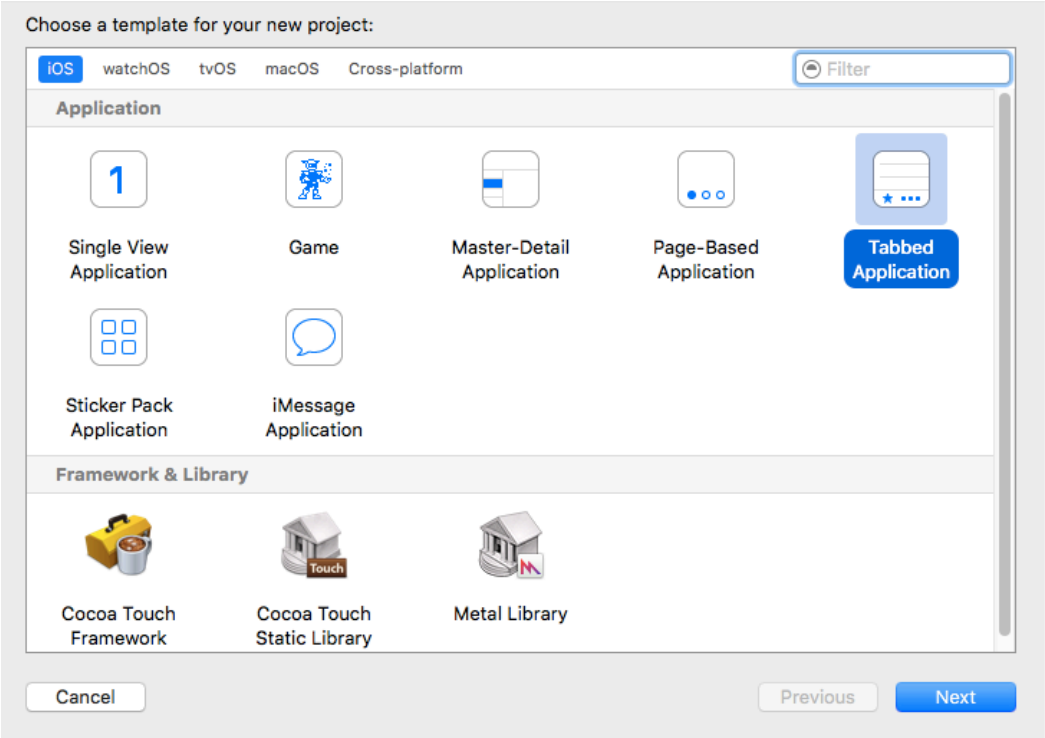
Practical 03: Tab bar and Picker Controller

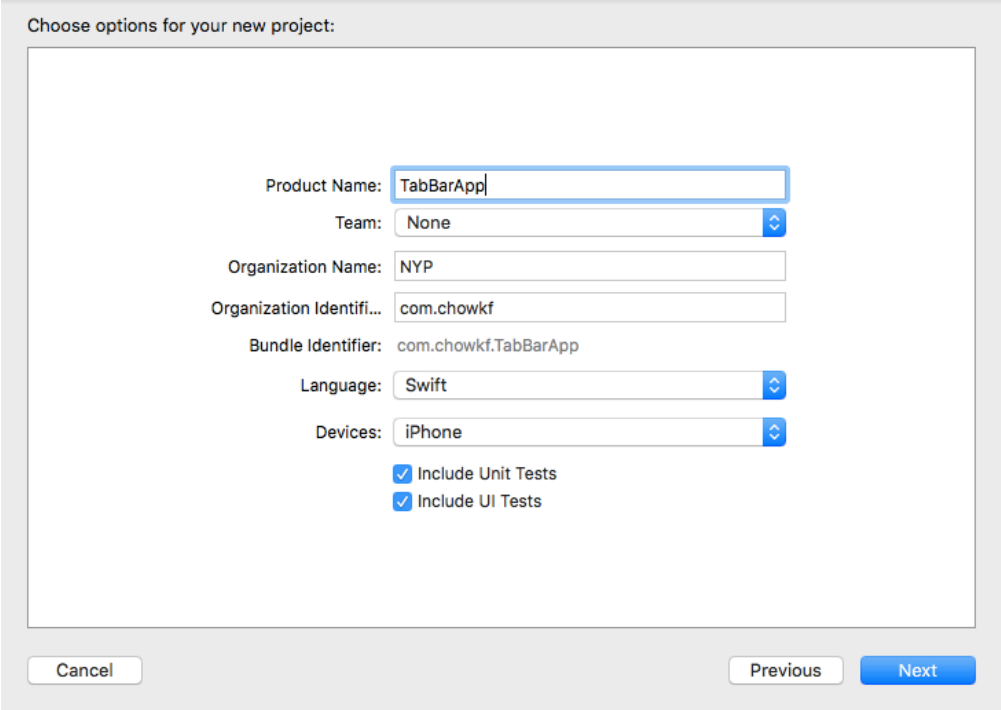
There are many ways to present your view controllers. In this practical, we will be using UITabBarController to swap between view controllers. UITabBarController keeps an array of UIViewControllers and maintain a tar bar on the bottom of the screen. In this lab, we will learn how to create a Tab bar application with 3 tabs. We will also introduce delegation, a design pattern, commonly used in Cocoa. We will used a Picker View to demonstrate the used of delegation.

Section 1: Tab Bar Controller

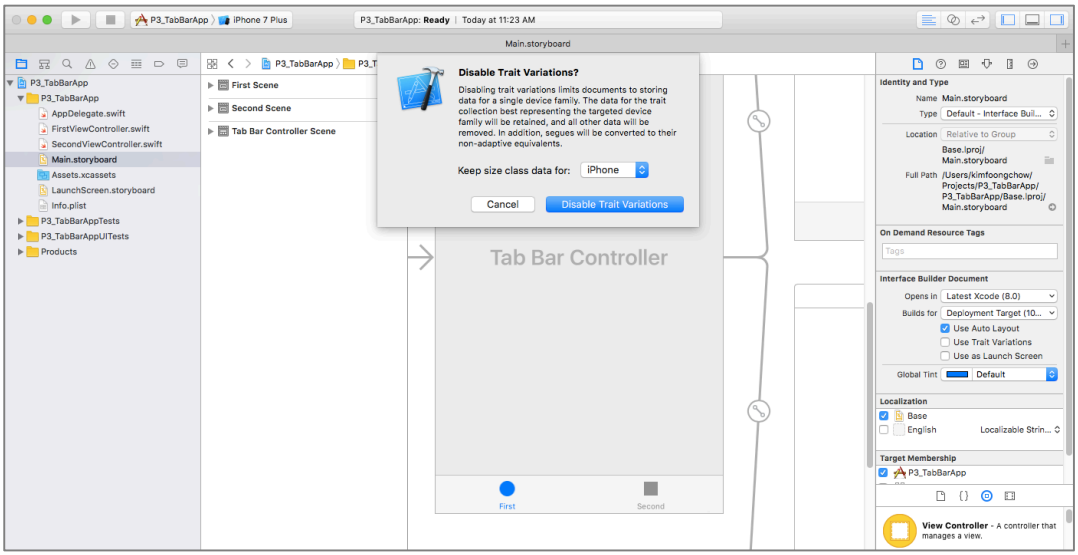
1. Using Xcode, choose a **Tabbed Application** template for this project.



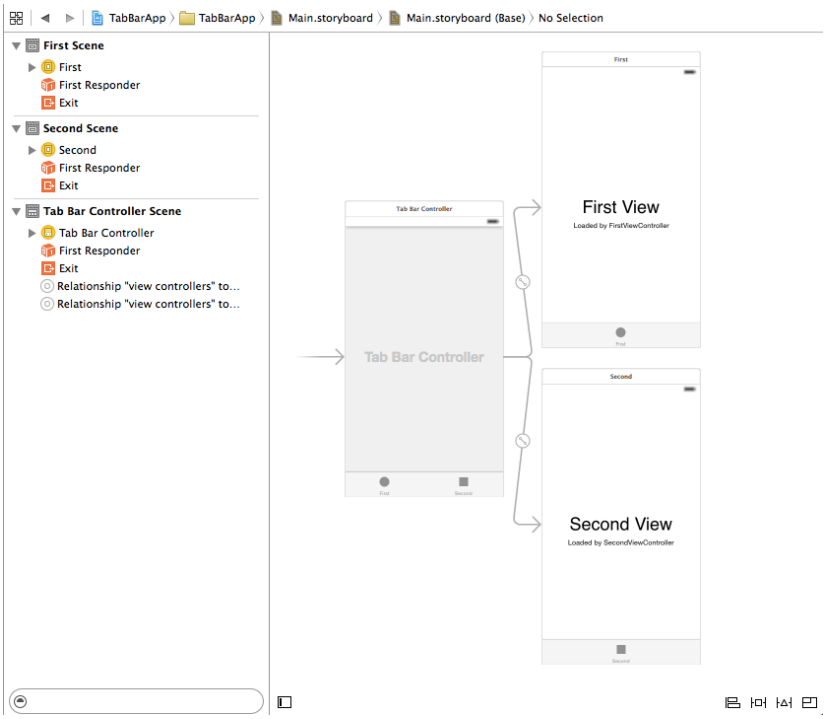
2. Next, configure this project and name it as **TabBarApp** as shown below and click “Next”, and then click “Create”.



3. Click on the **Main.Storyboard** and at the Utility navigator pane, under the **File Inspector**, make sure the **Use Trait Variations** is unchecked. You will be prompt to **Disable Trait Variations** and Keep size class data for **iPhone**.



4. Under the **Project Navigator** pane, you will notice that multiple files have been created for you. By Default, Xcode creates 2 View Controllers for us, namely **FirstViewController** and **SecondViewController**.

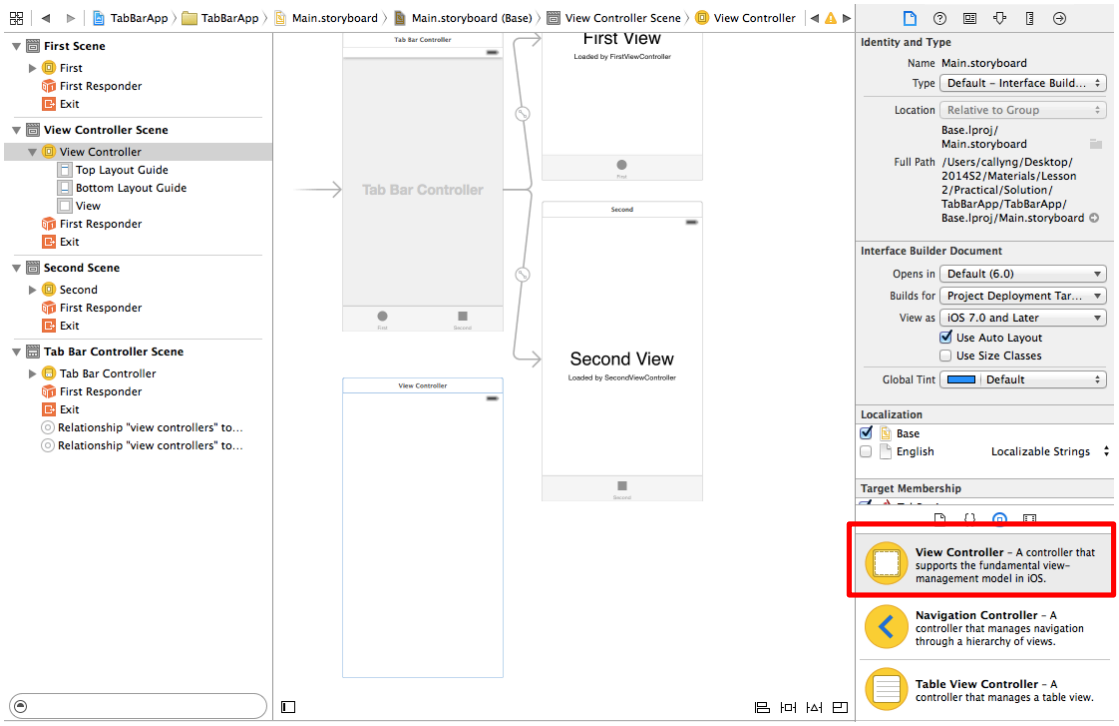


5. Build and run the application on the Simulator and you will see two tabs have already been created for you.

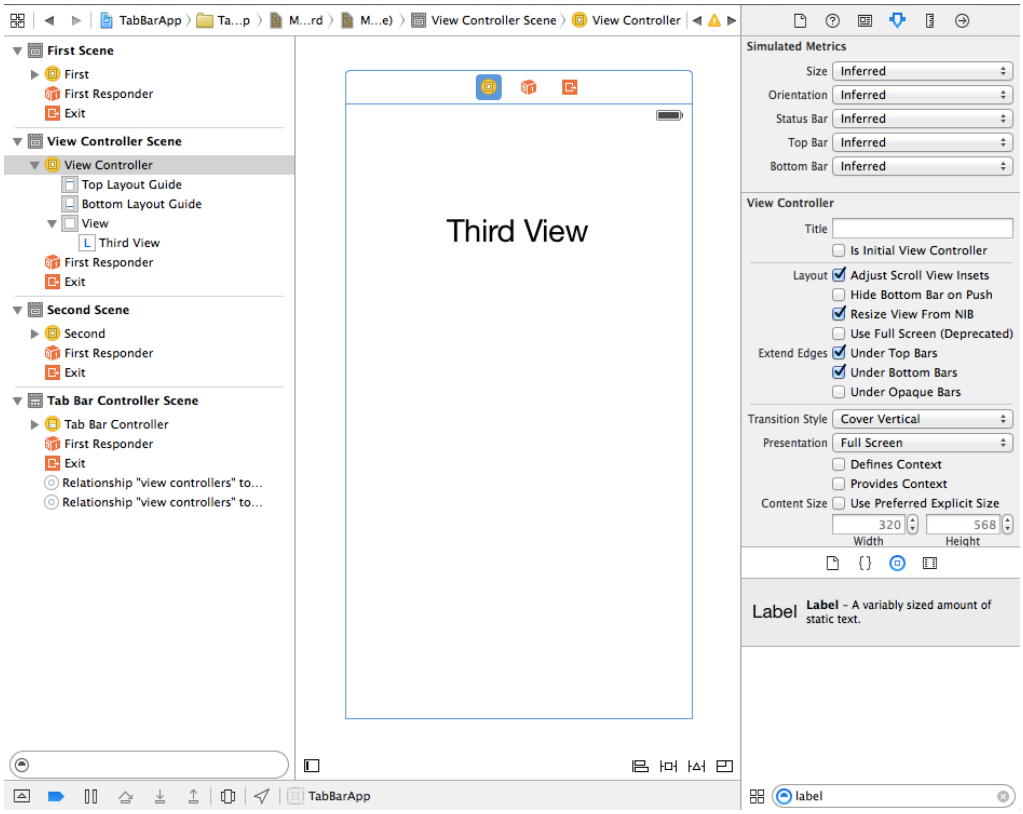
Adding tab bar item

In this part of the lab, we will learn how we can add in a new tab bar item to the existing tab bar controller.

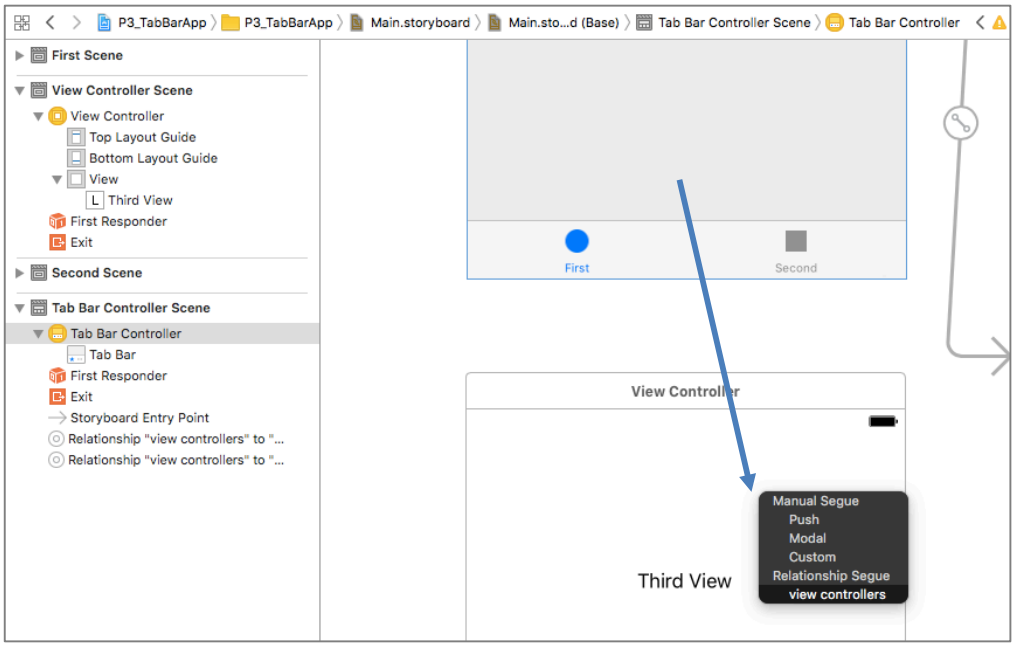
6. Click on the **Main.Storyboard**, drag and drop a **View Controller** to the storyboard editor.



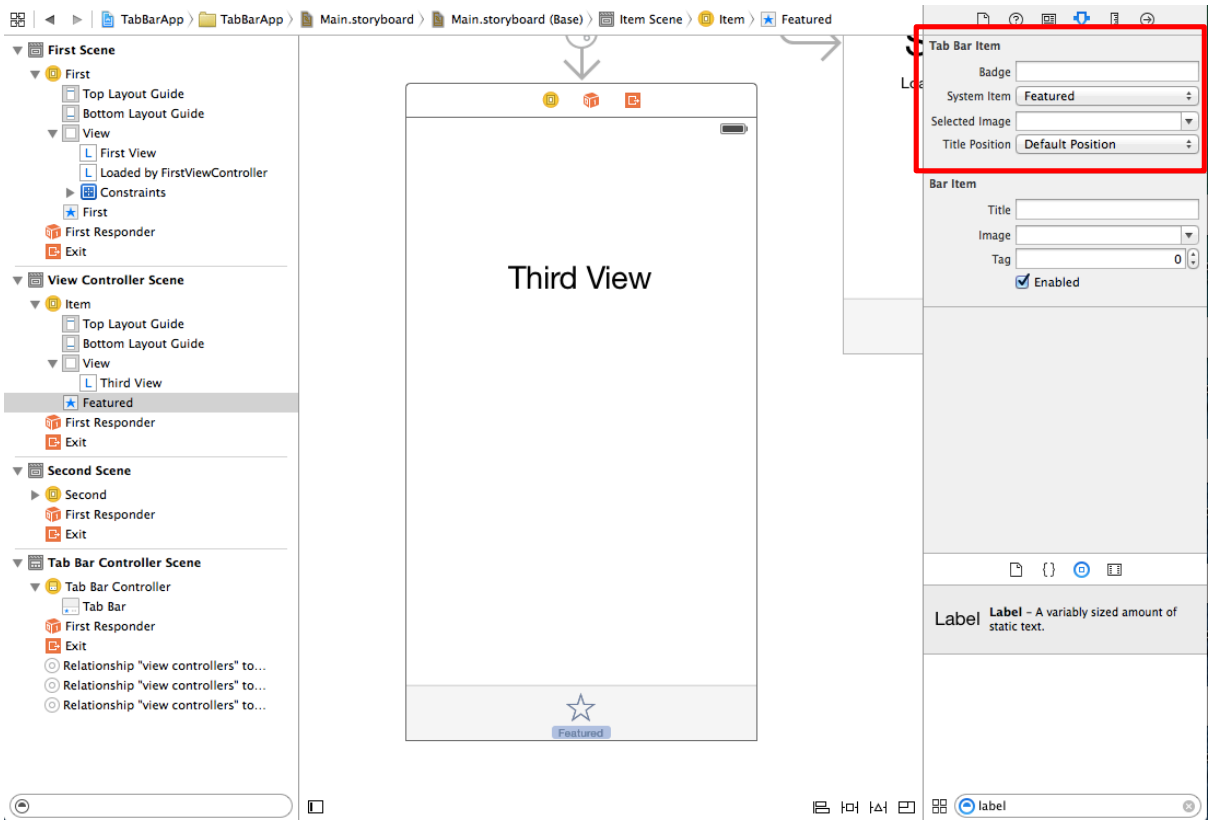
7. Drag and drop a **label** the newly created View Controller as shown:



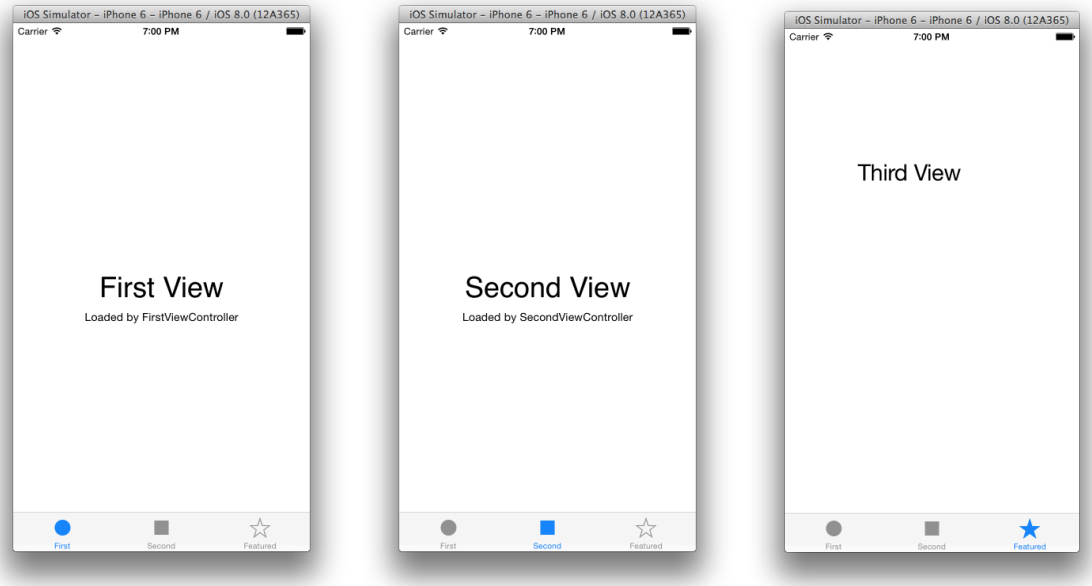
8. Next, connect the newly created view controller to our Tab Bar Controller. **Ctrl-drag** from the Tab Bar Controller to the new View Controller and they will form a relationship by **selected Relationship Segue->view controllers**.



9. Let's give a title and an image to the new tar bar item. Select the Third View and click on the tab bar icon. Under the **Attribute inspector** in Utility pane, select **Featured** at the System Item.



10. Build and run the application on the Simulator and you will see the third tab that you have added.

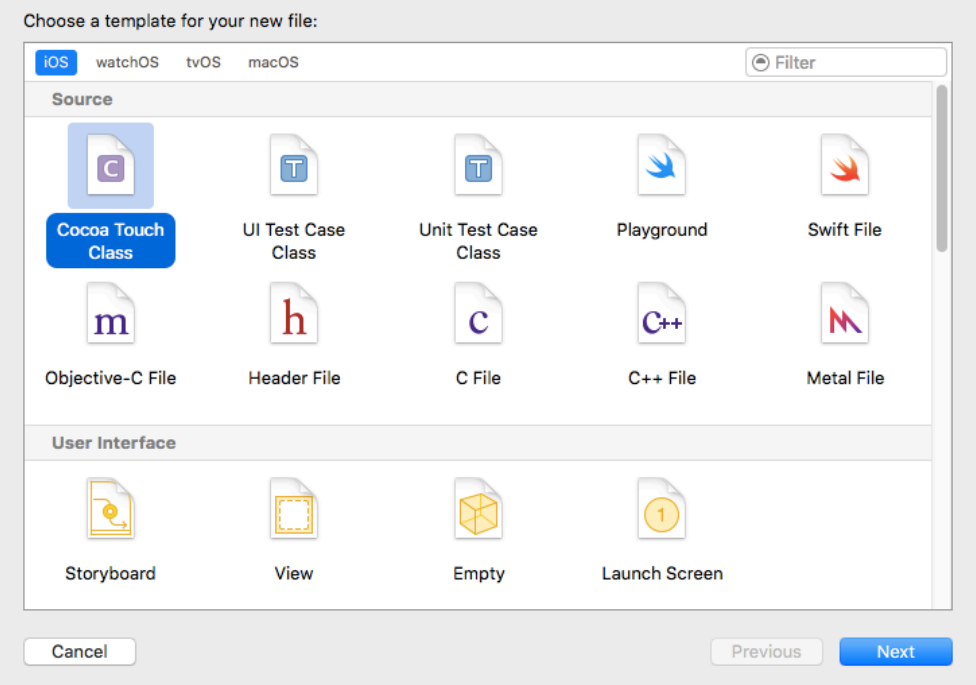


Section 2: Delegation and Picker View

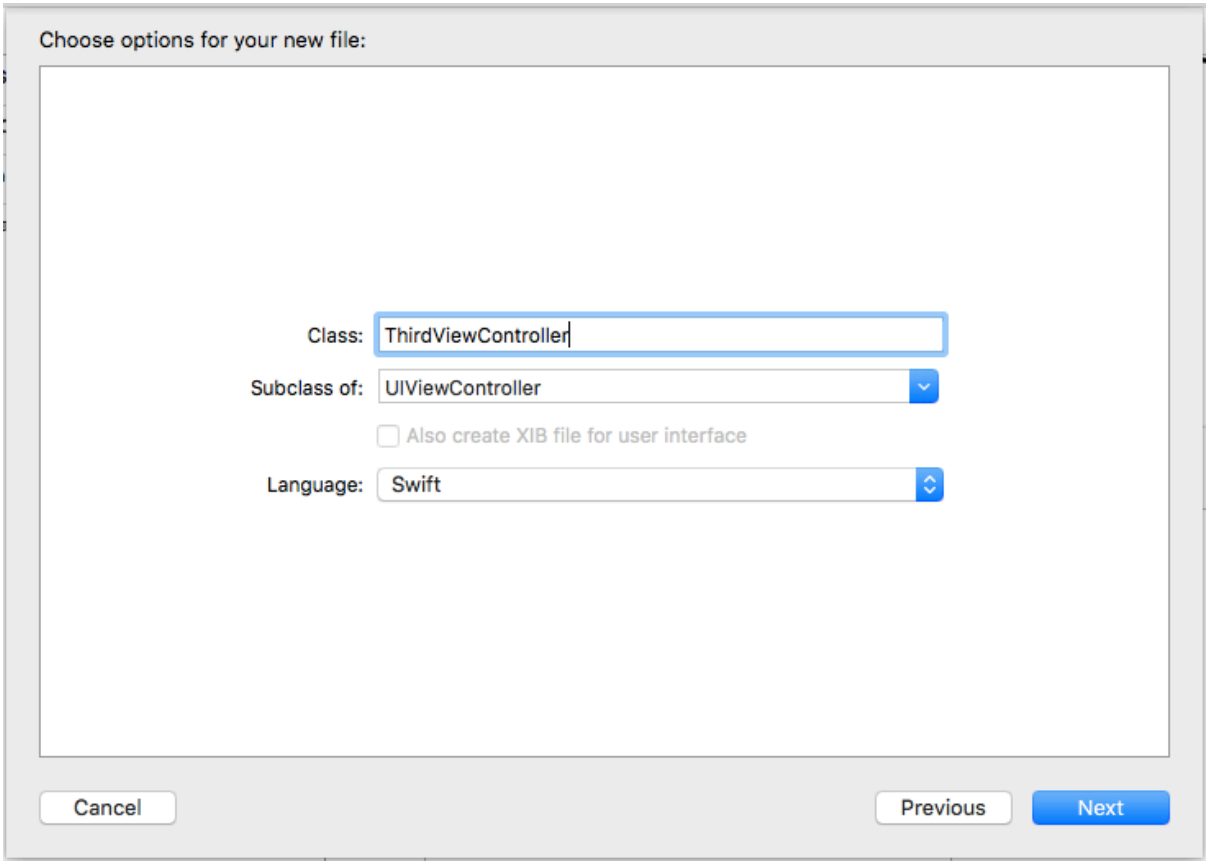
We will introduce to you delegation, a design pattern. It is used to respond to an event or request for information from a host object that contains a pointer reference to the delegate.

*In this lab, we will make use of UIPickerViewDelegate to render what to be displayed on the Picker View. The delegate of an instance of UIPickerView has to conform to the UIPickerViewDelegate protocol and implement all the **required** methods of that protocol. Picker view is used when you want to display a list of values for the user to choose from. We will be creating an array to hold the values we want to display in the picker.*

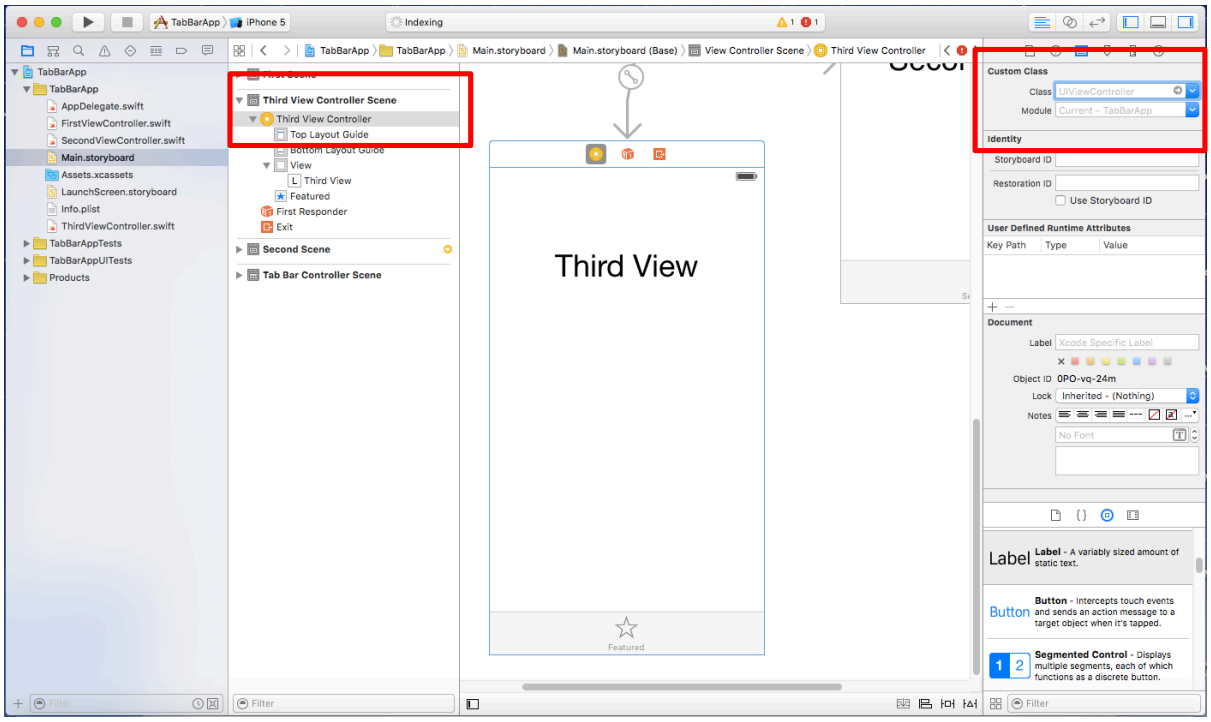
11. Under the **TabBarApp** folder of your project, add a new file and select **Cocoa Touch Class**. Click Next.



12. For the **Subclass** of, select **UIViewController**. Name the file as **ThirdViewController** and click **Next**, then click **Create**.



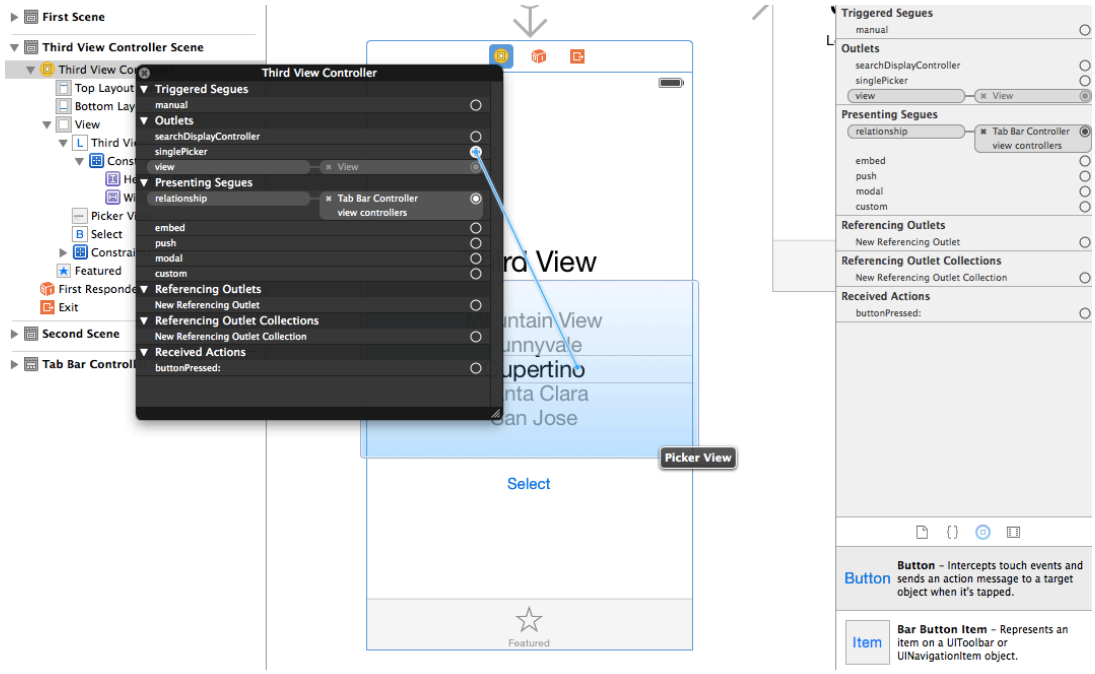
13. Next, we need to link the newly created View Controller with **ThirdViewController**. Select the Main.Storyboard, at the Outline View, select the View Controller Scene > View Controller. Select the **ThirdViewController** from the identity inspector.



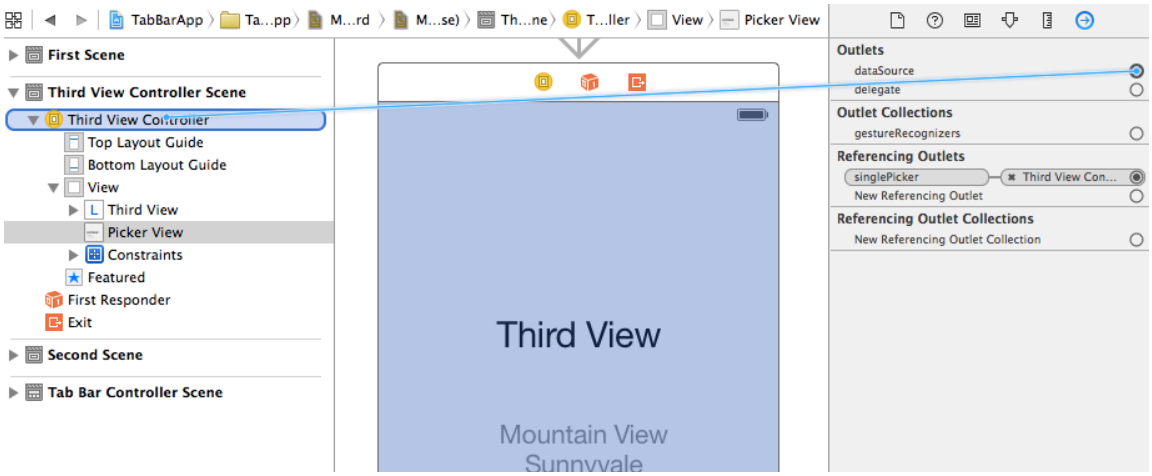
14. In the project navigator, select **ThirdViewController.swift** and open the file in the editor. Add / modify the following highlighted codes:

```
import UIKit

class ThirdViewController: UIViewController,
    UIPickerViewDelegate, UIPickerViewDataSource
{
    @IBOutlet weak var singlePicker : UIPickerView!

    // Declare a string array
    var pickerData : [String]

    // Declare an event that will be triggered when
    // a button is pressed.
    //
    // We will need to hook this up to the actual
    // button in the storyboard later.
    //
    @IBAction func buttonPressed(sender: AnyObject)
    {
    }

}
```

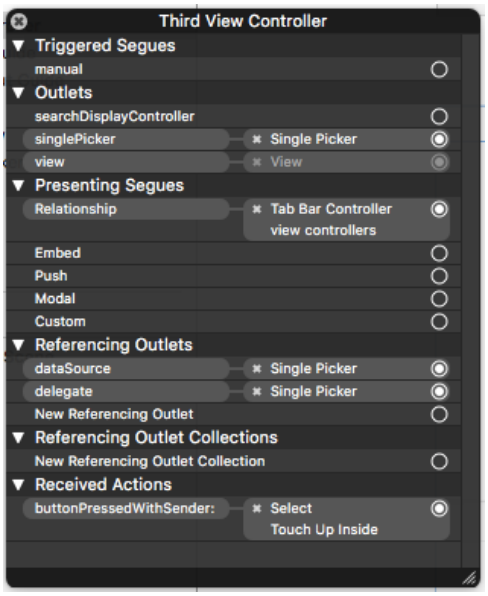
15. Before we start implementing the codes of the implementation file, drag and drop a **pickerview** and **button** from the library and connect the outlets and actions to the respective views on the View window.


First Scene
Third View Controller Scene
Second Scene
Tab Bar Controll
Top Layout
Bottom Lay
View
Picker Vi
Select
Featured
First Responde
Exit
Third View Controller
Triggered Segues
manual
Outlets
searchDisplayController
singlePicker
view
View
Presenting Segues
relationship
Tab Bar Controller
view controllers
embed
push
modal
custom
Referencing Outlets
New Referencing Outlet
Referencing Outlet Collections
New Referencing Outlet Collection
Received Actions
buttonPressed:
Picker View
Select
Featured
Button
Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Item
Bar Button Item – Represents an item on a UIToolbar or UINavigationItem object.

16. In the Inspector pane, ensure that the **Connections Inspector** is selected. Notice that the **dataSource** and **delegate** have yet to be connected to any outlet. Connect the outlet by clicking the small circle on the right side of the **dataSource**. Keep your mouse button pressed and drag to the **Third View Controller** located at the outline view. Repeat it for delegate.



17. Ensure that the Connections Inspector window for the **View** item in the **Third View Controller** window looks as shown:



18. When the ThirdViewController is initialized, we must make sure that our pickerData variable is initialized as well. To do so, we must override the init(code aDecoder: NSCoder) initializer.

```
    required init?(coder aDecoder: NSCoder) {
        // Initialize the pickerData string array with
        // some data.
        //
        self.pickerData = [
            "Strawberry",
            "Orange",
            "Mango",
            "Cherry",
            "Banana",
            "Coconut",
            "Kiwi"]
```

```
        super.init(coder: aDecoder)
    }
```

19. You will notice that you are getting warnings from the compiler. These warnings are telling you that you have yet to implement some methods that **UIPickerViewDataSource** protocol wants you to implement. So let's implement the **required** methods.

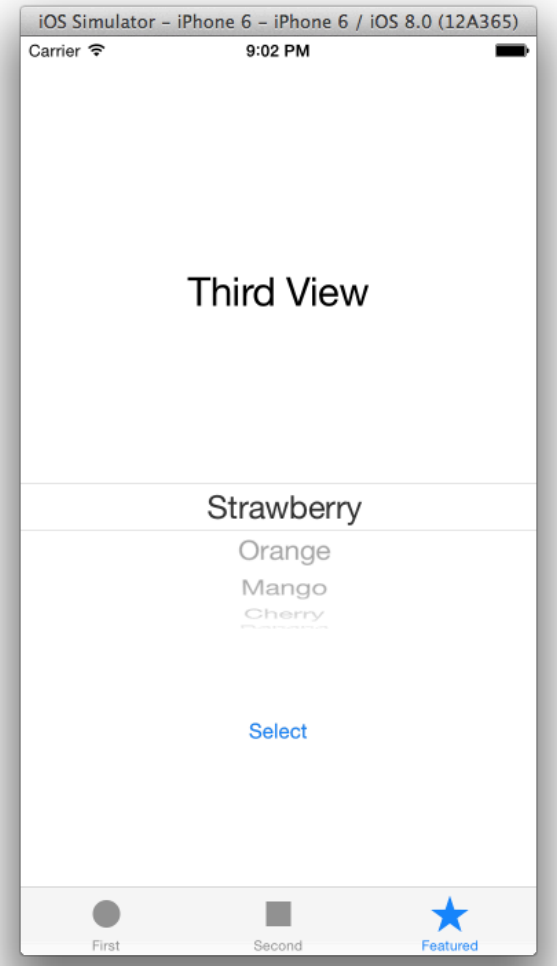
```
    // This function is one of the functions that a delegate for
    // the UIPickerDataSource should implement. It tells the
    // UIPicker how many components (vertical sections)
    // there are in the list to display.
    //
    func numberOfComponents(in pickerView: UIPickerView)
        -> Int
    {
        return 1
    }

    // This function is one of the functions that a delegate for
    // the UIPickerDataSource should implement. It tells the UIPicker
    // how many items in the list to display for a given component
    // (vertical section)
    //
    // Since we have only 1 section, this function will only be
    // called where the numberOfRowsInComponent == 1. We only
    // need to return the number of items in the pickerData.
    //
    func pickerView(_ pickerView: UIPickerView,
        numberOfRowsInComponent component: Int) -> Int {
        return pickerData.count;
    }
```

20. Next, the delegate of an instance of UIPickerView has to conform to the UIPickerViewDelegate protocol and implement all the **required** methods of the protocol. In UIPickerViewDeletegate, we are only interested in **pickerView(pickerView: titleForRow: forComponent:) -> String**, where this method renders the content of the Picker View.

    Add the following highlighted codes at the bottom of **ThirdViewController.swift**.

```
    func pickerView(pickerView: UIPickerView,
        numberOfRowsInComponent component: Int) -> Int {
        return pickerData.count;
    }

    // Now, this is another function that should be implemented
    // by a UIPickerDataSource delegate. This simply tells
    // the UIPicker what to display in the contents for the picker
    // given the row/item number of the picker.
    //
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
        forComponent component: Int) -> String? {
            return pickerData[row]
    }
```

21. Build and Run the application.

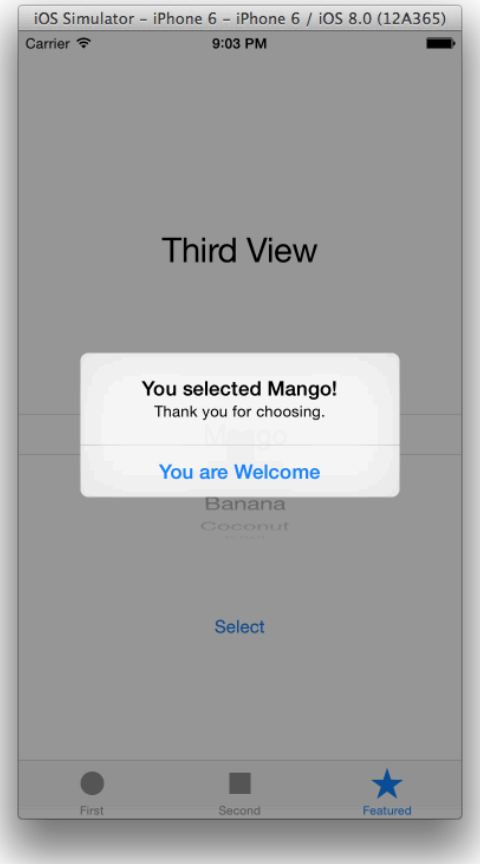


22. One more step to complete this lab. We have not implemented the method **buttonPressed()**. In the **ThirdViewController.swift**, add the following codes:

```
    @IBAction func buttonPressed(sender: AnyObject)
    {
        // Here we find out which row in the 1st component was
        // selected by the user.
        //
        let row = singlePicker.selectedRow(inComponent: 0)

        // Then we get the actual data from our
        // pickerData array.
        //
        let selected = pickerData[row]

        // Then we show an alert with the selected data.
        //
        let uiAlert = UIAlertController(
            title: "You selected \(selected)",
            message: "Thank you for choosing",
            preferredStyle: UIAlertControllerStyle.alert)

        uiAlert.addAction(UIAlertAction(
            title: "You are welcome",
            style: .default,
```

```
            handler:  nil))

        self.present(
            uiAlert, animated: true,
            completion: nil)
    }
```

23. Debug the application on the iPhone Simulator.



Challenge

1. Modify your UIPicker to show two sections of selections, the left side shows the following list of 4 characteristics:
    a. Juicy
    b. Tasty
    c. Sweet

d. Bland

And the right side shows the original list of f 5 fruits.

HINT: You need to modify the values from:

```
func numberOfComponents(in: UIPickerView)
func pickerView(UIPickerView, numberOfRowsInComponent: Int)
func pickerView(UIPickerView, titleForRow: Int,
      forComponent: Int)
```

2. Once the user clicks on Select, modify your alert to display the selected characteristics followed by the selected fruit.