How To Build a Single View App with Swift and Xcode

**Author:** Christine Yang

**Description:** Step-by-step tutorial about how to build a single view app with SwiftUI.

**Minutes to Complete: 75**

**Prerequisites:** MacOS

**Links:**

[Swift]: https://www.codecademy.com/resources/blog/what-is-swift-used-for/

[Apple Developer Downloads]: https://developer.apple.com/download/all/

[App Store]: https://apps.apple.com/us/app/xcode/id497799835?mt=12

[example bundle identifiers]: https://support.apple.com/guide/mdm/bundle-ids-for-native-ios-and-ipados-apps-mdm90f60c1ce/web

[State]: https://developer.apple.com/documentation/swiftui/state

[string interpolation]: https://www.codecademy.com/resources/docs/emojicode/interpolation

[Apple Developer sign-in page]: apple\_dev\_sign\_in.png

[Apple Developer’s more downloads page search for xcode version 12]: apple\_dev\_download.png

[Xcode version 12.5.1 downloadable 10.98GB .xip file]: xcode\_12\_download\_xip.png

[Welcome page for creating a new project with Xcode]: create\_new\_xcode\_project.png

[Different template options for a new Xcode project including platform options such as iOS, macOS, watchOS, tvOS, and others, application options such as an App, a Game, and a Messaging Apps]: new\_project\_template.png

[Last step to create a new Xcode project; filled in form with the desired application’s product name, organization identifier, choice of interface, choice of life cycle, language choice, and optional testing suite]: configure\_new\_project.png

[Project Navigator display of Swift template files]: project\_navigator.png

[Three panels from left to right: the ContentView.swift file is selected in the Project Navigator panel, followed by the code file itself, and the iPhone simulator]: hello\_world\_app.png

[Top toolbar button to change iOS device displayed in the simulator/visualizer]: simulator\_drop\_down.png

[Drop Down of different iOS device’s to choose from]: drop\_down.png

[Image assets for all six faces of the die; three varying image sizes per side of the die]: dice\_image\_sets.png

[Left to right; inside the assets folder with dice1 image set selected, and the images stored based on size]: assets\_stores\_dice\_images.png

[Adding another component under body creates a second view]: two\_components\_second\_view.png

[Embedding multiple components in a vertical stack allows for a single view]: vstack\_single\_view.png

[Enlarged visualizer with a styled "Roll Me!" text and unstyled dice image]: add\_text\_style.png

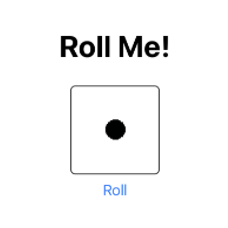
[Enlarged visualizer with a styled "Roll Me!" text, unstyled dice image, and a button]: add\_button.png

[Enlarged visualizer with a "Roll Me!" text, one side of dice image, and a "Roll" button]: dice\_roll\_final\_result.png

### Introduction

In this step-by-step tutorial, we will be using Xcode 12 which includes [Swift] version 5.3 and the SwiftUI framework. Xcode is an IDE used to develop software for Apple platforms and Swift is a programming language for mobile development.

We will go over the basic steps to create a single view Hello World app in Xcode for SwiftUI development. We will also build a simple Dice Roll Project.

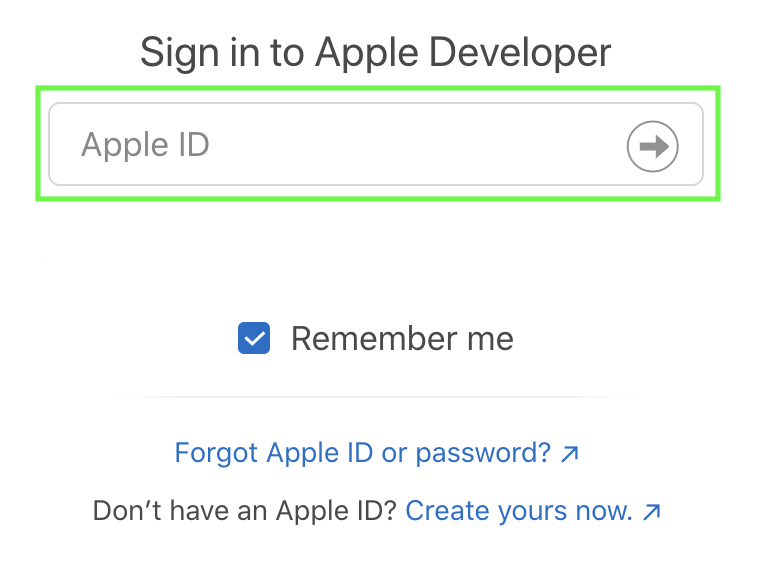


![Enlarged visualizer with a styled "Roll Me!" text, one side of dice image, and a "Roll" button]

#### Install Xcode

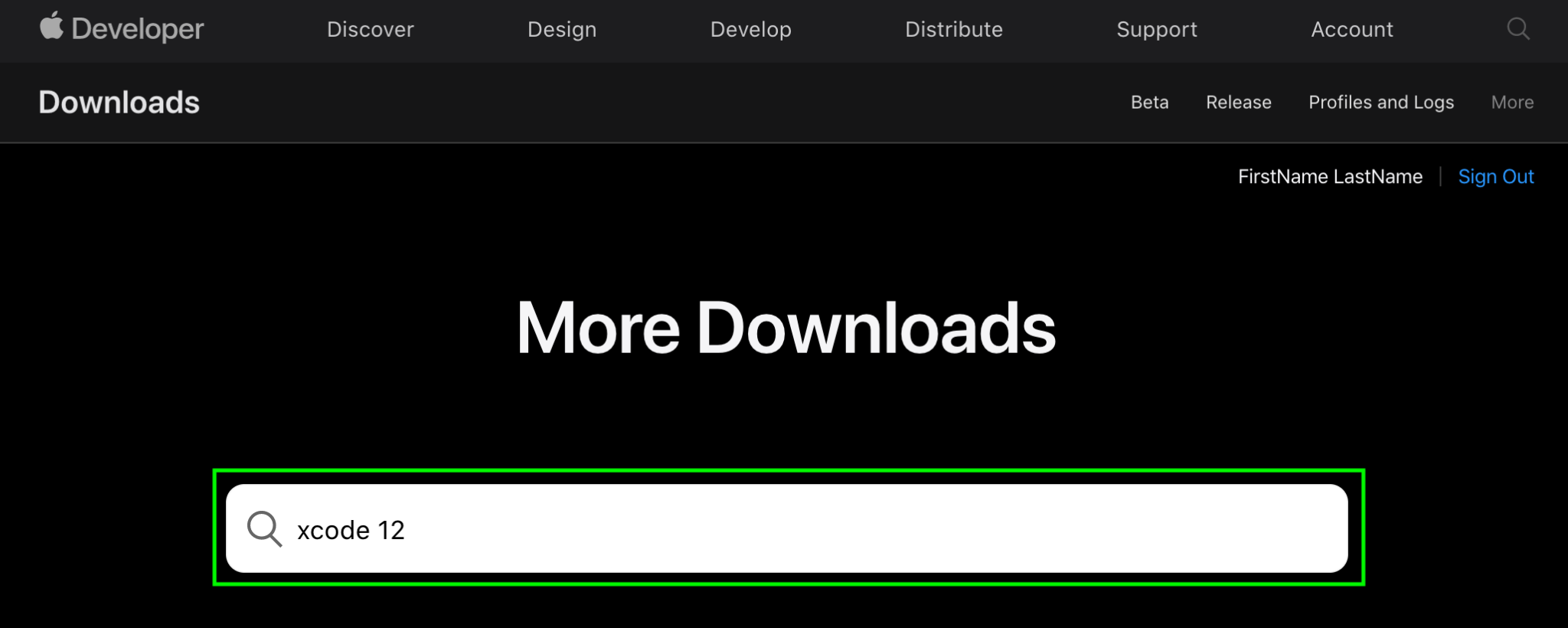
We can install the latest stable build of Xcode directly from the [App Store], or through [Apple Developer Downloads]. To install Xcode through Apple Developer, follow these steps:

1. Sign in to Apple Developer with a valid AppleID and password.

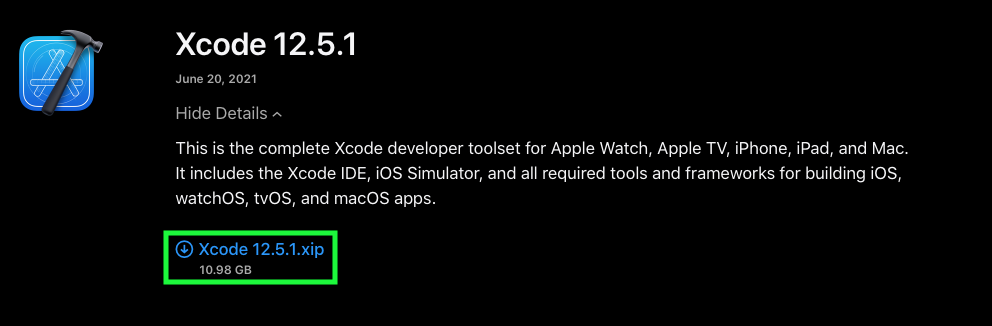


![Apple Developer sign-in page]

2. Under "More Downloads", search for Xcode 12. Download the \*\*.xip\*\* file (this process may take some time).



![Apple Developer’s more downloads page search for xcode version 12]



![Xcode version 12.5.1 downloadable 10.98GB .xip file]

3. Once downloaded, drag the app into your \*\*Applications\*\* folder.

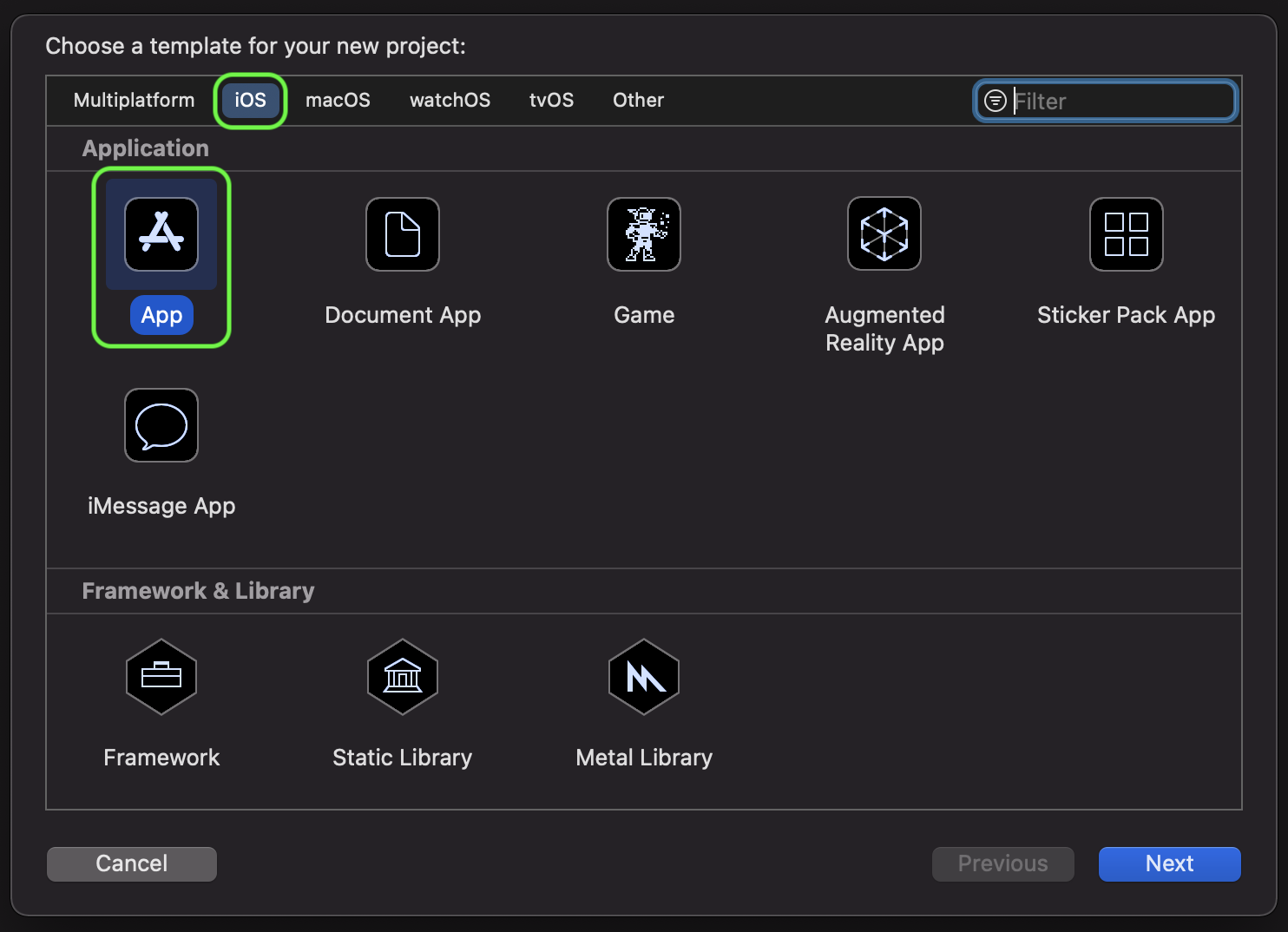
### Create a New Project

1. After successfully installing Xcode, we will run the application and select "Create a new Xcode project".



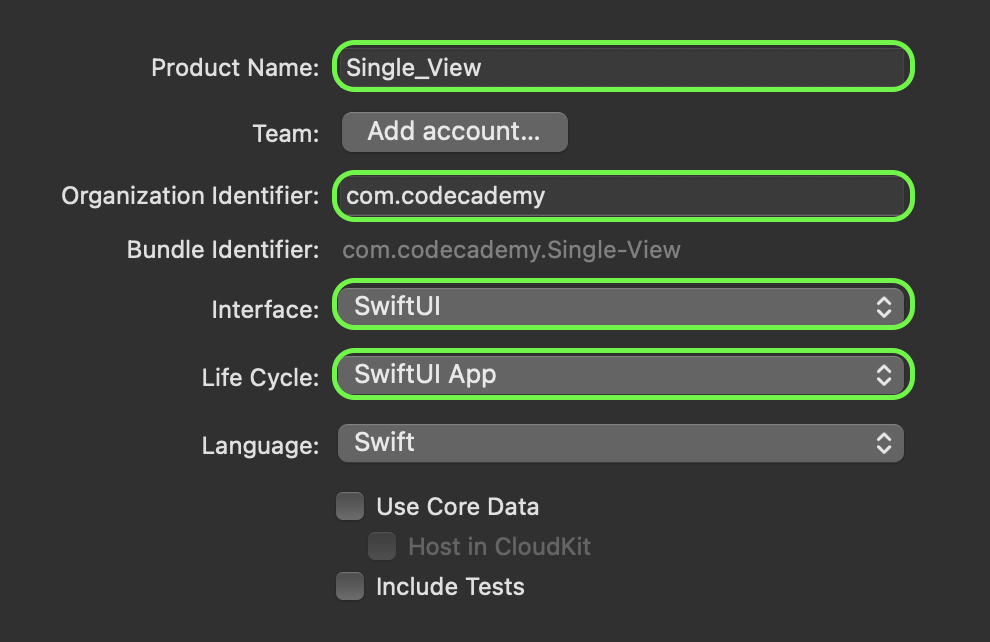
![Welcome page for creating a new project with Xcode]

2. Choose the iOS platform and App application type.



![Different template options for a new Xcode project including platform options such as iOS, macOS, watchOS, tvOS, and others, application options such as an App, a Game, and a Messaging App]

3. Let’s enter a project name and organization identifier. This will generate a bundle identifier unique to the application; here are some [example bundle identifiers].



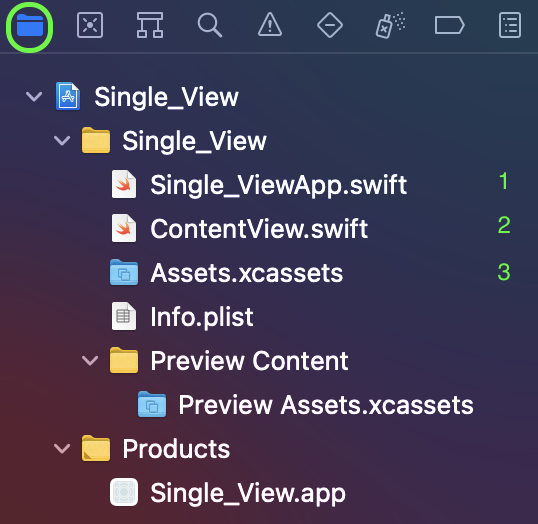
![Last step to create a new Xcode project; filled in form with the desired application’s product name, organization identifier, choice of interface, choice of life cycle, language choice, and optional testing suite]

4. We will select SwiftUI interface and SwiftUI App life cycle and click Next.

5. Finally, we will save the project into our desired directory.

#### Project Navigator

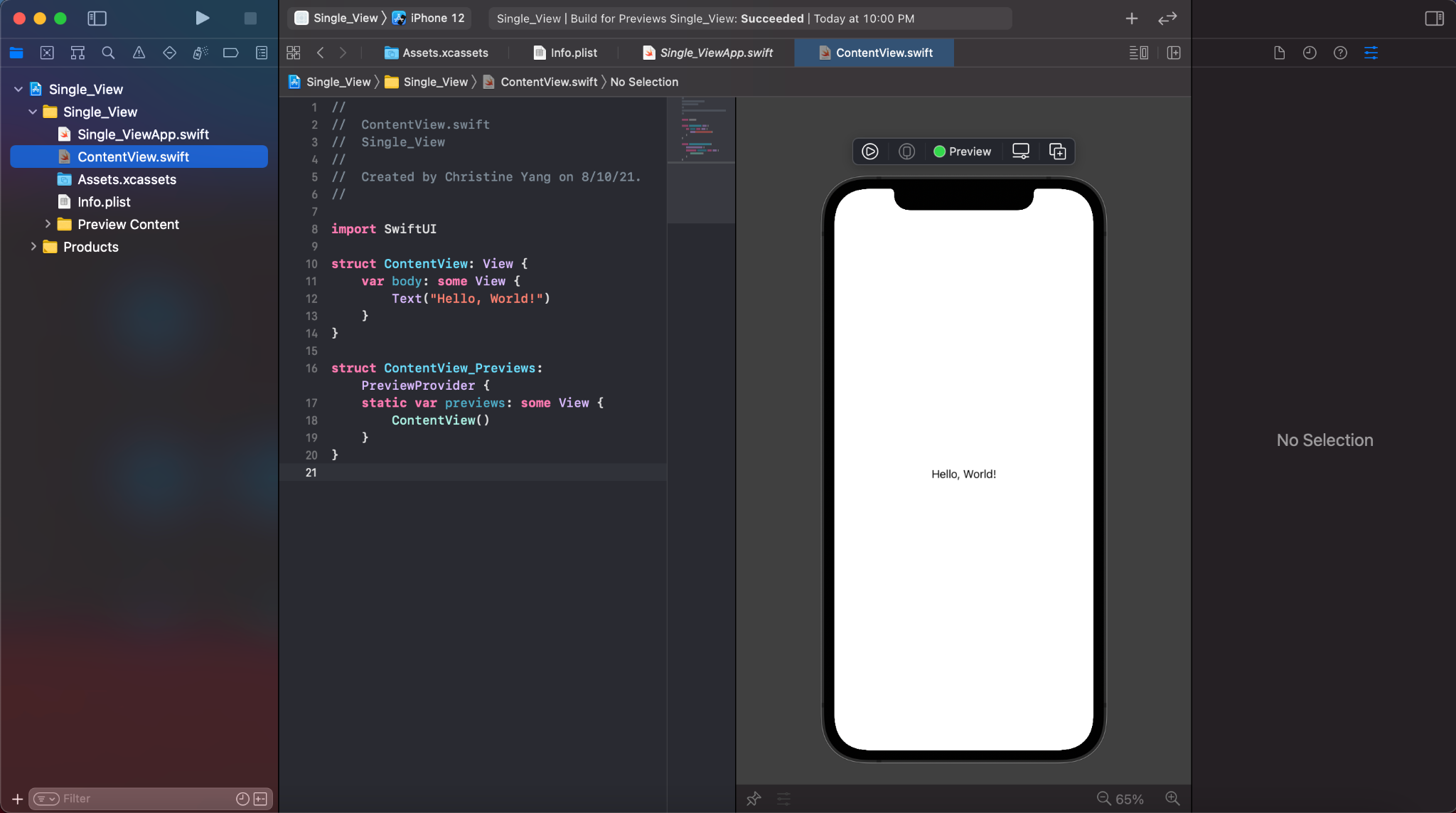
Let’s take a look at the boilerplate code that was generated after we created the new project. Our file tree is located on the left of the IDE. The Project Navigator should include as follows:



![Project Navigator display of Swift template files]

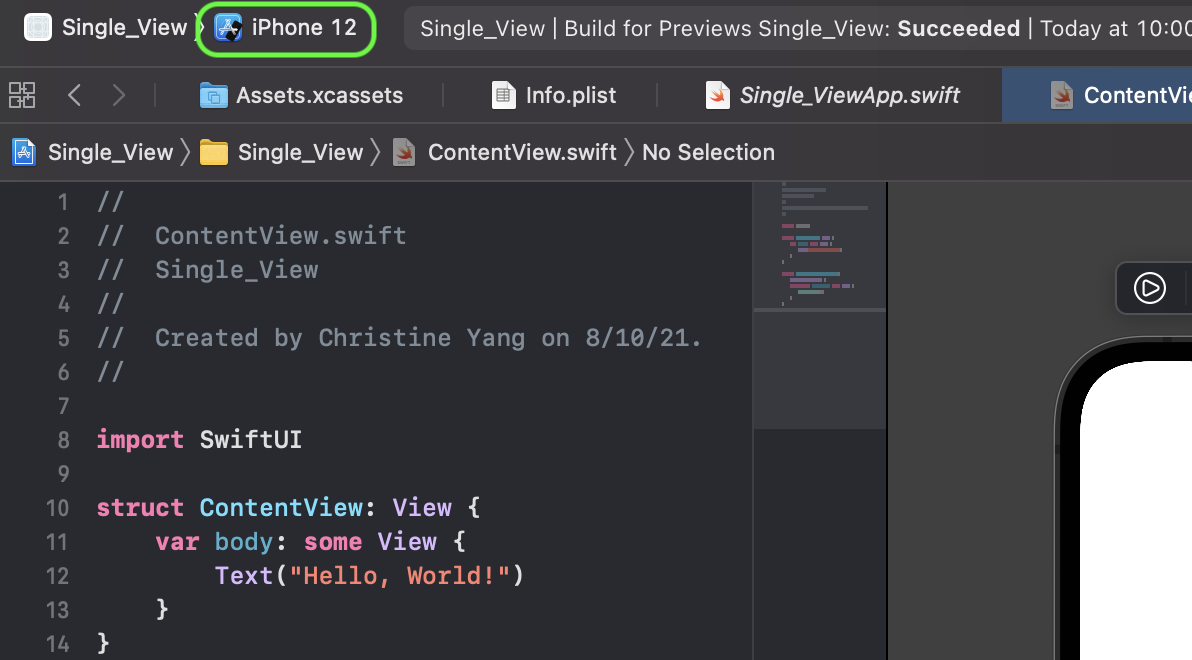
1. \*\*SingleViewApp.swift\*\* is the entry point file; this file will be processed first when the app is run.

2. \*\*ContentView.swift\*\* is the first user interface view. Notice the preview component below or beside the code file. The preview component simulates different iOS versions and allows us to simultaneously visualize changes in our code.

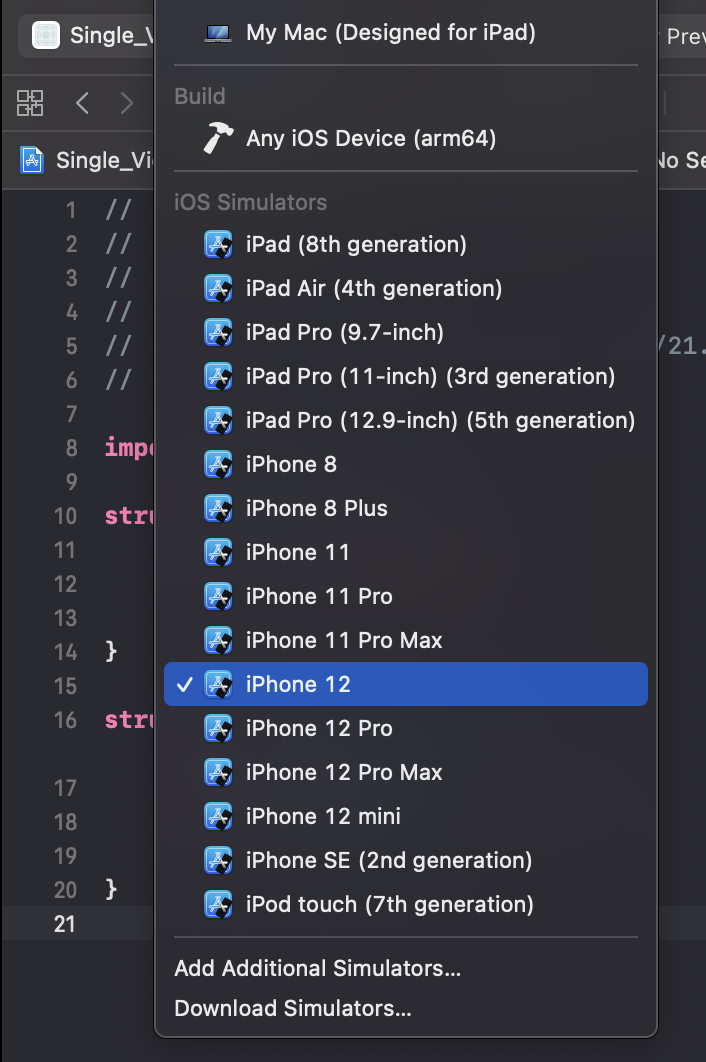


![Three panels from left to right: the ContentView.swift file is selected in the Project Navigator panel, followed by the code file itself, and the iPhone simulator]

To switch between different iOS versions, we will locate the simulator dropdown above the file tabs, and choose our preferred version.



![Top toolbar button to change iOS device displayed in the simulator/visualizer]



![Drop Down of different iOS device’s to choose from]

3. The \*\*Assets.xcassets\*\* folder is where we can add static images for building out our application.

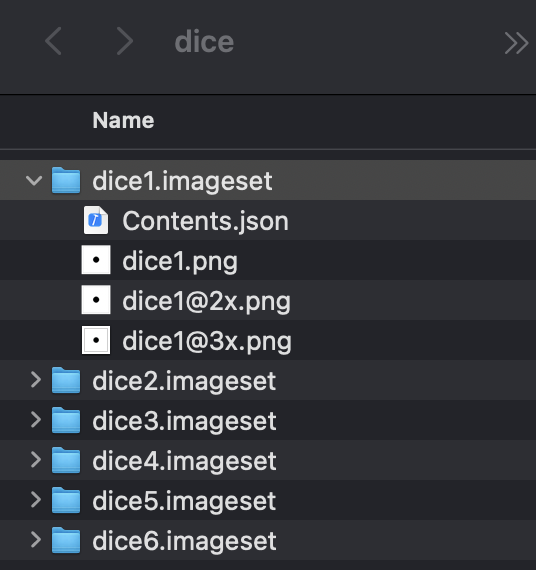
Our Hello World App is complete.

### Build a Simple Dice Roll Project with SwiftUI

Now that we have a general overview of Xcode and SwiftUI, let’s build a Simple Dice Roll Project. The idea is to create a button that will randomly roll a single die.

#### Adding Image Assets

In a folder, we have created some image sets.



![Image assets for all six faces of the die; three varying image sizes per side of the die]

Each image set folder contains three images of varying size. The correct image size will be displayed on the user's device depending on screen size. This is configured through the \*\*Contents.json\*\* file:

```swift

// dice1.imageset/Contents.json

// Objects are enclosed in curly brackets

{

// An object with a `images` key points to an array of three objects

"images" : [ // arrays are enclosed in square brackets

{

"filename" : "dice1.png", // corresponding image file

"idiom" : "iphone", // type of device

"scale" : "1x"

},

{

"filename" : "dice1@2x.png",

"idiom" : "iphone",

"scale" : "2x"

},

{

"filename" : "dice1@3x.png",

"idiom" : "iphone",

"scale" : "3x"

}

],

"info" : {

"author" : "xcode",

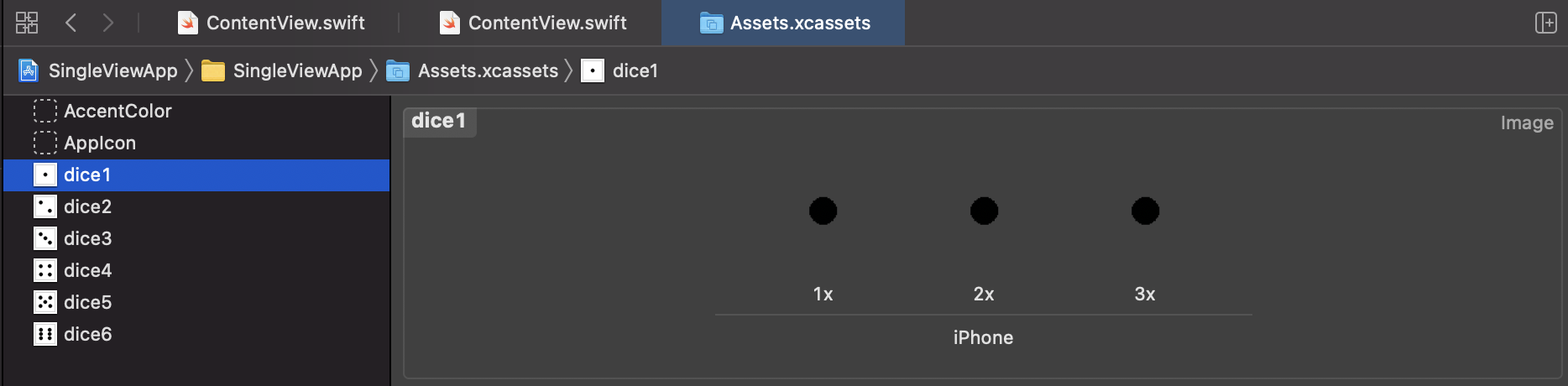
"version" : 1

}

}

```

Upon dragging the six folders into our assets folder, Xcode is able to identify each image accordingly.



![Left to right; inside the assets folder with dice1 image set selected, and the images stored based on size]

#### Build the UI

Before we begin building out the view, let’s take a closer look at the template code within \*\*ContentView.swift\*\*.

```swift

import SwiftUI

struct ContentView: View {

var body: some View {

Text("Hello World!")

}

}

struct ContentView\_Previews: PreviewProvider {

static var previews: some View {

Group {

ContentView()

}

}

}

```

To use the `SwiftUI` framework, we import it at the top of our file. The preview component is rendered via `struct ContentView\_Previews`.

`struct ContentView` contains the single view displayed in the preview component. If we change the `Hello, World!` string within our text component to `Roll Me!`, the view will change to display the new text. The preview component tends to stop when large changes in the code occur or when no activity is detected in the view. If this happens, we can either hit the resume, or play button.

If we try to add an `Image` component to our view, a second view appears in our preview component.

```swift

struct ContentView: View {

Var body: some View {

Text("Roll Me!")

Image("dice1")

}

}

```



![Adding another component under the body creates a second view]

We typically want to return one view with each `struct` data type and can do so by embedding the text components in a `VStack`. This will arrange multiple components vertically. <kbd>Command</kbd> and click the `Text` component and select `Embed in VStack`.

```swift

struct ContentView: View {

Var body: some View {

VStack {

Text("Roll Me!")

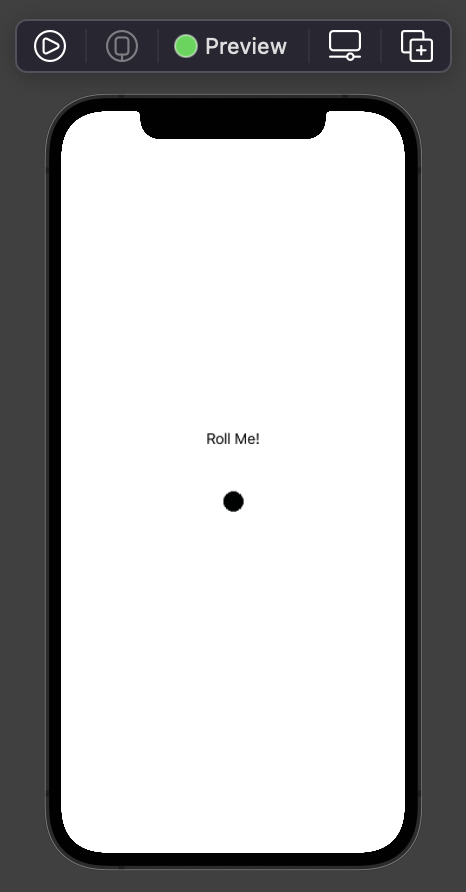
Image("dice1")

}

}

}

```



![Embedding multiple components in a vertical stack allows for a single view]

To add some simple styling to a component, we can chain `.method()` functions. Let’s make the text bold and give it a large title font.

```swift

VStack {

Text("Roll Me!")

.font(.largeTitle)

.bold()

```



![Enlarged visualizer with a styled "Roll Me!" text and unstyled dice image]

#### Build the View

Currently, our view consists of a text and image element. What we want is an image of the die and a button that will randomly roll it. In order to create our die, we’ll walk through the following steps:

\* On the line following `struct ContentView`, we'll create a variable `randomNum` and set its initial value to 1. We need the property wrapper, [State], to change the variable’s value when the button component is clicked.

```swift

struct ContentView: View {

@State var randomNum = 1

var body: some View {

```

\* Next, we'll create the button underneath the image component, and give it a label.

```swift

VStack {

Text("Roll Me!")

.font(.largeTitle)

.bold()

Image("dice1")

Button("Label Name",

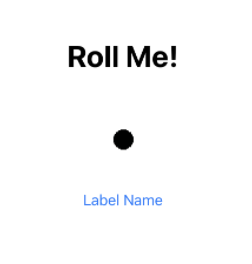
action: {

print("This line of code will run when the button is clicked")

})

}

```



![Enlarged visualizer with a styled "Roll Me!" text, unstyled dice image, and a button]

\* When the button is clicked, we will set `randomNum` to a random number between 1 and 6. We'll also utilize [string interpolation] to display the corresponding dice image and rename our button.

```swift

Image("dice\(randomNum)")

Button("Roll",

action: {

randomNum = Int.random(in: 1...6)

})

}

```

\* Finally, we can give our image a border and add any desired styling.

```swift

import SwiftUI

struct ContentView: View {

@State var randomNum = 1

var body: some View {

VStack {

Text("Roll Me!")

Image("dice\(randomNum)")

.overlay(RoundedRectangle(cornerRadius: 5.0).stroke(lineWidth: 1.0))

Button("Roll",

action: {

randomNum = Int.random(in: 1...6)

})

}

}

}

struct ContentView\_Previews: PreviewProvider {

static var previews: some View {

Group {

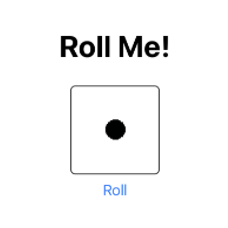
ContentView()

}

}

}

```



![Enlarged visualizer with a styled "Roll Me!" text, one side of dice image, and a "Roll" button]

### Conclusion

In summary, we created our first Xcode project and built a simple Single View Dice Roll app. We explored the project’s boilerplate code in the entry point file and content view file. As a bonus, we covered some basic view properties and components. Mobile Development is a highly demanded skill and hopefully, this article helps you get started.