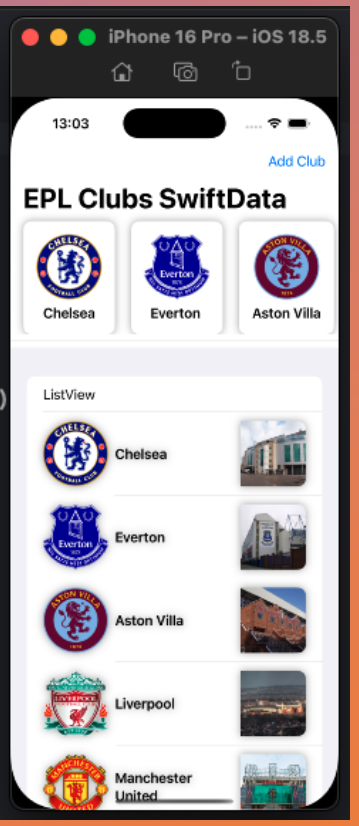
Mobile Application Development Week 1

Girish Lukka

```swift
//
//  EPL_SwiftDataApp.swift
//  EPL_SwiftData
//
//  Created by girish lukka on 20/10/2024.
//

import SwiftUI
import SwiftData

@main
struct EPL_SwiftDataApp: App {
    @StateObject var viewModel = ClubLocationViewModel()
    var body: some Scene {
        WindowGroup {
            ContentView().environmentObject(viewModel)
        }
        .modelContainer(for: EPLClub.self) { clubs in
            do { ••• } catch {
                print("Failed to pre-seed database.")
            }
        }
    }
}
```



Topics

- Welcome & Module Overview
- The Devlopment Environment
- SwiftUI Basics
- Anatomy of a SwiftUI App
- Views & Modifiers
- SwiftUI App Lifecycle
- Wrap-UP & Roadmap

Welcome to Mobile Application Development

- Welcome to your final-year Mobile App Development module
- Build **real iOS apps** using Swift & SwiftUI
- Learn how to design, code, and deploy apps across Apple platforms
- Get hands-on from Day 1 → today we dive into Xcode & SwiftUI basics

Week 1 is Crucial

- Both lecture & lab sessions lay the **foundation**
- Today: tools, SwiftUI basics, app structure, Xcode project setup
- Lab: hands-on in Xcode
- ⚠️ Missing early sessions makes catch-up very hard
- Attend, ask questions, and stay engaged

What You'll Build & How You're Assessed

- **Labs → Build three apps:**
- 🧮 Calculator (UI layout & interaction) – **under pins Lab-Practical assessment**
- 📍 Location & Maps (geolocation + map rendering) – **foundation for coursework**
- ☁️ Weather (live APIs & real-time data)) – **foundation for coursework**
- **Assessment:**
- **Week 7** → **Lab-Practical (live coding test)**
- Coursework → **Capstone mashup app**
  - Combines UI, UX, APIs, navigation, generics
  - Showcase for your portfolio
- By the end → you'll be able to build functional, scalable, and user-friendly apps

Essential iOS Development Tools

Xcode

- **The main workshop.** The IDE for writing code, designing UIs, debugging, and shipping to the App Store.

Swift

- **The programming language.** Used to write your app's logic and functionality.

SwiftUI

- *SwiftUI:* Modern & declarative (beginner-friendly).

iOS Simulator

- **The virtual device.** Lets you test your app on a simulated iPhone or iPad directly on your Mac.

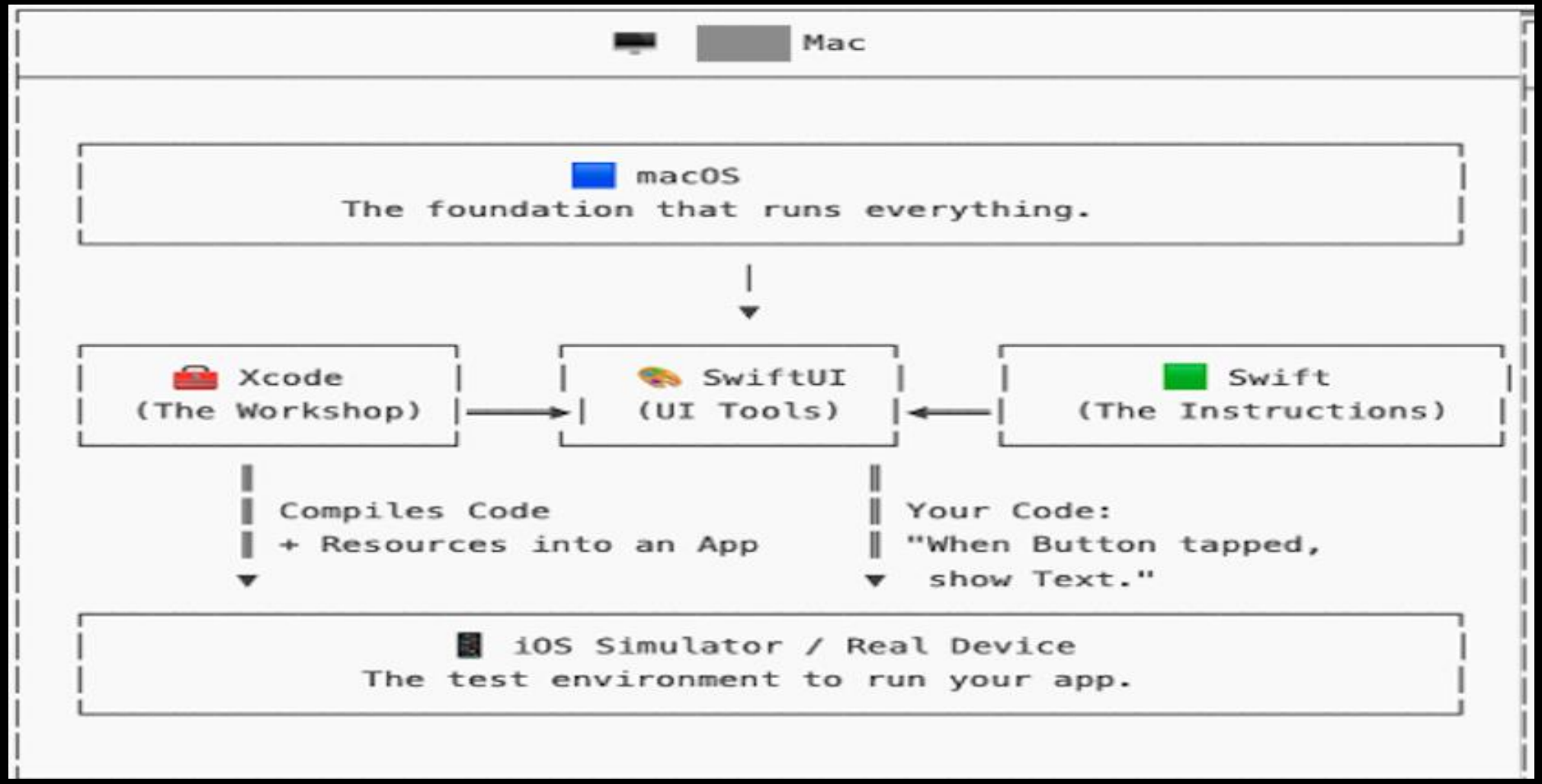
macOS

The foundation.

- The required operating system that runs all the tools above.

All tools are free and integrated into Apple's ecosystem.

The iOS/macOS App Development Stack



Xcode - Interface Layout

Xcode File Edit View Find Navigate Editor Product Debug Integrate Window Help

lifeCycleDemo
- lifeCycleDemo
  - lifeCycleDemo
    - Preview Content
    - Assets.xcassets
    - ContentView.swift
    - lifeCycleDemoApp.swift

lifeCycleDemo ⟩ iPhone 16 Pro (18.5)

lifeCycleDemo: **Ready** | Today at 13:47

ContentView.swift

lifeCycleDemo ⟩ lifeCycleDemo ⟩ ContentView.swift ⟩ body

```
1  //
2  //  ContentView.swift
3  //  lifeCycleDemo
4  //
5  //  Created by girish lukka on 20/01/2025.
6  //
7
8  import SwiftUI
9
10 struct ContentView: View {
11     // MARK:  monitor the scene's lifecycle within the views use
           the @Environment(\.scenePhase) property wrapper:
12
13     @Environment(\.scenePhase) private var scenePhase
14     @State private var message: String = "Waiting for scene phase
           change..."
15
16     var body: some View {
17         VStack(alignment: .center, spacing: 20)
18             Text("Current Scene Phase:")
19                 .font(.headline)
20             Text("\(scenePhase.description)")
21                 .font(.title)
22                 .foregroundColor(.blue)
23             Text(message)
24                 .padding()
25                 .multilineTextAlignment(.center)
26         }
27         .padding()
28         .onChange(of: scenePhase) { oldValue, newValue in
29             switch newValue {
30             case .active:
31                 message = "App is active."
```

ContentView

Current Scene Phase:

Background

Waiting for scene phase change...

Automatic – iPhone 16 Pro

Line: 17 Col: 13

Vertical Stack
Spacing 20
Alignment

Modifiers

Accessibility
Label Inherited
Identifier Inherited

Padding
Padding Default

Frame
Size Inher Width Inher Height

On Change
Of scenePhase
Argument 2 { oldValue, ne...

Toolbar (top)

Run/Stop app (Play/Stop buttons)

Select device or simulator

Scheme selector (choose which target to run)

Library of Views, Modifiers, Code Snippets, media and symbols that can be dropped in code editor.

Navigator Panel (Left)

File navigator: Browse your project files

Search: Find code across your project

Debug: View breakpoints and logs

Editor (center)

Where you write code or design UI.

Switch modes: Standard, Assistant (two editors), or Canvas (for SwiftUI previews).

Canvas (for SwiftUI)

Live preview of SwiftUI views.

Resume button: Refresh preview

Inspector updates: Drag and drop UI elements.

Inspector (right)

Shows properties of selected items (UI elements, code, or files).

Quick access: Cmd + Option + 0

Debug Area (Bottom)

Console: See app output and errors

Variables: Inspect values while debuggingDebug Area (Bottom)

Console: See app output and errors

Variables: Inspect values while debugging

Filter Auto Filter Executable Previews Filter

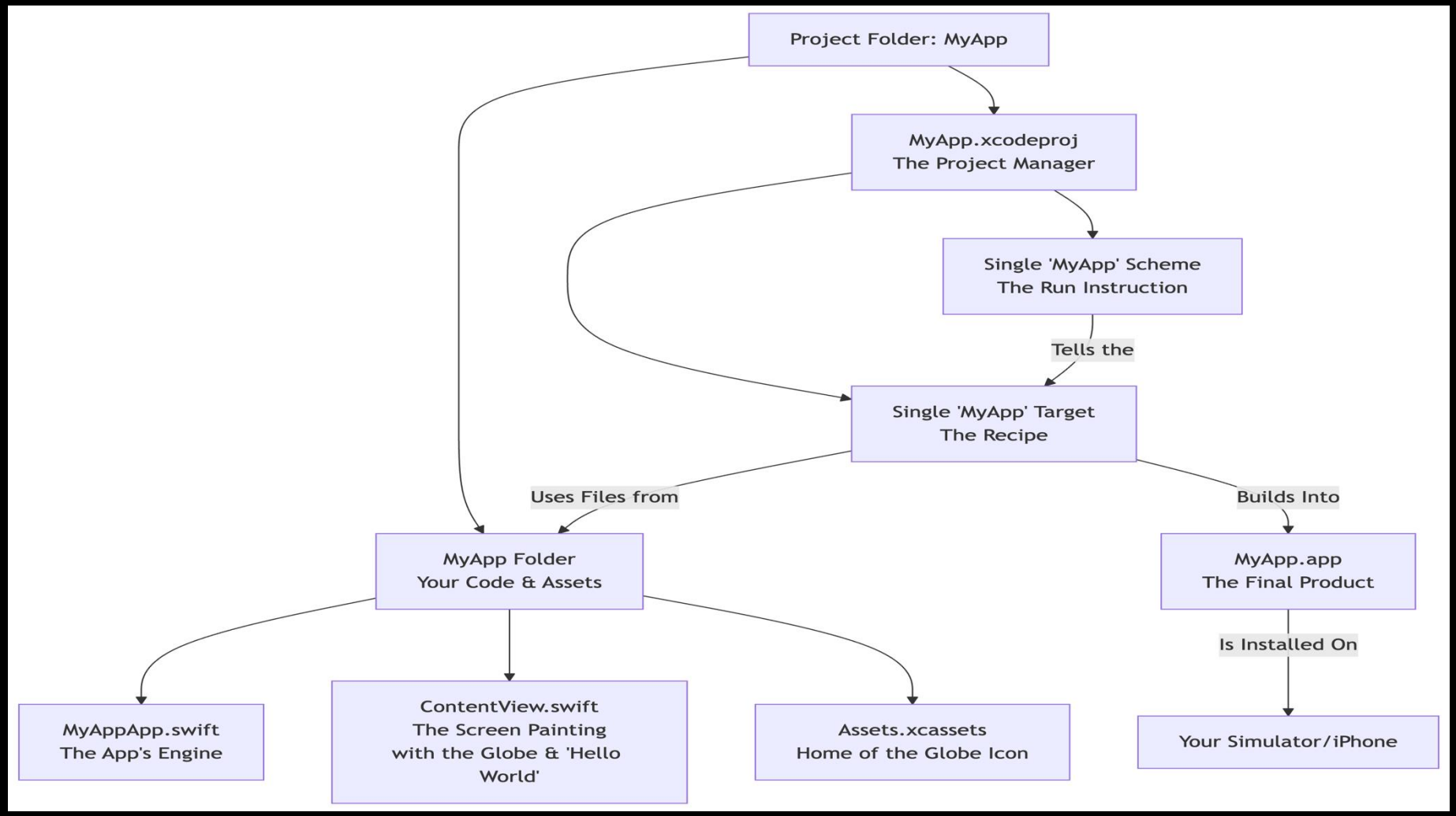
Xcode Project Anatomy: Files & Folders

```
📦 MyApp (Root Folder)
│
├── 📄 MyApp.xcodeproj
│      ├ Targets
│      │   ├ MyApp (Main App)
│      │   ├ MyAppTests (Unit Tests)
│      │   └ MyAppUITests (UI Tests)
│      └ Build Settings & Schemes
│
├── 📂 MyApp (Source Code)
│      ├ MyAppApp.swift    → @main struct MyAppApp : App { ... }  <-- ENTRY POINT
│      ├ ContentView.swift → Root SwiftUI View (UI starts here)
│      ├ Assets.xcassets   → App icons, colors, images
│      ├ Info.plist        → Bundle ID, permissions, etc.
│      └ Preview Content   → For SwiftUI live previews
│
├── 📂 MyAppTests
│      └ Unit test files
│
└── 📂 MyAppUITests
       └ UI test files
```

.xcodeproj
"brains" of project: holds targets, schemes, build phases, settings

Info.plist
key-value config file telling IOS how to run the app.

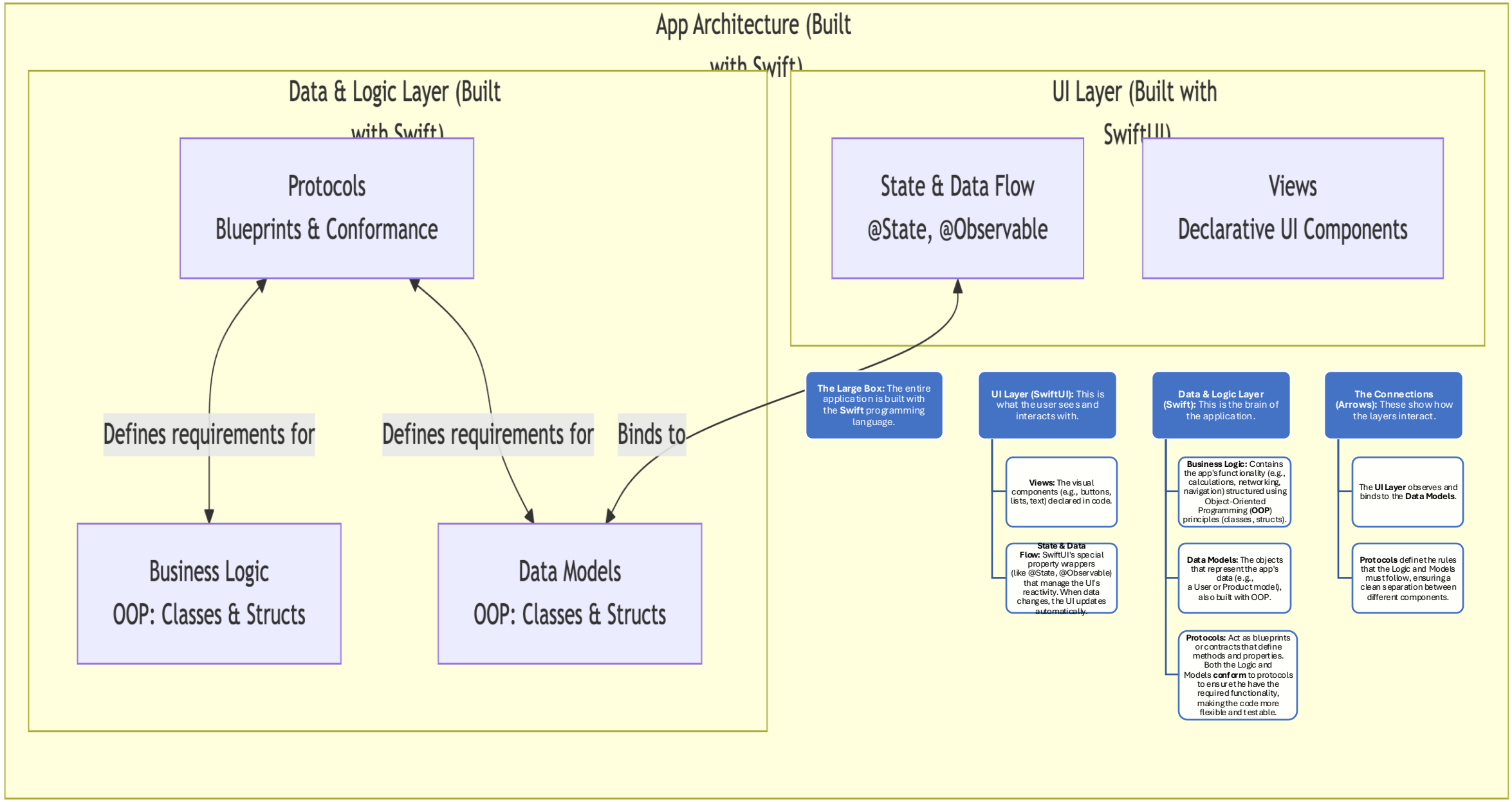
Anatomy of a Basic Xcode Project (iOS/macOS)



IOS app

- An iOS app build typically uses SwiftUI (written in Swift) for the UI layer, while data and logic are managed in Swift through object-oriented and protocol-oriented patterns.
- iOS apps use SwiftUI for the interface and Swift for managing data and app logic.

ios app


App Architecture (Built with Swift)
Data & Logic Layer (Built with Swift)
Protocols
Blueprints & Conformance
Business Logic
OOP: Classes & Structs
Data Models
OOP: Classes & Structs
Defines requirements for
Defines requirements for
Binds to
UI Layer (Built with SwiftUI)
State & Data Flow
@State, @Observable
Views
Declarative UI Components
The Large Box: The entire application is built with the Swift programming language.
UI Layer (SwiftUI): This is what the user sees and interacts with.
Views: The visual components (e.g., buttons, lists, text) declared in code.
State & Data Flow: SwiftUI's special property wrappers (like @State, @Observable) that manage the UI's reactivity. When data changes, the UI updates automatically.
Data & Logic Layer (Swift): This is the brain of the application.
Business Logic: Contains the app's functionality (e.g., calculations, networking, navigation) structured using Object-Oriented Programming (OOP) principles (classes, structs).
Data Models: The objects that represent the app's data (e.g., a User or Product model), also built with OOP.
Protocols: Act as blueprints or contracts that define methods and properties. Both the Logic and Models conform to protocols to ensure they have the required functionality, making the code more flexible and testable.
The Connections (Arrows): These show how the layers interact.
The UI Layer observes and binds to the Data Models.
Protocols define the rules that the Logic and Models must follow, ensuring a clean separation between different components.

SwiftUI - Overview

- One framework for **iOS, iPadOS, macOS, watchOS, tvOS**
- Build interfaces for all Apple devices with the **same tools & APIs**
- **Design-driven** approach (think Dreamweaver for apps)
- Use **views + modifiers** (like CSS styles) to create layouts quickly
- Interfaces feel **intuitive & web-like**

Declarative vs Imperative

- **Declarative** → say *what* you want (framework figures out *how*)
- **Imperative** → spell out *how* to get it step by step
- 🍞 *Apple analogy*: “Declarative is like ordering avocado toast — you say what you want, not every step of making it.” ”

🎨 Imperative (UIKit-style)

swift

```swift
let label = UILabel()
label.text = "Hello"
label.textColor = .blue
label.frame = CGRect(x: 20, y: 50, width: 200, height: 50)
view.addSubview(label)
```

✨ Declarative (SwiftUI)

swift

```swift
Text("Hello")
    .foregroundColor(.blue)
    .padding()
```

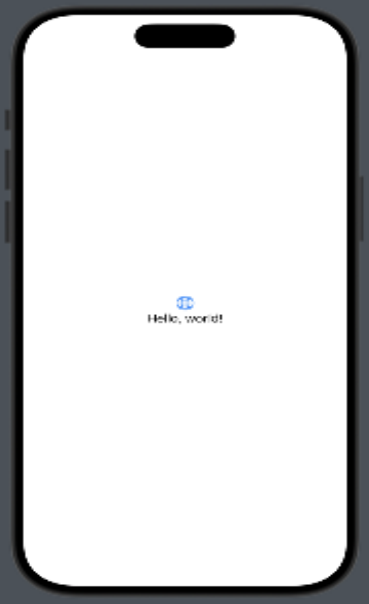
Anatomy of a SwiftUI app – MyApp.swift

- SwiftUI-based code will only run on devices running iOS 13 onwards
- App can be deployed on IOS, iPadOS, macOS watchOS and tvOS with little change.

```swift
import SwiftUI   // Import SwiftUI framework
// @main marks the entry point of the app
@main
struct myAppApp: App {   // Conforms to the App protocol

    // The 'body' describes the structure of the app
    var body: some Scene {
        // WindowGroup creates a window (on iOS, the main app window)
        // Inside it, we define which View appears first (the root view)
        WindowGroup {
            ContentView()  // The first screen of the app
        }
    }
}
```

Anatomy of a SwiftUI app – ContentView.swift

```swift
import SwiftUI   // Imports the SwiftUI framework for declarative UI

// Defines a struct that represents a screen or view in the app
struct ContentView: View {

    // Every SwiftUI view must implement this computed property
    // `body` describes what the UI looks like
    var body: some View {

        // VStack arranges its child views vertically (top to bottom)
        VStack {

            // A system-provided image (SF Symbol "globe")
            Image(systemName: "globe")
                .imageScale(.large)      // Makes the symbol larger
                .foregroundStyle(.tint)  // Applies the current accent/tint color

            // A simple text label
            Text("Hello, world!")
        }
        .padding()  // Adds spacing around the VStack content
    }
}

// This is a preview provider — lets Xcode show a live preview of ContentView
#Preview {
    ContentView()
}
```



🔑 SwiftUI App Key Concepts

- **@main** → Entry point of the app (where execution starts)
- **App protocol** → Defines the whole app (like a “director”), must have a body returning one or more **Scene**
- **Scene** → Container for windows (iOS = one main window, macOS = can have many)
- **WindowGroup** → A Scene that manages app windows/screens, usually holds the **root view**
- **ContentView (View protocol)** → The app’s root UI screen (what the user sees first)
- **body: some View** → Describes layout of a screen (Text, Button, VStack, etc.)

🔑 SwiftUI Key Terms

🔑 SwiftUI Key Terms

@main
Entry point of the app → tells SwiftUI where execution starts

MyApp: App
Your app's "director" → conforms to 'App' protocol, defines lifecycle & structure

var body: some Scene
App's main scene container → describes what the app shows (windows, UI)

WindowGroup
Scene that manages windows/screens → holds root view (usually 'ContentView)

ContentView: View
A UI screen → conforms to 'View', represents what the user sees

var body: some View
Describes the layout → builds UI hierarchy (Text, Button, VStack, etc.)

🎭 SwiftUI App = Theatre Production

- **App (Director):** Orchestrates the whole play → app lifecycle & structure
- **Scenes (Views):** Acts of the play → each screen or segment of UI
- **Actors (Components):** Text, Image, Button, custom views → perform roles
- **Sets & Props (Styling/Data):** Colors, fonts, assets, and data as stage setup
- **Lighting & Effects (Animations):** Transitions & animations add flair
- **Audience (Users):** Experience & interact with the performance

🎭 SwiftUI App = Theatre Production

SwiftUI App = Theatre Production

App (Director): Orchestrates the whole play → app lifecycle & structure

Scenes (Views): Acts of the play → each screen or segment of UI

Actors (Components): Text, Image, Button, custom views → perform roles

Sets & Props (Styling/Data): Colors, fonts, assets, and data as stage setup

Lighting & Effects (Animations): Transitions & animations add flair

Audience (Users): Experience & interact with the performance

Var body: some View

var body: **some** View

means the **body property** returns an **opaque type** (some View).

- some View = **opaque return type**
- Hides the exact view type (e.g., Text, VStack, Button)
- Guarantees only: “This is a View”
- Benefit: **flexibility + simpler code**
- Without it: you’d need to **fully specify every exact type**, making code verbose and rigid

☕ var body: some View

- **What it means**
  - Describes **what the UI should look like**
  - SwiftUI decides **when and how to render it**
  - Always returns **“some View”** (a screen element)
- ☕ **Coffee Shop Analogy**
  - You order: *“I’d like a hot drink”*
  - Barista can choose latte, cappuccino, or tea
  - You don’t care which → only that it’s **a hot drink**
  - some View = flexibility, hides exact type
- ❌ **Without some**
  - You’d have to **name the exact drink every time**
    - “One medium latte with oat milk, extra hot, two sugars…”
  - In SwiftUI: you’d need to **fully specify every exact view type**
  - Becomes **verbose, rigid, and harder to maintain**

Views

What does this error mean?

```
import SwiftUI

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

Type 'ContentView' does not conform to protocol 'View'

Use of protocol 'View' as a type must be written 'any View'; this will be an error in a fut...

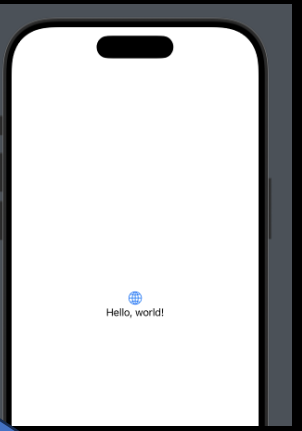
```
import SwiftUI

struct ContentView: View {
    var body: some View {

    }
}
```

What will this render and why is there no error?

```
import SwiftUI

struct ContentView: View {
    var body: Text {
        Text("Hello, World!")
    }
}


#Preview {
    ContentView()
}
```

some is missing but it renders, why?

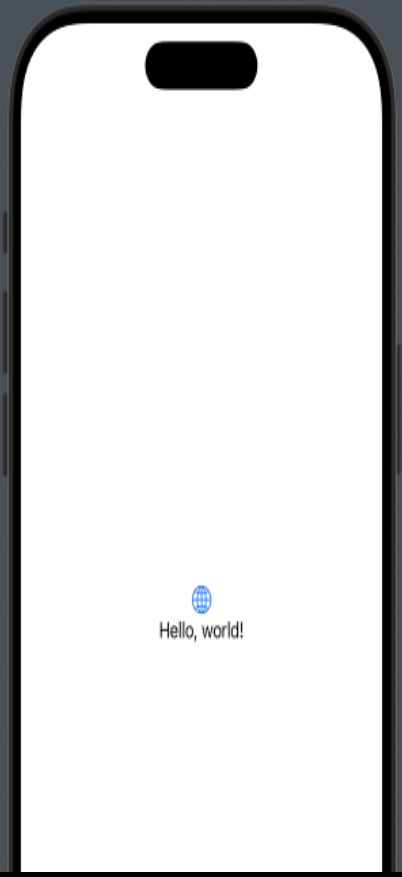
```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)
            Text("Hello, world!")
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```



multiple views - name them?

Compare view with "some" and without

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)
            Text("Hello, world!")
        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

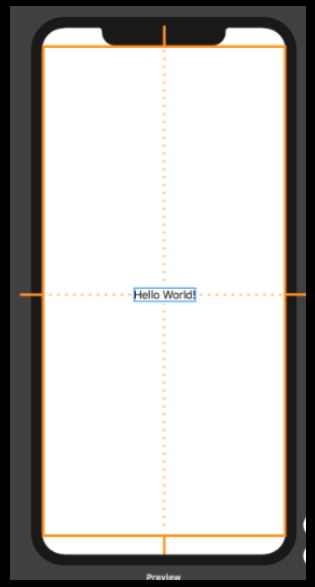



```swift
import SwiftUI

struct ContentView: View {
    var body: VStack<TupleView<(Image, Text)>> {
        VStack {
            Image(systemName: "globe")
            Text("Hello, world!")
        }

    }
}
#Preview {
    ContentView()
}
```

Trying to render same views as screen on left without using some – note specification required just to render and no modifications

View management – Layout Process

- **Parent Proposes Size for Child**
- First, the root view offers the text a proposed size – in this case, the entire safe area of the screen, represented by an orange rectangle.



Child Chooses its Size

- Text only requires that much size to draw its content.
- The parent has to respect the child's choice. It doesn't stretch or compress the child.

Parent Places Child in Parent's Coordinate Space

- And now the root view has to put the child somewhere, so it puts in right in the middle.

Views

Layout

- Control Group
- Depth Stack
- Geometry Reader
- Horizontal Stack
- Lazy Horizontal Grid
- Lazy Horizontal Stack
- Lazy Vertical Grid
- Lazy Vertical Stack
- Scroll View Reader
- Spacer
- Vertical Stack
- View That Fits

Paint

Controls

- Button
- Color Picker
- Date Picker
- Disclosure Group
- Edit Button
- Form
- Gauge
- Group Box
- Label
- Labeled Content
- Link
- List

Controls

- Menu
- Multi Date Picker
- Navigation Link
- Navigation Split View
- Navigation Stack
- Navigation View
- Outline Group
- Paste Button
- Picker
- Progress View
- Rename Button
- Scroll View

Controls

- Section
- Secure Field
- Share Link
- Sign In With Apple Button
- Slider
- Stepper
- Tab View
- Table
- Text
- Text Editor
- Text Field
- Toggle

Other

- Async Image
- Canvas
- Capsule
- Circle
- Color
- Container Relative Shape
- Content Unavailable View
- Divider
- Ellipse
- Empty View
- Group
- Image
- Path
- Rectangle
- Rounded Rectangle
- Timeline View

Modifier Samples

Other
- Allows Hit Testing
- Animation
- Content Transition
- Default App Storage
- Defers System Gestures
- Disabled

Shapes
- Fill
- Offset
- Rotation
- Scale
- Size
- Stroke

Controls
- Action Sheet
- Alert
- Badge
- Confirmation Dialog
- Context Menu
- Control Size

Style
- Background Style
- Button Style
- Control Group Style
- Date Picker Style
- Disclosure Group Style
- Form Style

List
- Delete Disabled
- Header Prominence
- List Item Tint
- List Row Background
- List Row Insets
- List Row Separator

Image
- Antialiased
- Image Rendering Mode
- Image Resizable
- Interpolation
- Symbol Rendering Mode
- Symbol Variant

Effects
- Accent Color
- Blend Mode
- Blur
- Border
- Brightness
- Button Border Shape

Gestures
- Gesture
- High Priority Gesture
- On Long Press Gesture
- On Long Touch Gesture
- On Tap Gesture
- Simultaneous Gesture

Events
- Digital Crown Rotatio
- Draggable
- Drop Destination
- Exports Item Provide
- File Exporter
- File Importer

Accessibility
- Accessibility Actions
- Accessibility Activation
- Accessibility Add Traits
- Accessibility Chart Des
- Accessibility Children
- Accessibility Custom C

Navigation Bar
- Navigation Bar Back
- Navigation Bar Hid
- Navigation Bar Title
- Navigation Title

Layout
- Alignment Guide
- Anchor Preference
- Aspect Ratio
- Background
- Background Preference Value
- Container Relative Frame

Text
- Allows Tightening
- Autocapitalization
- Autocorrection Disabled
- Baseline Offset
- Bold
- Disable Autocorrection

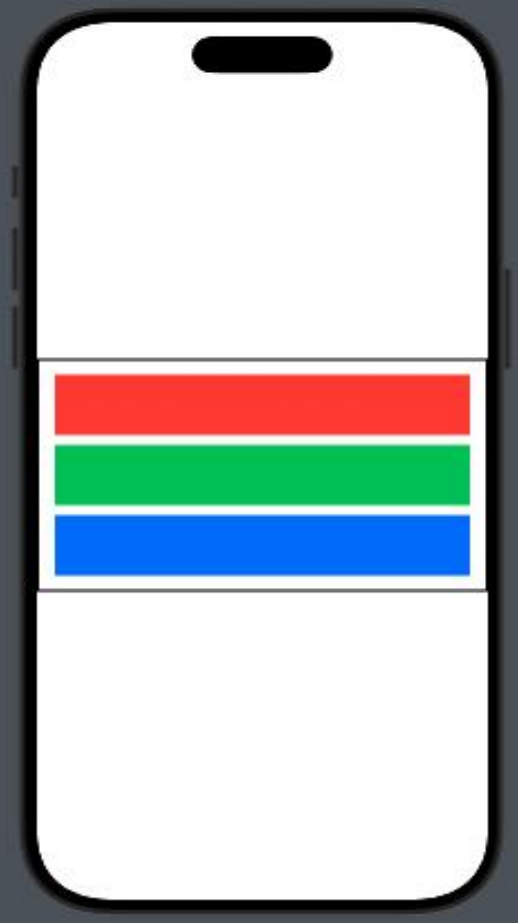
🎨 How SwiftUI Modifiers Work

- **Order matters** → applied top-to-bottom, each wraps the previous view
- **Types of modifiers:**
- 🧩 *Layout* → size & position (.padding(), .frame())
- 🎨 *Style* → appearance (.foregroundColor(), .font())
- ⚡ *Behavior* → interaction or hierarchy (.onTapGesture(), .sheet())
- **Key idea** → every modifier creates a *new view*, nested in sequence

```
Text("Hello")
  .padding()       // Creates: ModifiedContent<Text, _PaddingLayout>
  .background(.red) // Wraps: ModifiedContent<ModifiedContent<Text, _PaddingLayout>, _Backgroun
dStyleModifier<Color>>
```

Demo Code Views – Container VStack

```swift
import SwiftUI

struct VStackDemo: View {
    var body: some View {
        VStack(spacing: 10) {
            Rectangle()
                .fill(.red)
                .frame(height: 60)
            Rectangle()
                .fill(.green)
                .frame(height: 60)
            Rectangle()
                .fill(.blue)
                .frame(height: 60)
        }
        .padding()
        .border(.black, width: 2)
    }
}

#Preview {
    VStackDemo()
}
```

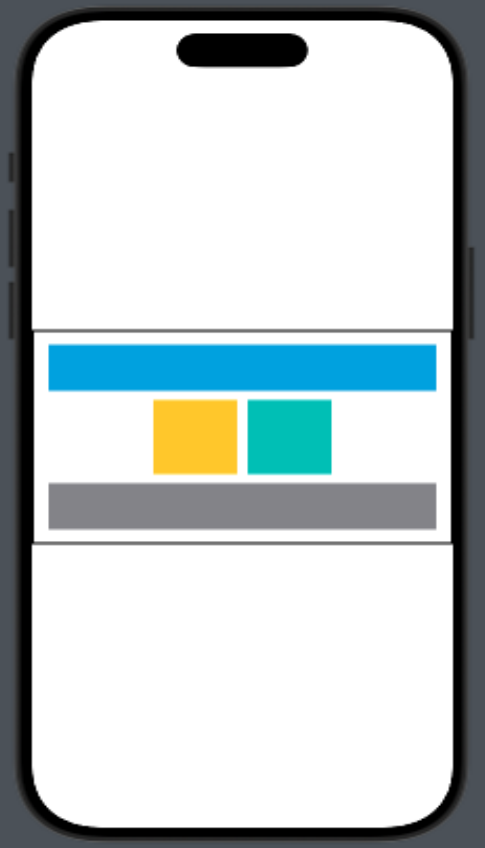


Demo Code Views – Container HStack

```swift
import SwiftUI

struct HStackDemo: View {
    var body: some View {
        HStack(spacing: 10) {
            Rectangle().fill(
                .orange)
                .frame(width: 80,
                height: 80)
            Rectangle().fill(
                .purple)
                .frame(width: 80,
                height: 80)
            Rectangle().fill(
                .pink)
                .frame(width: 80,
                height: 80)
        }
        .padding()
        .border(.black, width: 2)
    }
}

#Preview {
    HStackDemo()
}
```

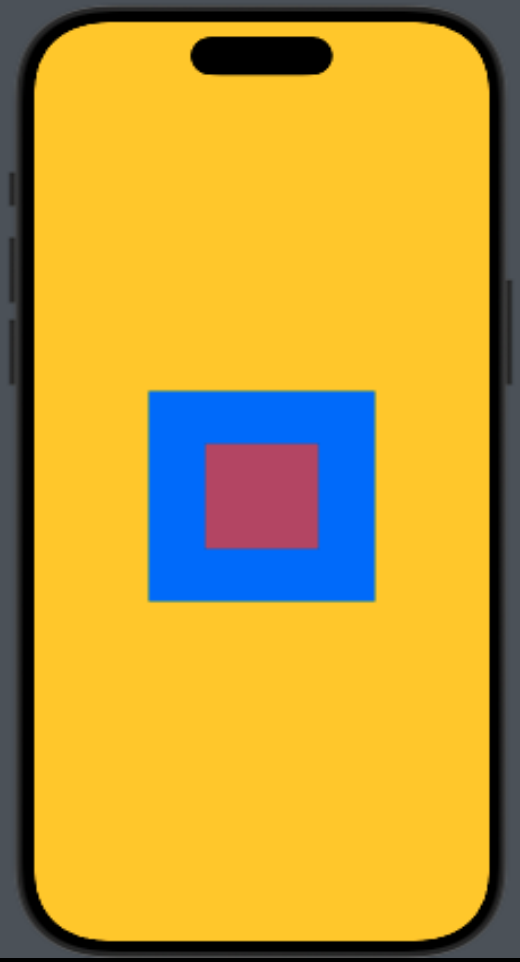
Demo Code Views – Container Combo

```swift
import SwiftUI

struct ComboStackDemo: View {
    var body: some View {
        VStack(spacing: 10) {
            Rectangle().fill(.cyan)
                .frame(height: 50)

            HStack(spacing: 10) {
                Rectangle().fill(
                    .yellow)
                    .frame(width: 80,
                    height: 80)
                Rectangle().fill(
                    .mint)
                    .frame(width: 80,
                    height: 80)
            }

            Rectangle().fill(.gray)
                .frame(height: 50)
        }
        .padding()
        .border(.black, width: 2)
    }
}

#Preview {
    ComboStackDemo()
}
```



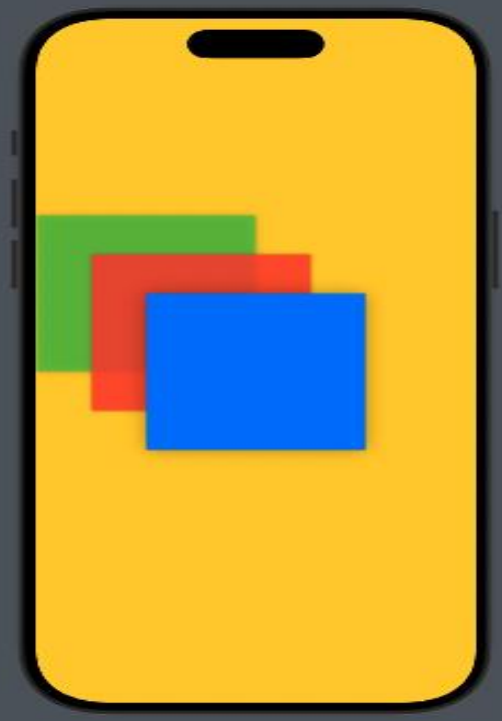
Demo Code Views – Container ZStack

```swift
import SwiftUI

struct ZStackDemo: View {
    var body: some View {
        ZStack {
            Rectangle().fill(.yellow)
                .ignoresSafeArea() //
                background
            Rectangle().fill(.blue)
                .frame(width: 200,
                height: 200)
            Rectangle().fill(.red
                .opacity(0.7))
                .frame(width: 100,
                height: 100)
        }
    }
}

#Preview {
    ZStackDemo()
}
```



Demo Code Views – Container ZStack Depth

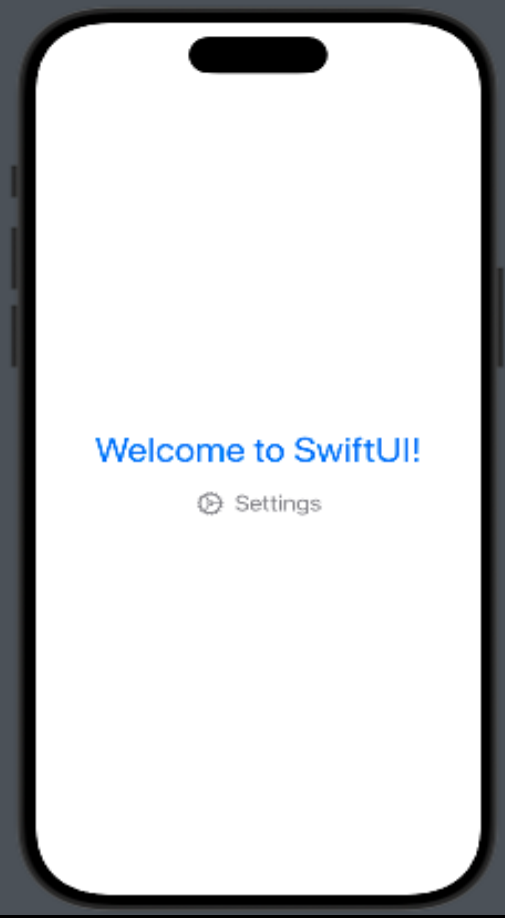
```swift
import SwiftUI

struct ZStackDepthDemo: View {
    var body: some View {
        ZStack {
            Rectangle().fill(.yellow)
                .ignoresSafeArea() // background

            // Back card
            Rectangle()
                .fill(Color.green.opacity(0.6))
                .frame(width: 200, height: 200)
                .offset(x: -100, y: -100)
                .shadow(radius: 5)

            // Middle card
            Rectangle()
                .fill(Color.red.opacity(0.8))
                .frame(width: 200, height: 200)
                .offset(x: -50, y: -50)
                .shadow(radius: 3)

            // Front card
            Rectangle()
                .fill(Color.blue)
                .frame(width: 200, height: 200)
                .shadow(radius: 10)
        }
        .frame(maxWidth: .infinity, maxHeight:
            .infinity)
    }
}

#Preview {
    ZStackDepthDemo()
}
```



Automatic – iPhone 16 Pro

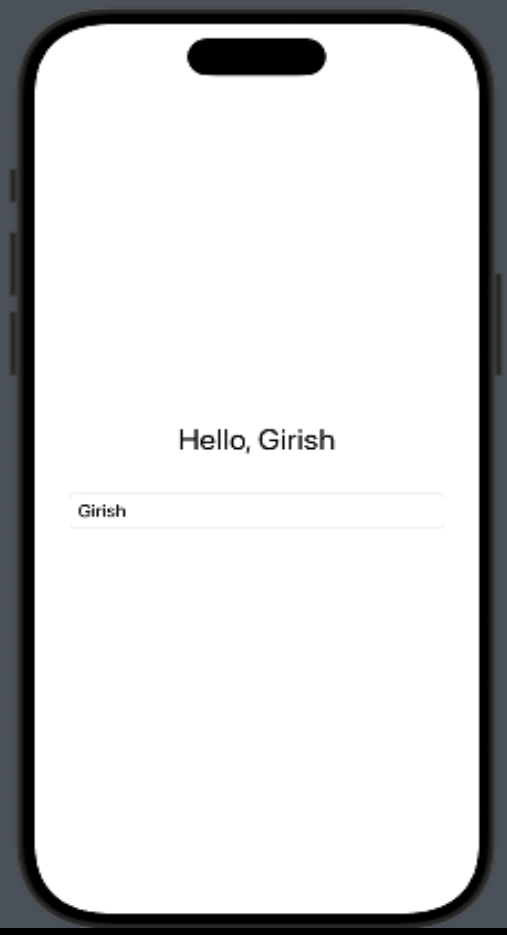
Demo Code Views – – Core View: Text&Label

```swift
import SwiftUI

struct TextAndLabelView: View {
    var body: some View {
        VStack(spacing: 20) {
            Text("Welcome to SwiftUI!")
                .font(.largeTitle)
                .foregroundStyle(.blue)

            Label("Settings", systemImage:
                "gear")
                .font(.title2)
                .foregroundStyle(.gray)
        }
        .padding()
    }
}

#Preview {
    TextAndLabelView()
}
```



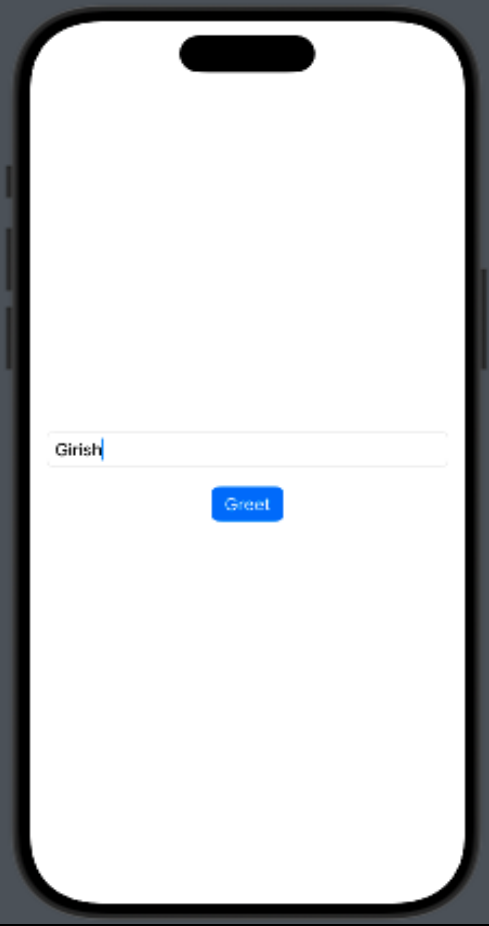
Demo Code Views – – Core View: TextField

```swift
import SwiftUI

struct TextFieldView: View {
    @State private var name = ""

    var body: some View {
        VStack(spacing: 20) {
            Text("Hello, \(name)")
                .font(.title)

            TextField("Enter your name", text:
                $name)
                .textFieldStyle(.roundedBorder)
                .padding()
        }
        .padding()
    }
}

#Preview {
    TextFieldView()
}
```



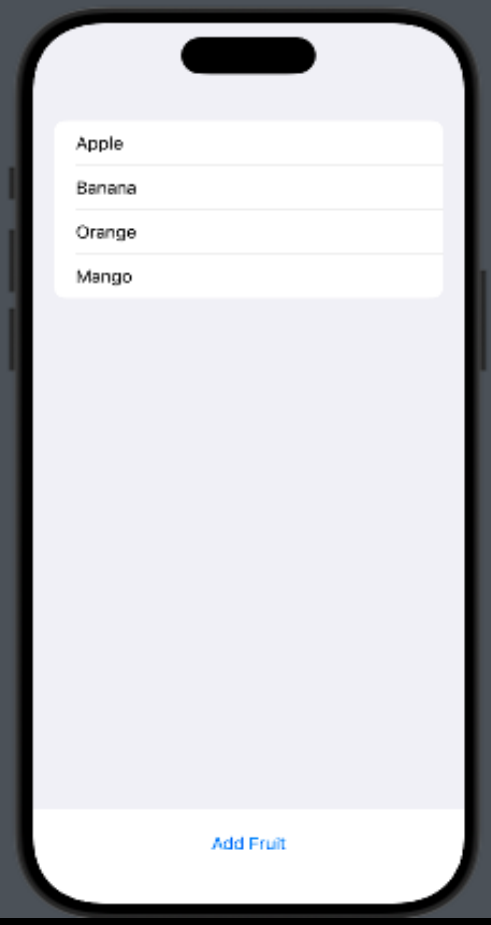
Demo Code Views – – Core View: ButtonView

```swift
import SwiftUI

struct ButtonView: View {
    @State private var name = ""

    var body: some View {
        VStack(spacing: 20) {
            TextField("Enter your name", text: $name)
                .textFieldStyle(.roundedBorder)

            Button("Greet") {
                print("Hello, \(name)!")

            }
            .buttonStyle(.borderedProminent)
        }
        .padding()
    }
}

#Preview {
    ButtonView()
}
```



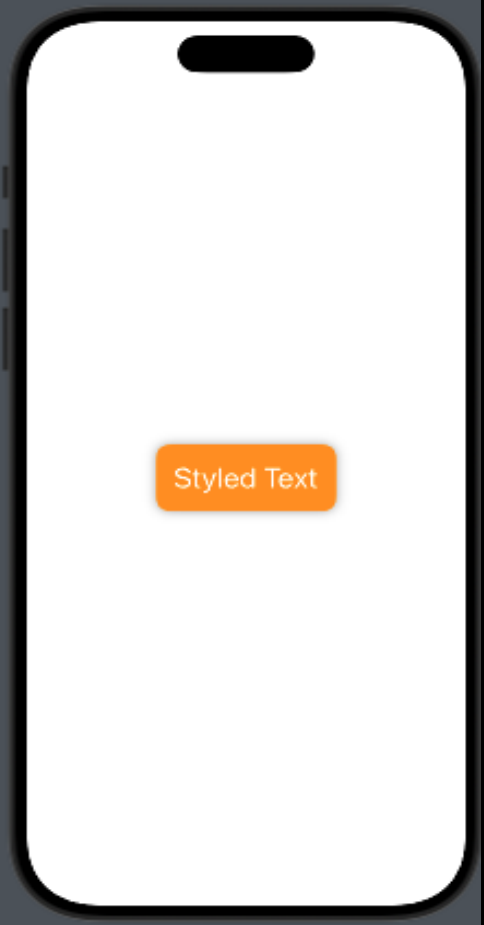
Demo Code Views – Core View: ListView

```swift
import SwiftUI

struct ListExampleView: View {
    @State private var fruits = ["Apple", "Banana",
        "Orange"]

    var body: some View {
        VStack {
            List(fruits, id: \.self) { fruit in
                Text(fruit)
            }

            Button("Add Fruit") {
                fruits.append("Mango")
            }
            .padding()
        }
    }
}

#Preview {
    ListExampleView()
}
```



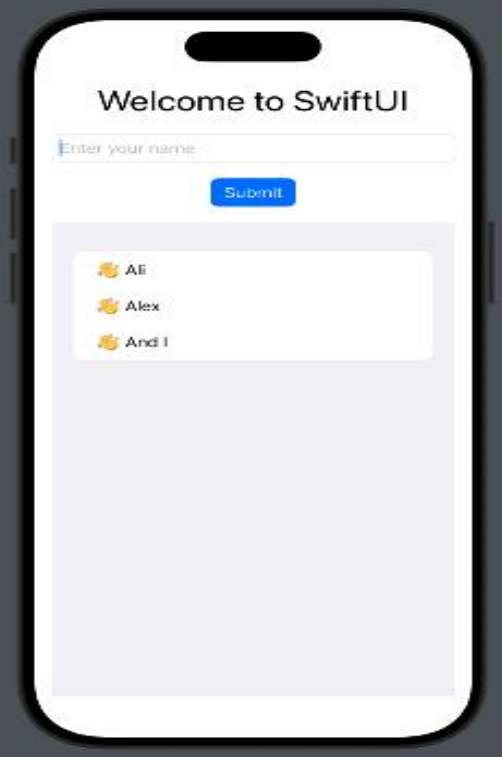
Demo Code Views – Core View: ModifierView

```swift
import SwiftUI

struct ModifiersView: View {
    var body: some View {
        Text("Styled Text")
            .font(.title)
            .foregroundStyle(.white)
            .padding()
            .background(.orange)
            .cornerRadius(12)
            .shadow(radius: 5)
    }
}

#Preview {
    ModifiersView()
}
```



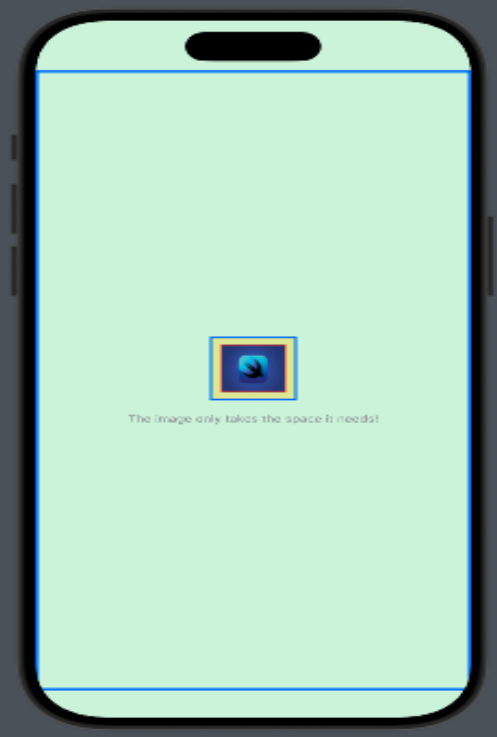
Demo Code Views – MiniAppView

```swift
import SwiftUI

struct MiniAppView: View {
    @State private var name = ""
    @State private var names: [String] = []

    var body: some View {
        VStack(spacing: 20) {
            Text("Welcome to SwiftUI")
                .font(.largeTitle)

            TextField("Enter your name", text: $name)
                .textFieldStyle(.roundedBorder)

            Button("Submit") {
                if !name.isEmpty {
                    names.append(name)
                    name = ""
                }
            }
            .buttonStyle(.borderedProminent)

            List(names, id: \.self) { name in
                Text("👋 \(name)")
            }
        }
        .padding()
    }
}

#Preview {
    MiniAppView()
}
```



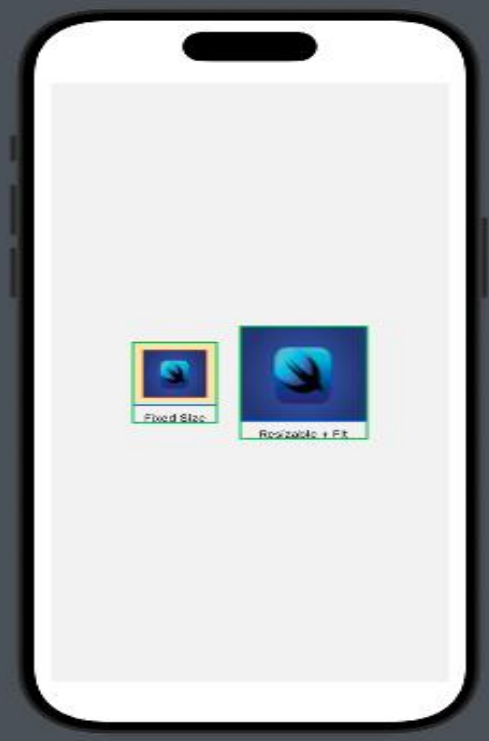
Demo Code Views – ImageLayoutView

```swift
import SwiftUI

struct ImageLayoutView: View {
    var body: some View {
        VStack(spacing: 16) {
            Image("swift") // Use an asset named
                "swift"
            // Uncomment to demonstrate resizing:
            // .resizable()
            // .scaledToFit()
                .border(.red) // Child's border
                .frame(width: 80, height: 80) // Child
                    sets its own size
                .border(.blue, width: 2) // Frame's
                    border
                .background(.yellow.opacity(0.3)) //
                    Child background

            Text("The image only takes the space it
                needs!")
                .font(.caption)
                .foregroundStyle(.gray)
        }
        // Parent VStack stretches to fill
        .frame(maxWidth: .infinity, maxHeight:
            .infinity)
        .border(.blue, width: 3) // Parent border
        .background(.green.opacity(0.2)) // Parent
            background
    }
}

#Preview {
    ImageLayoutView()
}
```



Demo Code Views – ComparisonView

```swift
import SwiftUI

struct ImageComparisonView: View {
    var body: some View {
        HStack(spacing: 20) {
            // Fixed-size image
            VStack(spacing: 8) {
                Image("swift")
                    .border(.red)
                    .frame(width: 80, height: 80)
                    .border(.blue, width: 2)
                    .background(.yellow.opacity(0.3))

                Text("Fixed Size")
                    .font(.caption)
            }
            .border(.green, width: 2)

            // Resizable image
            VStack(spacing: 8) {
                Image("swift")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 120, height: 120)
                    .border(.blue, width: 2)
                    .background(.yellow.opacity(0.3))

                Text("Resizable + Fit")
                    .font(.caption)
            }
            .border(.green, width: 2)
        }
        .frame(maxWidth: .infinity, maxHeight:
            .infinity)
        .background(.gray.opacity(0.1))
```



SwiftUI App Lifecycle States

- **Active** → App in foreground, fully interactive (normal use).
- **Inactive** → App visible, but not responding (e.g., call, notification).
- **Background** → App not visible, still runs background tasks (e.g., after switching apps).

🛠️ Managing App Lifecycle in SwiftUI

- Use @Environment(\.scenePhase) to detect state changes
- Attach .onChange to a **top-level container view** (not each subview)
- Why?
- 🌍 Scene phase = global, applies to the whole app
- 📌 Keep **one source of truth**
- 🚫 Avoid duplicate triggers from multiple views

App Lifecycle & ScenePhase

🔄 State	Description	Example
.active	**Foreground** & interactive.	User is using your app.
.inactive	**Foreground** but *not receiving events*	An incoming call covers the screen.
.background	**Not visible**, still runs background tasks	User switched to another app.

🛠️ **Watching State Changes**

SwiftUI provides a property wrapper to observe lifecycle changes:

```
@Environment(\.scenePhase) var scenePhase
```

```
// The system automatically updates this value
@Environment(\.scenePhase) var scenePhase
```

Manage State Changes

- Place a .onChange modifier on your **root view** (e.g., ContentView) to manage your UI's state across lifecycle changes.
- **This is the most common and correct place for it.**

```
import SwiftUI

struct ContentView: View {
    @Environment(\.scenePhase) var scenePhase
    var body: some View {
        NavigationStack { ••• } // All UI elemnets contained in this
            stack
        .onChange(of: scenePhase) { oldValue, newValue in
            switch newValue {
            case .active:
                print("✅ App is active, change from \(oldValue) to
                    \(newValue)")
            case .inactive:
                print("⏸️ App is inactive, change from \(oldValue) to
                    \(newValue)")
            case .background:
                print("🌙App isin background, change from \(oldValue)
                    to \(newValue)")
            default:
                break
            }

        }
    }
}
```

SwiftUI - Roadmap