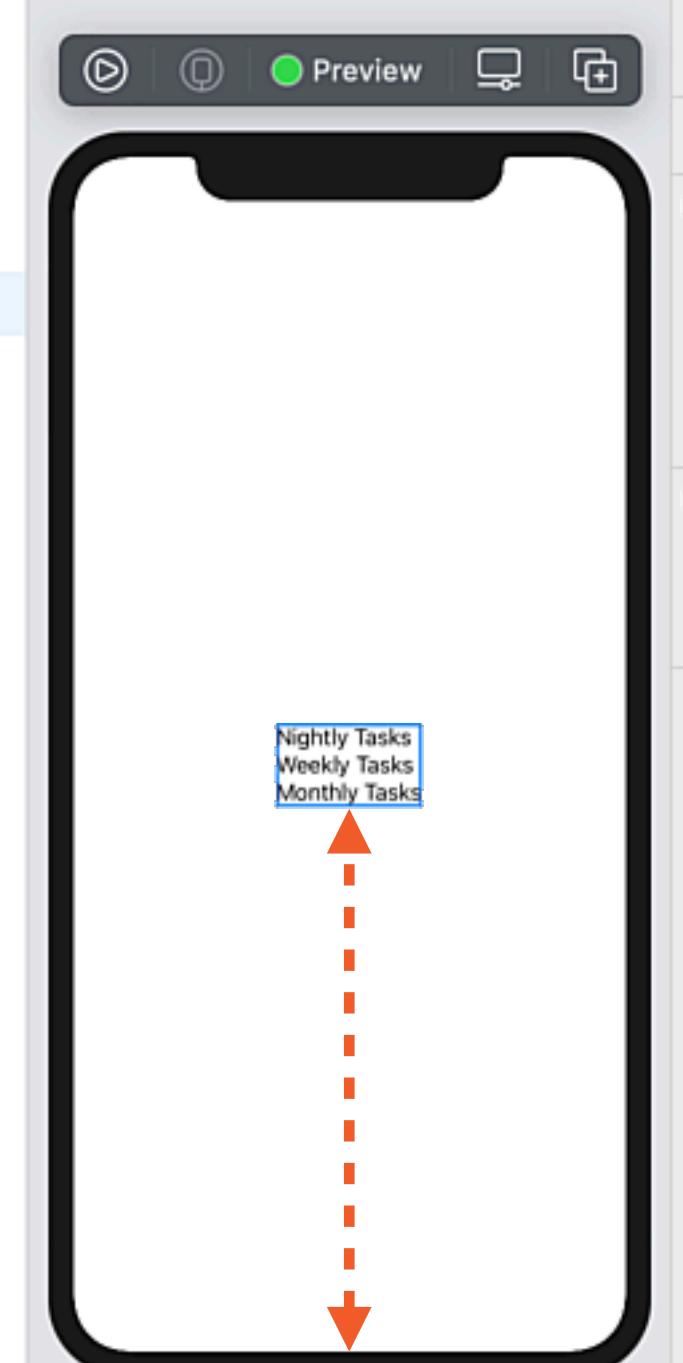
Padding: - Inherited +

Frame

Size: - Inher + Width: - Inher + Height: - Inher +

Add Modifier

Preview



Nightly Tasks
Weekly Tasks
Monthly Tasks

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:41 AM

ContentView.swift NightWatchApp.swift

NightWatch

NightWatch

NightWatchApp.swift ContentView.swift Assets.xcassets Info.plist Preview Content Products

```
1
2 import SwiftUI
3
4 struct ContentView: View {
5     var body: some View {
6         VStack(alignment: .leading) {
7             Text("Nightly Tasks")
8             Text("Weekly Tasks")
9             Text("Monthly Tasks")
10        }
11    }
12 }
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         ContentView()
17     }
18 }
19
```

Vertical Stack

Spacing: - Inherited +

Alignment:   

Modifiers

Padding

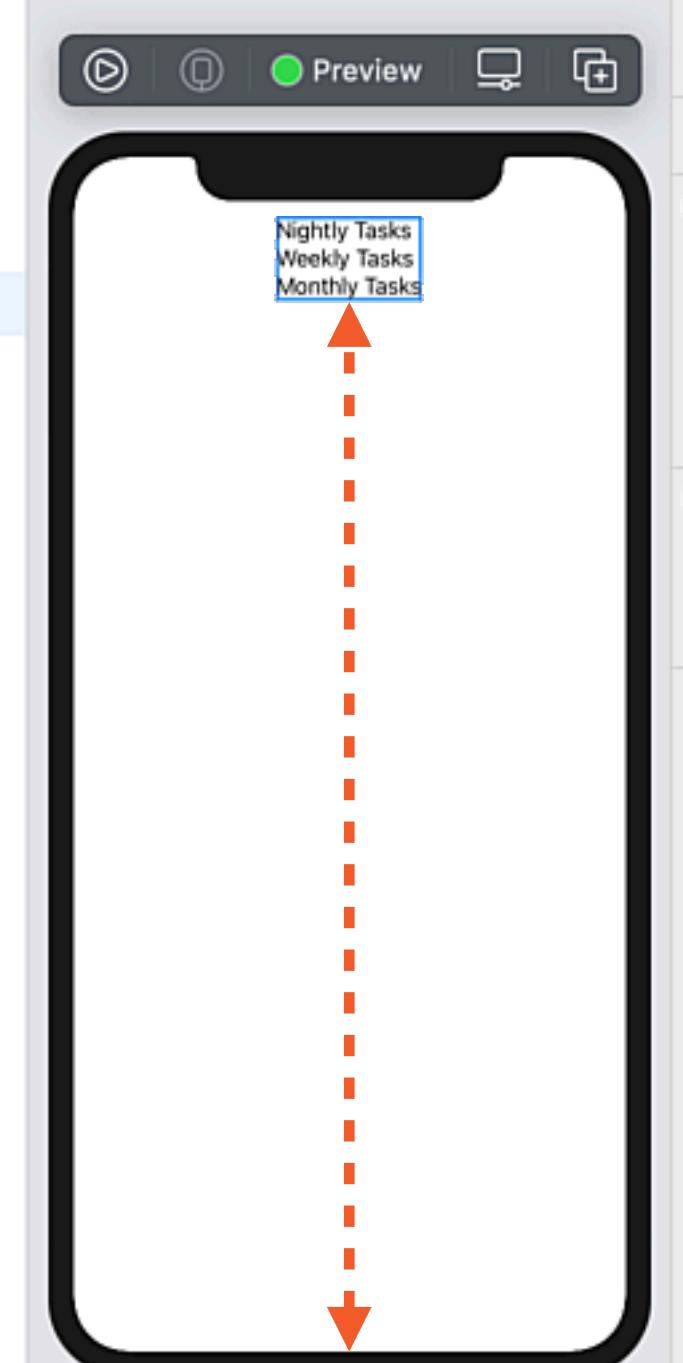
Padding: - Inherited +

Frame

Size: - Inher + Width: - Inher + Height: - Inher +

Add Modifier

Preview



A screenshot of the Xcode interface showing a Swift file named ContentView.swift. The code defines a struct ContentView that returns a VStack with three Text elements: "Nightly Tasks", "Weekly Tasks", and "Monthly Tasks". The alignment is set to .leading. A preview of the app on an iPhone 11 shows the three tasks stacked vertically. The Xcode interface also includes a sidebar with project files like NightWatchApp.swift and ContentView.swift, and a right-hand panel for modifying the preview's alignment.

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:41 AM

ContentView.swift

NightWatchApp.swift

NightWatch

NightWatchApp.swift

ContentView.swift

Assets.xcassets

Info.plist

Preview Content

Products

Vertical Stack

Spacing: Inherited

Alignment: 

Modifiers

Padding

Padding: Inherited

Frame

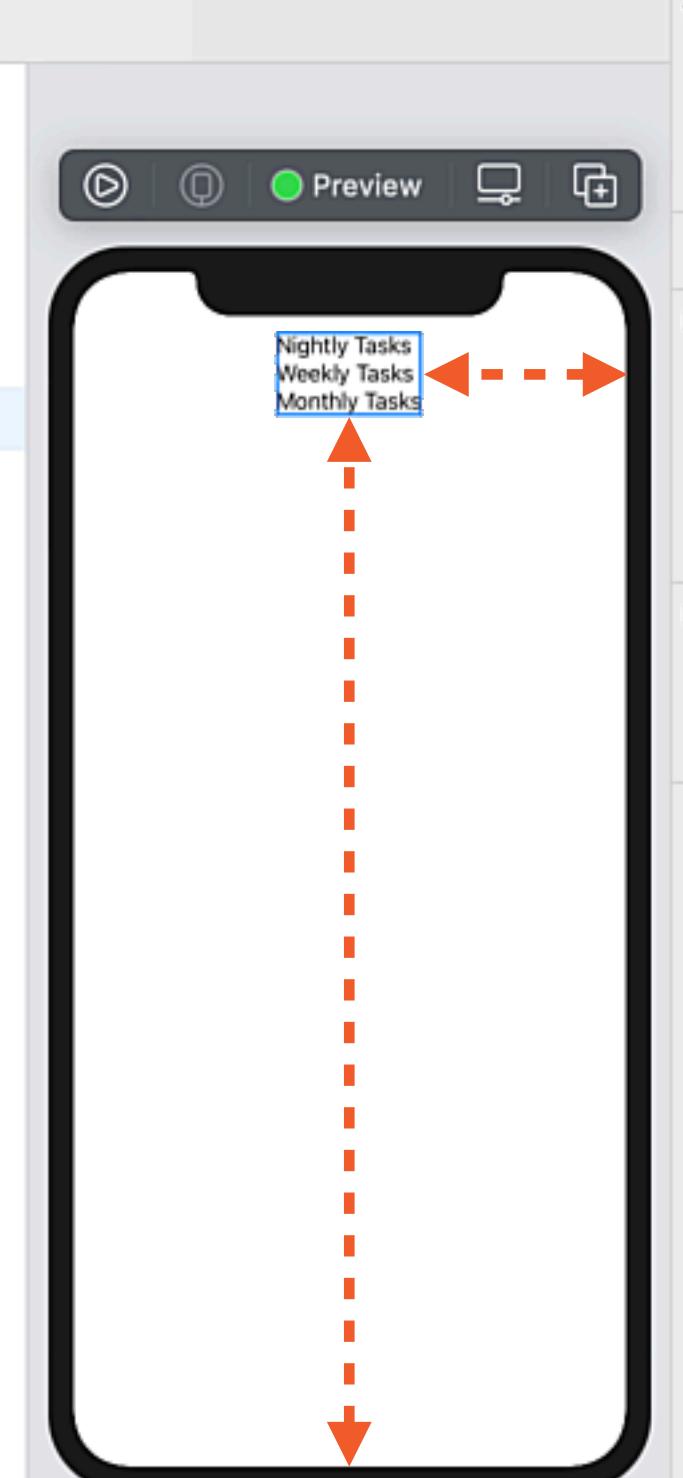
Size: Inher + Width: Inher + Height:

Add Modifier

Filter

50%

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         VStack(alignment: .leading) {
6             Text("Nightly Tasks")
7             Text("Weekly Tasks")
8             Text("Monthly Tasks")
9         }
10    }
11 }
12
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         ContentView()
17     }
18 }
19
```



The screenshot shows the Xcode interface with the ContentView.swift file open. The code defines a simple SwiftUI view structure. The preview window displays three lines of text: "Nightly Tasks", "Weekly Tasks", and "Monthly Tasks", all aligned to the leading edge of the screen. The right-hand sidebar contains the 'Vertical Stack' configuration panel, where the 'Alignment' dropdown is set to 'Leading'. A dashed red arrow points from the 'Leading' icon in the Alignment dropdown to the 'Leading' alignment icon in the preview window's toolbar, indicating the connection between the configuration and the resulting UI.

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:41 AM

ContentView.swift NightWatchApp.swift

NightWatch

NightWatch

NightWatchApp.swift ContentView.swift Assets.xcassets Info.plist Preview Content Products

```
1 import SwiftUI
2
3 struct ContentView: View {
4     var body: some View {
5         VStack(alignment: .leading) {
6             Text("Nightly Tasks")
7             Text("Weekly Tasks")
8             Text("Monthly Tasks")
9         }
10    }
11 }
12
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         ContentView()
17     }
18 }
19
```

Vertical Stack

Spacing: - Inherited +

Alignment:

Modifiers

Padding

Padding: - Inherited +

Frame

Size: - Inher + Width: - Inher + Height: - Inher +

Add Modifier

Preview

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:41 AM

ContentView.swift NightWatchApp.swift

NightWatch

NightWatch

NightWatchApp.swift ContentView.swift Assets.xcassets Info.plist Preview Content Products

NightWatch > NightWatch > ContentView.swift body

```
1
2 import SwiftUI
3
4 struct ContentView: View {
5     var body: some View {
6         VStack(alignment: .leading) {
7             Text("Nightly Tasks")
8             Text("Weekly Tasks")
9             Text("Monthly Tasks")
10        }
11    }
12 }
13
14 struct ContentView_Previews: PreviewProvider {
15     static var previews: some View {
16         ContentView()
17     }
18 }
19
```

Vertical Stack

Spacing: - Inherited +

Alignment:   

Modifiers

Padding

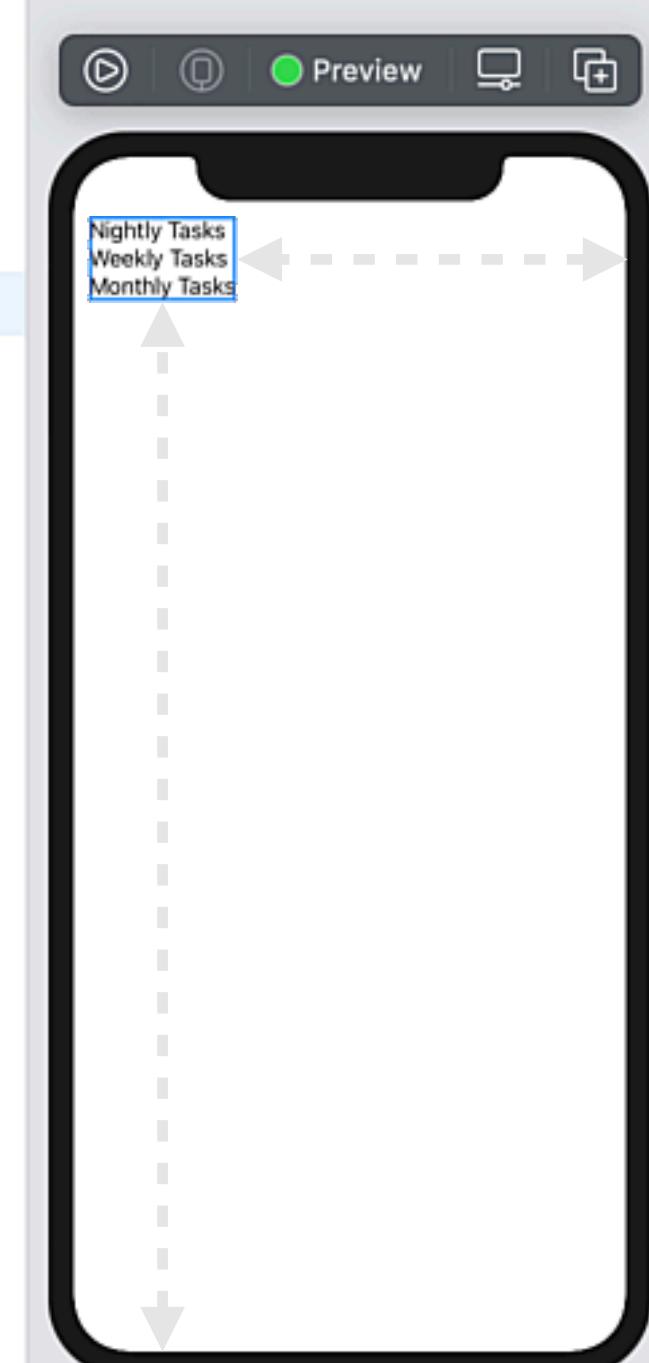
Padding: - Inherited +

Frame

Size: - Inher + Width: - Inher + Height: - Inher +

Add Modifier

Preview



Filter

V... 10... 50%

Customizing UI Appearance

9:41



Nightly Tasks

Weekly Tasks

Monthly Tasks

9:41



Nightly Tasks

Weekly Tasks

Monthly Tasks

9:41



Nightly Tasks

Weekly Tasks

Monthly Tasks

9:41



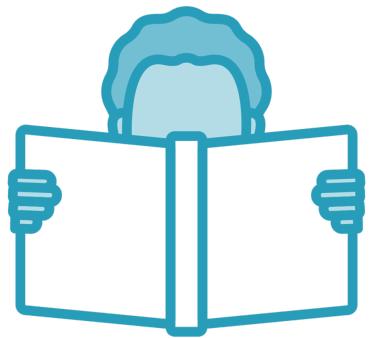
Nightly Tasks

Weekly Tasks

Monthly Tasks



Swift**UI**



Swift**UI**



How do we customize the overall user experience of SwiftUI Views and **modify their appearance to our liking?**

```
let text = Text("Nightly Tasks")
text.font = Font.title3
text.fontWeight = .bold
text.foregroundColor = .yellow
```

Traditional UI Framework

Create a **text object**

Update its **font property**

Update its **fontWeight property**

Update its **foregroundColor property**

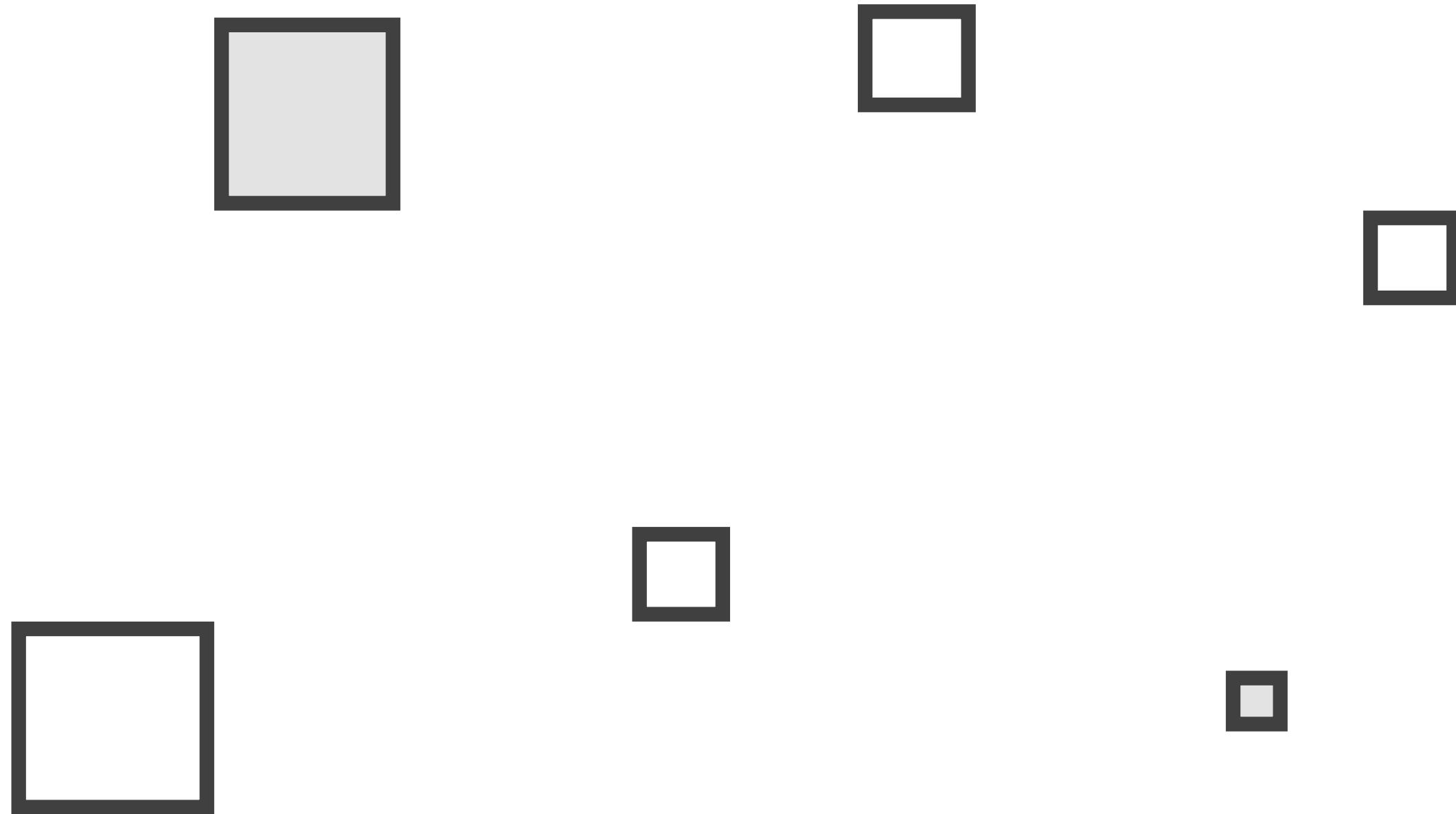
SwiftUI Views do not have
any publicly-accessible
stored properties.

Developing a SwiftUI Mindset

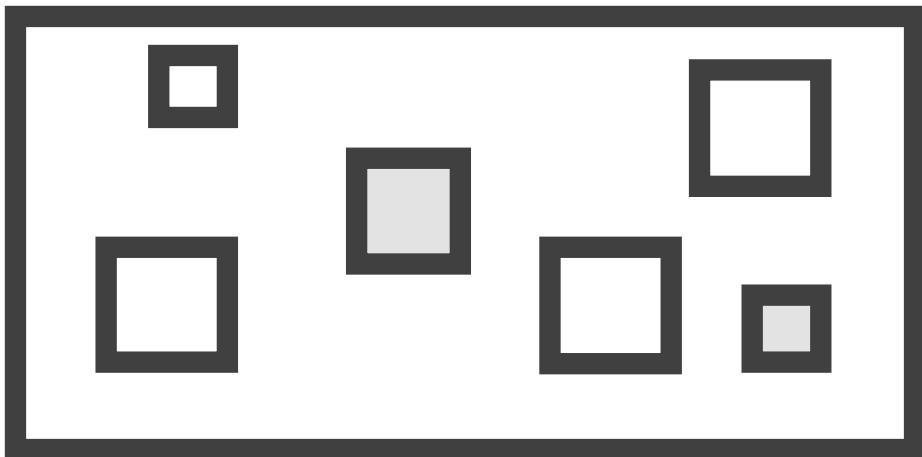


View Protocol

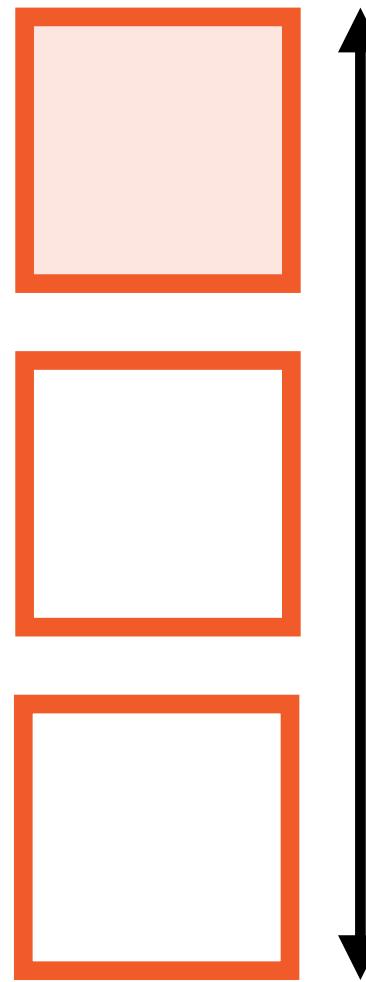
- ✓ A **computed property...**
- ✓ **named body...**
- ✓ **that returns some View**



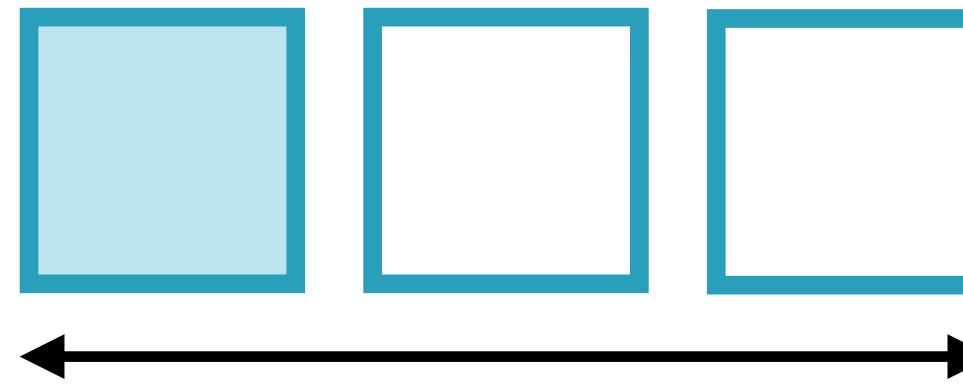
Layout container



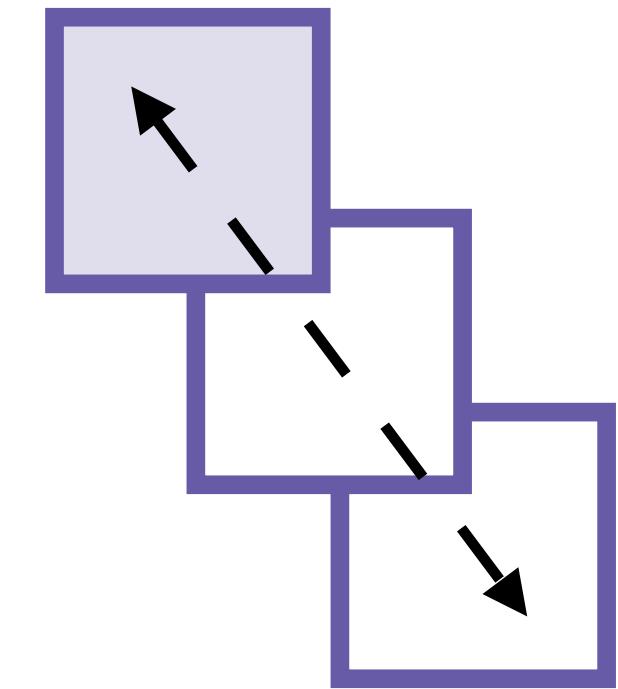
Layout Directions



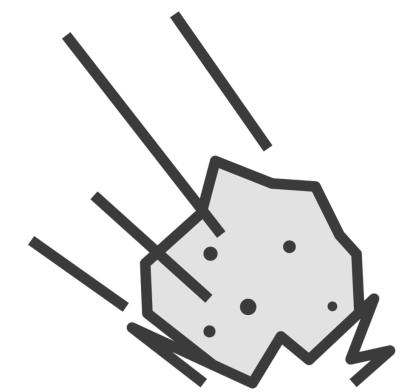
V Stack

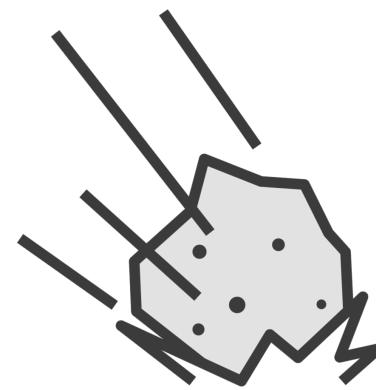


H Stack

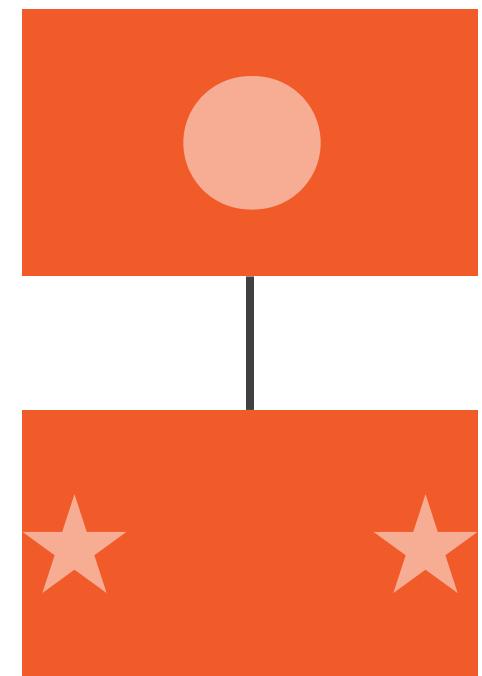


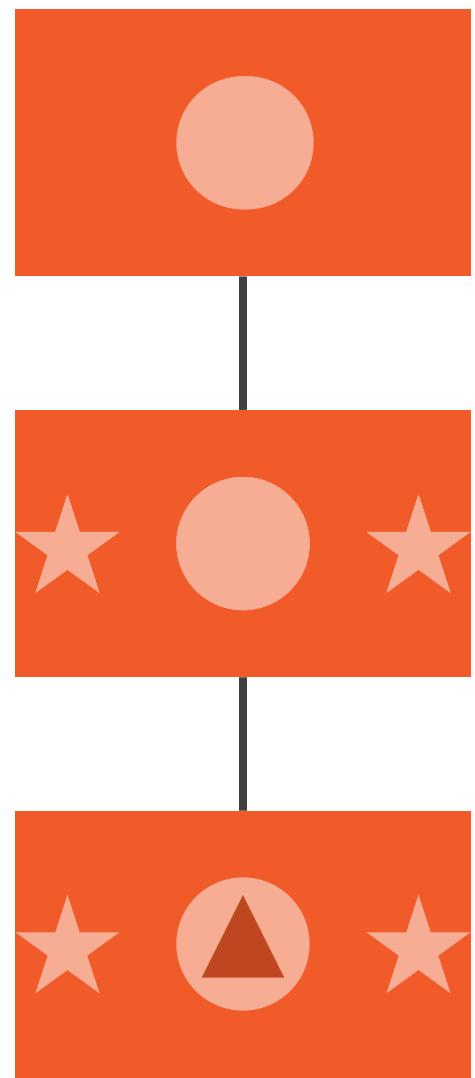
Z Stack

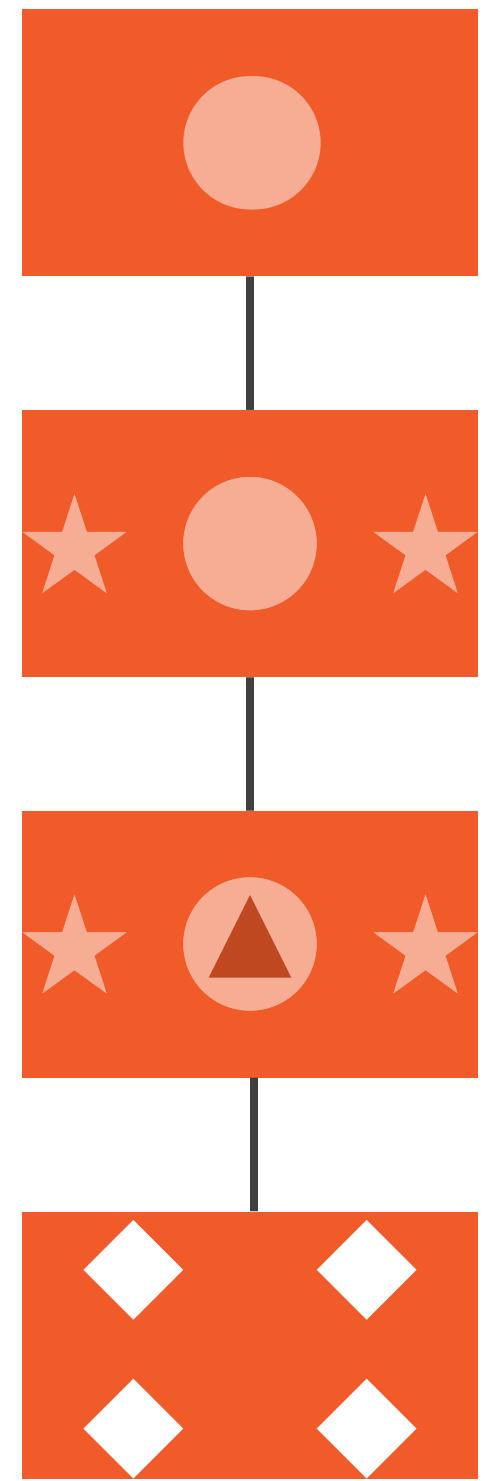




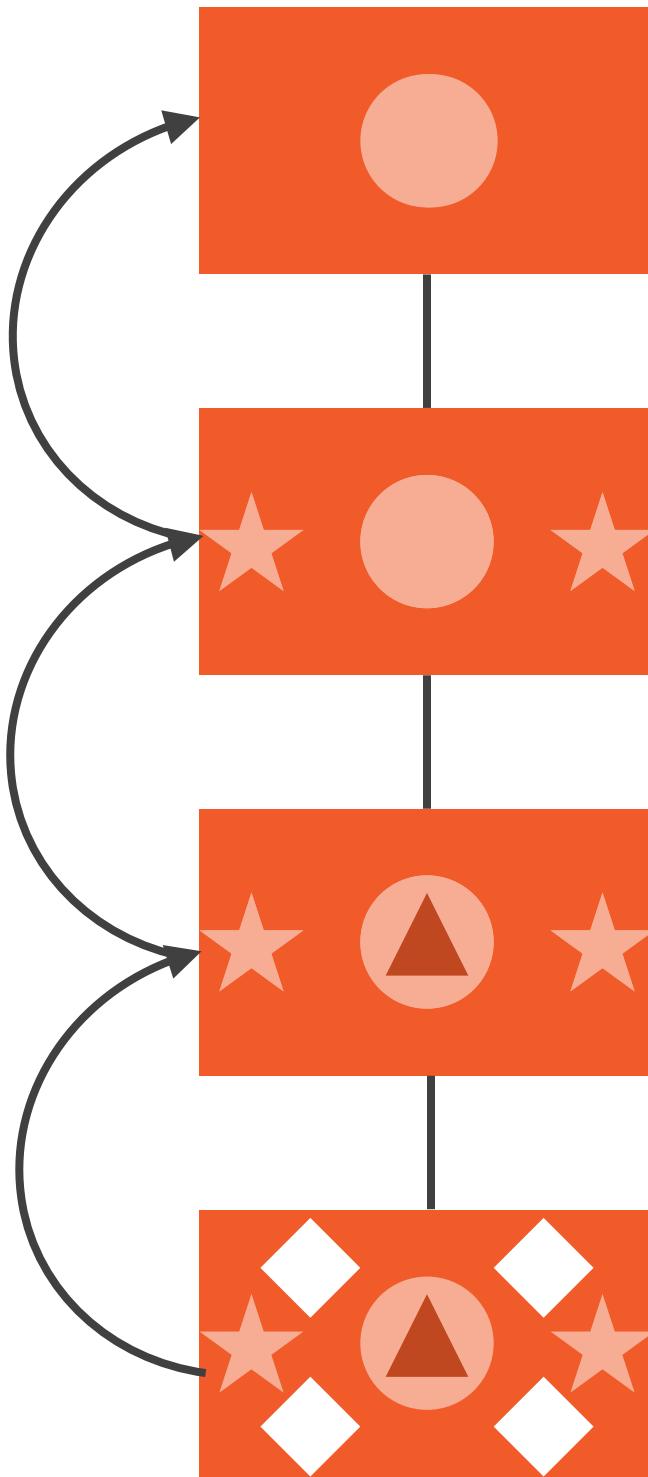
SwiftUI Views are **structs**







Base Class



developer.apple.com

Apple Developer Discover Design Develop Distribute Support Account

Documentation > UIKit > Views and Controls > UIView

Language: Swift API Changes: None

Class

UIView

An object that manages the content for a rectangular area on the screen.

Availability

iOS 2.0+
Mac Catalyst 13.0+
tvOS 9.0+

Declaration

```
class UIView : UIResponder
```

Framework

UIKit

On This Page

Declaration 
Overview 
Topics 
Relationships 
See Also 

Overview

Views are the fundamental building blocks of your app's user interface, and the [UIView](#) class defines the behaviors that are common to all views. A view object renders content within its bounds rectangle and handles any interactions with that content. The [UIView](#) class is a concrete class that you can instantiate and use to display a fixed background color. You can also subclass it to draw more sophisticated content. To display labels, images, buttons, and other interface elements commonly found in apps, use the view subclasses provided by the

Topics

Creating a View Object

`init(frame: CGRect)`

Initializes and returns a newly allocated view object with the specified frame rectangle.

`init?(coder: NSCoder)`

Configuring a View's Visual Appearance

`var backgroundColor: UIColor?`

The view's background color.

`var isHidden: Bool`

A Boolean value that determines whether the view is hidden.

`var alpha: CGFloat`

The view's alpha value.

`var isOpaque: Bool`

A Boolean value that determines whether the view is opaque.

`var tintColor: UIColor!`

The first nondefault tint color value in the view's hierarchy, ascending from and starting with the view itself.

A Boolean value that determines whether the view's autoresizing mask is translated into Auto Layout constraints.

Adjusting the User Interface

`var overrideUserInterfaceStyle: UIUserInterfaceStyle`

The user interface style adopted by the view and all of its subviews.

`var semanticContentAttribute: UISemanticContentAttribute`

A semantic description of the view's contents, used to determine whether the view should be flipped when switching between left-to-right and right-to-left layouts.

`var effectiveUserInterfaceLayoutDirection: UIUserInterfaceLayoutDirection`

The user interface layout direction appropriate for arranging the immediate content of the view.

`class func userInterfaceLayoutDirection(for: UISemanticContentAttribute) -> UIUserInterfaceLayoutDirection`

Returns the user interface direction for the given semantic content attribute.

`class func userInterfaceLayoutDirection(for: UISemanticContentAttribute, relativeTo: UIUserInterfaceLayoutDirection) -> UIUserInterfaceLayoutDirection`

Returns the layout direction implied by the specified semantic content attribute, relative to the specified layout direction.



“There’s no such thing as a free lunch...”

Inheritance



Inheritance



9:41



Button



Inheritance

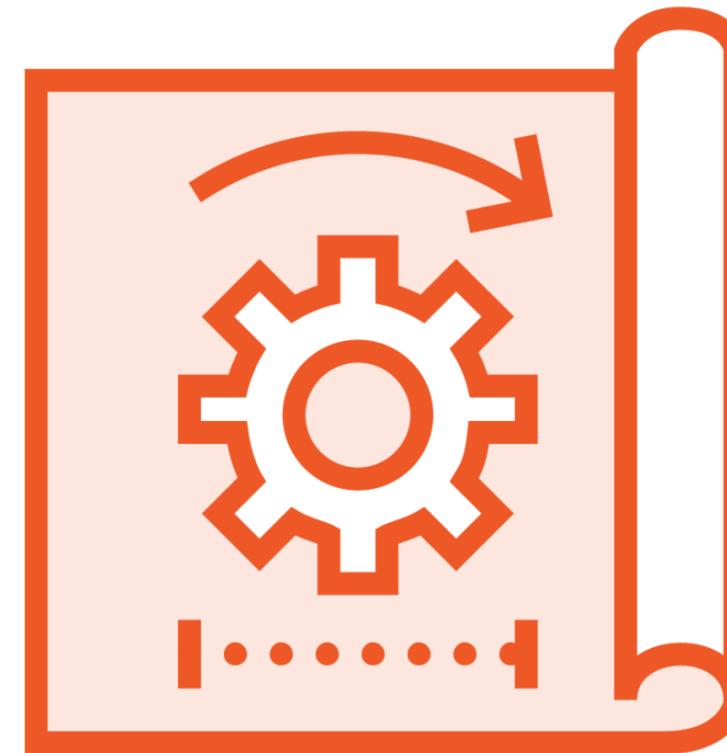


9:41

Button



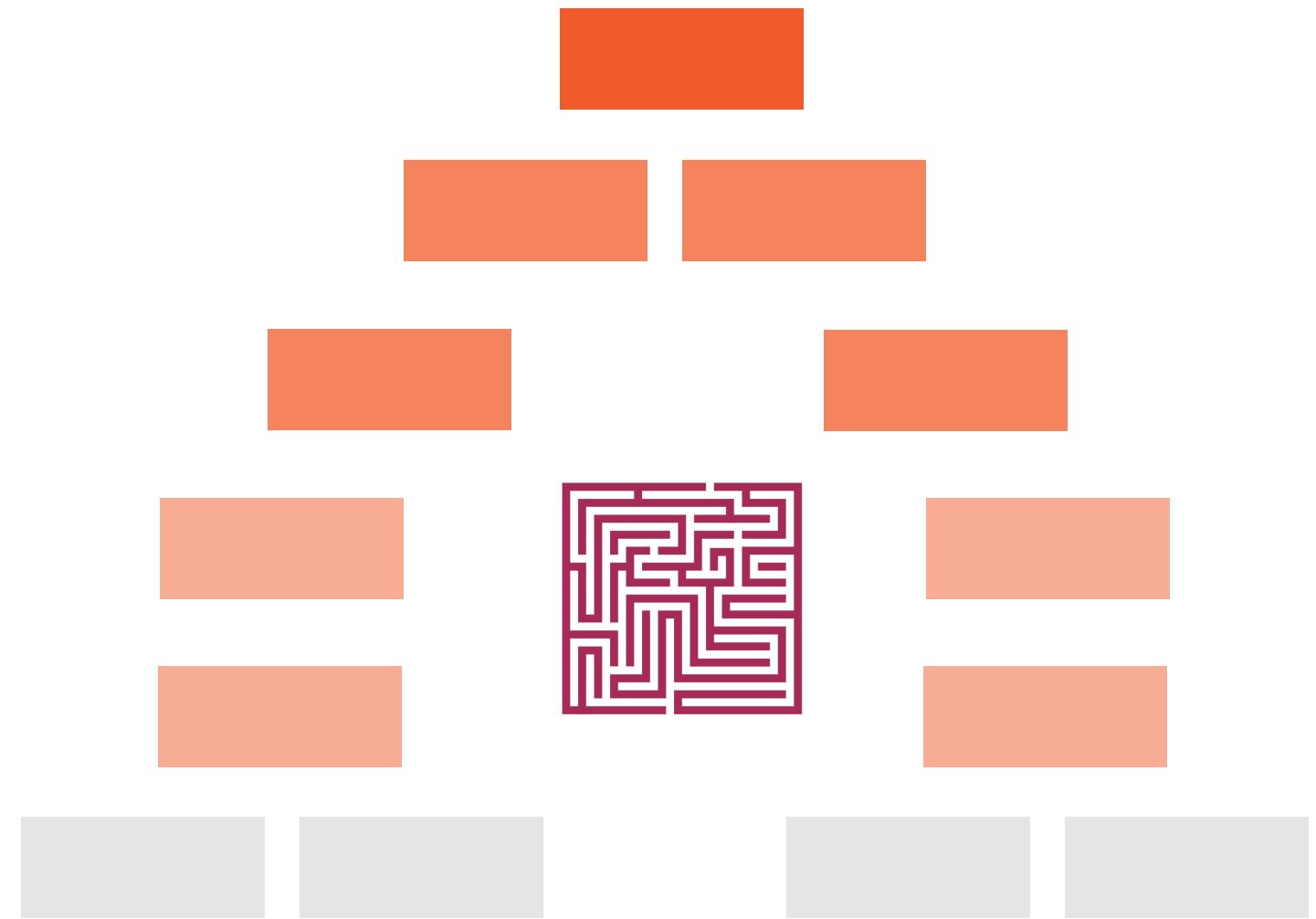
“Free” things



All-new UI framework



Swift is the way forward



Protocol A

Protocol B

Protocol C

Protocol D

MyStruct

Protocol A

Protocol B

Protocol C

Protocol D

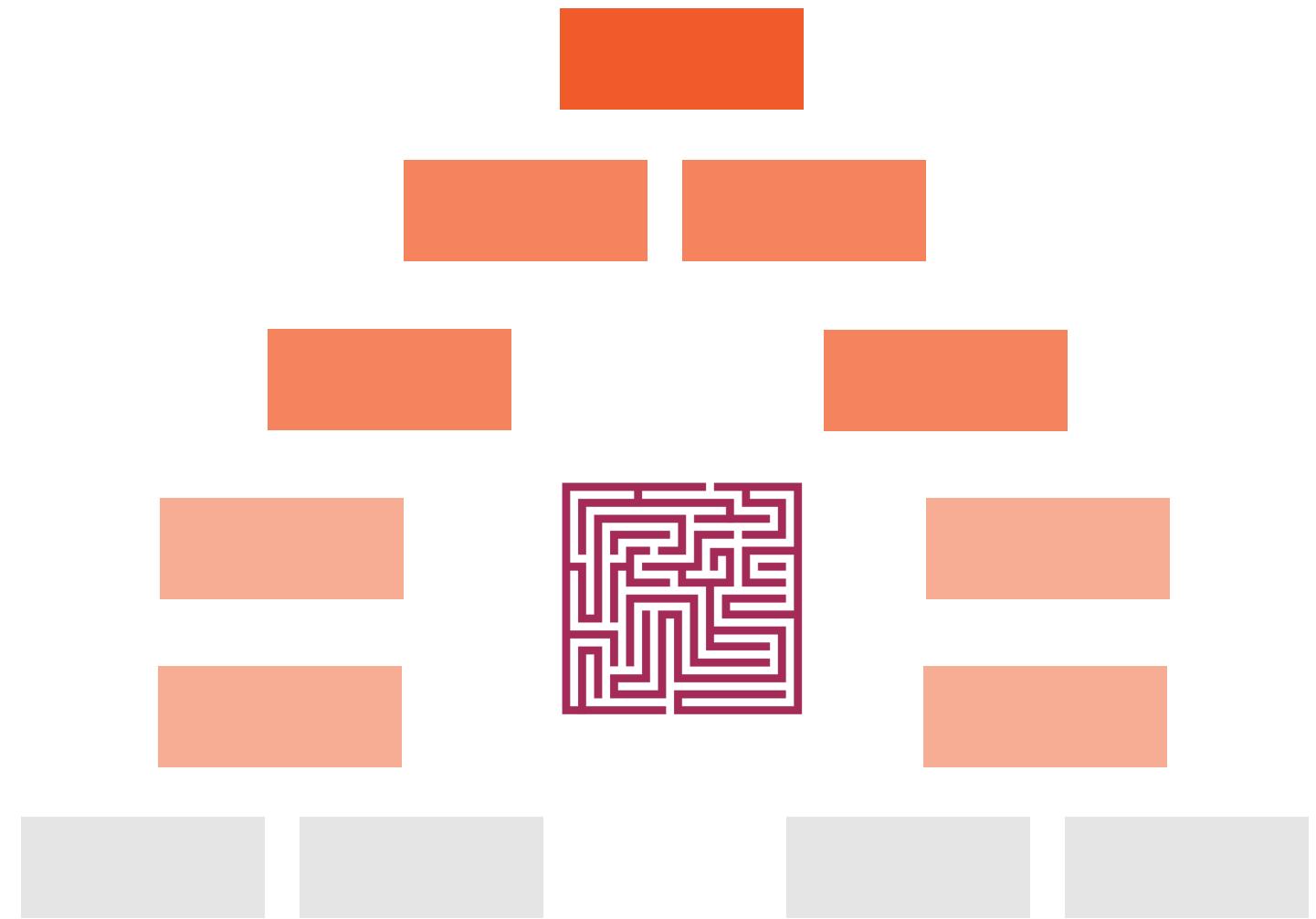
MyStruct

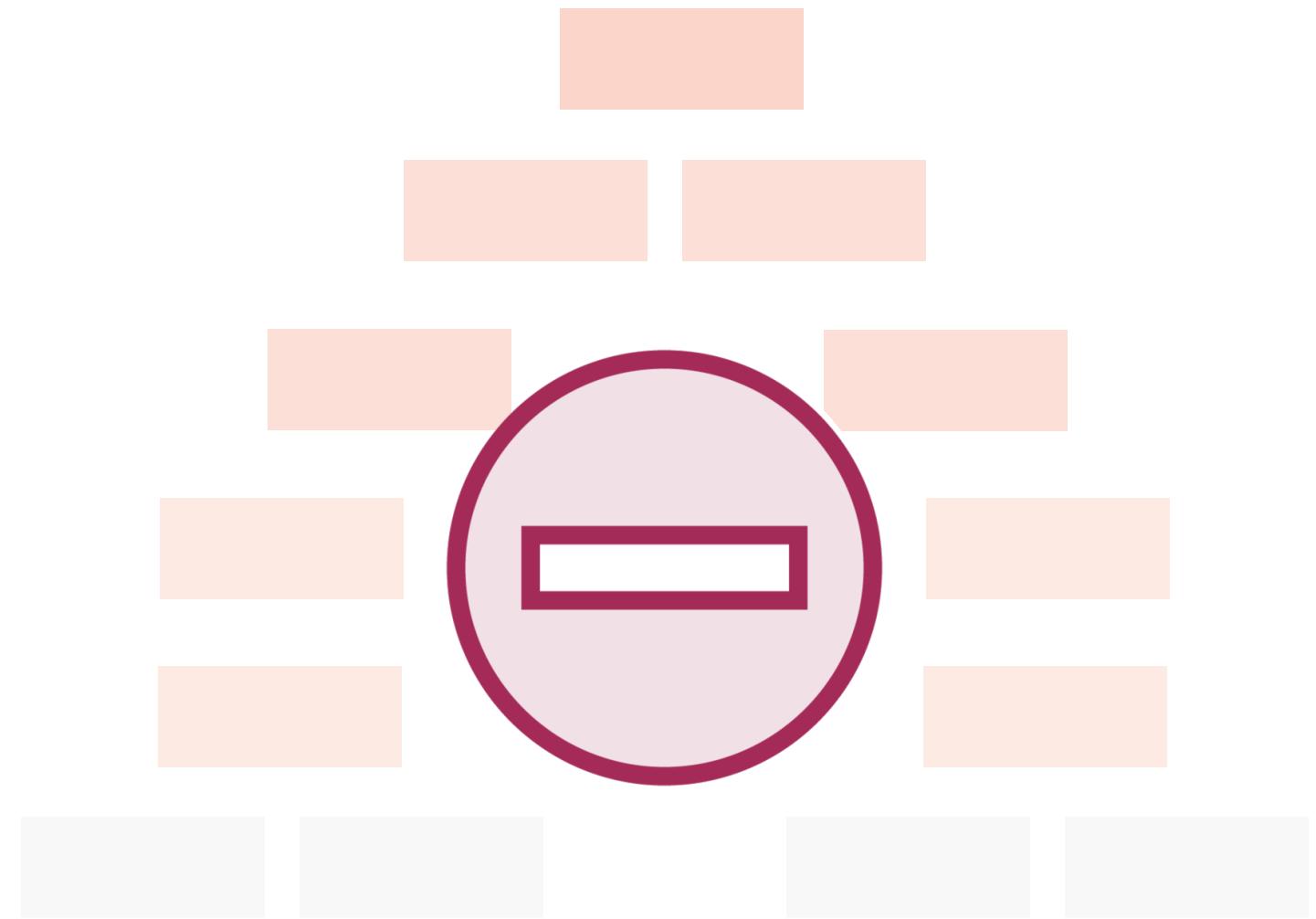
Protocol A

{ ... }

Protocol C

{ ... }





View

A type that represents part of your app's user interface and provides modifiers that you use to configure views.

Availability
iOS 13.0+
macOS 10.15+
Mac Catalyst 13.0+
tvOS 13.0+
watchOS 6.0+

Framework

SwiftUI

On This Page

[Declaration](#) ⓘ
[Overview](#) ⓘ
[Topics](#) ⓘ
[Relationships](#) ⓘ
[See Also](#) ⓘ

```
protocol View
```

Declaration

```
struct MyView: View {  
    var body: some View {  
        Text("Hello, World!")  
    }  
}
```

Assemble the view's body by combining one or more of the primitive views provided by SwiftUI, like the [Text](#) instance in the example above, plus other custom views that you define, into a

macOS 10.15+

Mac Catalyst 13.0+

tvOS 13.0+

watchOS 6.0+

Framework

SwiftUI

On This Page

[Declaration](#) ⓘ

[Overview](#) ⓘ

[Topics](#) ⓘ

[Relationships](#) ⓘ

[See Also](#) ⓘ

```
protocol View
```

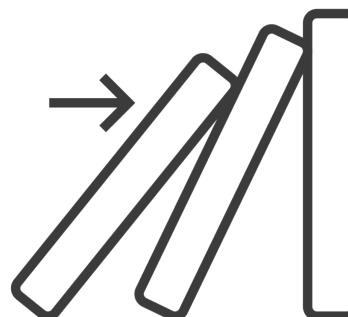
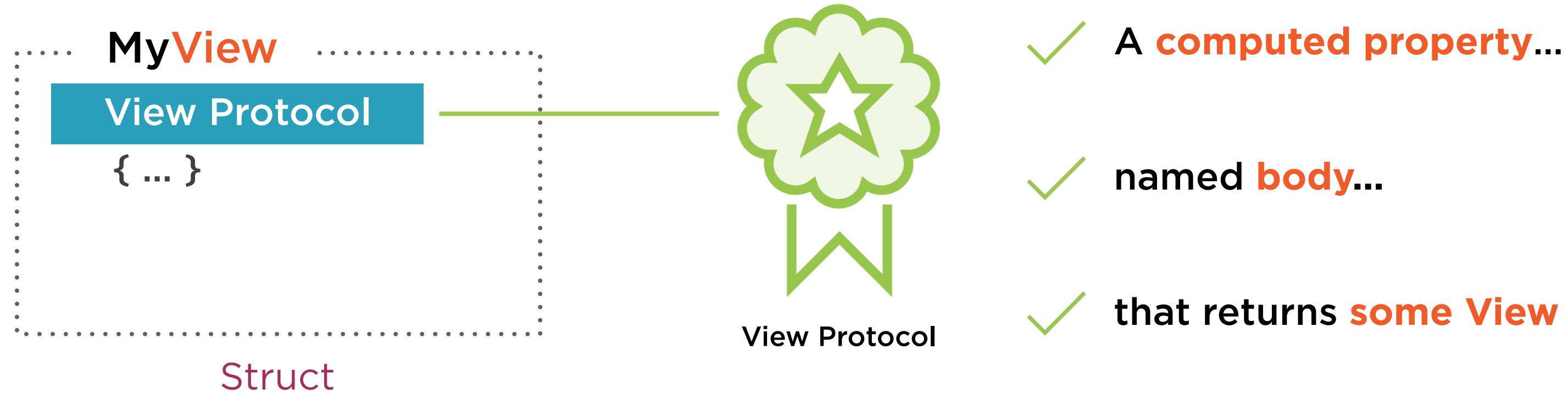
Declaration

You create custom views by declaring types that conform to the [View](#) protocol. Implement the required [body](#) computed property to provide the content for your custom view.

```
struct MyView: View {  
    var body: some View {  
        Text("Hello, World!")  
    }  
}
```

Assemble the view's body by combining one or more of the primitive views provided by SwiftUI, like the [Text](#) instance in the example above, plus other custom views that you define, into a hierarchy of views.

The [View](#) protocol provides a large set of modifiers, defined as protocol methods with default implementations, that you use to position and configure views in the layout of your app. Modifiers typically work by wrapping the view instance on which you call them in another view with the specified characteristics. For example, adding the [opacity\(_:\) modifier](#) to a text view returns a new view with some amount of transparency:



We will feel and experience the cascading effects of Apple's design choice here, every single day of our iOS developer experience.

```
struct MyView : View
```

```
var body {
```

```
    HStack
```

```
        Image View
```



```
        Text View
```

```
Hello, world!
```

```
}
```

```
struct MyView : View
```

```
var body {
```

```
    HStack
```

```
        Image View
```



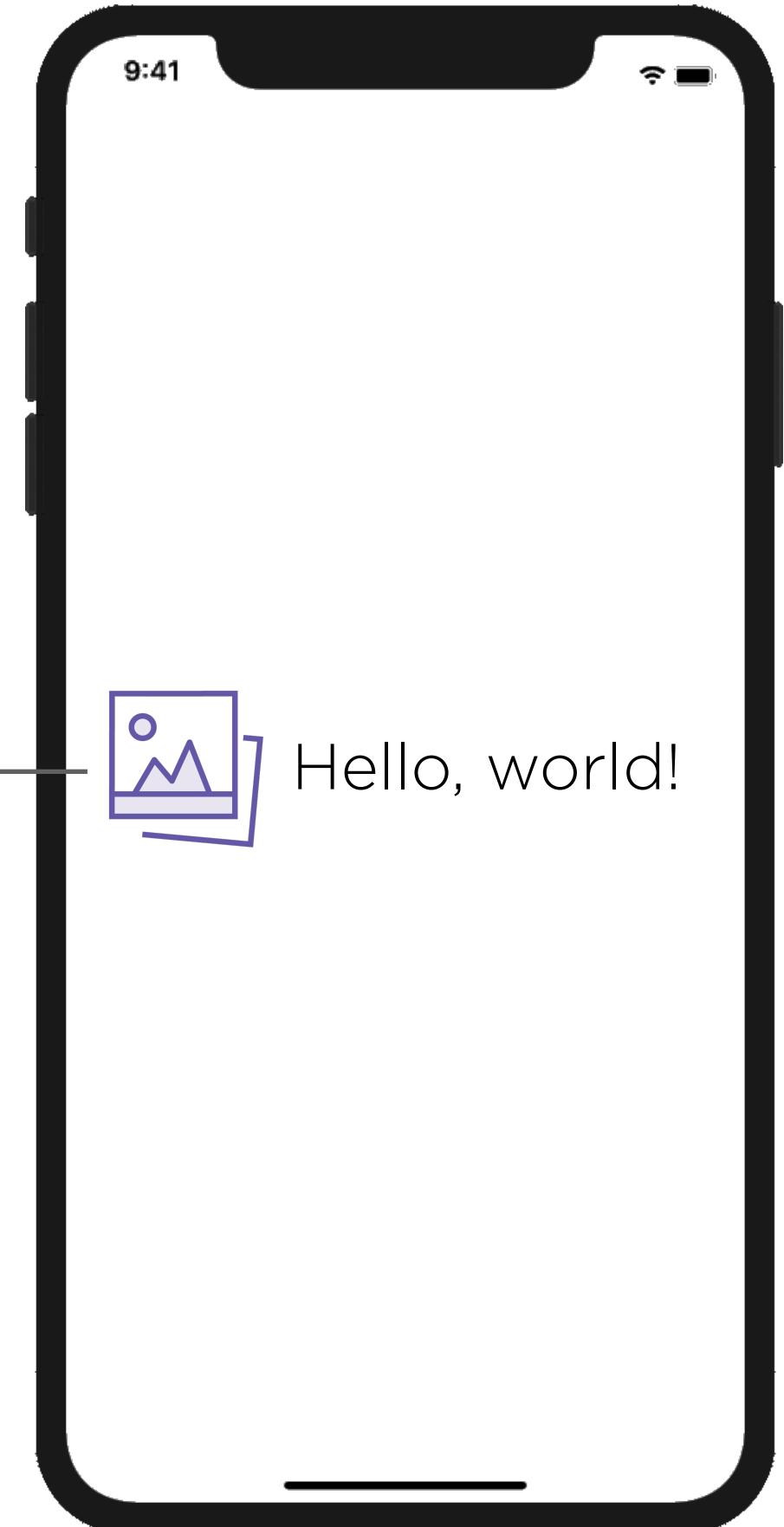
```
        Text View
```

```
Hello, world!
```

```
}
```

```
struct MyView : View {  
    var body {  
        HStack {  
            Image View  
            Text View  
        }  
    }  
}
```

MyView instance
value





It one of the
nfluential
rs, creators and
preneurs of all
Steve Jobs was
the greatest CEO
generation.

p. CEO Rupert Murdoch

Think Different

Once in a rare while,
somebody comes along
who doesn't just raise
the bar, they create an
entirely new standard of
measurement.

witter CEO Dick Costol

The world has lost
a visionary. And there
may be no greater
tribute to Steve's
success than the fact
that much of the world
learned of his passing on
a device he invented.

President Barack Obama

For those of us lucky
enough to get to work
with him, it's been
an insanely great
honor. I will miss Steve
immensely.

Microsoft Chairman Bill Gates

Steve Jobs Watch



Seiko 1977
Chronograph
Vintage LCD
M159-5026
Stainless Steel

Apple has lost a visionary
and creative genius, and
the world has lost an
amazing human being.
Those of us who have
been fortunate enough
to know and work with
Steve have lost a dear
friend and an inspiring
mentor

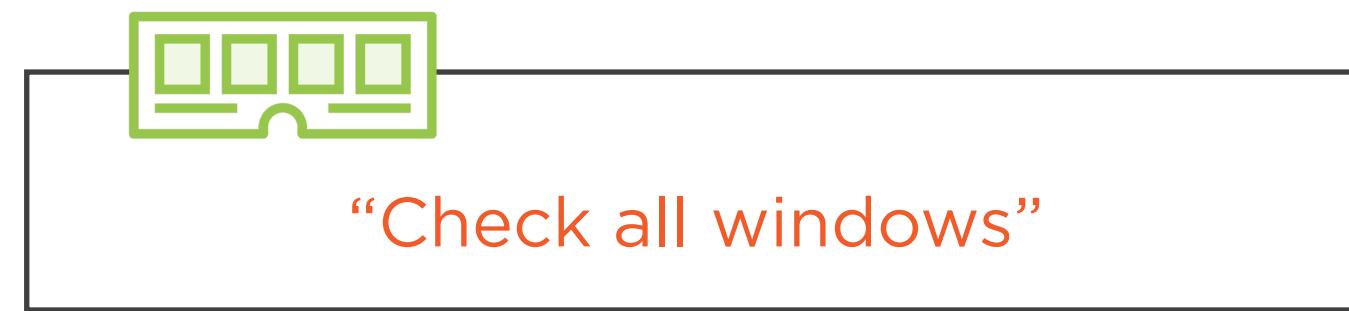
Apple CEO Tim Cook

Steve, thank you for
being a mentor and
a friend. Thanks for
showing that what you
build can change the
world. I will miss you.

Facebook CEO Mark Zuckerberg

In his public life,
Steve was known
as a visionary; in
his private life, he
cherished his family.

Steve Jobs' Family



```
var firstTask = “Check all windows”
```

```
var secondTask = firstTask
```

Reference Type (Class)
Semantics



```
var firstTask = “Check all windows”
```



```
var secondTask = firstTask
```

Value Type (Struct)
Semantics



`var firstTask = "Check all windows"`

`var secondTask = firstTask`



Application Complexity

Value Type (Struct)
Semantics



`var firstTask = "Check all windows"`

`var secondTask = firstTask`

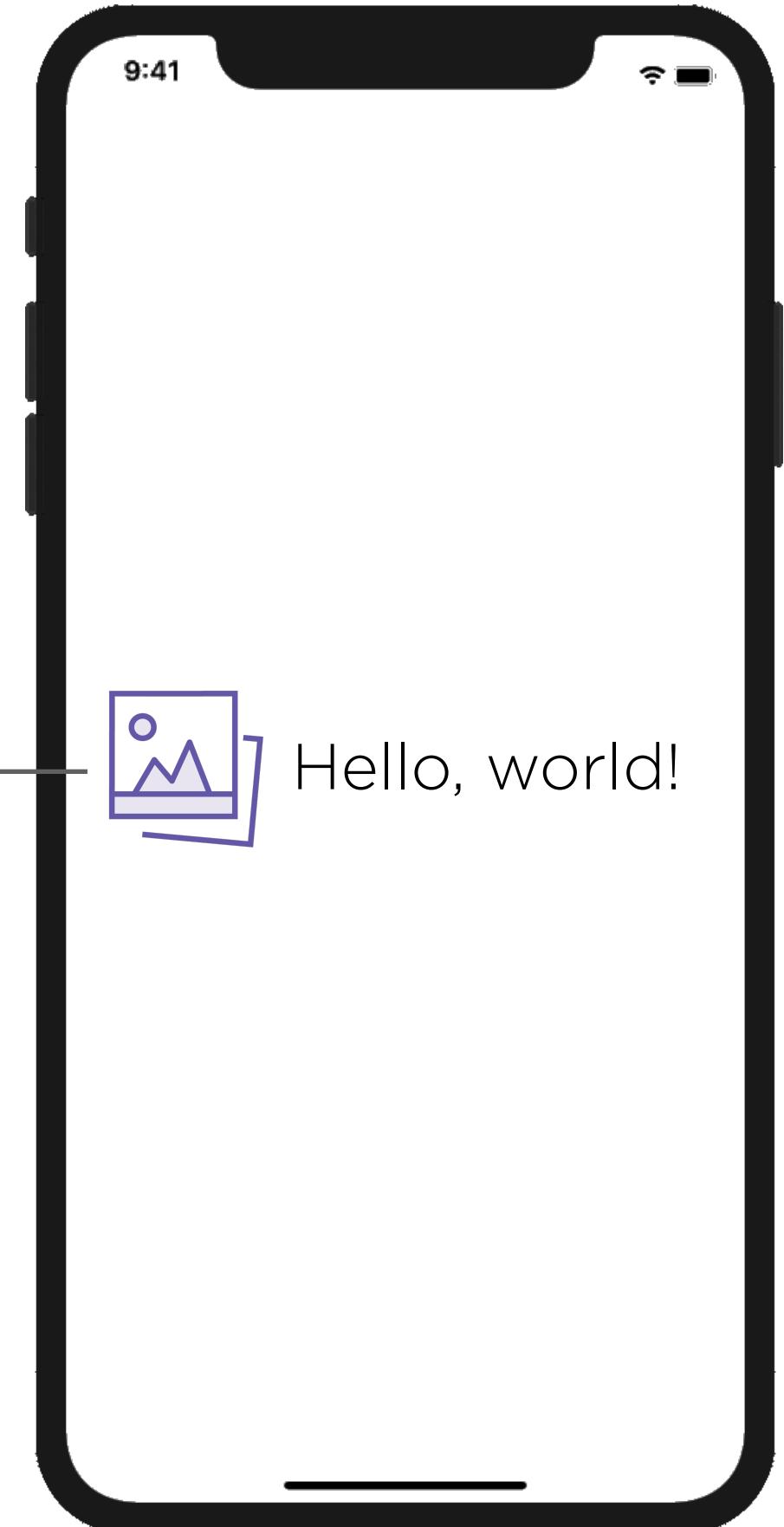


Application Complexity

Value Type (Struct)
Semantics

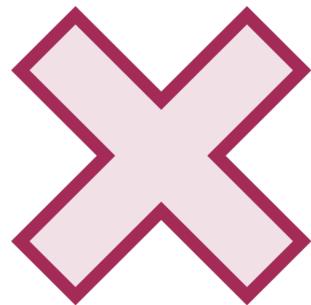
```
struct MyView : View {  
    var body {  
        HStack {  
            Image View  
            Text View  
        }  
    }  
}
```

MyView instance
value

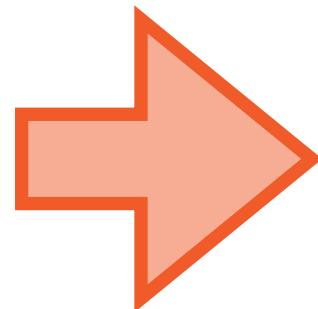


Values are **immutable**

“Lies!”, you say! “We can change views -- just use add on view modifiers, and you’re good to go!”



“Immutable” does NOT mean that we can **never change anything**.



“Immutable” means that alterations **cannot be made directly** to existing values **in-place**.









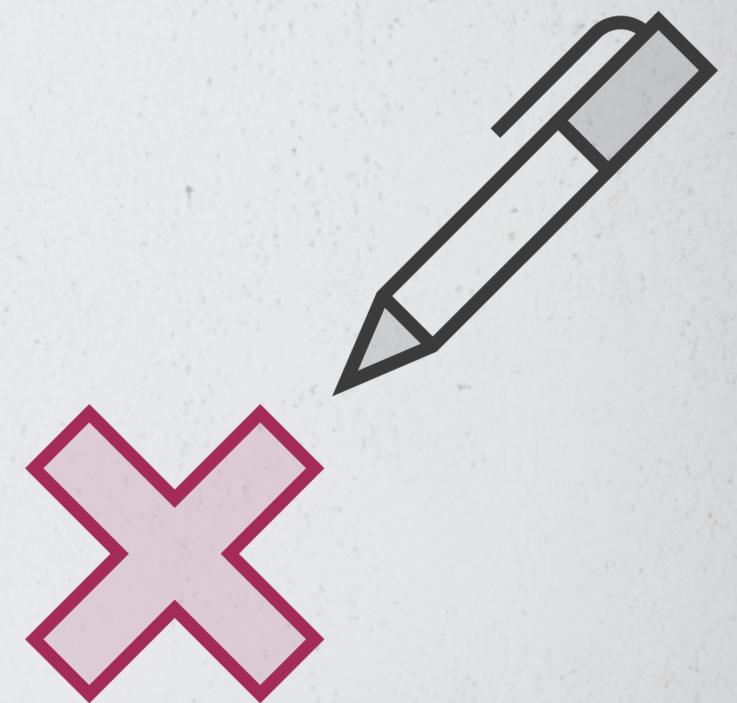


value



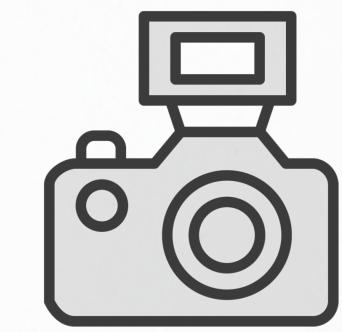


value









new value



Did a **change** occur with the photo?

new value



Did a **change** occur with the photo?



value

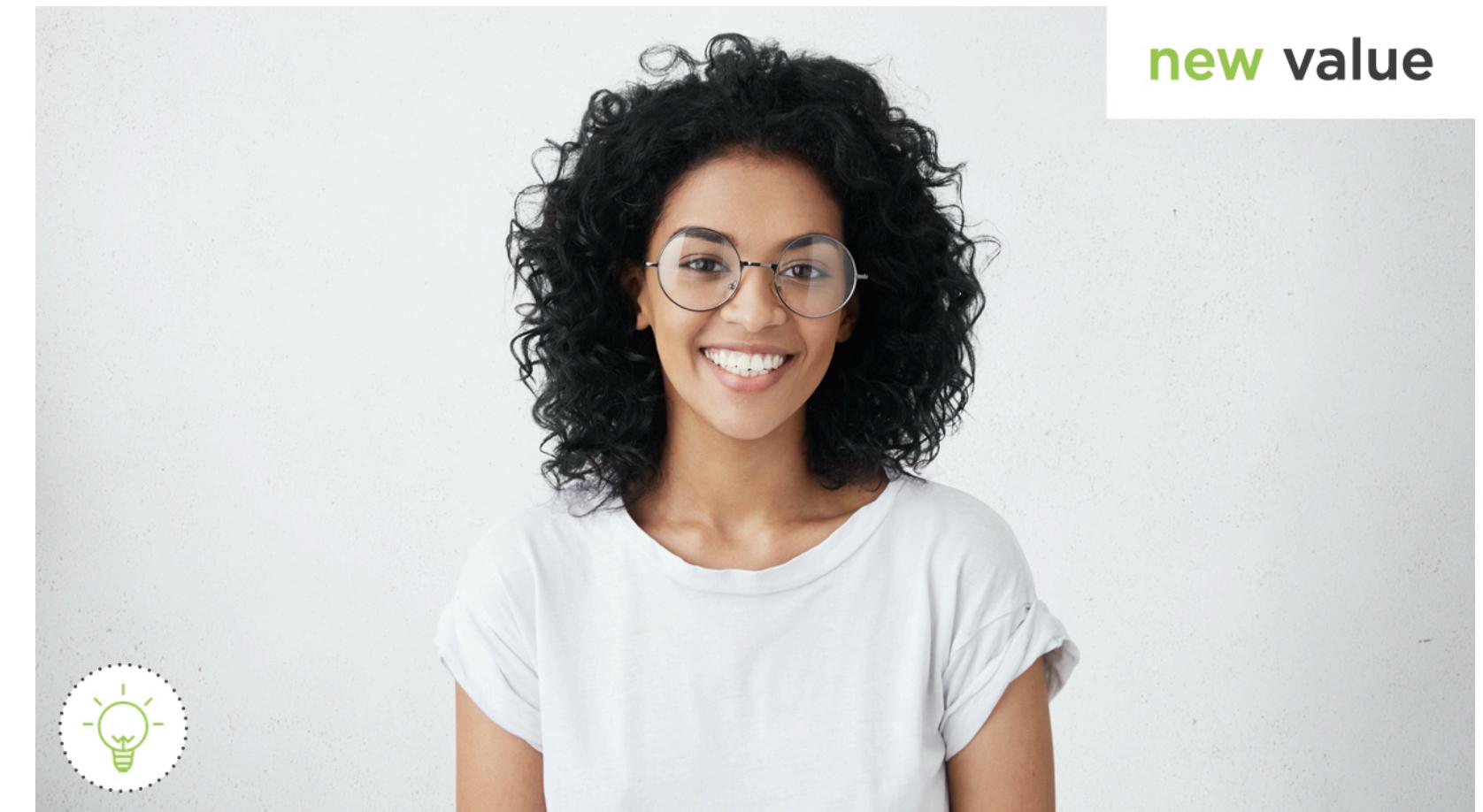


new value

Did a **change** occur with the photo?



value

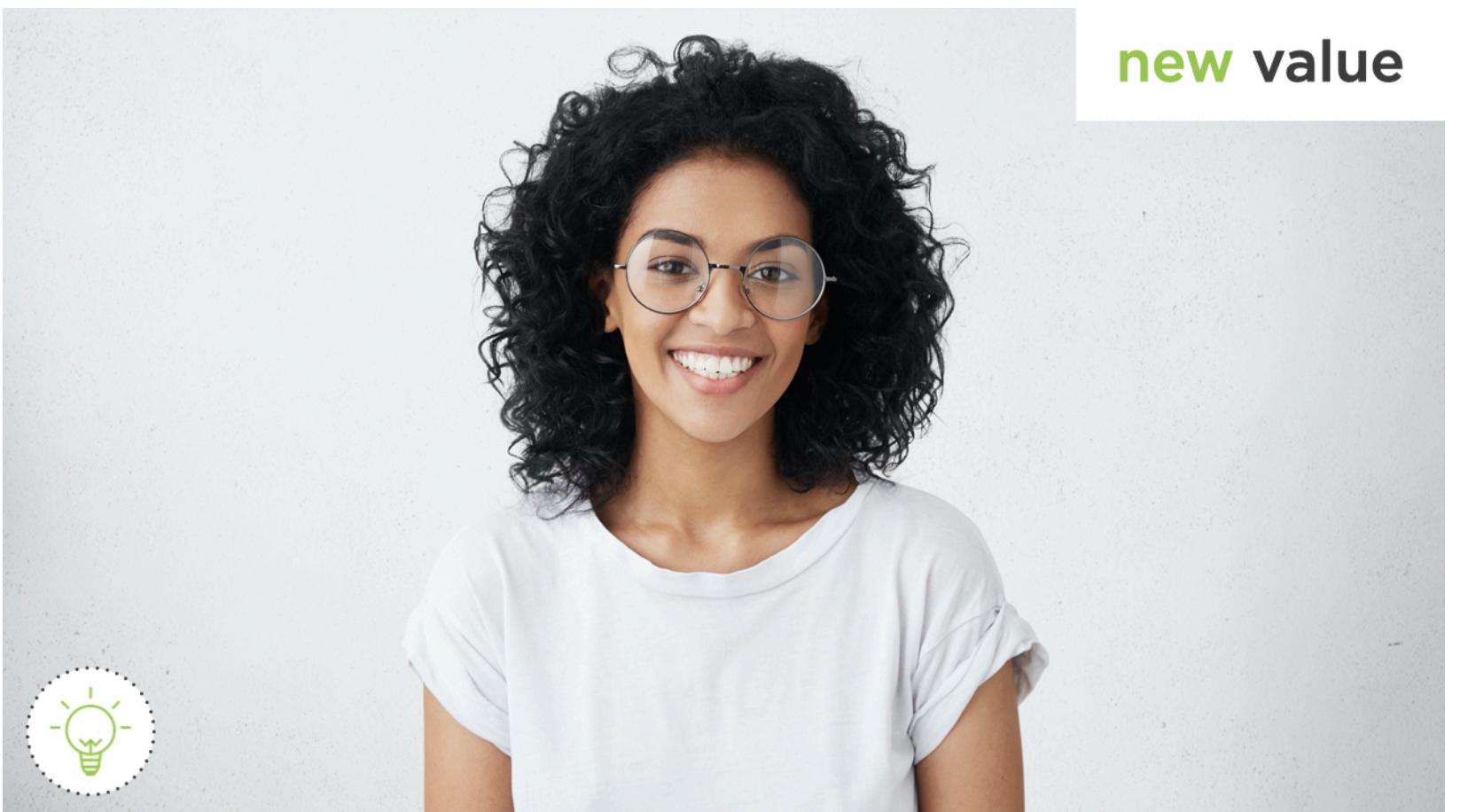


new value

We modified the photo by completely replacing it with a new value.



value



new value

MyStruct Instance

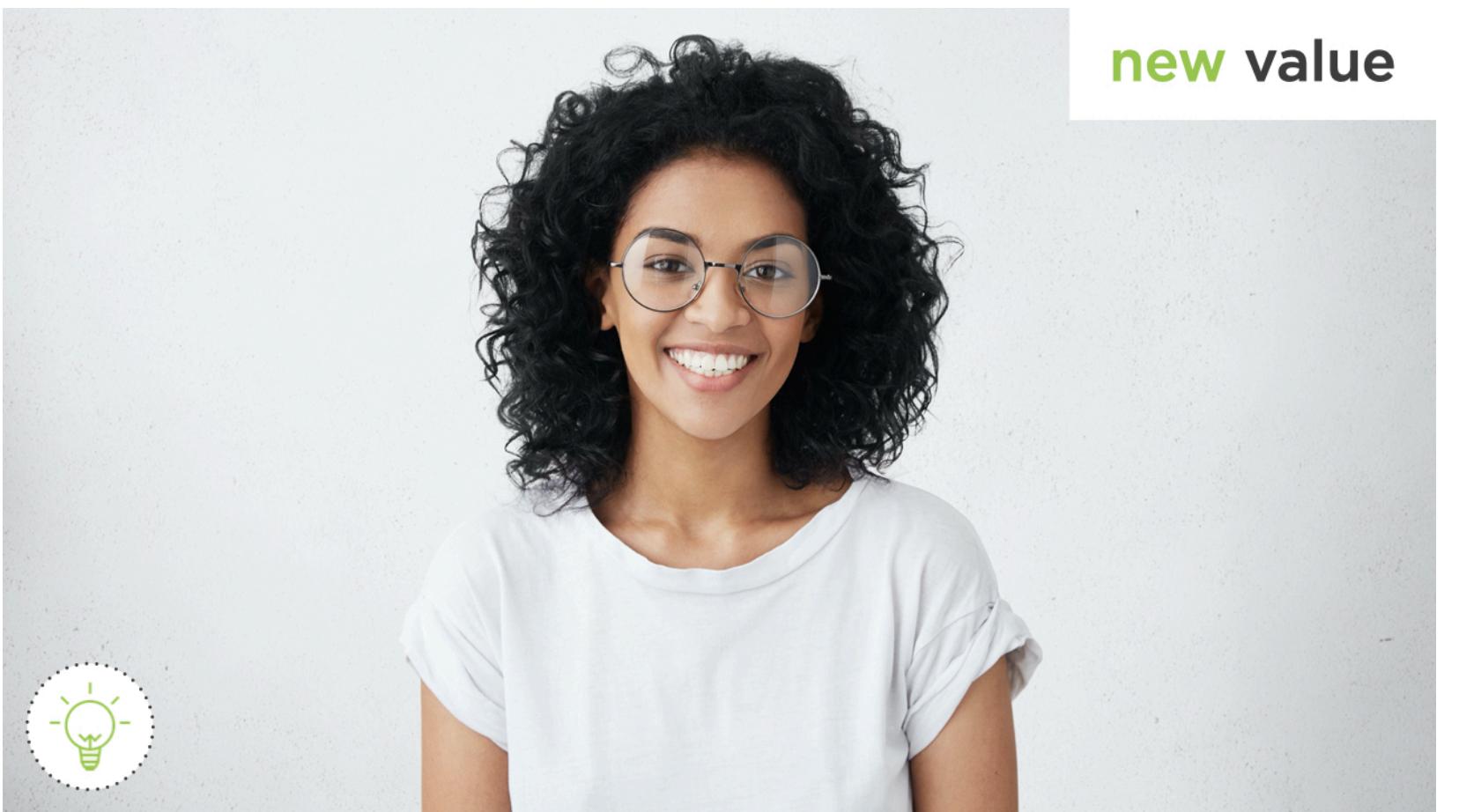


value

Instances of structs are
“immutable” in the same exact
way as that photo was.



value

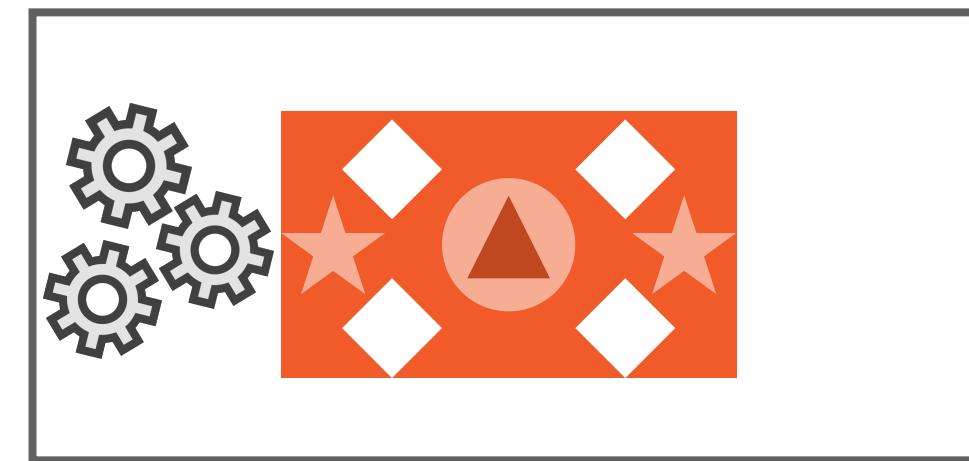


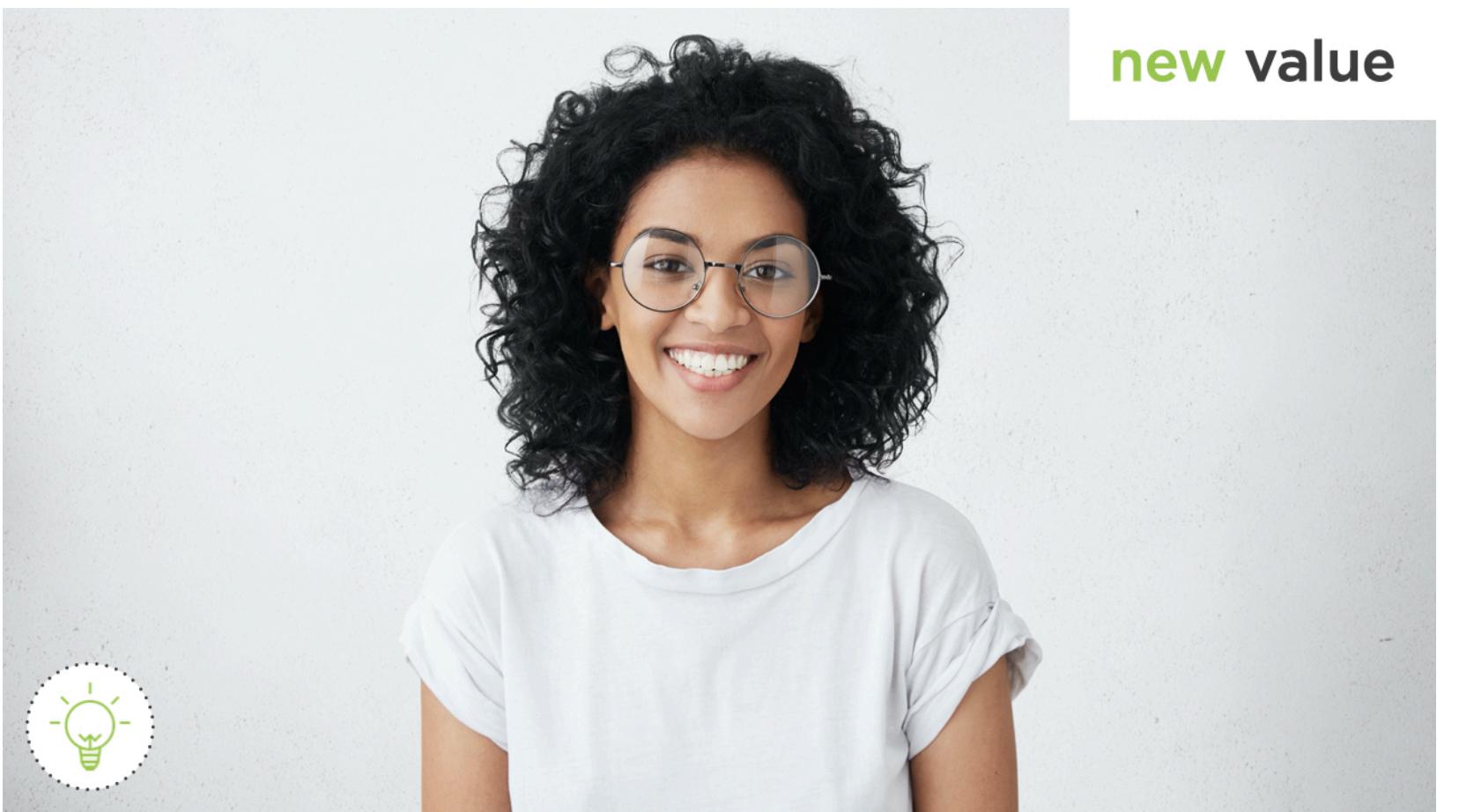
new value

Input Struct Instance

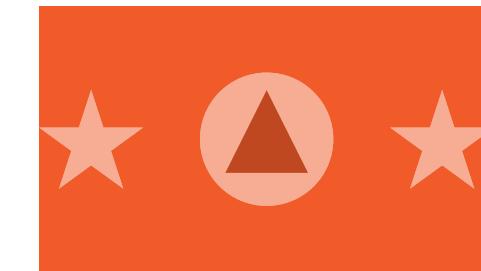


value

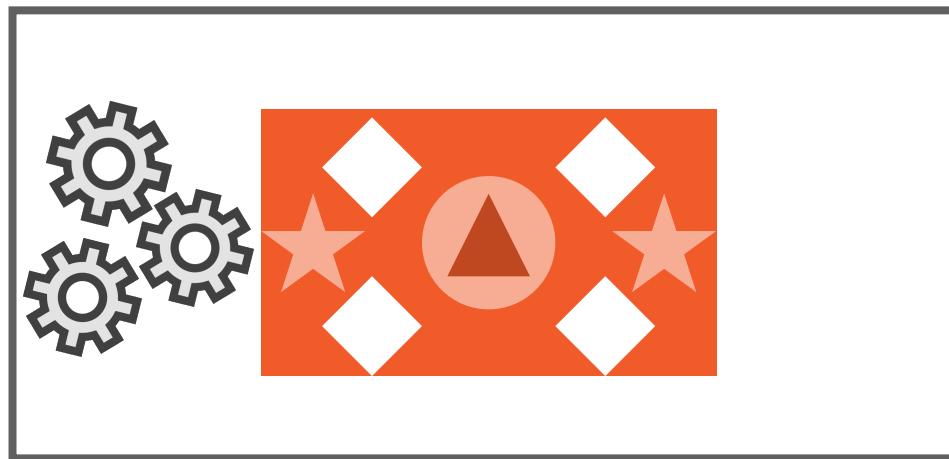




Input Struct Instance

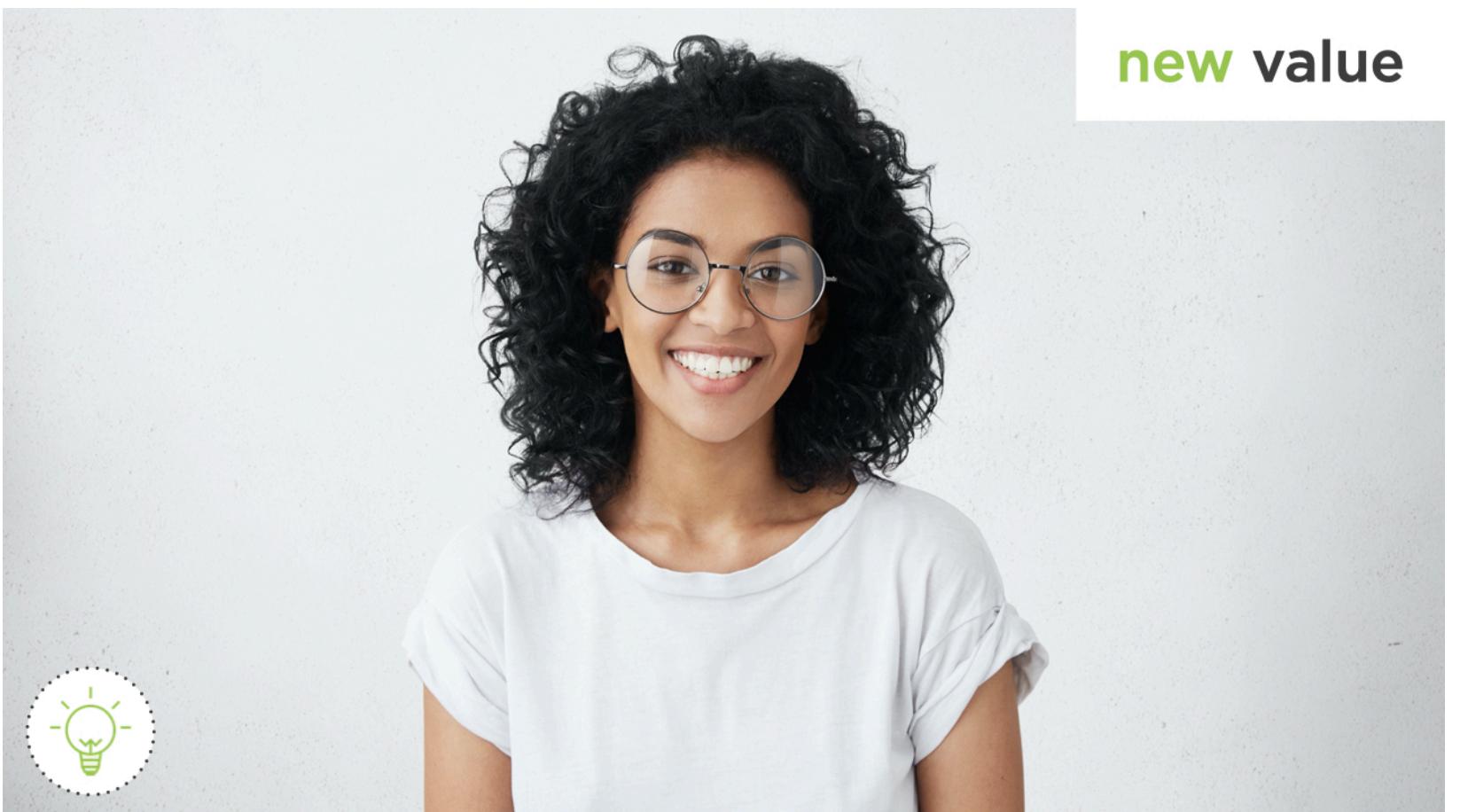


value





value



new value

Input Struct Instance



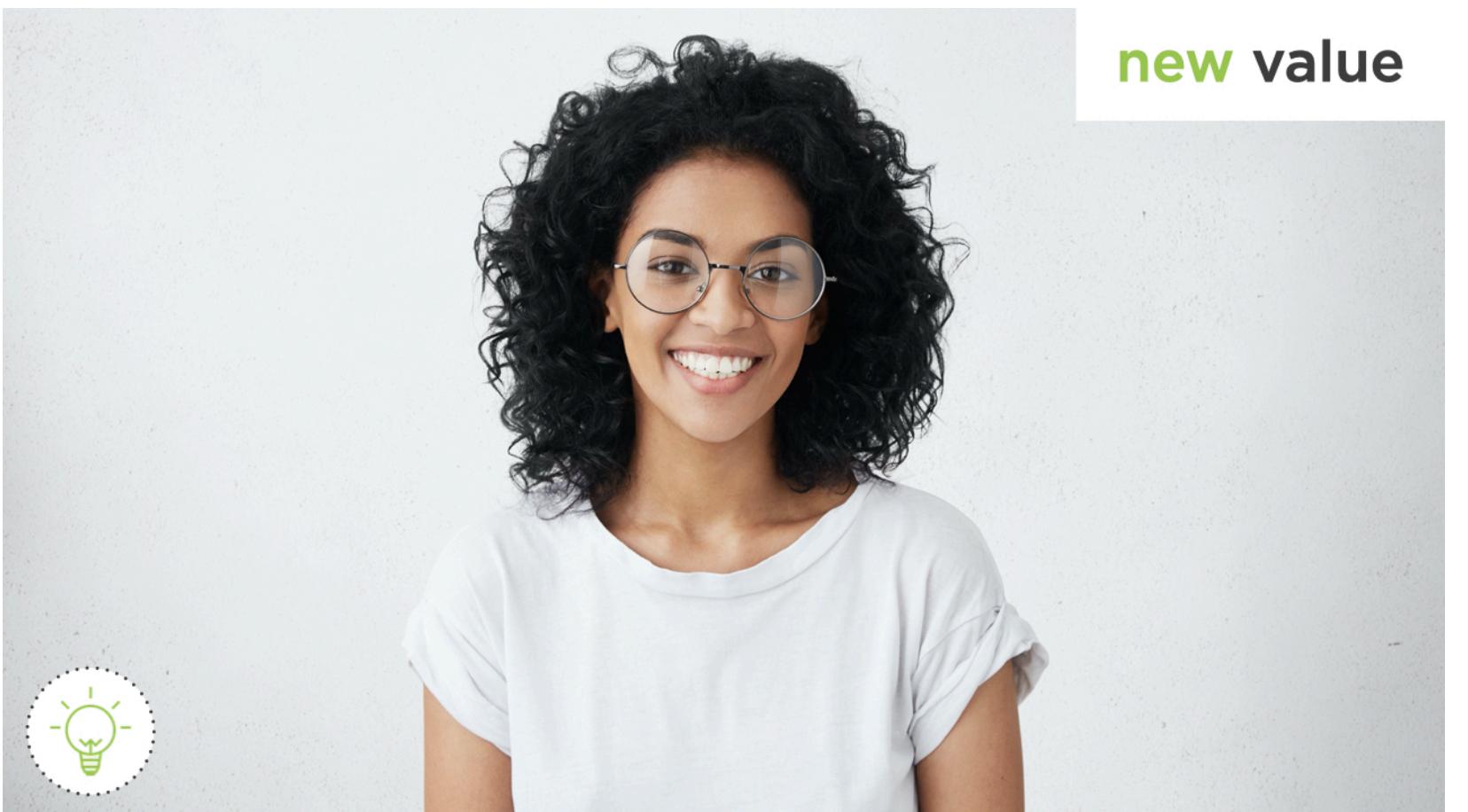
value



Output Struct Instance



new value



value

Input Struct Instance



Function



transformation

Output Struct Instance



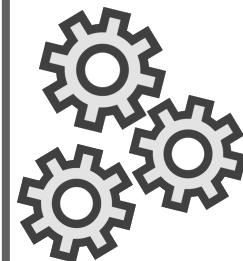
new value

Input Struct Instance



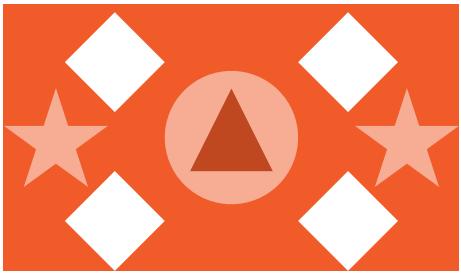
value

Function



transformation

Output Struct Instance



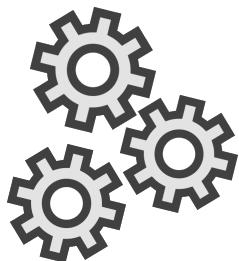
new value

Input View Instance



value

View Modifier



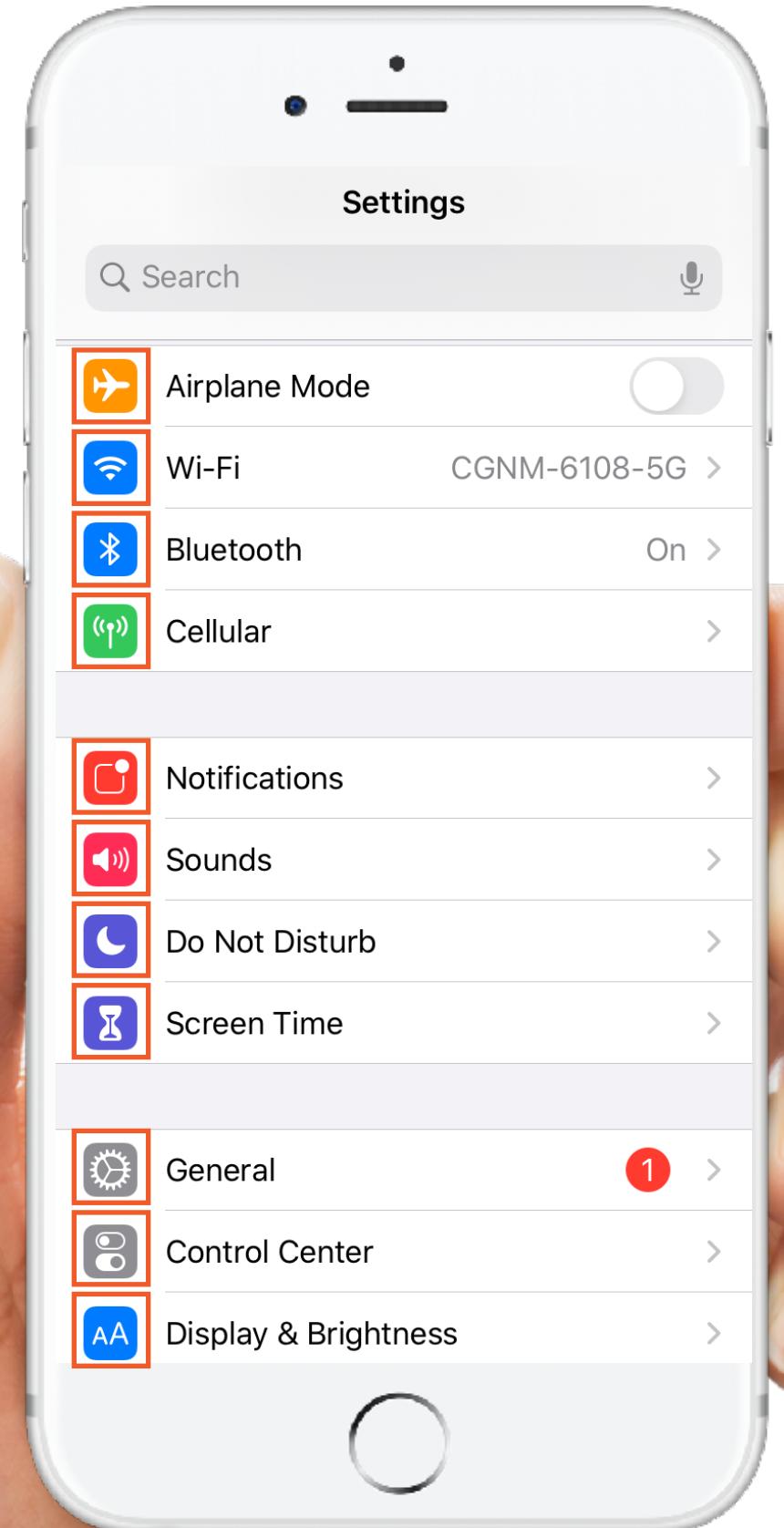
transformation

Output View Instance

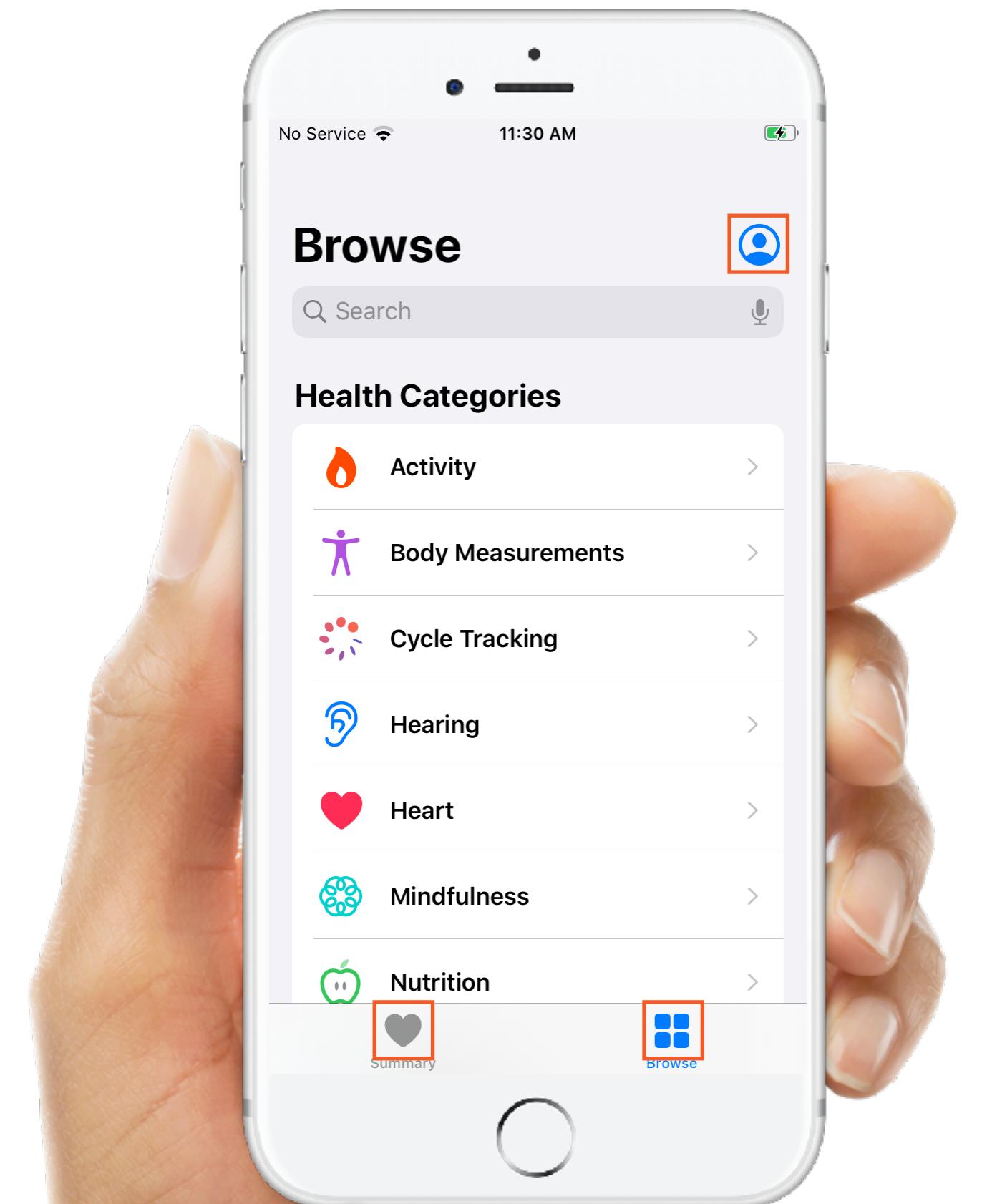


new value

Using Icons and Symbols







Those days are **over**.

Managing Views with Groups

NightWatch > iPhone 11 NightWatch | Build NightWatch: Failed | Today at 5:15 PM × 1

ContentView.swift

NightWatch | ContentView.swift | body

```
REAL DOCUMENT \ strange and unusual occurrences")
HStack {
    Text(Image(systemName: "sunset"))
        .foregroundColor(.yellow)
        .font(.title3)
        .fontWeight(.heavy)
    Text("Weekly Tasks")
        .underline()
        .font(.title3)
        .fontWeight(.heavy)
        .foregroundColor(.yellow)
        .padding(.top)
        .textCase(.uppercase)
}

Text("Check inside all vacant rooms")
HStack {
    Text(Image(systemName: "calendar"))
        .foregroundColor(.yellow)
        .font(.title3)
        .fontWeight(.heavy)
    Text("Monthly Tasks")
        .underline()
        .font(.title3)
        .fontWeight(.heavy)
}
```

Failed to... Try Again Diagnostics

Preview

NIGHTLY TASKS

- Check all windows
- Check all doors
- Check that the safe is locked
- Check the mailbox
- Inspect security cameras
- Clear ice from sidewalks
- Document "strange and unusual" occurrences

WEEKLY TASKS

MONTHLY TASKS

A screenshot of an iPhone displaying a user interface. The interface includes sections for 'NIGHTLY TASKS' (Check all windows, Check all doors, Check that the safe is locked, Check the mailbox, Inspect security cameras, Clear ice from sidewalks, Document "strange and unusual" occurrences), 'WEEKLY TASKS' (Check inside all vacant rooms), and 'MONTHLY TASKS'. A magnifying glass icon highlights the error message 'Extra argument in call' in the code editor.

+ Filter

Text 204.5x40

50%

NightWatch > iPhone 11 NightWatch | Build NightWatch: Failed | Today at 5:15 PM × 1

ContentView.swift

NightWatch | ContentView.swift | body

```
REAL DOCUMENT \ strange and unusual occurrences")
HStack {
    Text(Image(systemName: "sunset"))
        .foregroundColor(.yellow)
        .font(.title3)
        .fontWeight(.heavy)
    Text("Weekly Tasks")
        .underline()
        .font(.title3)
        .fontWeight(.heavy)
        .foregroundColor(.yellow)
        .padding(.top)
        .textCase(.uppercase)
}
Text("Check inside all vacant rooms")|10
HStack { ✖ Extra argument in call
    Text(Image(systemName: "calendar"))
        .foregroundColor(.yellow)
        .font(.title3)
        .fontWeight(.heavy)
    Text("Monthly Tasks")
        .underline()
        .font(.title3)
        .fontWeight(.heavy)
}
```

Failed to... Try Again Diagnostics

Preview

1 2 NIGHTLY TASKS
3 Check all windows
4 Check all doors
5 Check that the safe is locked
6 Check the mailbox
7 Inspect security cameras
8 Clear ice from sidewalks
9 Document "strange and unusual" occurrences

WEEKLY TASKS

MONTHLY TASKS

Filter

Text 204.5x40 50%

Creating Data-driven Lists



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

List Views are a
core competency
for an iOS developer
working in SwiftUI.

Overview

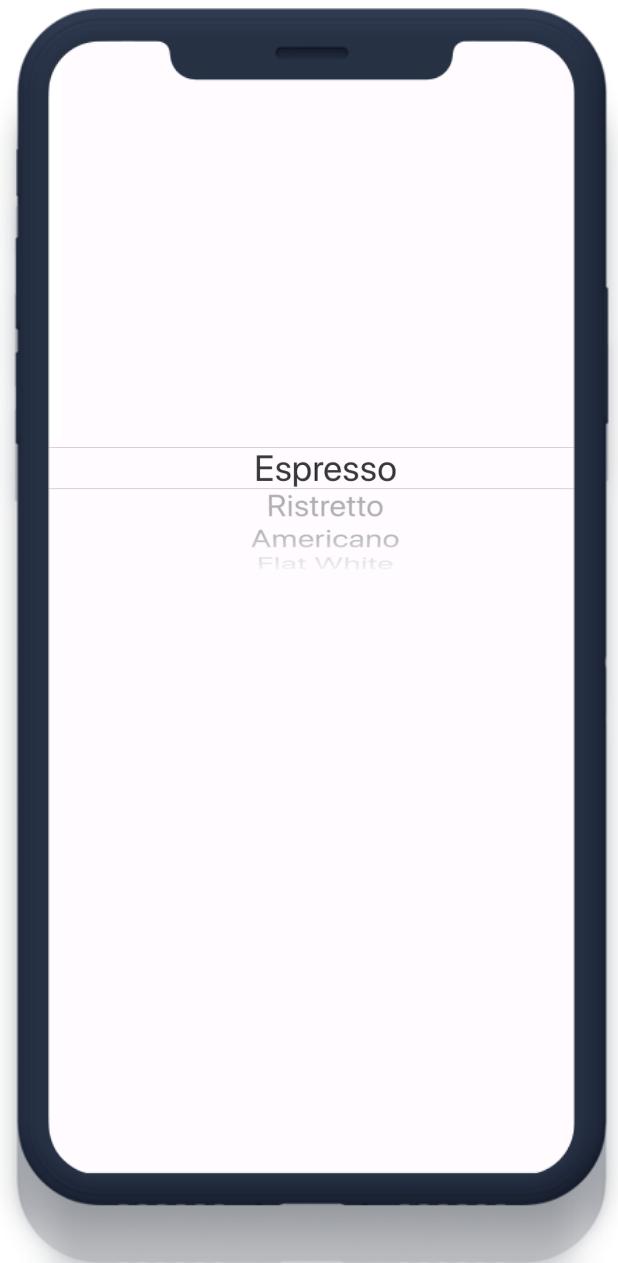
Add List Views to our apps

Configure a List

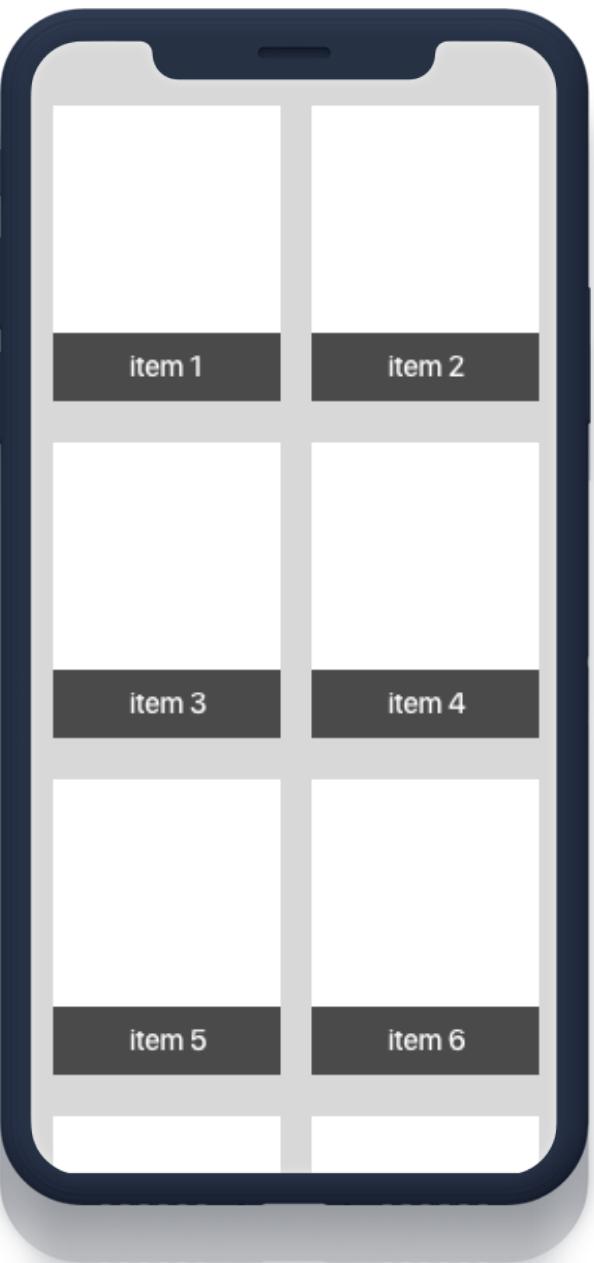
Provide Lists with data

**Data modeling and data management in
SwiftUI**

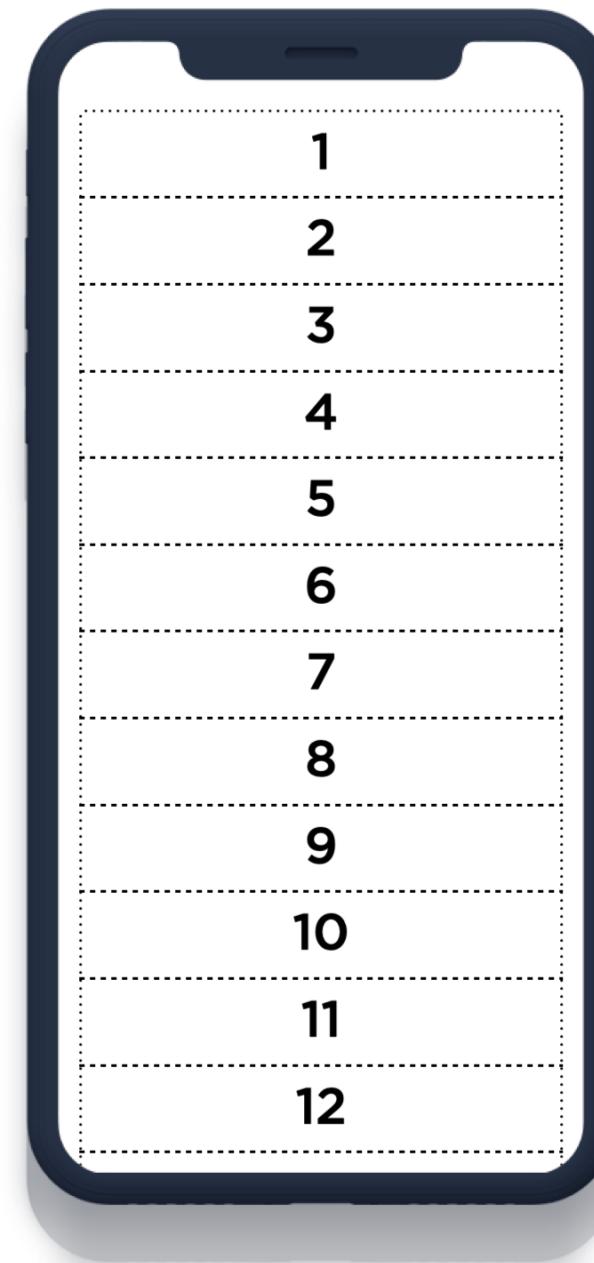
Picker View



Grid View

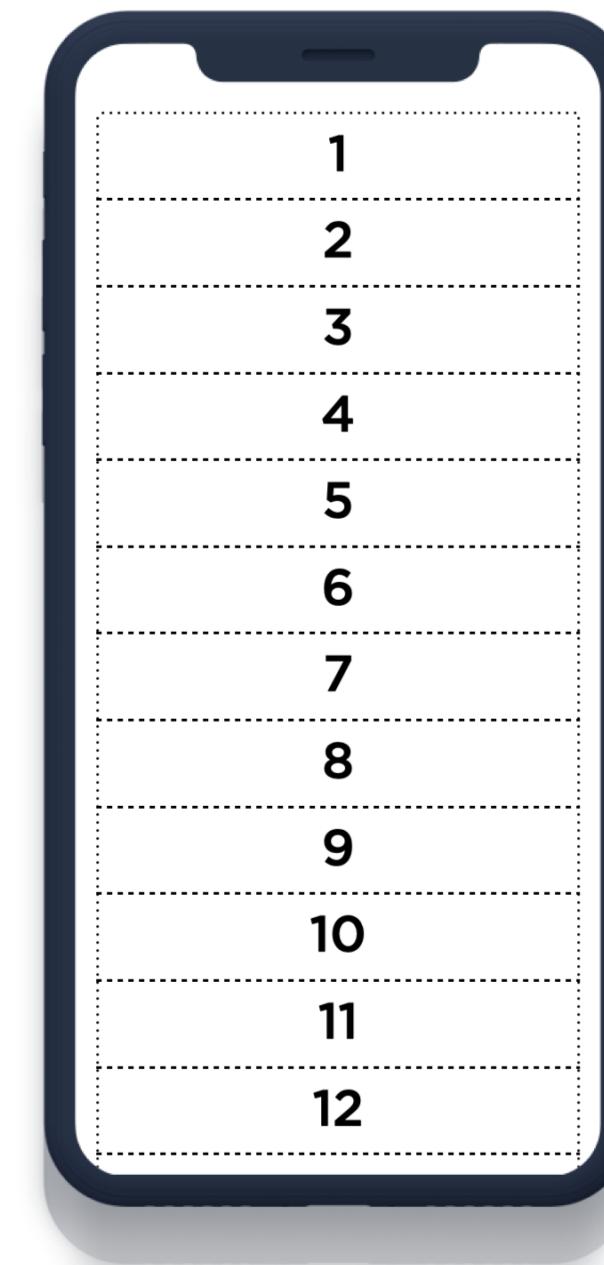


List View



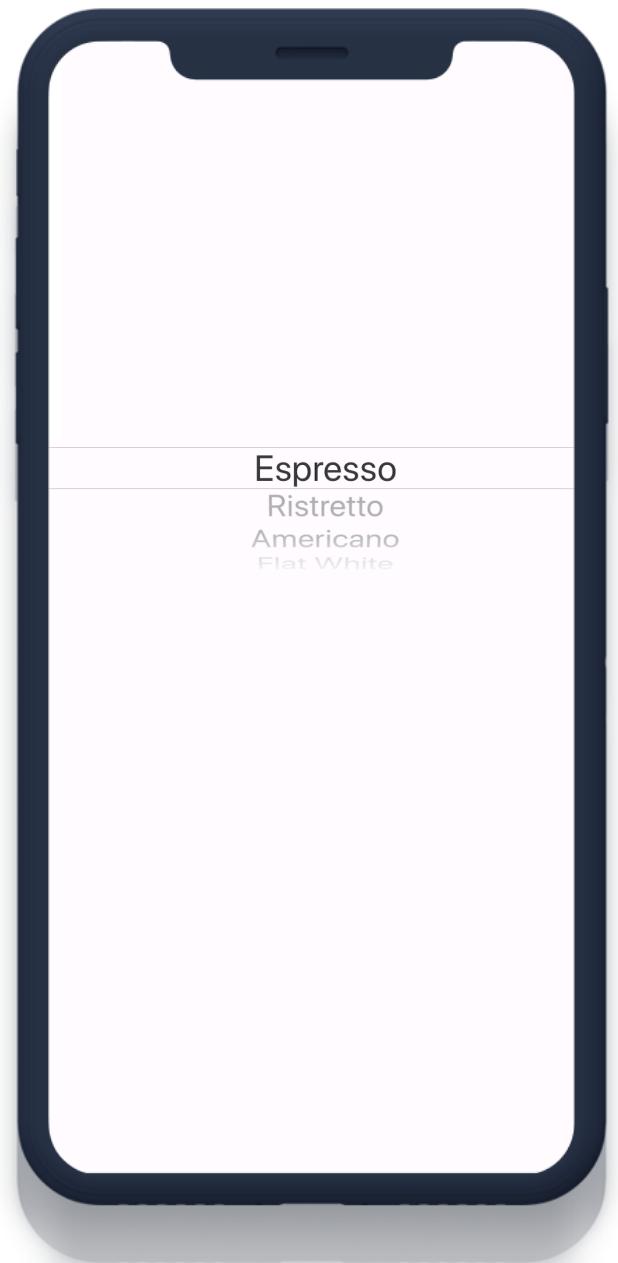


List View



Understanding Lists

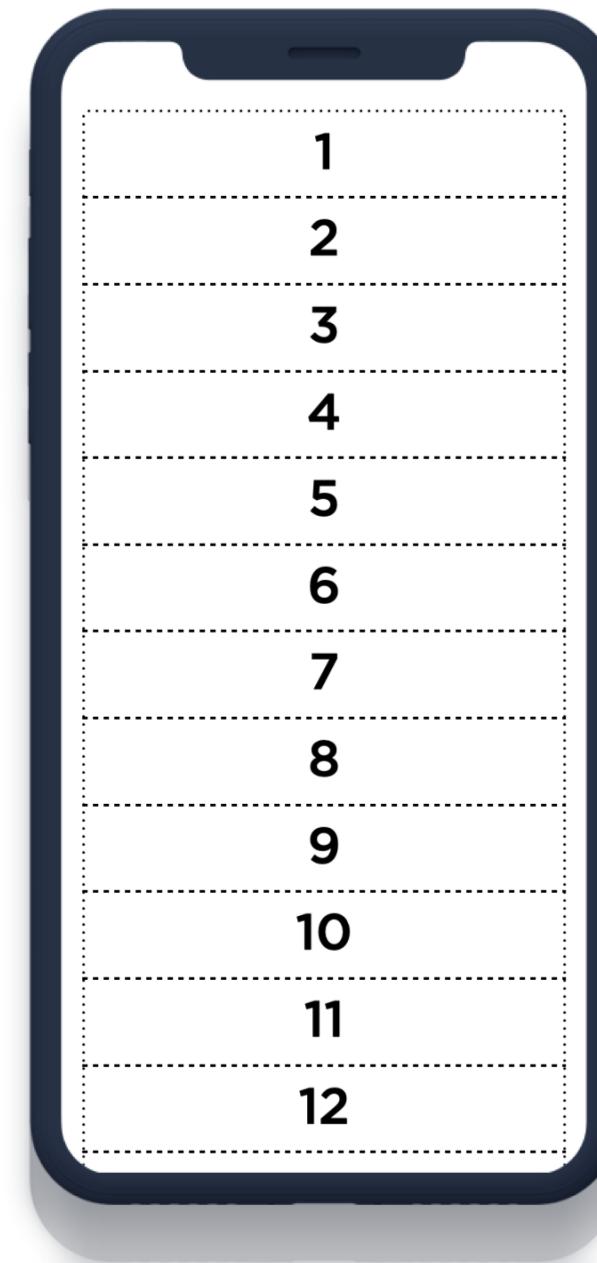
Picker View



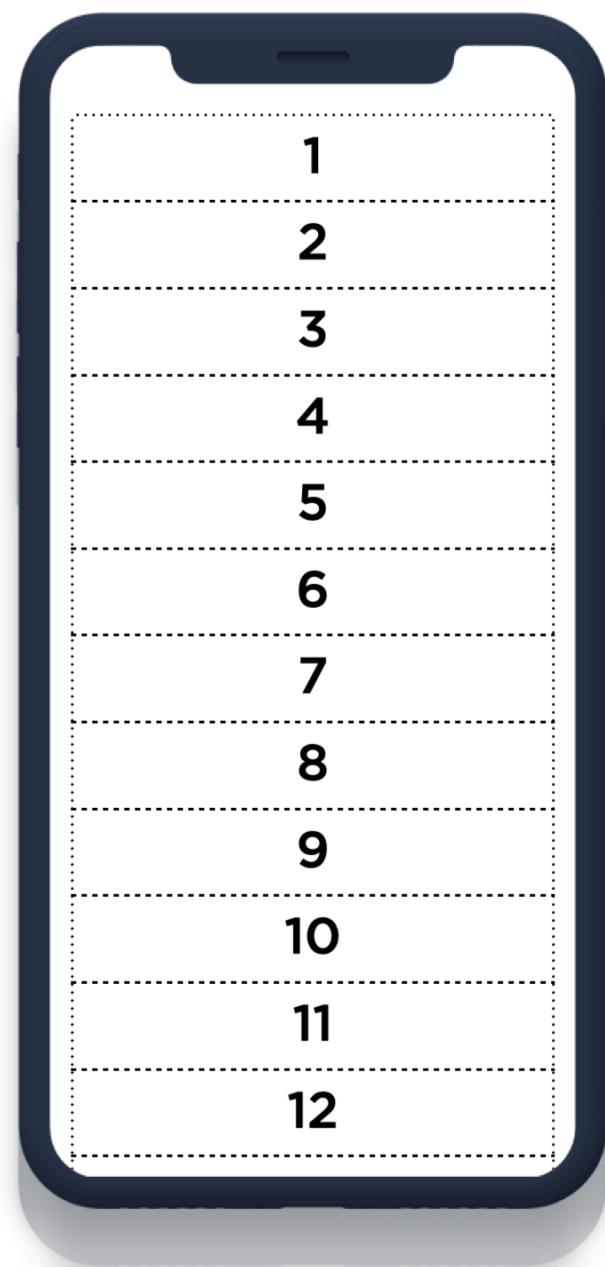
Grid View



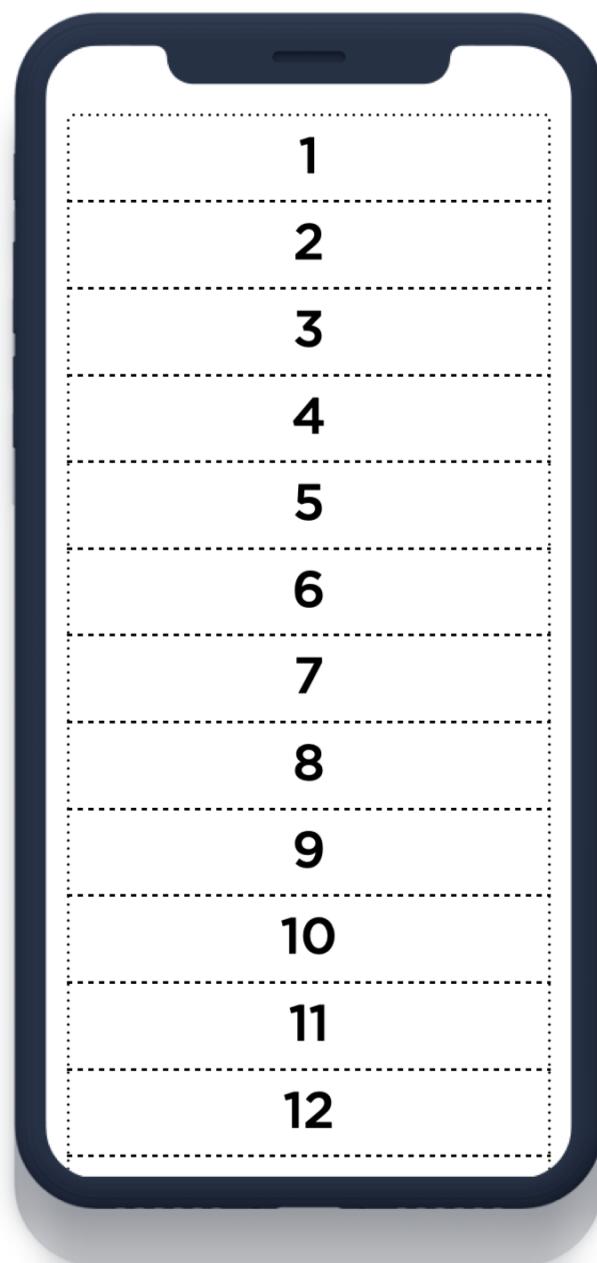
List View



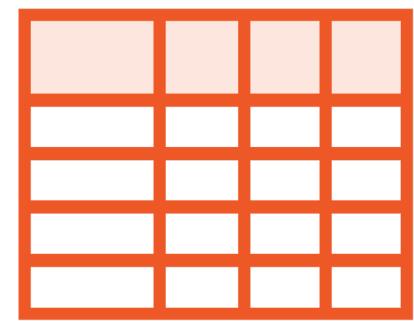
List View

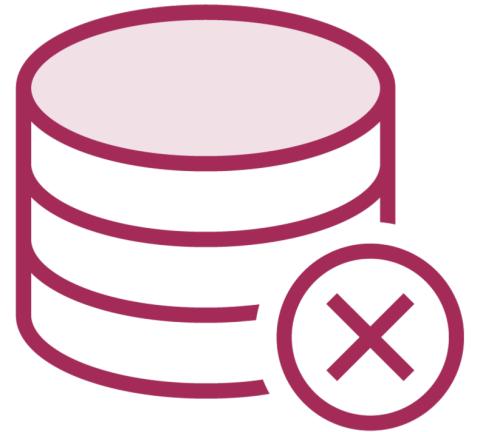


List View

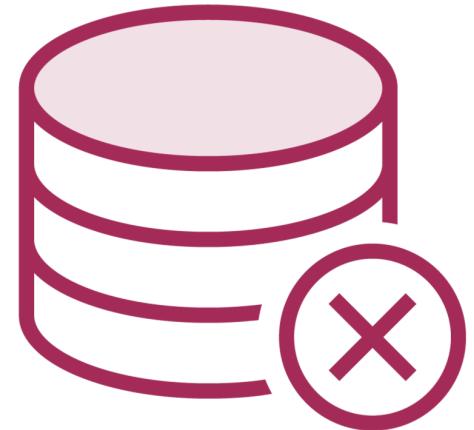


Helps us show a structured list of items.

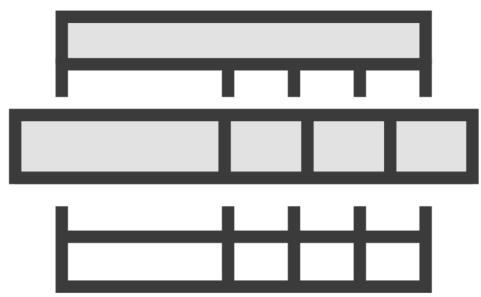




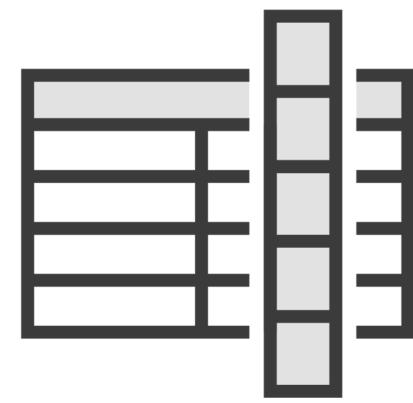
Lists are not storage containers.



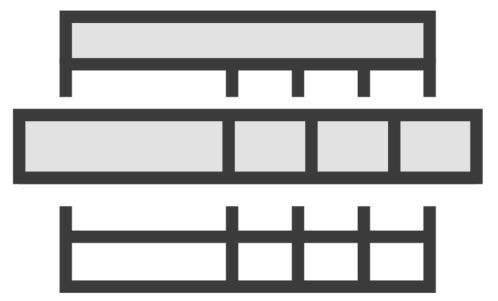
Lists are Views – they display data and facilitate interaction with it.



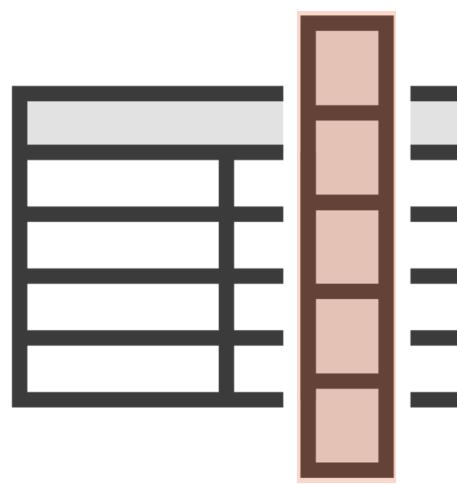
rows



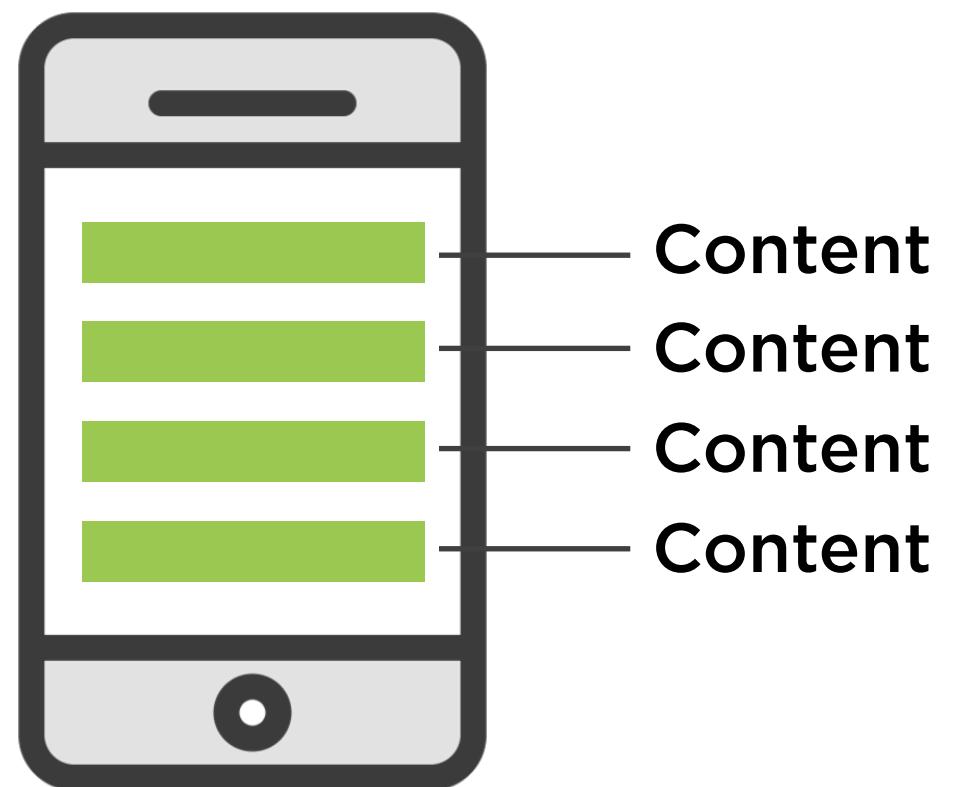
columns



rows



one column

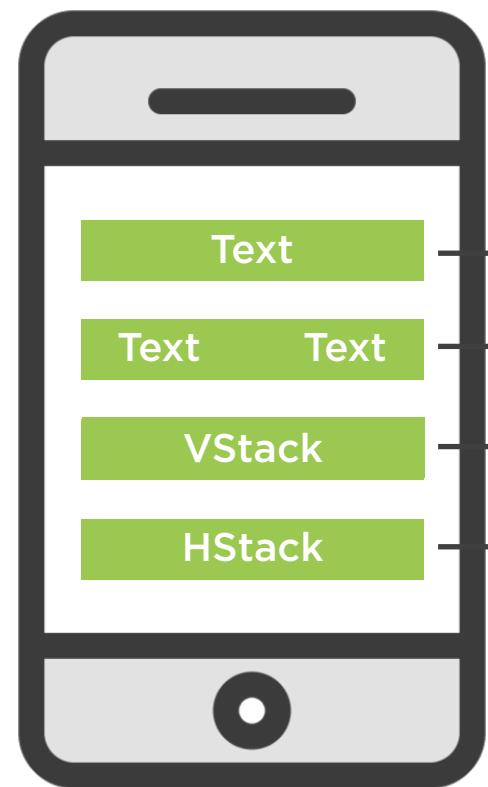


Content

Content

Content

Content

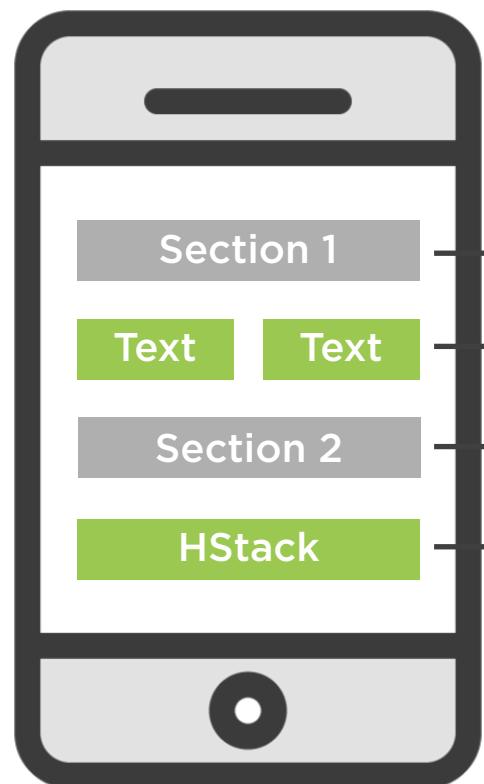


Content
Content
Content
Content



View Protocol

- ✓ A **computed property...**
- ✓ named **body...**
- ✓ that returns **some View**



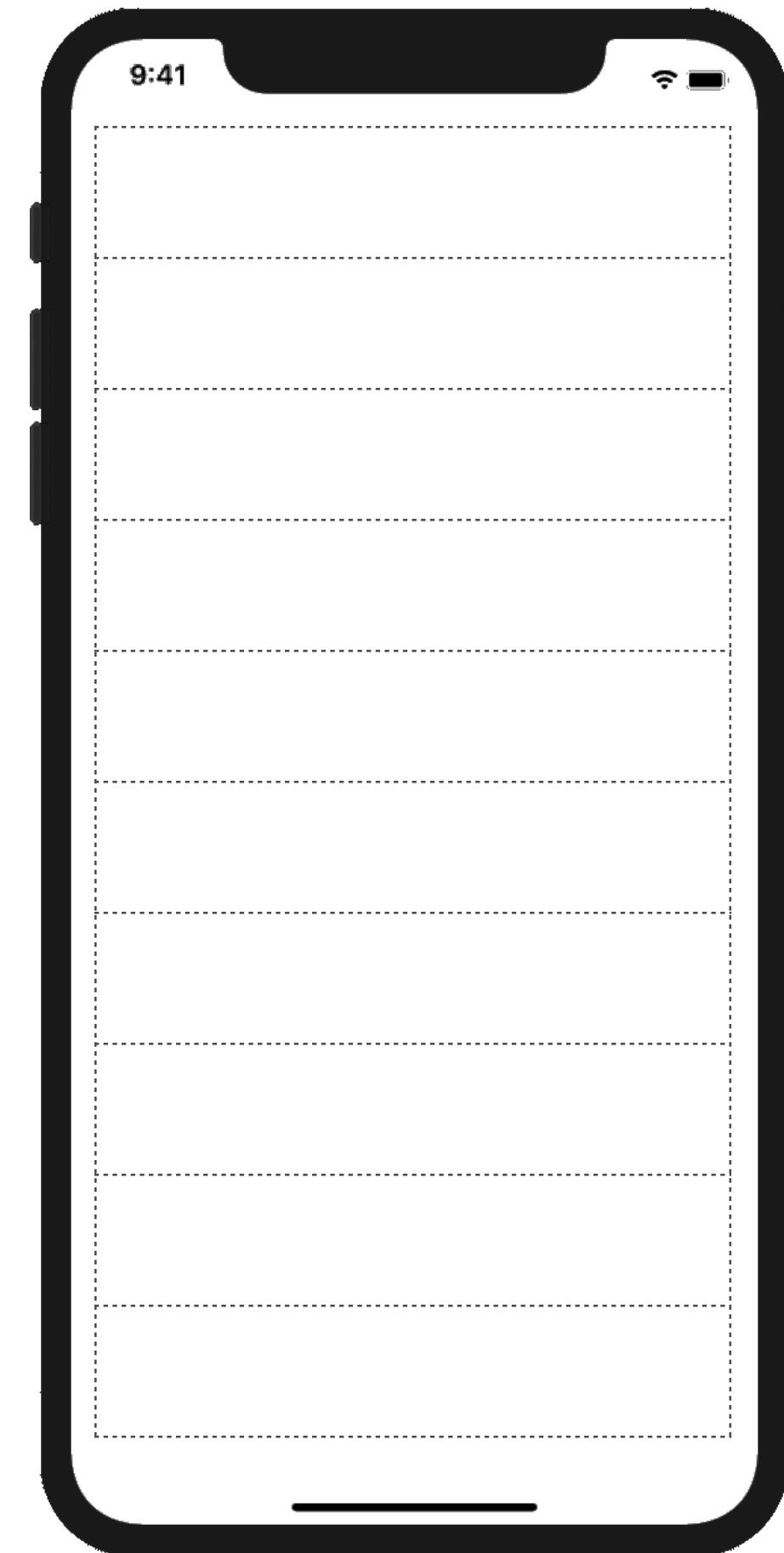
Content
Content
Content
Content



- ✓ A **computed property...**
- ✓ named **body...**
- ✓ that returns **some View**

Creating a List

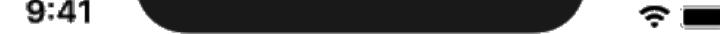
Controlling a List with Data



9:41



9:41



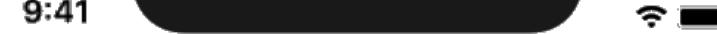
9:41

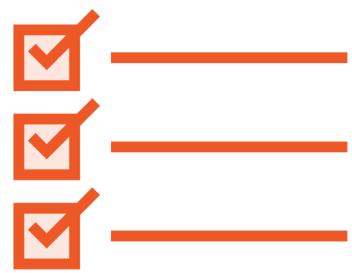


9:41

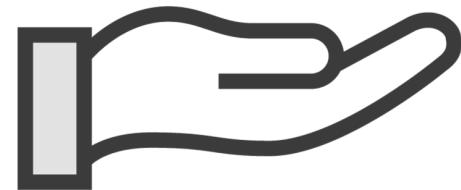


9:41

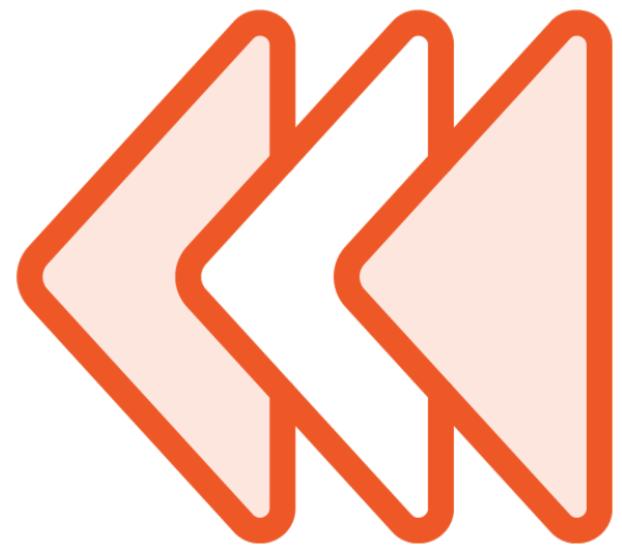




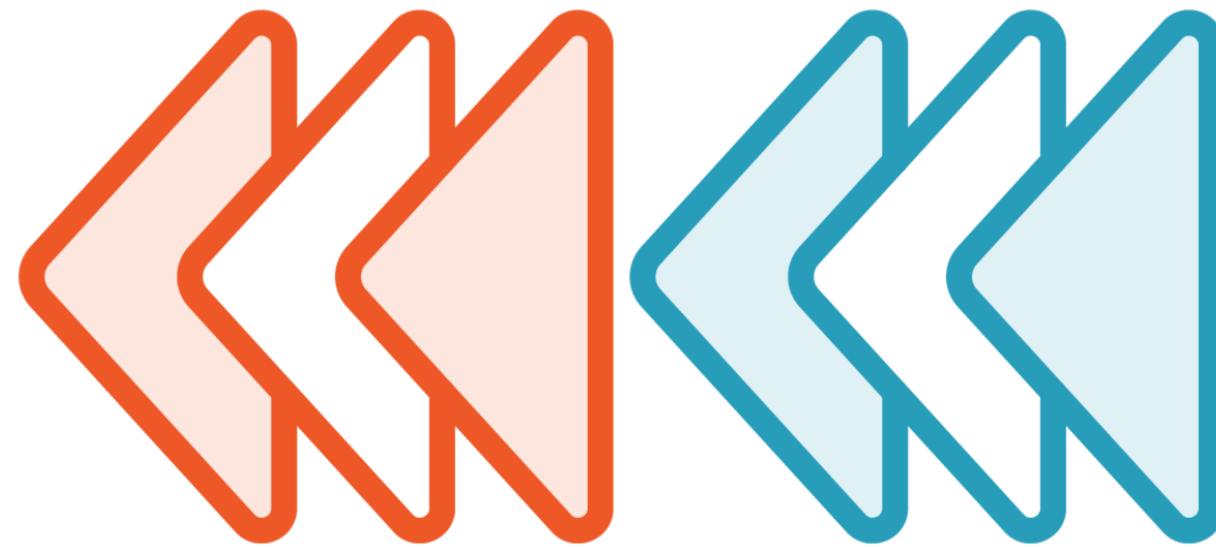
Data Source



Using ForEach to Supply List Content

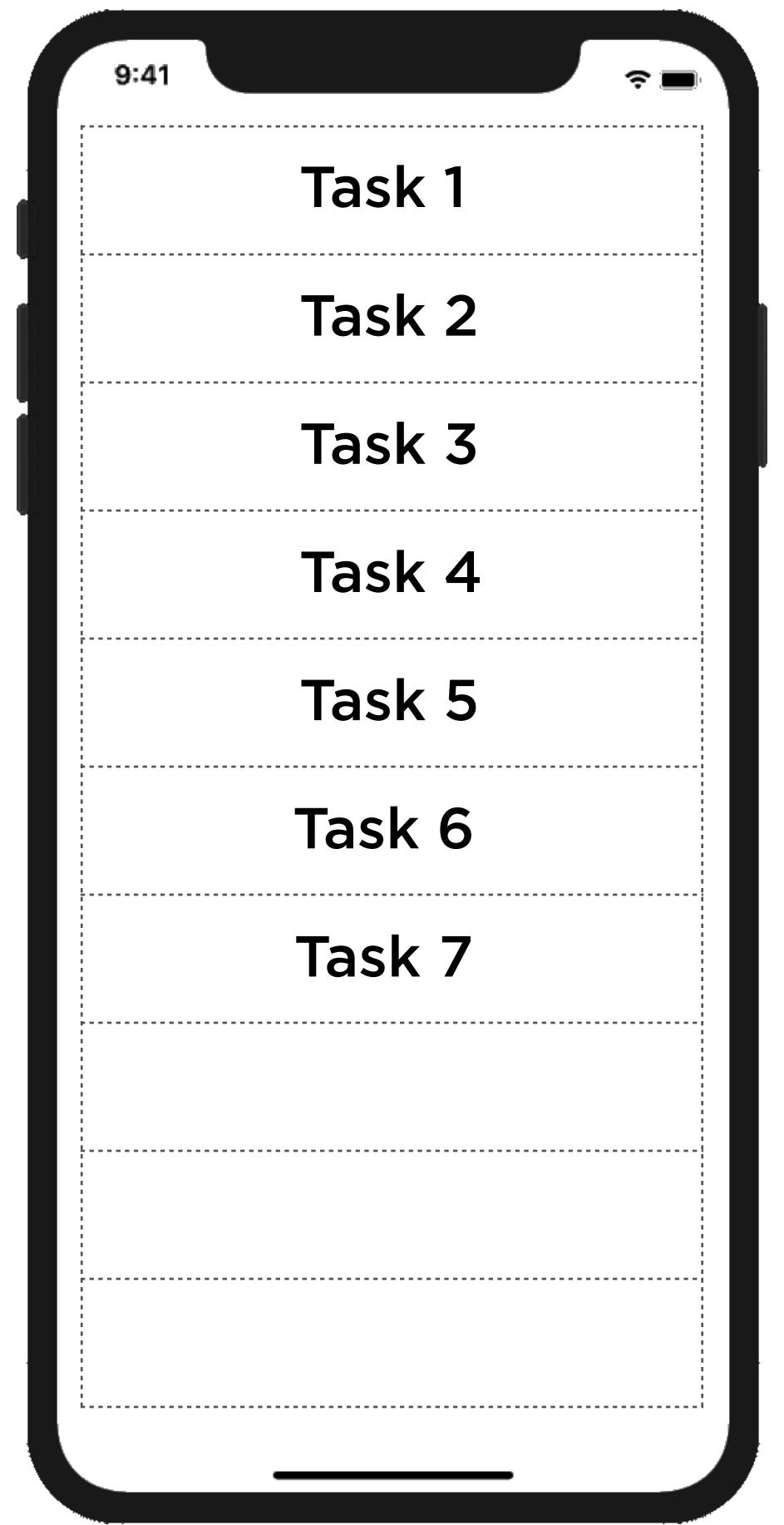


Flashback Moment



Another Flashback Moment

Adding Sections and Headings



Task 1

Task 2

Task 3

Task 4

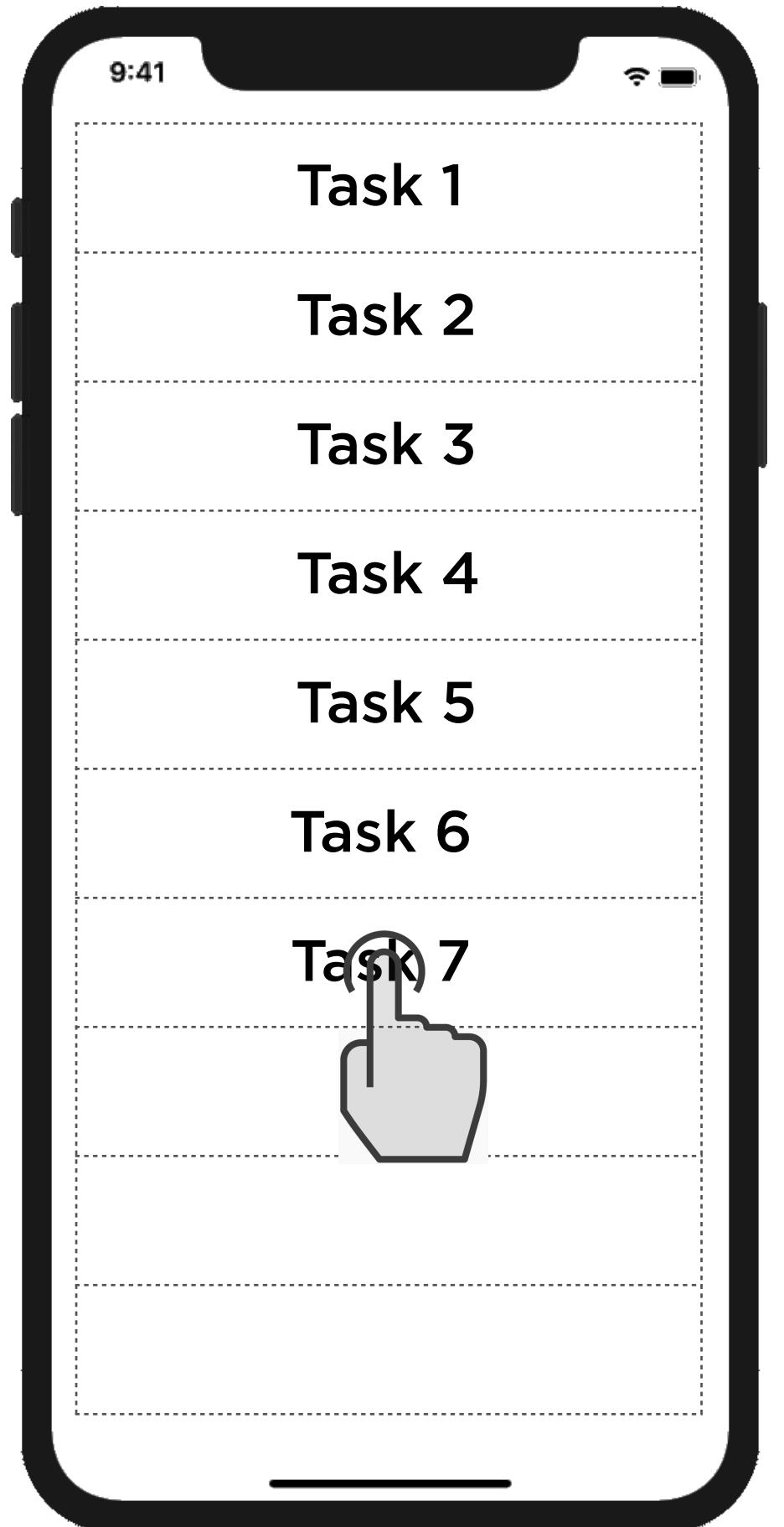
Task 5

Task 6

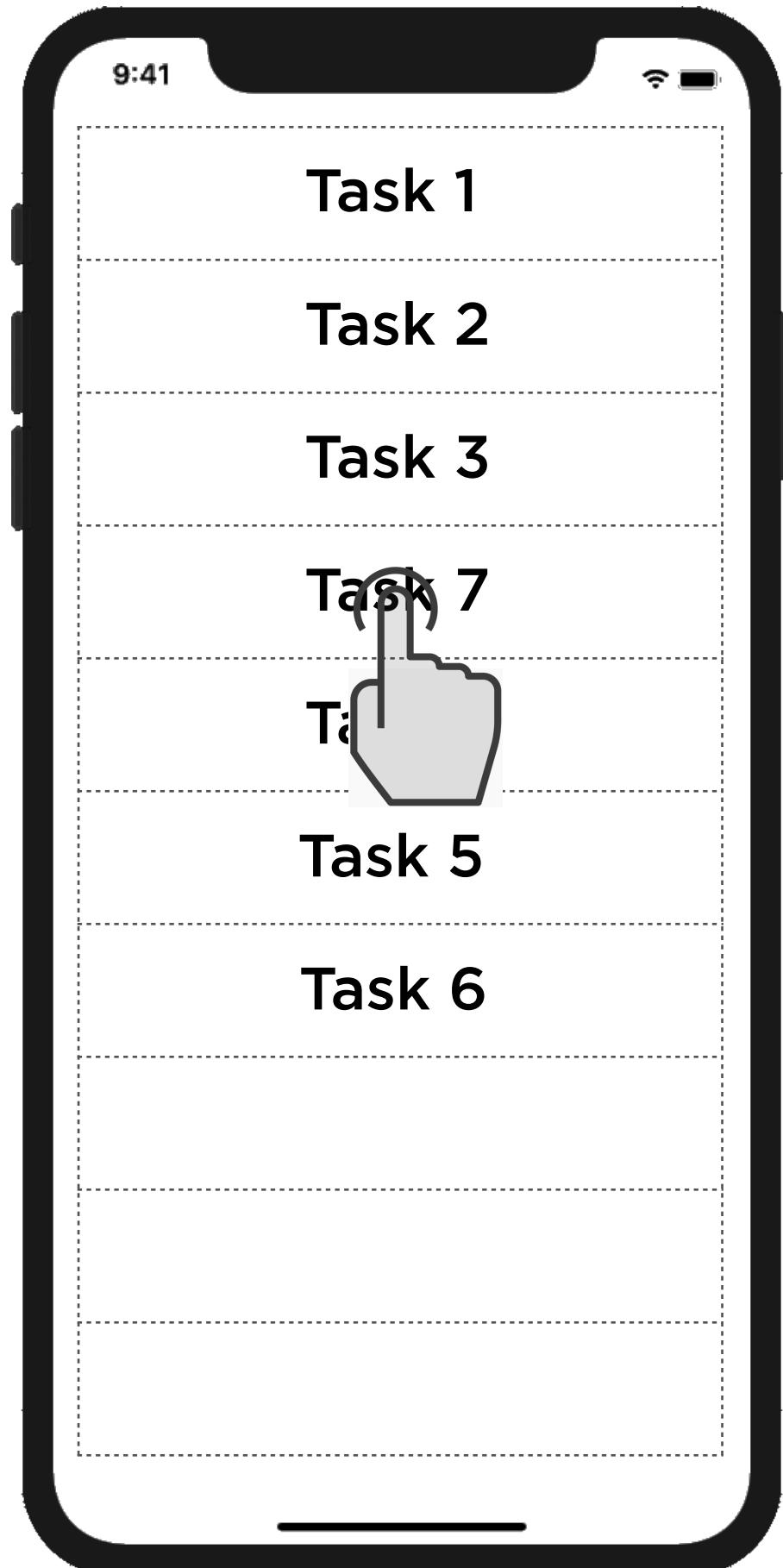
Task 7



Can we swipe to delete rows?



Can we swipe to delete rows?



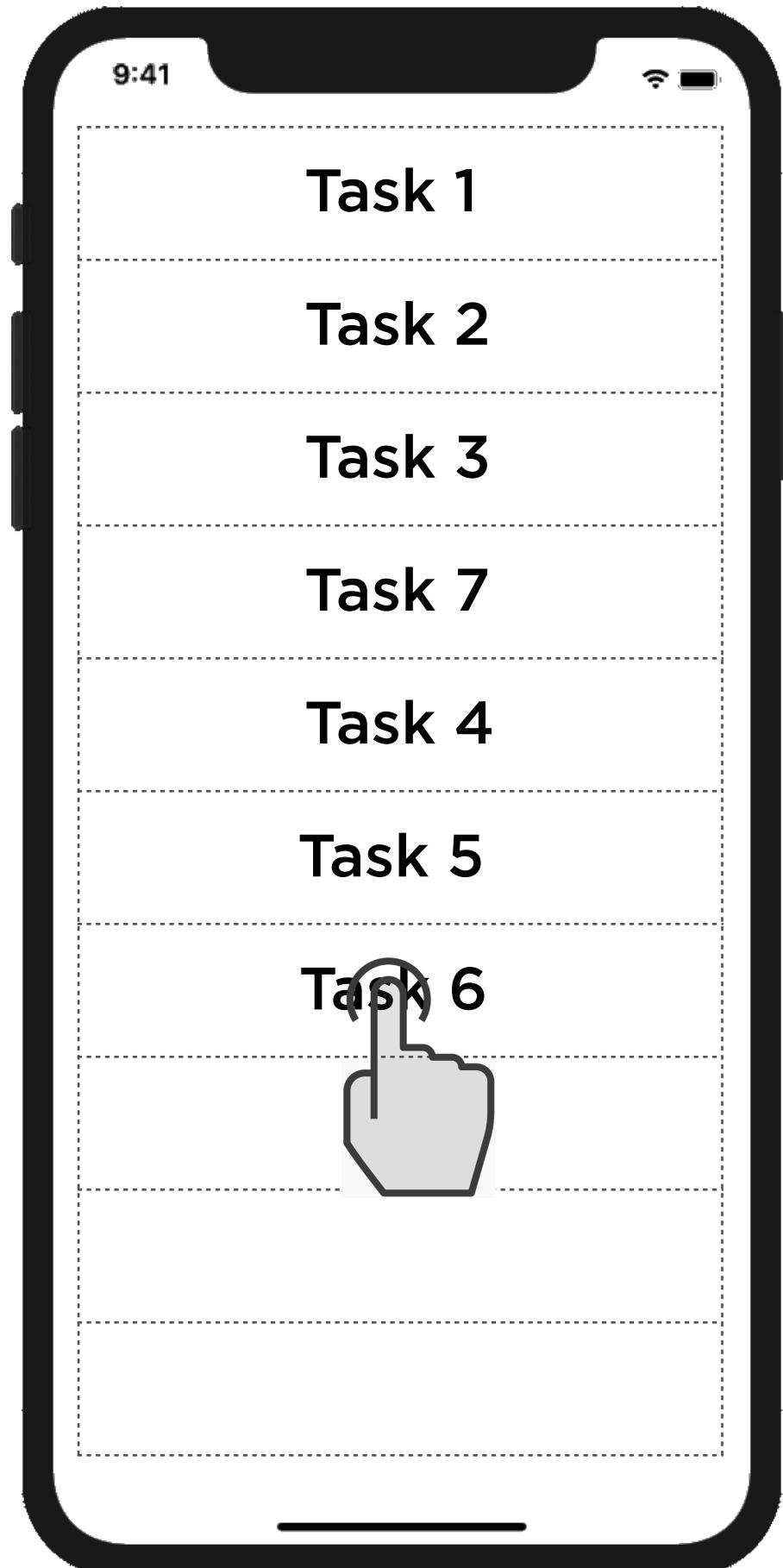
Can we swipe to delete rows?



Can we rearrange their order?



If we do delete something, what should happen to our data behind the scenes?



Can we swipe to delete rows?



Can we rearrange their order?



If we do delete something, what should happen to our data behind the scenes?



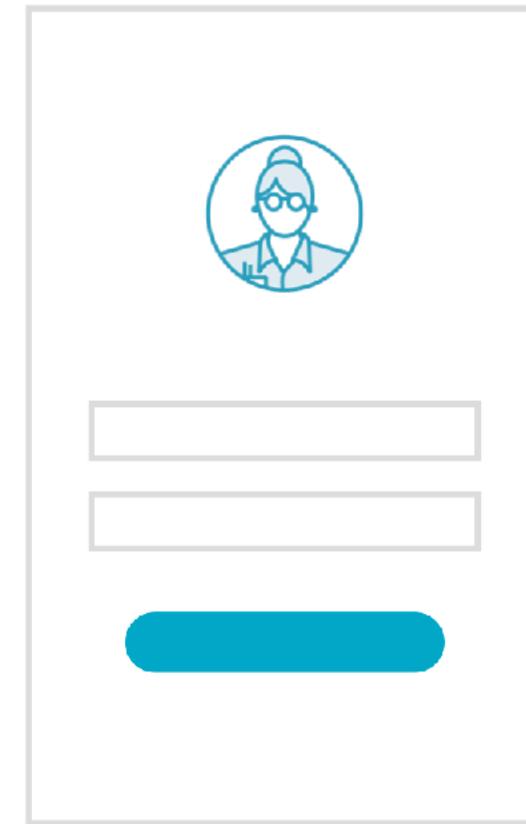
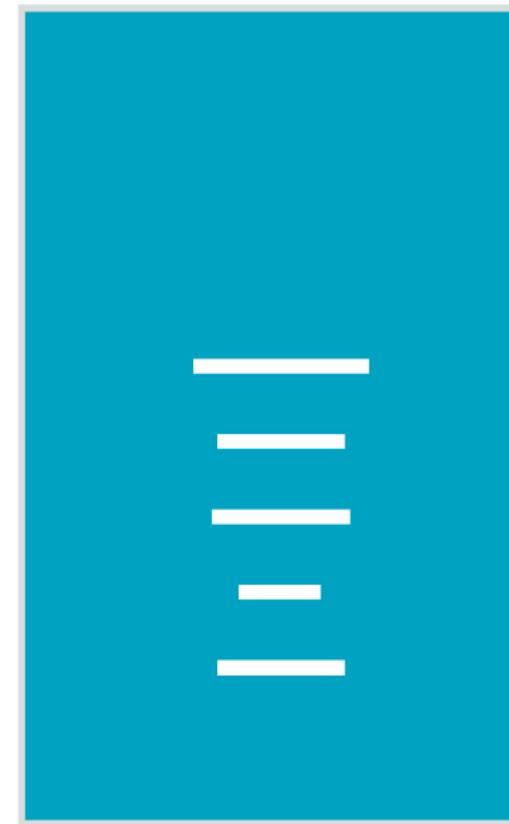
What happens when a row is tapped?

Building iOS Apps with Multiple Screens



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com



What do we actually add to the Xcode project to keep the right set of UI controls, logic, and data together for each screen?



How do we navigate from one section of our app
to another section... and back again?

Ask ourselves: “Why do we have multiple screens in this particular app?”



Ask ourselves: “Why do we have multiple screens in this particular app?”

Home

Details

Specifics

Distinctives

Drill-through Hierarchy

Library

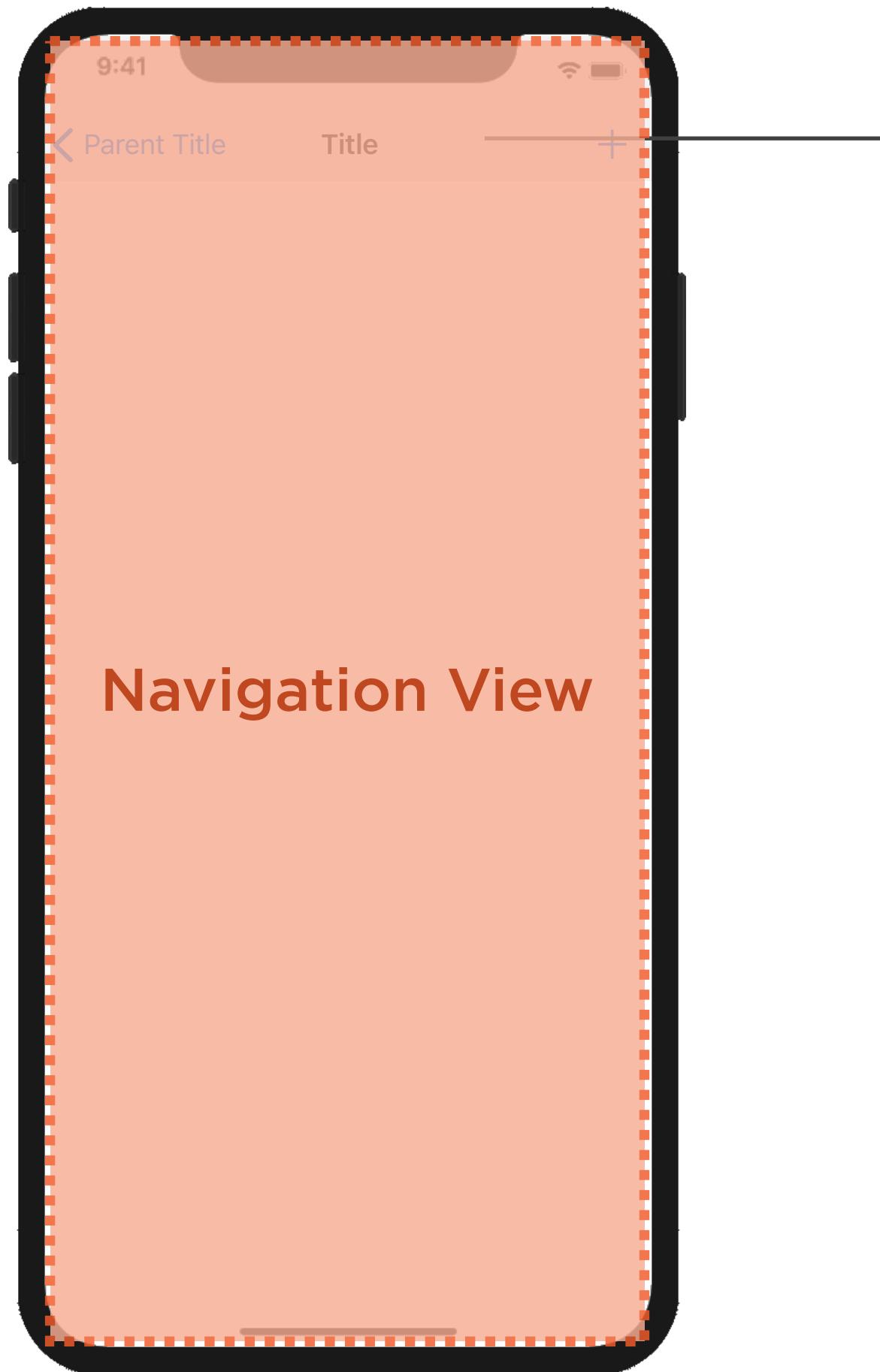
Favorites

For You

Browse

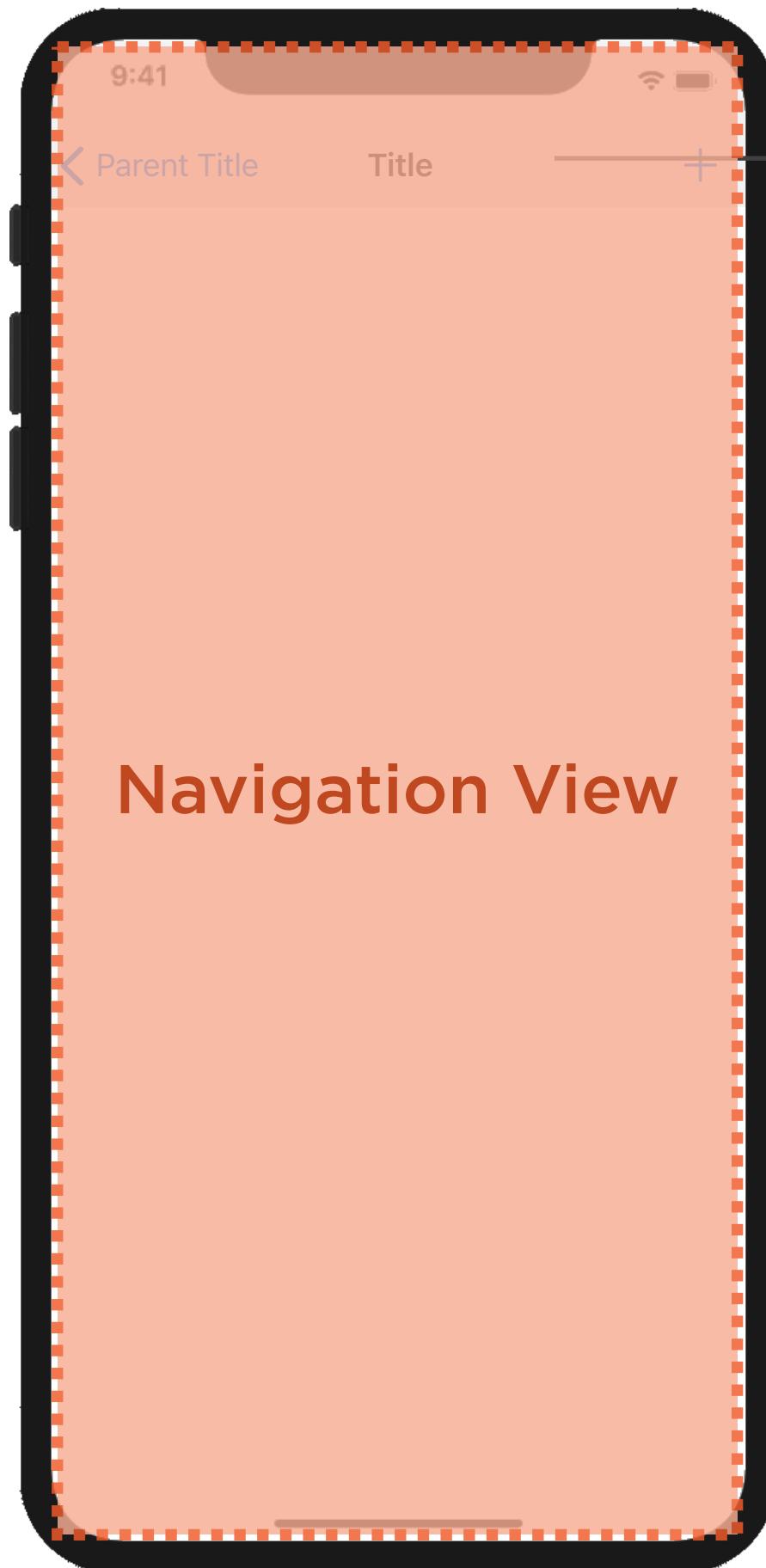
Search

Separate Sections



Navigation Bar

Navigation View



Navigation Bar

Home

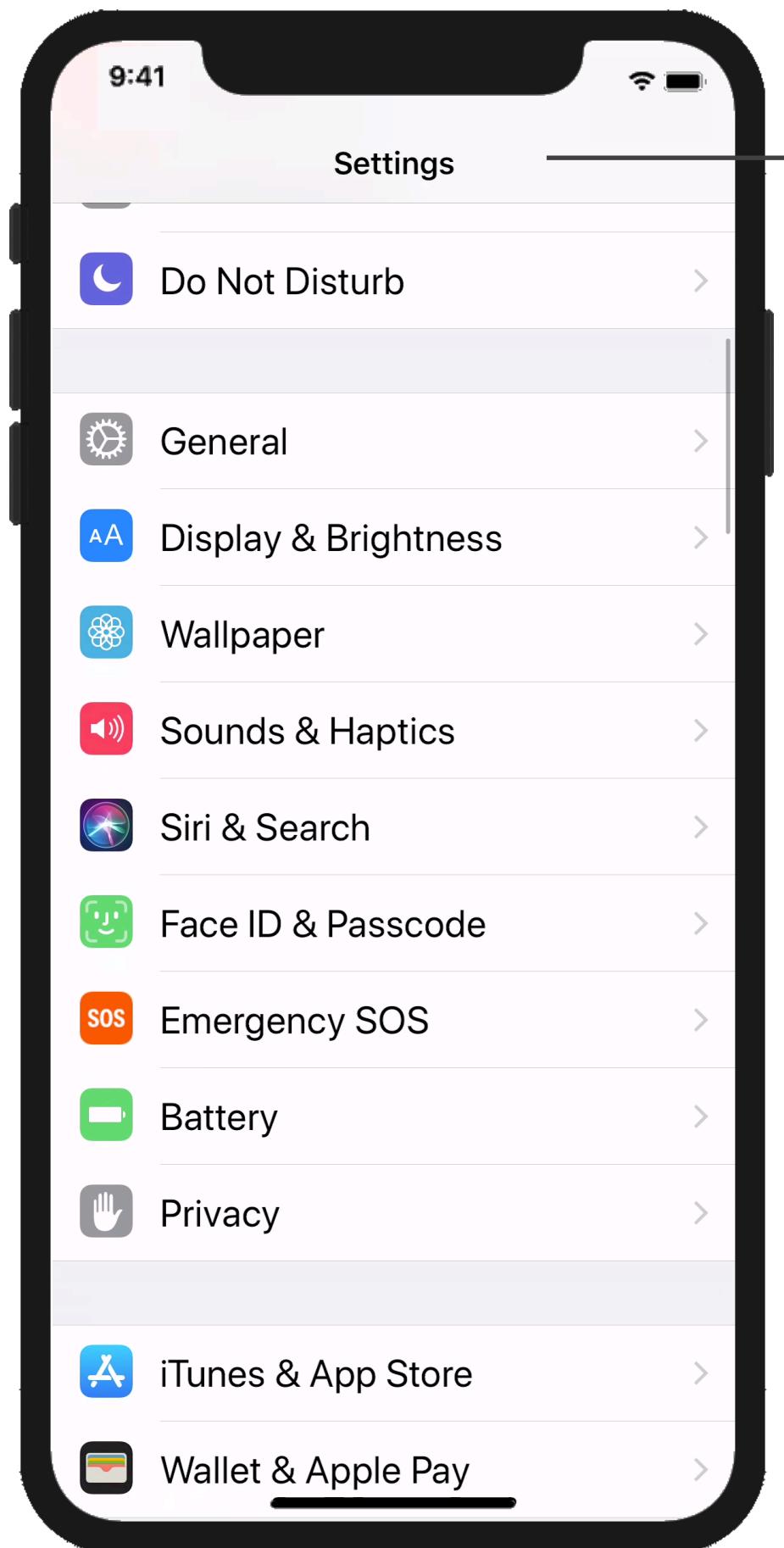
Details

Specifics

Distinctives

Drill-through Hierarchy

Navigation View



Navigation Bar

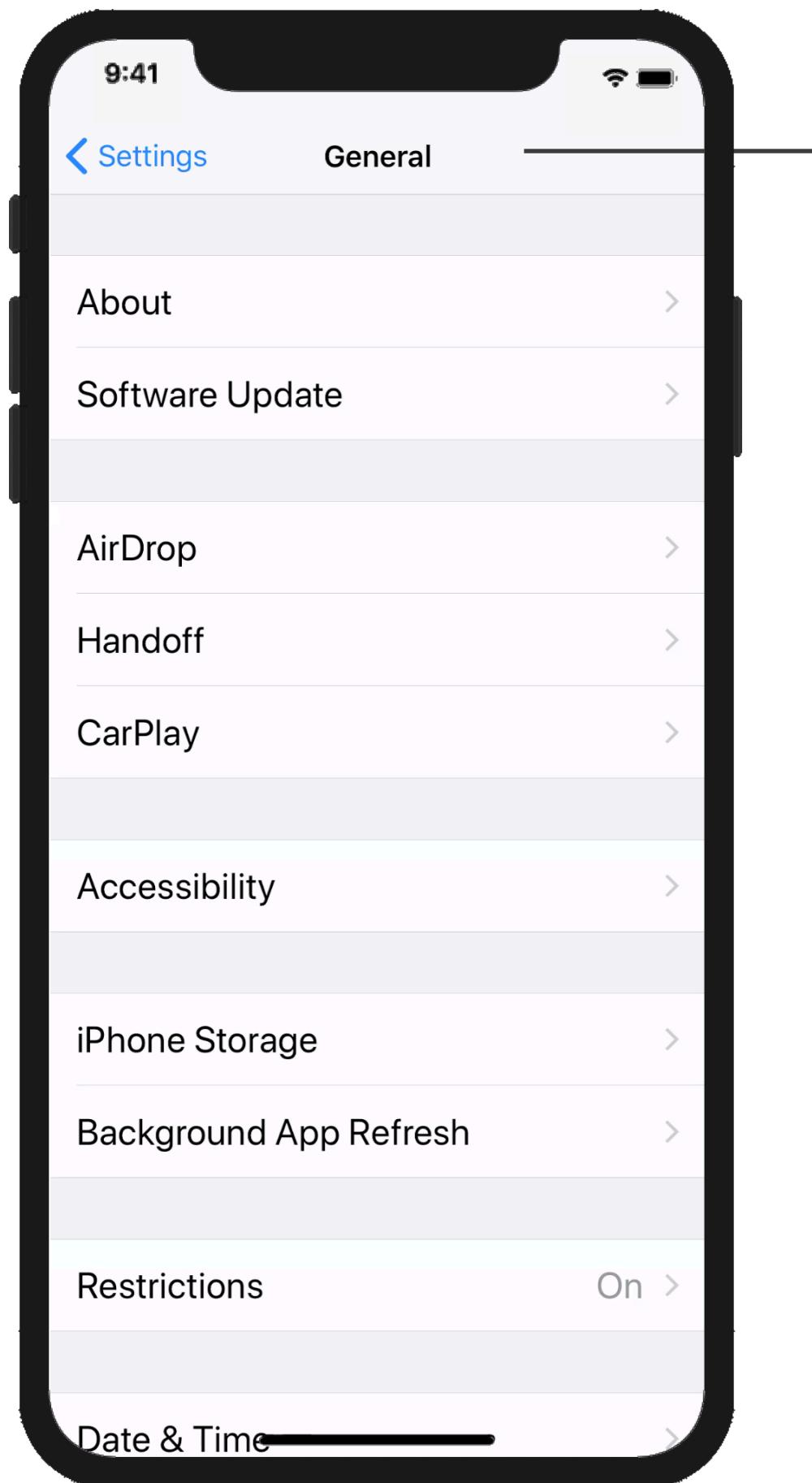
Home

Details

Specifics

Distinctives

Drill-through Hierarchy



Navigation Bar

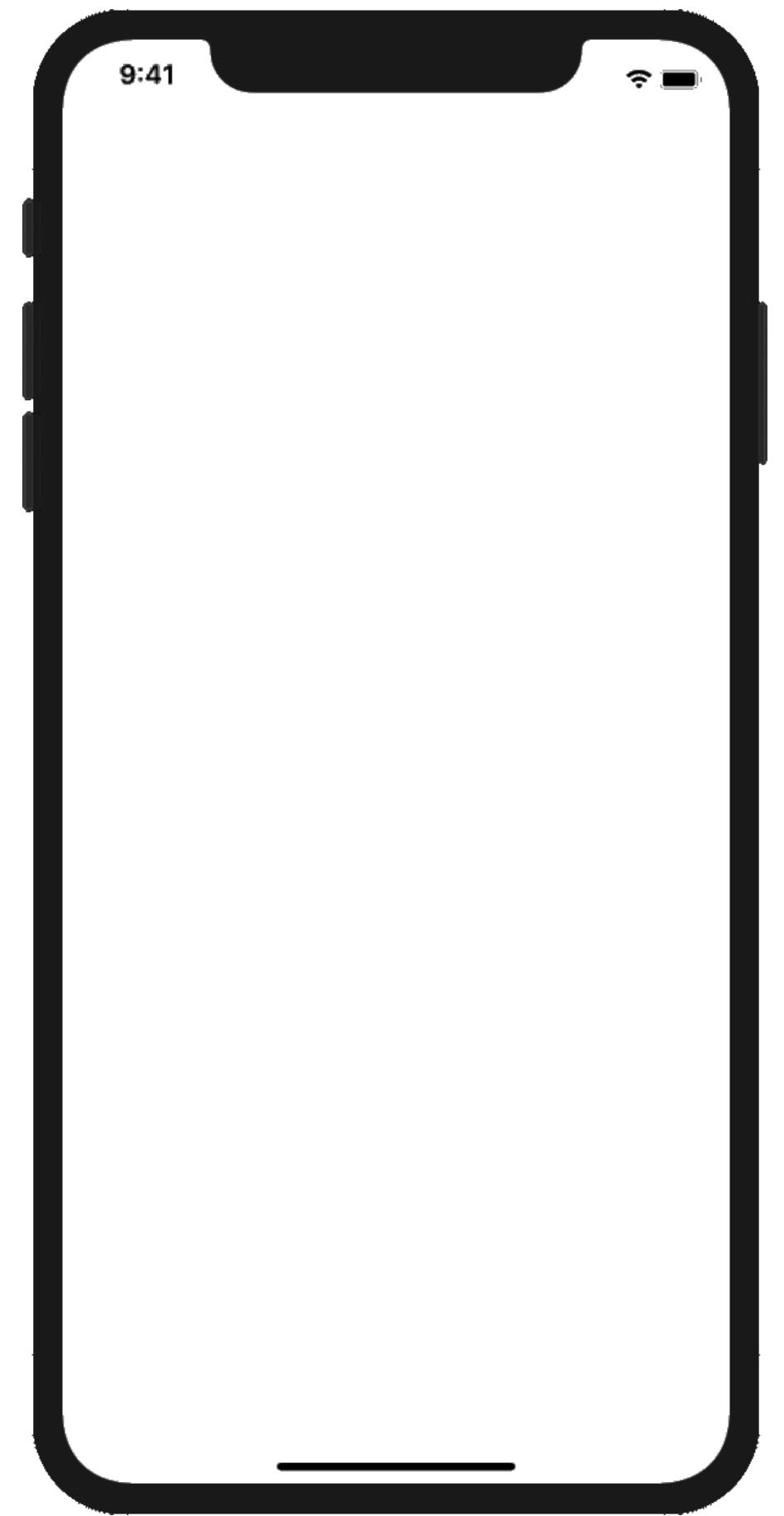
Home

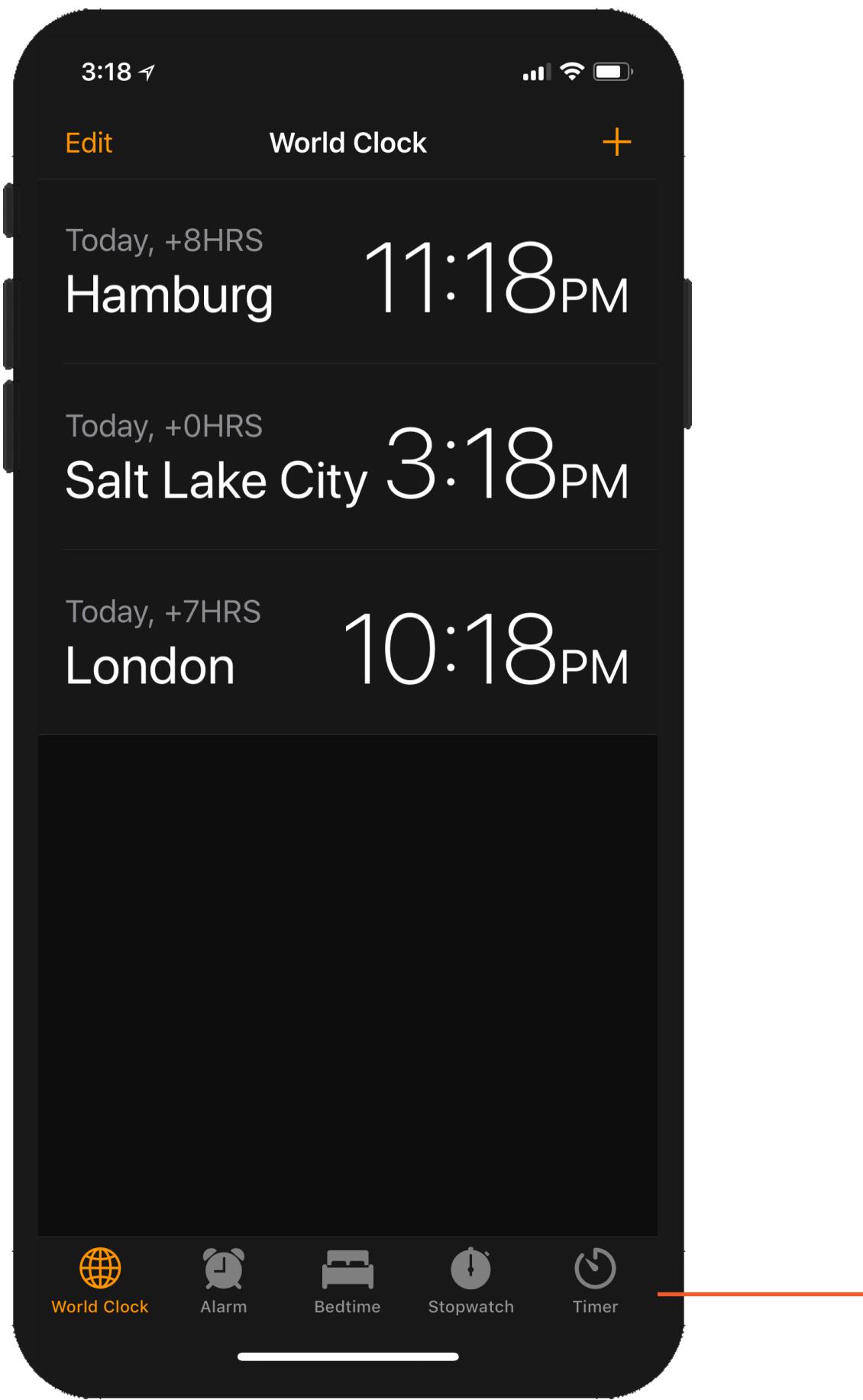
Details

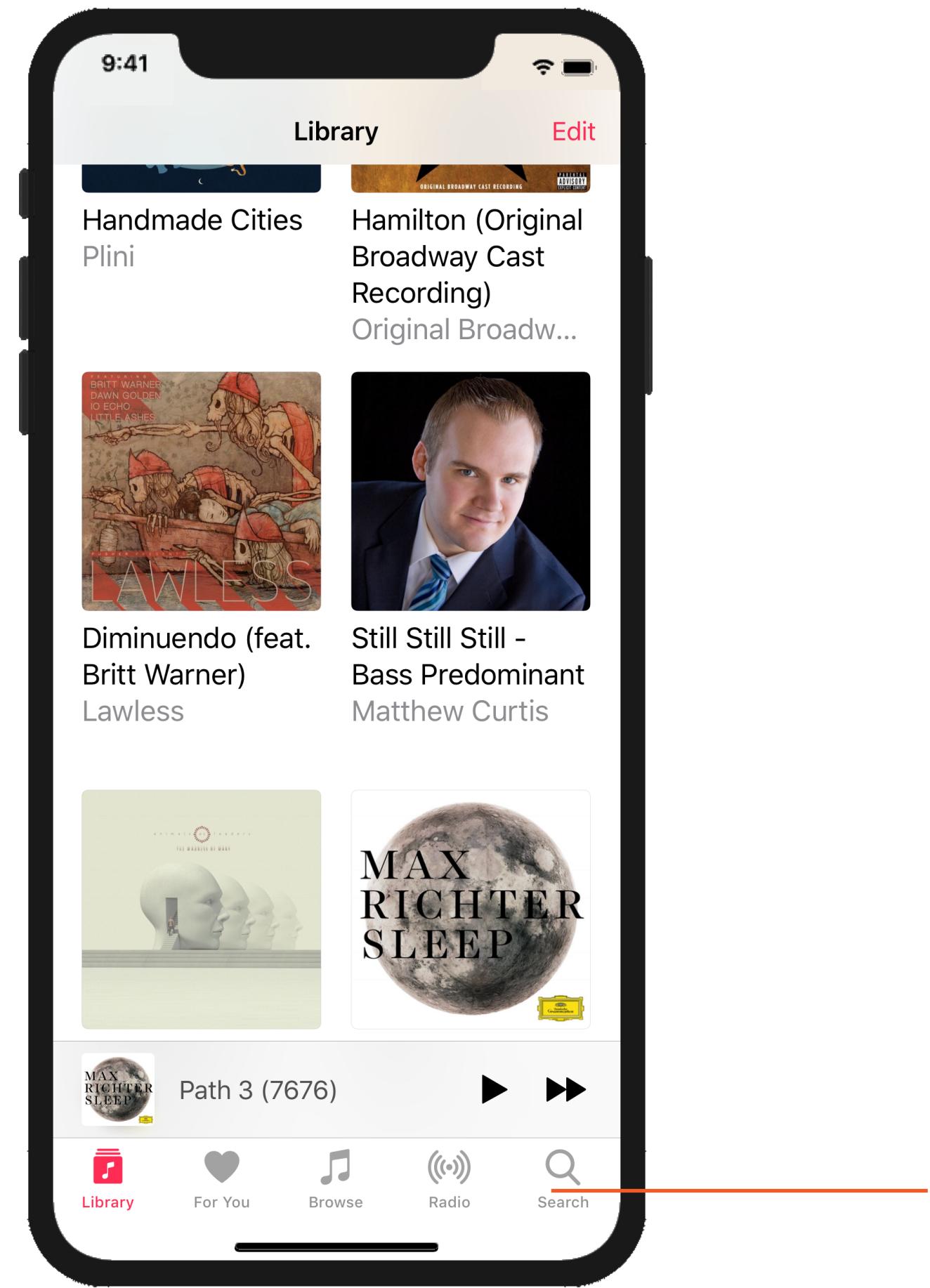
Specifics

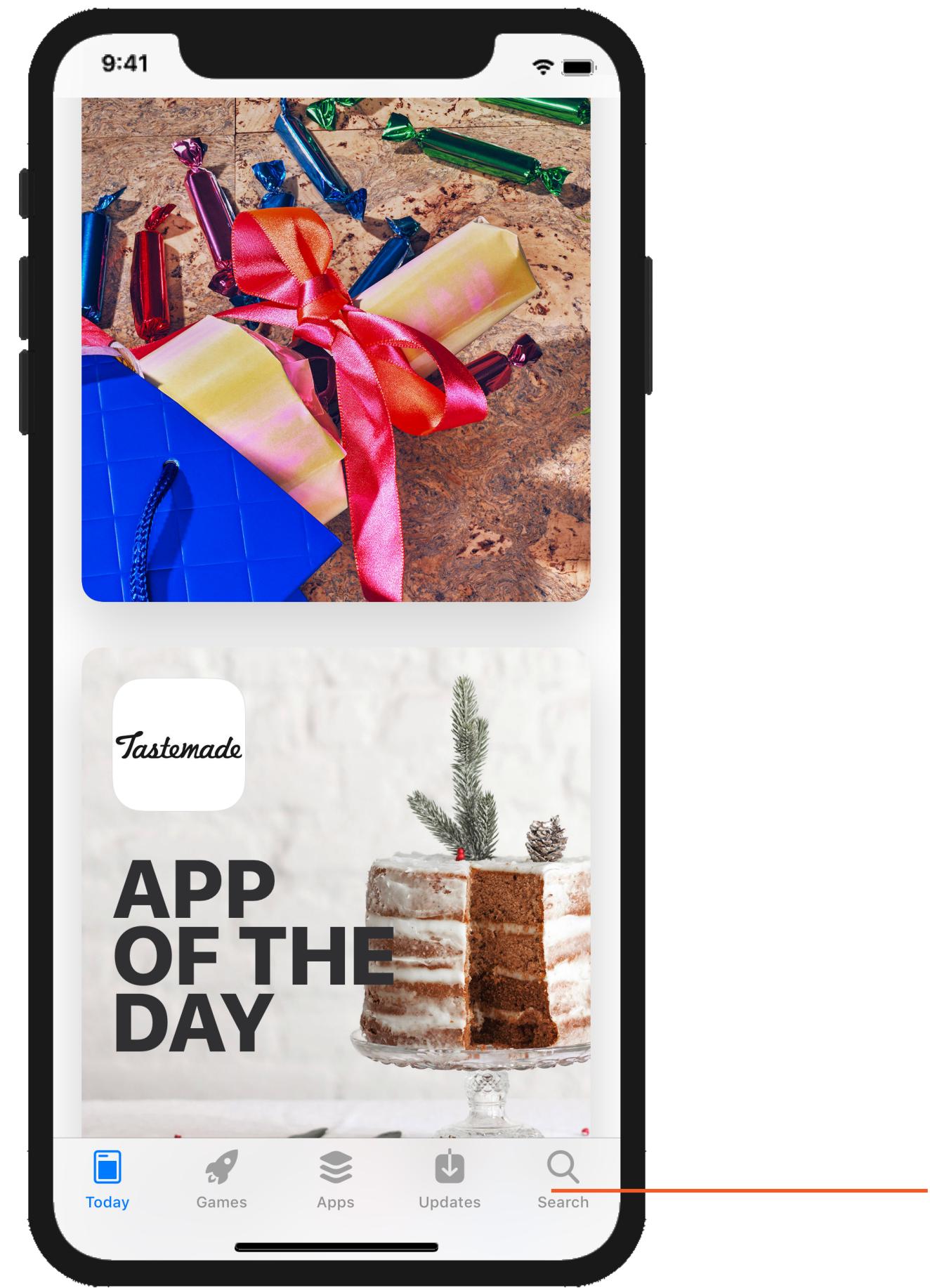
Distinctives

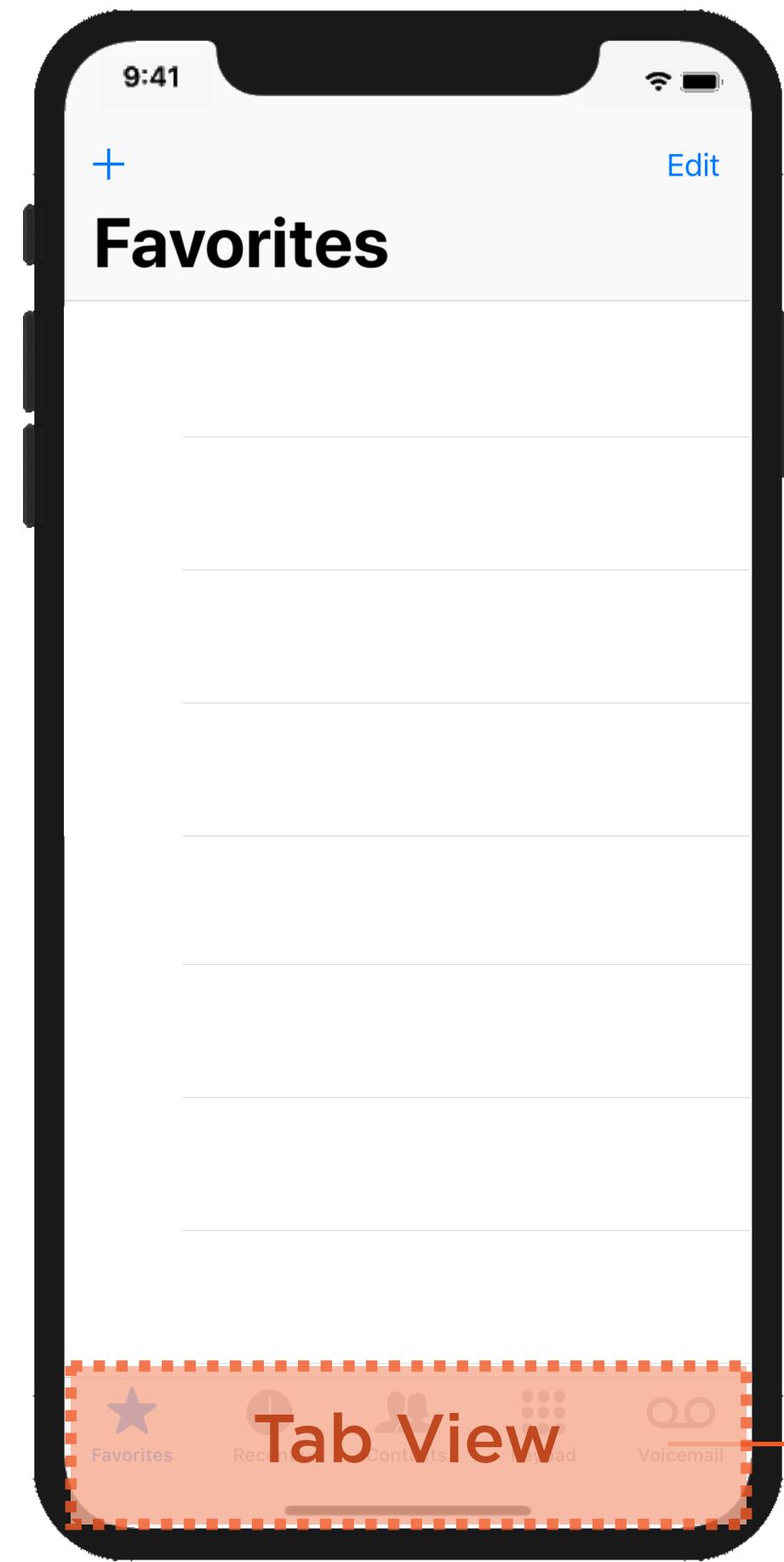
Drill-through Hierarchy





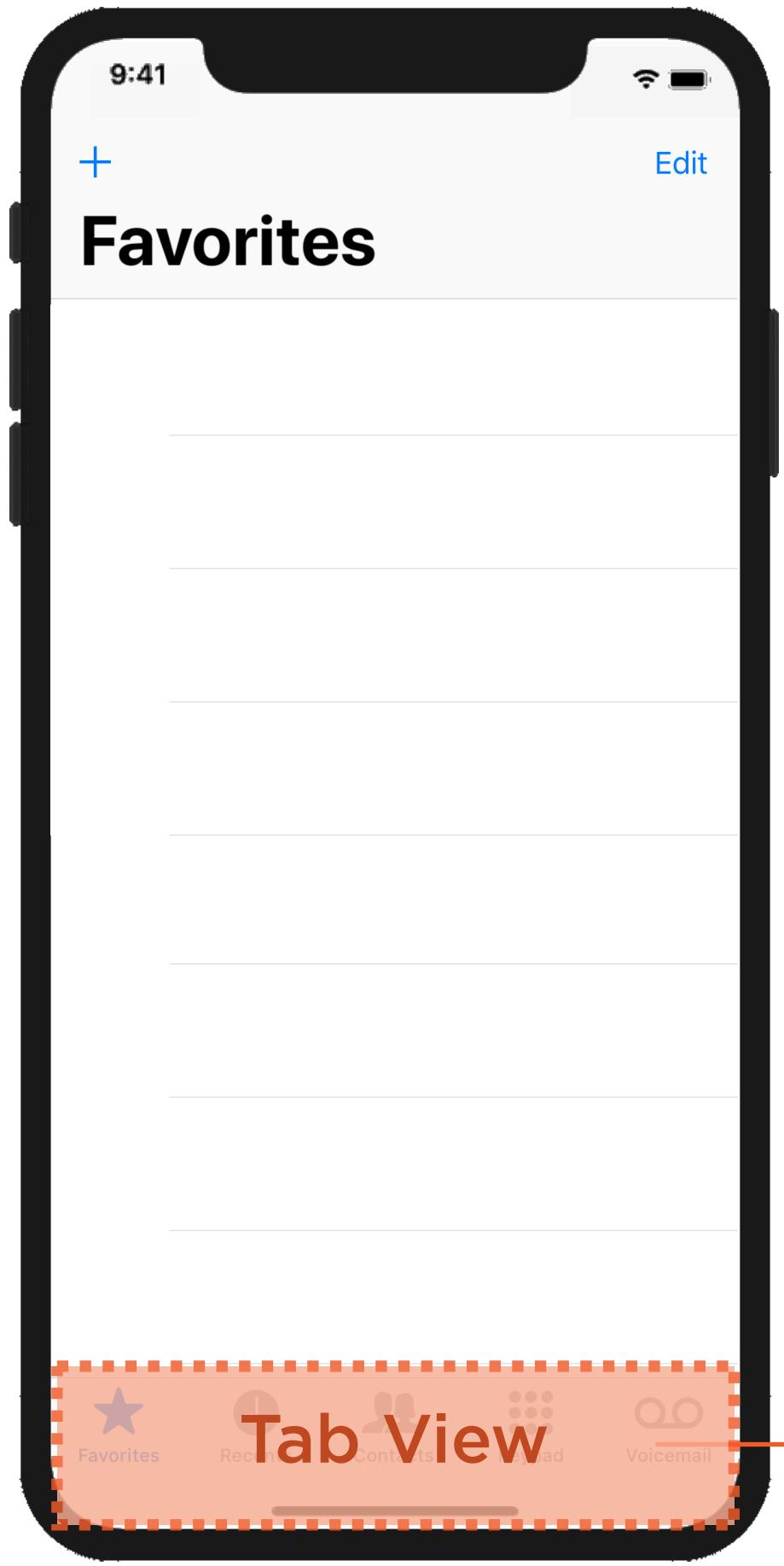






Tab View

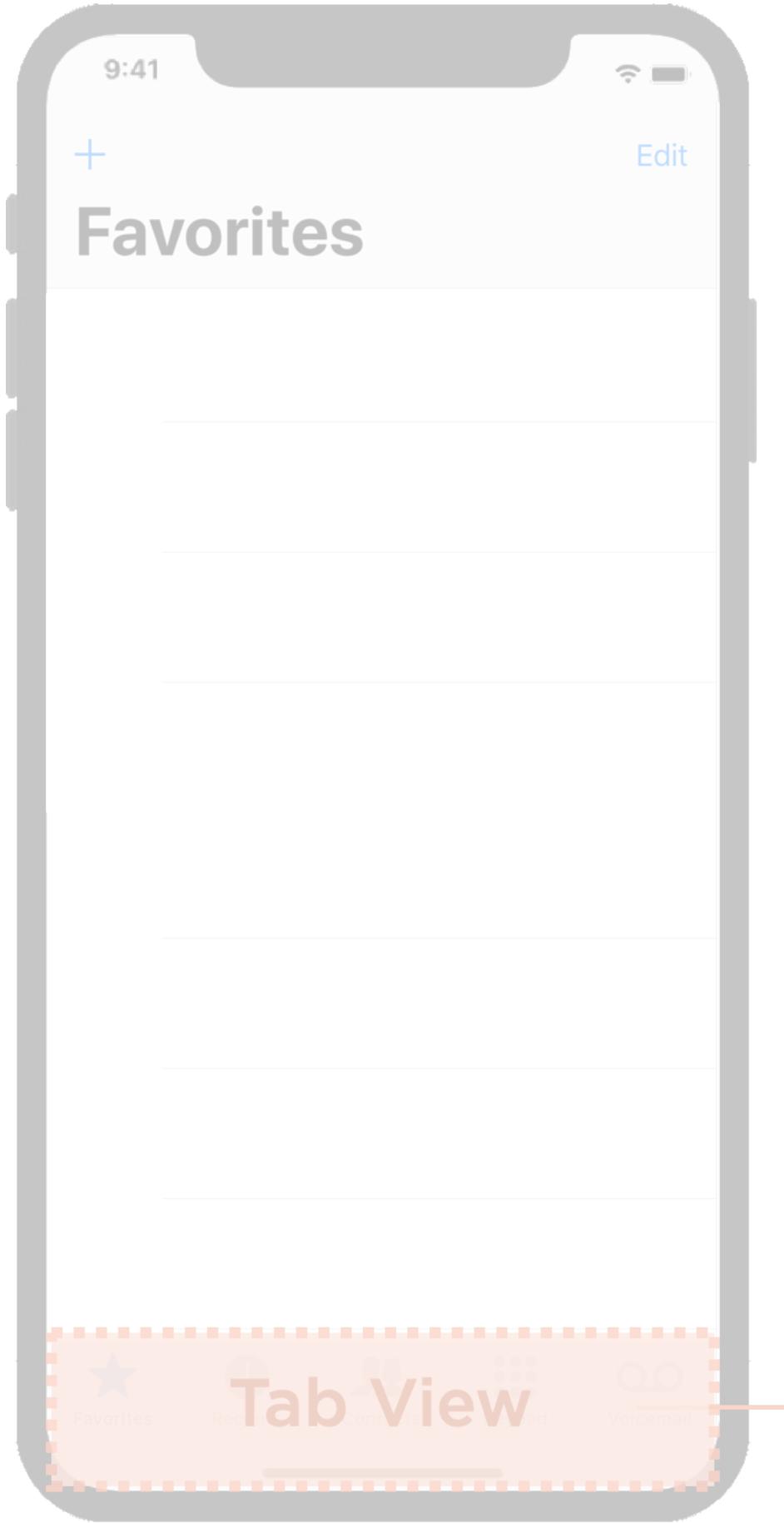
Tab Bar



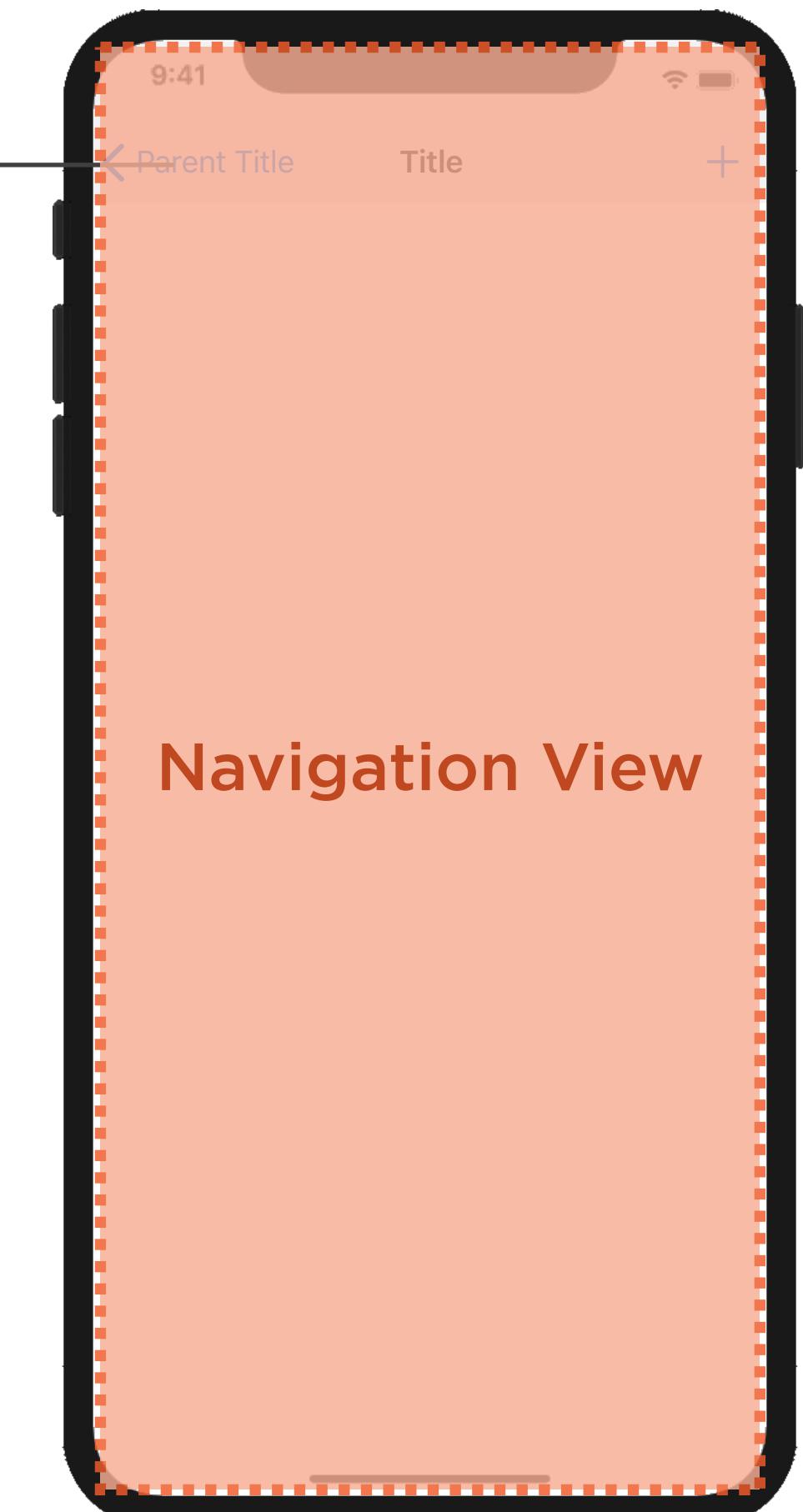
Navigation Bar

Tab Bar





Navigation Bar



Tab Bar

Implementing Navigation Views

Adding Navigation to an Existing Project

Extracting Reusable Subviews

NightWatch > iPhone 11 NightWatch | Build for Previews NightWatch: Succeeded | Today at 4:24 AM

Maybe I should make a separate View struct from this...

```
Section(header: HStack {  
    Image(systemName: "sunset")  
    Text("Weekly Tasks")  
}  
.font(.title3)) {  
    ForEach(weeklyTasks, id: \.self,  
        content: {  
            taskName in  
            Text(taskName)  
        })  
}  
  
Section(header: HStack {  
    Image(systemName: "calendar")  
    Text("Monthly Tasks")  
}  
.font(.title3)) {  
    ForEach(monthlyTasks, id: \.self,  
        content: {  
            taskName in  
            Text(taskName)  
        })  
}  
}.listStyle(GroupedListStyle())
```

NIGHTLY TASKS

- Check all windows
- Check all doors
- Check that the safe is locked
- Check the mailbox
- Inspect security cameras
- Clear ice from sidewalks
- Document "strange and unusual" occurrences

WEEKLY TASKS

- Check inside all vacant rooms
- Walk the perimeter of property

MONTHLY TASKS

- Test security alarm
- Test motion detectors
- Test smoke alarms

Filter

Section Multiple Sizes

Knowing how to extract
reusable views is something
you **must** know how to do as
an iOS developer.

NightWatch | Build for Previews NightWatch: Succeeded | Today at 5:13 AM

ContentView.swift

```
complete button")
    }
}

Section(header: HStack {
    Image(systemName: "sunset")
    Text("Weekly Tasks")
})
.font(.title3)) {
    ForEach(weeklyTasks, id: \.self,
        content: {
            taskName in
            NavigationLink(taskName,
                destination: Text(taskName))
        }
}

Section(header: HStack {
    Image(systemName: "calendar")
    Text("Monthly Tasks")
})
.font(.title3)) {
    ForEach(monthlyTasks, id: \.self,
        content: {
            taskName in
            NavigationLink(taskName,
                destination: Text(taskName))
        }
}
```

Preview

Home

NIGHTLY TASKS

- Check all windows
- Check all doors
- Check that the safe is locked
- Check the mailbox
- Inspect security cameras
- Clear ice from sidewalks
- Document "strange and unusual" occurrences

WEEKLY TASKS

- Check inside all vacant rooms
- Walk the perimeter of property

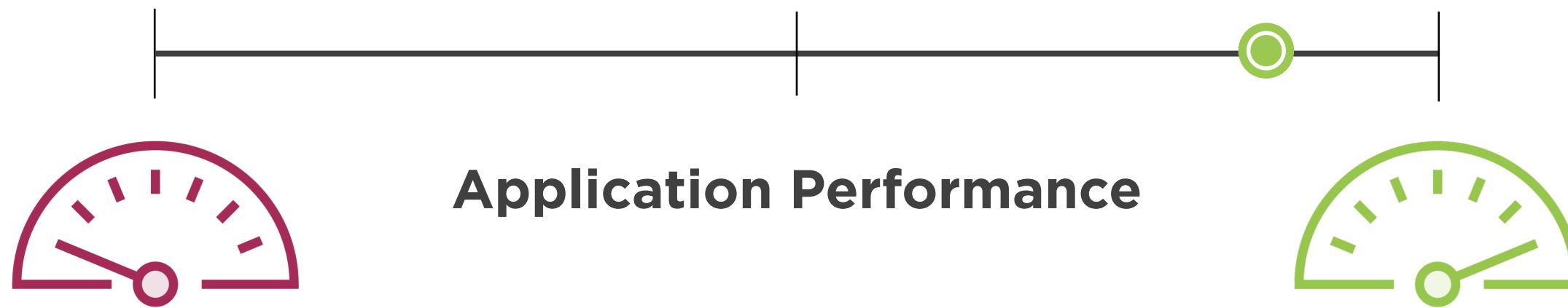
MONTHLY TASKS

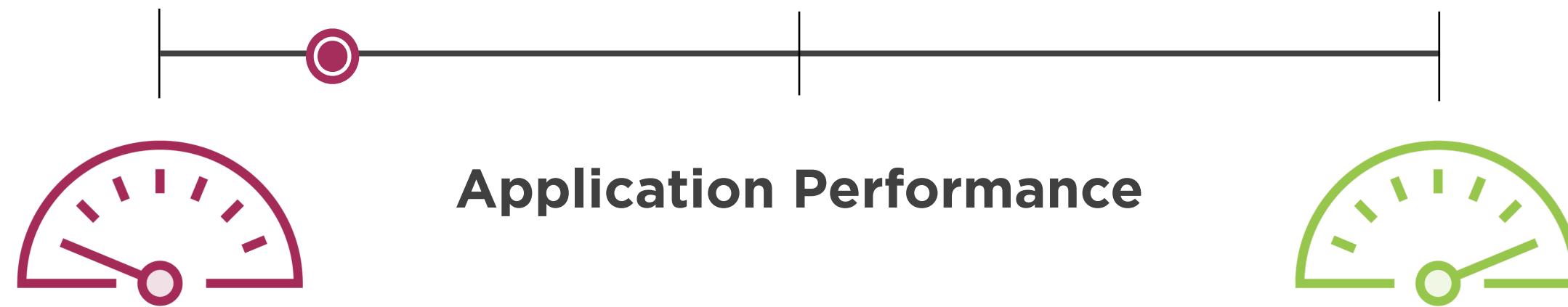
- Test security alarm
- Test motion detectors
- Test smoke alarms

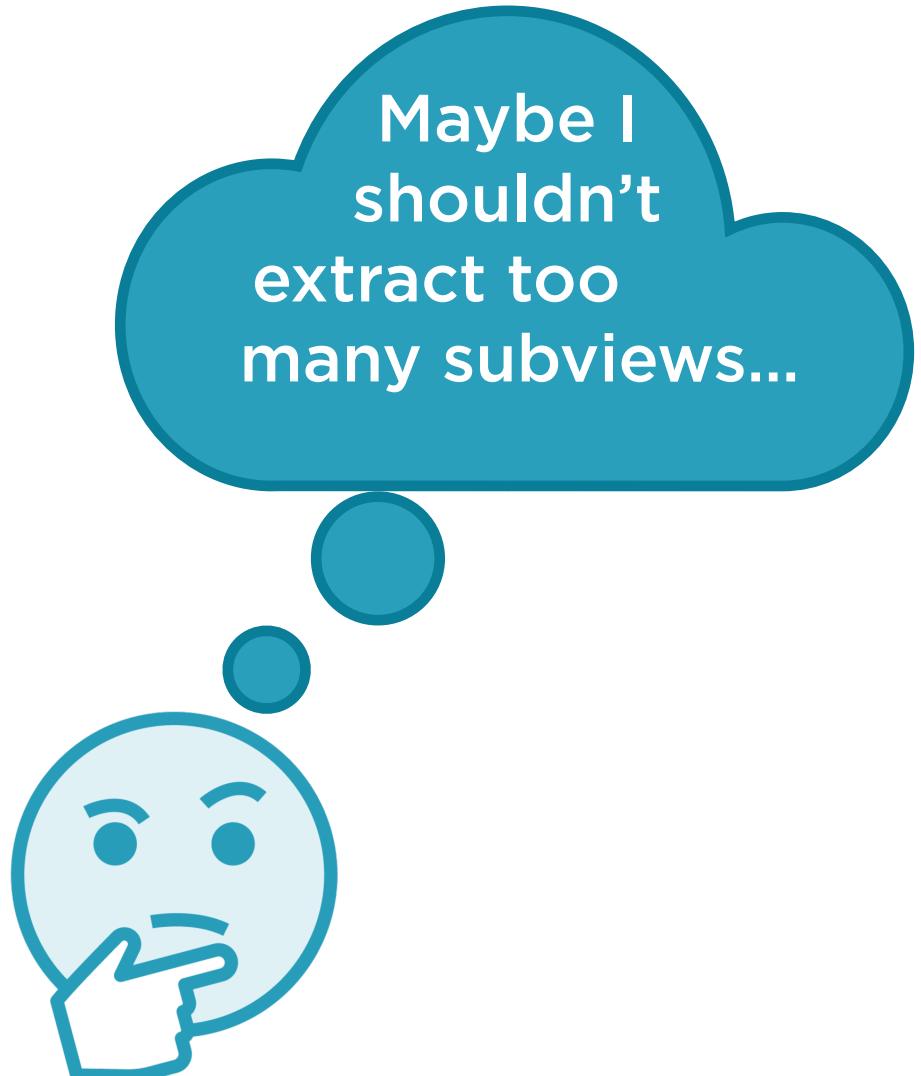
Filter

Navigation... 414x814

50%

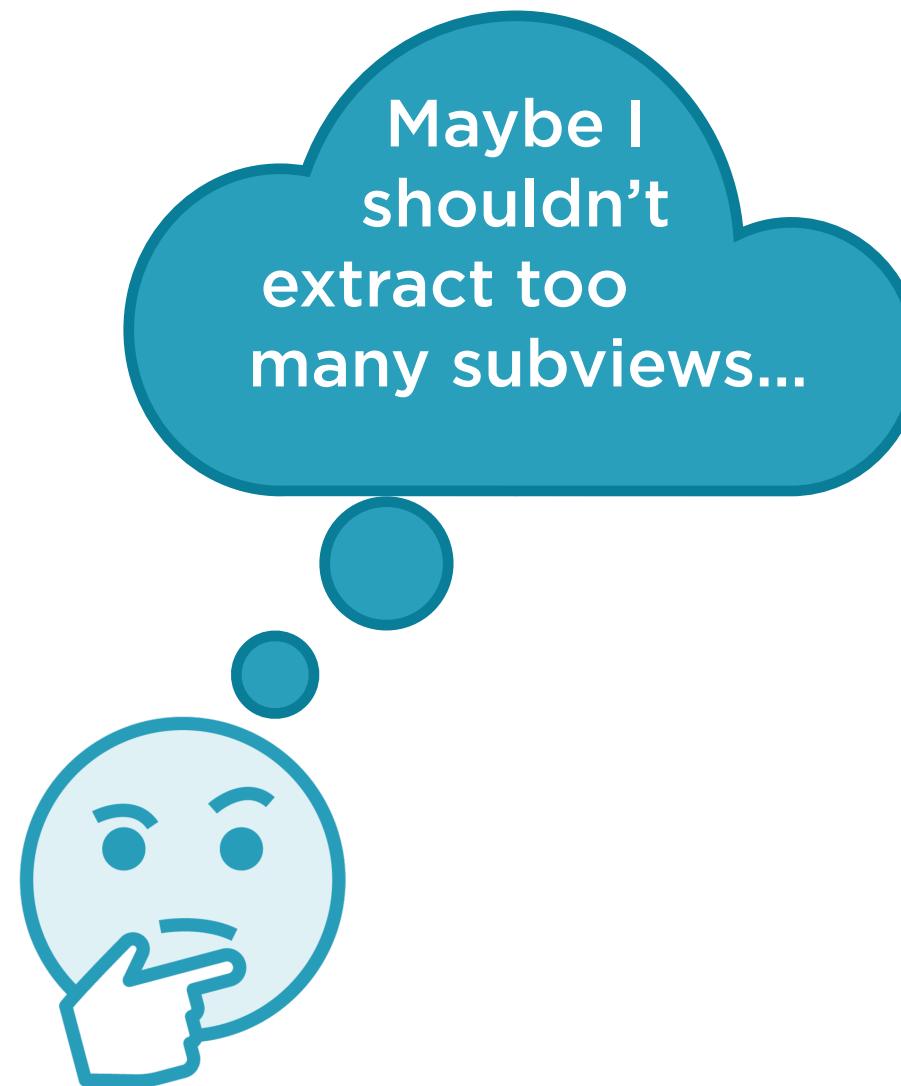






Maybe I
shouldn't
extract too
many subviews...

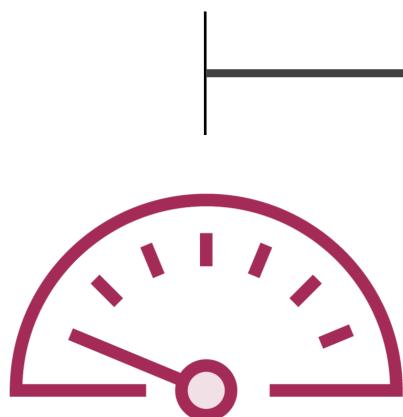




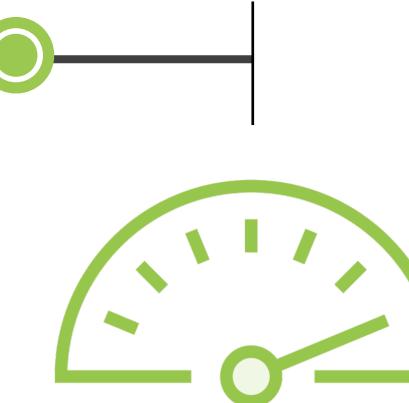
Application Performance

SwiftUI

- + Layout engine is smart
 - + Views are “light-weight” structs
 - + No inherited baggage
 - + Efficient memory footprint
 - + Fast and inexpensive to initialize and compose together with other Views
-



Application Performance



Swift**UI**

- + Layout engine is smart
 - + Views are “light-weight” structs
 - + No inherited baggage
 - + Efficient memory footprint
 - + Fast and inexpensive to initialize and compose together with other Views
-

Extract subviews whenever you feel it would be helpful for organizing your code, making it more maintainable and more reusable.

Creating a Tabbed Application



World Clock



Alarm



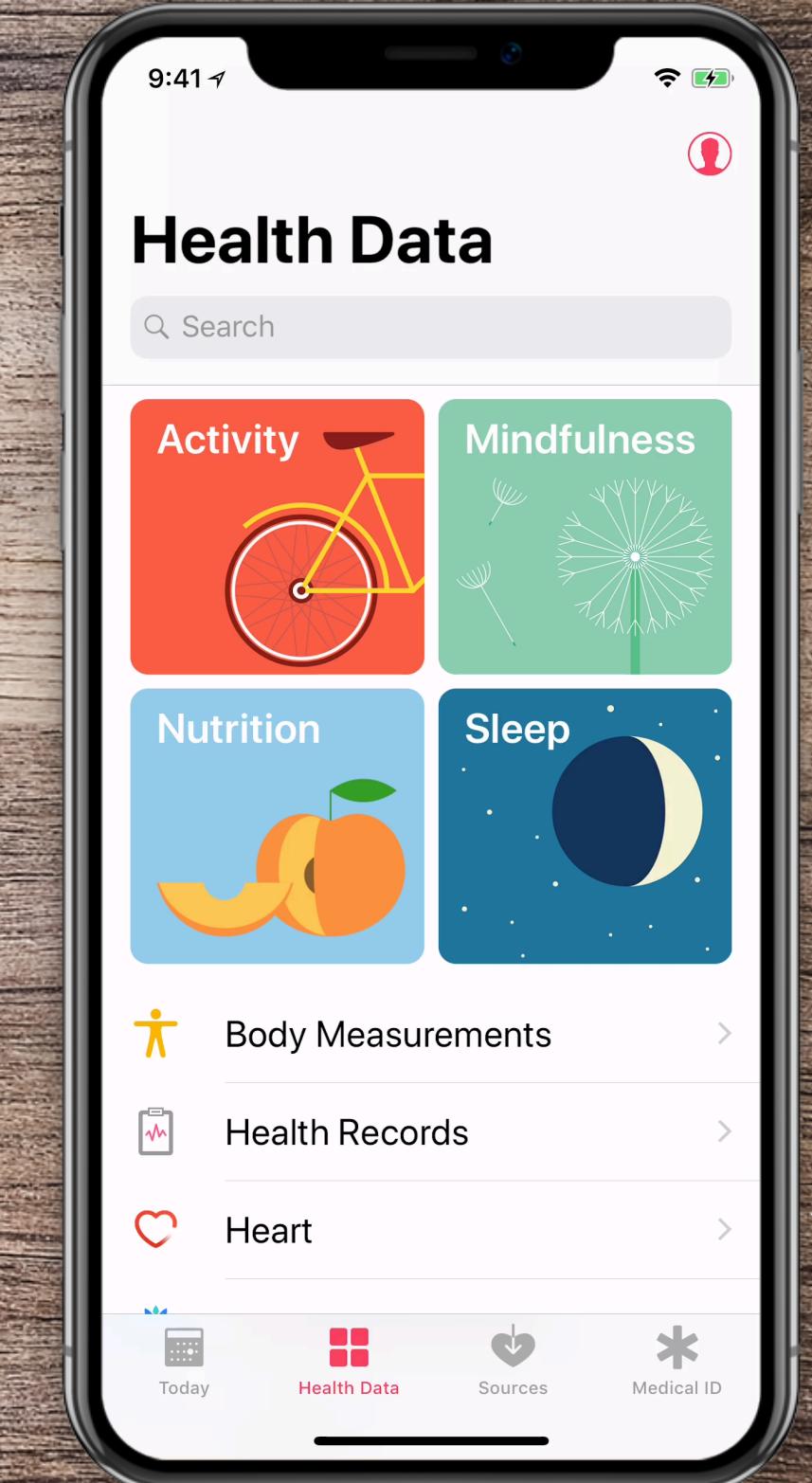
Bedtime

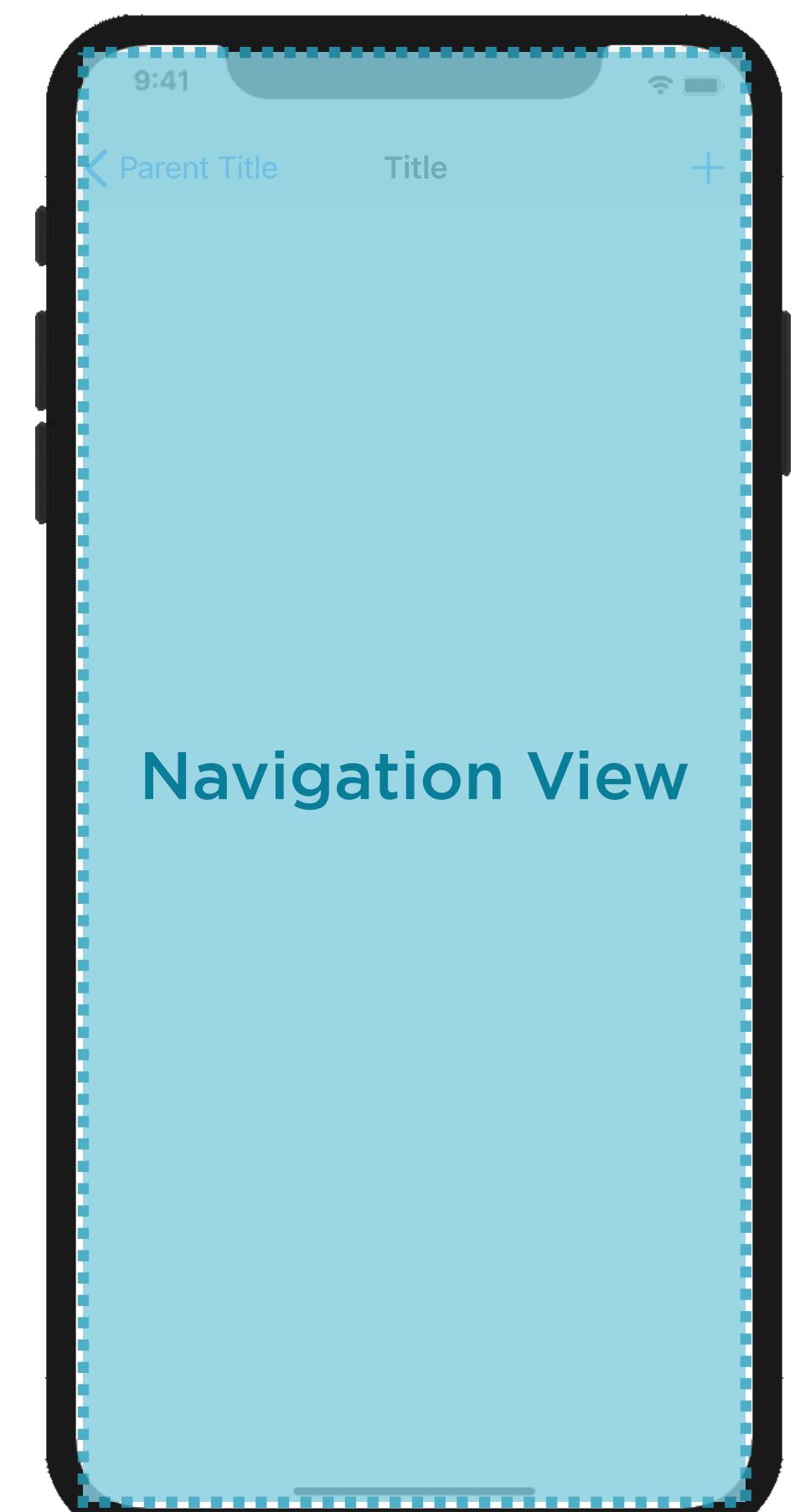
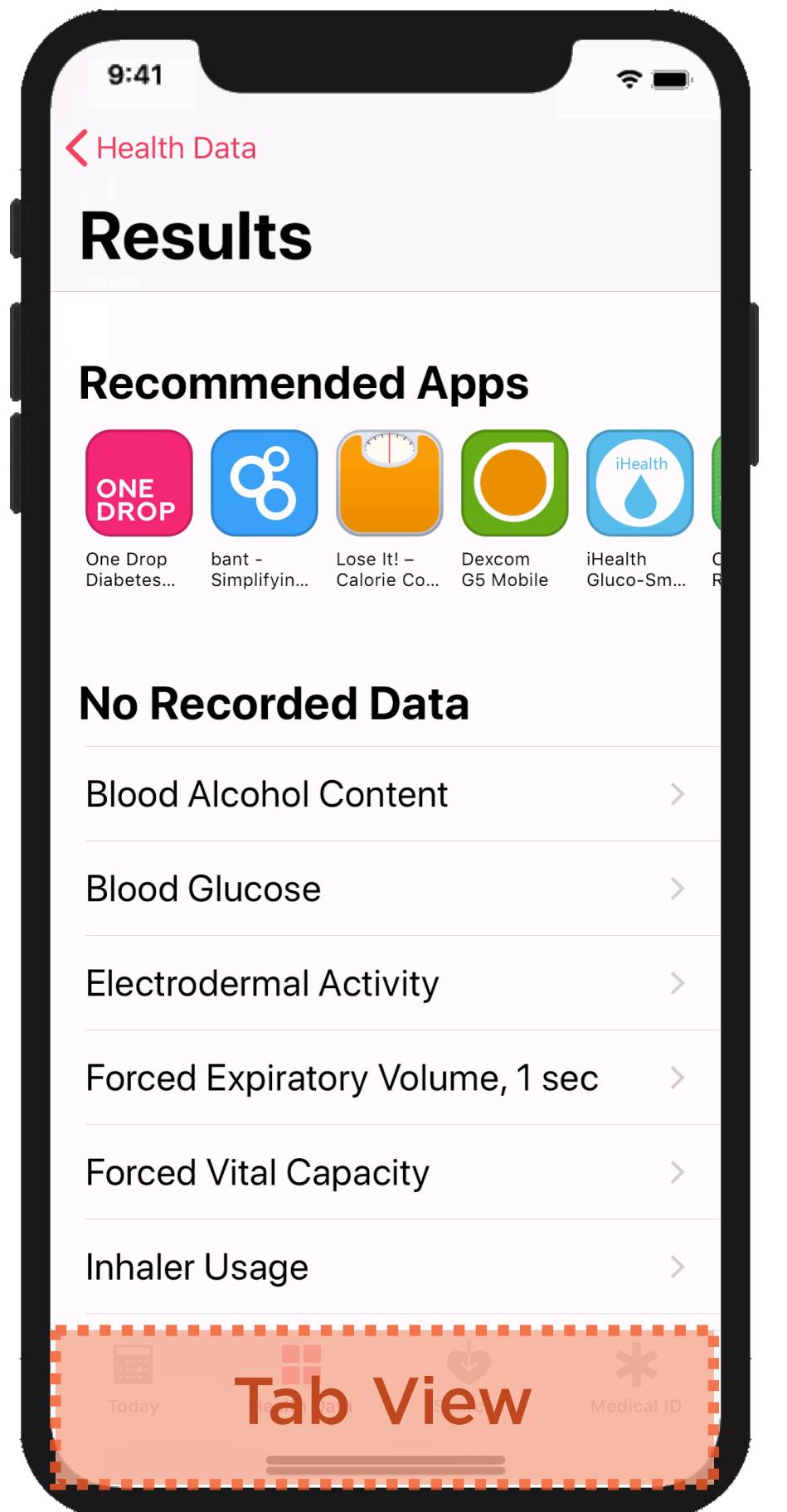


Stopwatch



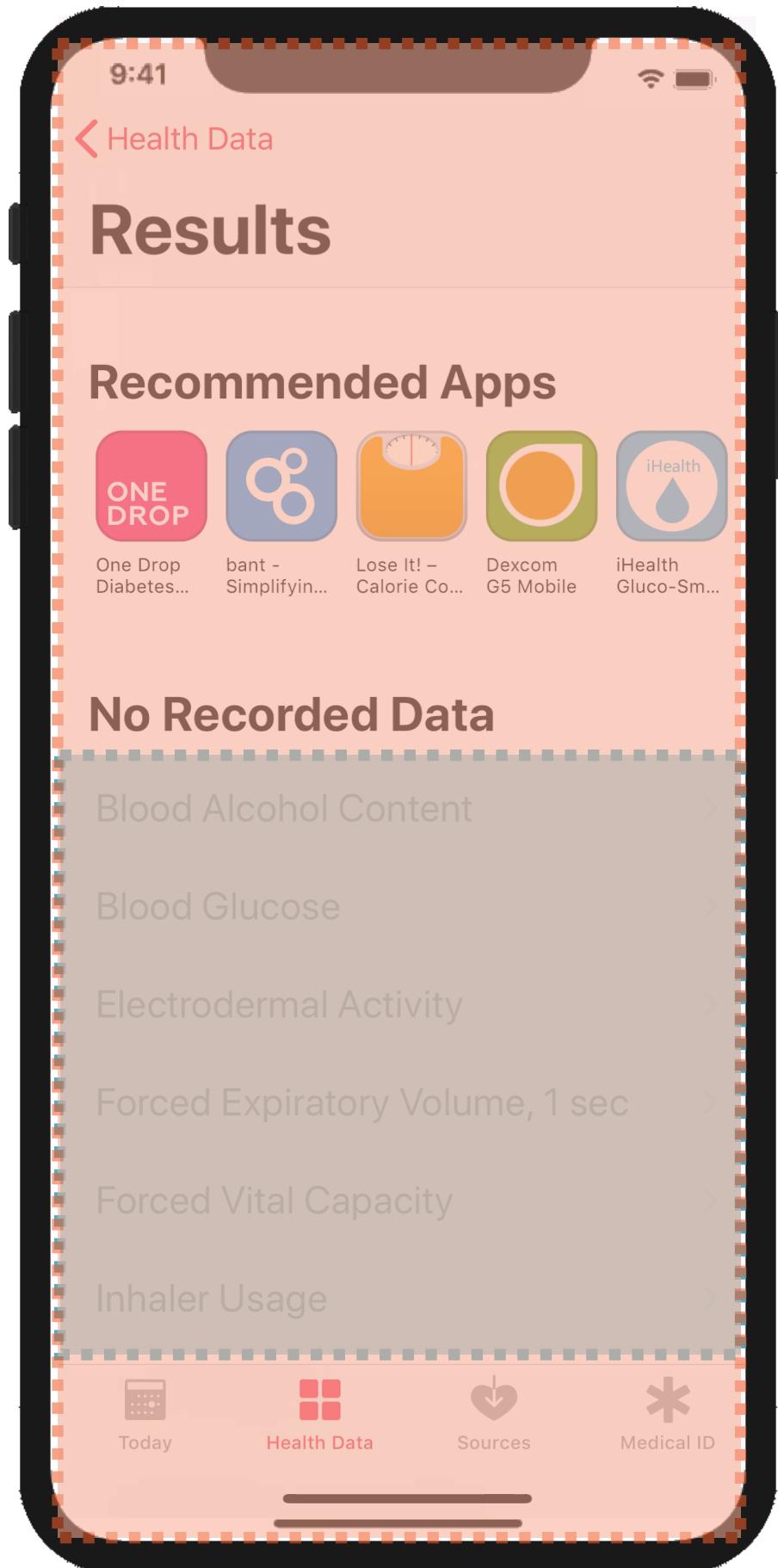
Timer





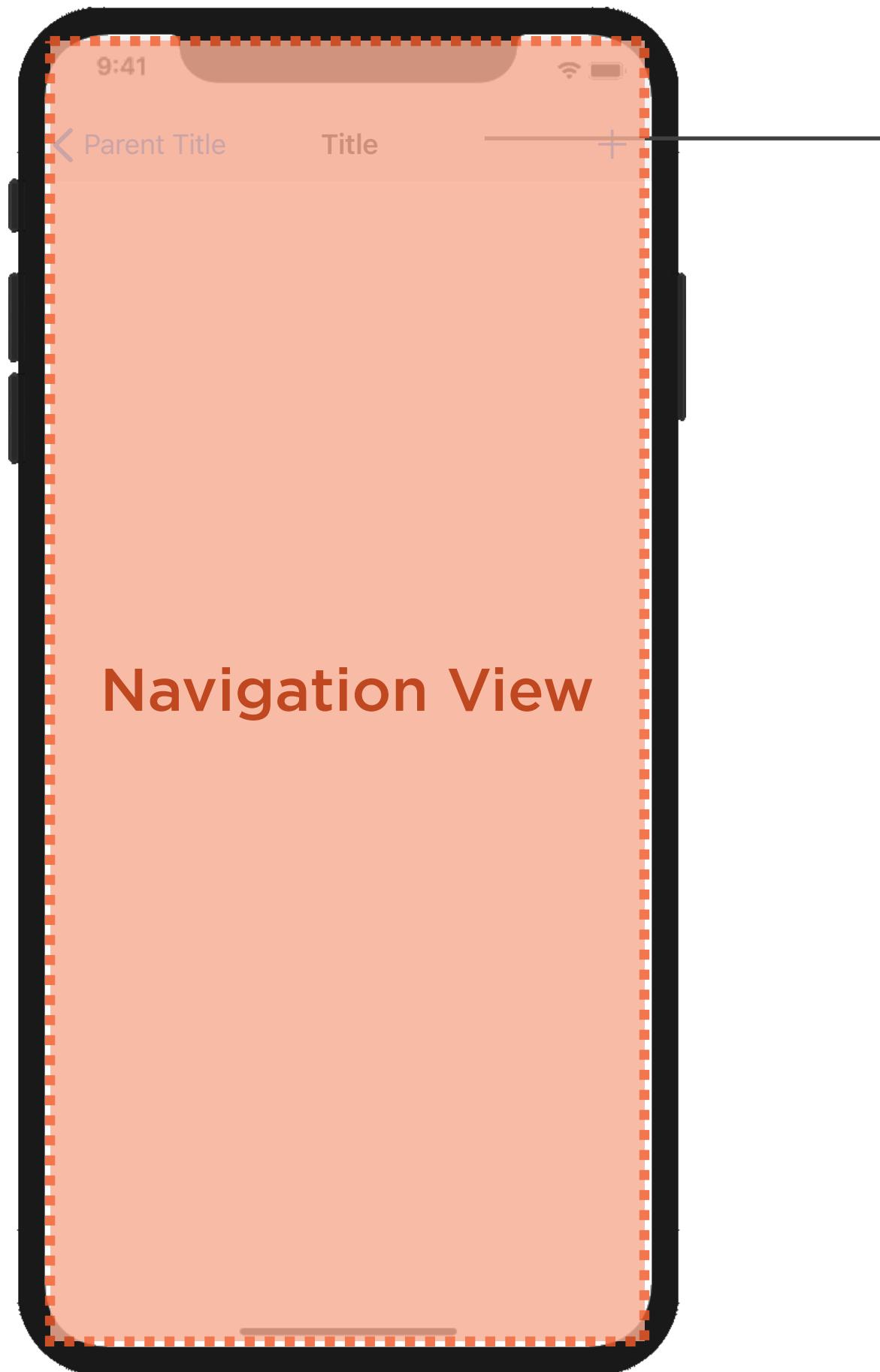
Navigation View

Parent Tab View



Child Navigation View

Recapping Multi-view Applications



Navigation Bar

Navigation View



Navigation Bar

Home

Details

Specifics

Distinctives

Drill-through Hierarchy

Navigation View



World Clock



Alarm



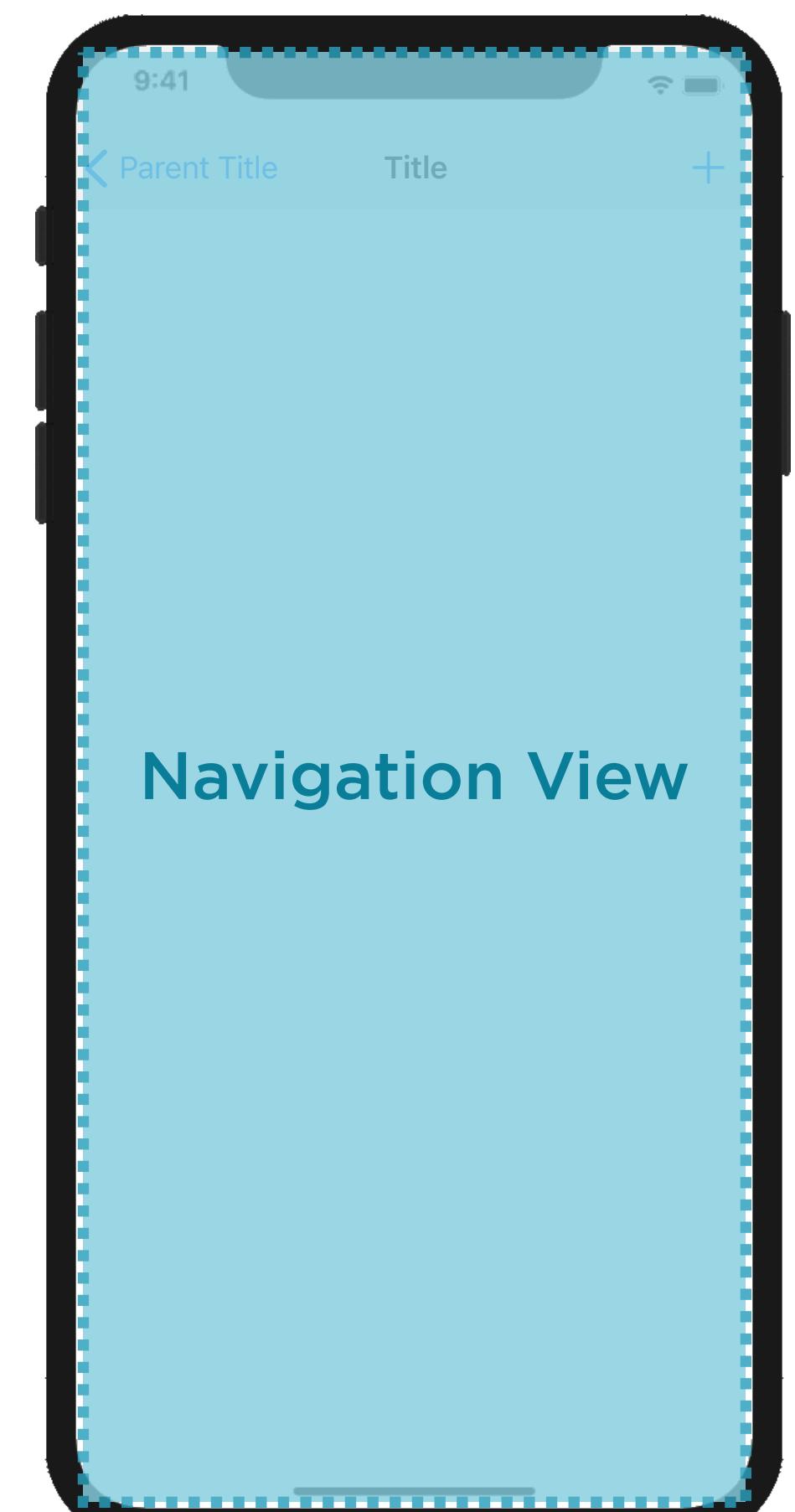
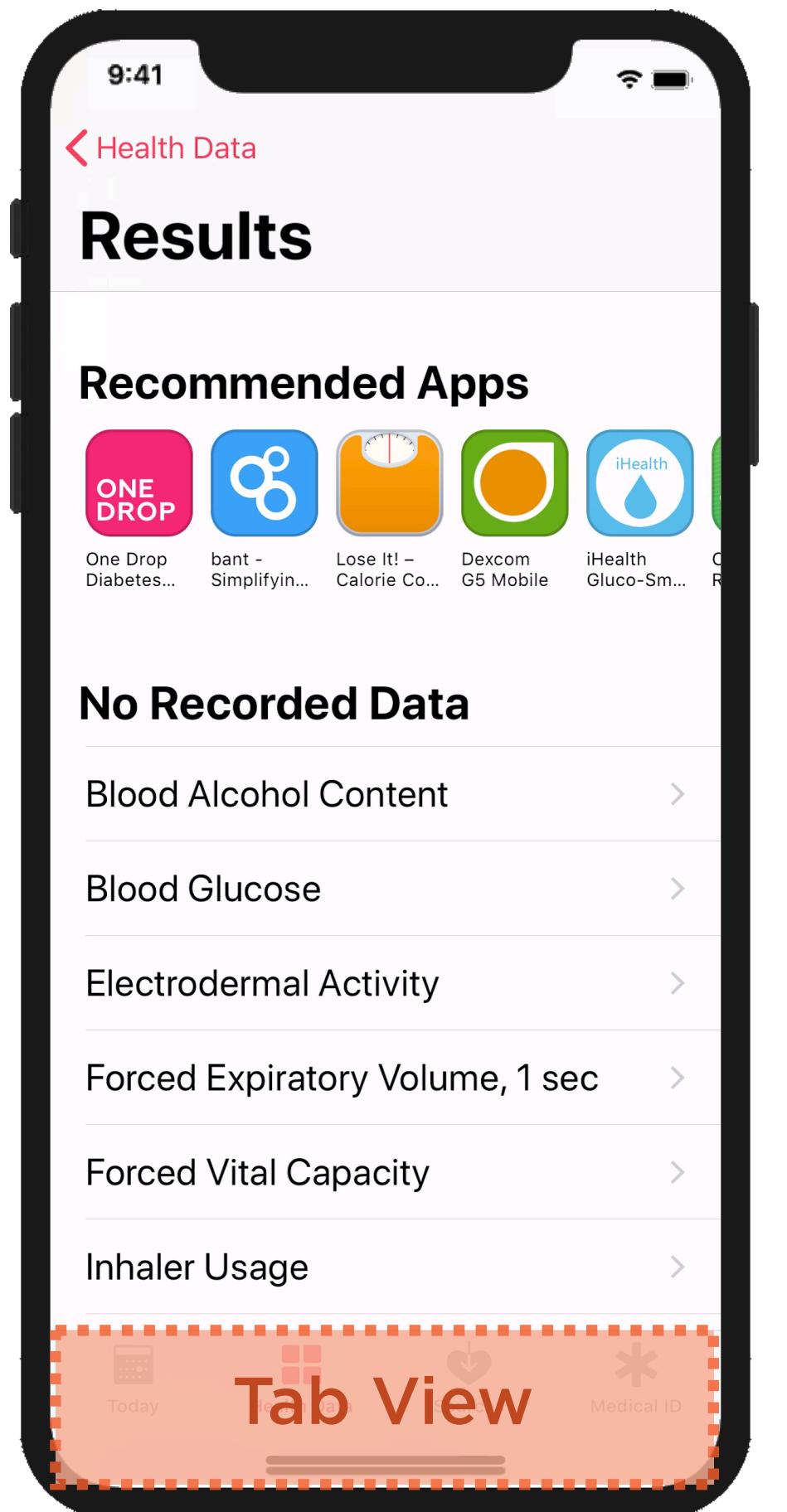
Bedtime



Stopwatch

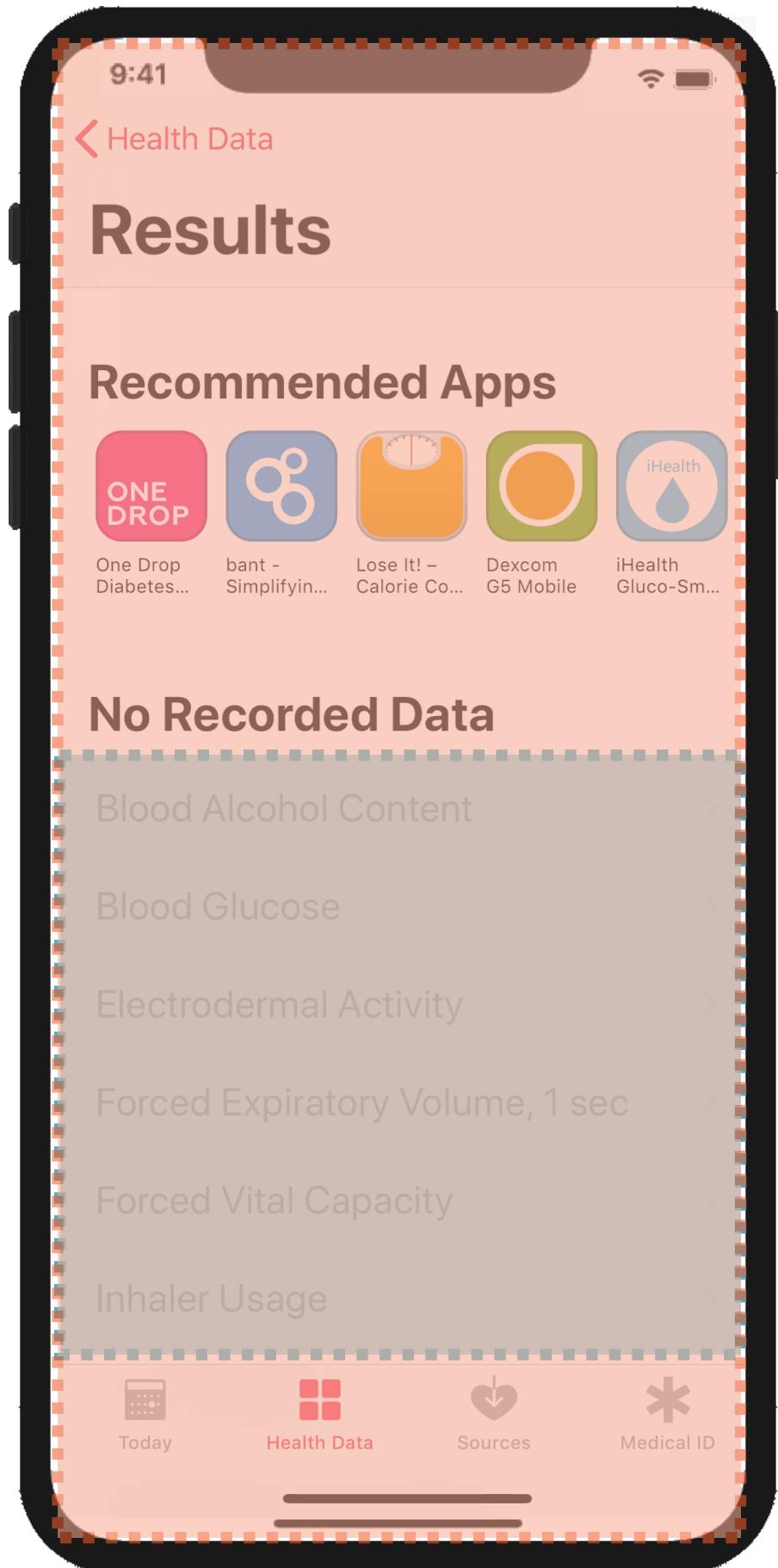


Timer



Navigation View

Parent Tab View



Child Navigation View

Adding Behavior and Working with Data

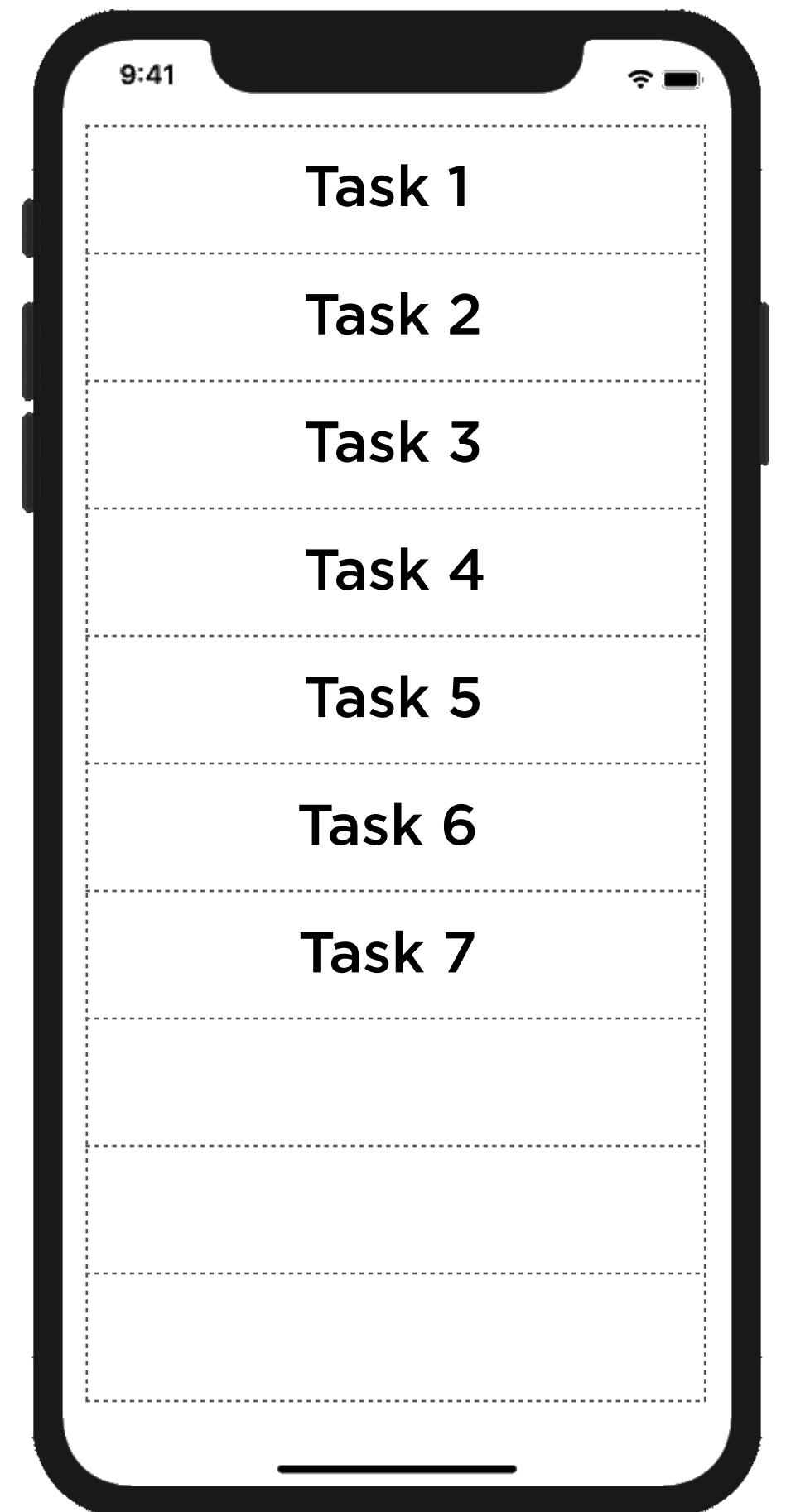


Andrew Bancroft

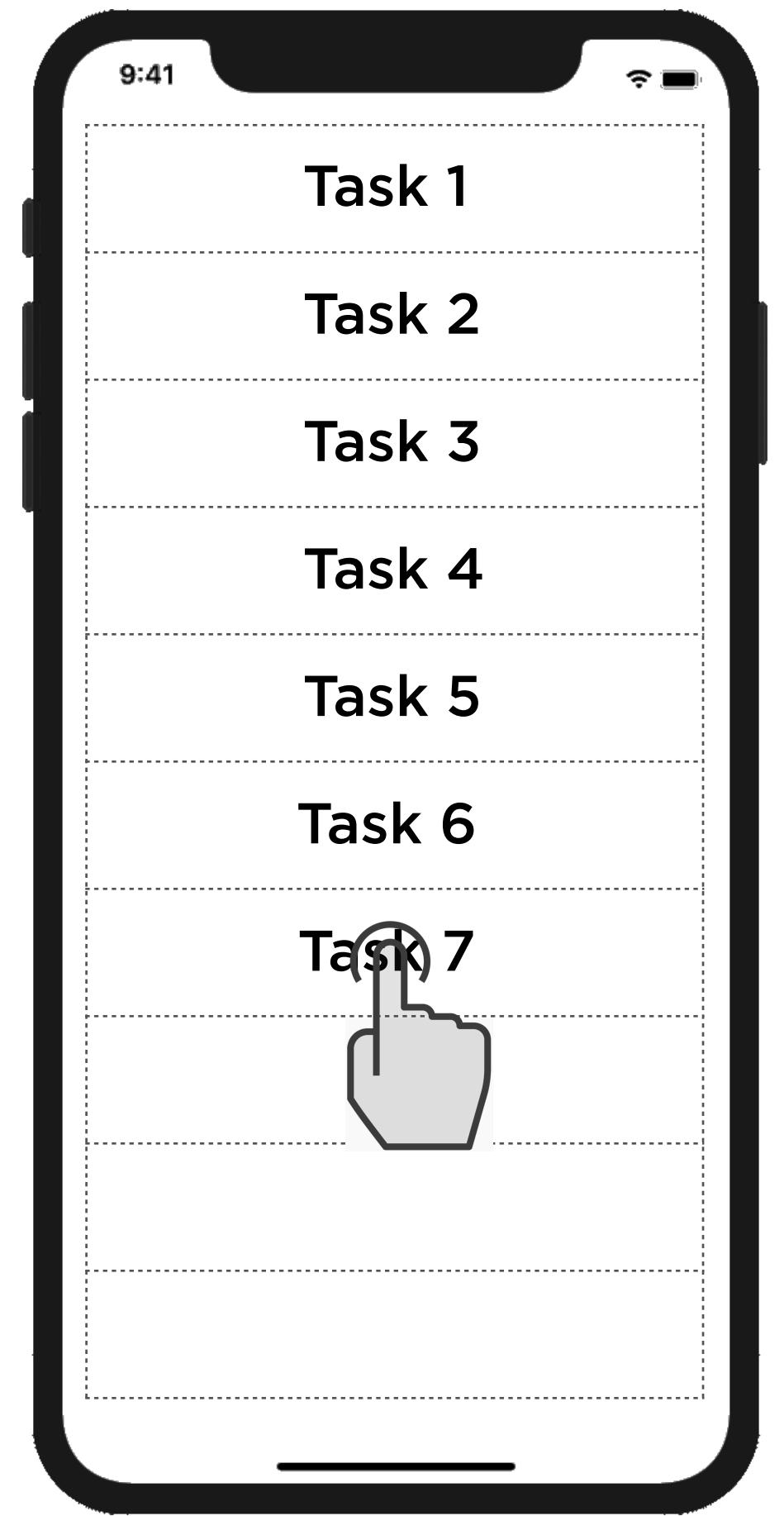
@andrewcbancroft www.andrewcbancroft.com

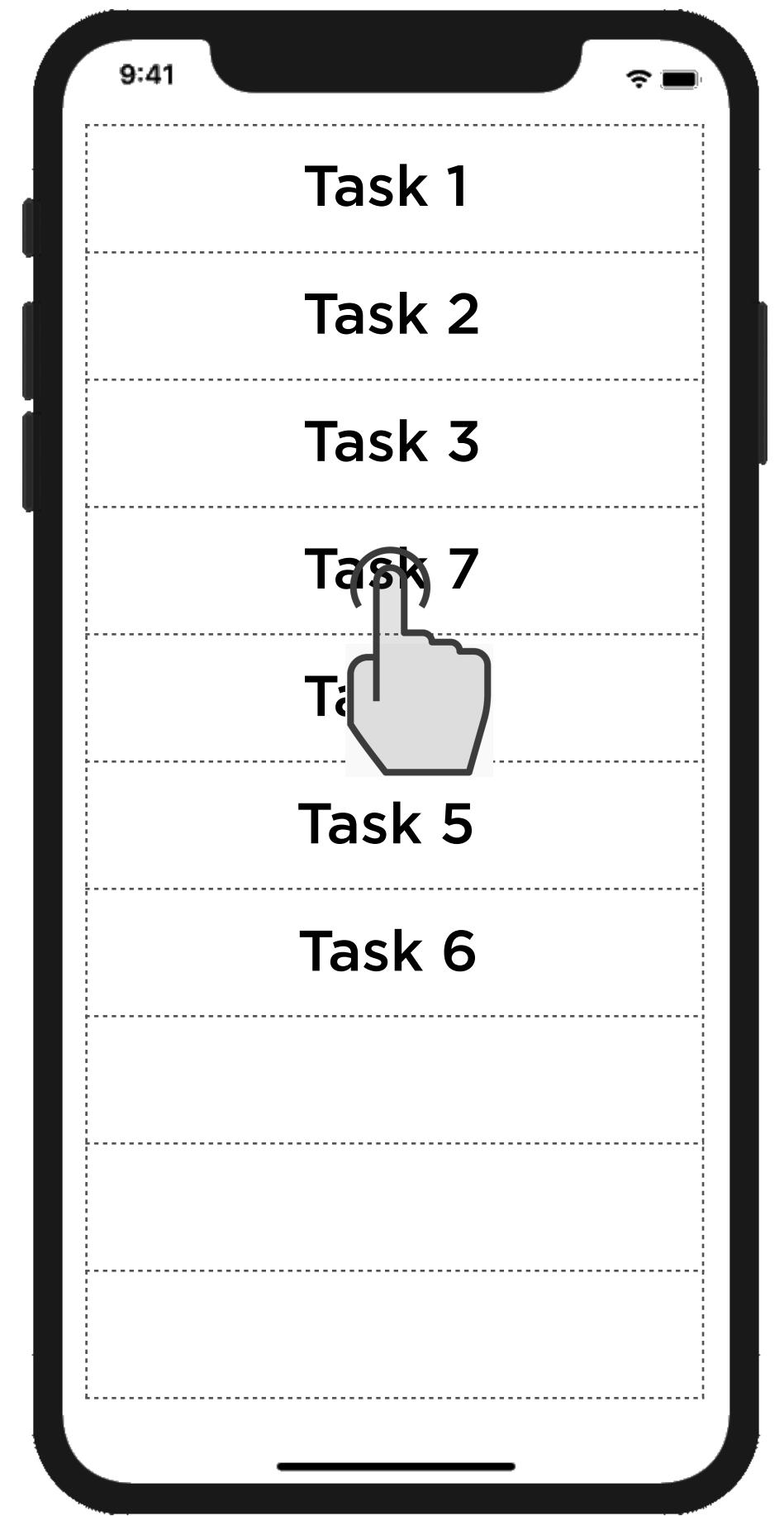
Why do we build a user interface in the first place?

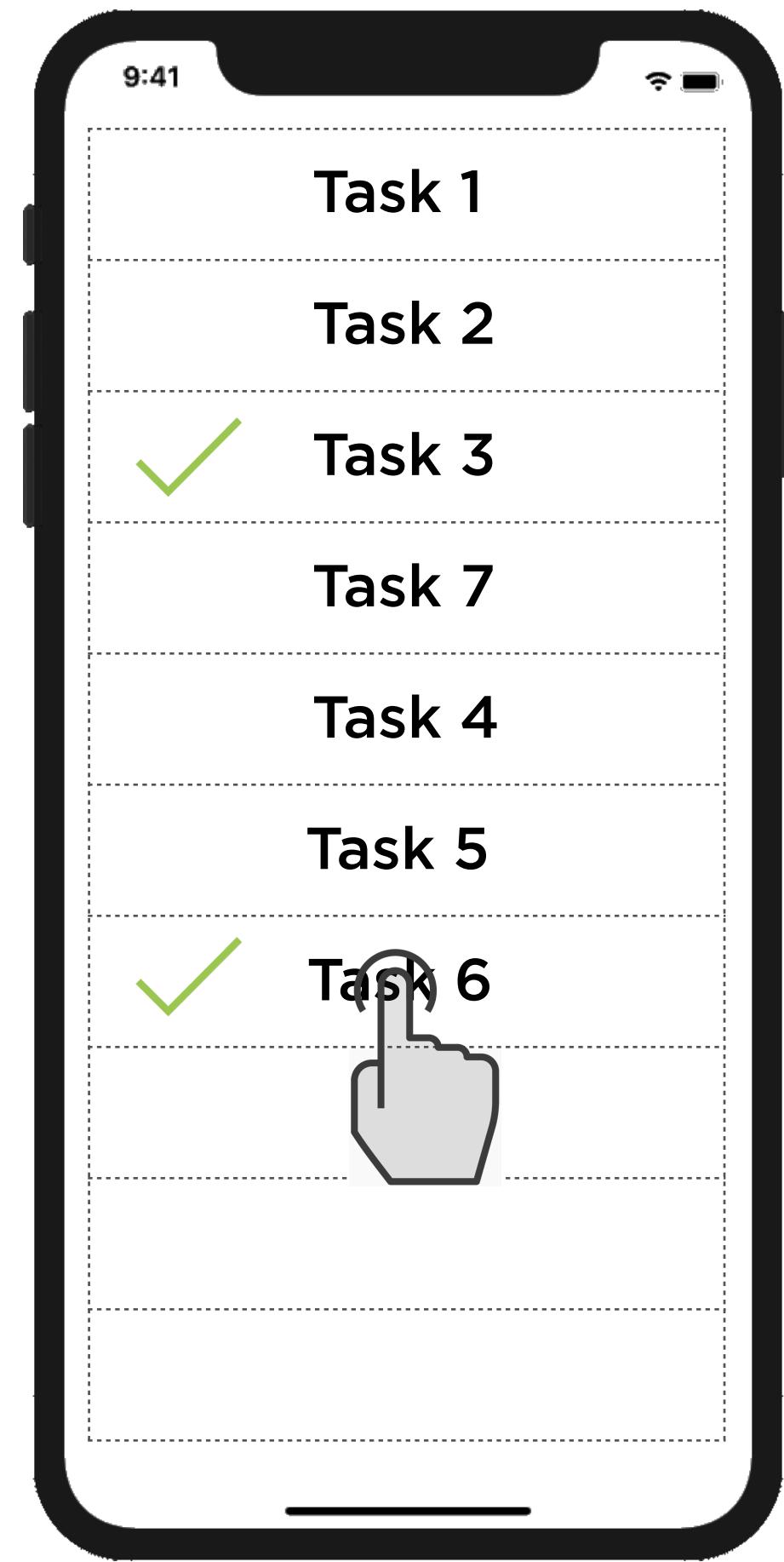
So that...











A View has one critical job:



Accurately **represent** the **data** of our application
in what it displays on the screen.

Interact



Modify Data

View's #1 Job



Accurately **represent** the **data** of our application
in what it displays on the screen.

Everything changes when
changes can be made to the data.

 Check all windows

Mark Complete

Reset



The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure under "NightWatch".
- Editor:** Displays the `ContentView.swift` file content.
- Preview:** Shows a simulated iPhone 11 displaying the app's home screen with sections for "NIGHTLY TASKS" and "WEEKLY TASKS".

```
1
2 import SwiftUI
3
4 let nightlyTasks = [...]
13
14 let weeklyTasks = [...]
18
19 let monthlyTasks = [...]
24
25 struct ContentView: View {
26     var body: some View {
27         NavigationView {
28             List {
29                 Section(header:
TaskSectionHeader(symbolSystemName:
"moon.stars", headerText: "Nightly
Tasks")) { ... }

```

// TODO:

-
-
-

- Mark tasks as complete
- Indicate “complete” status
- Take a look at the data model

Extending the Data Model

File □ ⚡ 🔍 ⚠️ ⌛ ⌂

NightWatch

NightWatch

- NightWatchApp.swift
- ContentView.swift
- DetailsView.swift
- Task.swift
- Assets.xcassets
- Info.plist

▶ Preview Content

▶ Products

ContentView.swift Task.swift

NightWatch > NightWatch > Task.swift > Task

```
1 import Foundation
2
3 struct Task {
4     let name: String
5     var isComplete: Bool
6     var lastCompleted: Date?
7 }
8
9
```

How do we...

- ① Initialize instances of this Task for our Views?
- ① Access and read Task instances in a SwiftUI View?
- ① Update Task instances?
- ① Share Task instances throughout the View hierarchy?
- ① Keep the UI in sync with the data?

Keep the UI in sync with the data



Keep the UI in sync with the data

Understanding the View Update Cycle in SwiftUI

View's #1 Job



Accurately **represent** the **data** of our application
in what it displays on the screen.

View's #1 Job



Accurately **represent** the **data** of our application
in what it displays on the screen.



How does SwiftUI know when to **update a View** so that
it's displaying the most up-to-date representation of our
app's data **when** that **data changes**?



theTask

-name: “Check all windows”

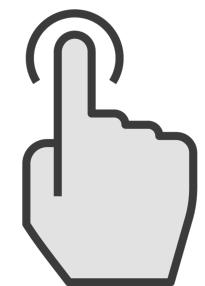
-isComplete: false

-lastCompleted: nil

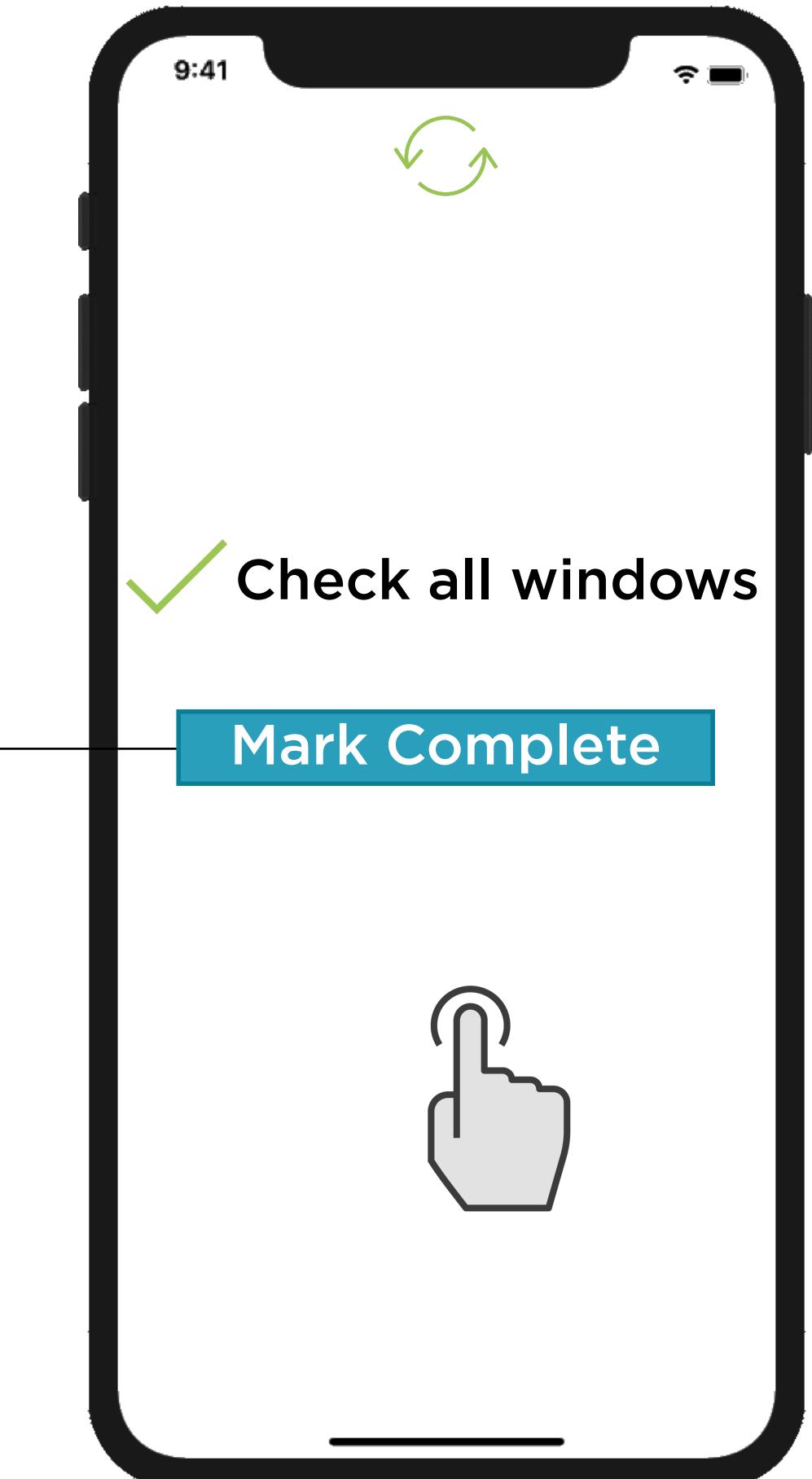
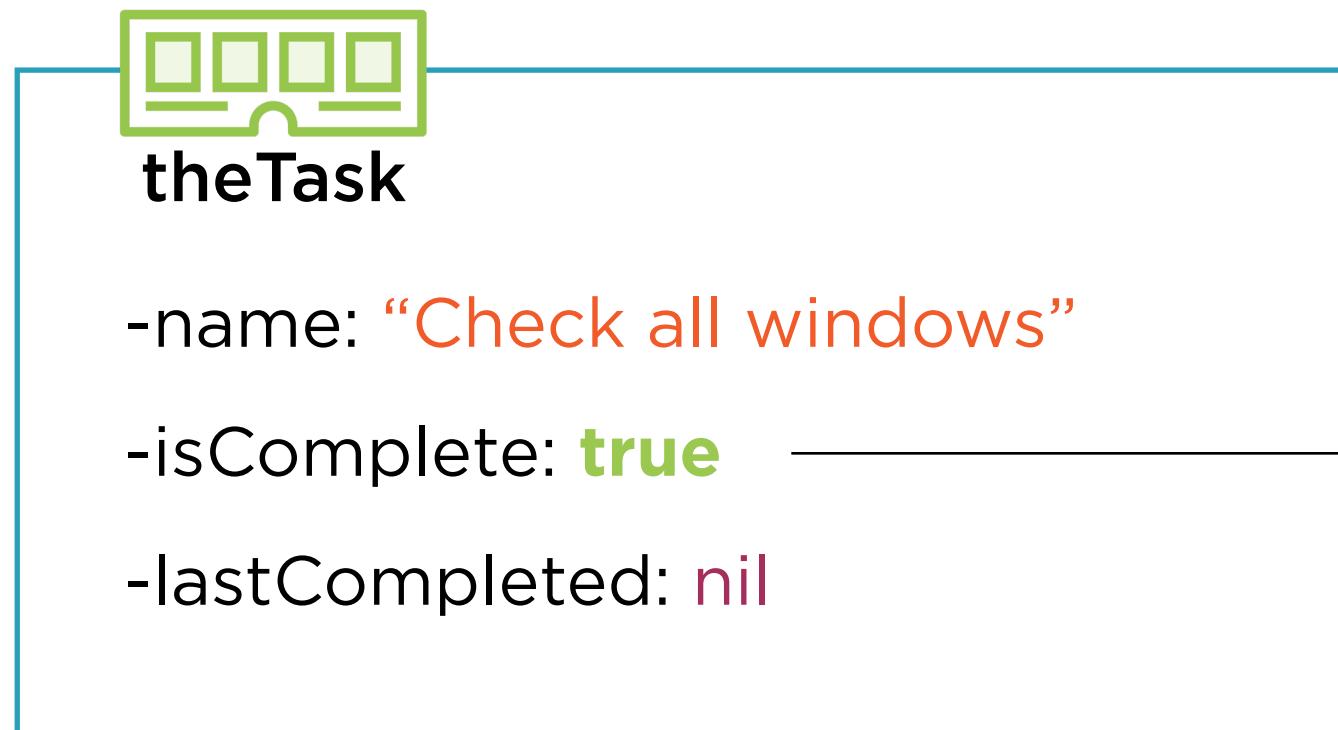
9:41

Check all windows

Mark Complete



What is the trigger that tells SwiftUI to
refresh what's on the screen?



Data Manager

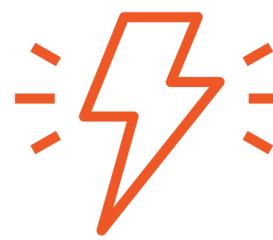
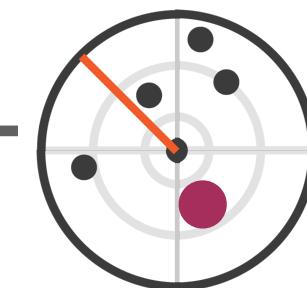


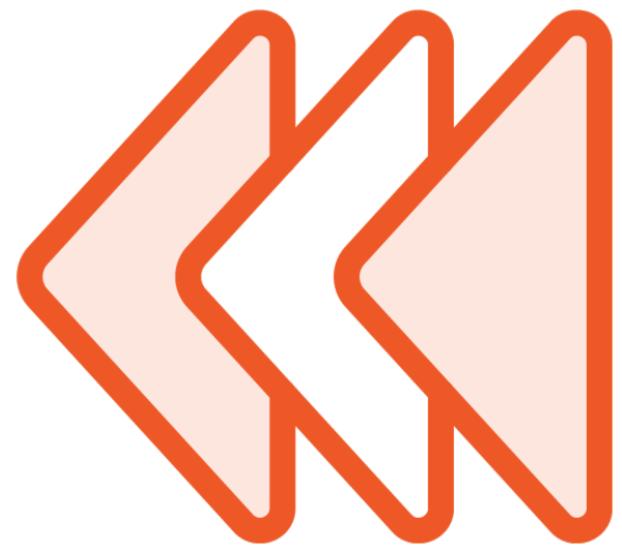
theTask

-name: "Check all windows"

-isComplete: **false**

-lastCompleted: nil





Flashback Moment

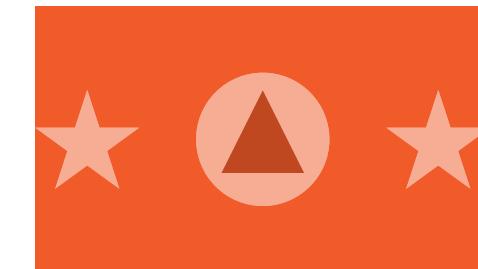


value



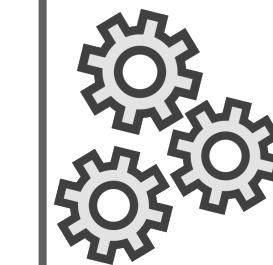
new value

Input View Instance



value

View Modifier

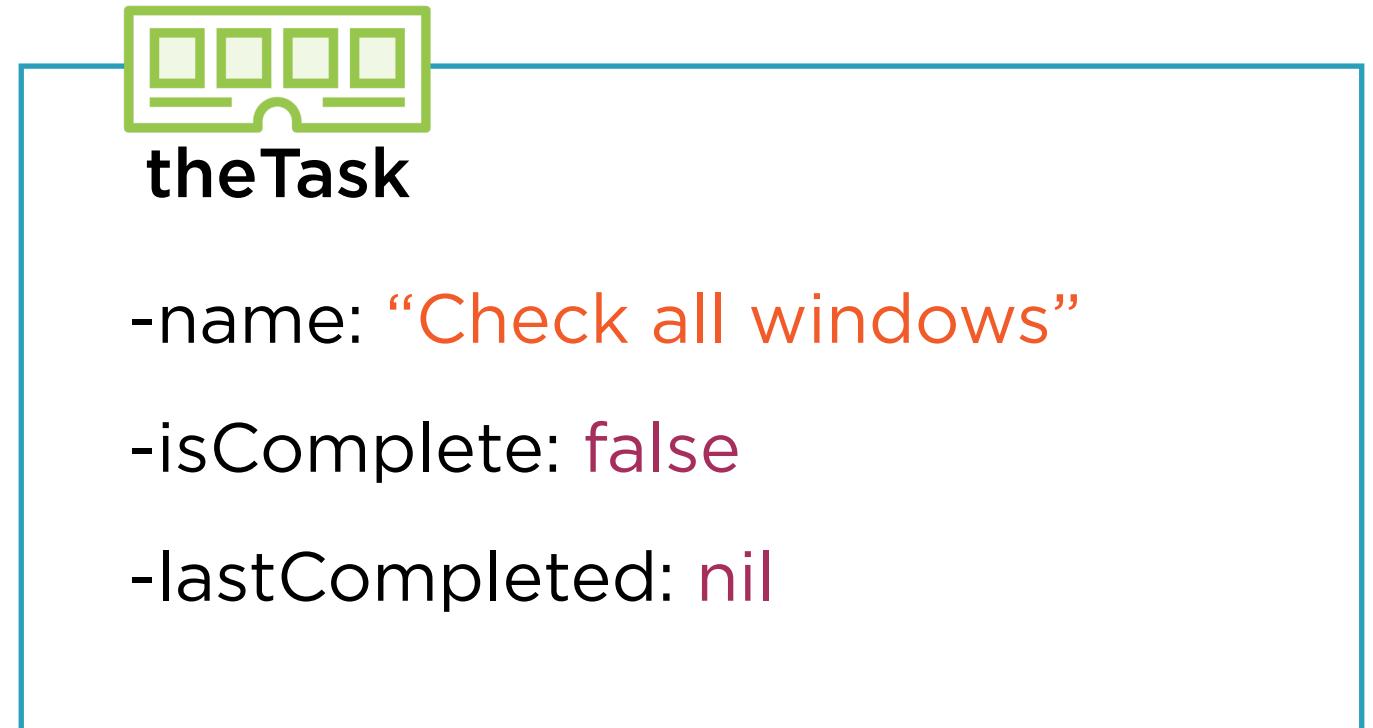


transformation

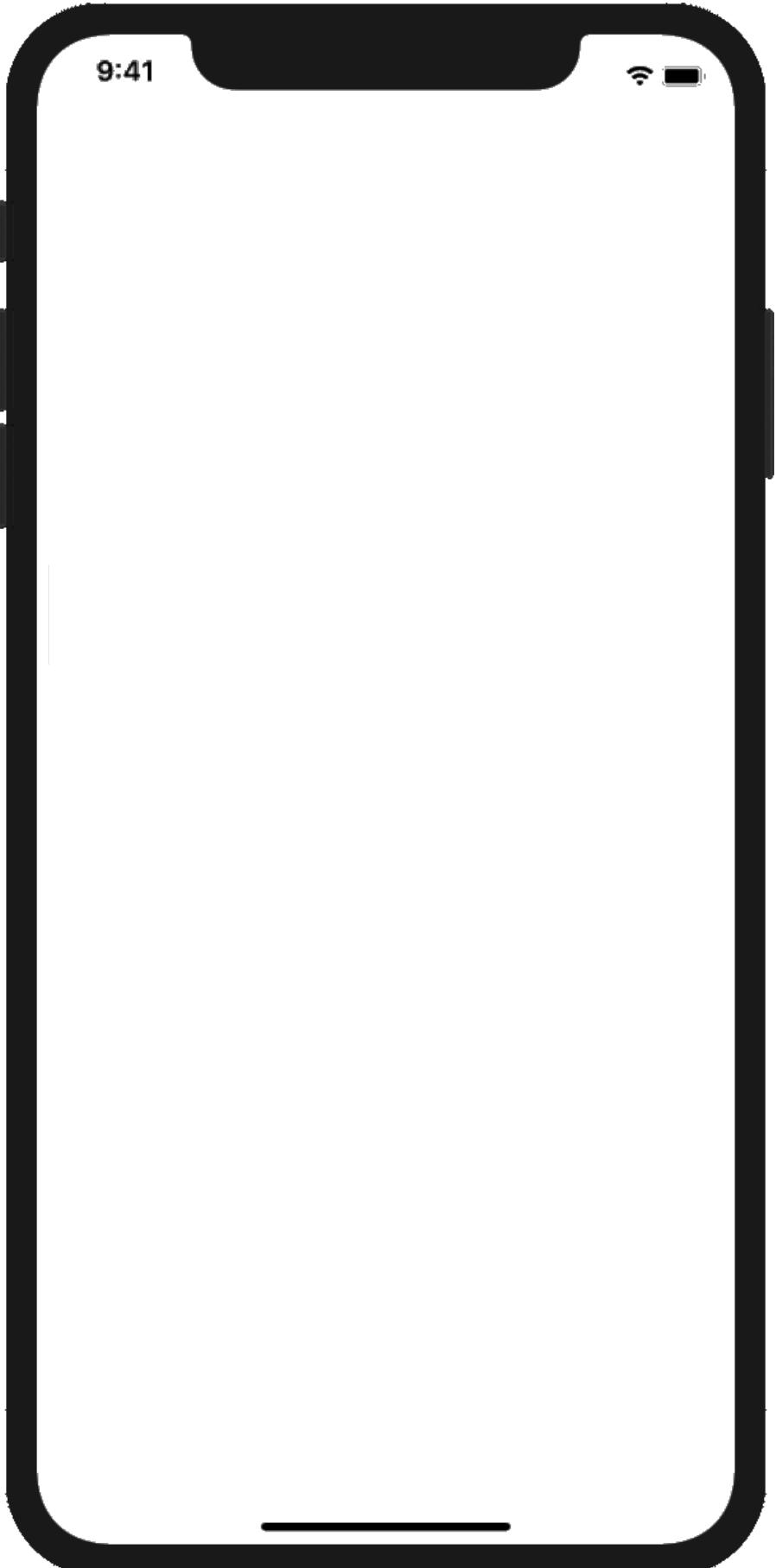
Output View Instance

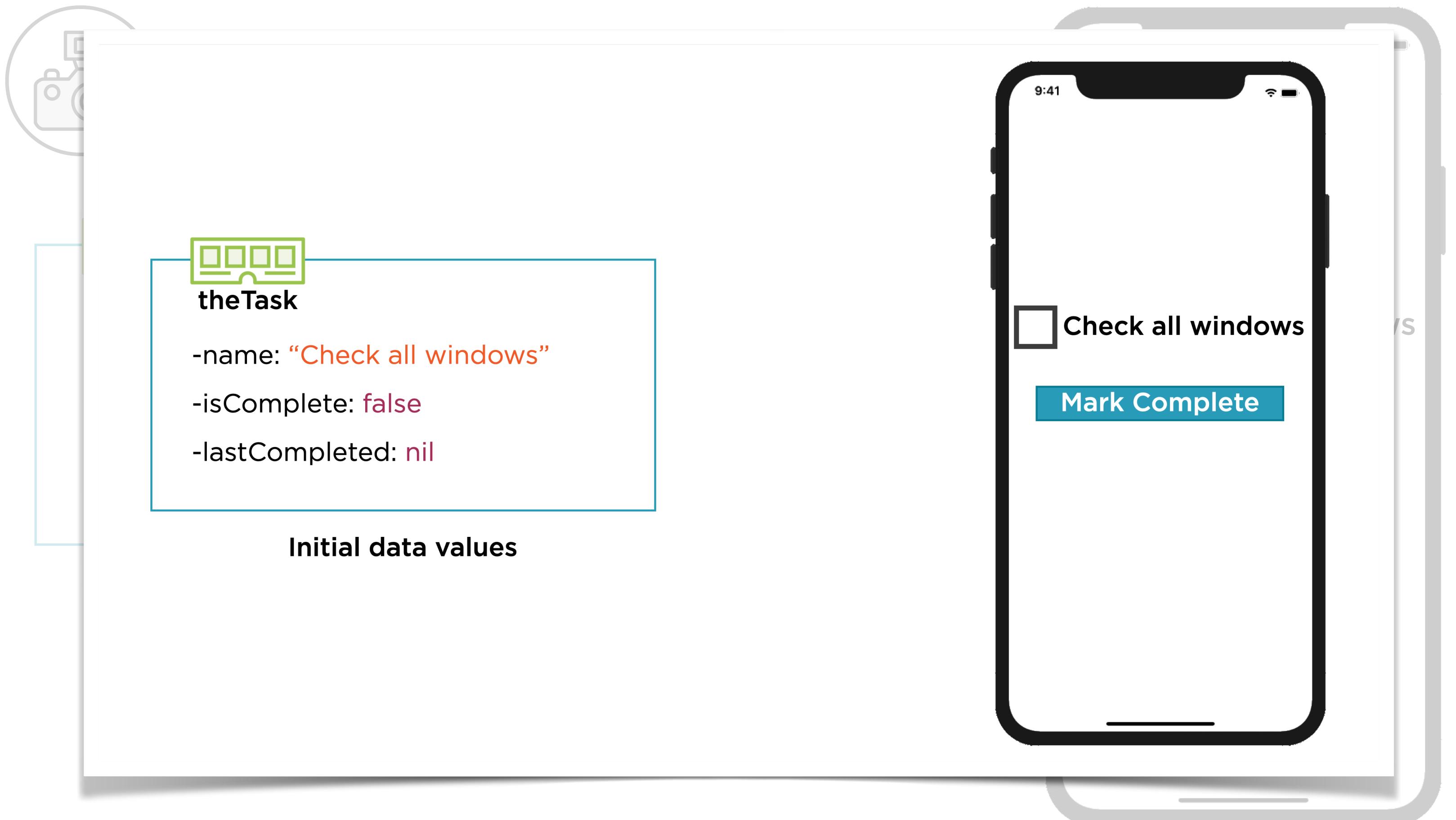


new value

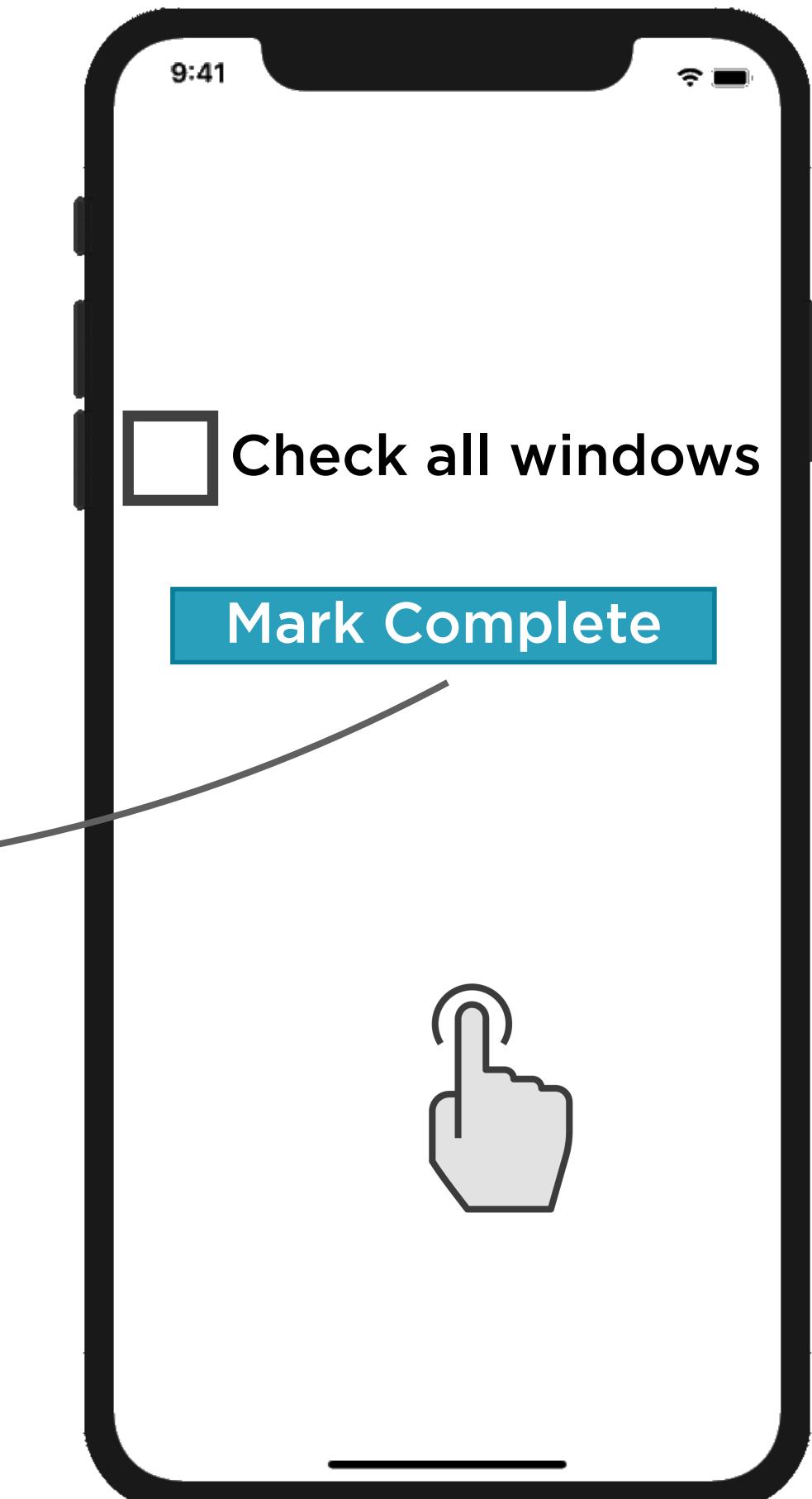
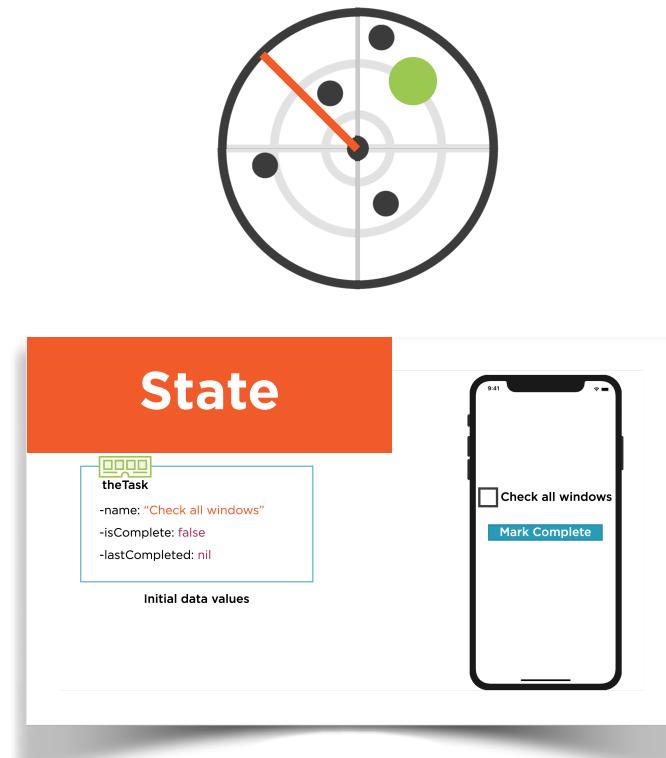
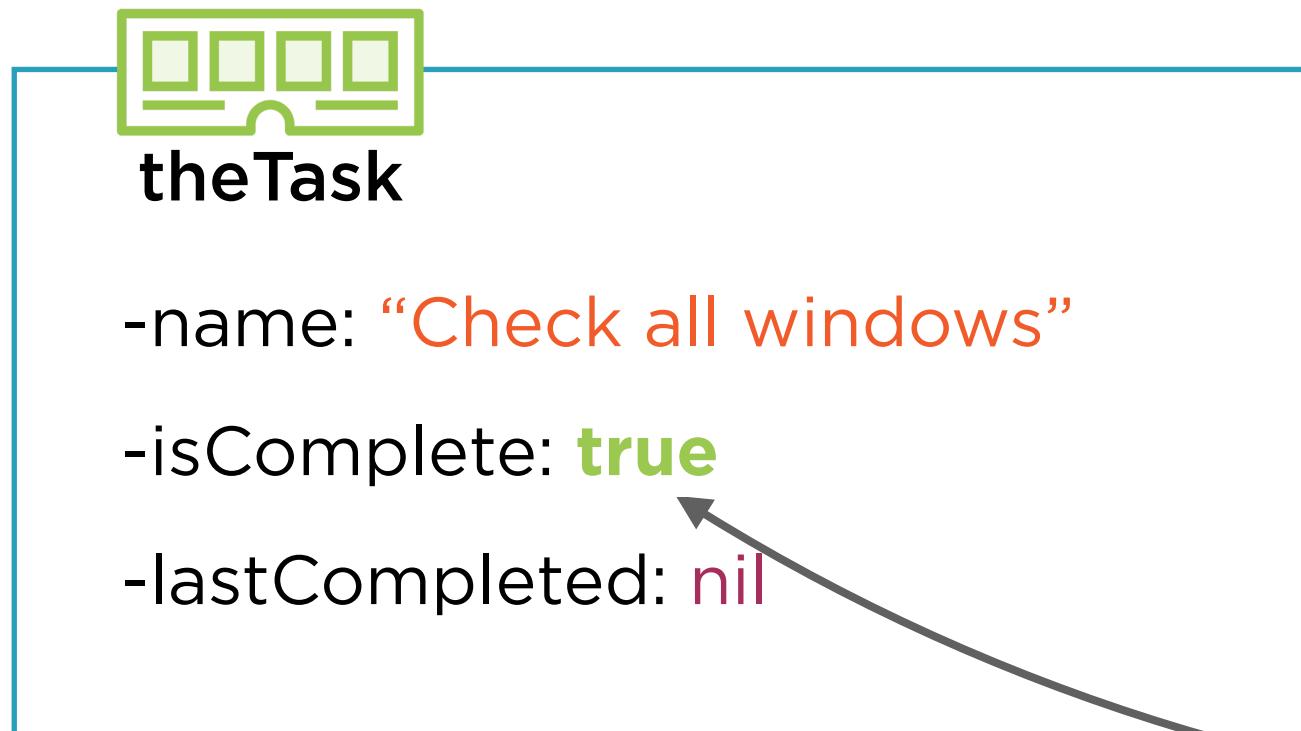


Initial data values





Data Manager

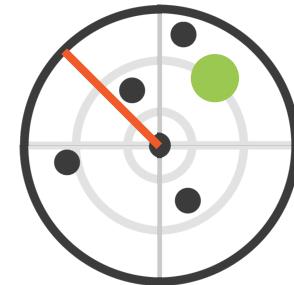


Data Manager



theTask

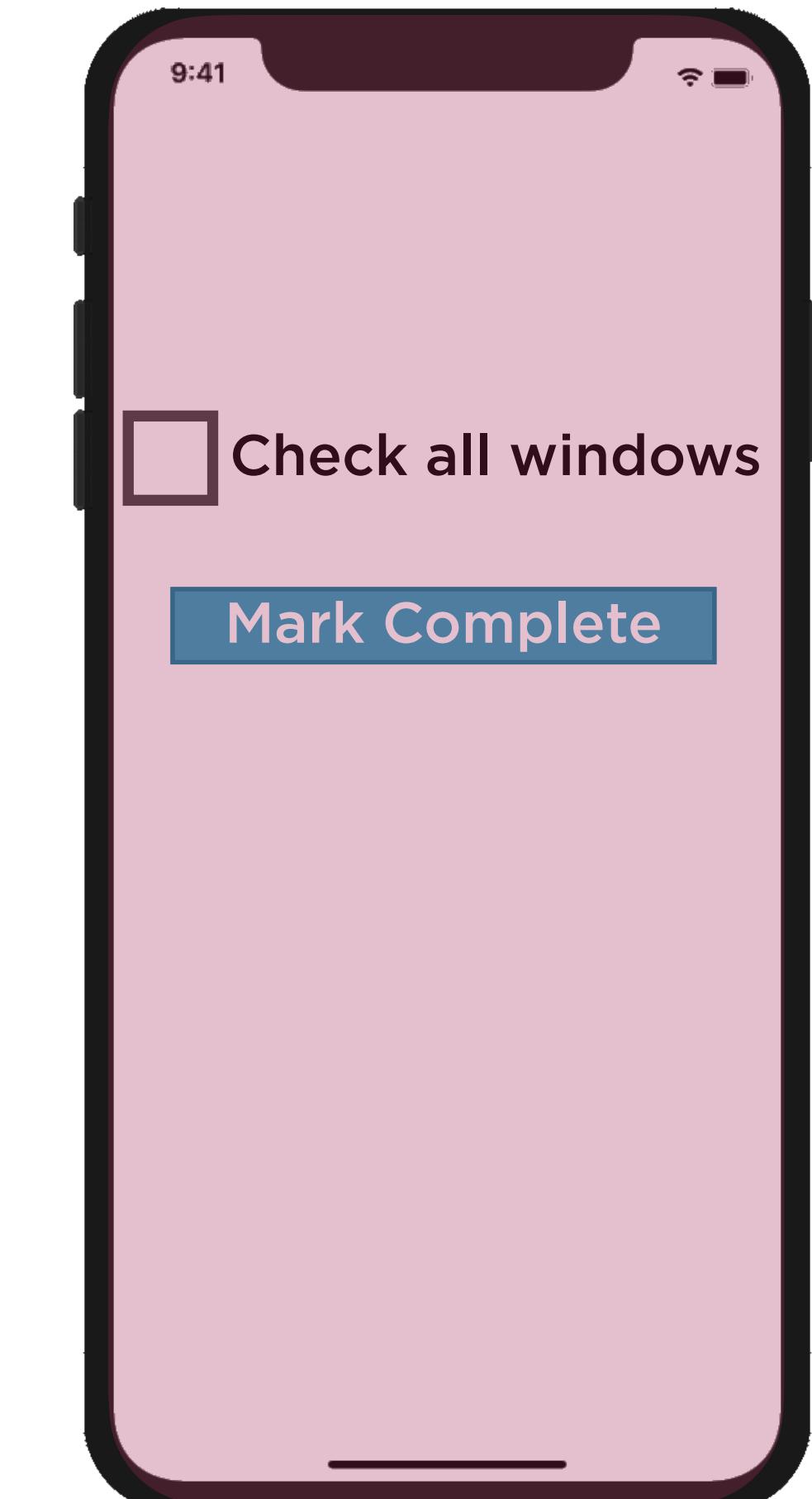
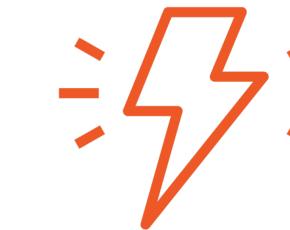
- name: "Check all windows"
- isComplete: true
- lastCompleted: nil



State

theTask
-name: "Check all windows"
-isComplete: false
-lastCompleted: nil

Initial data values

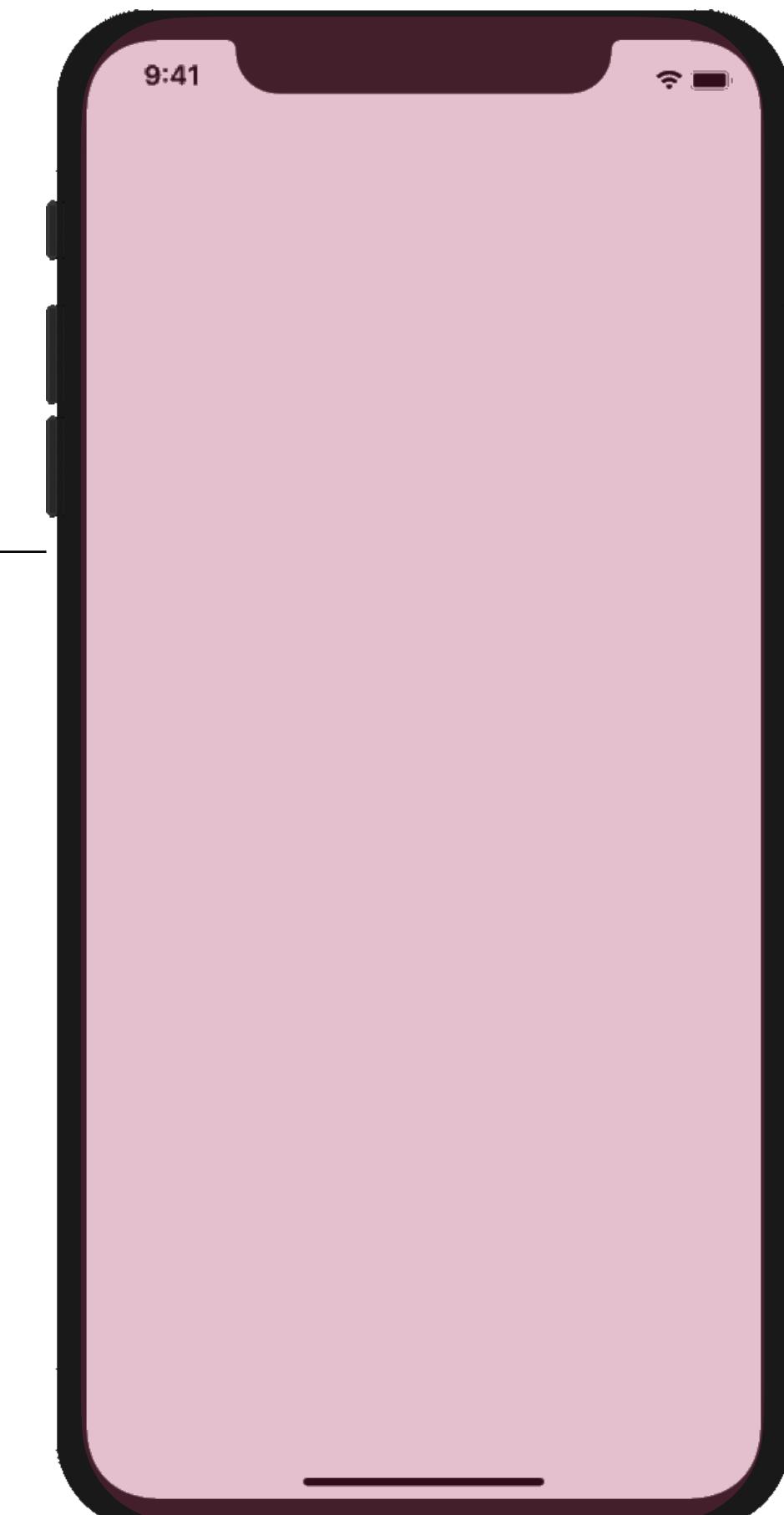
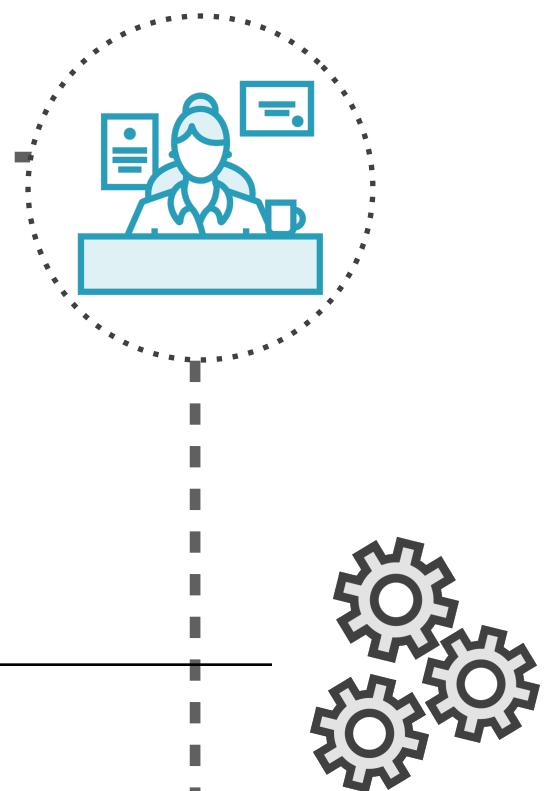
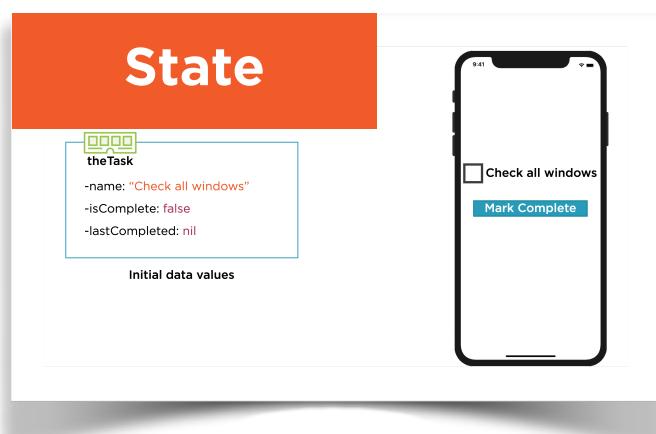


Data Manager



theTask

- name: “Check all windows”
- isComplete: **true**
- lastCompleted: **nil**



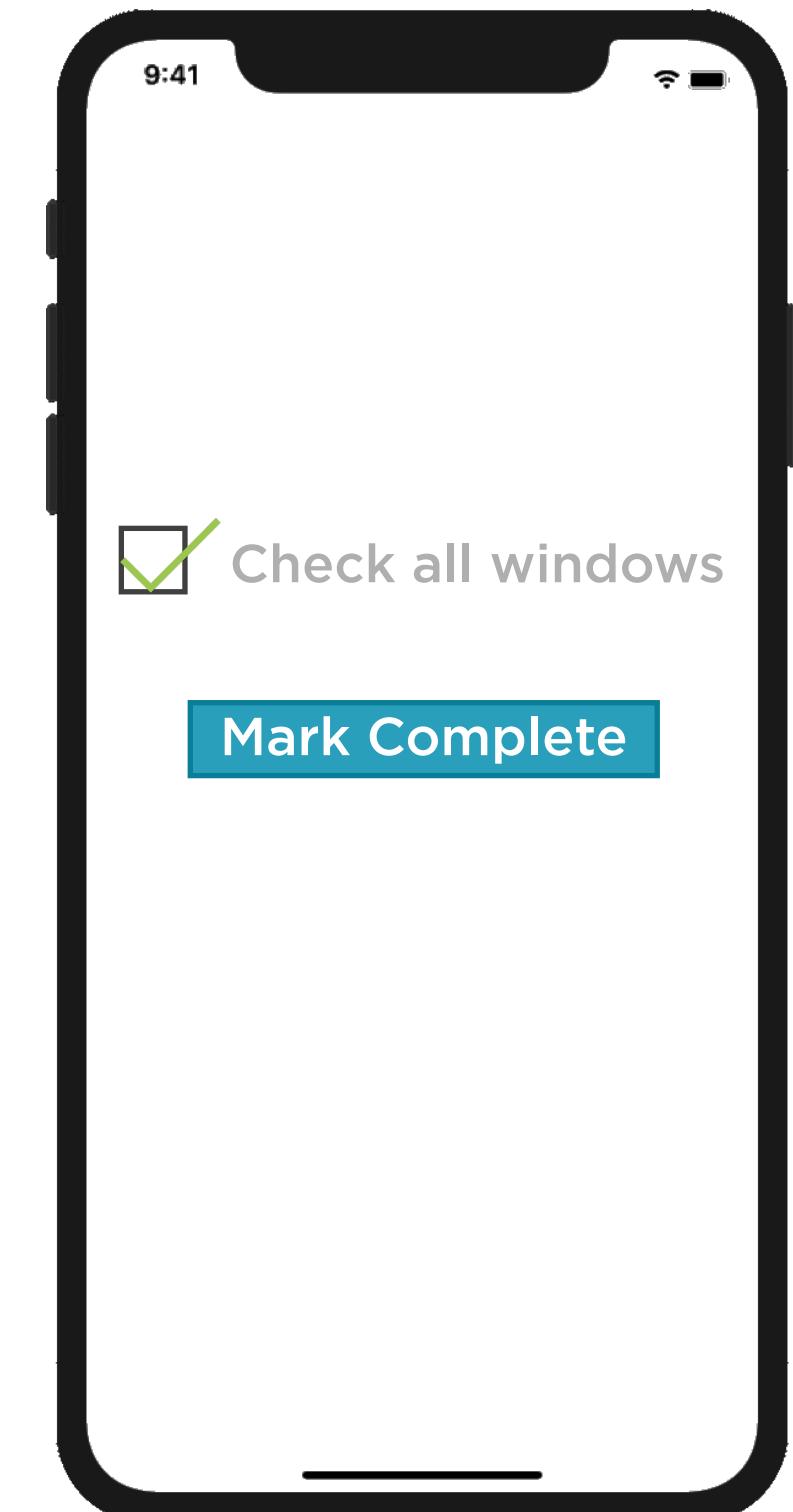
ContentView.swift

```
taskTextView.foregroundColor = .gray
```



```
checkmark.hidden = false
```

We never manipulate the View hierarchy directly
in place when we build apps with SwiftUI.



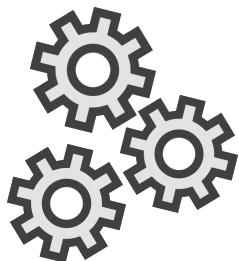
The only way for SwiftUI to
update an “immutable”
view... is to **compute a new
value for it.**

Input View Instance



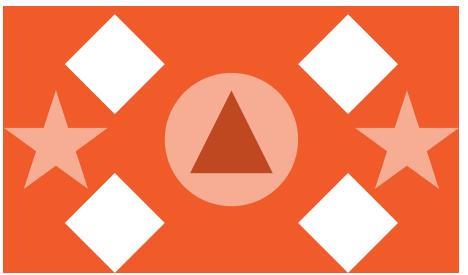
value

View Modifier



transformation

Output View Instance



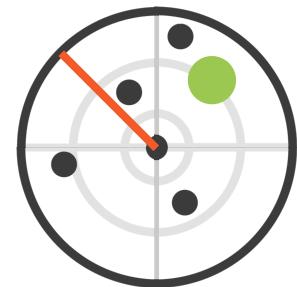
new value

Data Manager



theTask

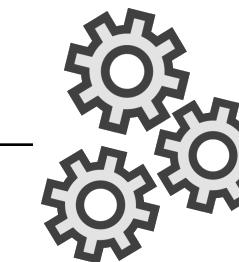
- name: "Check all windows"
- isComplete: **true**
- lastCompleted: **nil**



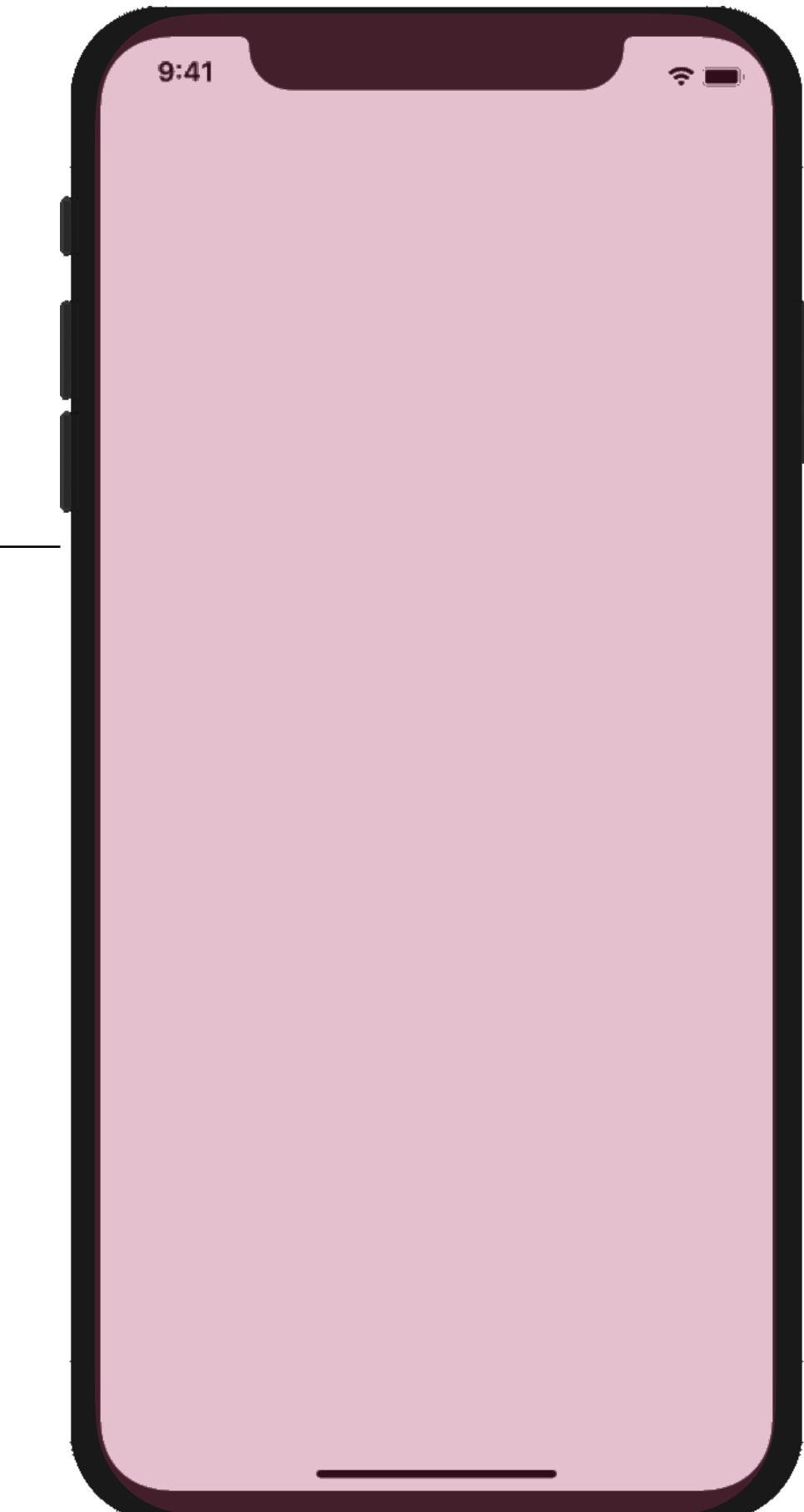
State

theTask
-name: "Check all windows"
-isComplete: **false**
-lastCompleted: **nil**

Initial data values



9:41



Data Manager

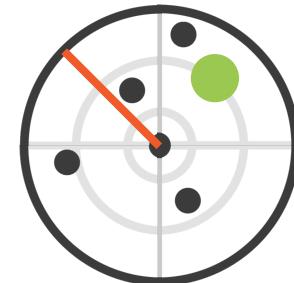


theTask

-name: "Check all windows"

-isComplete: true

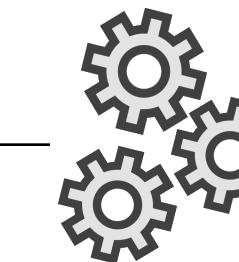
-lastCompleted: nil



State

theTask
-name: "Check all windows"
-isComplete: false
-lastCompleted: nil

Initial data values



9:41

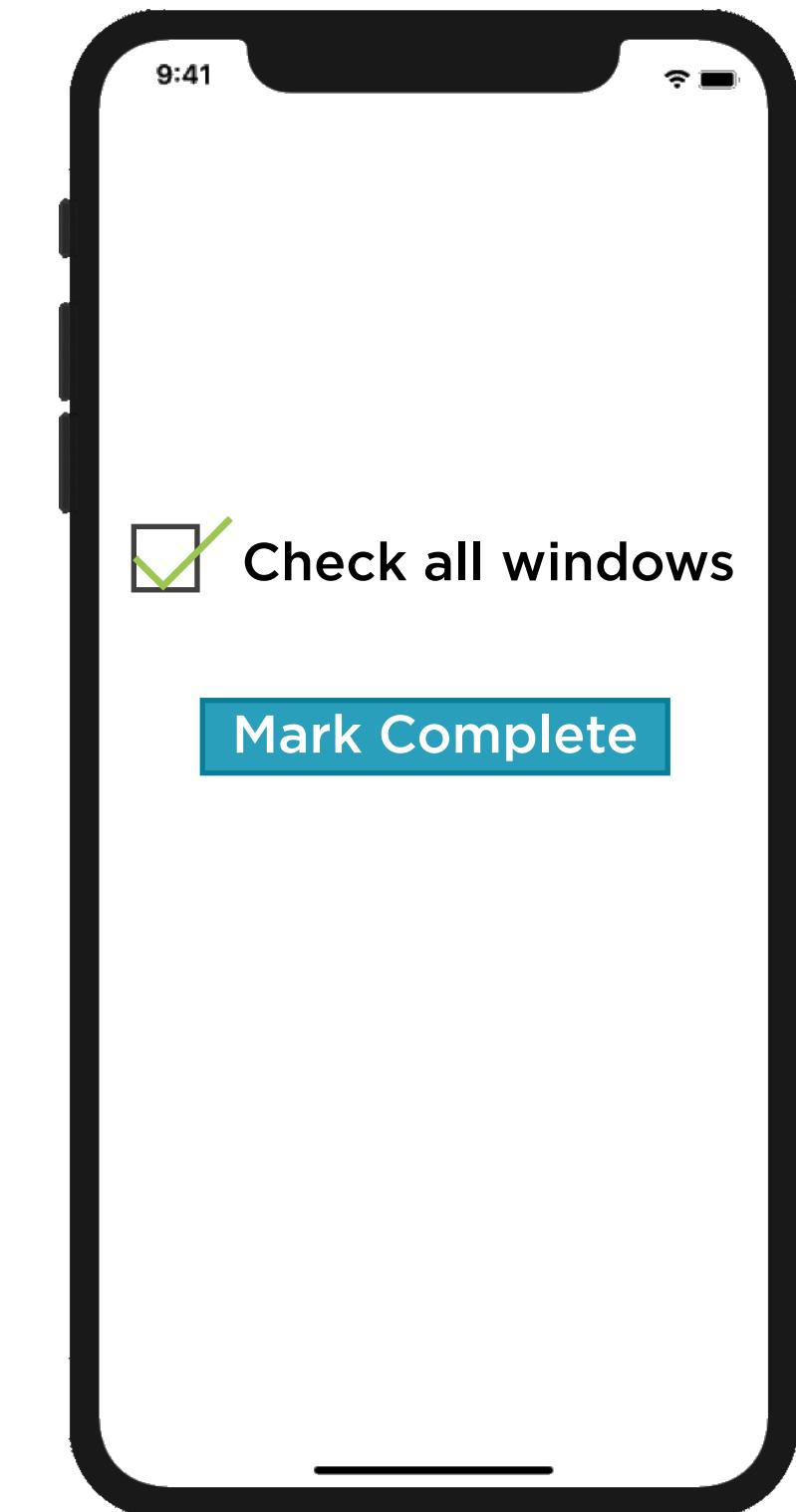


Check all windows

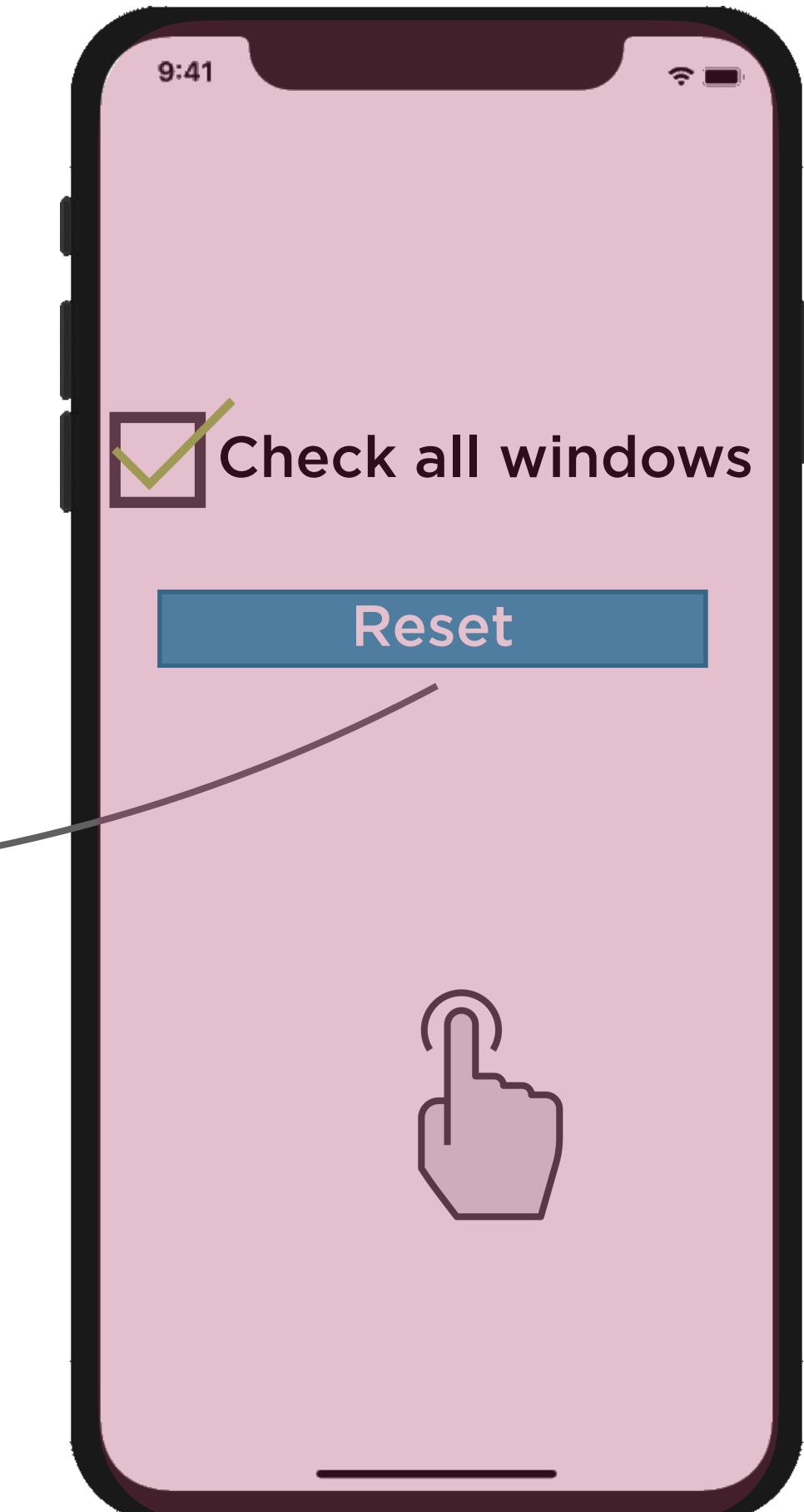
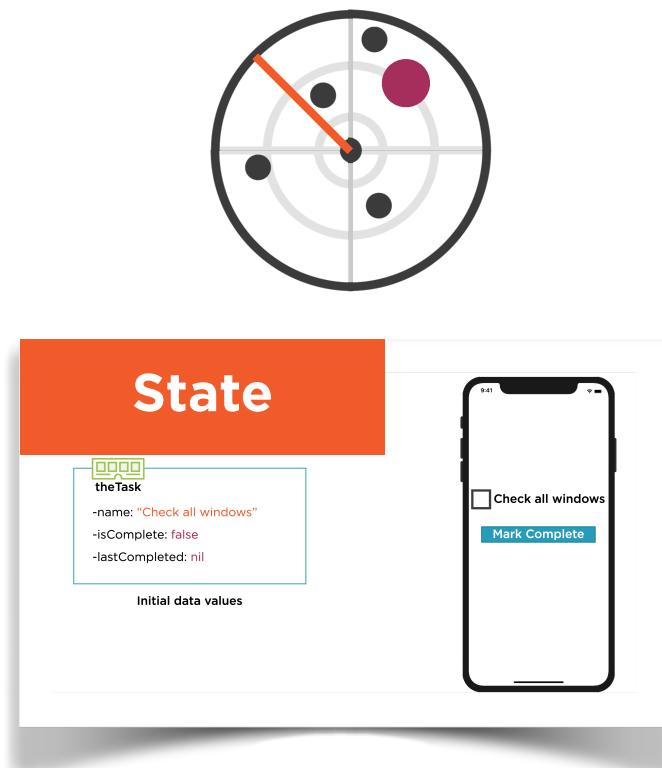
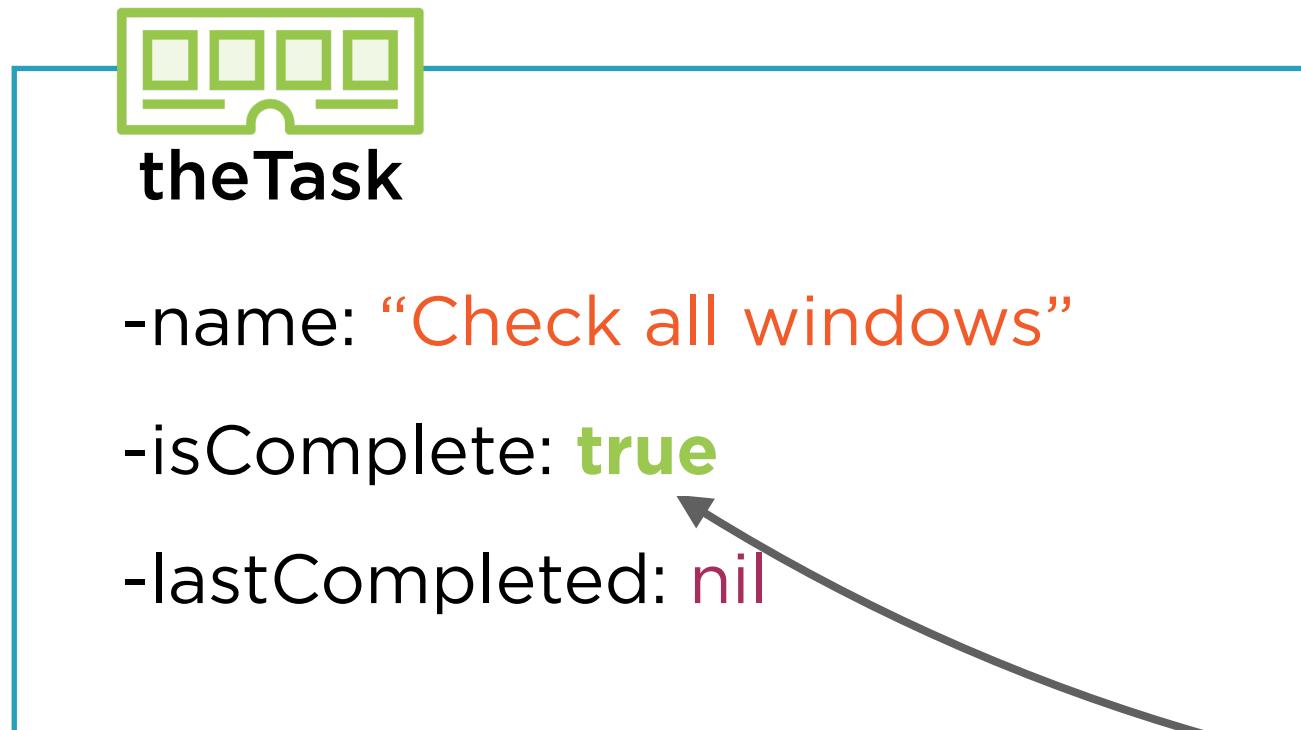
Mark Complete

ContentView.swift

```
if theTask.isComplete == false {  
    HStack {  
        Image(systemName: "square")  
        Text("Check all windows")  
    }  
} else {  
    HStack {  
        Image(systemName: "checkmark.square")  
        Text("Check all windows")  
    }  
}  
:  
:
```



Data Manager

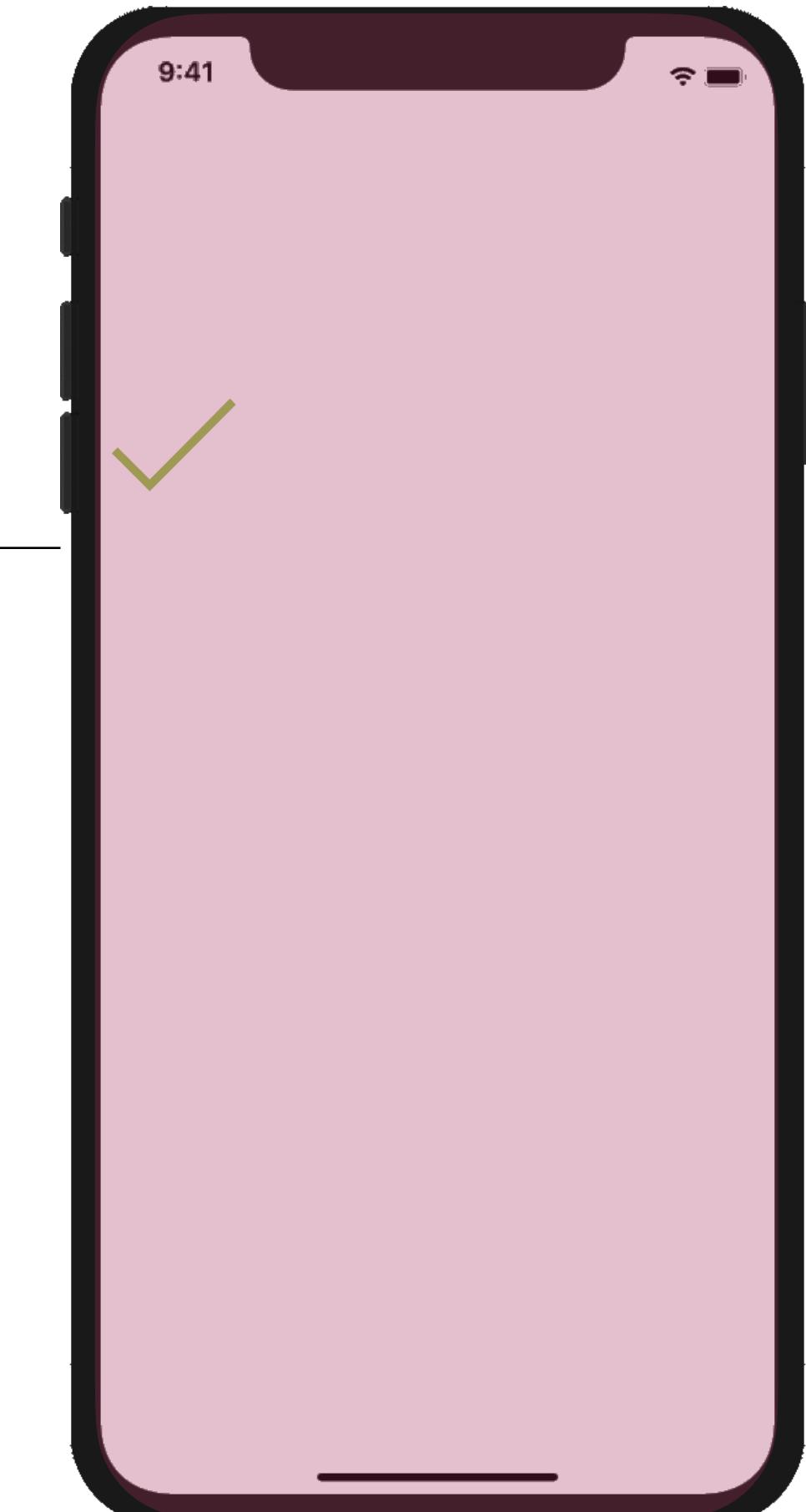
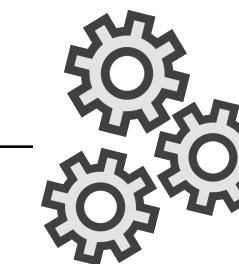
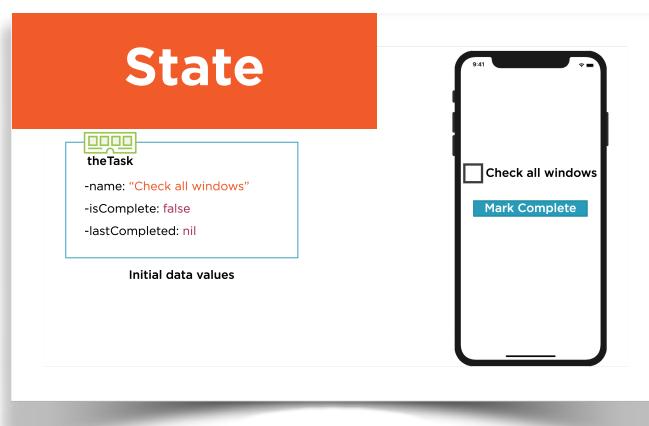
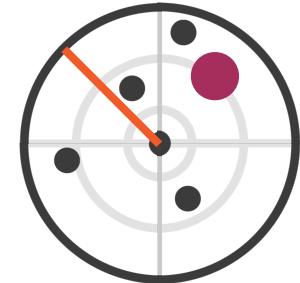


Data Manager



theTask

- name: "Check all windows"
- isComplete: false
- lastCompleted: nil

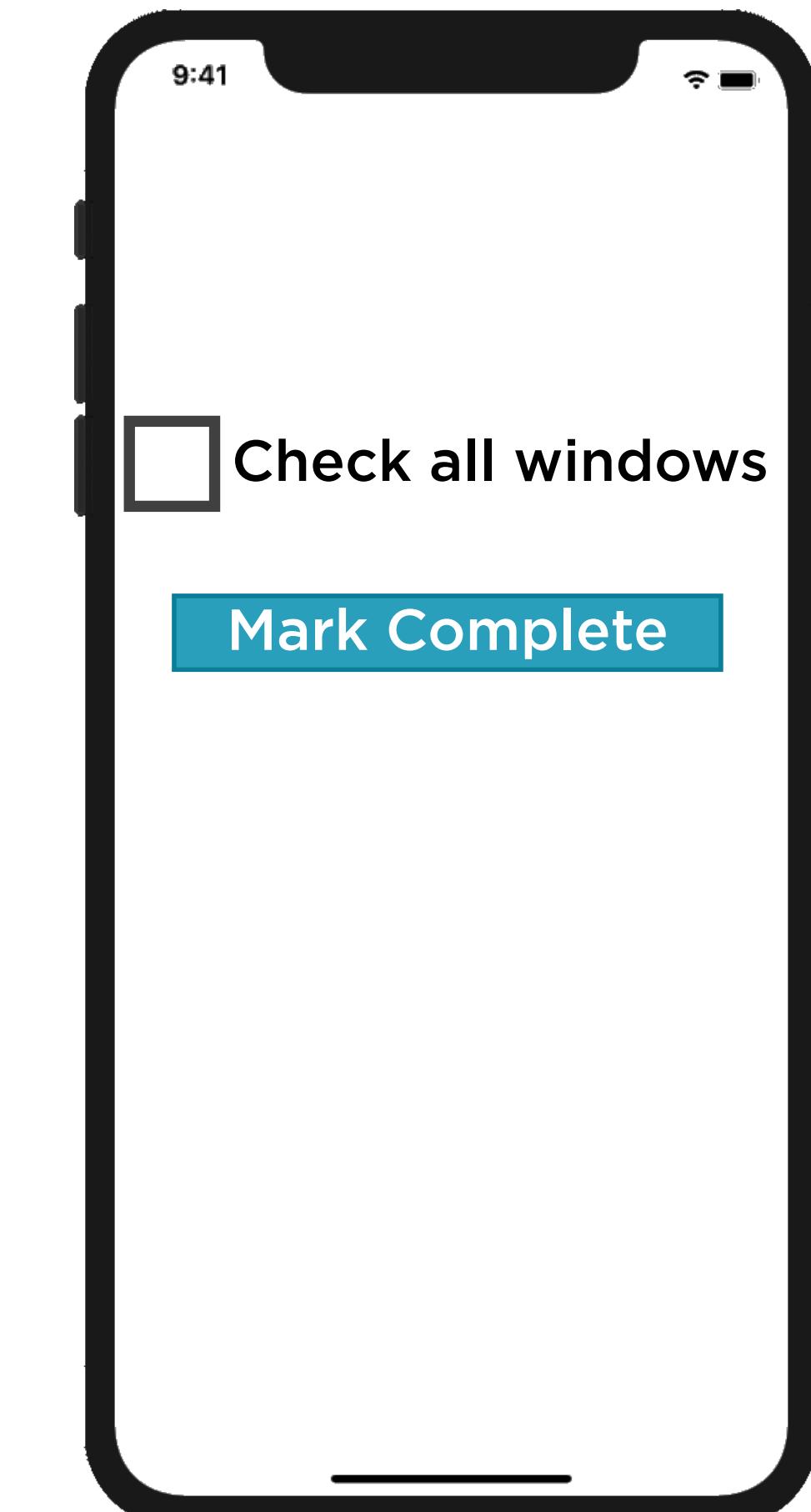
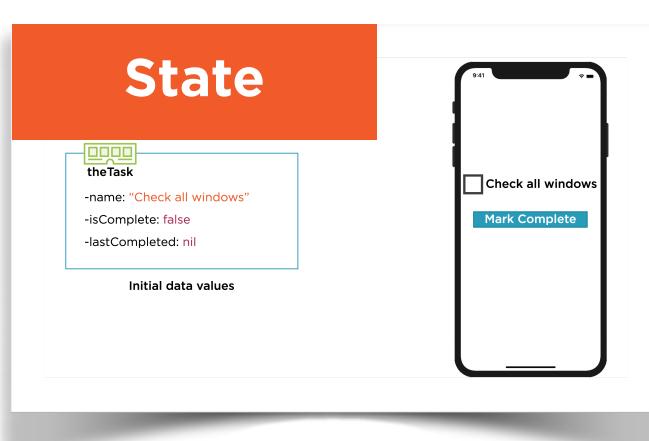
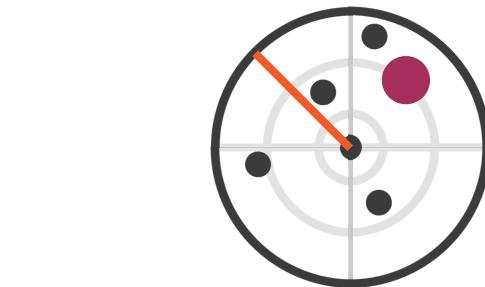


Data Manager



theTask

- name: "Check all windows"
- isComplete: false
- lastCompleted: nil



Data Manager



What are these SwiftUI “Data Managers”?

Adding Basic Behavior

Data Manager



theTask

- name: "Check all windows"
- isComplete: true
- lastCompleted: nil



Data Manager



theTask

-name: "Check all windows"

-isComplete: **true**

-lastCompleted: **nil**



false

Data Manager

Wrap

Property

State

Data Manager

Property

Wrapper

State

-- Property Wrapper --



theTask

- name: “Check all windows”
- isComplete: **true**
- lastCompleted: **nil**



-- Property Wrapper --

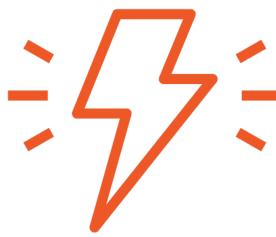
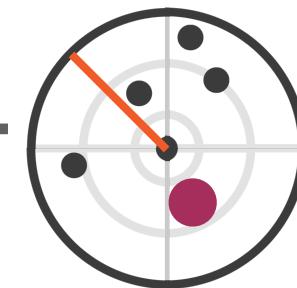


theTask

-name: "Check all windows"

-isComplete: **false**

-lastCompleted: **nil**



Data Manager

Property

Wrapper

@State



theTask

- name: “Check all windows”
- isComplete: **false**
- lastCompleted: **nil**

@State

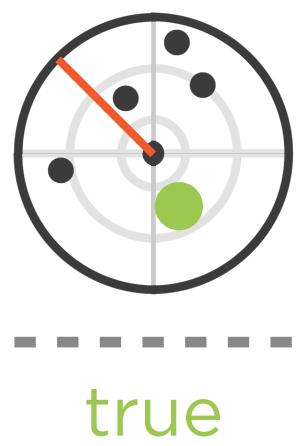


theTask

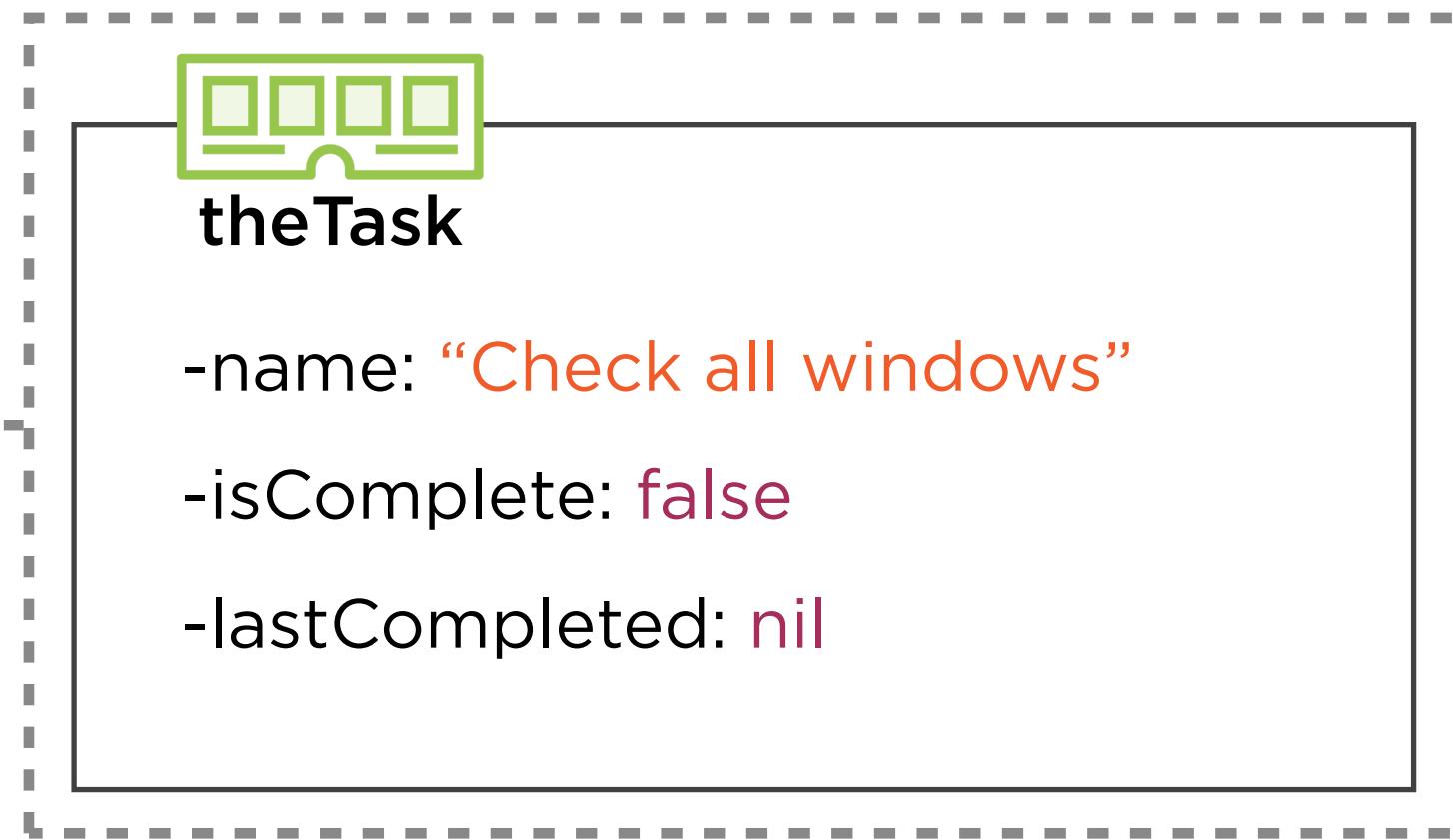
- name: “Check all windows”
- isComplete: **false**
- lastCompleted: **nil**



@State

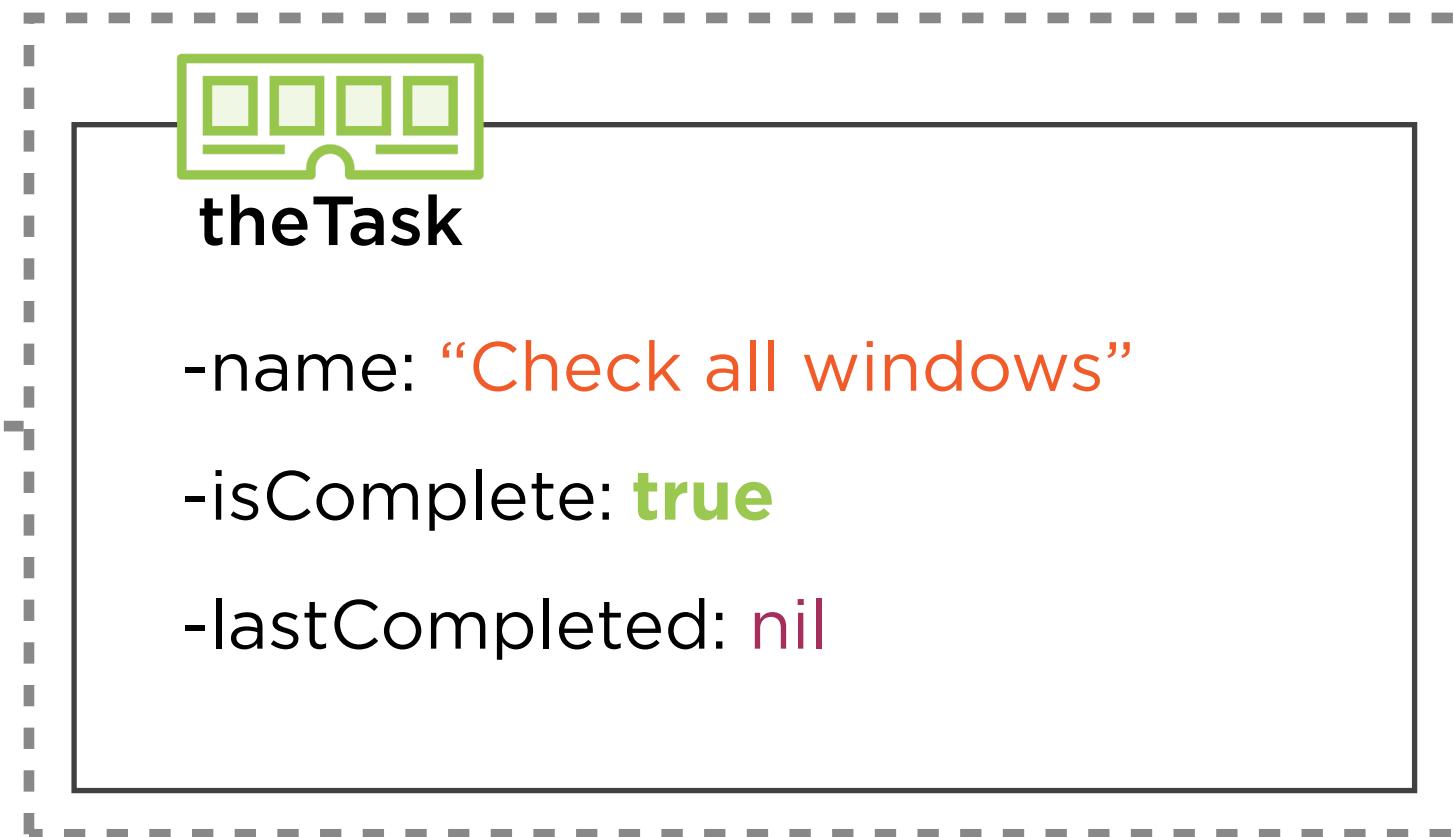


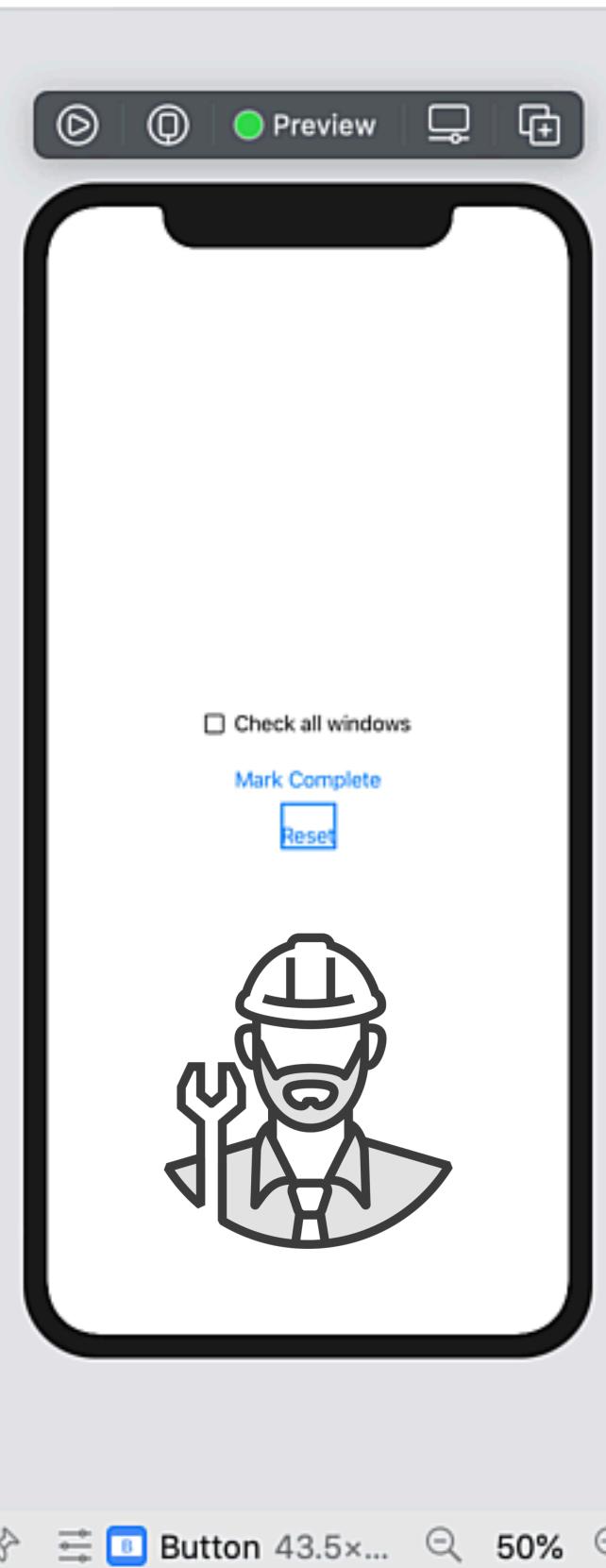
true





@State





Connecting a Subview to a Parent View's Data

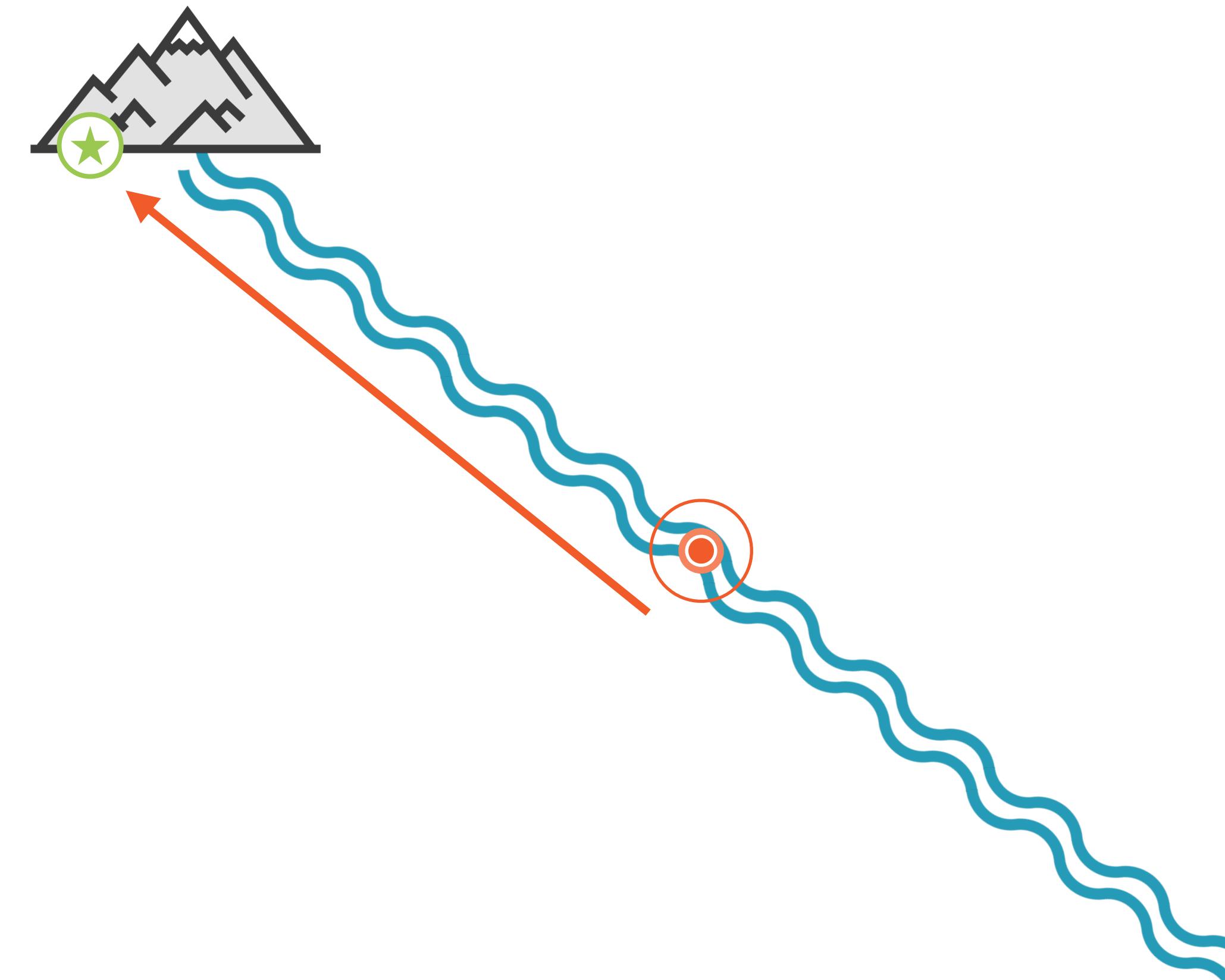
View's #1 Job



Accurately **represent** the **data** of our application
in what it displays on the screen.

All data has a source.







Home



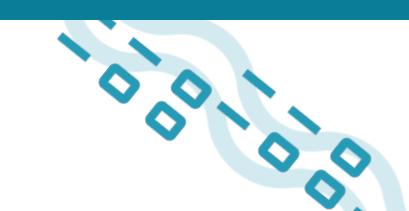
Details



Specifics



Distinctives



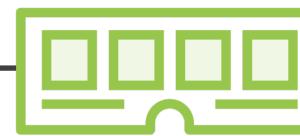


theTask

-name: "Check all windows"

-isComplete: false

-lastCompleted: nil



theTask

-name: "Check all windows"

-isComplete: false

-lastCompleted: nil



Which task?



theTask

-name: “Check all windows”

-isComplete: **false**

-lastCompleted: **nil**

ContentView



theTask

-name: “Check all windows”

-isComplete: **false**

-lastCompleted: **nil**

ControlPanel



Which task?



theTask

- name: “Check all windows”
- isComplete: false
- lastCompleted: nil

ContentView



theTask

- name: “Check all windows”
- isComplete: false
- lastCompleted: nil

ControlPanel



theTask

- name: “Check all windows”
- isComplete: **true**
- lastCompleted: **nil**

ControlPanel



How do we work with data models that are **classes** in SwiftUI?

Using Classes as Data Models

@State

Property Wrapper



theTask

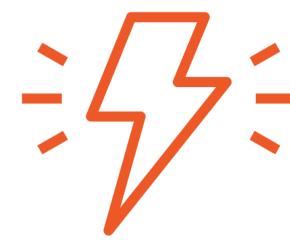
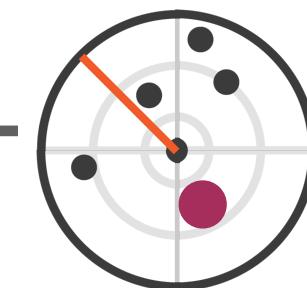
-name: "Check all windows"

-isComplete: **false**

-lastCompleted: **nil**



@Binding



Everything changes when
we switch from using a
struct, to **using a class** to
model our data.

@StateObject



theTask

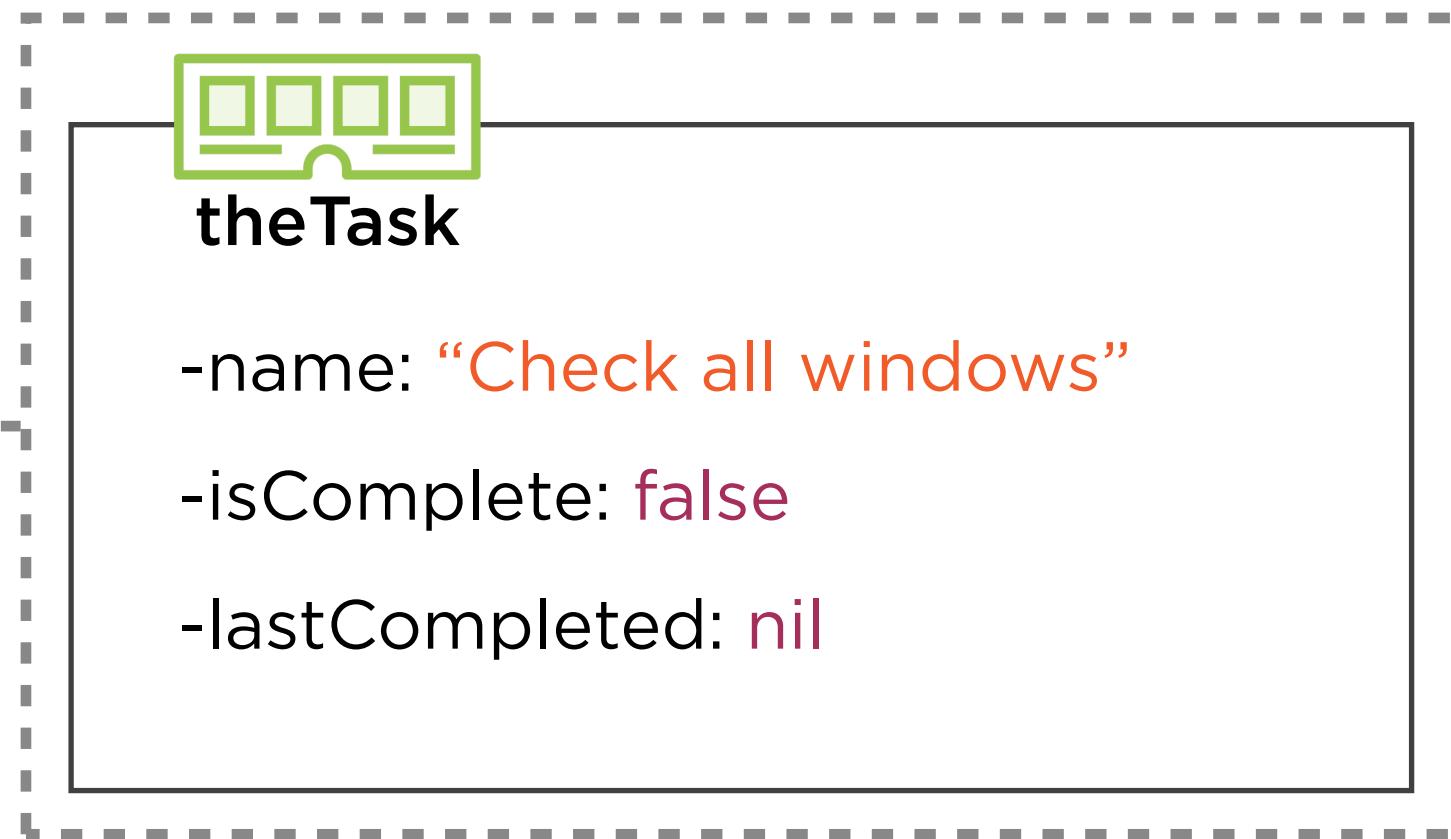
- name: “Check all windows”
- isComplete: **false**
- lastCompleted: **nil**



@StateObject

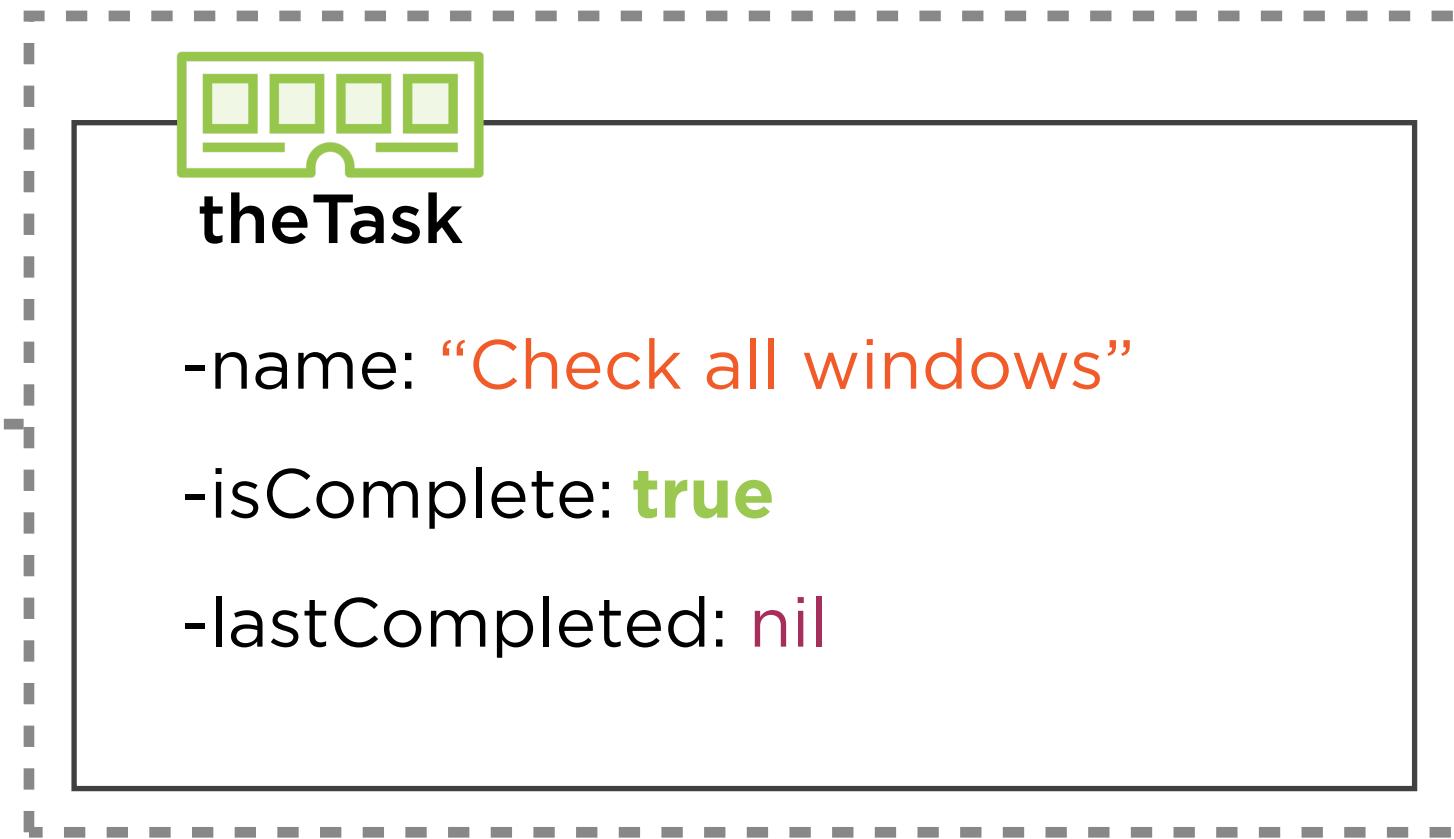


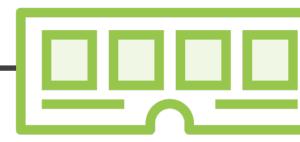
true





@StateObject





theTask

Struct

- name: “Check all windows”
- isComplete: false
- lastCompleted: nil



theTask

Struct

- name: “Check all windows”
- isComplete: false
- lastCompleted: nil

ContentView

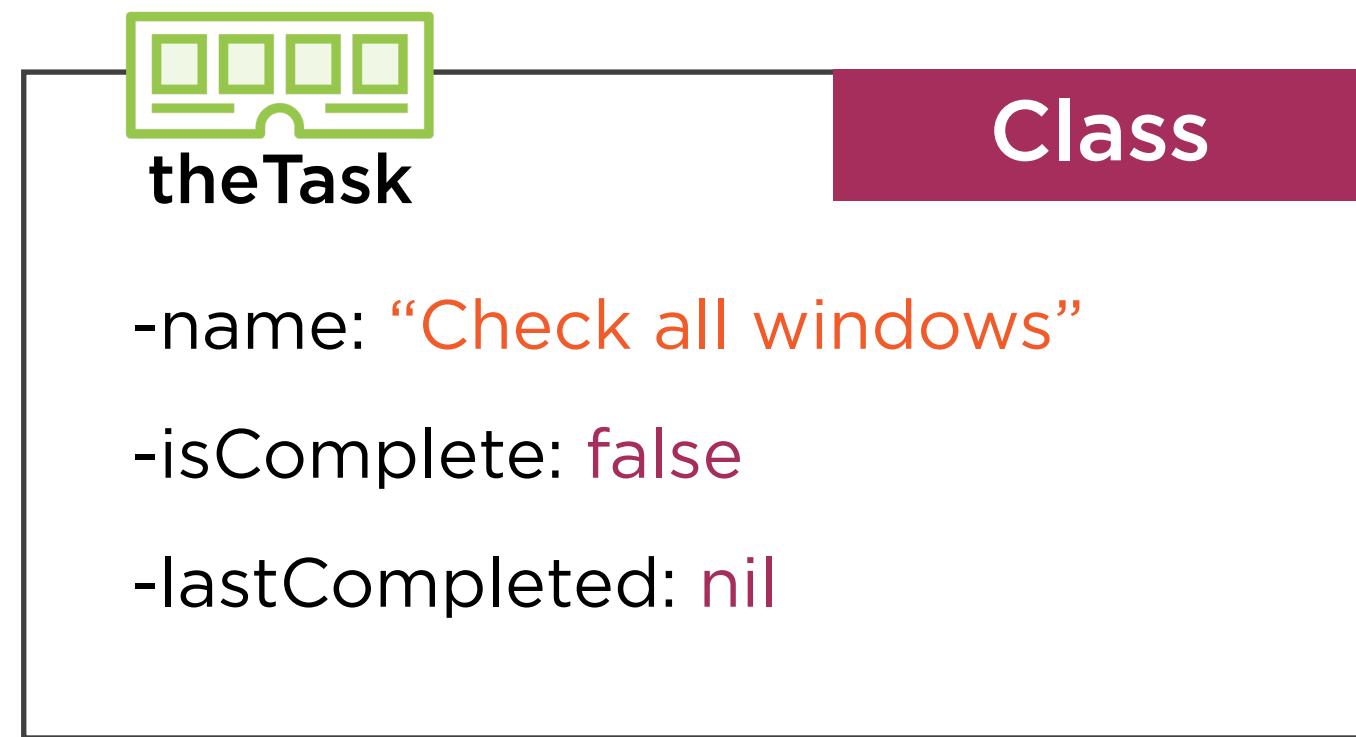


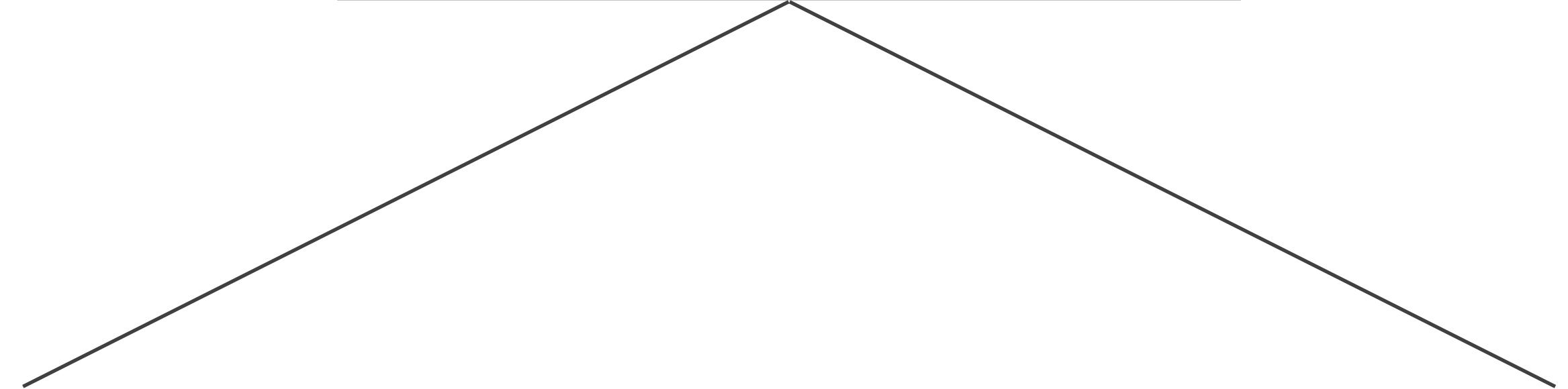
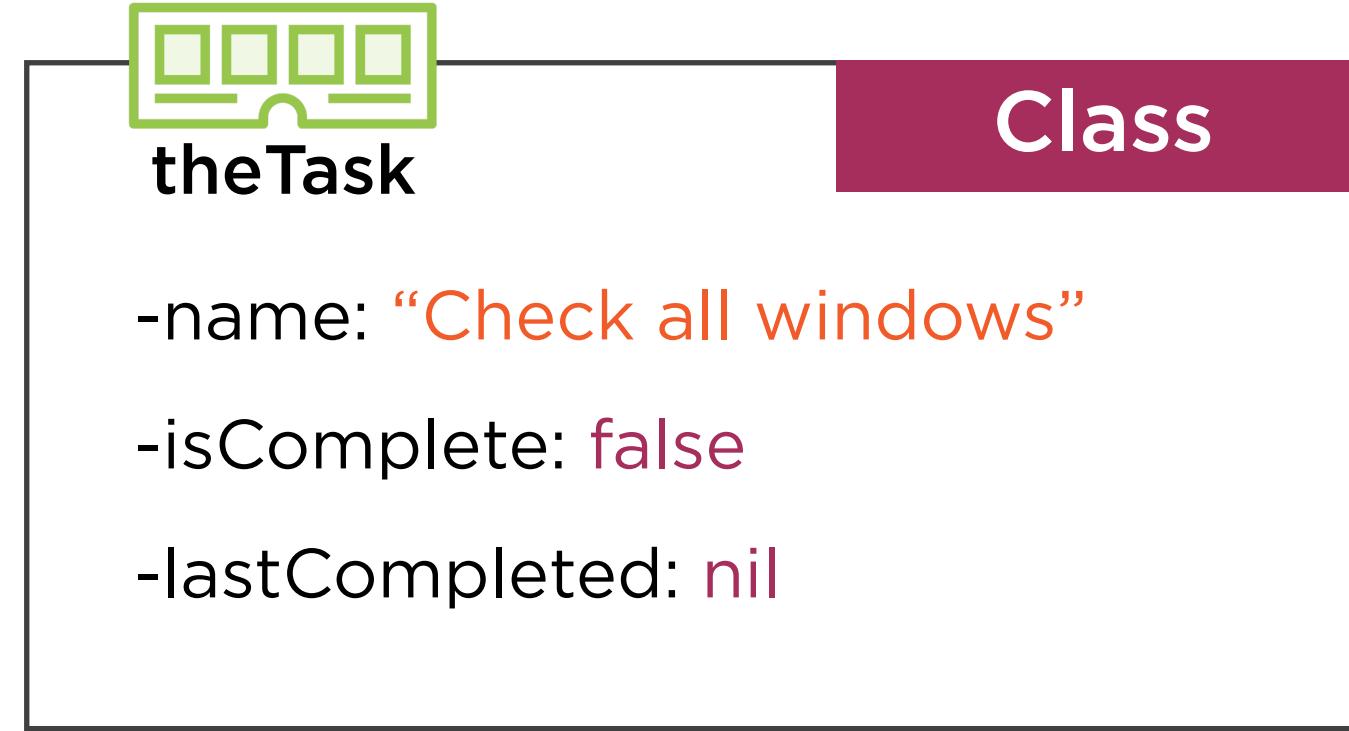
theTask

Struct

- name: “Check all windows”
- isComplete: false
- lastCompleted: nil

ControlPanel





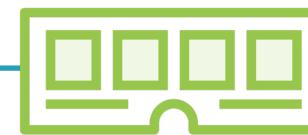
ContentView

ControlPanel

Reference Type (Class)
Semantics

@State

Property Wrapper



theTask

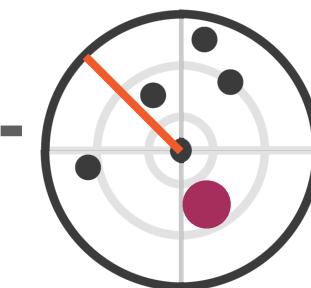
-name: "Check all windows"

-isComplete: **false**

-lastCompleted: **nil**

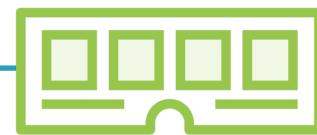


@Binding



Value Types (Structs)

Property Wrapper



theTask

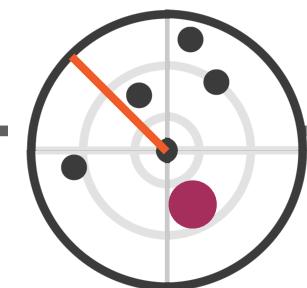
-name: "Check all windows"

-isComplete: **false**

-lastCompleted: nil

@StateObject

@ObservedObject



Reference Types (Classes)

One more **Property Wrapper**



Sharing a Data Model through the SwiftUI Environment

- Property Wrapper



theTask

- name: “Check all windows”
- isComplete: **false**
- lastCompleted: **nil**

@State

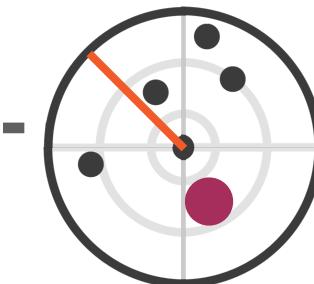
@Binding

Value Types (Structs)



@StateObject

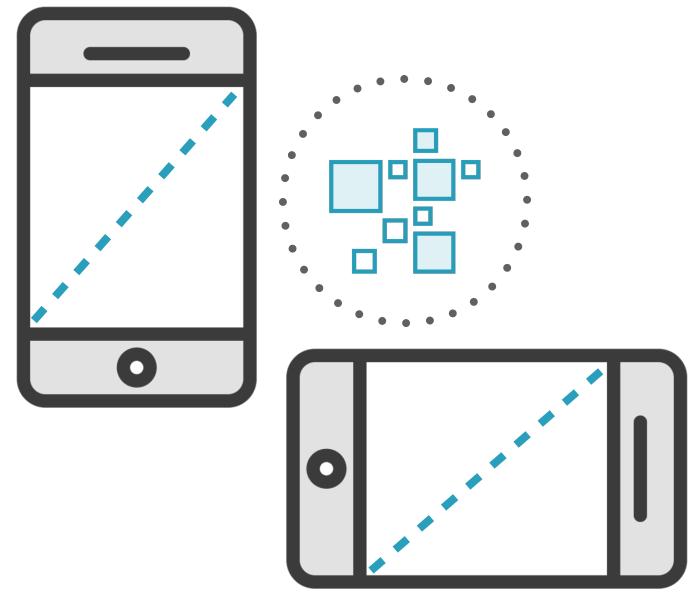
@ObservedObject



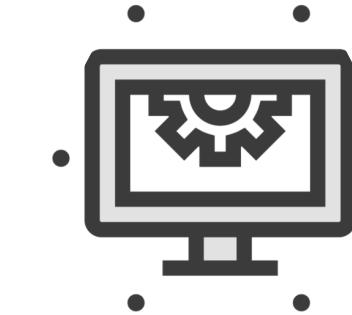
Reference Types (Classes)

All SwiftUI Views exist within
an **Environment**.

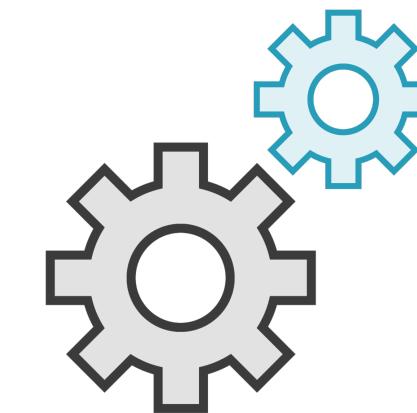
SwiftUI Environment



Device characteristics

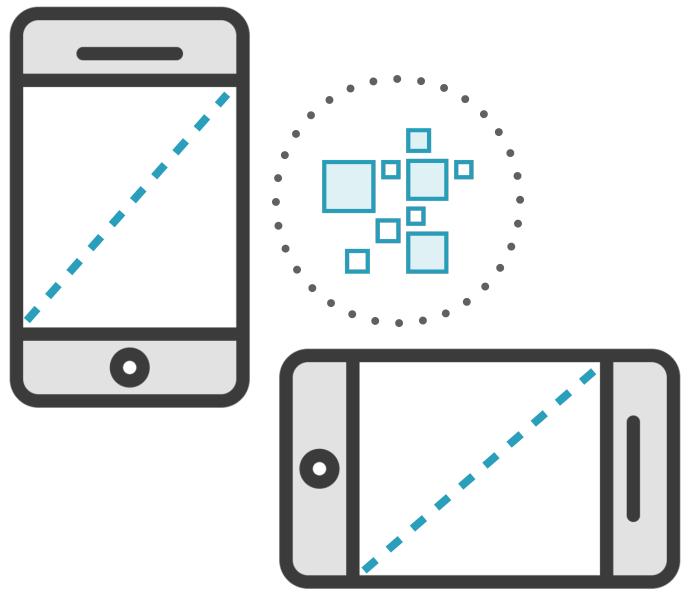


Operating system states

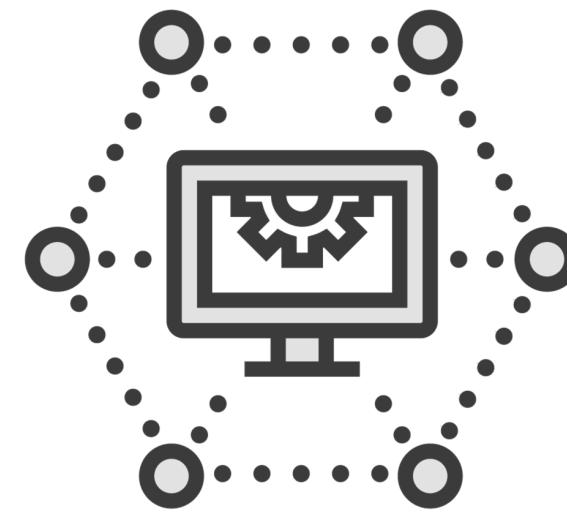


User settings

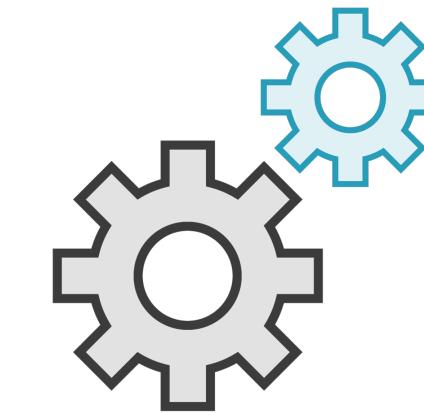
SwiftUI Environment



Device characteristics

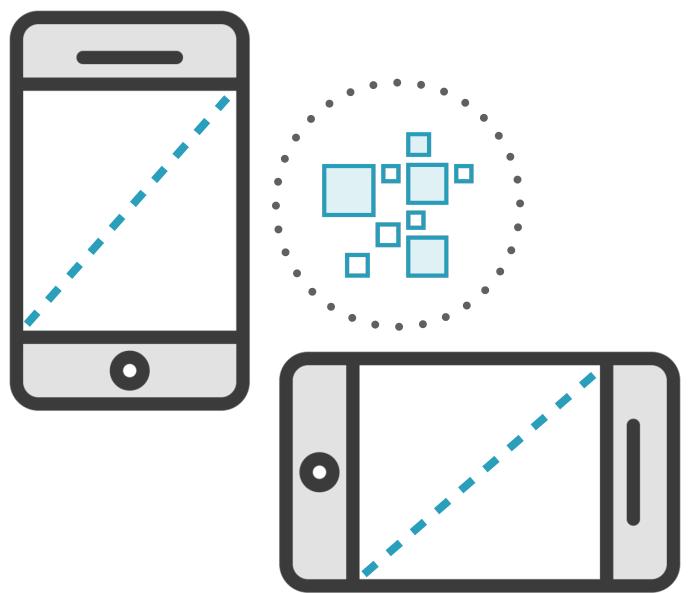


Operating system states

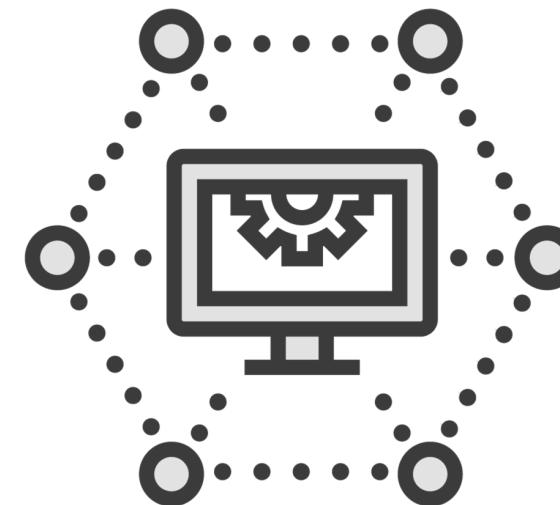


User settings

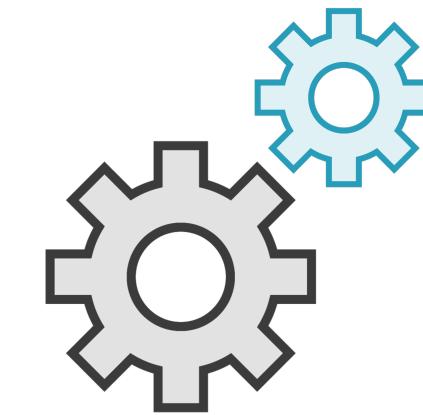
SwiftUI Environment



Device characteristics



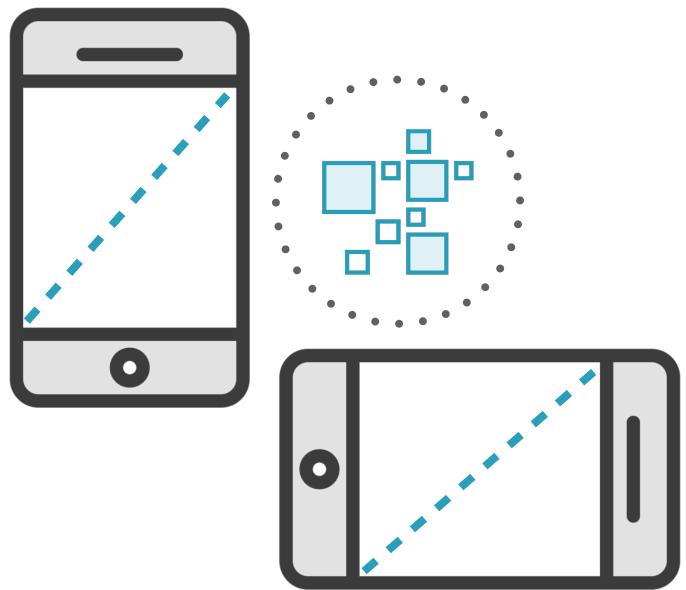
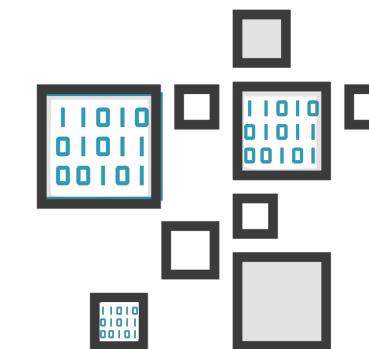
Operating system states



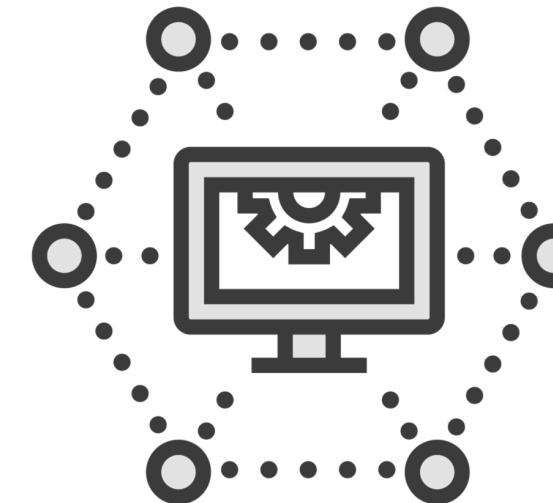
User settings

@EnvironmentObject

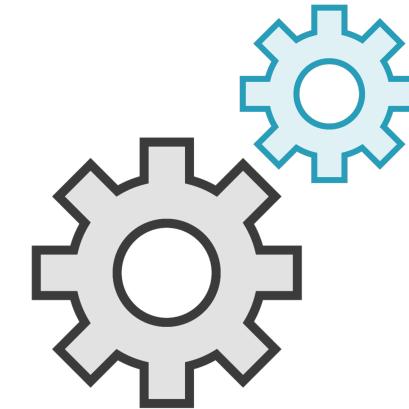
.environmentObject(taskObj)



Device characteristics



Operating system states



User settings

The screenshot shows the Xcode interface with the following details:

- Project Navigator:** Shows the project structure under "DemoClassDataModels".
- Editor:** The "ContentView.swift" file is open in the main editor area.
- Code Preview:** A preview of an iPhone 11 screen is visible on the right, showing a white screen with a black border.
- Toolbar:** Includes standard Xcode icons for file operations, search, and navigation.
- Top Bar:** Displays the project name "DemoClassDataModels", target "iPhone 11", and build status "Build for Previews Succeeded | Today at 11:31 AM".
- Bottom Bar:** Includes "Filter" and "Zoom" controls.

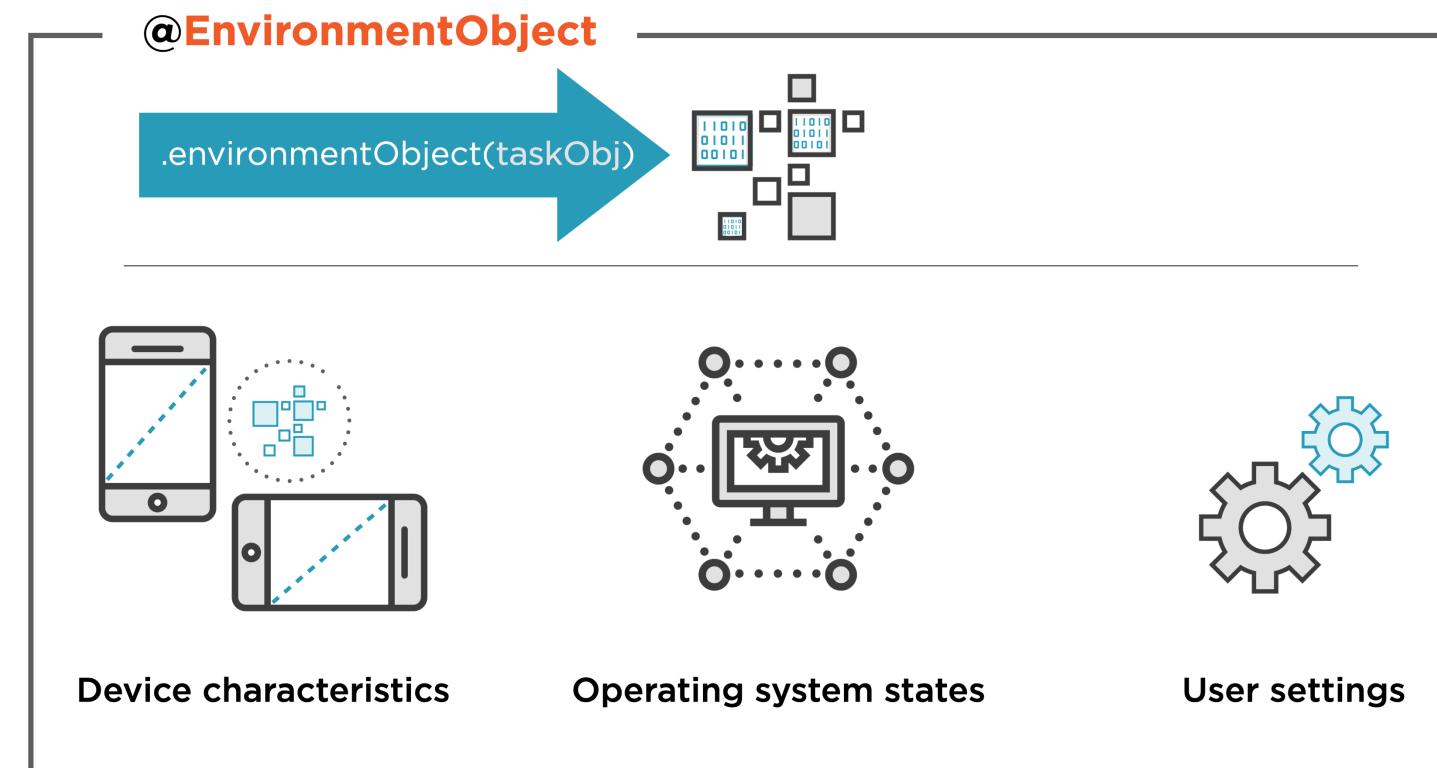
```
19
20 struct IntermediateSubview1 : View {
21     var theTask: Task
22
23     var body: some View {
24         IntermediateSubview2(theTask: self.theTask)
25     }
26 }
27
28 struct IntermediateSubview2 : View {
29     var theTask: Task
30
31     var body: some View {
32         IntermediateSubview3(theTask: self.theTask)
33     }
34 }
35
36 struct IntermediateSubview3 : View {
37     var theTask: Task
38
39     var body: some View {
40         IntermediateSubview4(theTask: self.theTask)
41     }
42 }
43
44 struct IntermediateSubview4 : View {
45     var theTask: Task
46 }
```

Bringing the Data Model and Behavior to the Night Watch App

@State

@Binding

Value Types (Structs)



@StateObject

@ObservedObject

Reference Types (Classes)



Night Watch



Night Watch

- ★ Integrate the extended data model
- ★ Work more deeply with view hierarchies
- ★ Enhance the List View
- ★ Techniques for using Xcode
- ★ Go deeper with Swift

Putting It All Together



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

Overview

Get back to the Night Watch app

Integrate the extended data model

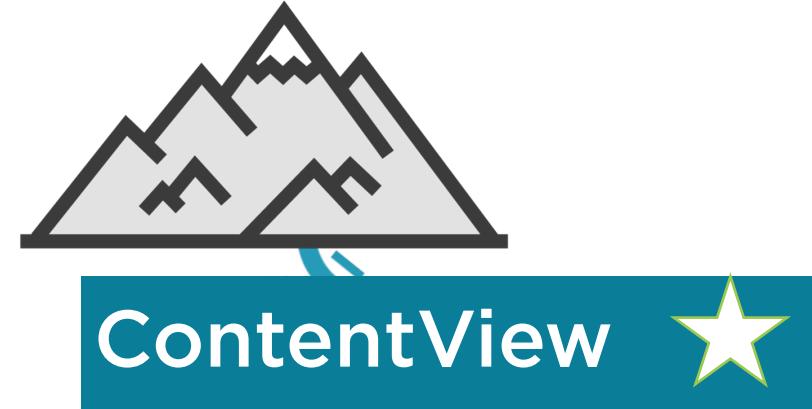
Use additional SwiftUI controls

- Toolbars
- Toggles

Enhance the List with swipe-to-delete and move actions

Show an alert

Integrating the Extended Data Model



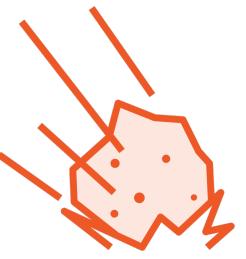
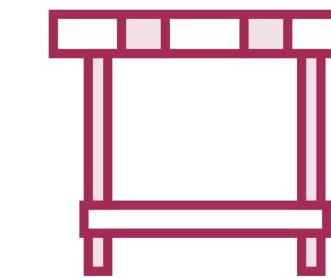
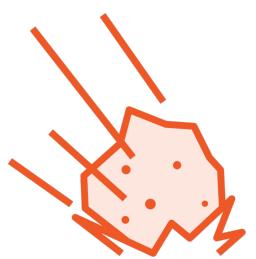
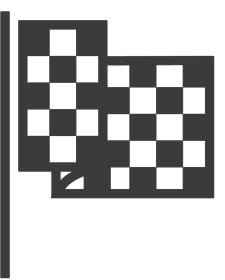
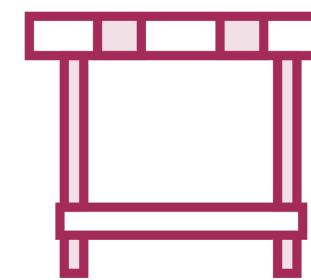
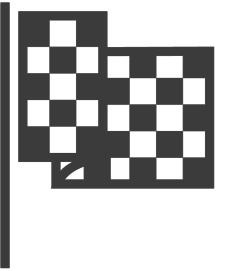
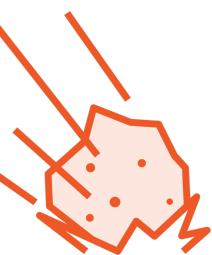
@State

@StateObject

@EnvironmentObject
+
.environmentObject()



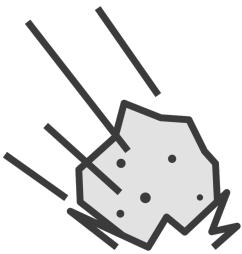
Adding the Mark Complete Behavior







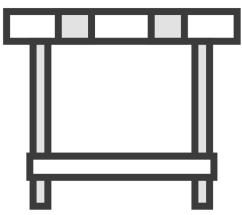
Reason and ask questions (out loud if you need)



Try things out step-by-step



Use Xcode's Option-click help pop-up



Lean on the Swift compiler to guide you

Adding the Mark Complete Behavior



Adding the Mark Complete Behavior

Adding a Toolbar and Toggle Control



1 Note





Snooze

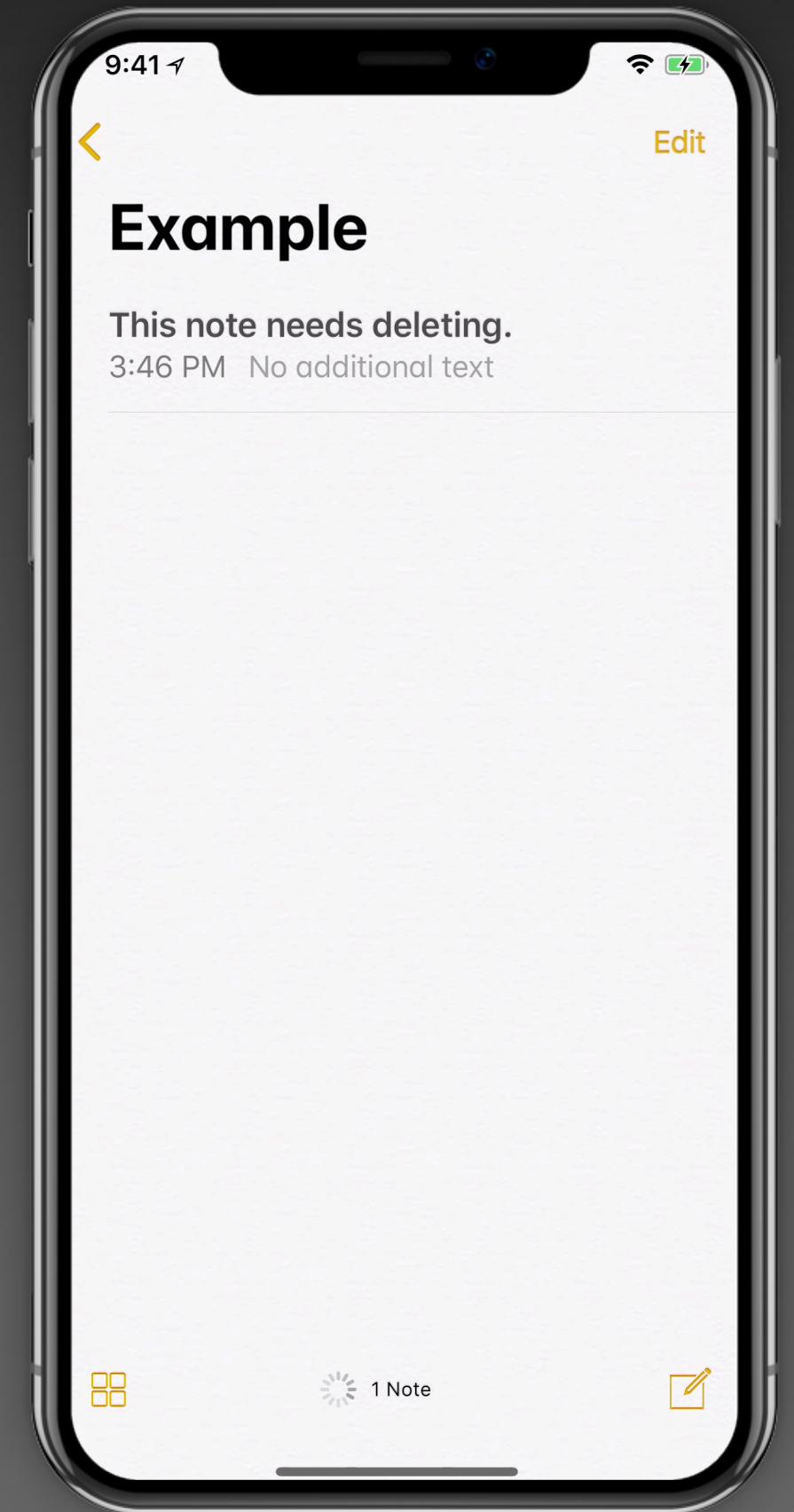


To Do



Checking for Mail...



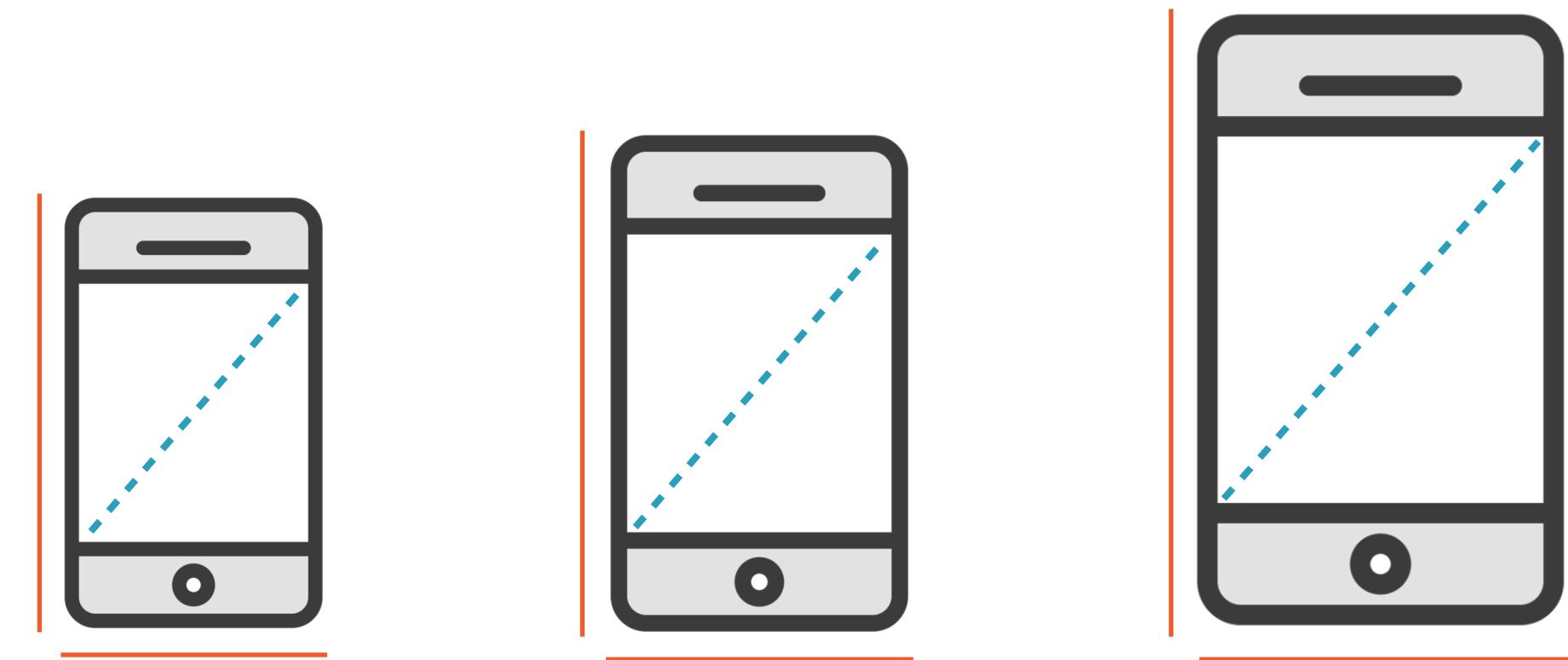


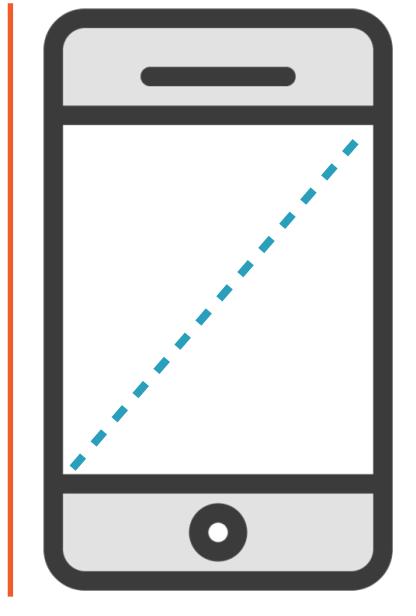
Implementing Swipe and Move Actions for a List

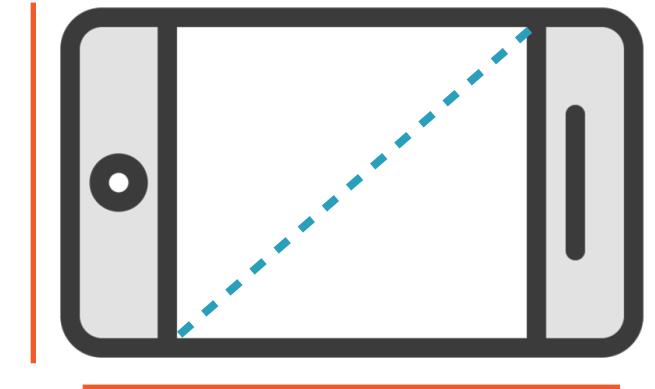
Resetting the List

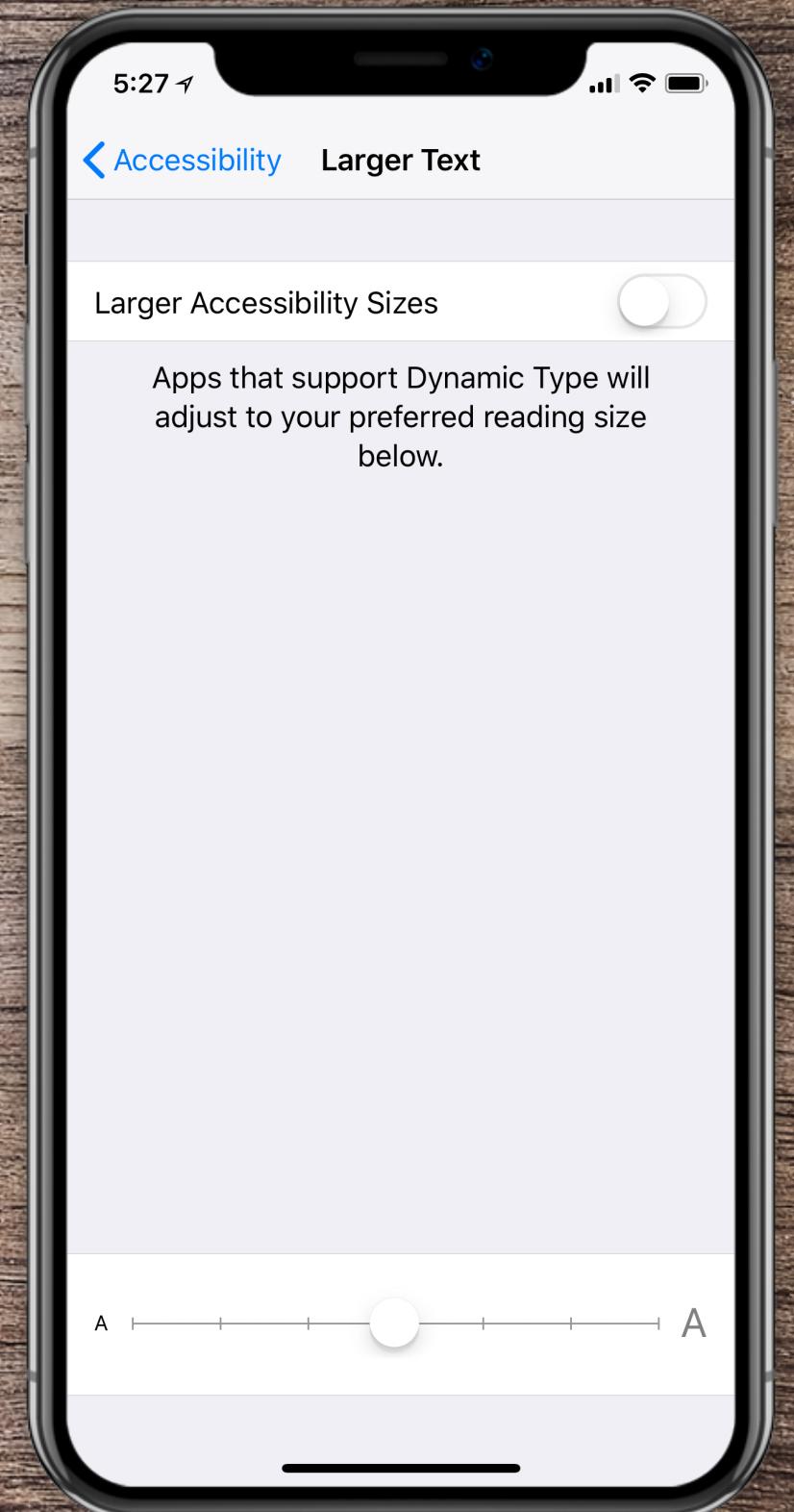
Showing an Alert

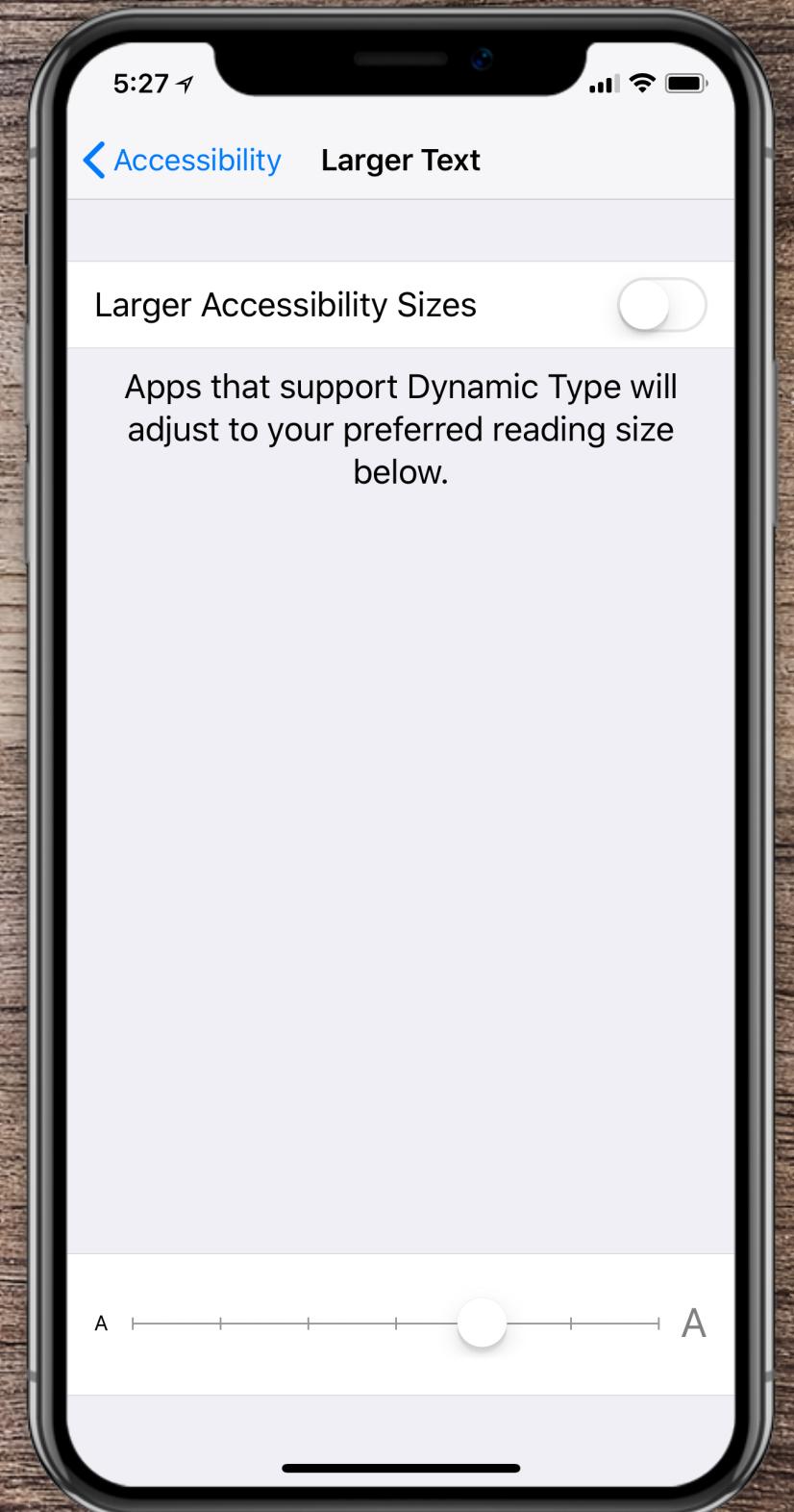
Creating an Adaptive UI



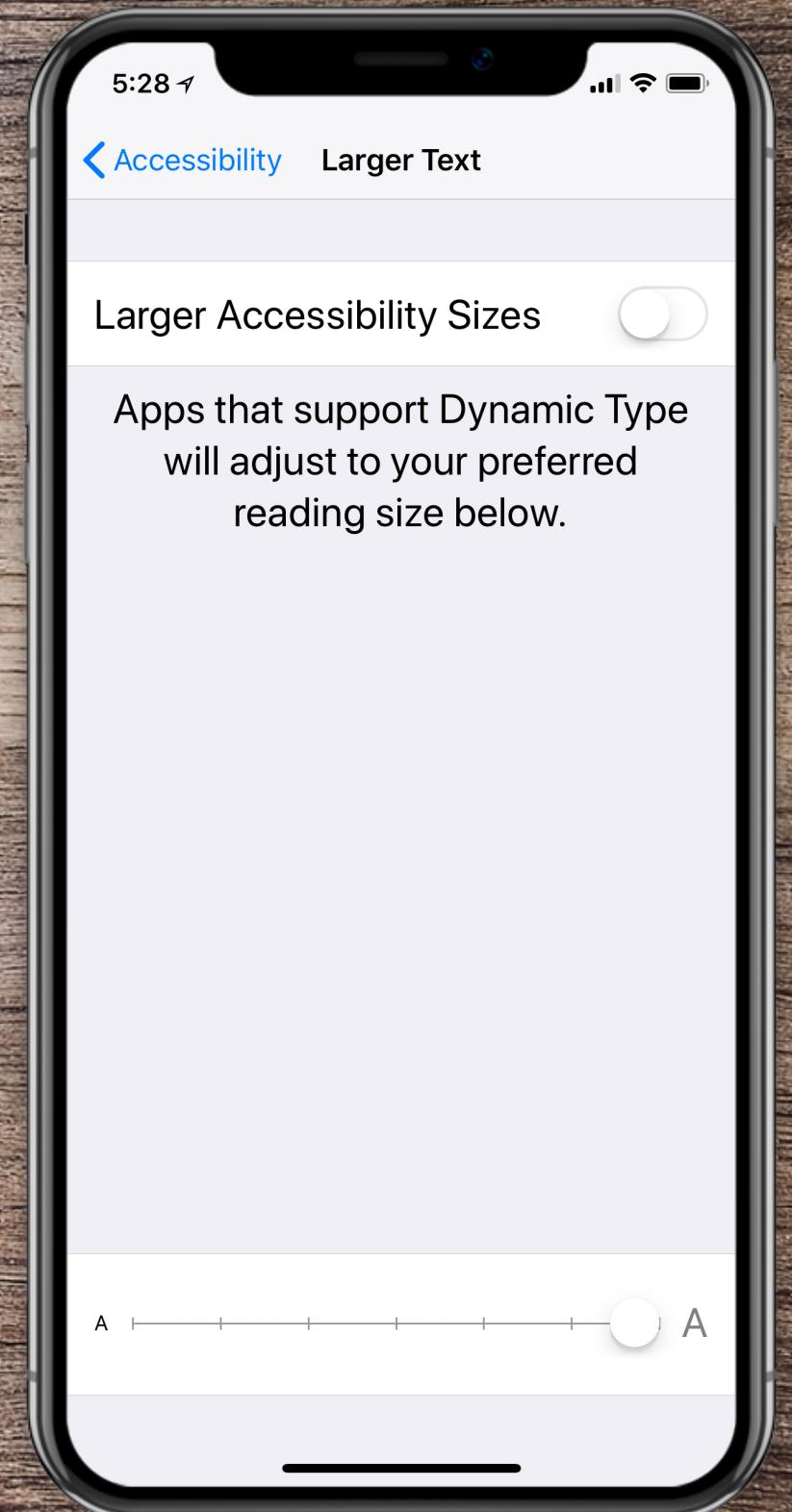












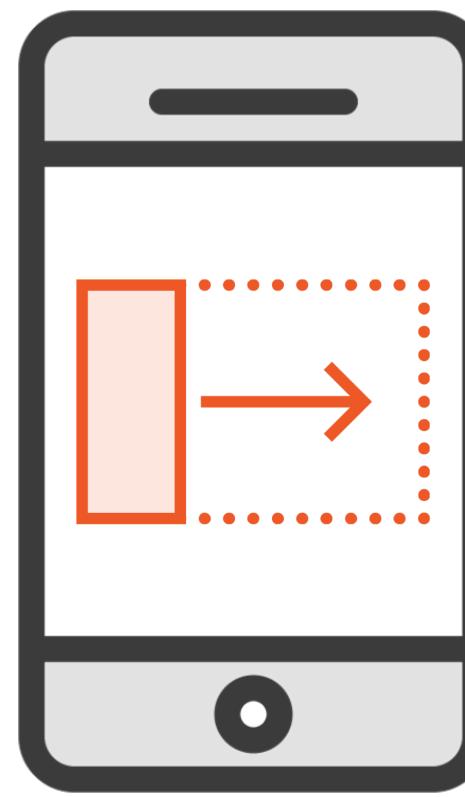
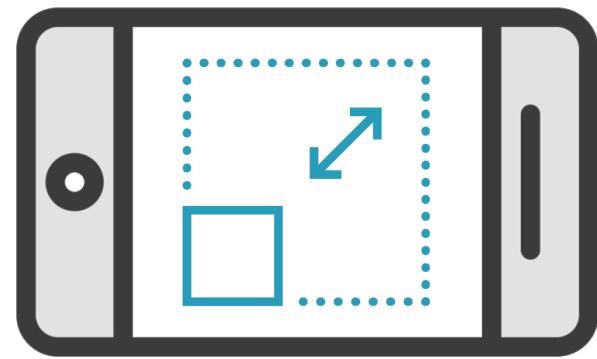
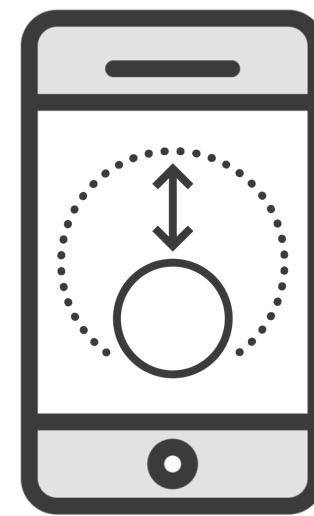
Submit

Soumettre

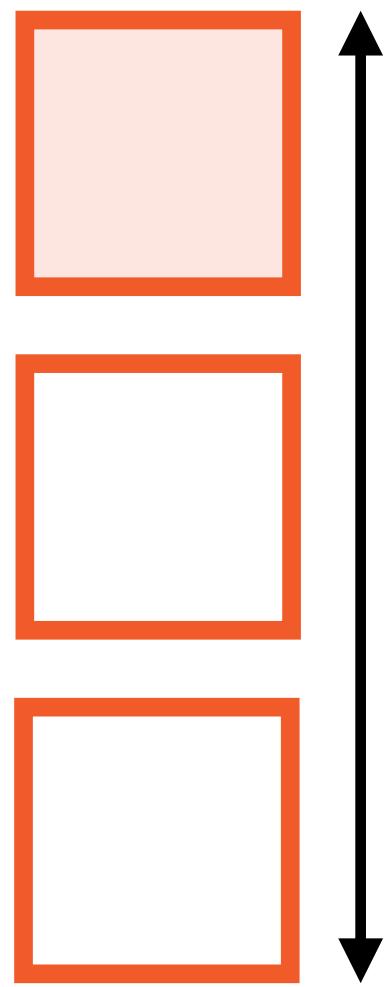
Absenden

Відправити

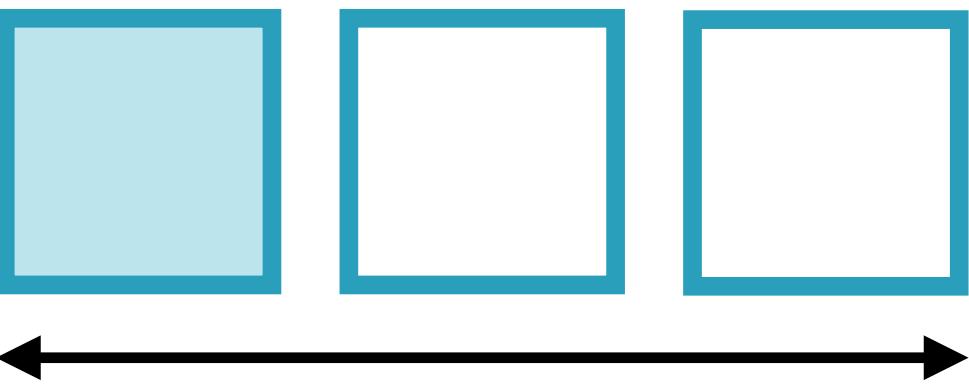
提交



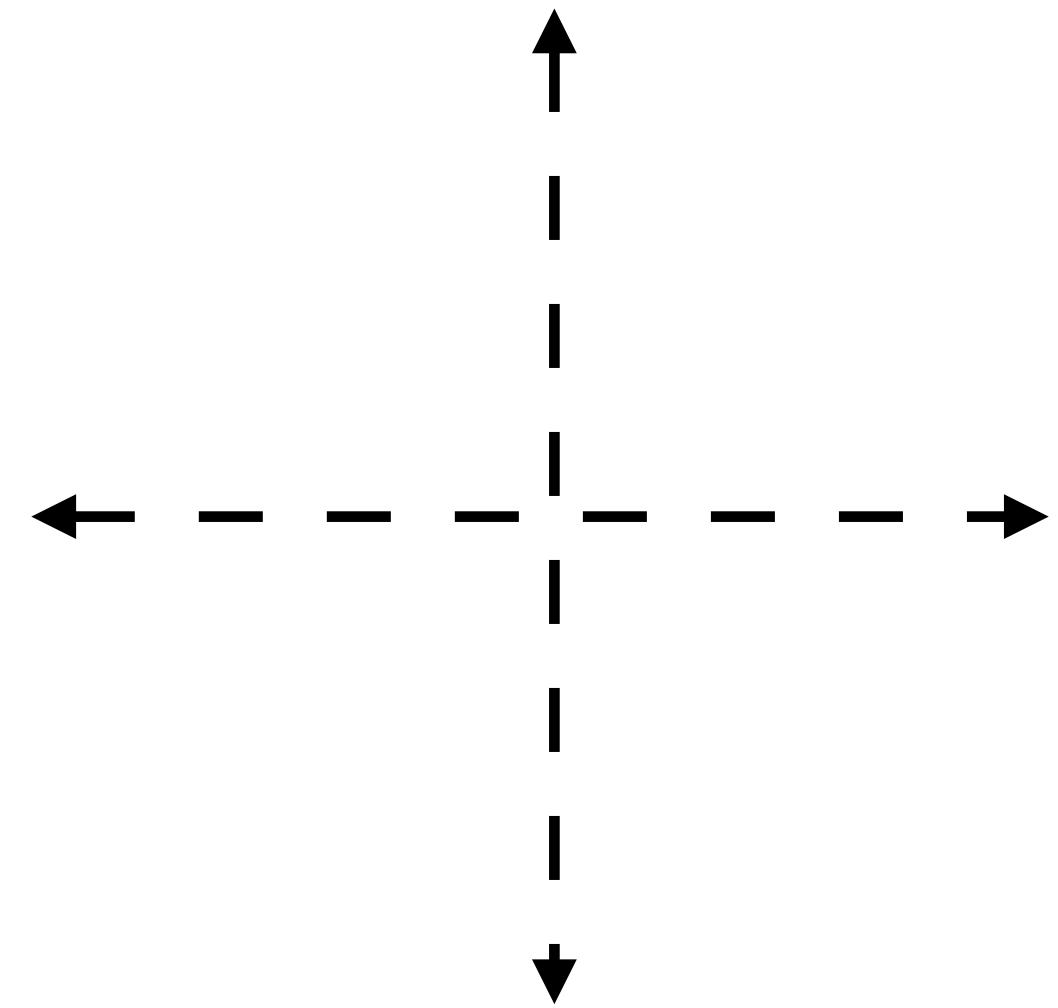
Adaptive User Interface



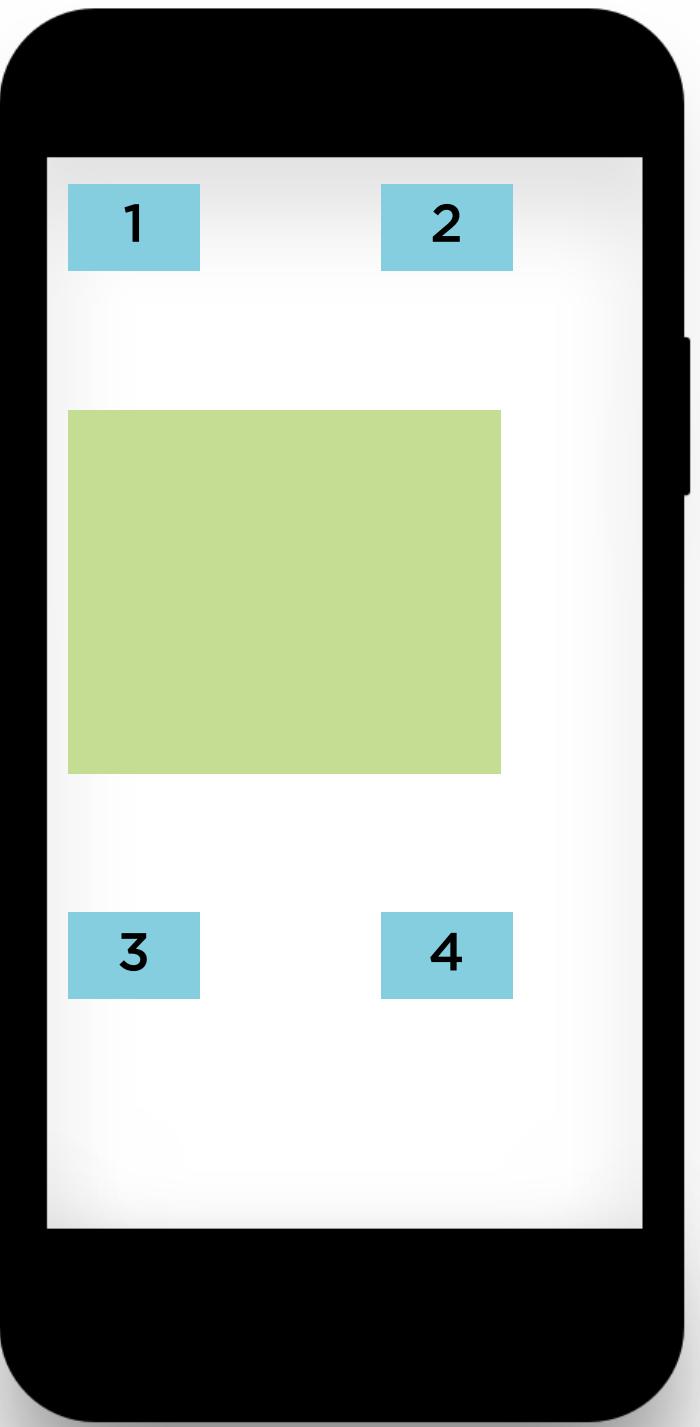
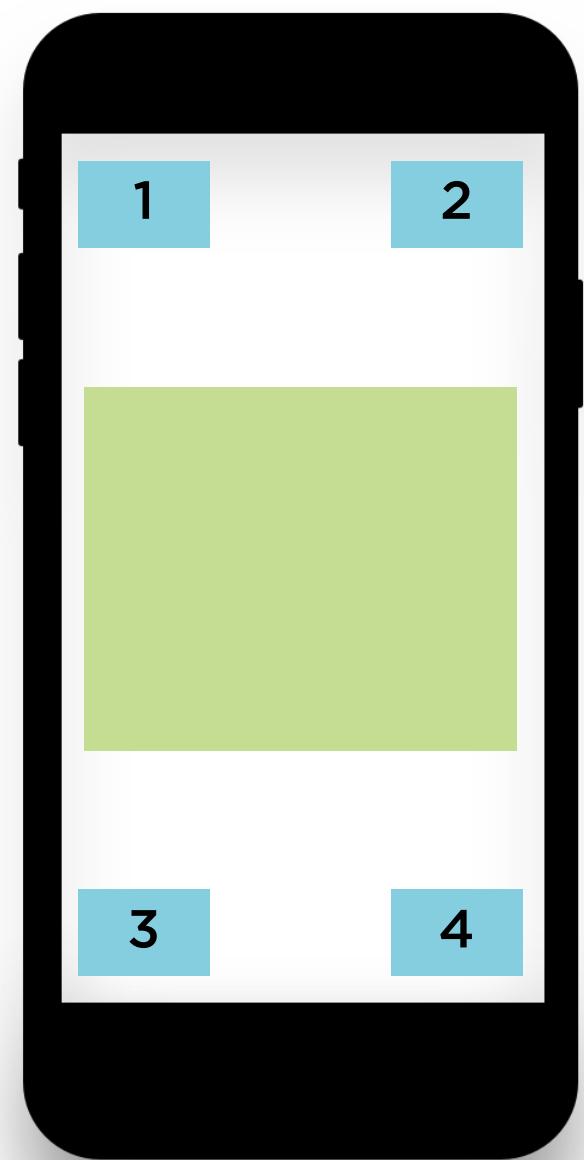
VStack

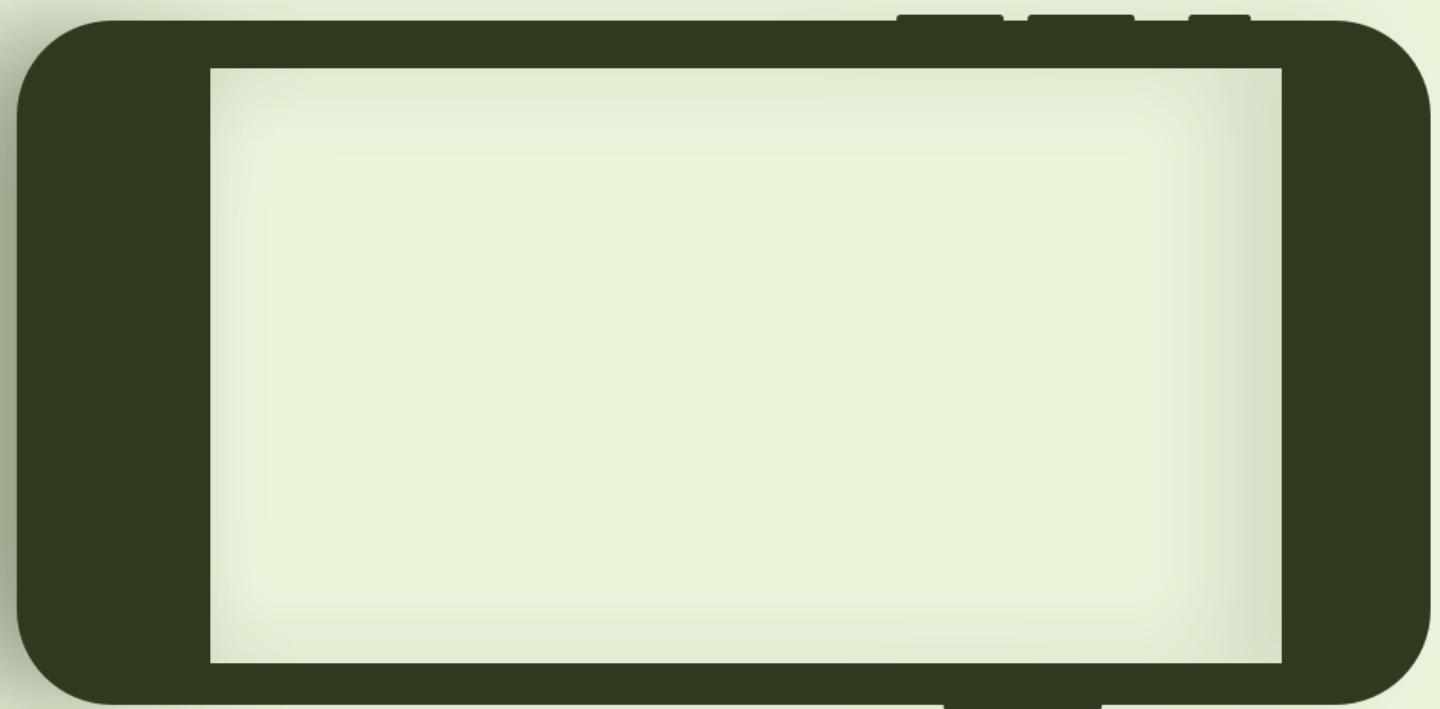
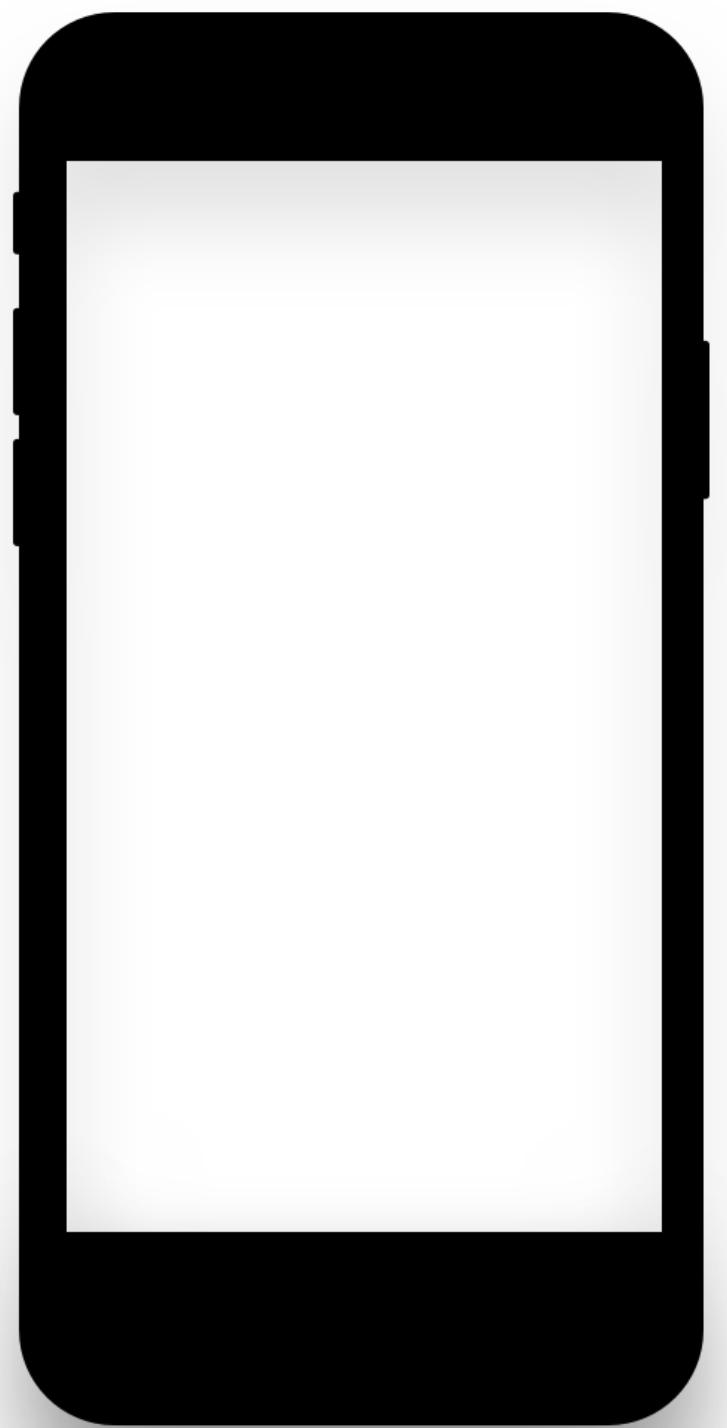


HStack



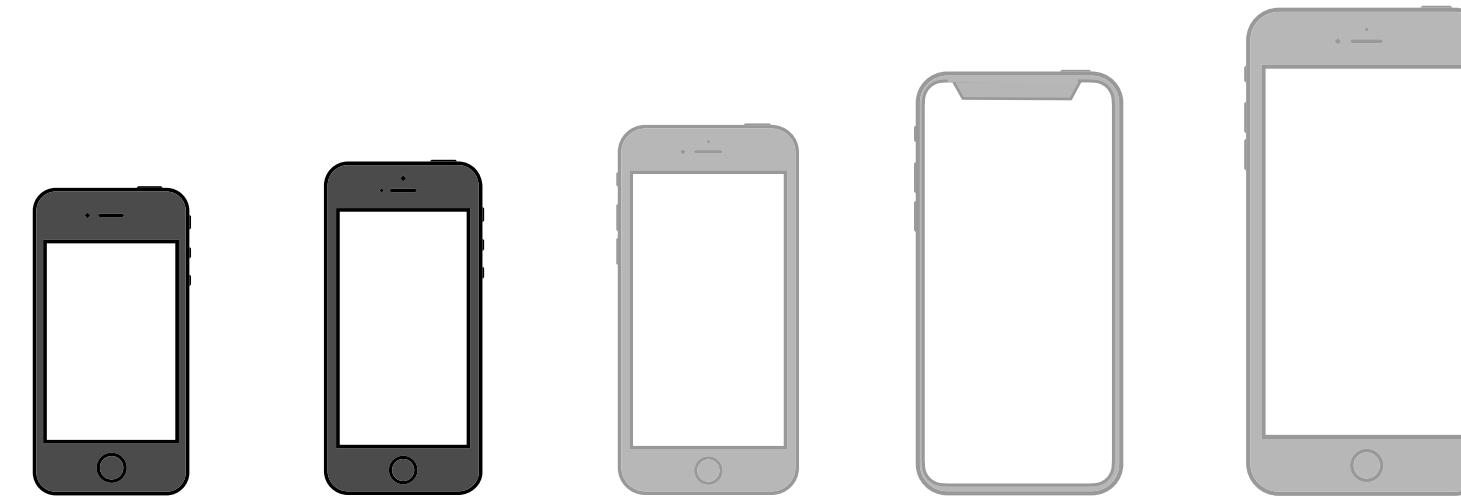
Spacer

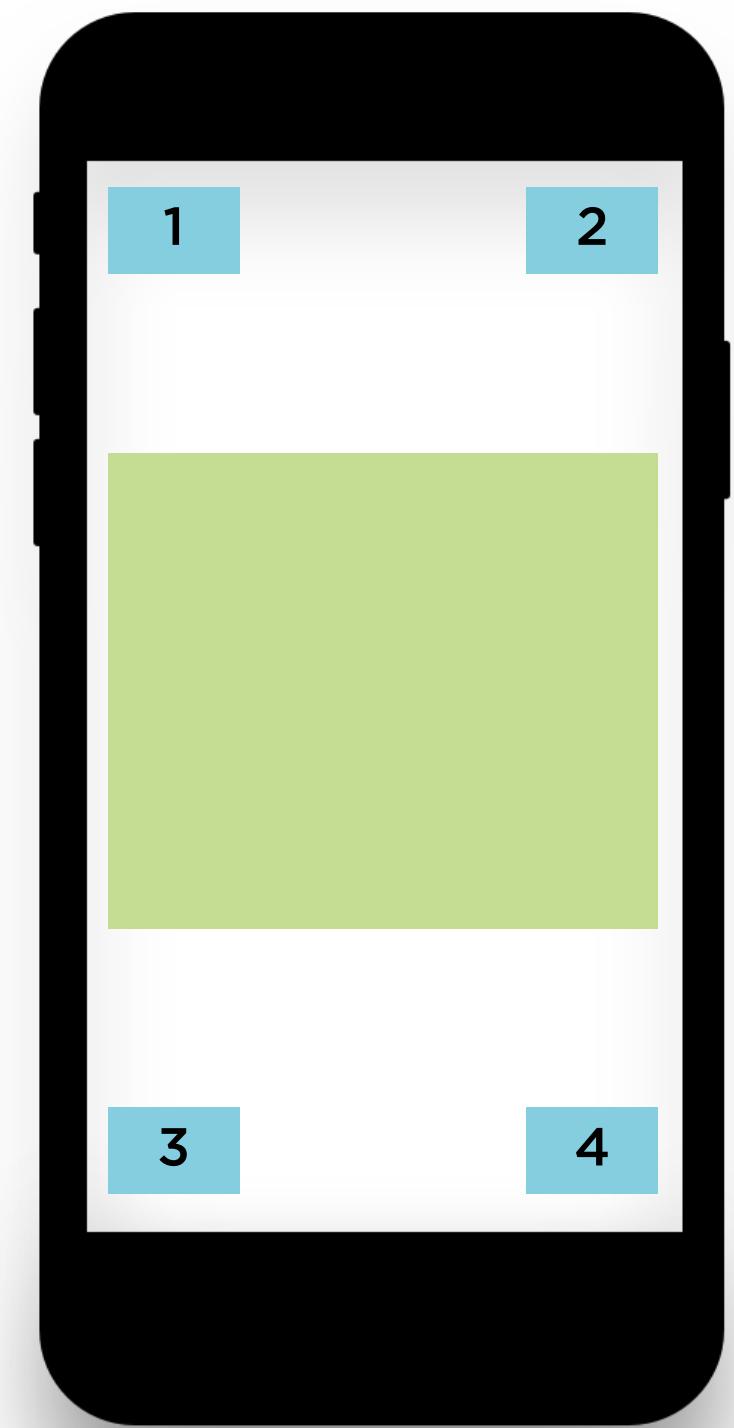


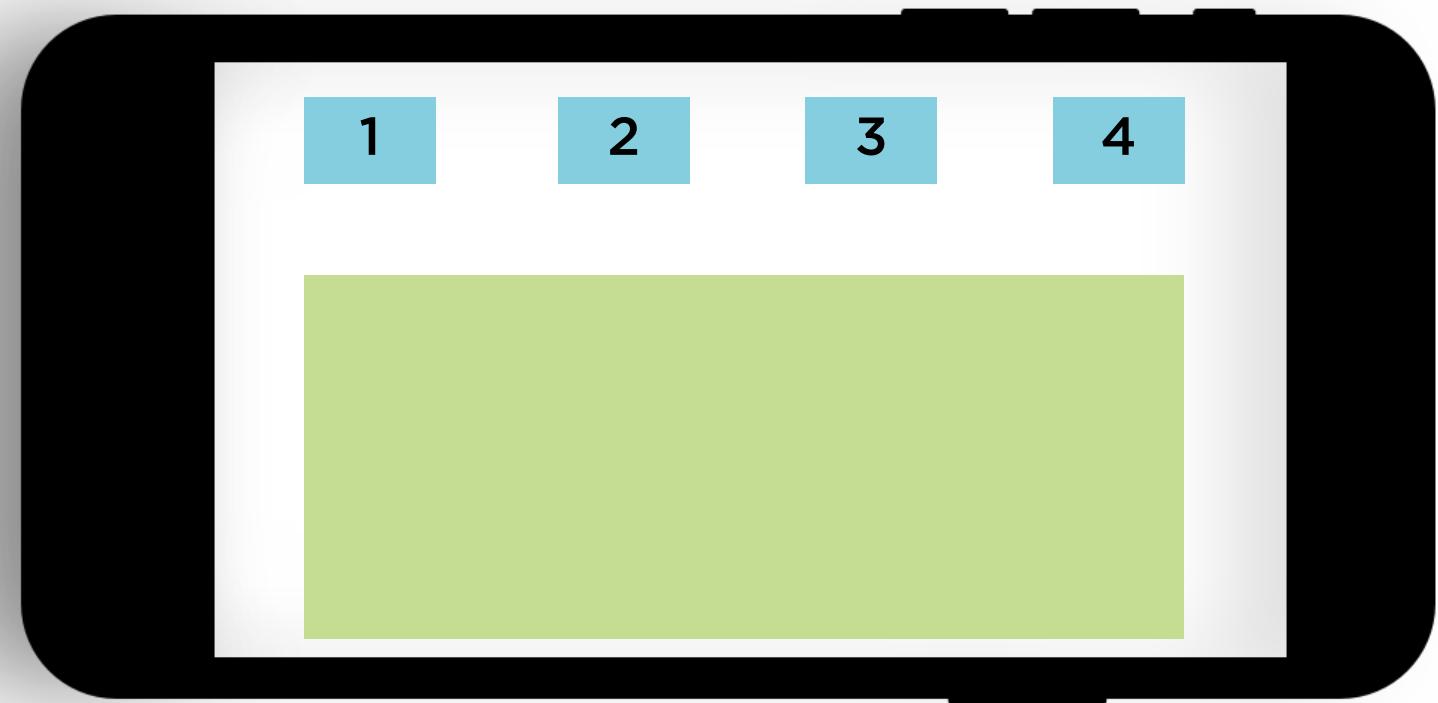


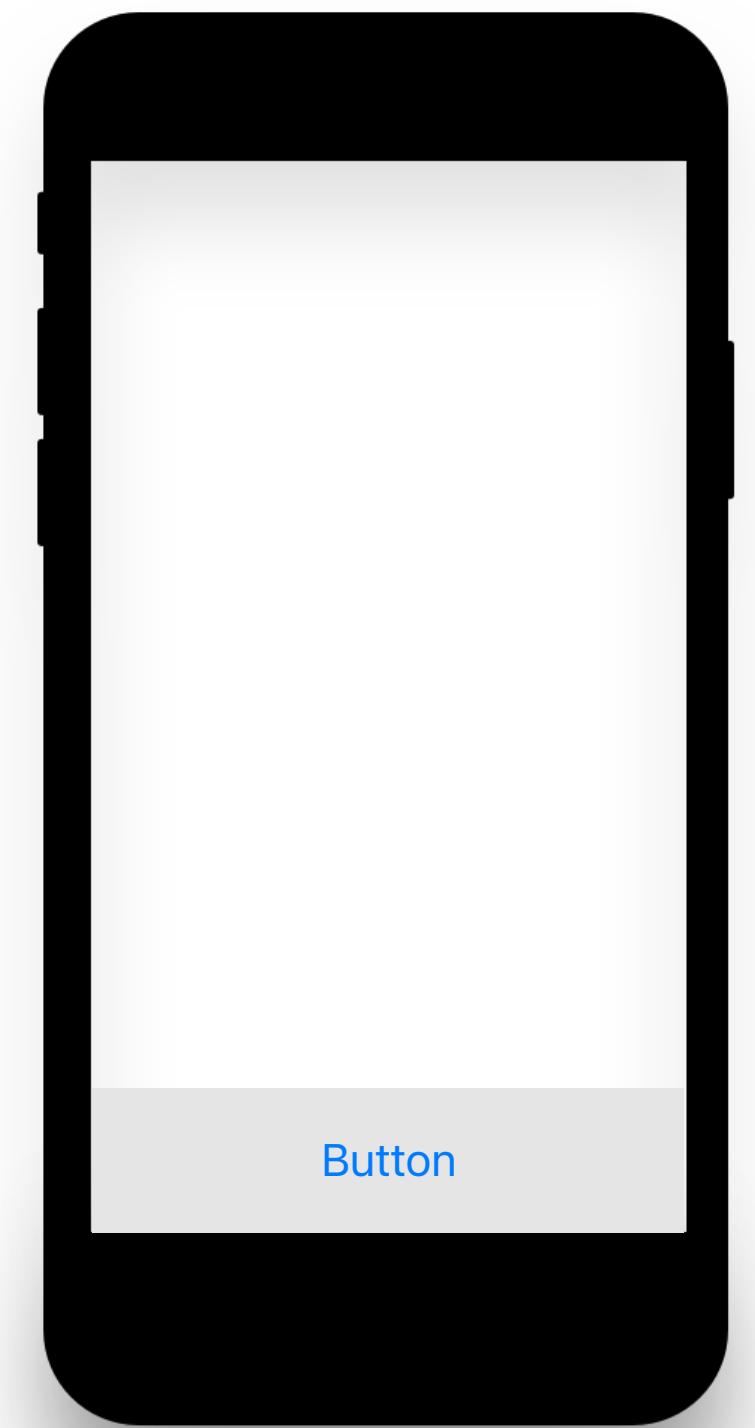
Size Classes for iPhones

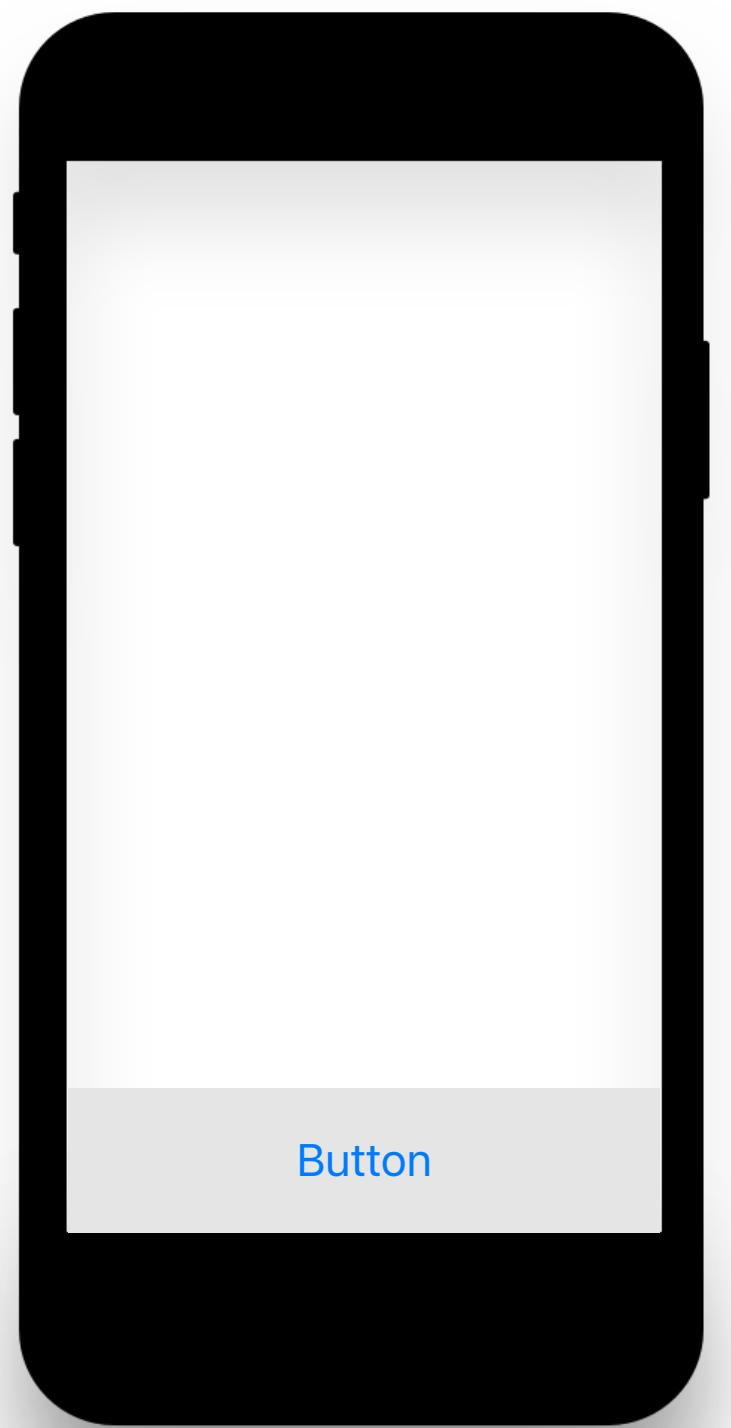
(portrait orientation)



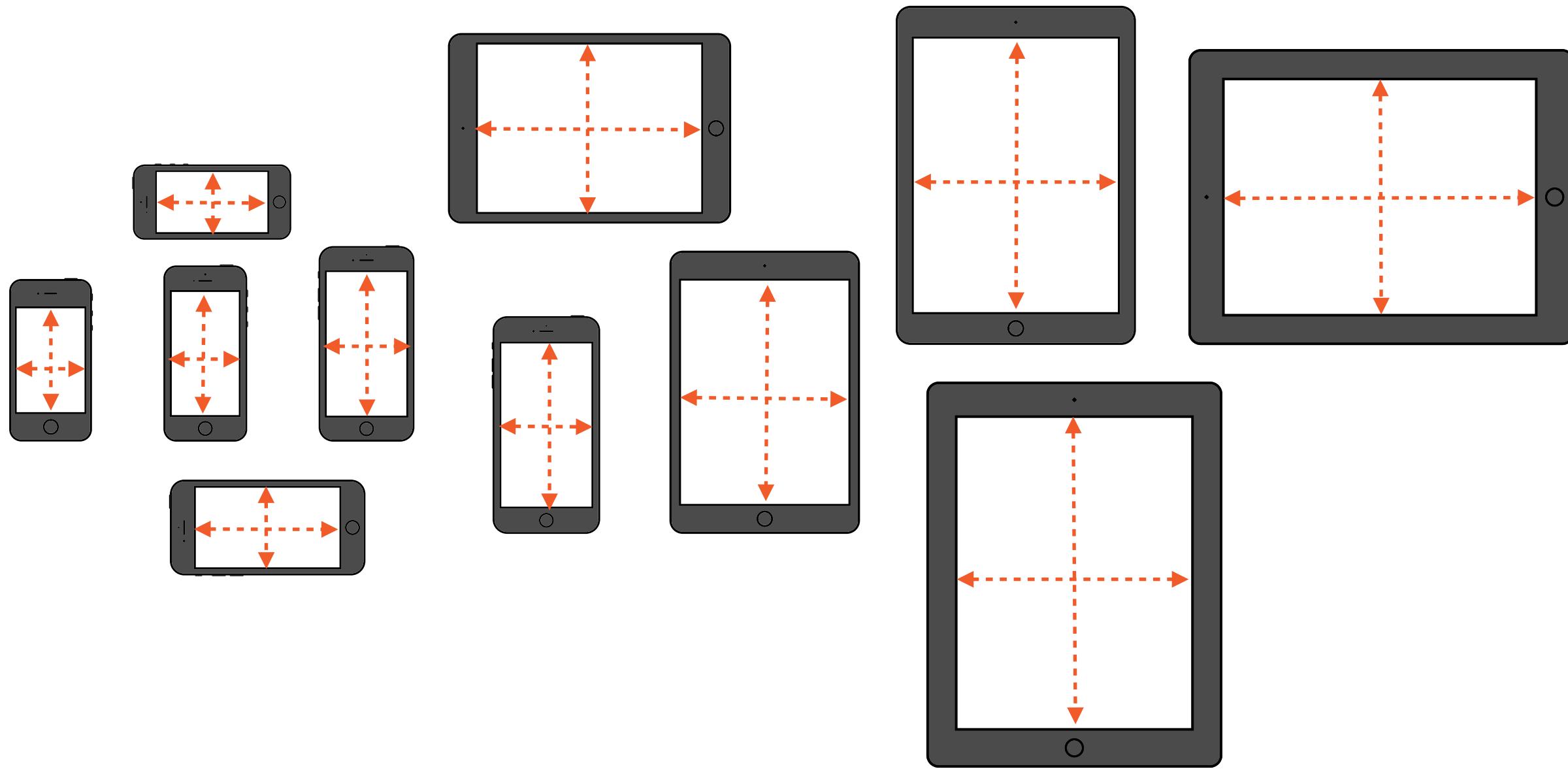


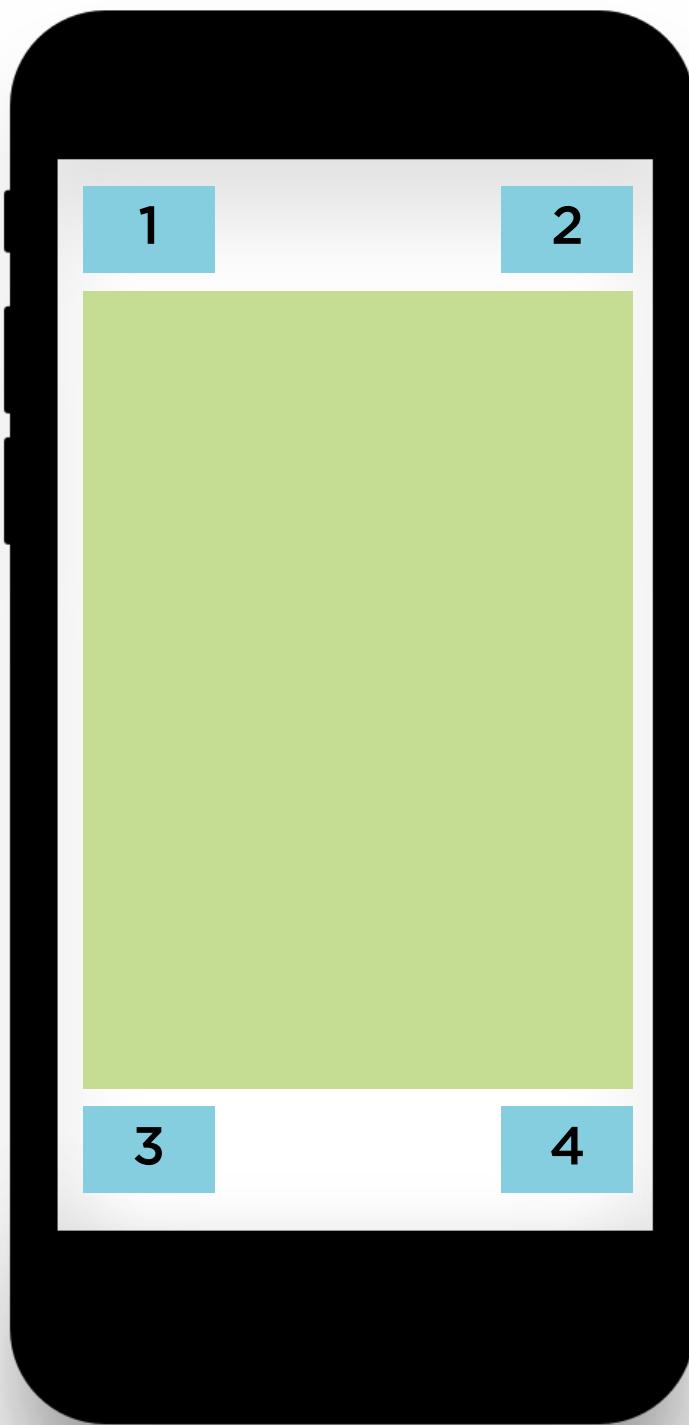






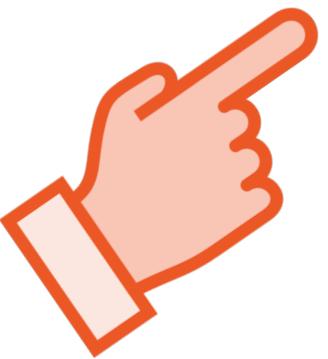
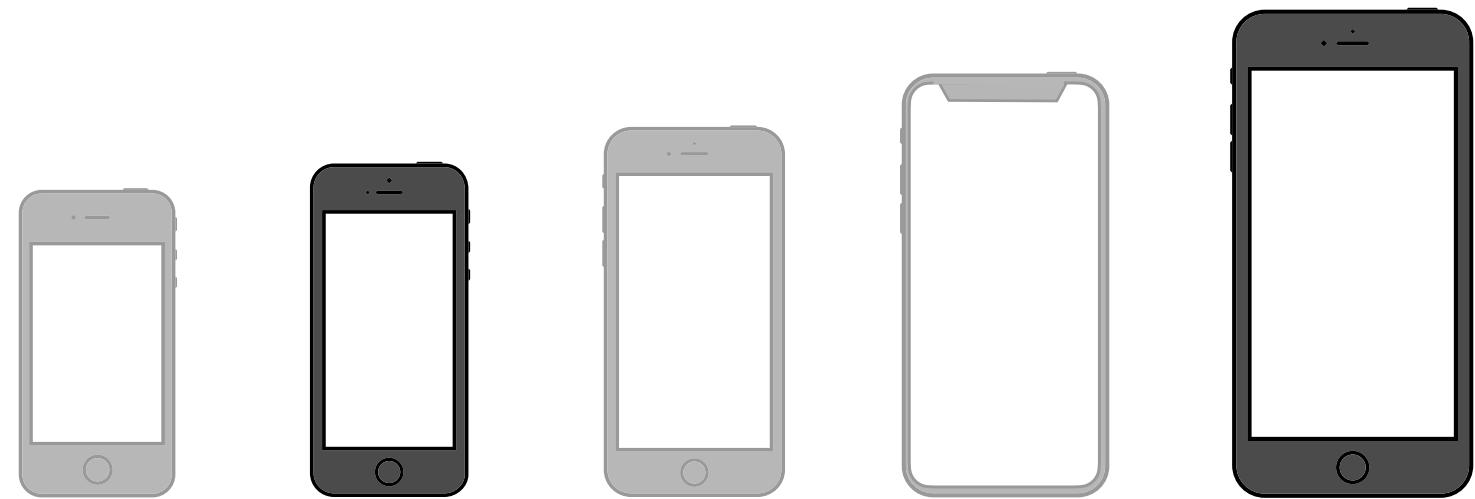
We **always** want to do as
much as we can with Stack
views.



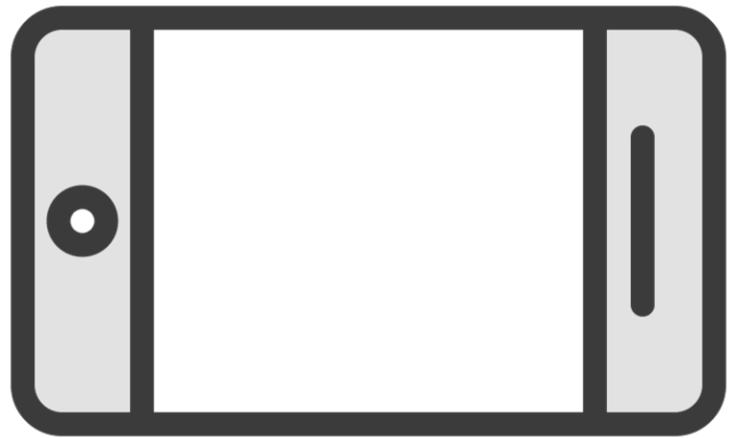




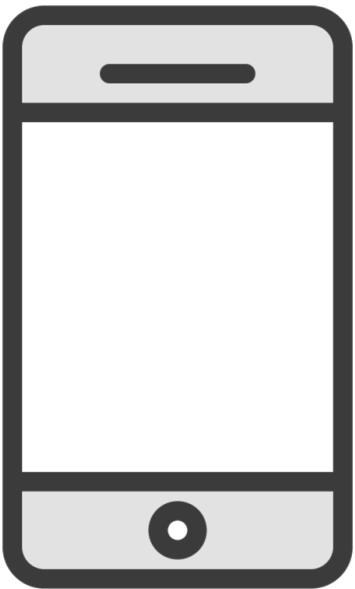
What differences are important for **our user interface** in **our application**?



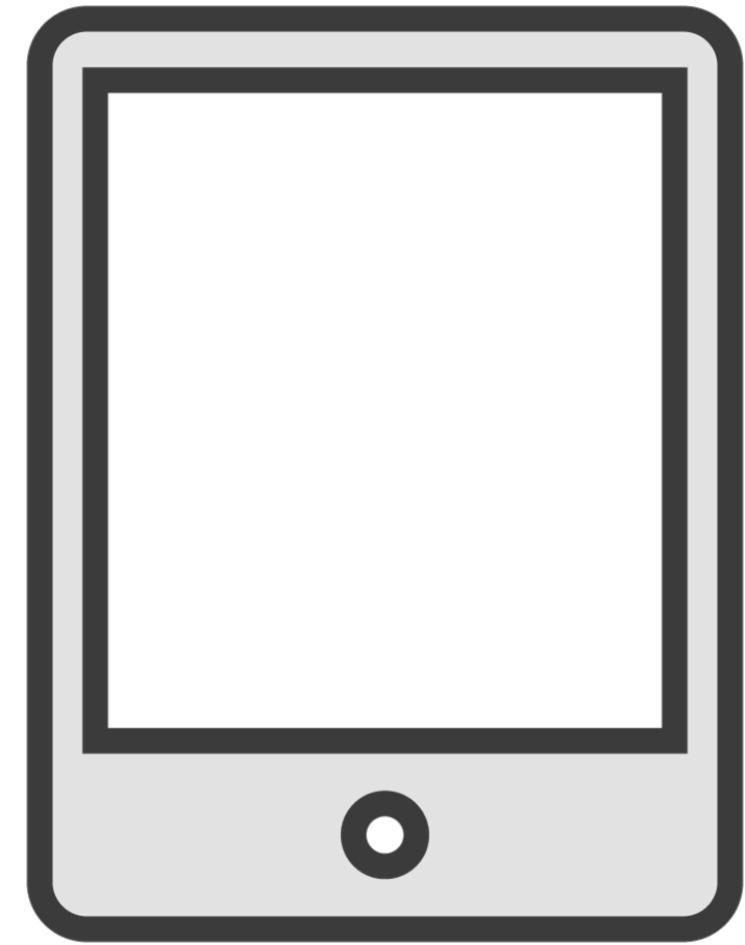
We don't work at the level of specific devices (and we don't want to).



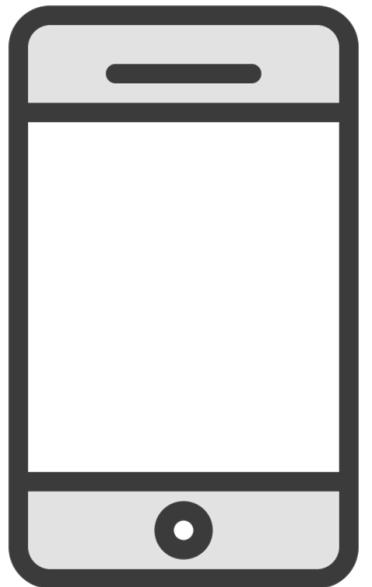
iPhone XS in landscape



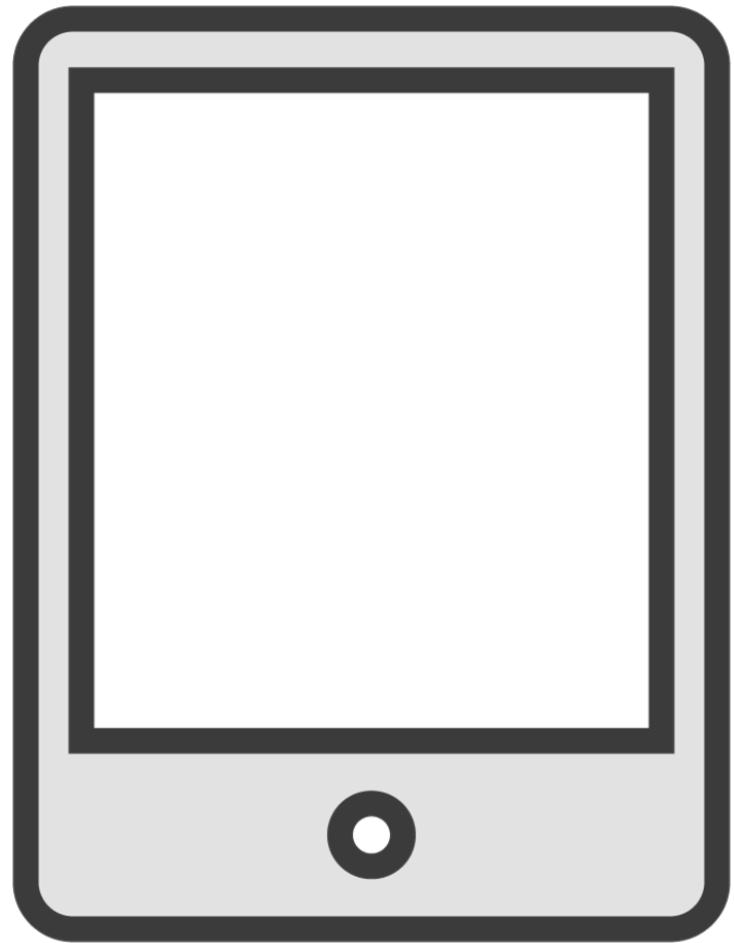
iPhone X



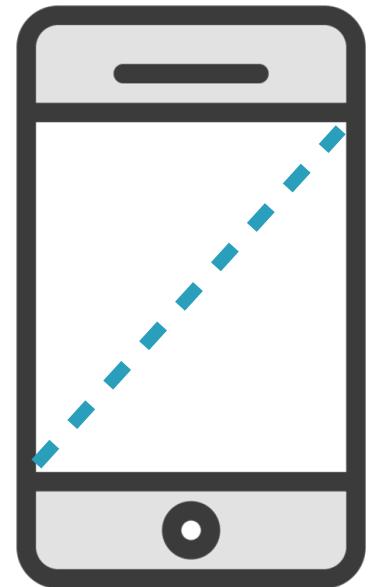
iPad Pro 9.7"



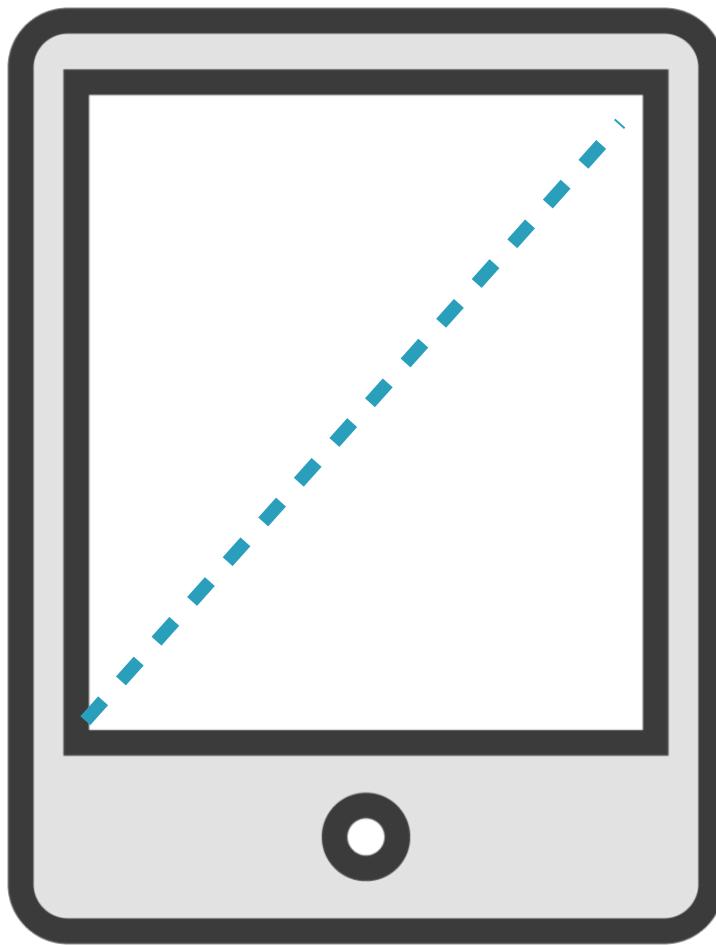
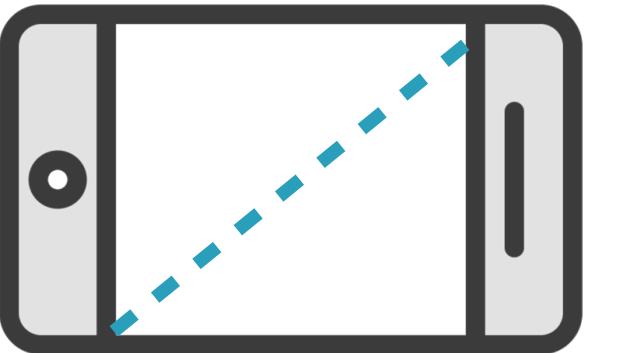
Small



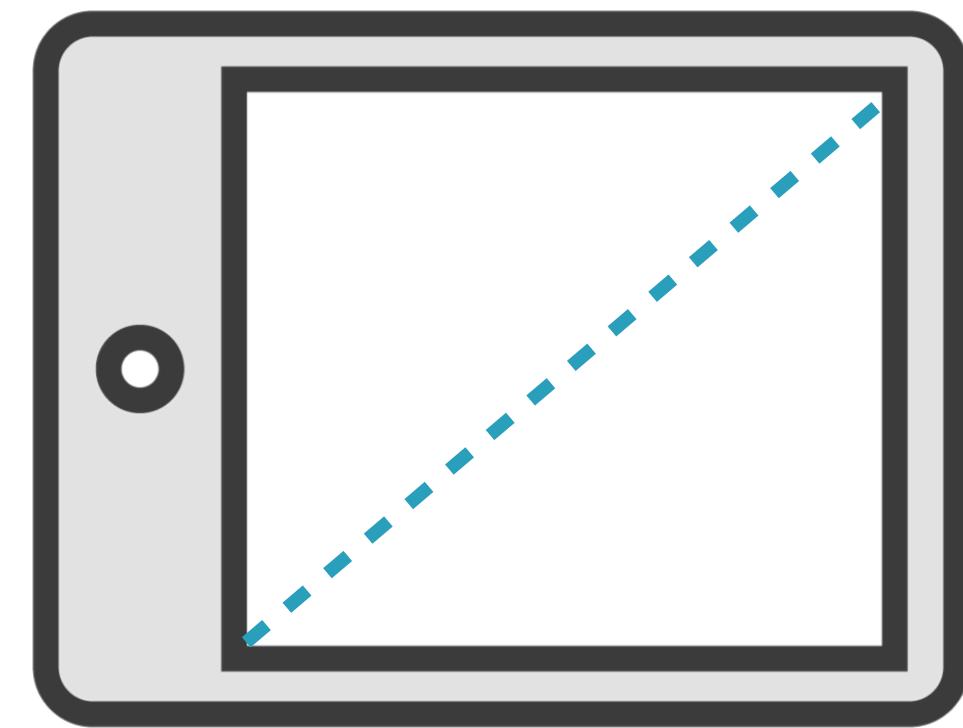
Large

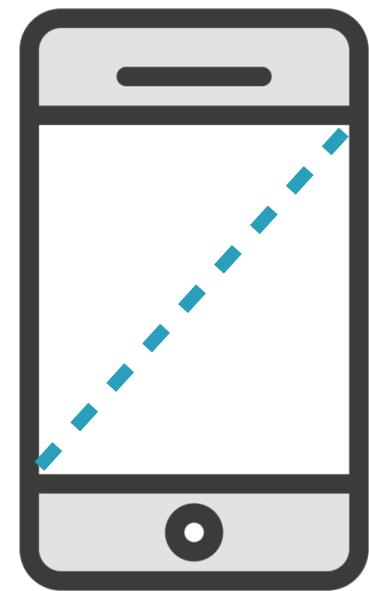


Small

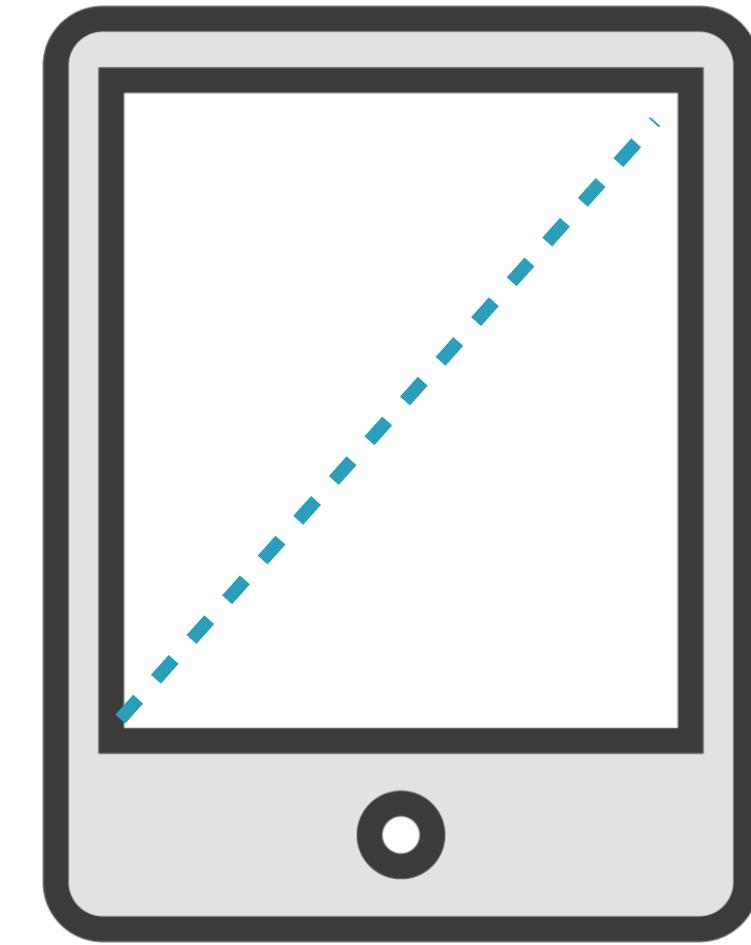
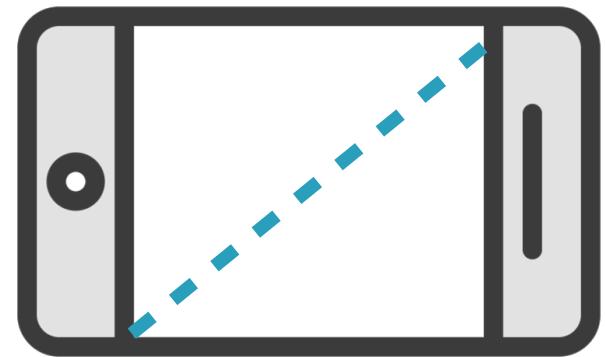


Large

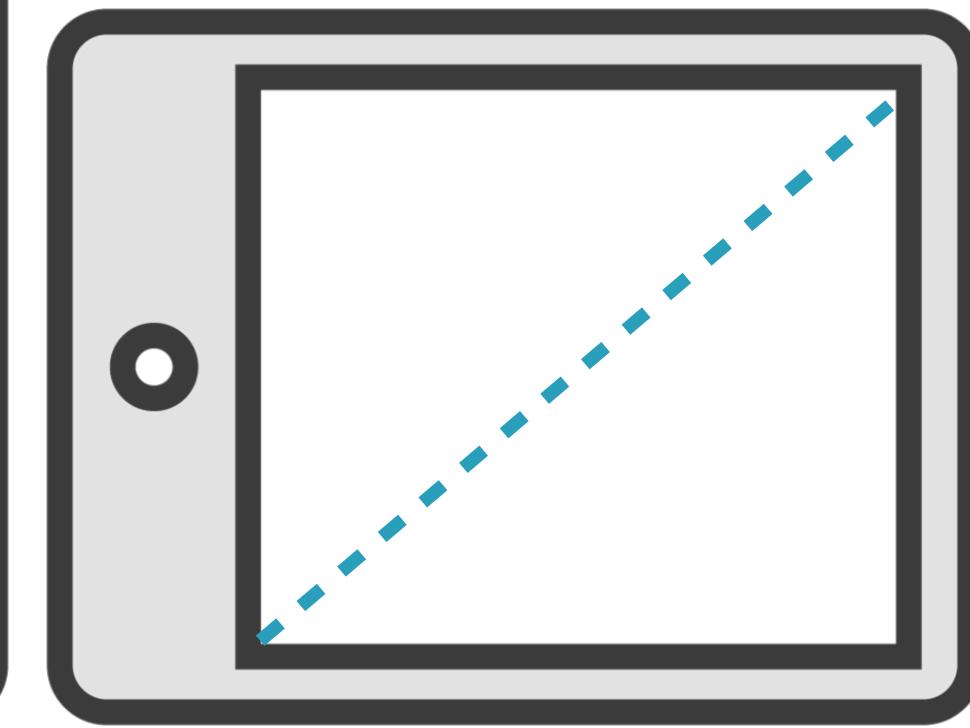




Small



Large

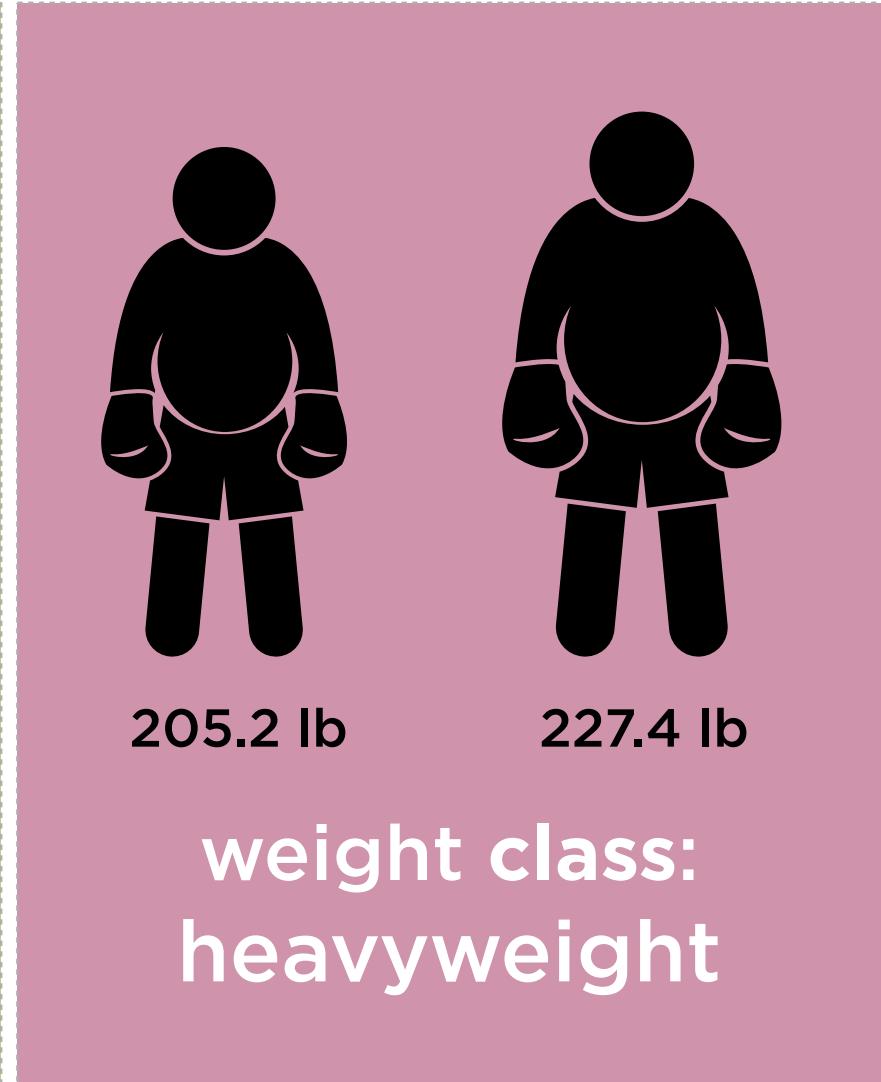
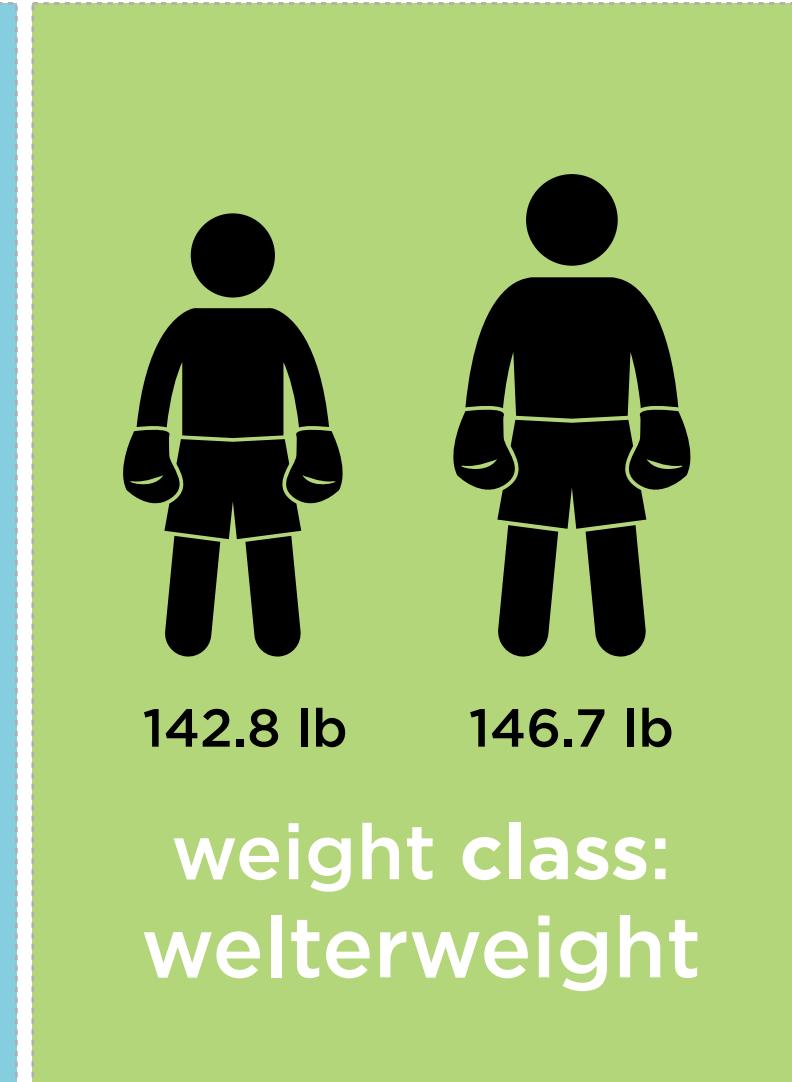
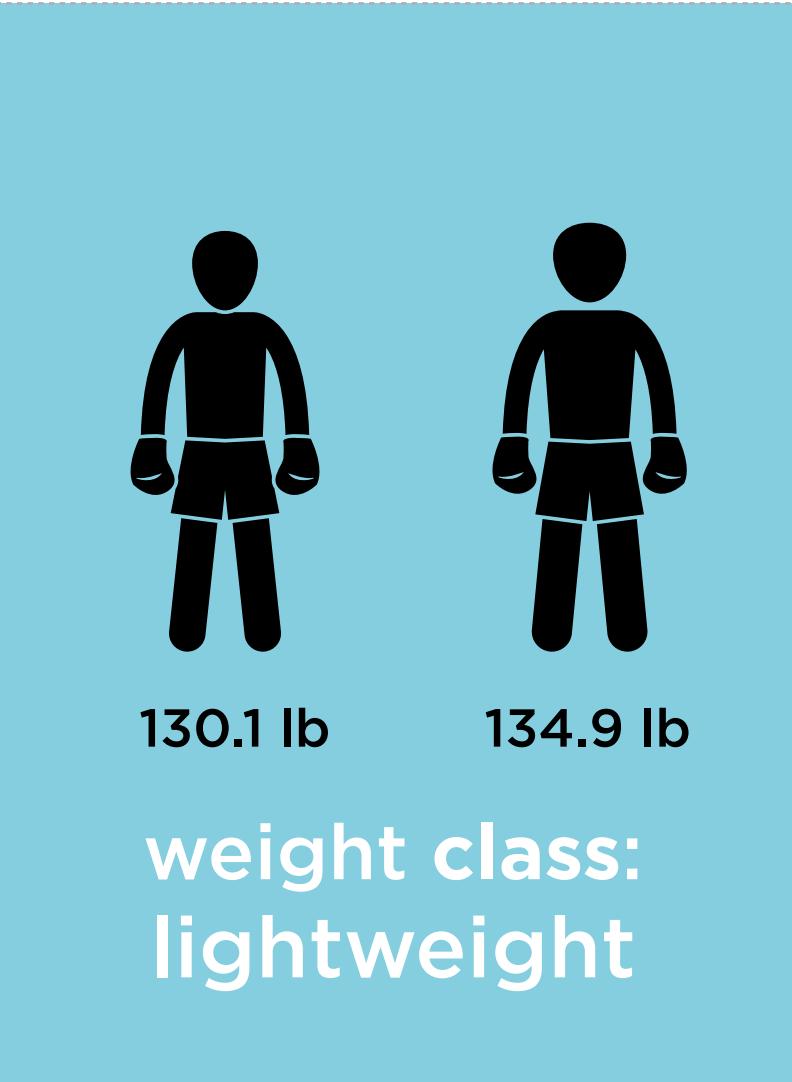


Size Class

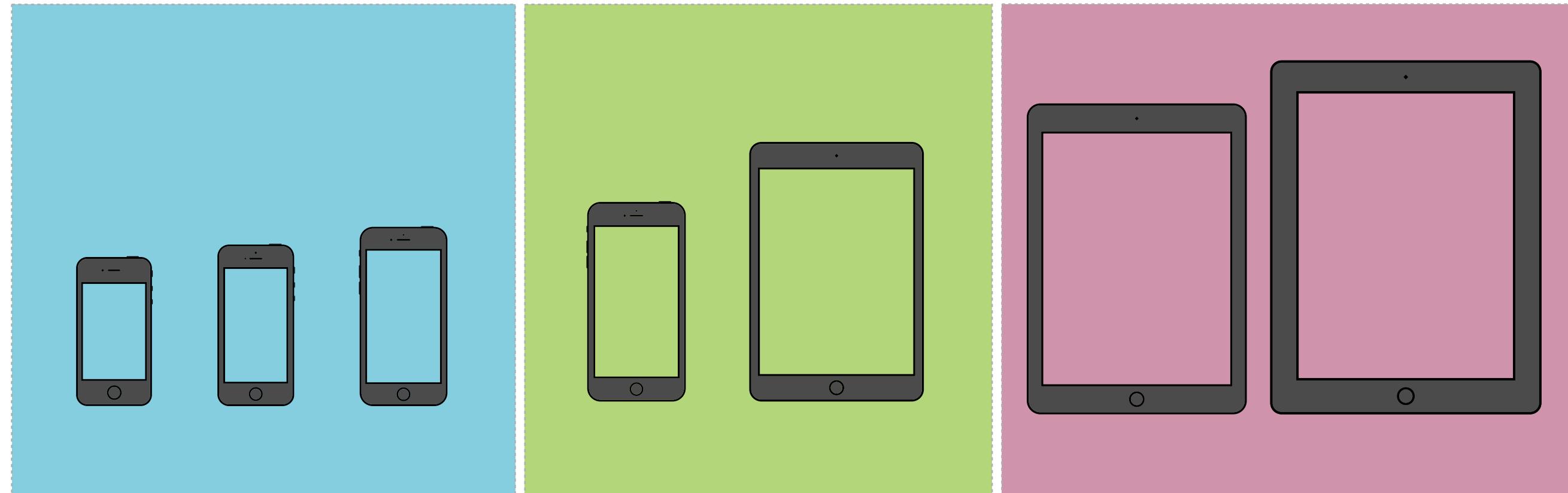
“Size Category”

“Size Grouping”

"Class" as in "Category"



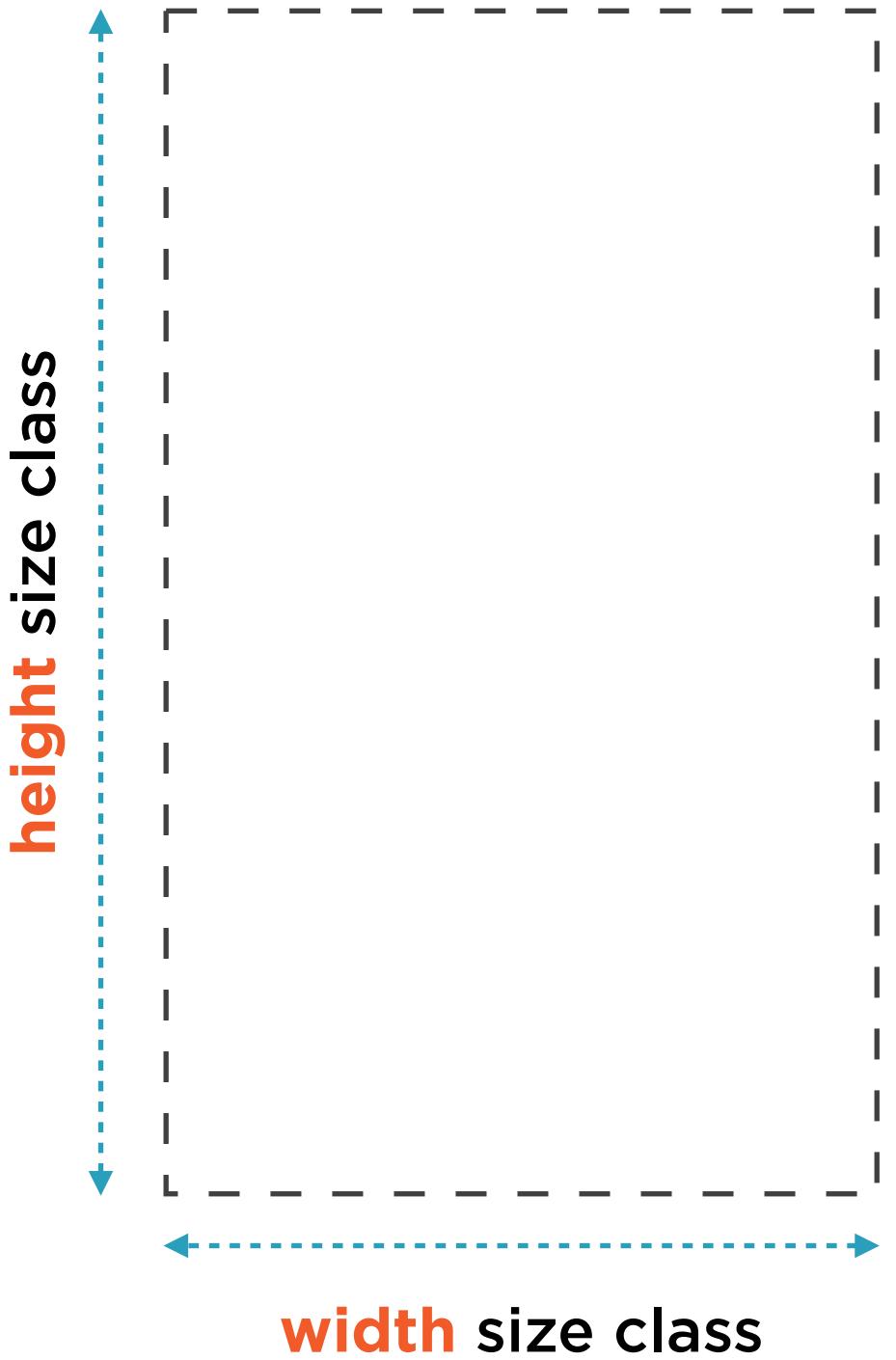
iOS Device Screens

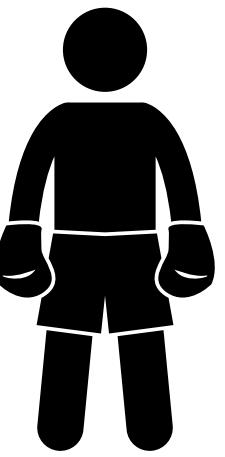


Size Class

“Size Category”

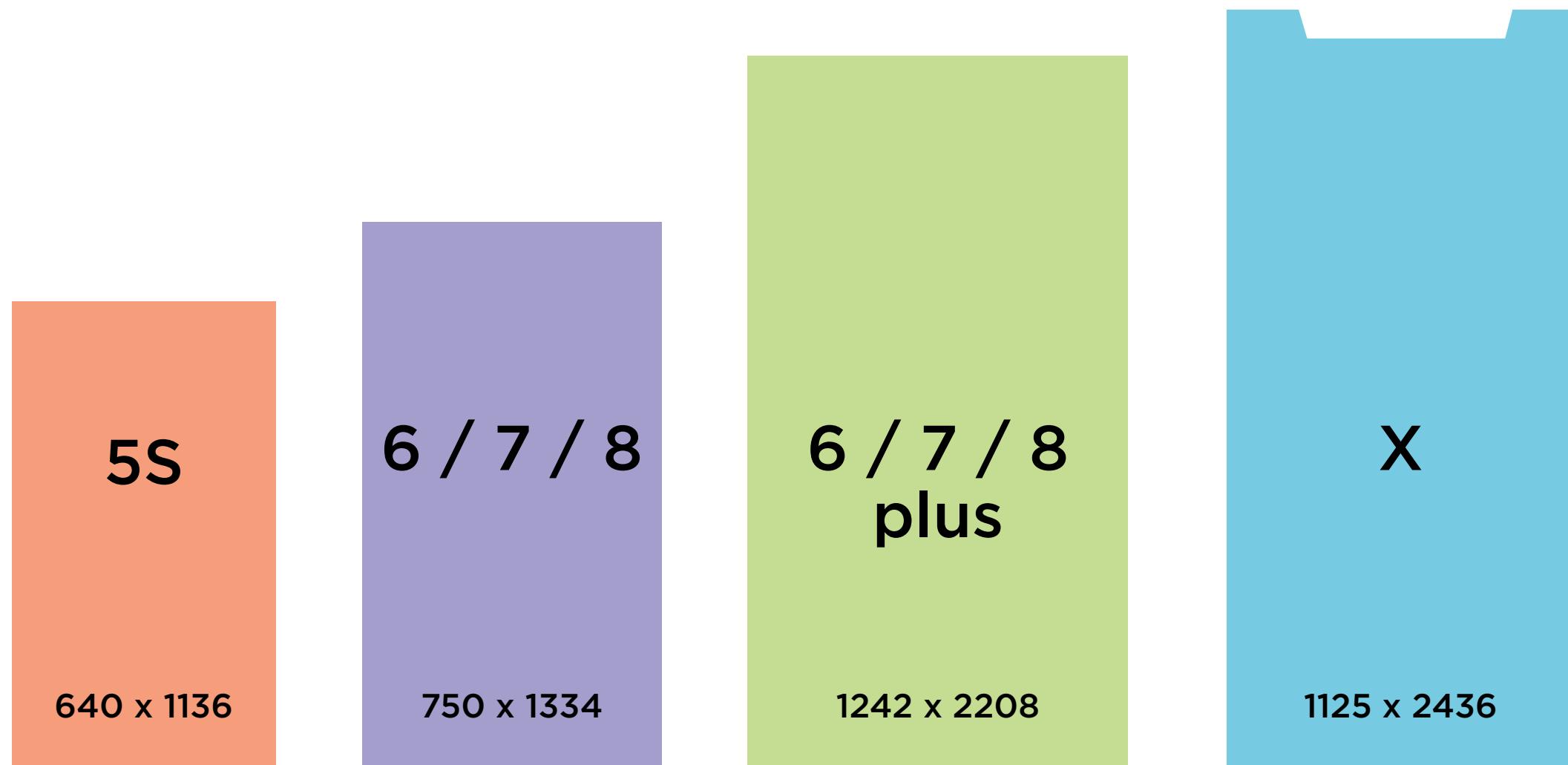
“Size Grouping”



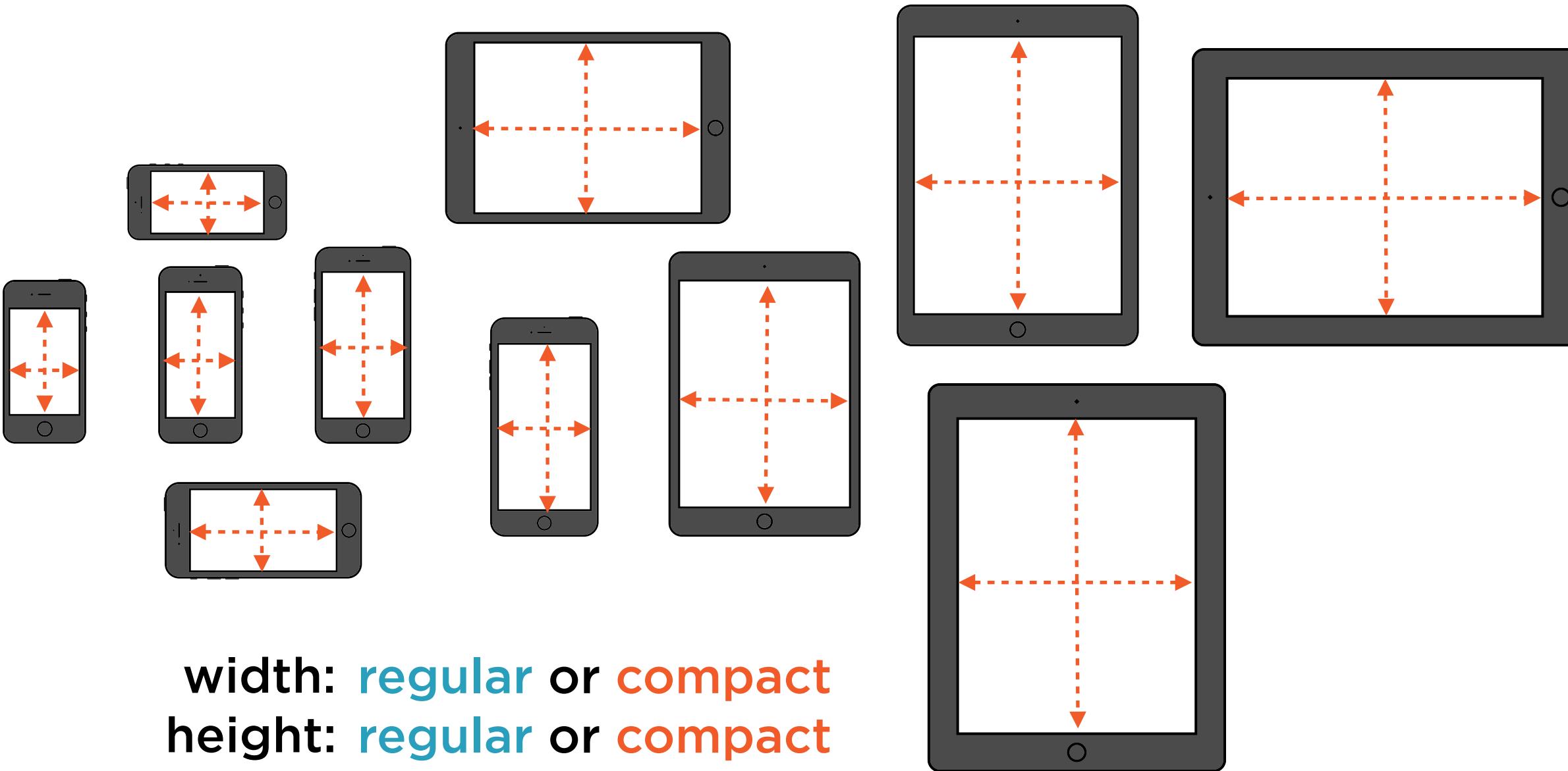


Weight Class: Minimumweight

iOS Size Class:



Size Classes



Size Classes for iPhones

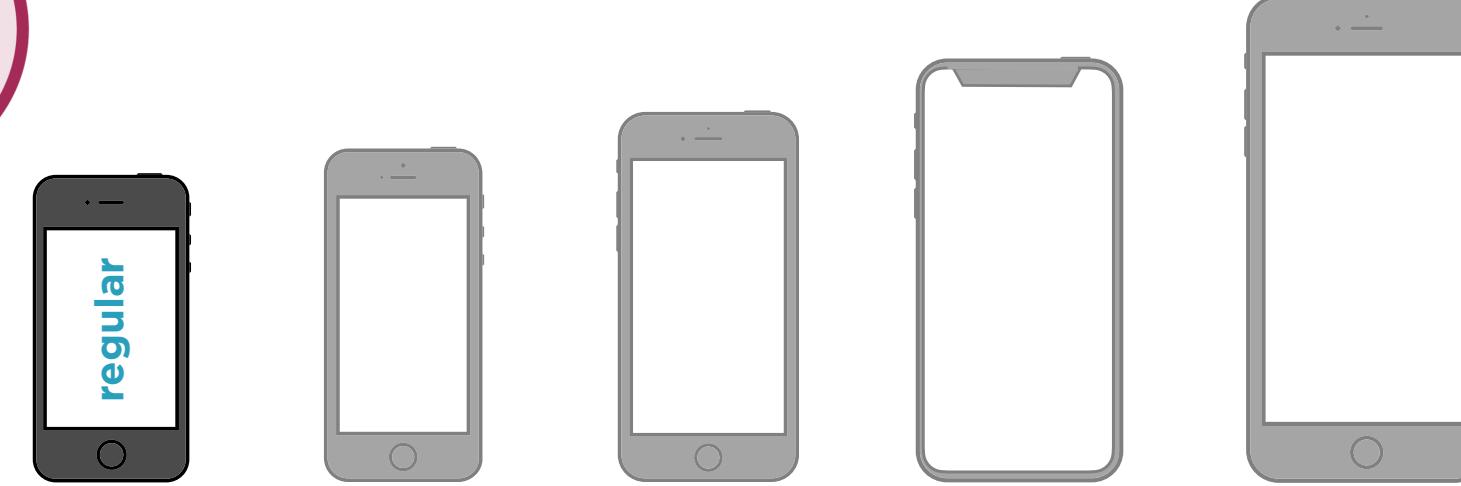
(portrait orientation)

width: compact

height: regular

Size Classes for iPhones

(portrait orientation)

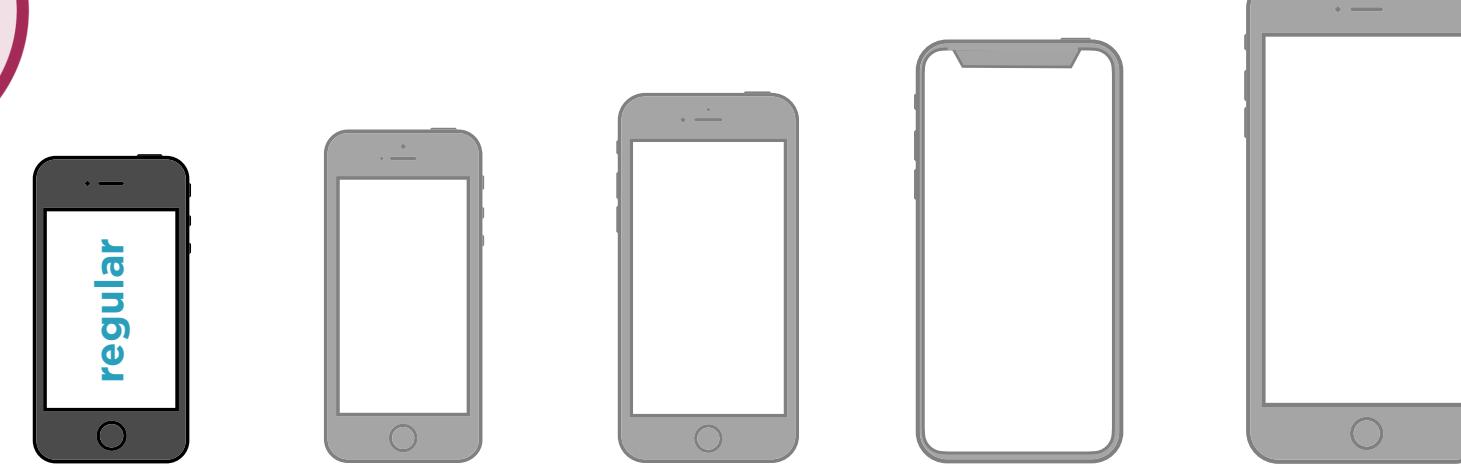


width: compact
height: regular

In portrait mode, the device is perfectly usable vertically.

Size Classes for iPhones

(portrait orientation)



width: compact
height: regular

But in portrait mode, horizontal interaction is never good.

Size classes are a way to focus on and think about a device's **usable space**.

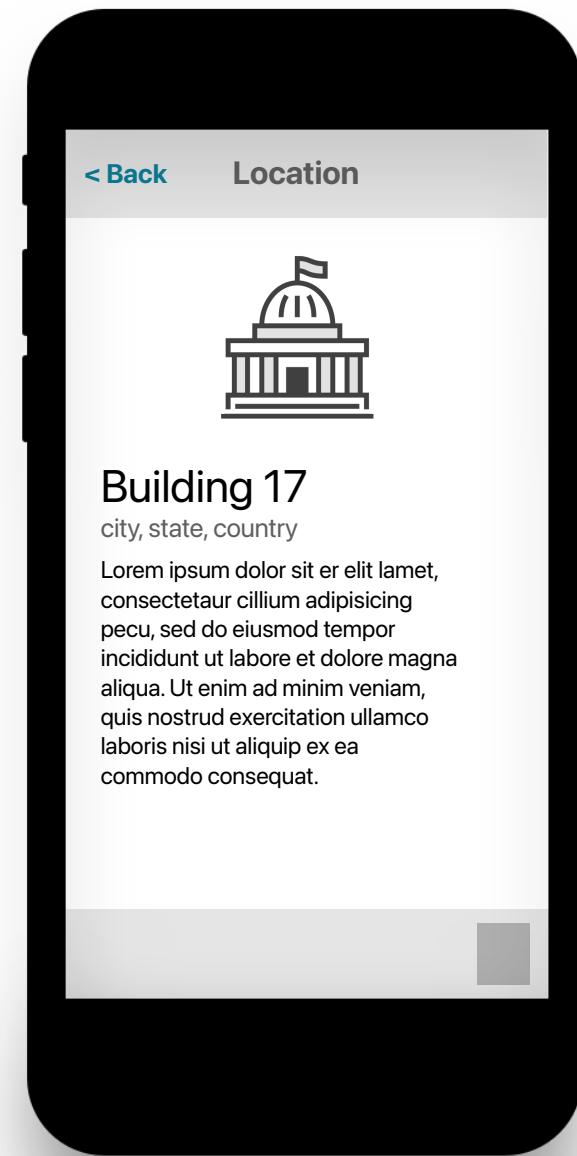
Size Classes for iPhones

(landscape orientation)

width: compact
height: regular



width: compact
height: compact



width: compact
height: compact

width: compact
height: regular

Size Classes for iPhones

(landscape orientation)



iPhone SE



iPhone 6S / 7 / 8



iPhone X / XS / 12 / 12 mini
iPhone 11 Pro / iPhone 12 Pro

width: compact
height: compact



iPhone 6 Plus / 7 Plus / 8 Plus



iPhone XR / XS Max / 11
iPhone 11 Pro Max / 12 Pro Max

width: regular
height: compact

Adding and Customizing Layout Variations

Getting Ready for the App Store

Publishing to the App Store



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

Duration: Full Time

Responsibilities:

- Proven working experience in software development
- 4+ years of working experience in iOS development
- Have published one or more iOS apps in the app store
- A deep familiarity with Swift (mandatory), Objective-C and Cocoa Touch
- Experience working with iOS frameworks such as Core Data, Core Animation, Core Graphics and Core Text, UIKit
- Solid experience on Test Driven Development and Behavioral Driven Development
- Deep experience in Source code management
- Experience with third-party libraries, Custom libraries and APIs
- Working knowledge of the general mobile landscape, architectures, trends, and emerging technologies

Solid experience on the full mobile

indeed.com

- Lead iOS-centric projects across the LIKEtoKNOW.it platform.
- Work with product and design team to inform planning and decisions.
- Execute designs with pixel perfect precision and flawless animation.
- Ensure the code base is sanely architected, well documented, and appropriately tested.
- Shepherd releases through to the App Store using our automated build system.
- Be a platform expert. Help to devise platform-specific features that leverage the unique properties of the iOS SDK.

What we're looking for..

- BS in Computer Science or equivalent experience (degree is not as important as being a talented iOS developer)
- Experience with shipping high-profile iOS applications. Preferred links to at least one live app in the App Store.
- 5+ years of experience in mobile development.
- 3+ years of development experience with iOS SDK.
- Proficiency in Objective-C. Top-notch ability to write clean code that is maintainable, documented and tested.
- Demonstrated mastery of UIKit, CoreData, Auto Layout, Core Animation.
- Significant experience efficiently interacting with RESTful network services.
- Skilled at application architecture, informed opinions on pros and cons of multiple MV* design patterns, dependency injection, asynchronous flow management techniques.
- Expertise writing unit tests in XCTests.

Full Job Description

Primary

- 2-5 years of experience in developing mobile apps for iOS platform.
- Proven experience with one or more apps published in the app store.
- Knowledge of Object Oriented concepts and a strong passion for software development.
- Good programming skills in Objective-C/Swift
- Experience with REST APIs
- Experience with iOS development tools such as: XCode, Cocoa, Cocoa Touch, XIB, storyboard
- Experience with third-party libraries and APIs.

Job Type: Contract

indeed.com

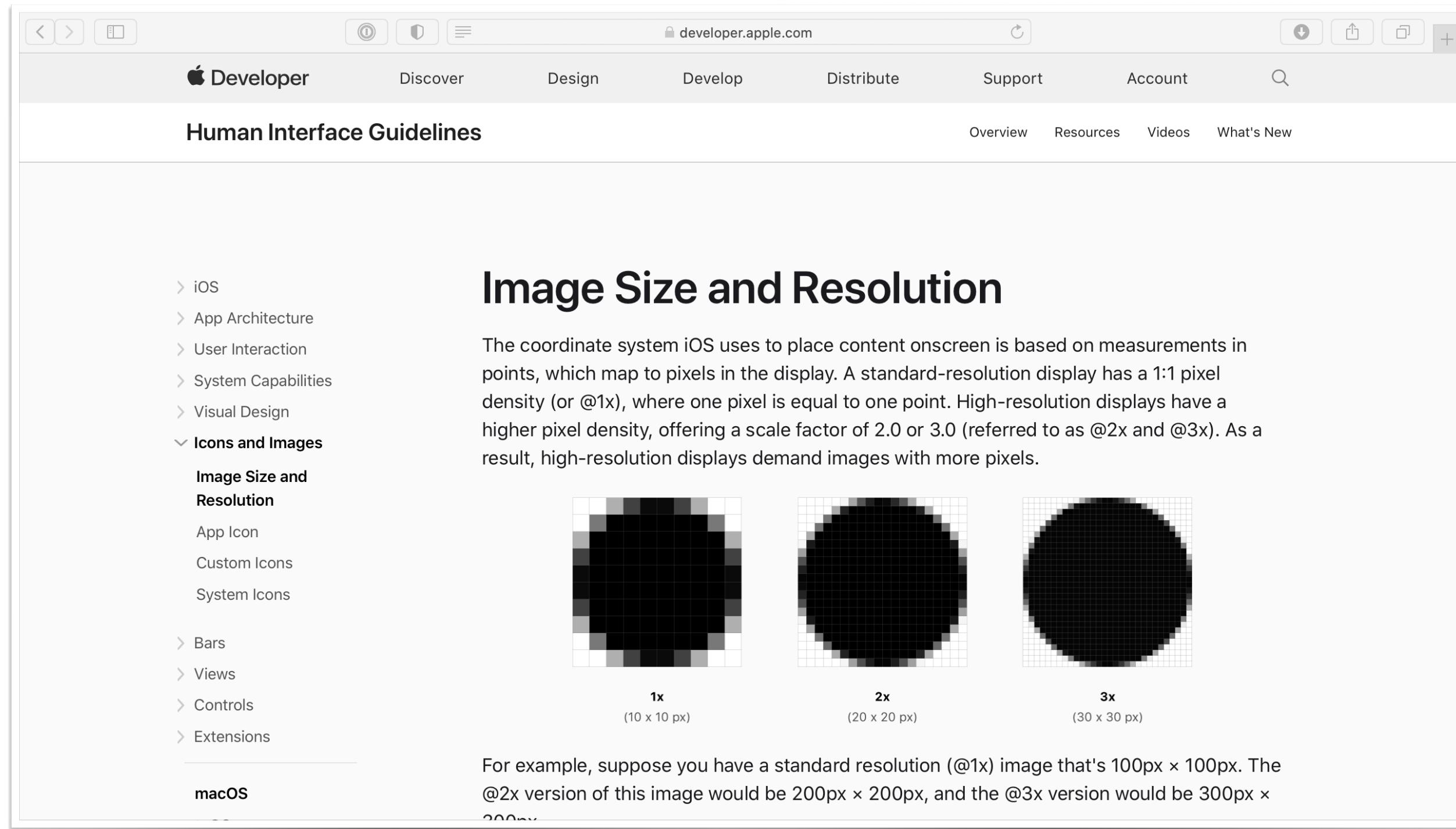


You've got to be a member of the
Apple Developer Program.

Adding Images and App Icons

iOS Image Dimensions

Points, not pixels



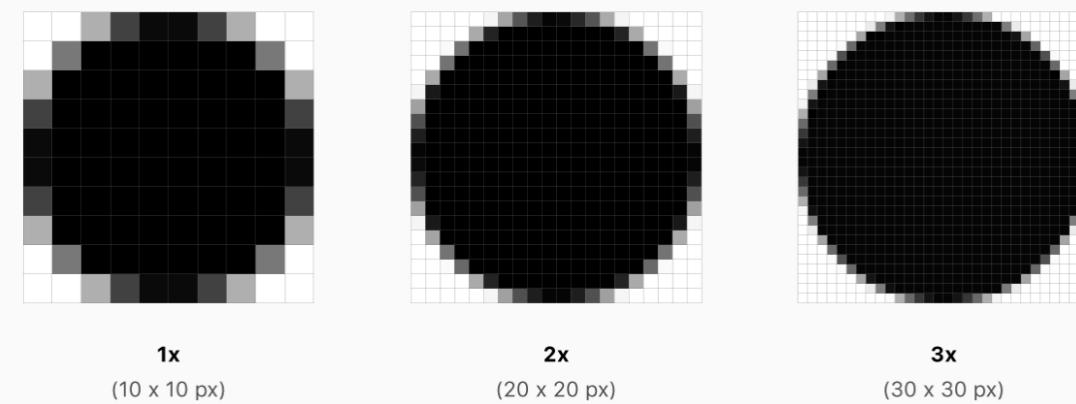
The screenshot shows a web browser window displaying the Apple Developer Human Interface Guidelines. The page title is "Human Interface Guidelines". The left sidebar contains navigation links for iOS, App Architecture, User Interaction, System Capabilities, Visual Design, Icons and Images (which is expanded to show "Image Size and Resolution", "App Icon", "Custom Icons", and "System Icons"), Bars, Views, Controls, and Extensions. The main content area is titled "Image Size and Resolution". It explains that iOS uses points as the coordinate system for placing content onscreen, which map to pixels in the display. It notes that standard-resolution displays have a 1:1 pixel density (@1x), while high-resolution displays have higher pixel densities (@2x or @3x). Three circular icons are shown, each with a grid overlay, labeled "1x (10 x 10 px)", "2x (20 x 20 px)", and "3x (30 x 30 px)". Below this, a text box states: "For example, suppose you have a standard resolution (@1x) image that's 100px × 100px. The @2x version of this image would be 200px × 200px, and the @3x version would be 300px × 300px".

> iOS
 > App Architecture
 > User Interaction
 > System Capabilities
 > Visual Design
 ▽ Icons and Images
 Image Size and Resolution
 App Icon
 Custom Icons
 System Icons
 > Bars
 > Views
 > Controls
 > Extensions

macOS

Image Size and Resolution

The coordinate system iOS uses to place content onscreen is based on measurements in points, which map to pixels in the display. A standard-resolution display has a 1:1 pixel density (or @1x), where one pixel is equal to one point. High-resolution displays have a higher pixel density, offering a scale factor of 2.0 or 3.0 (referred to as @2x and @3x). As a result, high-resolution displays demand images with more pixels.



1x
(10 x 10 px)

2x
(20 x 20 px)

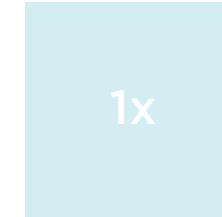
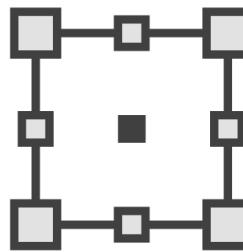
3x
(30 x 30 px)

For example, suppose you have a standard resolution (@1x) image that's 100px × 100px. The @2x version of this image would be 200px × 200px, and the @3x version would be 300px × 300px.

iOS Image Dimensions

Points, not pixels

50 pt x 50 pt



50 px x 50 px
someImage.png

iPhone 8 Plus (6 Plus, 7 Plus)
iPhone 12, 11, X

Retina screens



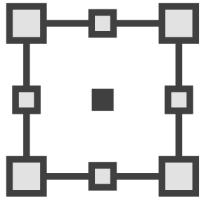
100 px x 100 px
someImage@2x.png



150 px x 150 px
someImage@3x.png

iPhone Application Icon

60 pt x 60 pt



iPhone 8 / iPad / iPad Pro



120 px x 120 px

iPhone 8 Plus / iPhone 12



180 px x 180 px

Human Interface Guidelines

[Overview](#)[Resources](#)[Videos](#)[What's New](#)

Apple Design Resources

Design apps quickly and accurately by using Sketch, Photoshop, and XD templates, guides, and other resources.



iOS

Apple Design Resources for iOS include Sketch, Photoshop, and Adobe XD templates, along with comprehensive UI resources that depict the full range of controls, views, and glyphs available to developers using the iOS SDK. These resources help you design apps that match the iOS design language. Icon and glyph production files are preconfigured to automate asset production using Sketch slices or Adobe Generator for Photoshop CC. Color swatches, dynamic type tables, and fonts are also included.

[View the iOS design guidelines >](#)



[iOS 14 \(Beta\) Sketch Library](#)

July 29, 2020
iOS 14 (96.1 MB)



[iOS 14 \(Beta\) Download for Sketch](#)

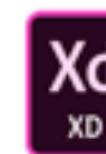


July 29, 2020
iOS 14 (90 MB)



[iOS 14 \(Beta\) Download for Photoshop](#)

July 29, 2020
iOS 14 (917 MB)



[iOS 14 \(Beta\) Download for Adobe XD](#)

October 23, 2020
iOS 14 (149.1 MB)

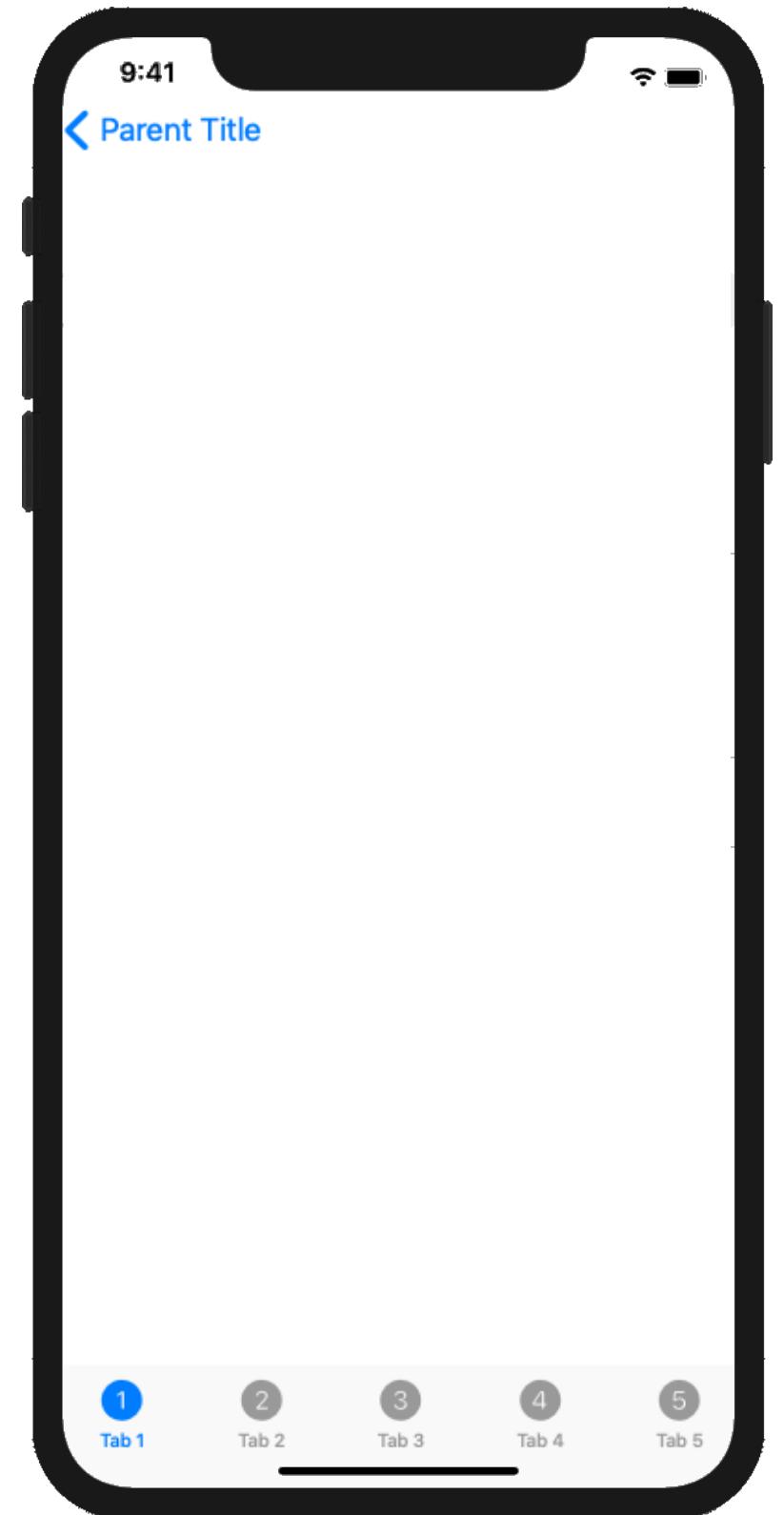


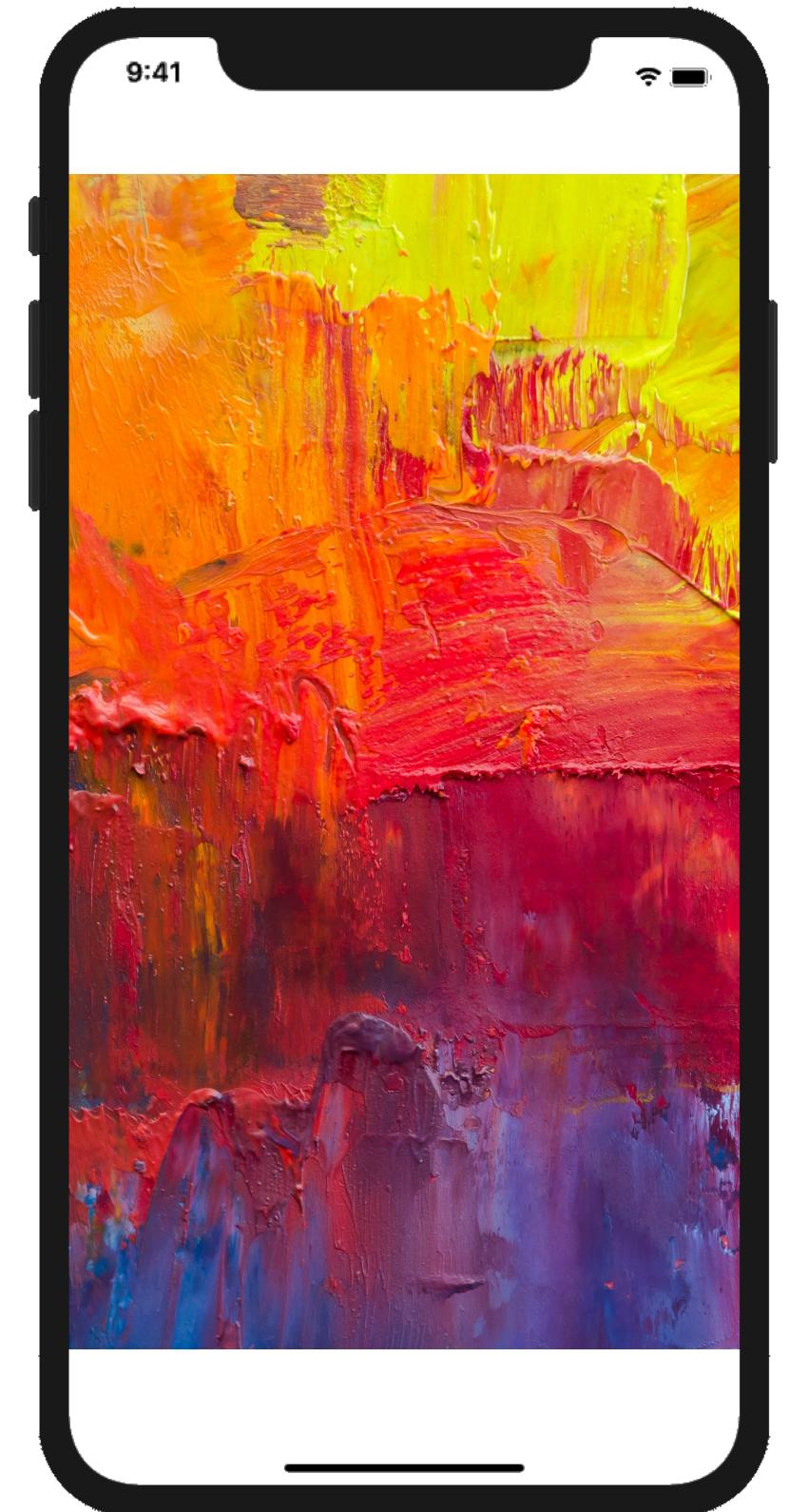
[Add iOS 13 Sketch Library](#)

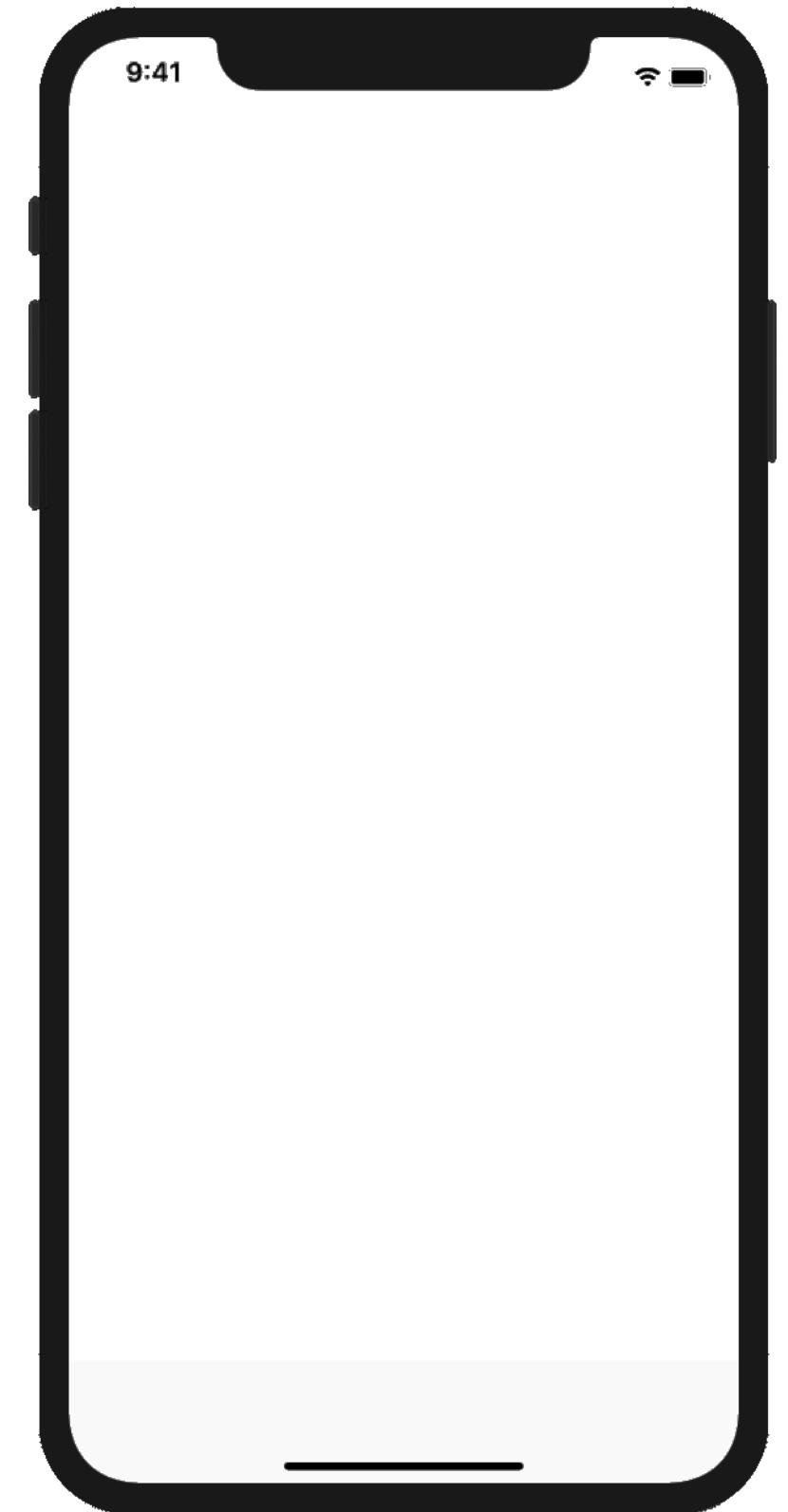
February 14, 2020 (v33)
iOS 13 (Requires Sketch 58 or greater)

Responding to Dark Mode

Creating an iOS App Launch Screen







Submit

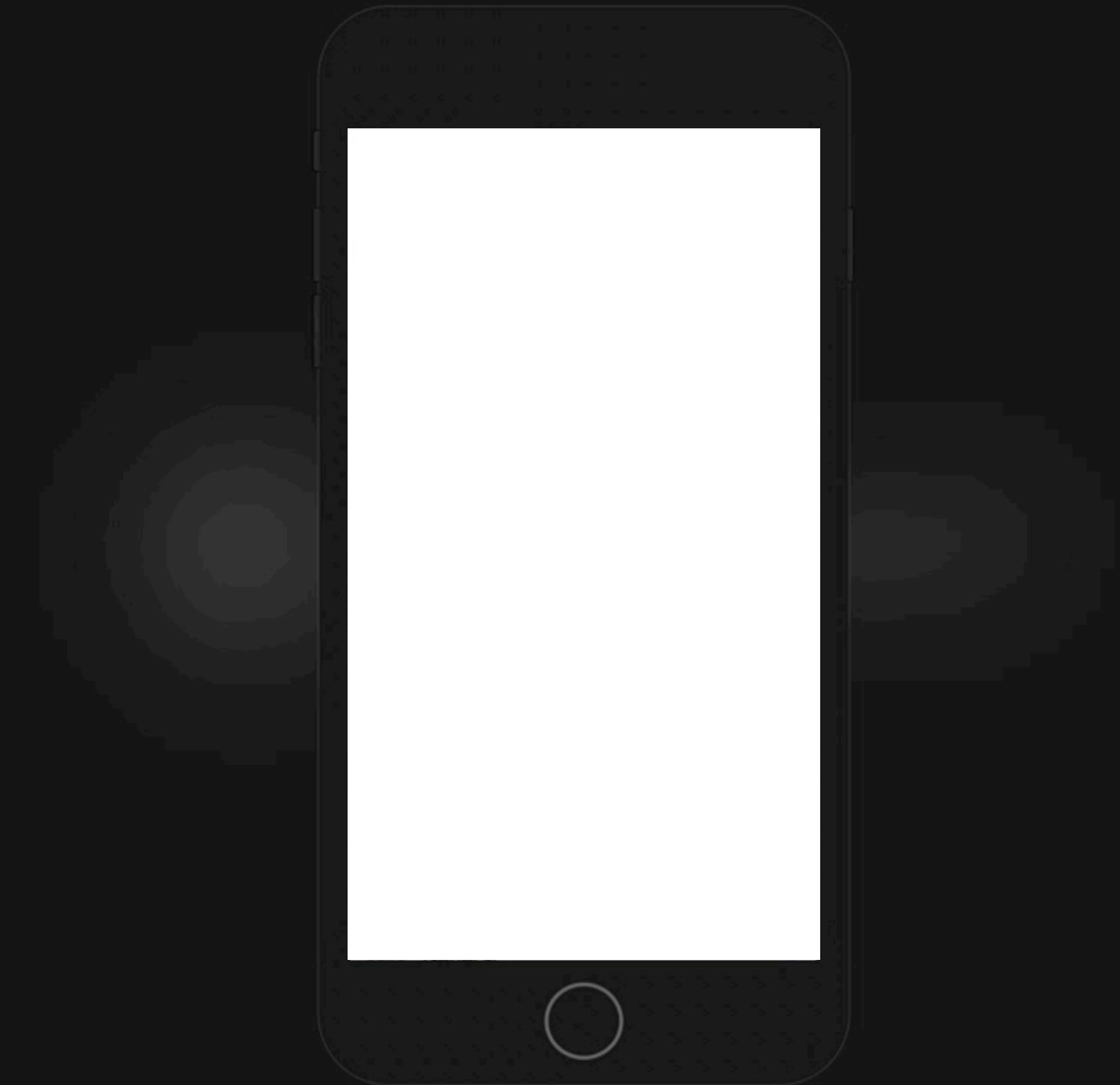
Soumettre

Absenden

Відправити

提交





Submitting an App for Review

The status of your (iOS) app, Night Watch PS, is now "Ready for Sale".

App Store Connect <no_reply@email.apple.com>
to me ▾

App Store Connect



Dear Andrew Bancroft,

The following app has been approved for the App Store:

App Name: Night Watch PS
App Version Number: 1.0
App SKU: NightWatch2020
App Apple ID: 1540457305

Continuing Your iOS Developer Journey

iOS 14: Getting Started



The End



Andrew Bancroft

@andrewcbancroft www.andrewcbancroft.com

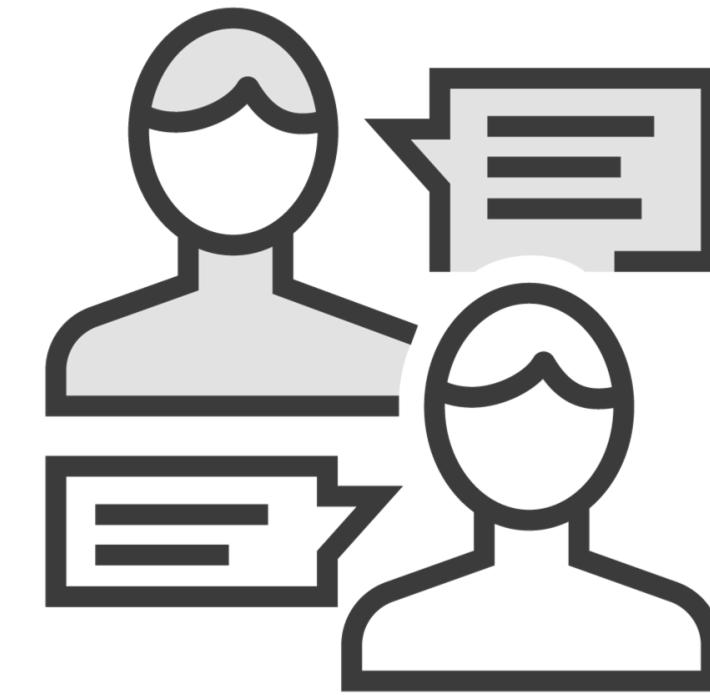
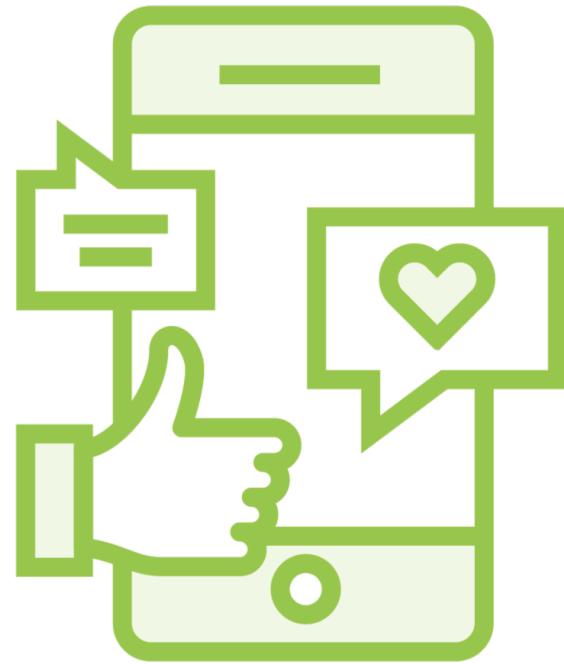


After you've “gotten started”, where do you go from here?

One of the most useful
things you can do as a
developer is to
read code and **write code**.

A lot of it.

The iOS Developer Community



**Thinking
in SwiftUI**



Chris Eidhof Florian Kugler

@chriseidhof

@floriankugler

Dave Verwer
@daveverwer
@iOSDevWeekly

Joe Groff
@jckarter

John Sundell
@johnsundell

Majid Jabrayilov
@mecid

Paul Hudson
@twostraws



Simon Allardice



Pluralsight Author

Follow

5461 Followers

Simon is a staff author at Pluralsight. With over three decades of software development experience, he's programmed in every discipline: from finance to transportation, nuclear reactors to game development. Prior to joining Pluralsight, Simon was the principal developer author at lynda.com. His...

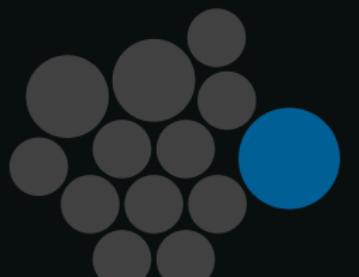
Show more...

CONTENT AUTHORED

15

All time

TOPICS AUTHORED



cloud platforms

TOTAL RATINGS

4,448

AVG CONTENT RATING

4.9

Content authored

Blockchain: Executive Briefing

Course · Beginner · 37m · May 11, 2020 · ★★★★★ (76)

AI: Executive Briefing

Course · Beginner · 40m · Feb 10, 2020 · ★★★★★ (129)

Cloud: Executive Briefing

Course · Beginner · 47m · Jan 2, 2020 · ★★★★★ (221)

TypeScript: The Big Picture

Course · Beginner · 44m · Nov 21, 2019 · ★★★★★ (322)



Thank you!