Introduction to SwiftUI

Amit Kumar Follow
Jun 2 · 3 min read

In WWDC 2019 apple announced a completely new user interface development framework called SwiftUI. SwiftUI is a user interface toolkit that lets us design apps in a declarative way. There was lots of talk about whether we should use storyboard or build UI programmatically. SwiftUI is the answer of all these Questions. You can now develop the app's UI with a declarative Swift syntax.

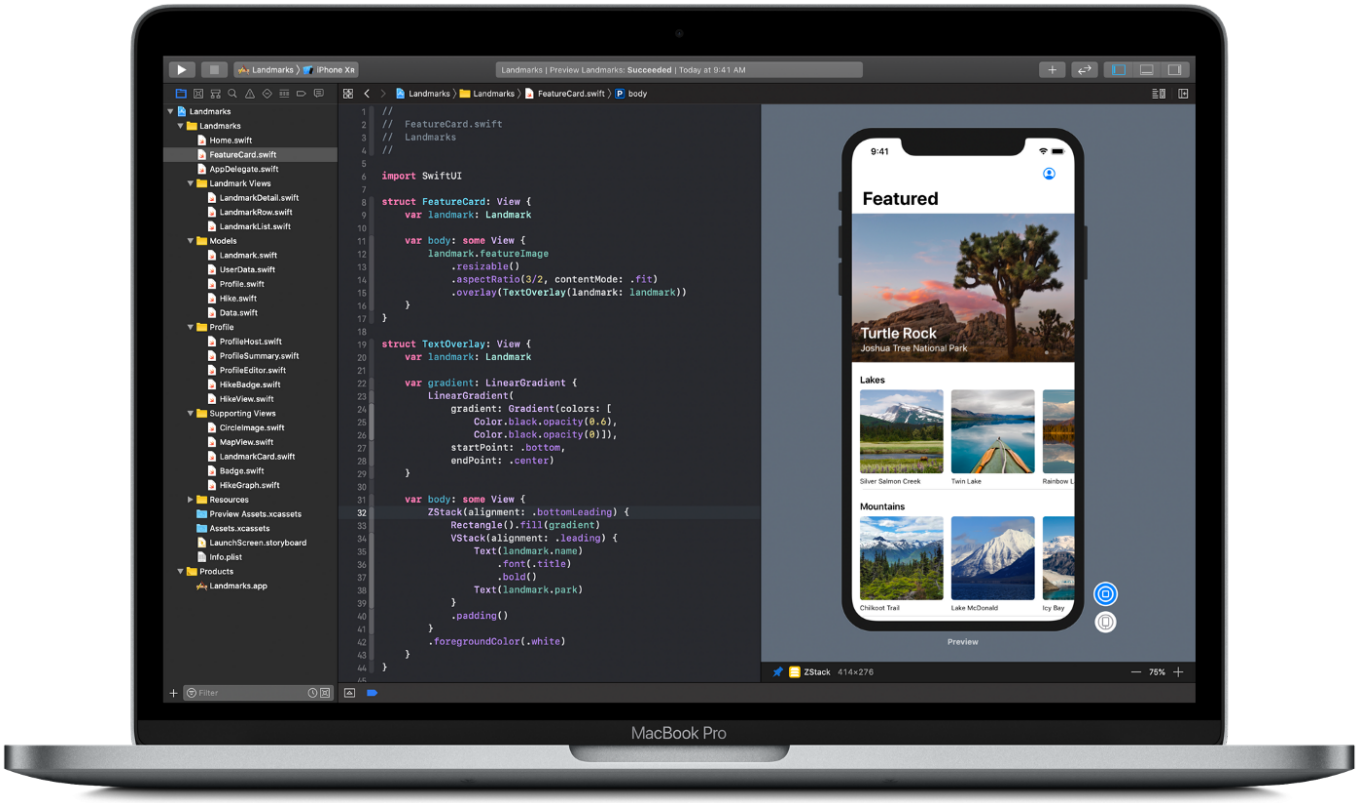
. . .

Note:

> *SwiftUI is an innovative, exceptionally simple way to build user interfaces across all Apple platforms with the power of Swift. Build user interfaces for any*

> *Apple device using just one set of tools and APIs — Apple*

. . .



. . .

Declarative vs Imperative Programming :

> ***Declarative programming*** *is a programming paradigm — a style of building the structure and elements of computer programs — that expresses the logic of a computation without describing its control flow.*
>
> ***Imperative programming*** *is a programming paradigm that uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform.*

1. In declarative programming developers describe how the UI looks like and what you want to do when a state changes. Imperative programming requires developers to write detailed instructions to layout the UI and control its states.
2. The declarative programming looks easier to read and understand. Most importantly, the SwiftUI framework allows you to write way less code to create a user interface. Imperative programming require lot of code and a bit complex to understand.
3. Declarative Programming means we are not going to show and hide a swiftUI component, we're just telling it all the rules we want it to follow

and leaving SwiftUI to make sure those rules are enforced. But in imperative programming we need to handle the component state to show and hide.

. . .

No more storyboard, Interface Builder and Auto-layout :

1. Earlier we used to develop user interface with storyboard, xib's and auto layout. But with SwiftUI, storyboard and auto layout are completely out of the picture. Now code editor is available with preview canvas .
2. Auto layout has always been a challenge in iOS user interface development. In SwiftUI, no need to define a layout constraint and resolve the conflicts. Now we can develop the user interface by using spacers, padding and stacks.
3. In SwiftUi, write the code in code editor and Xcode renders the user interface and display it in the canvas.

. . .

Note:

> SwiftUI is designed to work with the existing frameworks like UIKit for iOS. UIKit and SwiftUI can be used in a same project same as swift and objective-C. Apple provides several representable protocols for you to adopt in order to wrap a view or controller into SwiftUI.

. . .

Migrating from UIKit to SwiftUI :

If you've used UIKit before then you need to understand what's UIKit component equivalent in SwfitUI. Many of the classes you know just by removing prefix UI but few has renamed. Lets have a look at list.

- **`UIViewController : View`**
- **`UITableViewController : List`**
- **`UITabbarController : TabView`**
- **`UINavigationController : NavigationView`**
- **`UIAlertController : Alert`**

- `UIAlertController` with style `.actionSheet` : `ActionSheet`
- `UILabel` : `Text`
- `UITextField` : `TextField`
- `UITextField` with secure entry: `SecureField`
- `UISwitch` : `Toggle`
- `UISlider` : `Slider`
- `UIButton` : `Button`
- `UIStackView` with horizontal axis: `HStack`
- `UIStackView` with vertical axis: `VStack`
- `UIImageView` : `Image`
- `UISegmentedControl` : `SegmentedControl`
- `UIStepper` : `Stepper`
- `UIDatePicker` : `DatePicker`
- `UIPickerView` : `Picker`
- `UITabbar` : `TabView`

- `UINavigationBar : NavigationView`
- `UIScrollView : ScrollView`

There are many other components in SwiftUI that developers need to know about them. some of them are ZStack, Padding, Spacer, contentView, UIHostingController, *UIViewRepresentable, Binding, Form etc.*

. . .

Note:

> *There are some important properties in SwiftUI that developers need to know about them such as @State, @Bind, @EnvironmentalObject.*

. . .

The Combine Framework :

UIKit use view controller as a central block for communication between view and the model. But in SwiftUI this communication and data sharing

between views are now done via new framework called combine. This new approach completely replaces the role of view controller in UIKit.

. . .

Conclusion:

SwiftUI is new to everyone and might take time to get stable as a framework. But SwiftUI is the future of iOS development. There are lot of things to learn in SwiftUI.

I hope this article helps you to understand the basics of SwiftUI. We will explore more about it in next articles.

Follow me on medium.

Thank you for reading. Cheers :) 🍻 🥂

Swiftui Swift iOS iOS App Development Xcode11