iOS & iPadOS App Development Course - UI/UX Workbook

Arun Patwardhan

Copyright

About this Guide

This guide is meant to be used as a part of the iOS & iPadOS App Development Course. The guide is designed to be an independently used material too if needed.

202327611082322384

Disclaimer

The information Is provided “As Is”, without warranties of any kind, express or implied, including but not limited to the warranties Of merchantability, fitness for a particular purpose and non-infringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the information or the use or other dealings in the information.

20232761108232384

Terms and conditions

Amaranthine's training course materials, including the book, are Amaranthine's confidential and proprietary information.
Such materials may not be copied, reproduced, distributed, disclosed or published by Customer or utilised for any purpose other than as required during a training course. Customer may not alter or remove any proprietary notices or legends contained in or on any of the training course materials. Violation of this policy may result in immediate dismissal from a training course, and/or barred from future training courses, in addition to any other remedies Amaranthine may have at law or in equity.

About the author

I am an Apple Certified Master Trainer. I have been doing trainings on Apple since the past 8 years. My current areas of research include App Development for Apple's Operating Systems. Apart from training. I am also involved in course design & content development.

www.arunpatwardhan.com
http://www.linkedin.com/in/arunpatwardhan
www.amaranthine.in

202327611082322384

Table of Contents

20232761108232384

6
Chapter 6: Designing UI for a single device

In this chapter the reader will get familiar with the different UI components available to developers within the UIKit framework

Exercise 6.1: Getting familiar with storyboard

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn how to create a user interface using storyboard.

Tools Required

Xcode 14.x or later

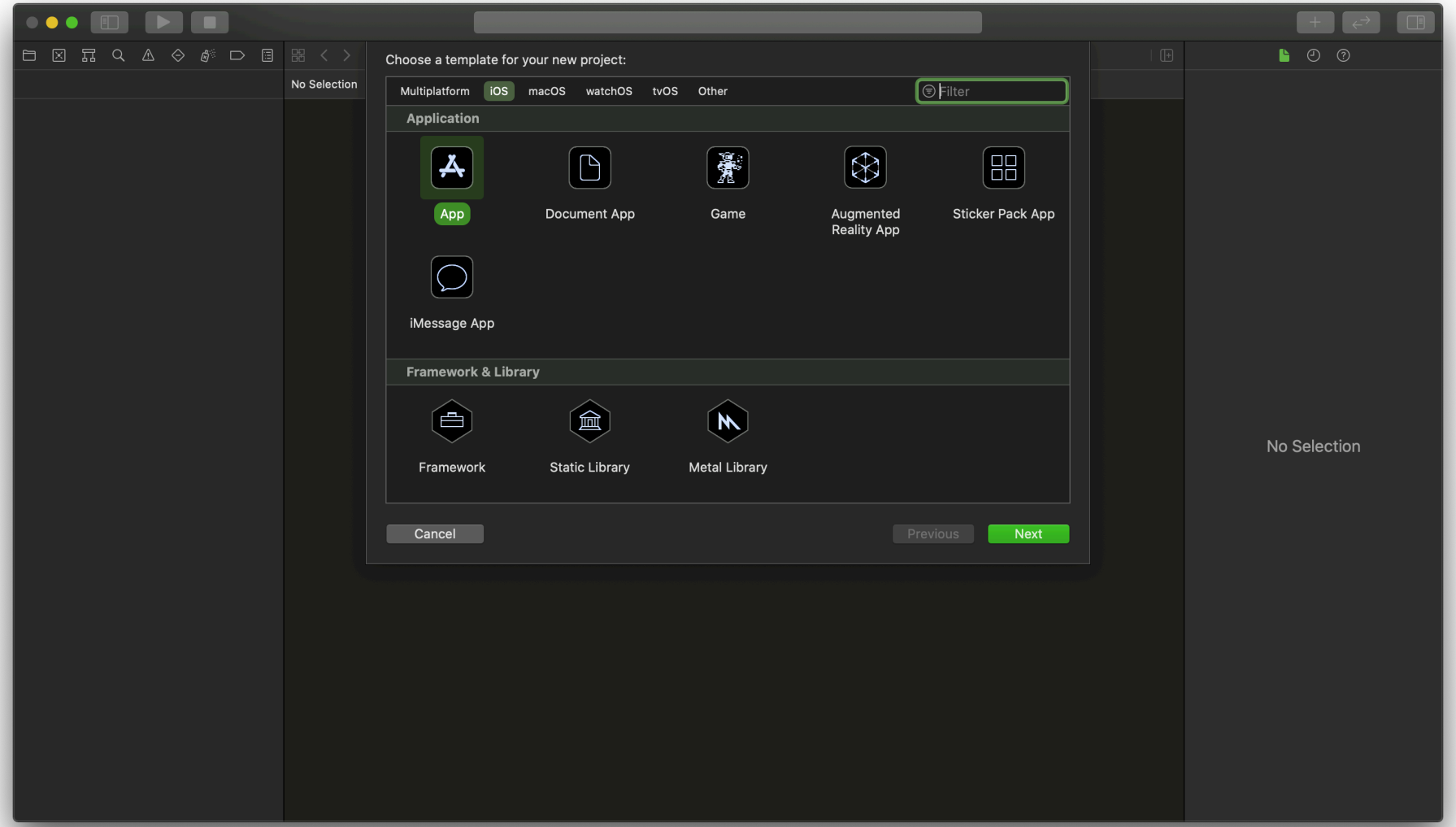
Outcome of the exercise

You should be comfortable with storyboards.

Steps

1. Open Xcode.
2. Create a new project by clicking *File* > *New* > *Project.*
3. From the template wizard select iOS..

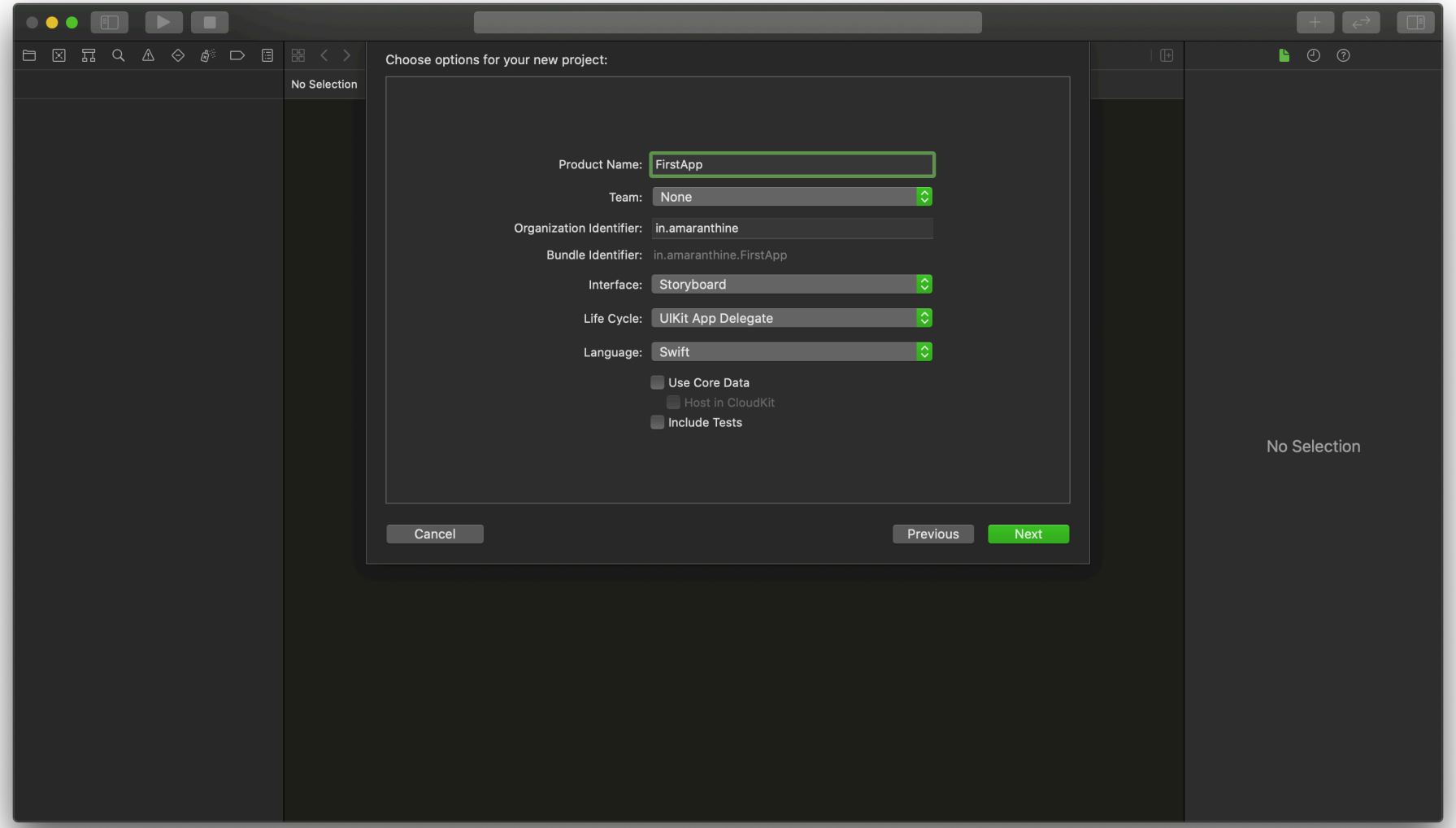
4. Select the option to create an app.



5. Click next.

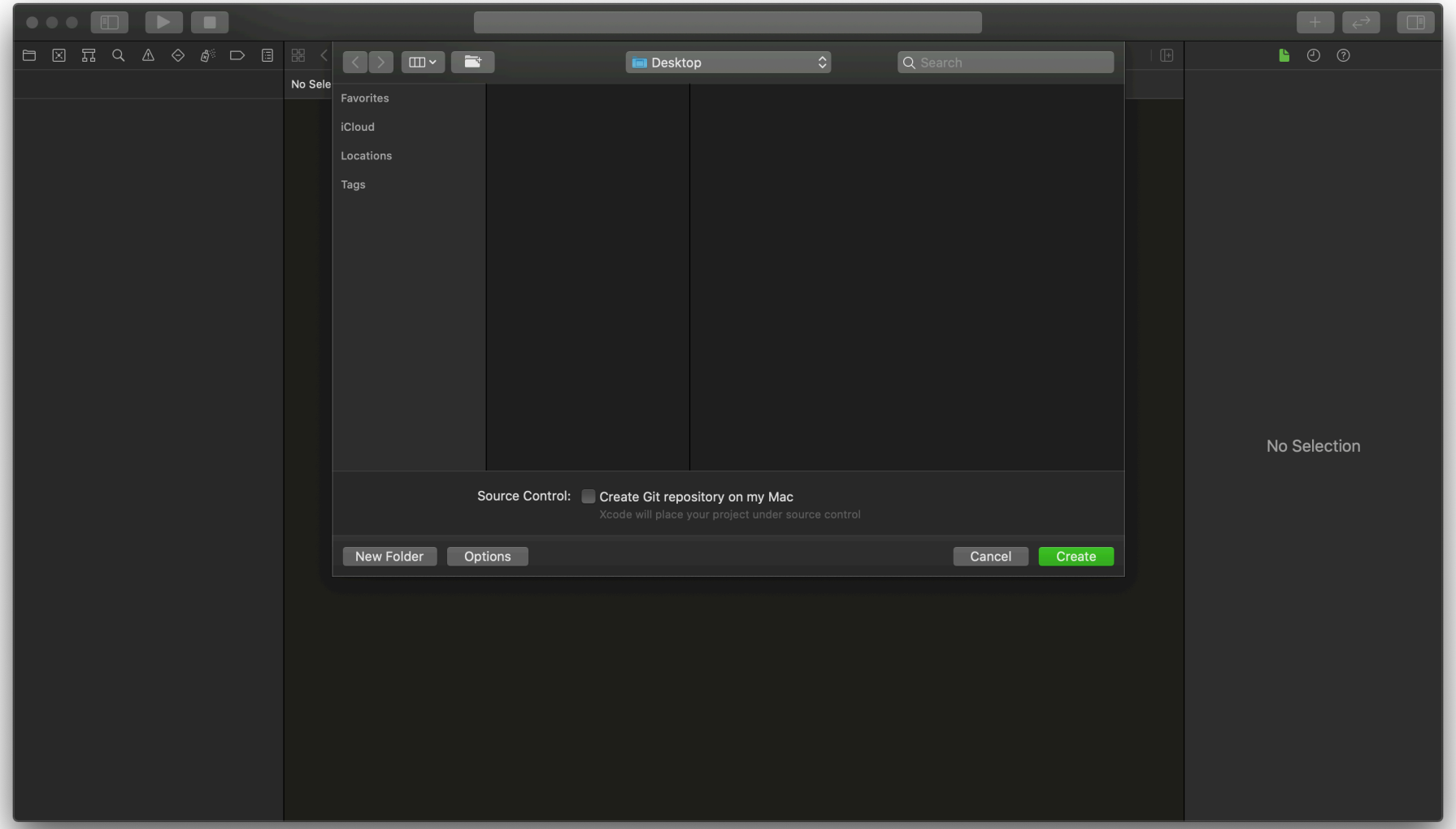
6. Provide the following details:
   - Name: FirstApp
   - Organisation ID: *<your company url in reverse format>*
   - Life cycle: *UIKIt App Delegate*
   - Interface: *Storyboard*
   - Language: *Swift*

Leave all the checkboxes unchecked.



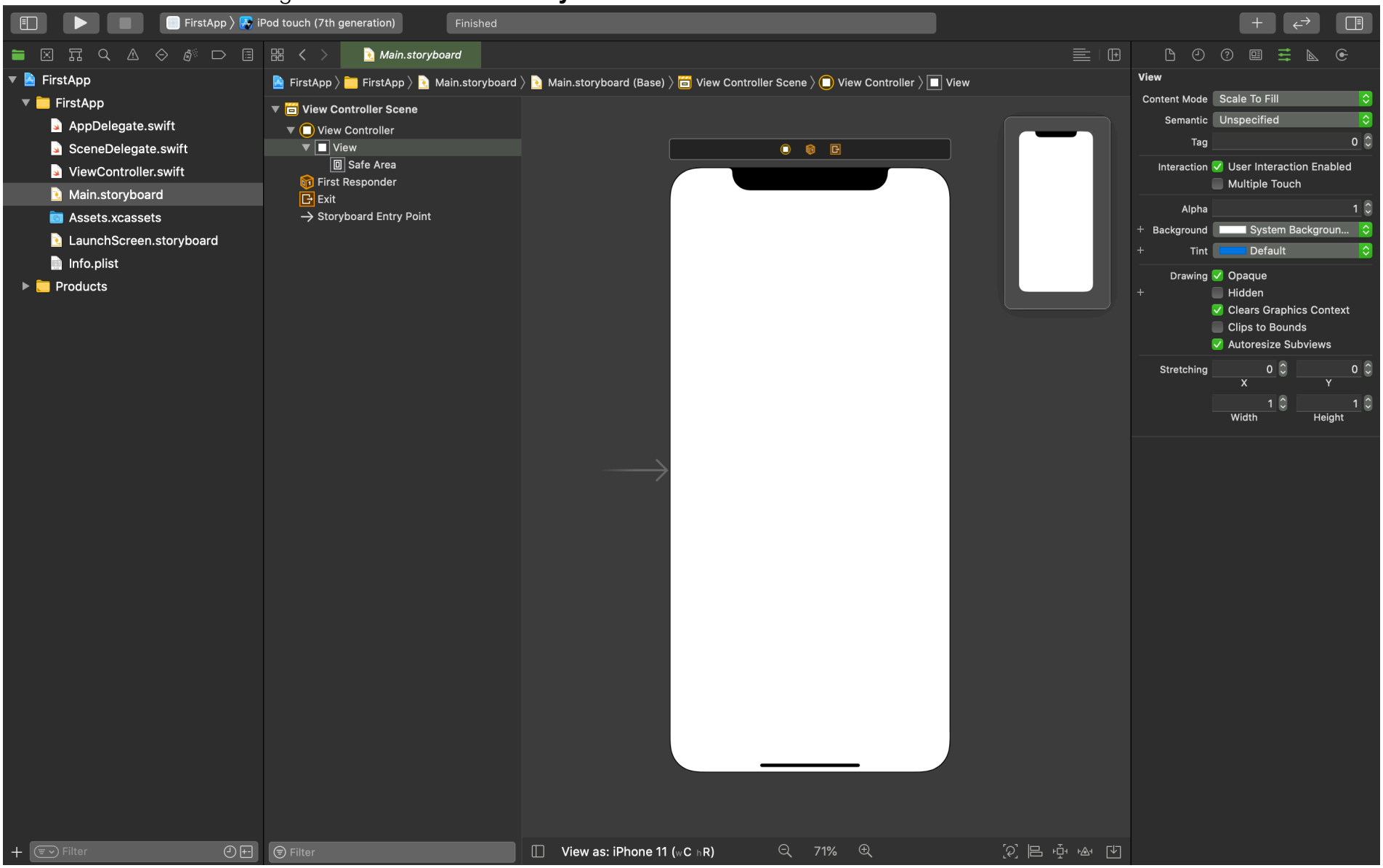
7. Click Next.

8. Choose where you wish to save the project.

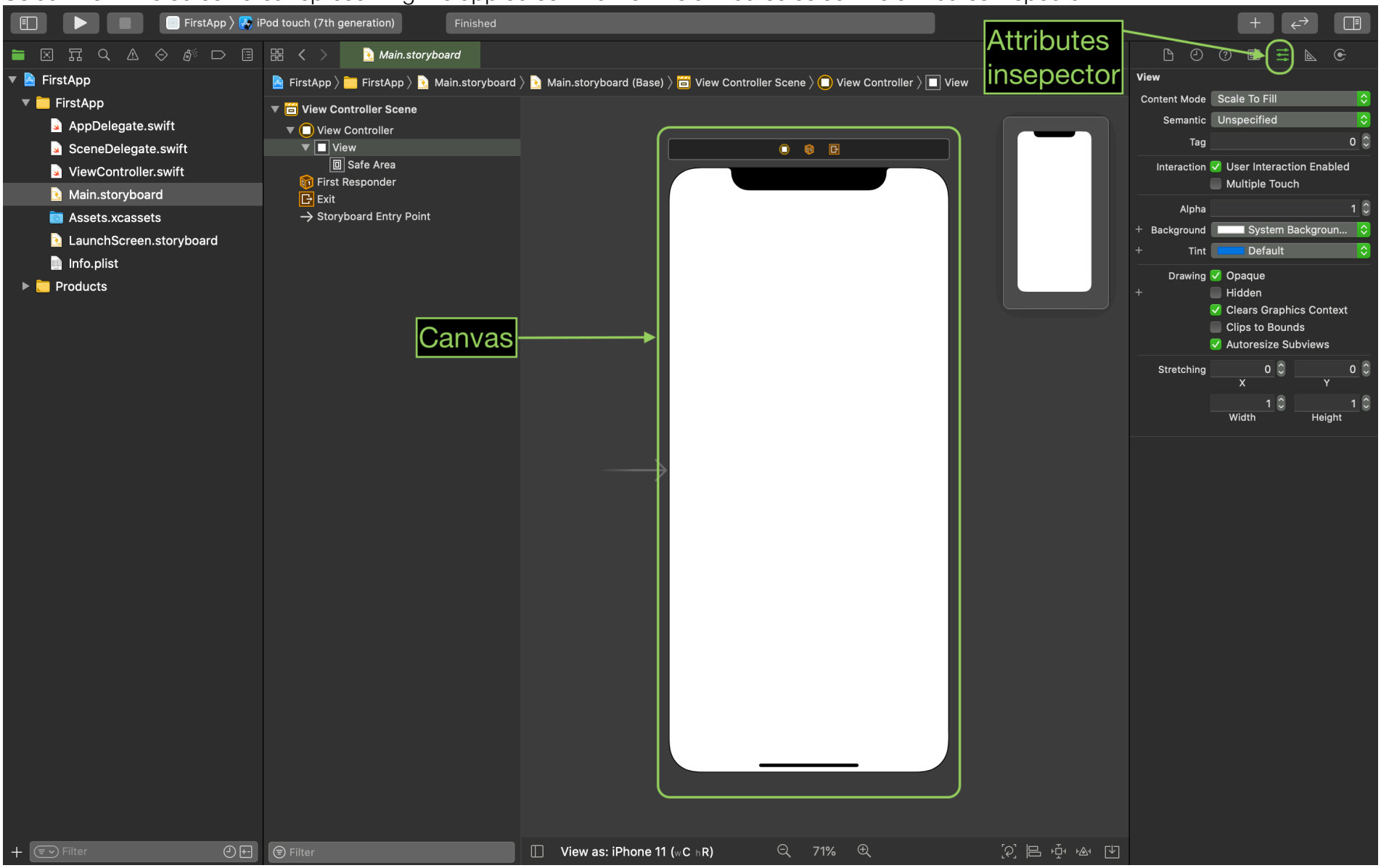


9. Click create.

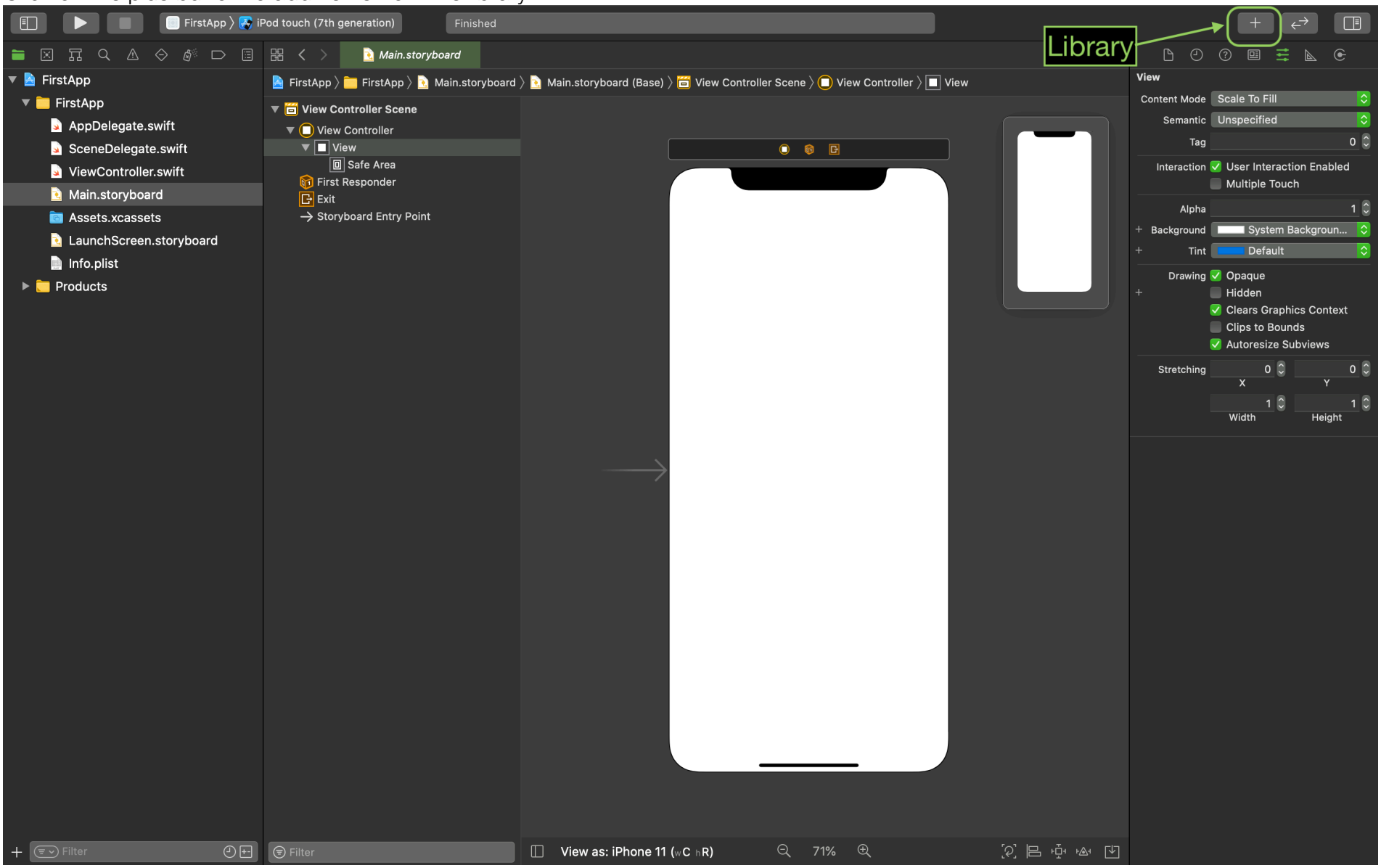
10. From the left hand side navigation bar select **Main.Storyboard**.

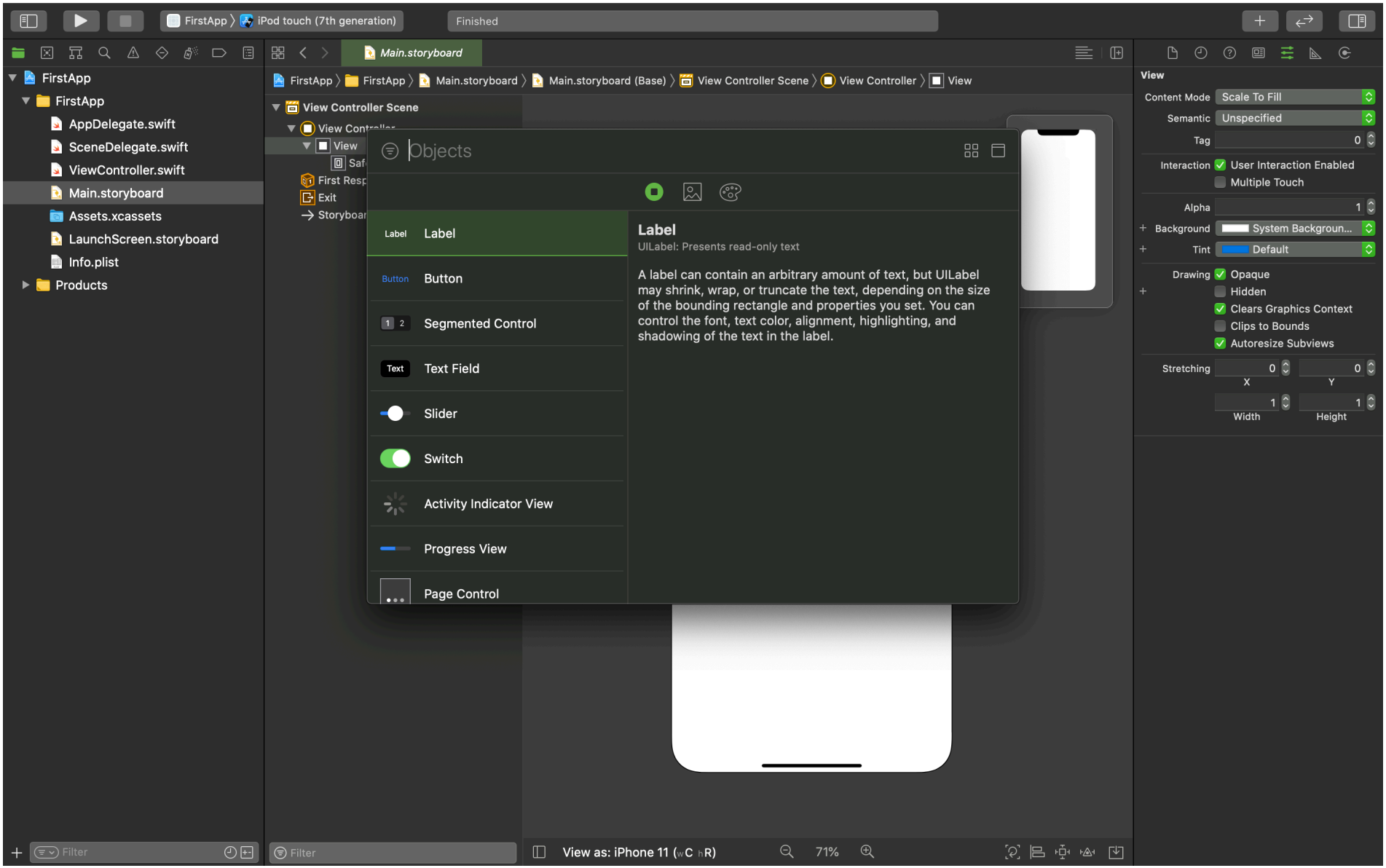


11. Select the white screen area representing the app screen. To view its attributes select the attributes inspector.

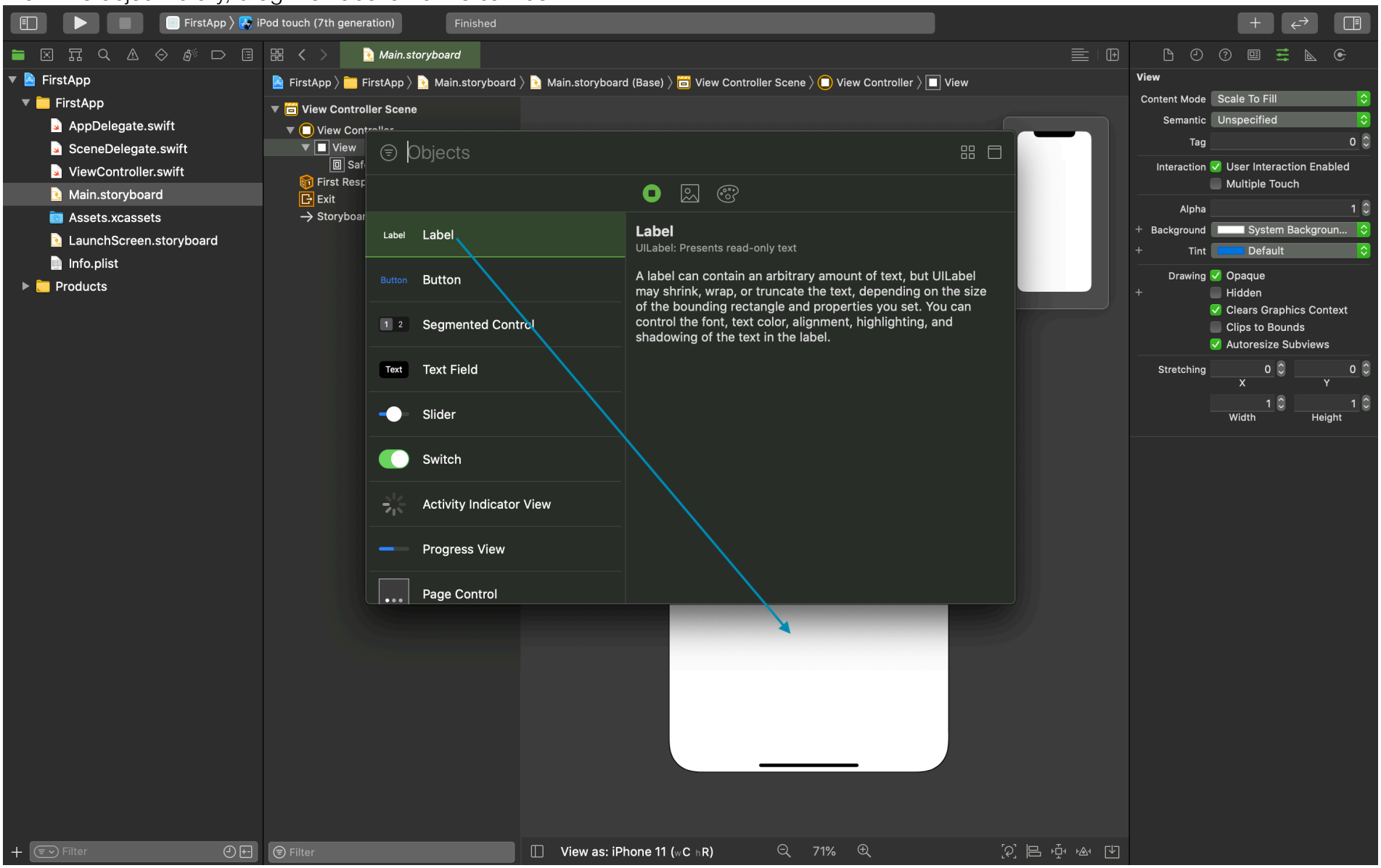


12. Click on the plus button to add items from the library.

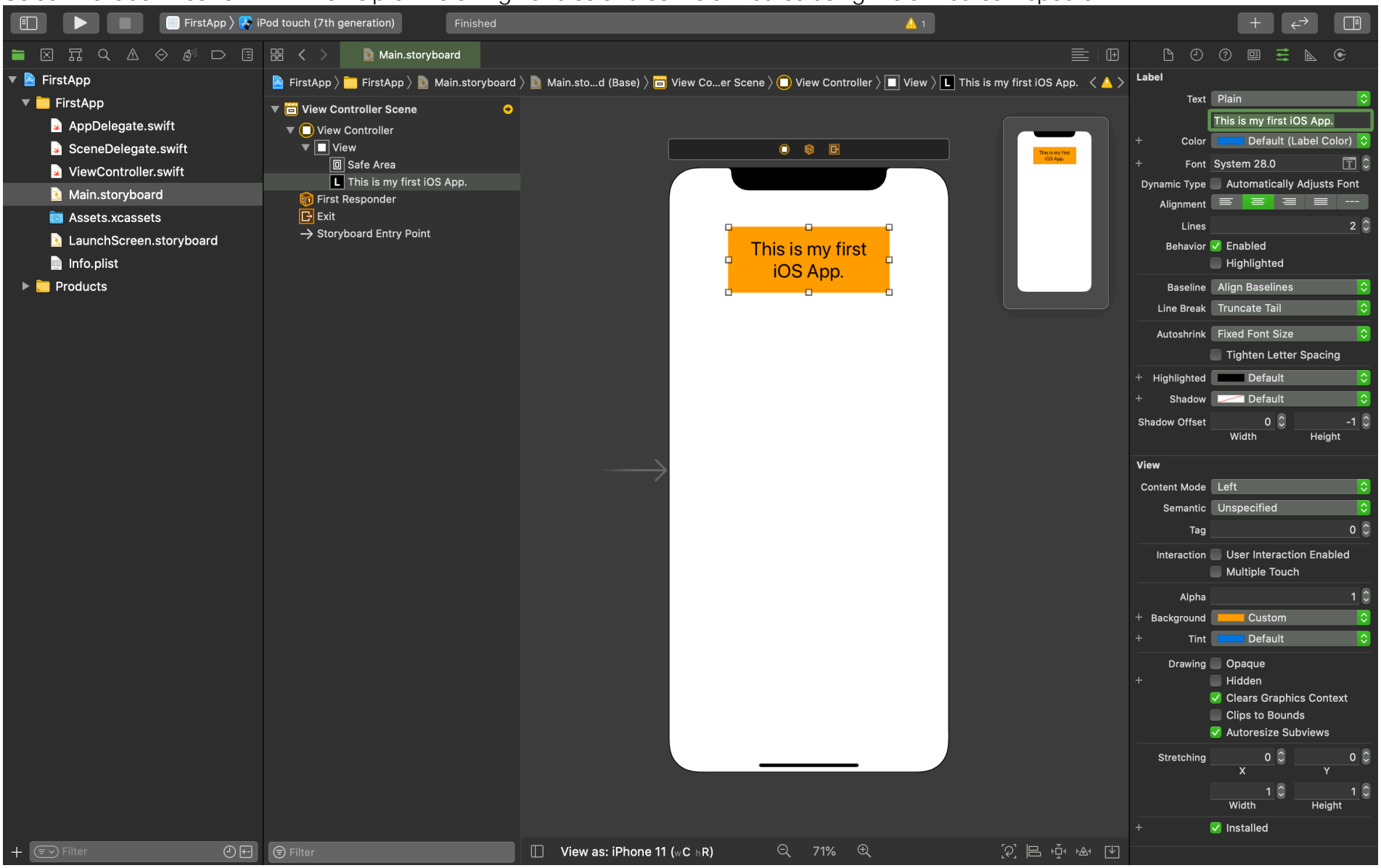




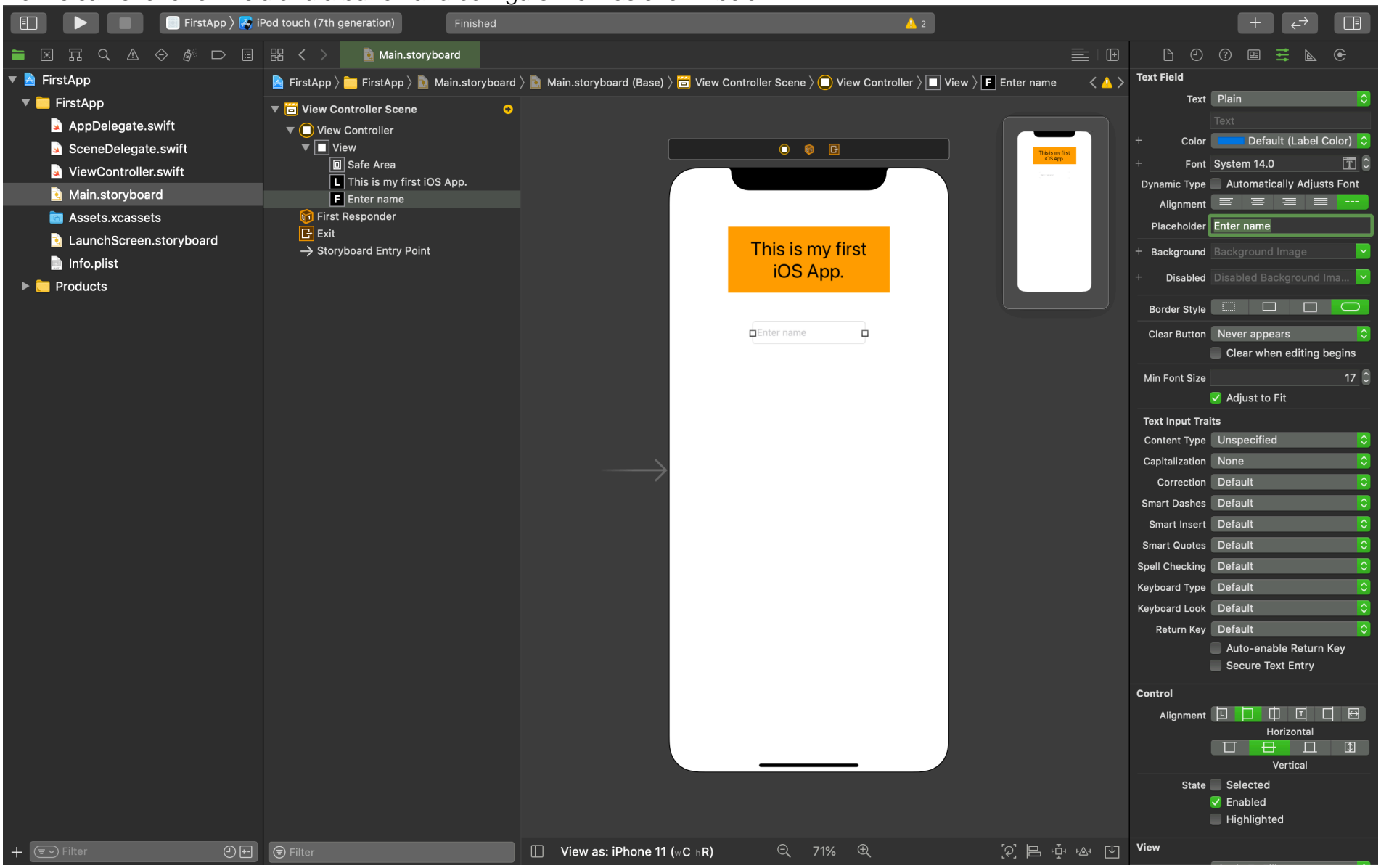
13. From the object library, drag the Label onto the canvas.

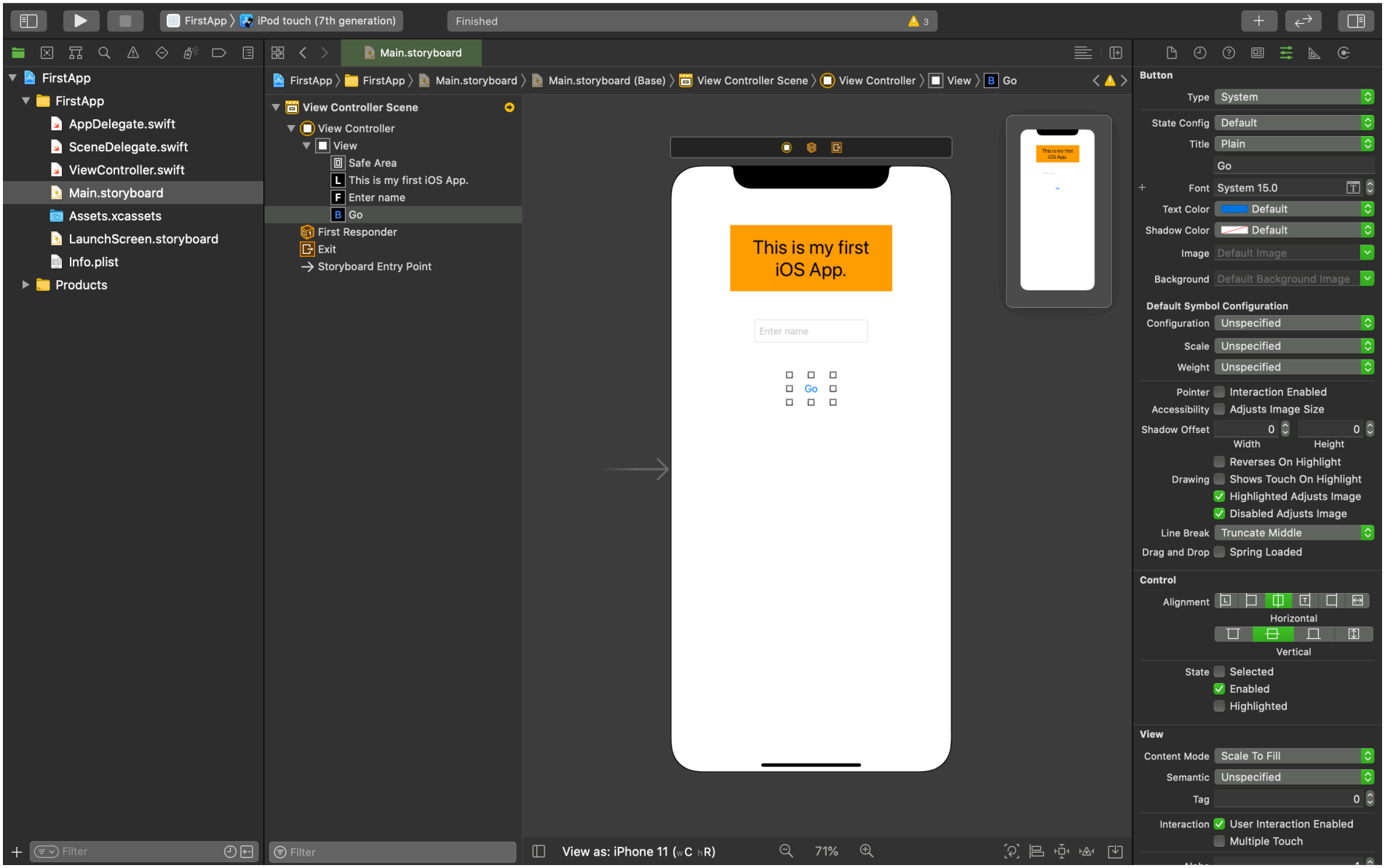


14. Select the label. Resize it with the help of the sizing handles and set its attributes using the attributes inspector.

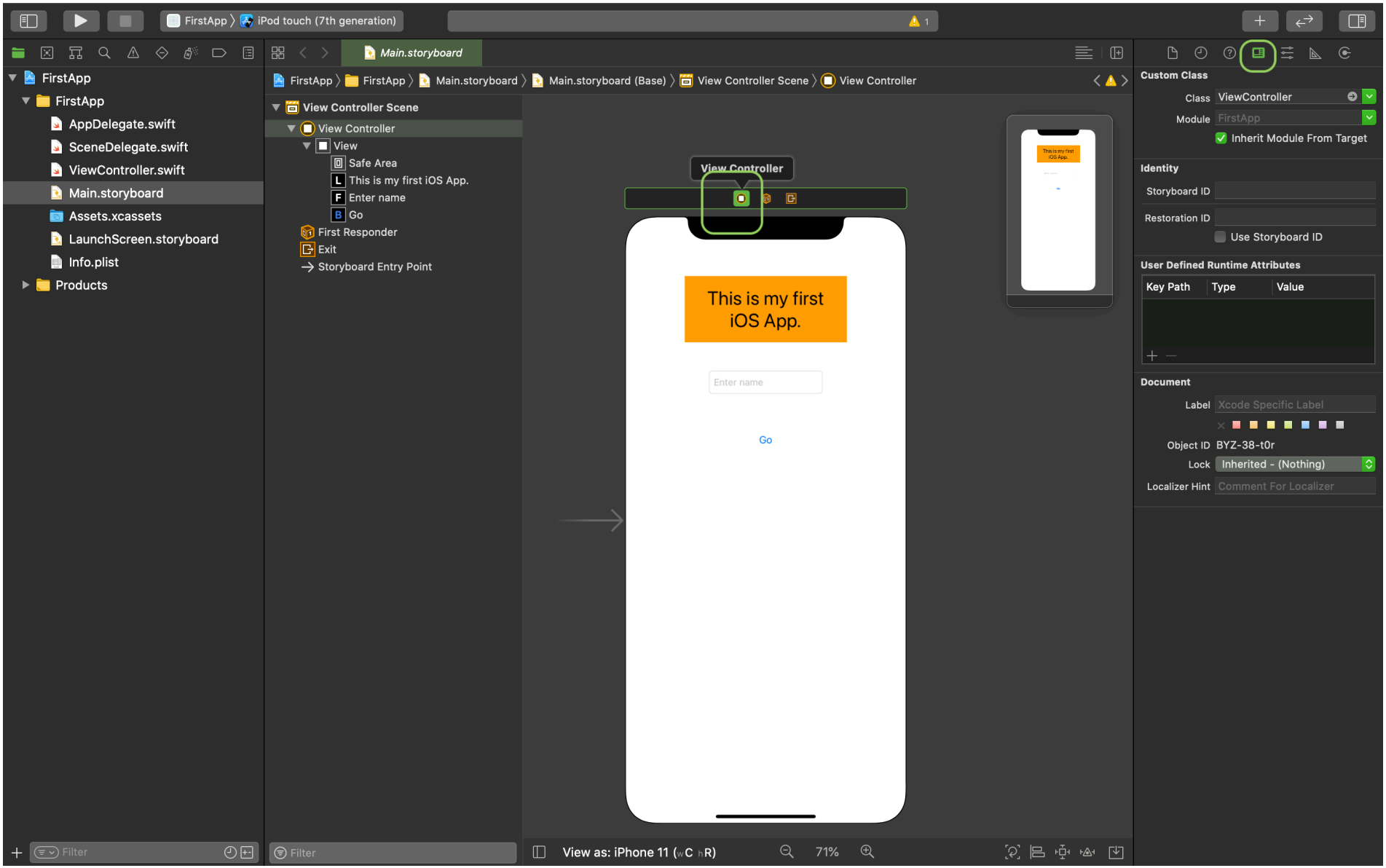


15. Do the same for a text field and a button and configure them as shown below.





16. Next we will establish IBOutlets and IBActions. To do that we will confirm that the ViewController representing our canvas is ViewController.swift.Select the circle with the white square at the top. Make sure the identity inspector is selected on the right hand side.

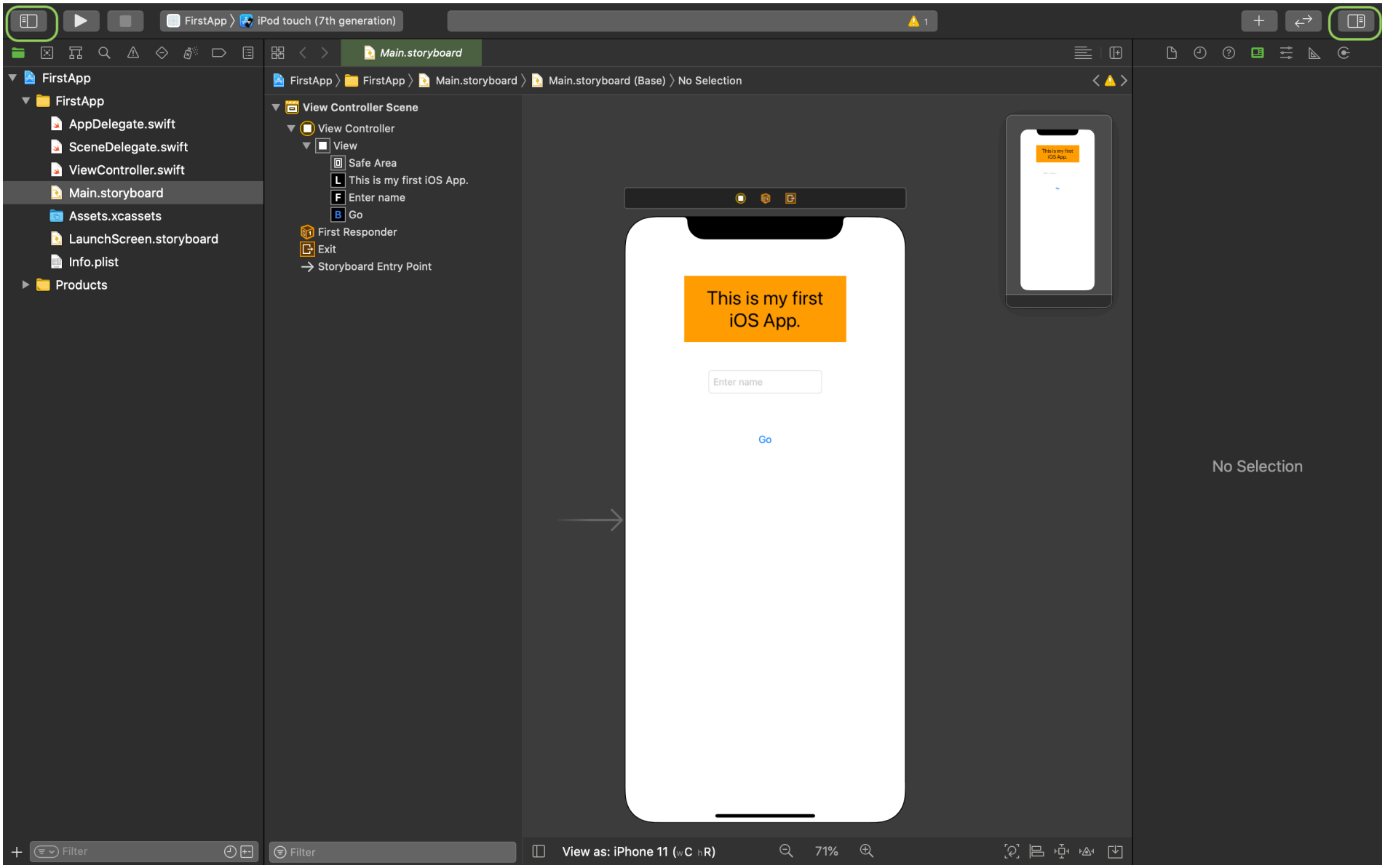


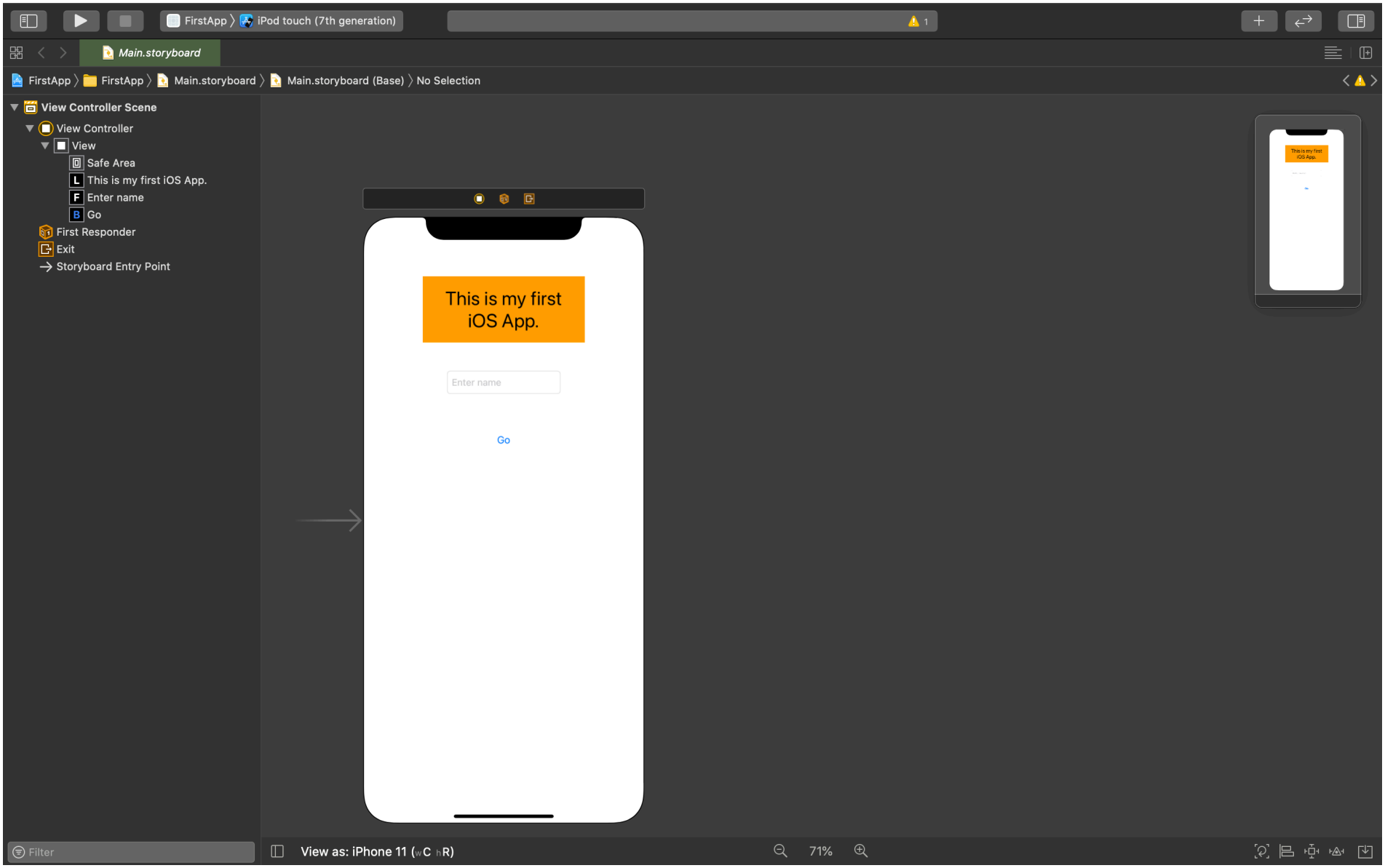
17. Make sure the *Class* is **ViewController** in the Identity inspector.

18. Let us make some space on the screen. We will hide the left and right hand side bars for now as we don't immediately need them.[1] Click the panel buttons in the left and right hand corners to hid/unhide the panels.

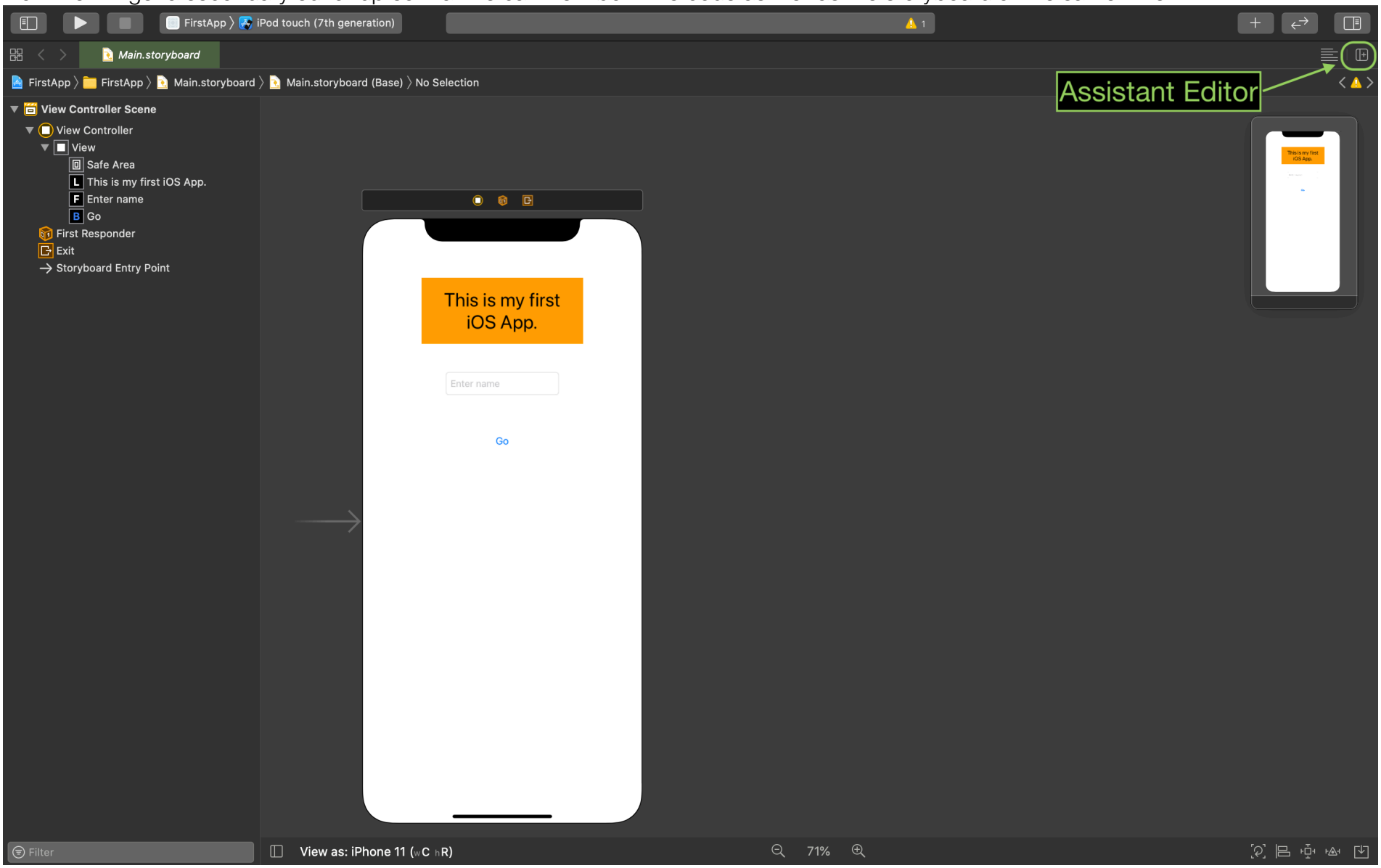
[1] This is not strictly required. However, it does help if there is more area on the screen. This makes the screen are less cluttered. Of course, computers with a large monitor like an iMac would not need such a step.

202327611082322384

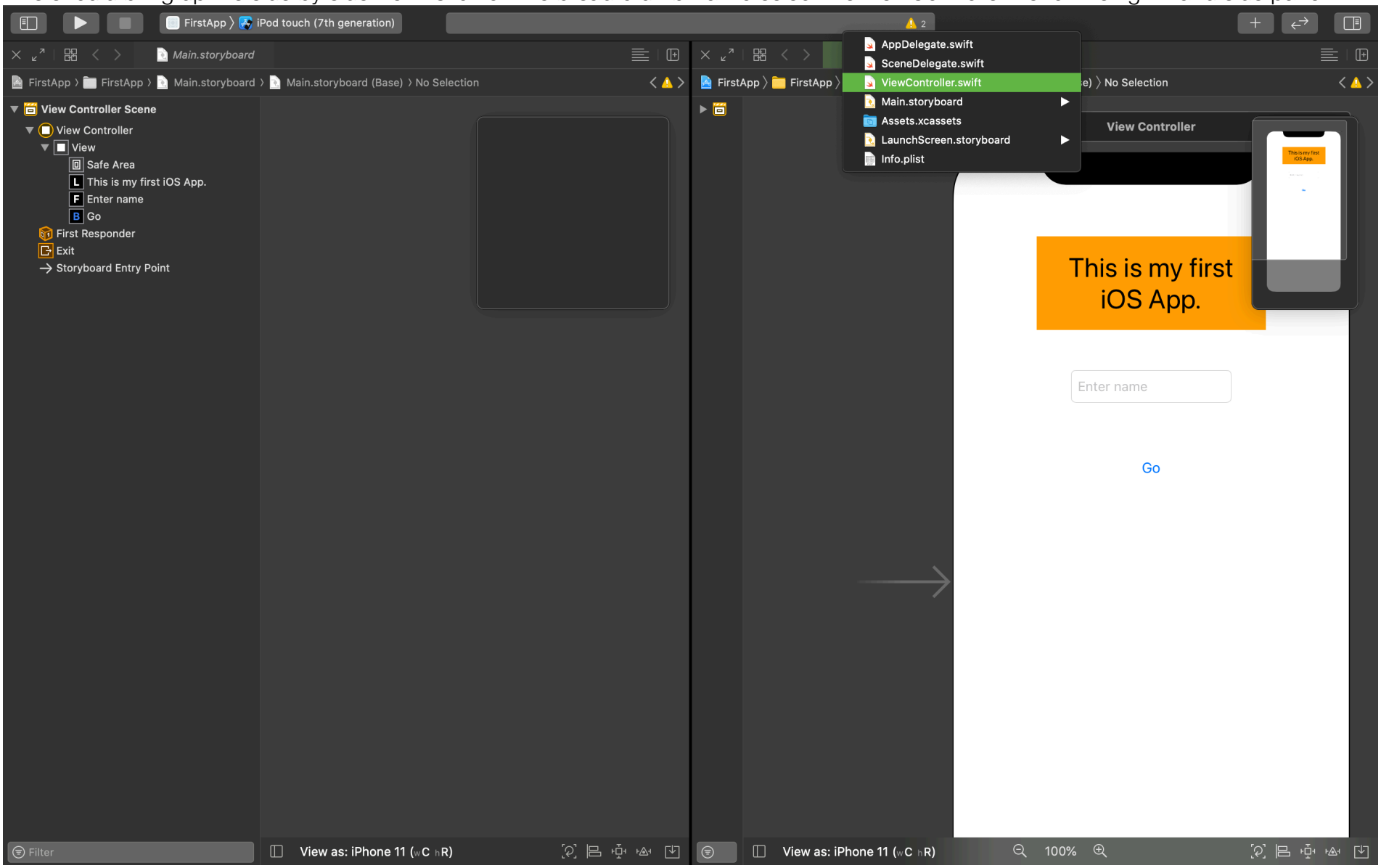




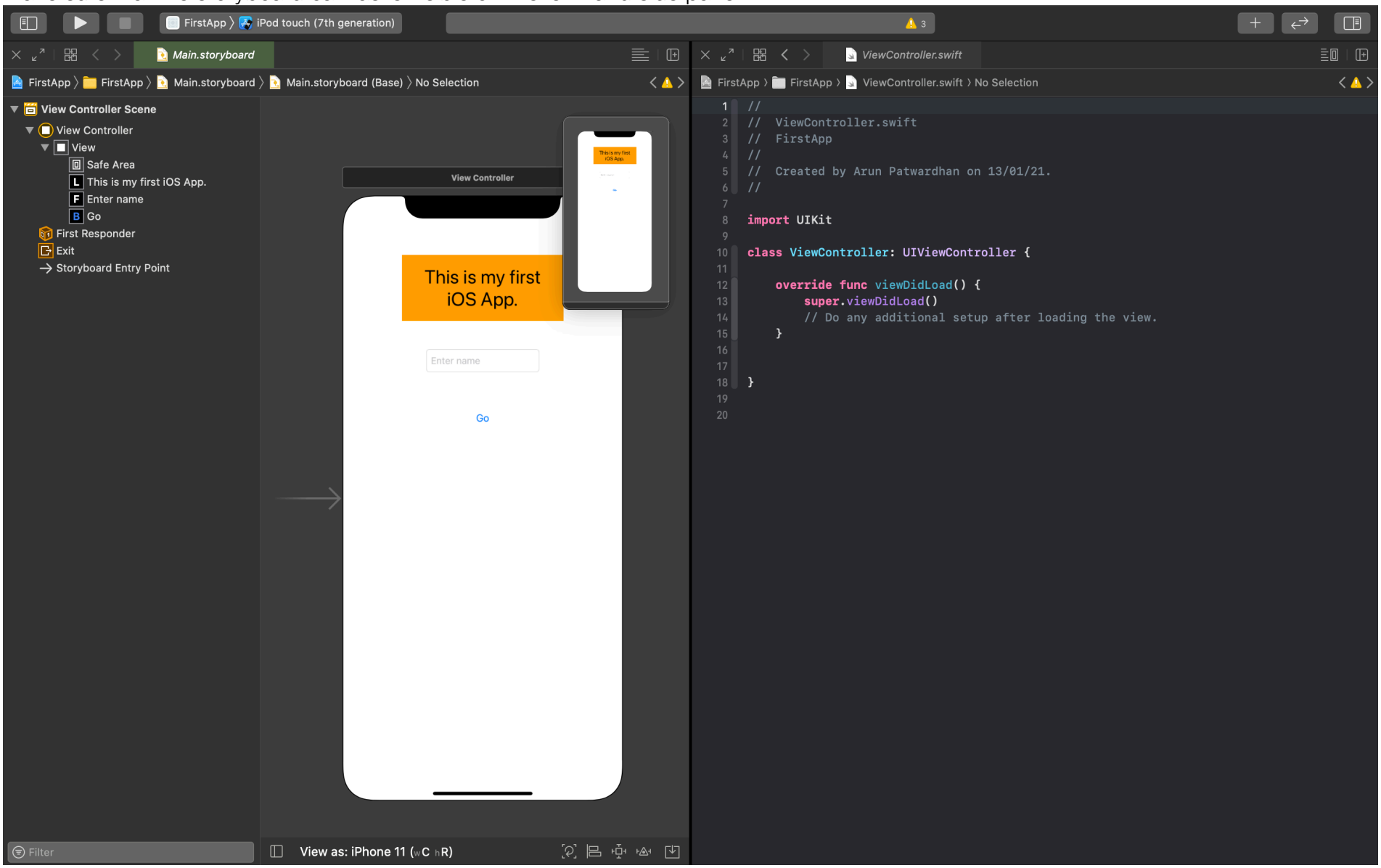
19. Now we will get a secondary editor up so that we can view both the code as well as the storyboard at the same time.



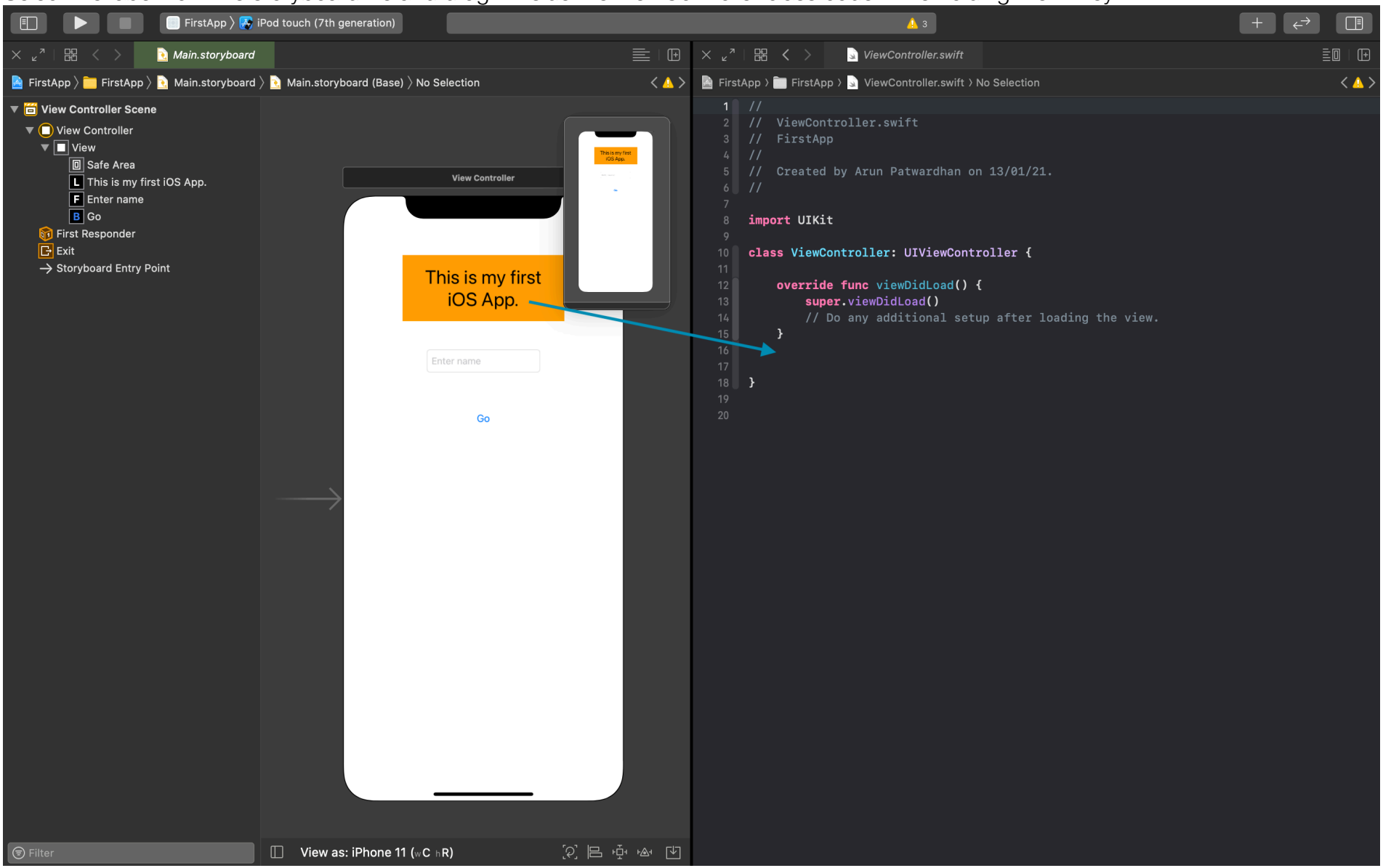
20. This should bring up the side by side view. Click on the bread crumb trail to select the ViewController file for the right hand side pane.



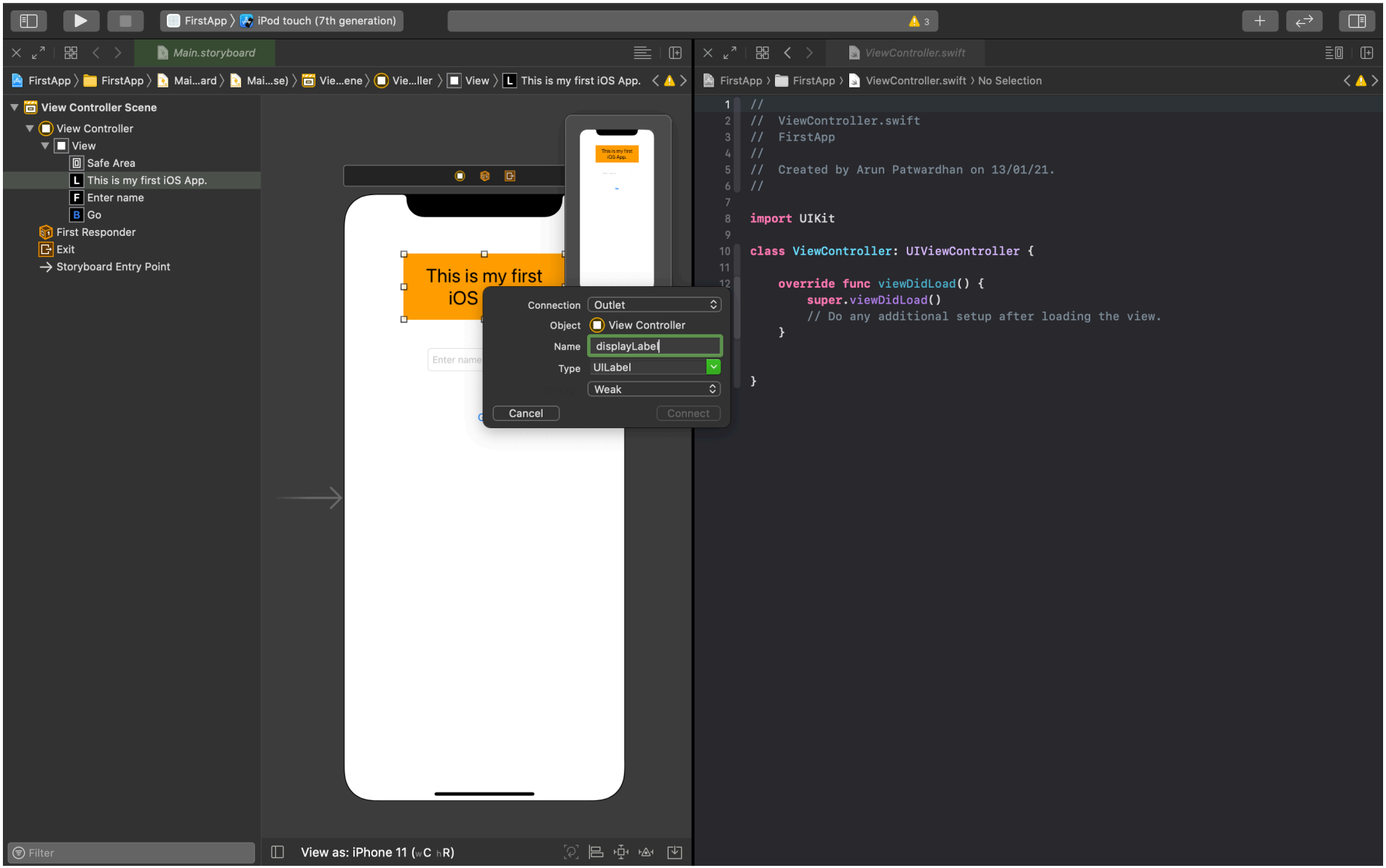
21. Make sure that the storyboard canvas is visible on the left hand side pane.



22. Select the label from the storyboard file and drag it inside the ViewController class code while holding the ^ key.



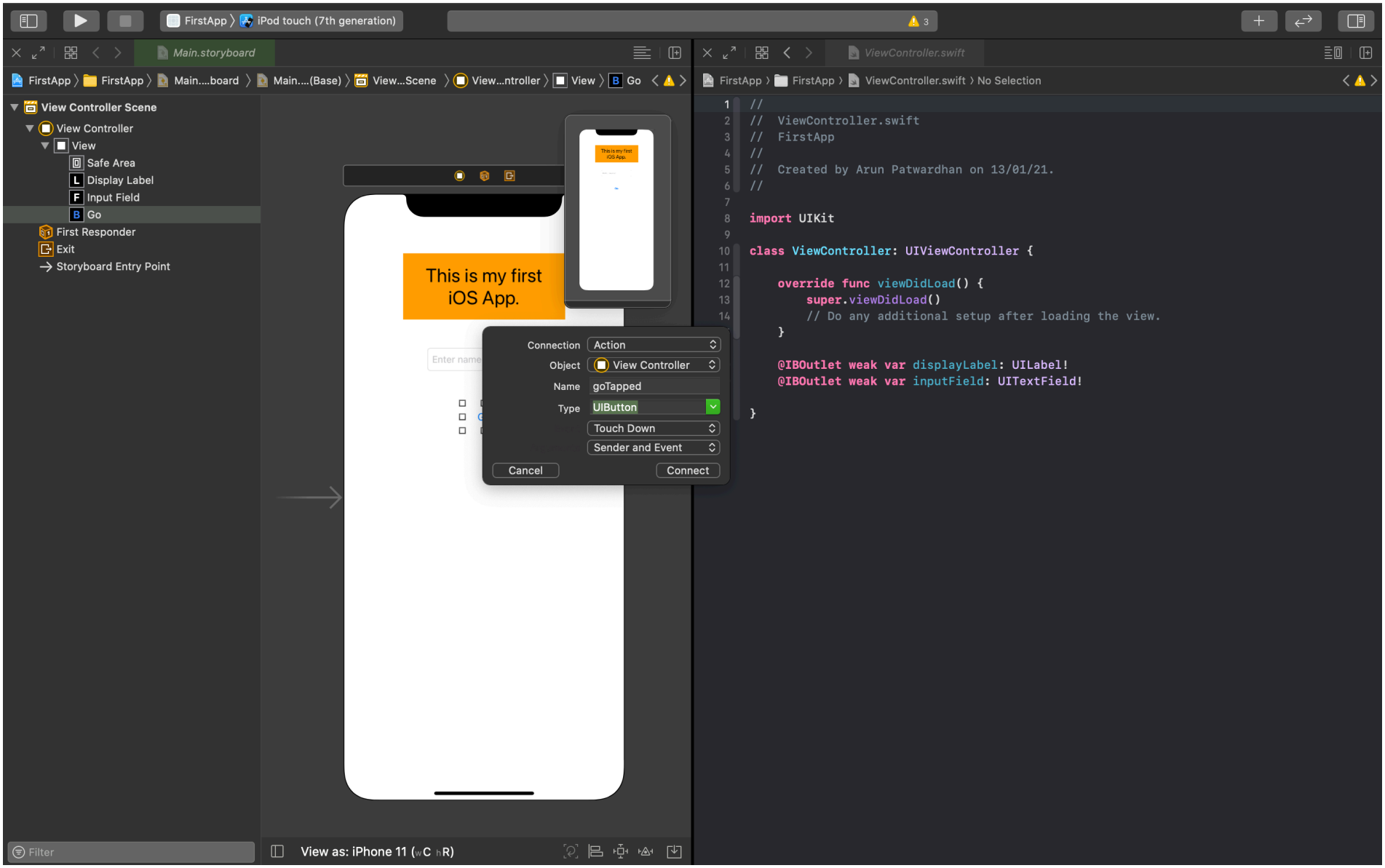
23. Once you let go a popup should appear. Let the connection type be outlet, name be displayLabel, type be UILabel. Leave everything else as is.



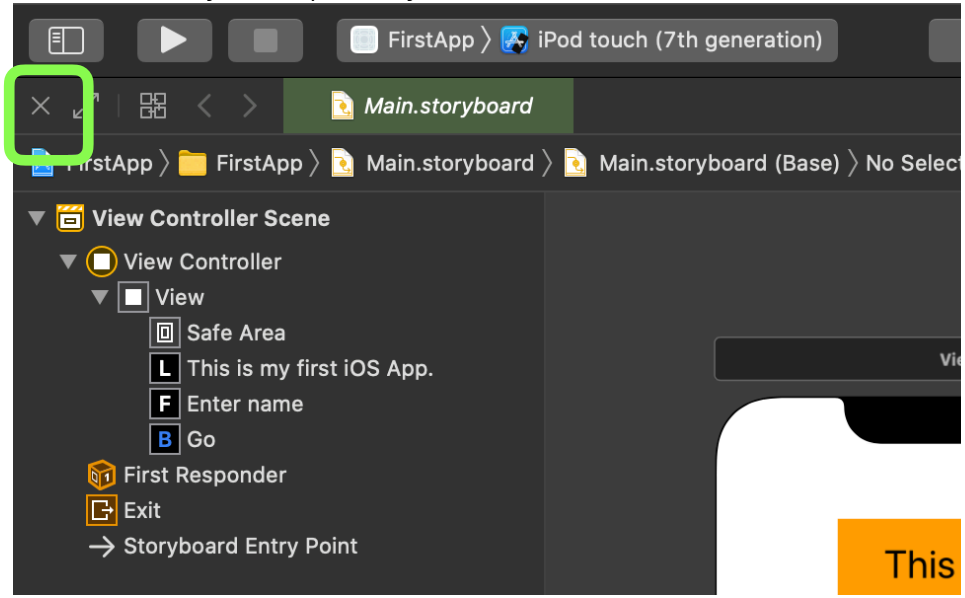
24. Click connect.

25. Do the same for a text field.
26. For the ‘Go’ button. We will establish an action. ^ drag it onto the code and select action instead of outlet. Give the action the following details:
    NAME: *goTapped*
    TYPE: *UIButton*
    EVENT: *Touch Down*

ARGUMENTS: *Sender and Event*



27. Click connect.
28. Close the storyboard pane by click on the ‘X’ in the left hand corner of the pane.

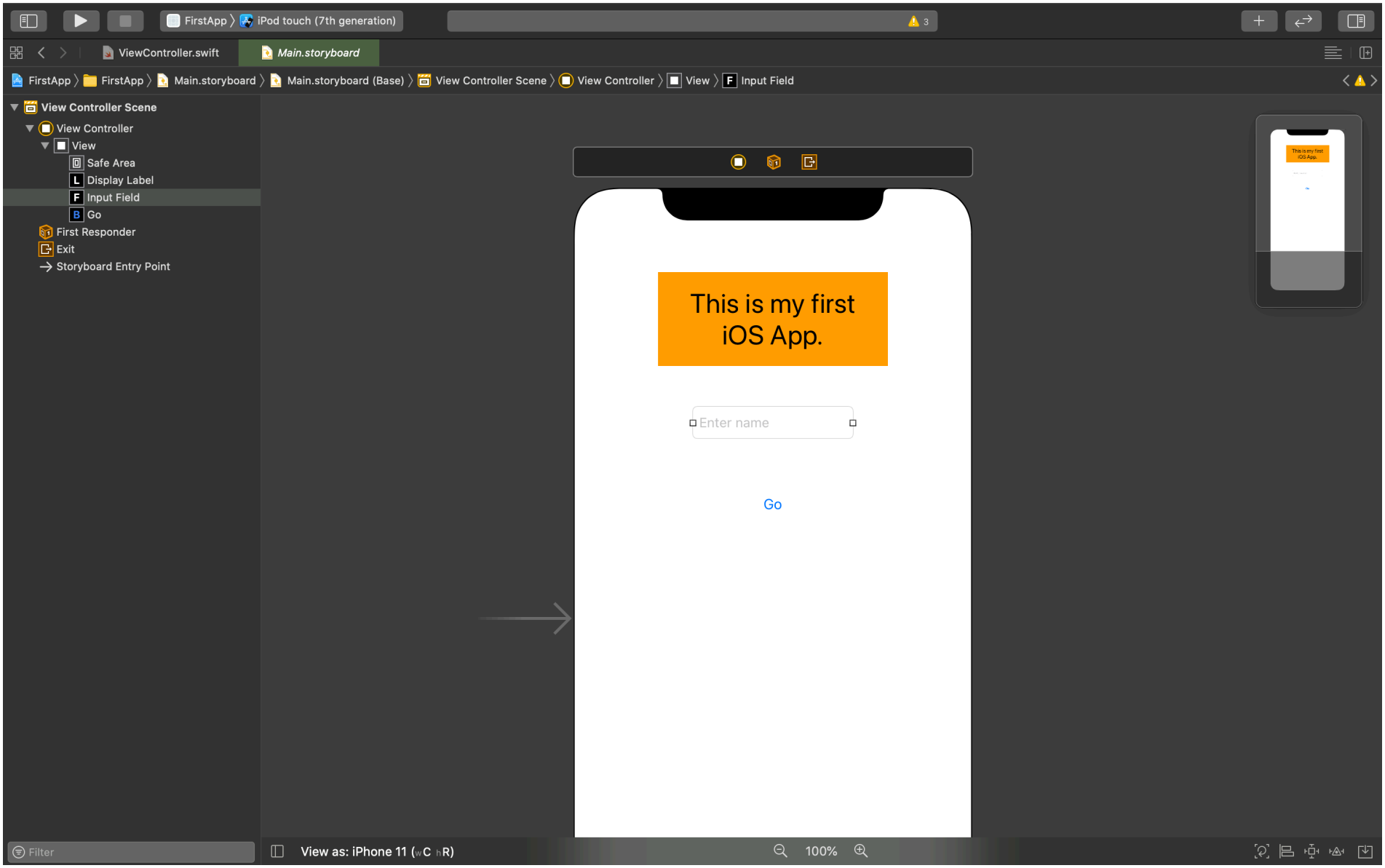


29. Update the IBAction method to take the text from the textfield and place it into the label.

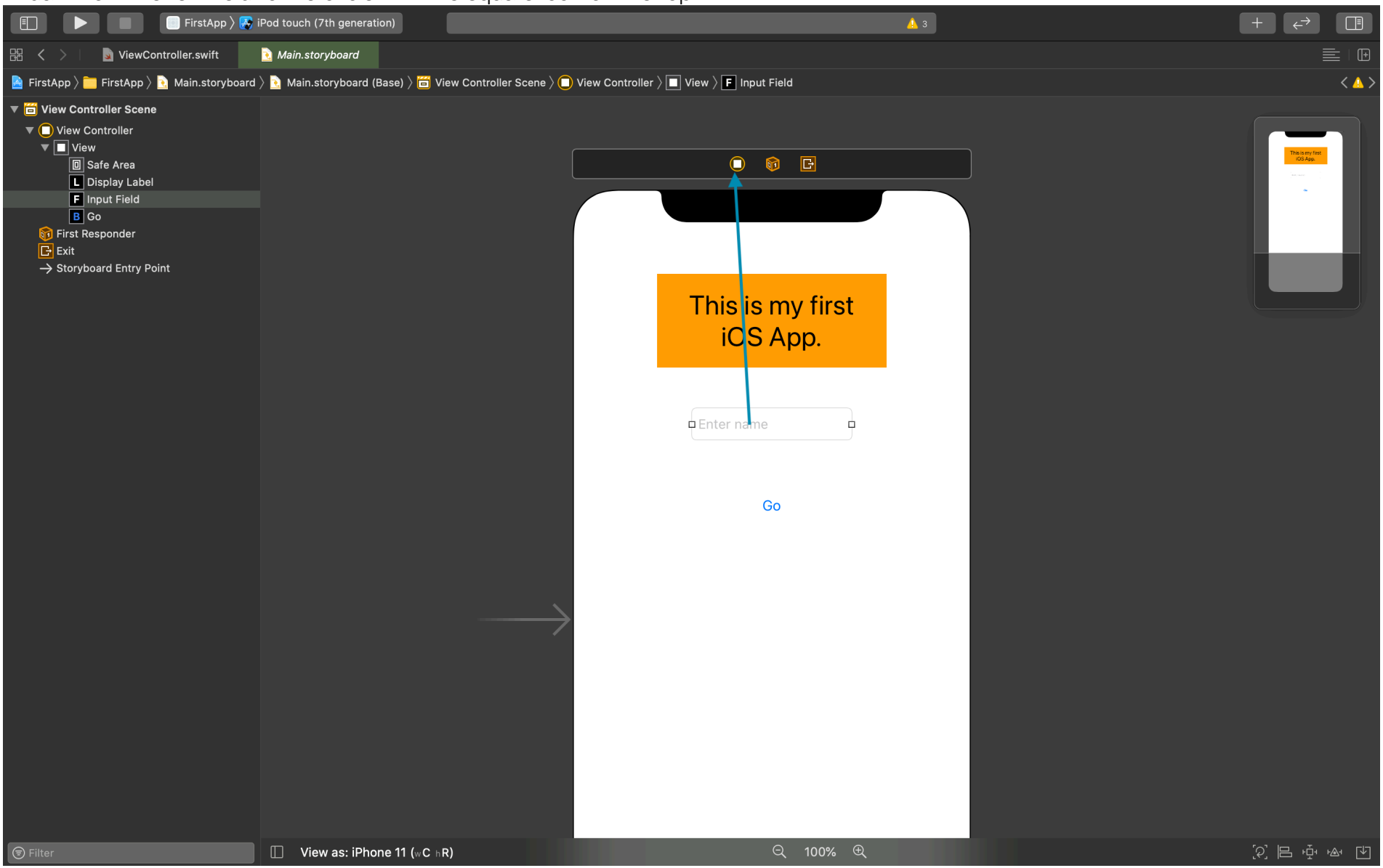
```
@IBAction func goTapped(_ sender: UIButton, forEvent event: UIEvent) {
    displayLabel.text = inputField.text
}
```

30. Run the app test the app to make sure it works well. Do this by clicking the play button in the upper left hand corner. You can also choose the device you wish to run the simulation in from the list.

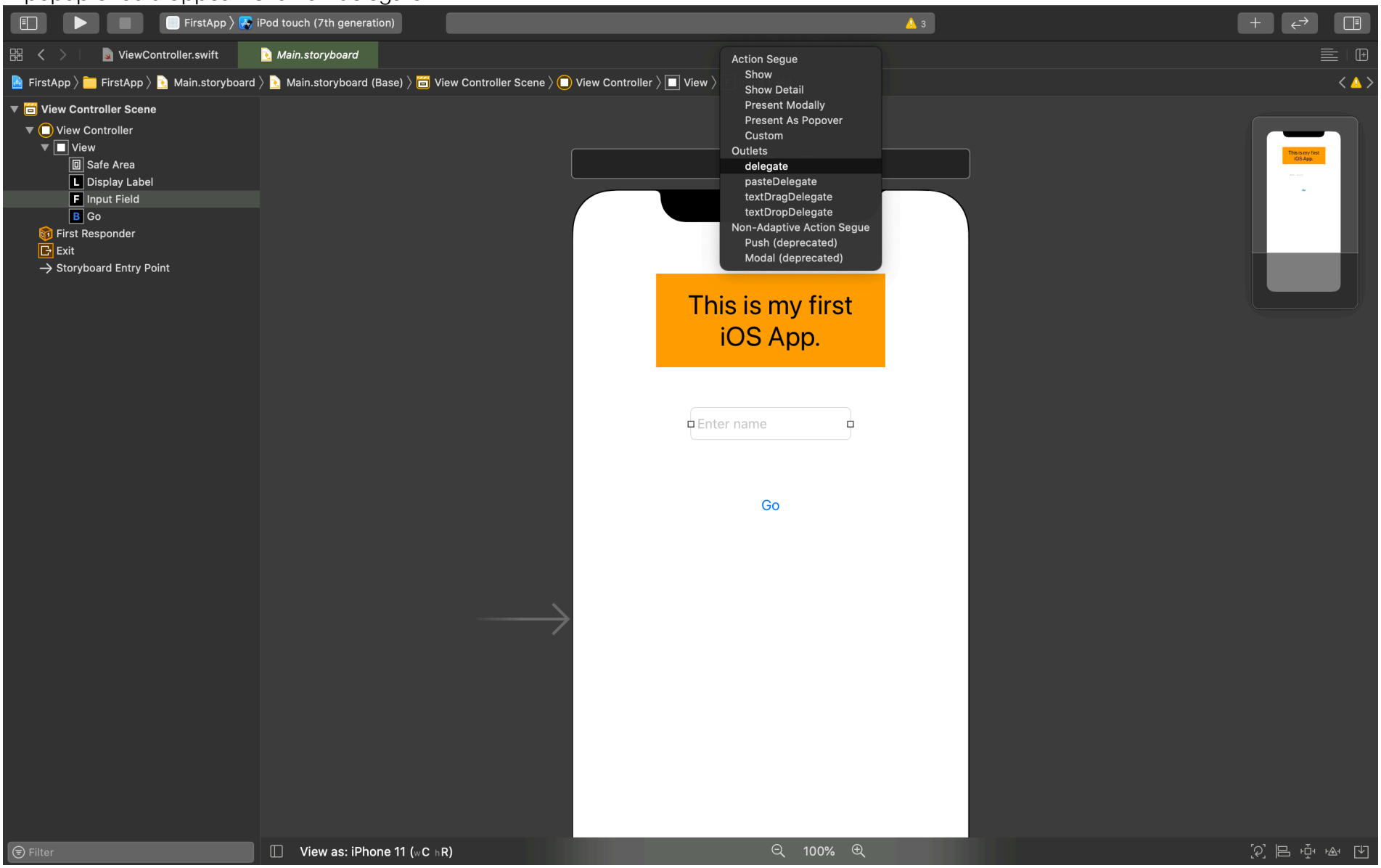
31. Next up we will implement the logic to dismiss the keyboard which appears. Bring back the Main.storyboard file. You can do this either by making the left hand side sidebar appear or select the file from the bread crumb trail. Select eh text field.



32. ^ dark from the text field to the circle with the square icon on the top.



33. A popup should appear. Click on delegate.



This establishes a connection to the delegate for the protocol responsible for handling the interaction with the textfield.

34. Switch back to the ViewController.swift file.
35. Implement the logic to hide the keyboard in an extension as shown below.

```
extension ViewController : UITextFieldDelegate {
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
}
```

36. Run the app. See if you are able to hide the keyboard that pops up when you try to type in the text field.

7
Chapter 7: Complex UI for a single device

In this chapter the reader will get familiar with the different UI components available to developers within the UIKit framework

Exercise 7.1: Getting familiar with Tables

Prerequisites

Should have completed exercise 3.1

Scope of Work

Learn how to create a user interface using storyboard.

Tools Required

Xcode 14.x or later

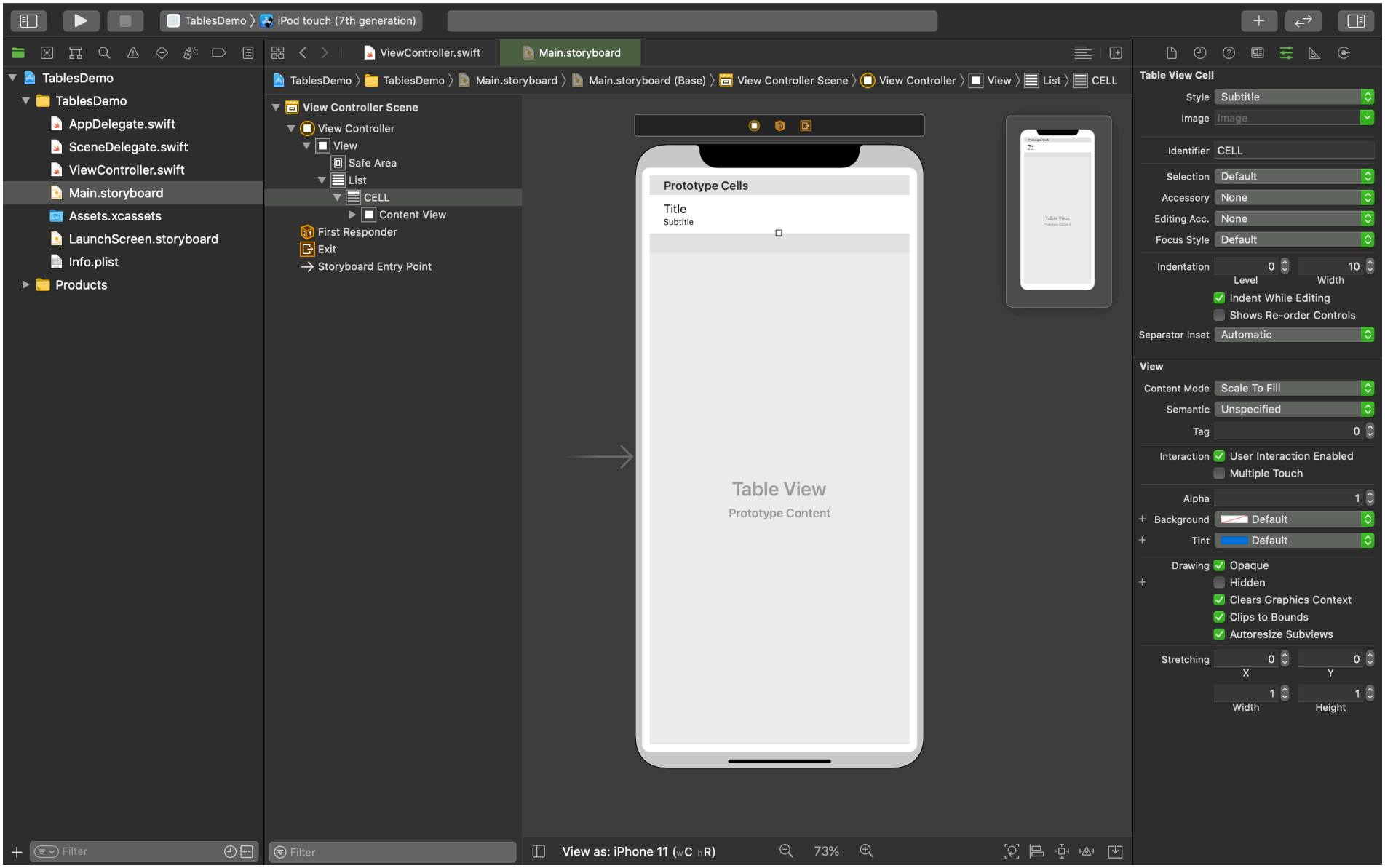
Outcome of the exercise

You should be comfortable with implementation of UITableViews.

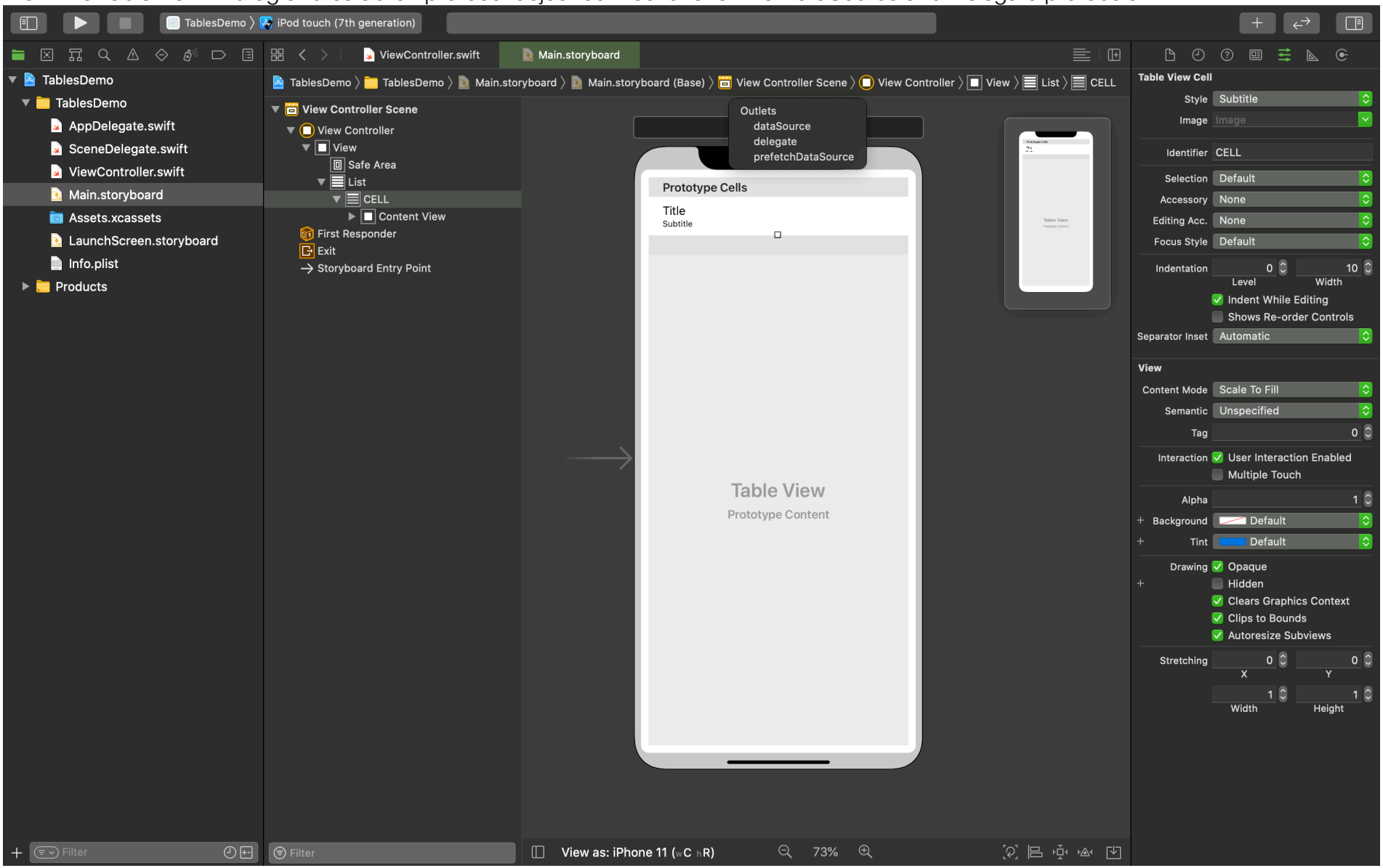
Steps

1. Open Xcode.
2. Create a new project by clicking *File* > *New* > *Project.* Call it TablesDemo.
3. Switch to Main.storyboard
4. From the object library drag a table on to the canvas.
5. Resize the table so that it occupies the entire area on the screen.
6. Drag a UITableViewCell from the object library. Place it inside the table view.
7. Select it and set its identifier as “CELL” from the attributes inspector.

8. Your screen should look like this:



9. From the Table view ^ drag and establish protocol object connections for the DataSource and Delegate protocols.



10. Create an IBOutlet for the Table view in the ViewController.swift file.
11. Switch to the ViewController.swift file.
12. Declare a struct that will hold all the constants.

```
/**
 Contains table constants

 `rowCount`:
 Contains the number of rows in the table.

 `cellIdentifier`:
 Holds the cell identifier for the prototype cell.

 - date: 15th January 2021
 - author: Arun Patwardhan
 - version: 1.0
 - copyright: Amaranthine 2021.
 */
struct TableValues {
    static let rowCount         : Int       = 10
    static let cellIdentifier   : String    = "CELL"
}
```

13. Create an extension for the ViewController class which conforms to the UITableViewDataSource protocol.
14. You may get an error saying that protocol implementation not complete. That is okay. We will be adding stubs to complete the implementation momentarily. You can either do it manually or click on the red and white circle representing the error to let Xcode fix the

issue by adding the stubs for you. Either way, your code should look like this once done.

```
extension ViewController : UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        <#code#>
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        <#code#>
    }
}
```

15. Let us start off by implementing the first method “numberOfRowsInSection”. Write the following piece of code in the function.

```
return TableValues.rowCount
```

16. Next we will focus on the “cellForRowAt" method. Write the following code in the function.

```
var cell    : UITableViewCell?  = tableView.dequeueReusableCell(withIdentifier:
TableValues.cellIdentifier)

if cell == nil {
    cell = UITableViewCell(style: UITableViewCell.CellStyle.subtitle,
                           reuseIdentifier: TableValues.cellIdentifier)
}

cell?.textLabel?.text           = "Section: \(indexPath.section)"
cell?.detailTextLabel?.text     = "Row: \(indexPath.row)"

return cell!
```

17. Run the app. Play around with extra row values to see how a large table appears.

Exercise 7.2: Customising Tables

Prerequisites

Should have completed exercise 7.1

Scope of Work

Learn how to create a user interface using storyboard.

Tools Required

Xcode 14.x or later

Outcome of the exercise

You should be comfortable with implementation of UITableViews.

Steps

1. Duplicate the previous project. We will be working on the duplicate.
2. Create a new model class called DataSource.[2]

```
import UIKit

struct DataSource {
    fileprivate var subjectName : String = ""
    var subjectImage            : UIImage?
    var subjectTitle            : String = ""
    var subjectDescription      : String = ""

    init(SubjectNamed name : String, havingIcon image : UIImage, titled title : String,
andDescribedAs description : String) {
        self.subjectName = name
        self.subjectImage = image
        self.subjectTitle = title
        self.subjectDescription = description
    }
}

extension DataSource : CustomStringConvertible {
    var description: String {
        return "Subject Image name: \(self.subjectName)\nSubject title: \(self.subjectTitle)
\nSubject description: \(self.subjectDescription)"
    }
}
```

[2] If you are performing this activity in class the please ask the instructor to share the ready code with you.

3. Next create a new class called TableData. This class represents the data that is to be shown on the table.
[3]

```
struct TableModel {
    private var internalModel : [Character : [DataSource]] = [:]

    fileprivate let sections : [Character] =
["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x",
"y","z"]

    var model : [Character : [DataSource]] {
        return self.internalModel
    }
```

[3] As before. If you are performing these activities in class then the instructor will share this code with you.

```
    init() {
        for key in sections {
            self.internalModel[key] = []
        }

        //m
        let tempData : DataSource = DataSource(SubjectNamed: "Mic",
                                               havingIcon: UIImage(systemName:
"mic.circle.fill")!,
                                               titled: "Mic",
                                               andDescribedAs: "Object used to capture audio
recordings from the user. Usually placed in front of the user.")
        self.internalModel["m"]?.append(tempData)

        let tempData10 : DataSource = DataSource(SubjectNamed: "Moon",
                                               havingIcon: UIImage(systemName:
"moon.circle.fill")!,
                                               titled: "Moon",
                                               andDescribedAs: "Celestial object that is found
revolving around the earth. One side of the moon is called as the dark side of the moon.")
        self.internalModel["m"]?.append(tempData10)
```

```
        //i
        let tempData1 : DataSource = DataSource(SubjectNamed: "Rupeee",
                                                havingIcon: UIImage(systemName:
"indianrupeesign.circle.fill")!,
                                                titled: "Rupee symbol",
                                                andDescribedAs: "Represents the indian currency
symbol. It is a combination of the indian devnagiri script and english script.")
        self.internalModel["i"]?.append(tempData1)

        //b
        let tempData2 : DataSource = DataSource(SubjectNamed: "Bandage",
                                                havingIcon: UIImage(systemName: "bandage.fill")!,
                                                titled: "Bandage",
                                                andDescribedAs: "Object used to quickly seal a
wound. It also plays the role of providing medical treatment to the wound while being selaed.")
        self.internalModel["b"]?.append(tempData2)

        let tempData9 : DataSource = DataSource(SubjectNamed: "Bus",
                                                havingIcon: UIImage(systemName:
"bus.doubledecker")!,
                                                titled: "Double decker bus",
                                                andDescribedAs: "Multi level bus which allows
transportation of many more passengers than a conventional bus.")
        self.internalModel["b"]?.append(tempData9)
```

```
        //w
        let tempData3 : DataSource = DataSource(SubjectNamed: "Wand",
                                                havingIcon: UIImage(systemName:
"wand.and.stars")!,
                                                titled: "Wand",
                                                andDescribedAs: "Magic wand, typically shown in
movies where some kind of magic is involved.")
        self.internalModel["w"]?.append(tempData3)

        let tempData12 : DataSource = DataSource(SubjectNamed: "Waveform",
                                                havingIcon: UIImage(systemName:
"waveform.path.ecg.rectangle")!,
                                                titled: "ECG",
                                                andDescribedAs: "This is the shape of the
electrical signal that is produced with the help of the Elctro-cardio-graph (ECG) machine.")
        self.internalModel["w"]?.append(tempData12)
```

```
        //a
        let tempData4 : DataSource = DataSource(SubjectNamed: "Ant",
                                                havingIcon: UIImage(systemName: "ant.fill")!,
                                                titled: "Ant",
                                                andDescribedAs: "A small insect that crawls on
the ground. Known to burrow and build complex structures for housing underground.")
        self.internalModel["a"]?.append(tempData4)

        let tempData5 : DataSource = DataSource(SubjectNamed: "Airplane",
                                                havingIcon: UIImage(systemName: "airplane")!,
                                                titled: "Airplane",
                                                andDescribedAs: "A flying machine that is used to
transport people and objects over vast distances.")
        self.internalModel["a"]?.append(tempData5)

        //t
        let tempData6 : DataSource = DataSource(SubjectNamed: "TV",
                                                havingIcon: UIImage(systemName:
"tv.music.note.fill")!,
                                                titled: "TV",
                                                andDescribedAs: "An object used by billions of
people for watching events that take place far far from home.")
        self.internalModel["t"]?.append(tempData6)
```

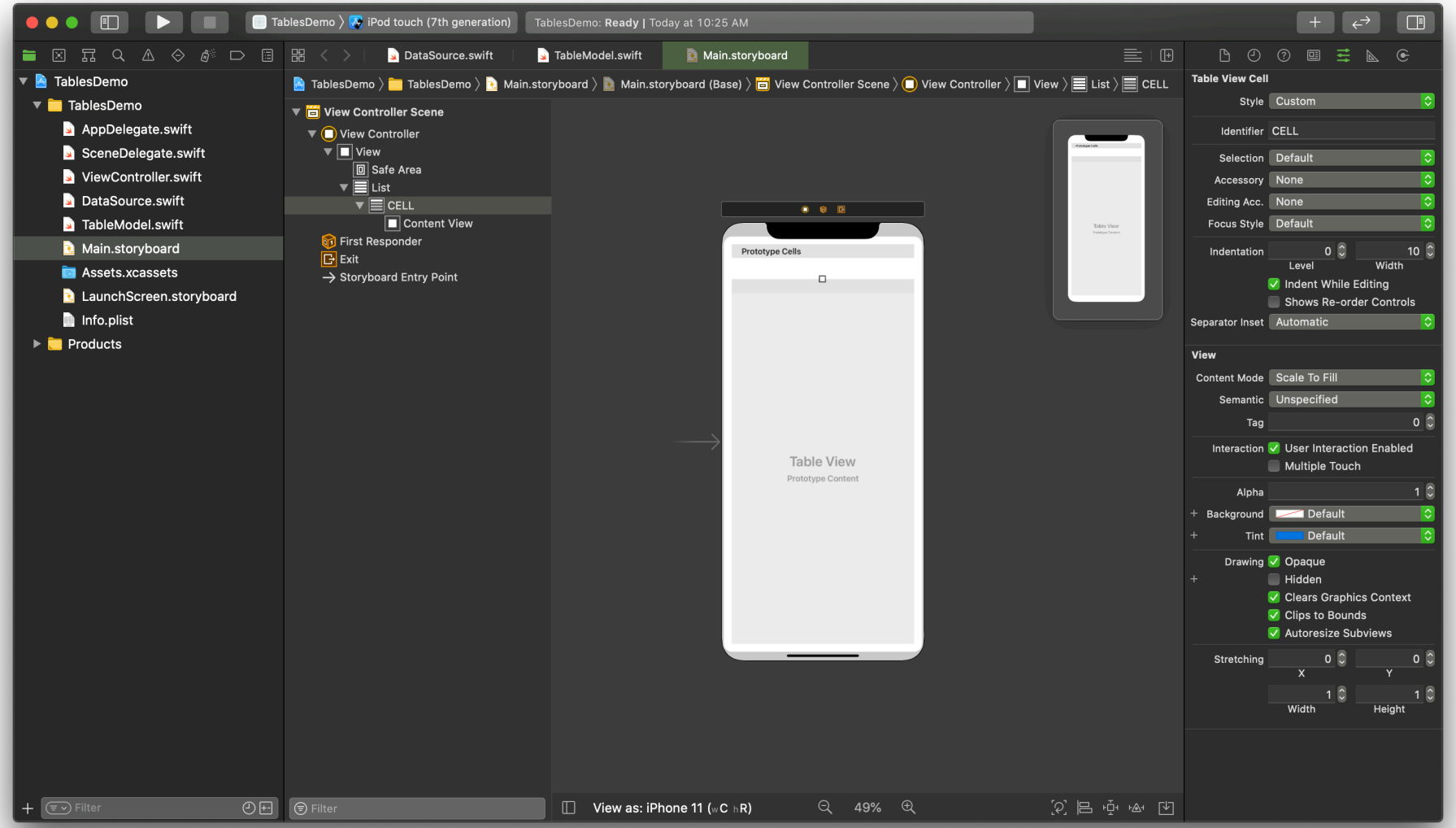
```
        //c
        let tempData7 : DataSource = DataSource(SubjectNamed: "Cart",
                                                havingIcon: UIImage(systemName:
"cart.fill.badge.plus")!,
                                                titled: "Cart",
                                                andDescribedAs: "Used to represent an object that
allows patrons to carry all the items they wish to purchase around a shopping mart.")
        self.internalModel["c"]?.append(tempData7)

        let tempData8 : DataSource = DataSource(SubjectNamed: "Credit Card",
                                                havingIcon: UIImage(systemName:
"creditcard.fill")!,
                                                titled: "Credit Card",
                                                andDescribedAs: "Used to represent an object that
allows patrons to carry all the items they wish to purchase around a shopping mart.")
        self.internalModel["c"]?.append(tempData8)

        let tempData11 : DataSource = DataSource(SubjectNamed: "Cloud",
                                                havingIcon: UIImage(systemName:
"cloud.sun.fill")!,
                                                titled: "Cloud",
                                                andDescribedAs: "A much beloved weather pattern
where clouds appear during the day and they partially cover the sun creating a nice effect in the
sky.")
        self.internalModel["c"]?.append(tempData11)
```

```
    }

    //MARK: - Subscript operator
    subscript(index : Character) -> [DataSource] {
        get {
            return self.internalModel[index]!
        }
    }
}
```

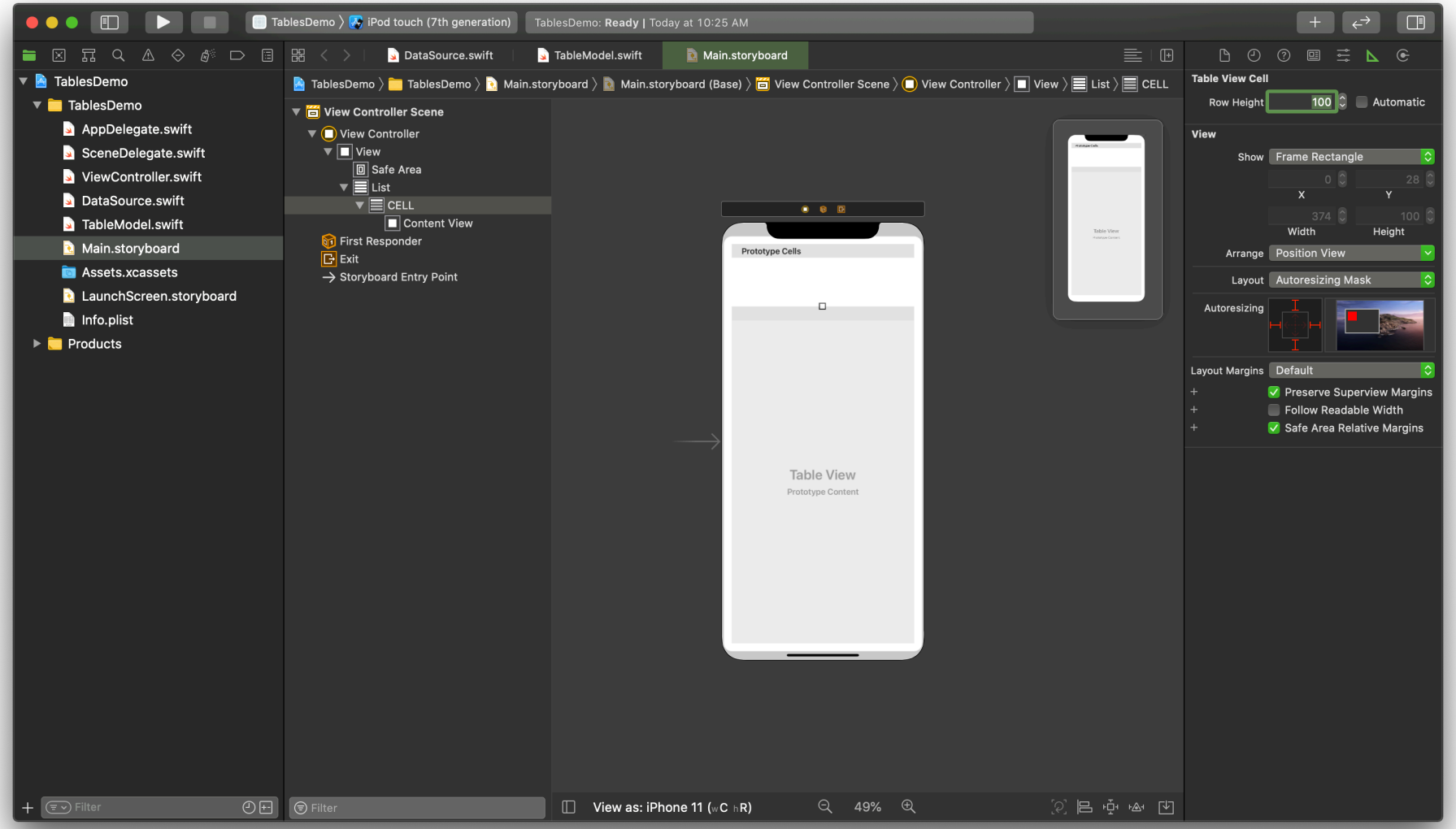
We are mocking the data in this case for testing purpose only. In the real world this data would be coming from somewhere else.

4. Switch to the Main.storyboard file.

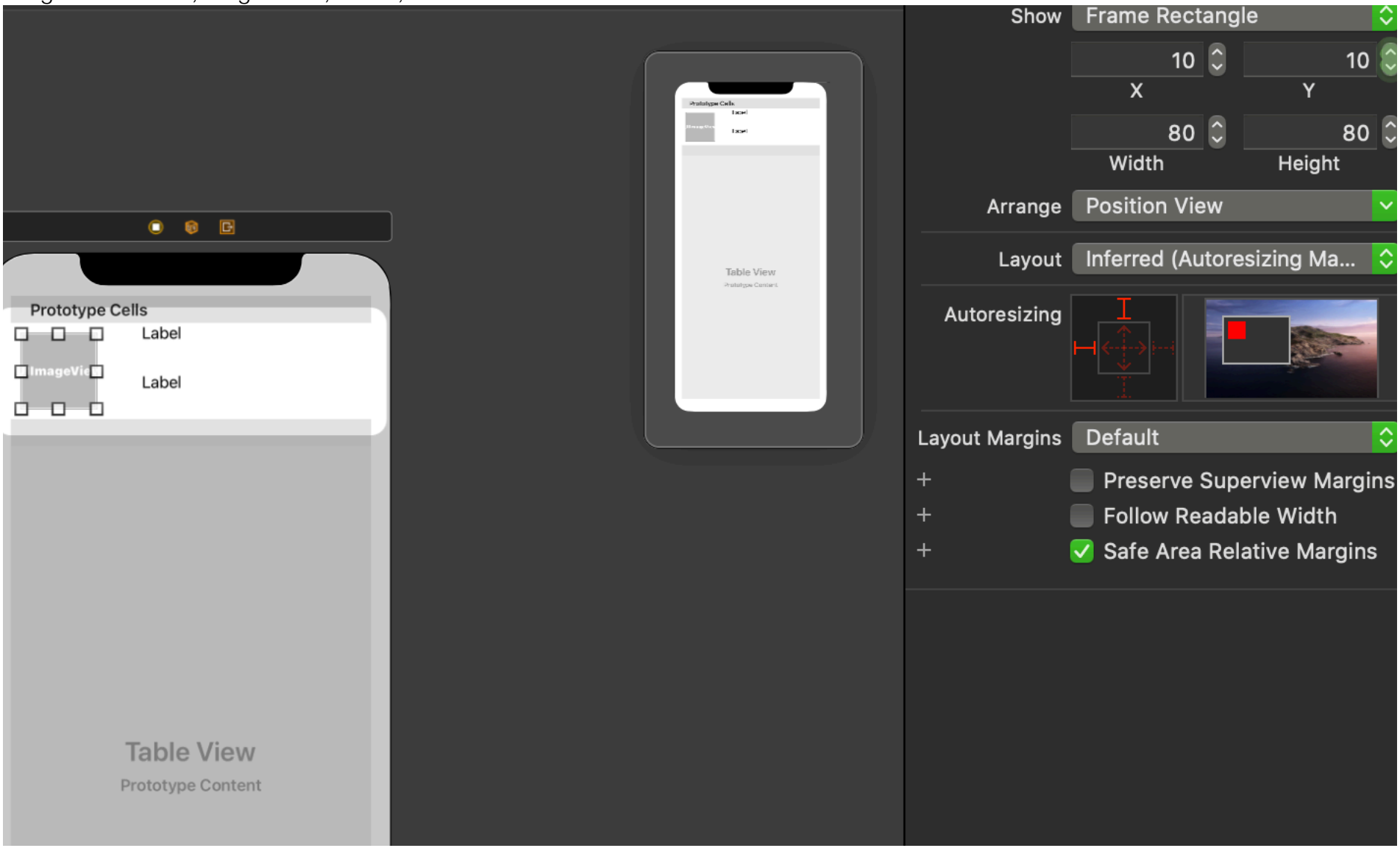
5. Change the cell style to custom.



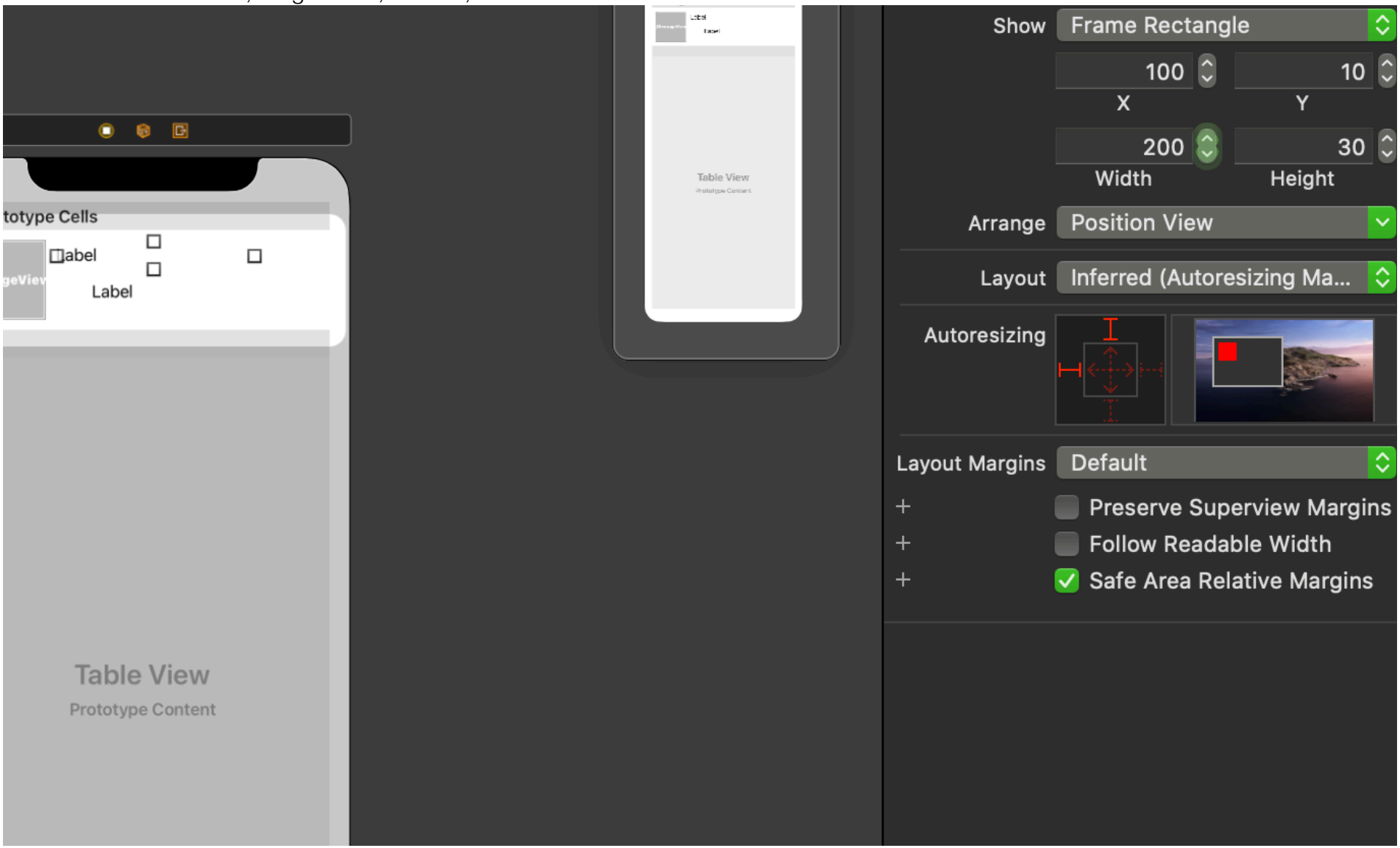
6. Fix the cell height to 100.



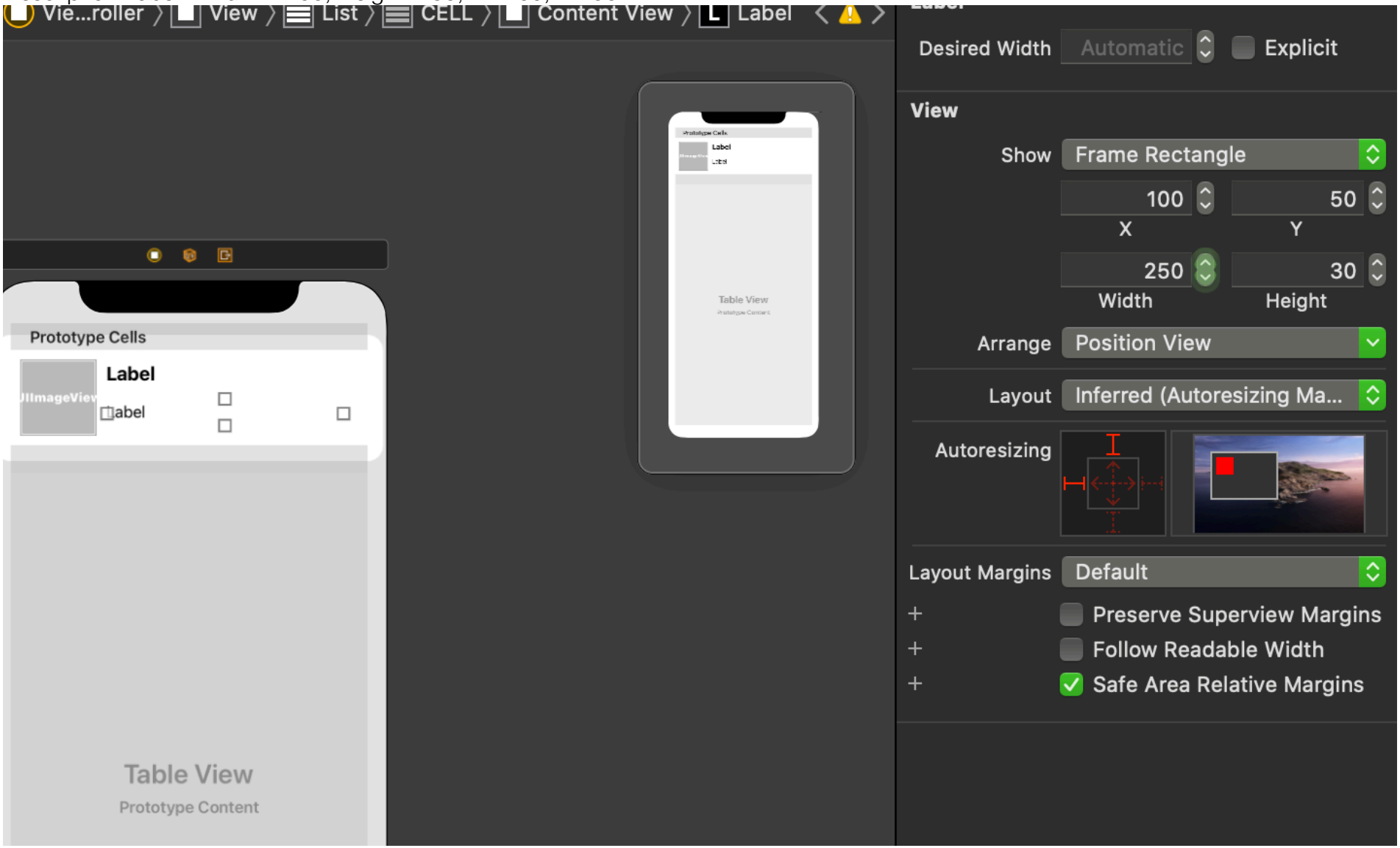
7. Add an UIImageVIew and 2 labels inside the cell. Give them the following dimensions.
   Image: Width = 80, Height = 80, X= 10, Y = 10



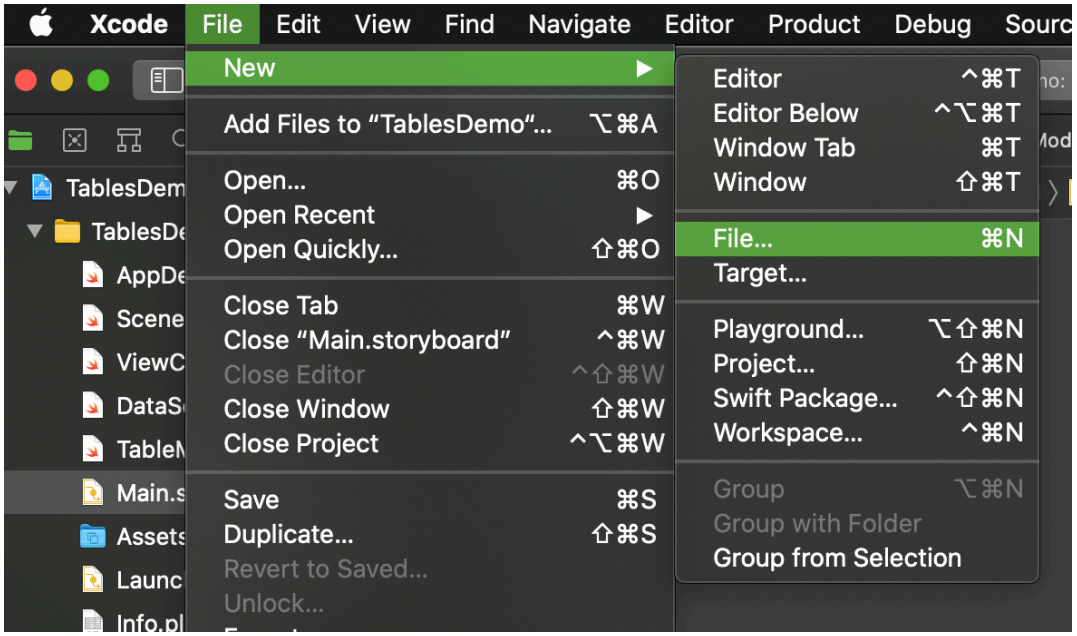
Title Label: Width = 100, Height = 30, X= 100, Y= 10



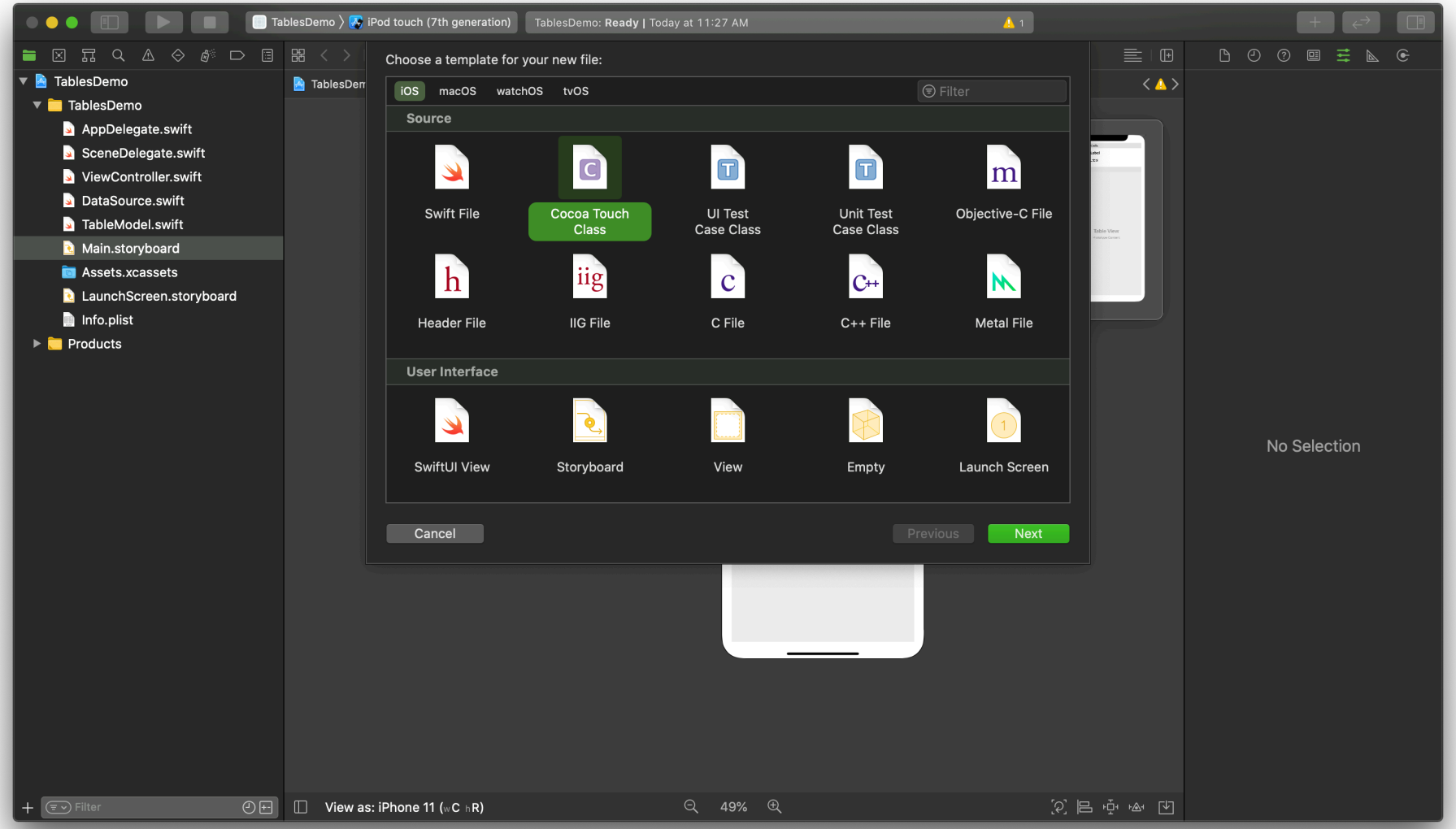
Description Label: Width = 250, Height = 30, X= 100, Y= 50



8. Now we will create a new class for our custom cell. To do that click on *File* > *New* > *File* in the menu bar.

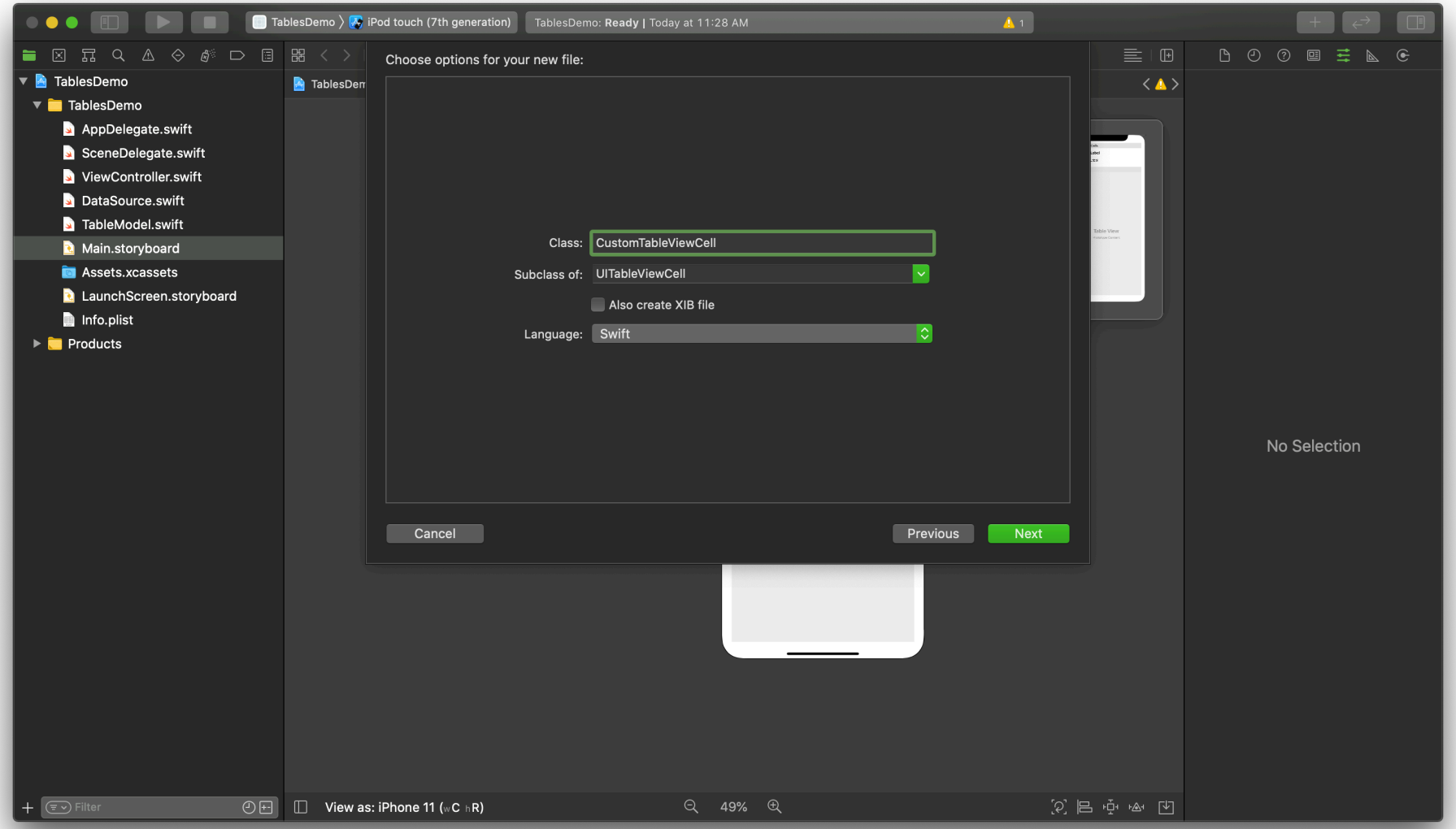


9. This brings up the template wizard. Select *Cocoa Touch Class* from the options.



10. Click next.

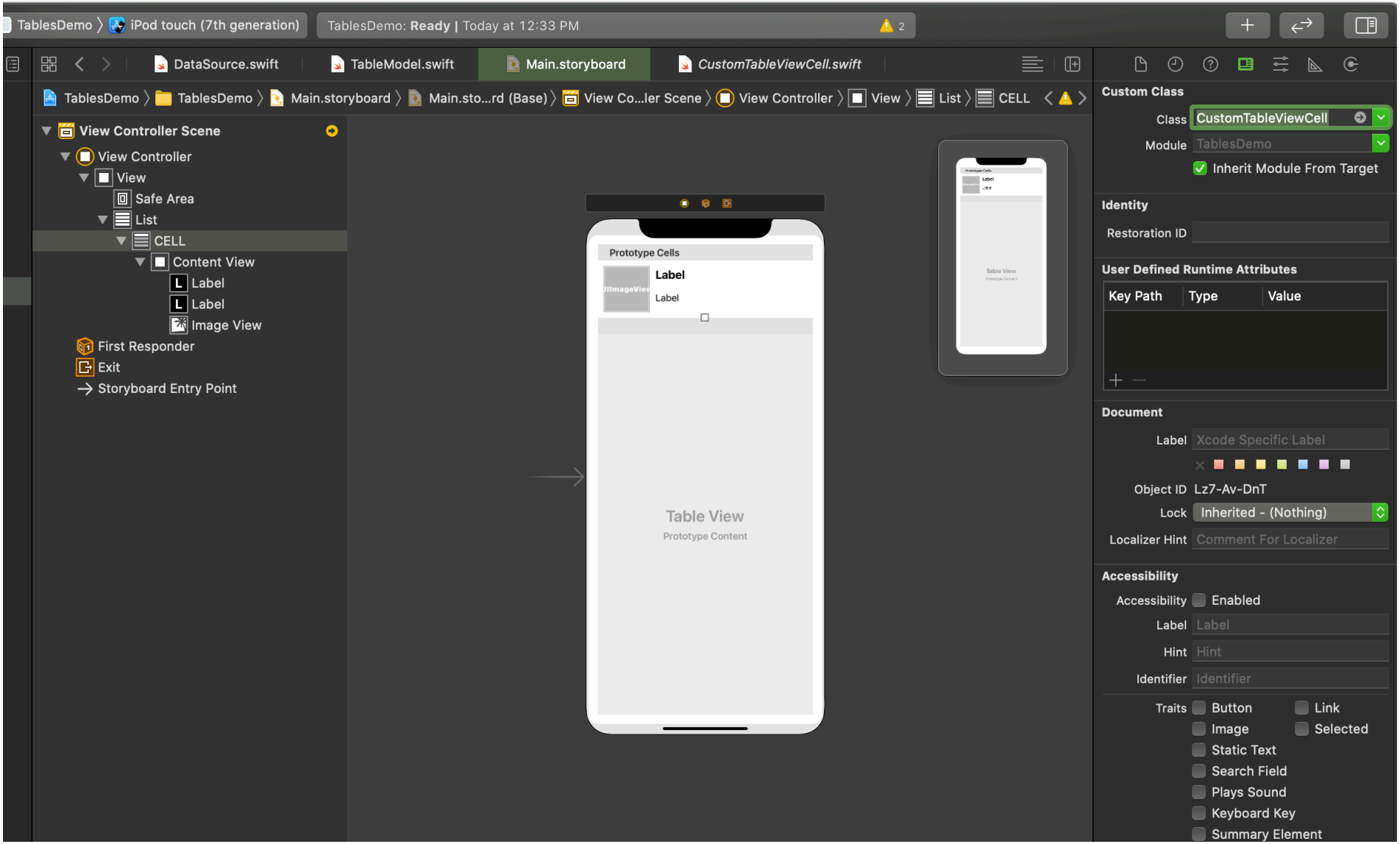
11. Set the subclass to *UITableViewCell* give the class name as *CustomTableViewCell*.



12. Click next.

13. Save the file into your project folder.
14. Switch to Main.storybooard.
15. Select your custom cell.

16. From the identity inspector set the identity of this cell to be your newly created class.



17. Create IBOutlet connections for the image view and labels in your newly created class. Your completed class should look like this:

```
class CustomTableViewCell: UITableViewCell {

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    //MARK: - IBOutlets
    @IBOutlet weak var icon: UIImageView!
    @IBOutlet weak var iconTitle: UILabel!
    @IBOutlet weak var iconDescription: UILabel!

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)

        // Configure the view for the selected state
    }
}
```

18. Switch to the ViewController.swift file.
19. In the main ViewController class declare a variable called listData of type TableModel. Create the object.

```
var listData : TableModel = TableModel()
```

20.

9
Chapter 9: Introduction to SwiftUI

In this chapter the reader will get familiar with the different options available to developers when it comes to implementing UI with SwiftUI.

Exercise 9.1: Explore SwiftUI

Prerequisites

Should be familiar with Swift programming

Scope of Work

Learn how to create and use SwiftUI in your project

Tools Required

Xcode 14.x

Playground

Outcome of the exercise

You should be comfortable with SwiftUI.

Steps

1. Open Xcode
2. Create a new iOS Project. Call it SwiftUIDemo.
3. Select the ContentView.swift file.
4. Add the following property to the struct ContentView.

```
//MARK: - Gradient
    let someshade = Gradient(colors: [Color.red,
                                      Color.orange,
                                      Color.yellow,
                                      Color.white,
                                      Color.blue,
                                      Color.gray])
```

5. Create a label by invoking the Text initializer.

```
 //MARK: - Displaying text
var body: some View {
    Text("Hello, World! \nThis is the first app.")
}
```

6. Now we will add some modifiers to the Text. All the modifiers will be added one after the other. Make sure the previews are loading as you add the modifiers to see if the desired effect is achieved. Start off by adding the the background modifer after the init call.

```
//Background
.background(Color.green)
```

7. Add the Padding modifier.

```
//Padding
.padding(10)
```

8. Add font modification.

```
//Font
.font(Font.custom("Marker Felt", size: 32.0))
```

9. Add line count.

```
//Line count
.lineLimit(4)
```

10. Similarly add the following modifiers.

```
//text alignment
.multilineTextAlignment(TextAlignment.leading)
```

```
//Foreground or text color
.foregroundColor(Color.init(red: 0.7, green: 0.3, blue: 0.1))
```

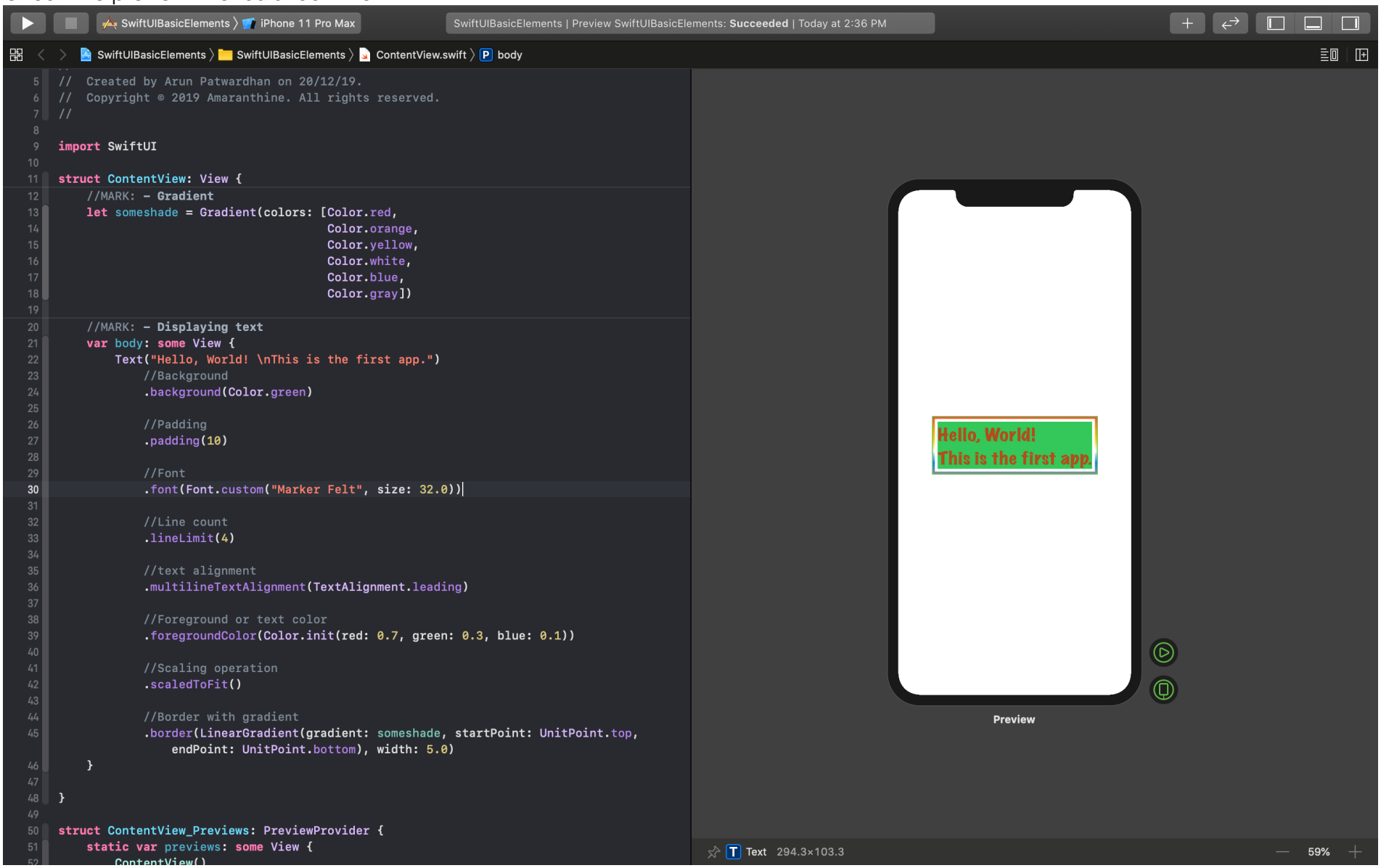
```
//Scaling operation
.scaledToFit()
```

```
//Border with gradient
.border(LinearGradient(gradient: someshade, startPoint: UnitPoint.top, endPoint: UnitPoint.bottom),
width: 5.0)
```

11. The completed code should look like:

```
//MARK: - Displaying text
    var body: some View {
        Text("Hello, World! \nThis is the first app.")
            //Background
            .background(Color.green)

            //Padding
            .padding(10)

            //Font
            .font(Font.custom("Marker Felt", size: 32.0))

            //Line count
            .lineLimit(4)

            //text alignment
            .multilineTextAlignment(TextAlignment.leading)

            //Foreground or text color
            .foregroundColor(Color.init(red: 0.7, green: 0.3, blue: 0.1))

            //Scaling operation
            .scaledToFit()

            //Border with gradient
            .border(LinearGradient(gradient: someshade, startPoint: UnitPoint.top, endPoint:
UnitPoint.bottom), width: 5.0)
    }
```

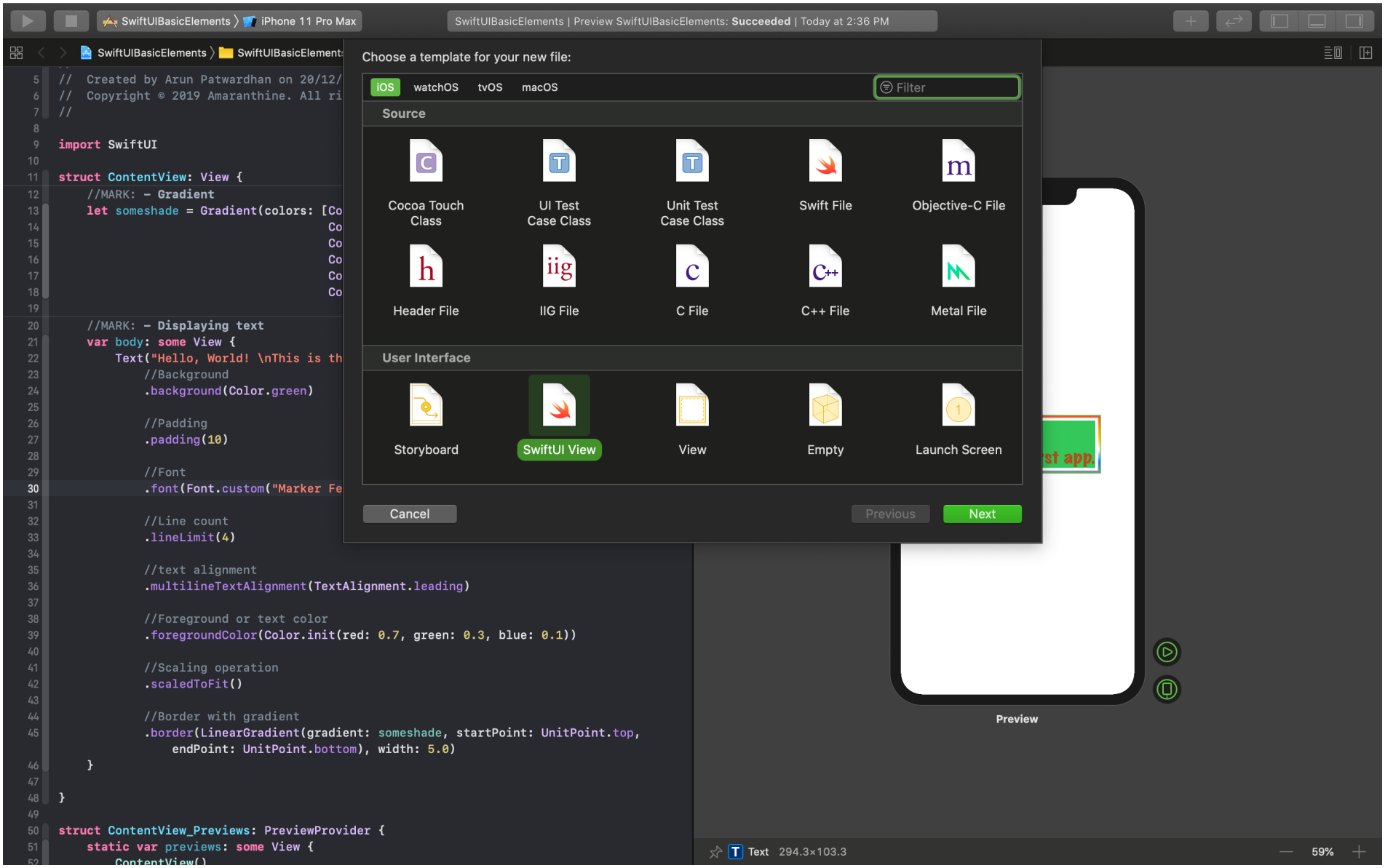
12. Check the preview. It should look like:



13. If you can see the preview make sure you have selected show canvas.

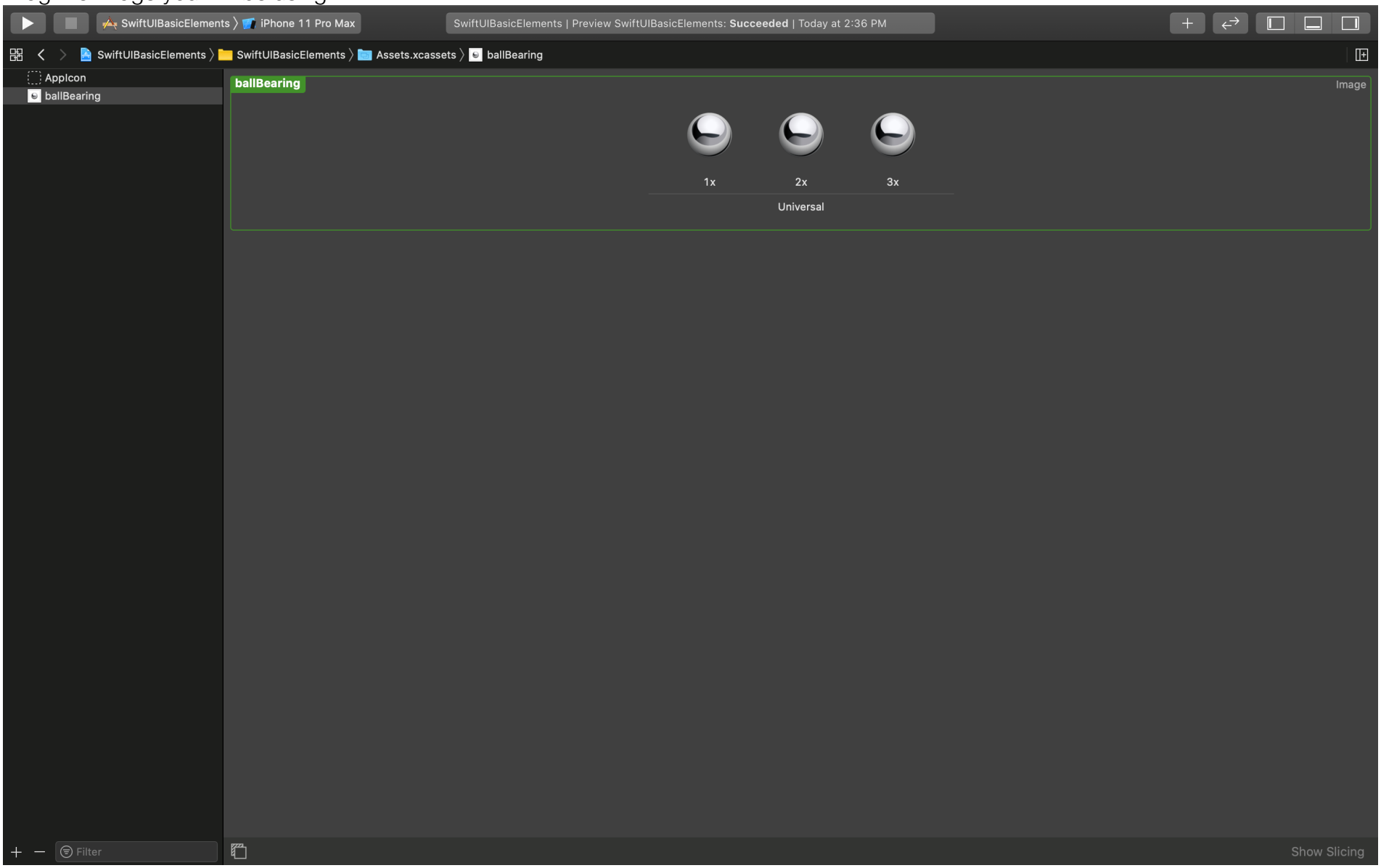
14. Run the app see how it works
15. Now create a new SwiftUI view. Click on *File* > *New*.

16. Select the SwiftUI from the user interface section.

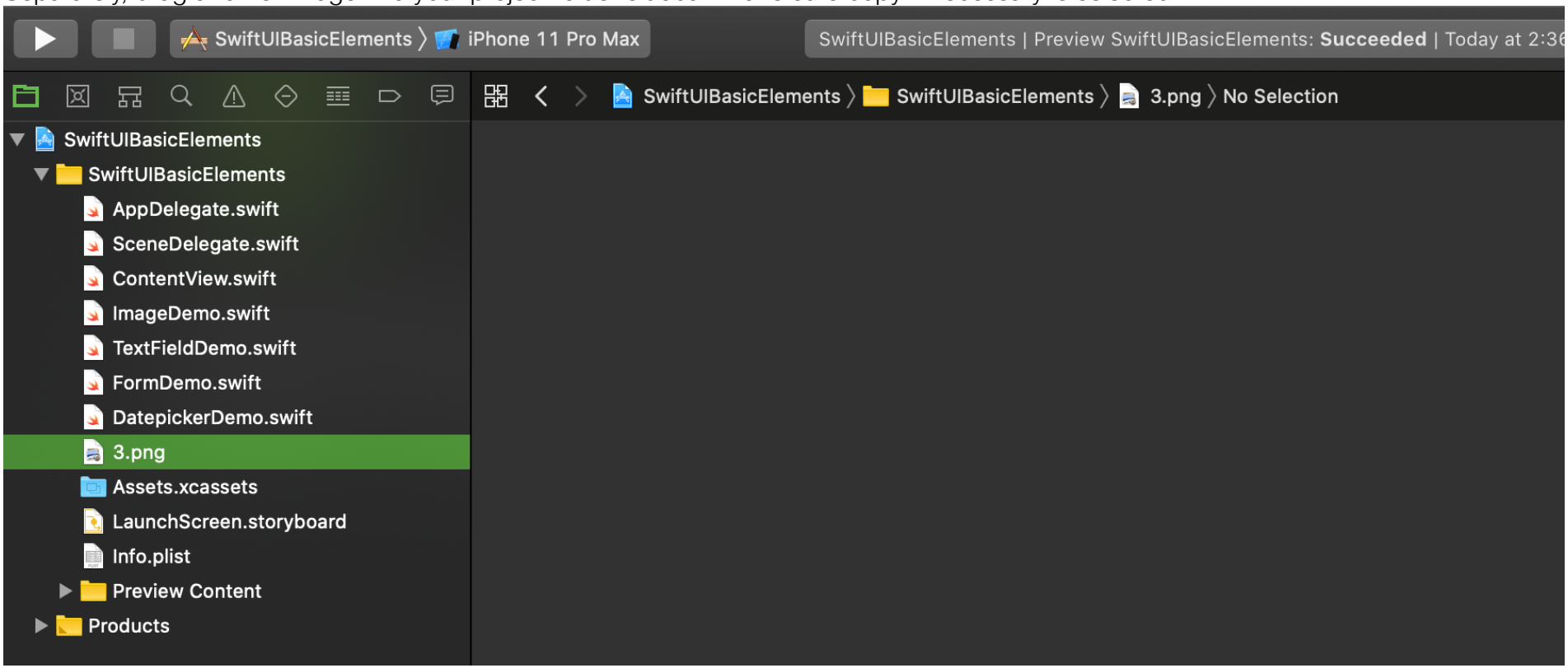


17. Click next.
18. Save the file as ImageDemo or any name you like.
19. Select the Assets.Catalogue

20. Drag the image you will be using in it.



21. Separately, drag another image into your project folder sidebar. Make sure copy if necessary is selected.



22. Select the newly created SwiftUI file. Mine is called ImageDemo.swift

23. In the struct for ImageDemo add a function to create an image.

```
/**
     This function creates an Image from an UIImage object
     - important: This function validates to see if the image can be created
     - note: This process does not expect the image to be part of the assets catalogue
     - returns: String.
     - requires: iOS 13 or later
     - Since: iOS 13
     - author: Arun Patwardhan
     - copyright: Copyright (c) Amaranthine 2015
     - version: 1.0
     - Date: 22nd December 2019
     */
    func createImage() -> Image
    {
        guard let img = UIImage(named: "3.png")
            else
        {
            fatalError("Unable to load image")
        }

        return Image(uiImage: img)
    }
```

24. In the body property of the struct add the following lines of code.

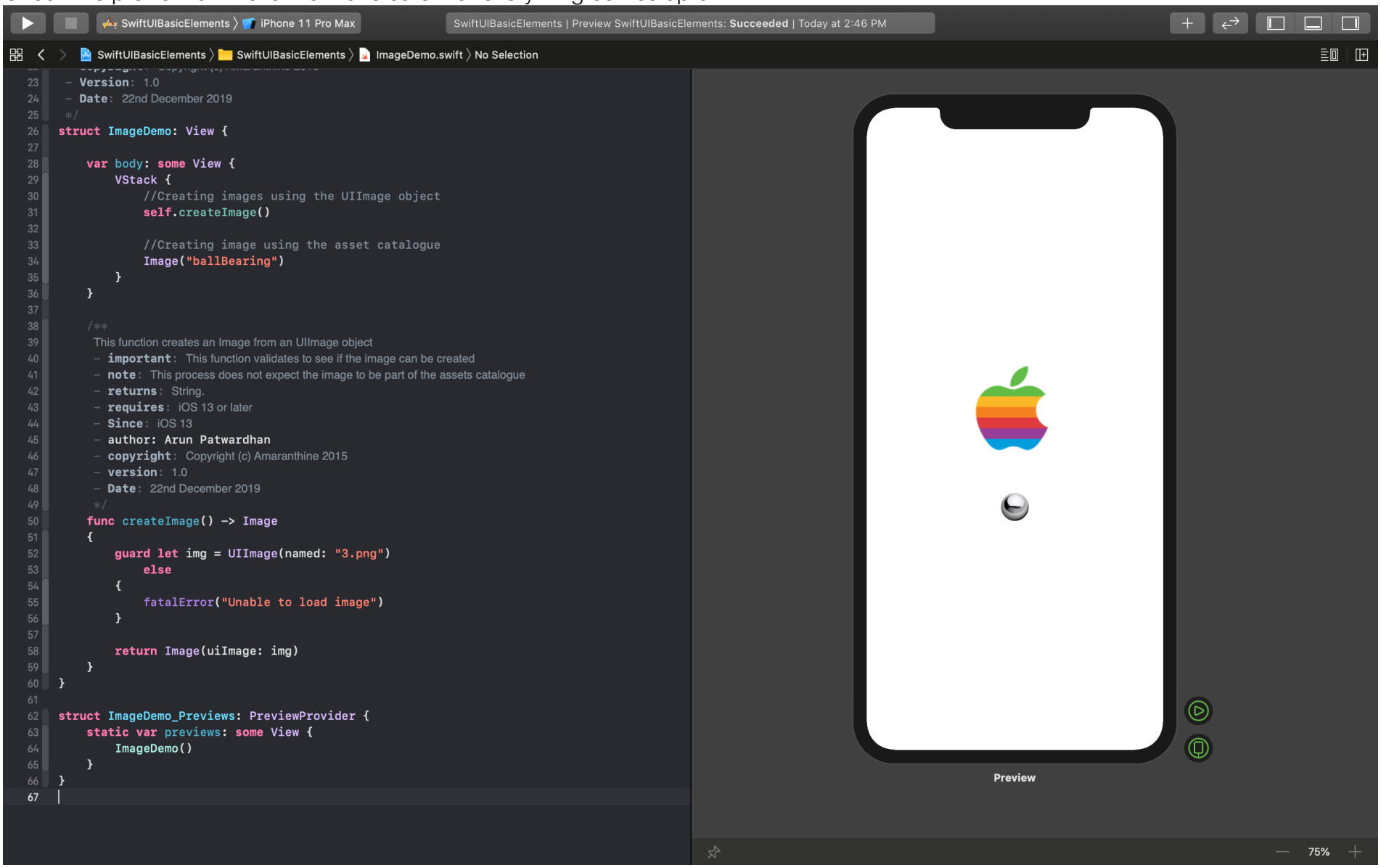
```
VStack {
    //Creating images using the UIImage object
    self.createImage()

    //Creating image using the asset catalogue
    Image("ballBearing")
}
```

In this piece we are creating 2 images, one using UIImage, through the function call, the other by directly invoking the Image init.

25. Your completed code should look like.

```
struct ImageDemo: View {

    var body: some View {
        VStack {
            //Creating images using the UIImage object
            self.createImage()

            //Creating image using the asset catalogue
            Image("ballBearing")
        }
    }

    func createImage() -> Image
    {
        guard let img = UIImage(named: "3.png")
            else
        {
            fatalError("Unable to load image")
        }

        return Image(uiImage: img)
    }
}
```

26. Check the preview on the left to make sure that everything comes up ok.

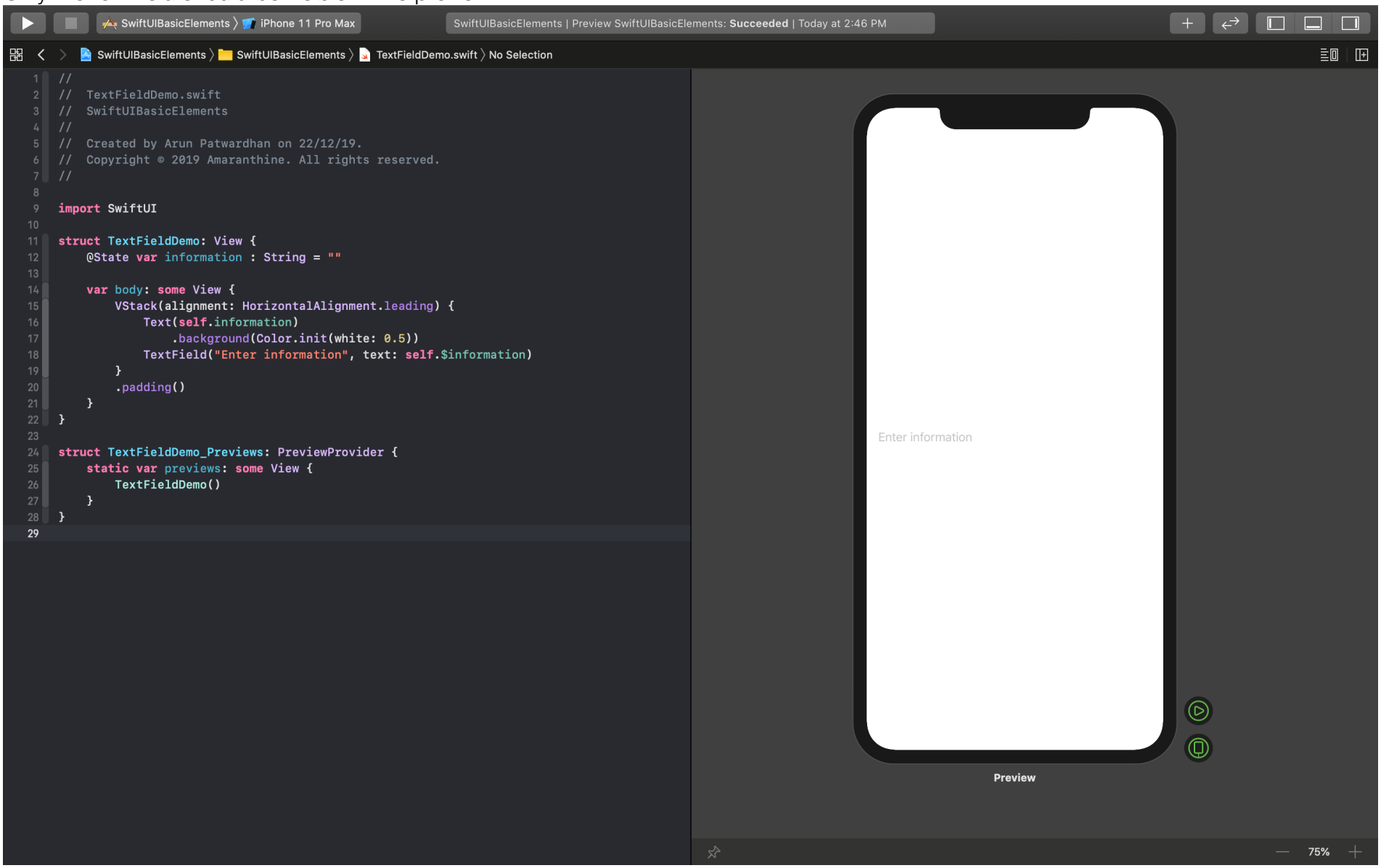


27. Similarly create a new SwiftUI file for a TextField. Call it TextFieldDemo.
28. Implement the view struct as shown below.

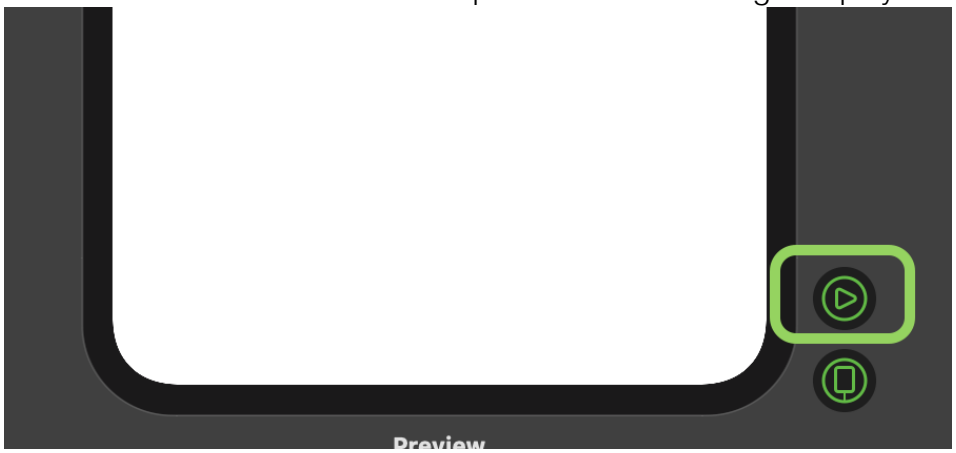
```
struct TextFieldDemo: View {
    @State var information : String = ""

    var body: some View {
        VStack(alignment: HorizontalAlignment.leading) {
            Text(self.information)
                .background(Color.init(white: 0.5))
            TextField("Enter information", text: self.$information)
        }
        .padding()
    }
}
```

The @State property wrapper allows the SwiftUI views to modify the property itself.

29. Only the text field should be visible in the preview.

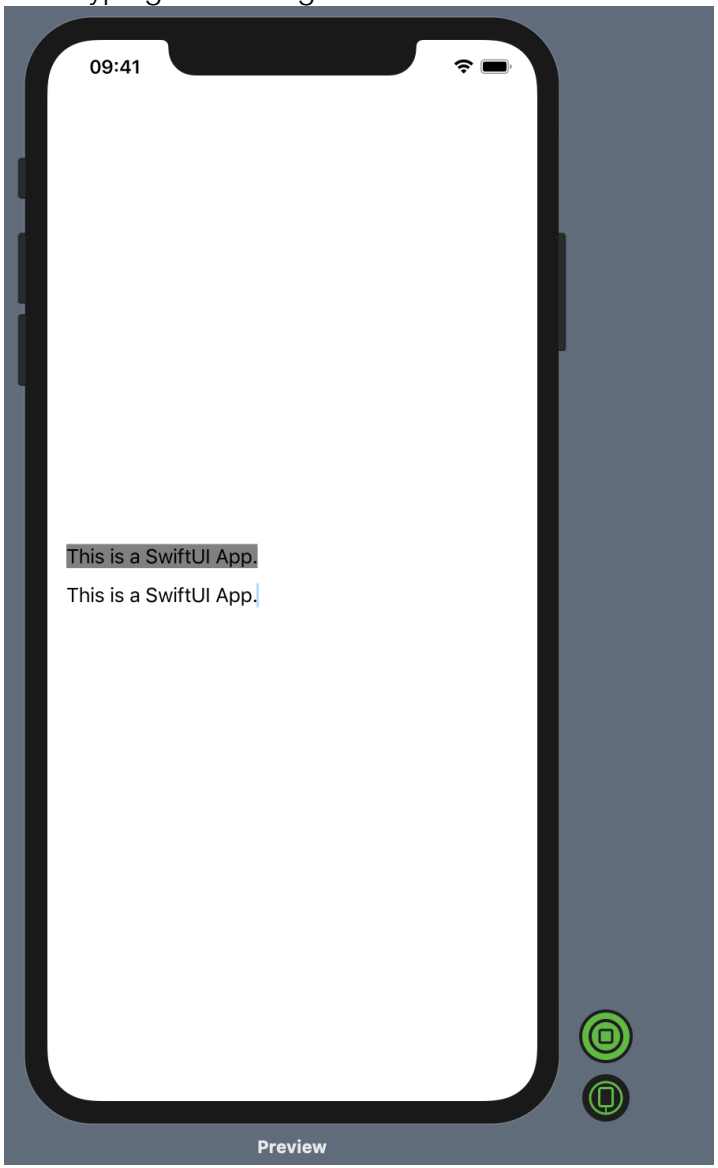


30. There are 2 ways to test if it works. One is to run the app and bring up the simulator. The other is to start the execution within the preview itself. To start the execution in the preview click on the green play button.



The background color of the preview window changes and you get a spinner next to the word “Preview’ indicating the the preview is getting ready.

31. Start typing something in the textfield. It should immediately appear on the label.



32. Once you are done click on the stop button.

33. Last we will test the date picker. Create a new SwiftUI file called DatePickerDemo.swift.
34. Update the struct as shown below.

```
struct DatepickerDemo: View {
    @State var dob : Date = Date()

    var body: some View {
        DatePicker(selection: self.$dob, displayedComponents: DatePickerComponents.date, label:
{ Text("Select Date of Birth") })
            .foregroundColor(Color.orange)
    }
}
```

35. Check the preview. Run the execution in the preview to see if it works.
36. To check the different screens in the simulator simply update the line, SceneDelegate.swift file, to whichever view you want.

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions
    // Use this method to optionally configure and attach the UIWindow `window` to the provided UIWindowScene `scene`.
    // If using a storyboard, the `window` property will automatically be initialized and attached to the scene.
    // This delegate does not imply the connecting scene or session are new (see `application:configurationForConnecting

    // Create the SwiftUI view that provides the window contents.
    let contentView = DatepickerDemo()

    // Use a UIHostingController as window root view controller.
    if let windowScene = scene as? UIWindowScene {
        let window = UIWindow(windowScene: windowScene)
        window.rootViewController = UIHostingController(rootView: contentView)
        self.window = window
        window.makeKeyAndVisible()
    }
}
```

A
Appendix A: Xcode

Common tasks that need to be performed with Xcode are found here.

202327611082322384

A1: Creating playground pages

Steps

1. Open Xcode.
2. File > New > Playground. This will create a playground file.
3. Click File > New > Playground Page. To create a new playground page. The first playground page and the playground itself will share the same code space.
4. Create subsequent playground pages by repeating step number 3.
5. Add the following code to incorporate previous and next hyperlinks to navigate all the pages.

```
//: [Previous](@previous)
```

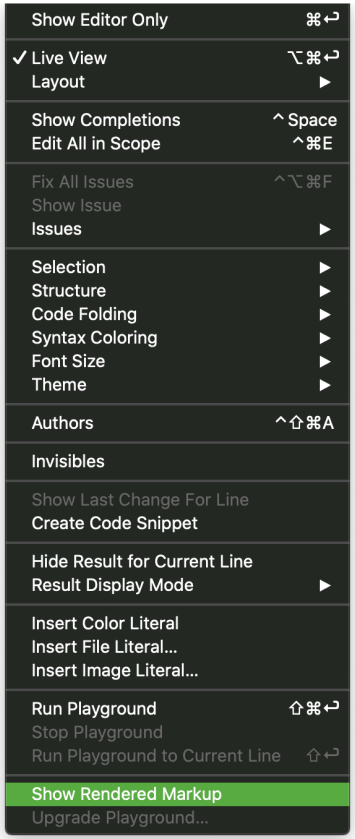
```
//: [Next](@next)
```

6. To get the rendered version of the formatted comments read article in appendix A on showing rendered markup.

A2: Rendering formatted comments in playground

Steps

1. Open Xcode.
2. From the menu bar, click on Editor > Show rendered markup.



3. Toggle this option to see the raw markup.

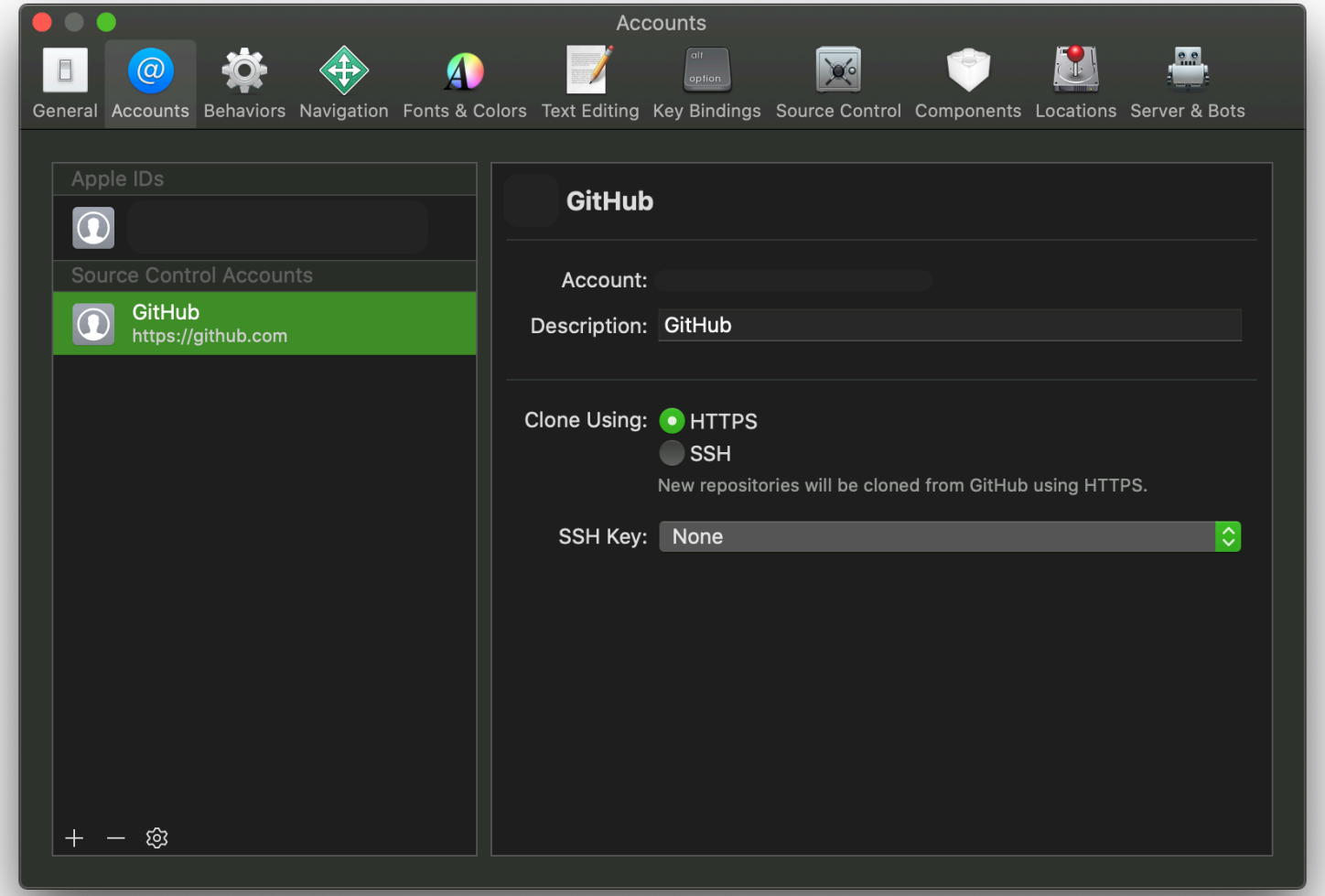
A3: Configuring Github on Xcode

You will need to ensure that you have a GitHub account available with you.

Steps

1. Open Xcode.
2. Click Xcode > Preferences.
3. Select Accounts.
4. Click on ‘+’ and select Github from the dropdown.
5. Login with your credentials.

6. You have now successfully configured GitHub on your computer.



A4: Erasing persistent stores on the simulator

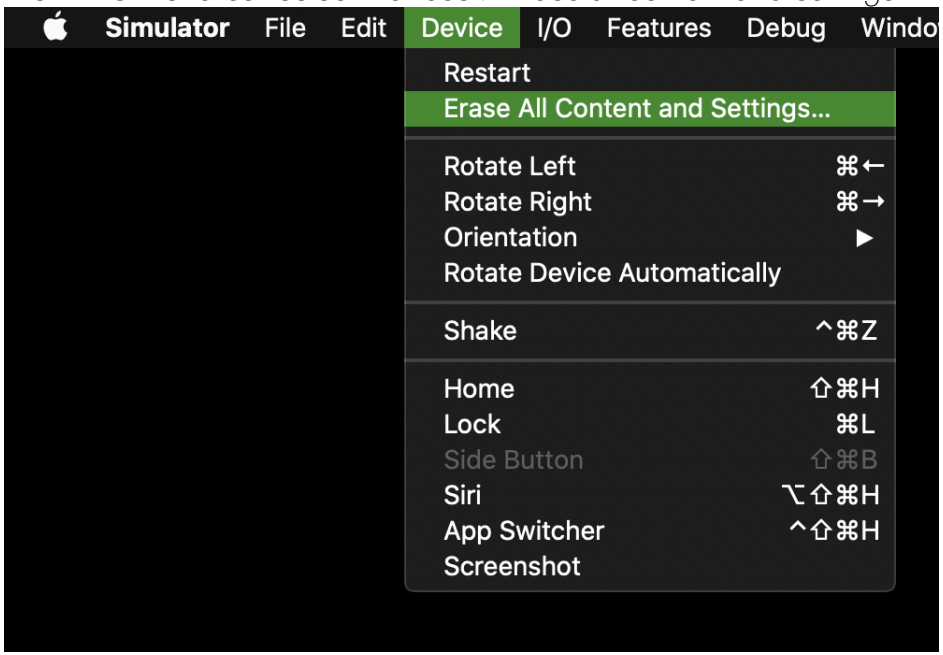
This is something you would have to do often when you are testing different persistent store solutions on the simulator.

Warning

This will wipe out all the apps and their content on the simulator. You will have to reload the apps all over again.

Steps

1. Open Simulator.
2. From the menu bar select *Devices > Erase all content and settings.*



3. Wait for the simulator to restart.

B
Appendix B: Third party tools

Common tasks that need to be performed with Xcode are found here.

B1: Installing & using Cocoapods

Steps

1. Open Terminal.
2. Run the following command:
   `sudo gem install Cocoapods`
3. Wait for Cocoapods to install.
4. To use them we first need an existing Xcode project. Take any project that you have or create a new one just to test cocoapods.
5. Quit Xcode.
6. Back in terminal, navigate to the folder where your Xcode project is. So supposing you created your project on the desktop then navigate to:
   `cd ~/Desktop/ProjectName/`
7. Your xcodeproj file would be located in here.
8. Run the command to initialise a pod.
   pod init
9. Change the generated PodFile by listing out the pods you wish to use.
10. Then run the command to install the pods.
    pod install[4]
11. Wait for the pods to install.
12. Once completed you are ready to start using them in your code.[5]
13. Open the xcworkspace file. Do not open the xcodeproj file.

---

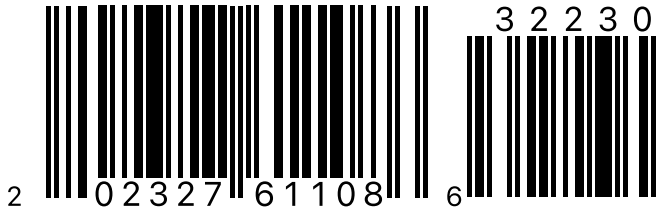
[4] if you are running this on an Apple Silicon computer run the command **`sudo arch -x86_64 gem install ffi`** to install ruby. Then run the same command with the x86 architecture: **`arch -x86_64 pod install`**

[5] You will have to check for the appropriate version numbers.

Contact

For any queries, corrections at the following

Website	www.amaranthine.in/contact-us
Email	arun@amaranthine.co.in
Facebook	https://www.facebook.com/amaranthinelabs
LinkedIn	https://www.linkedin.com/company/9410642?trk=tyah&trkInfo=clickedVertical%3Acompany%2CclickedEntityId%3A9410642%2Cidx%3A1-1-1%2CtarId%3A1438164427255%2Ctas%3Aamaranthine
Twitter	https://twitter.com/amaranthineTech
Youtube	https://www.youtube.com/channel/UC127UHd8V7bxPQYnd9QrN8w
Flipboard	https://flipboard.com/@AmaranthineTech
Instagram	https://www.instagram.com/amaranthinetech/



Arun Patwardhan
arun@amaranthine.co.in
www.amaranthine.in