Programare pe dispozitive mobile

iOS Lab - Lesson 3

Lesson goals:

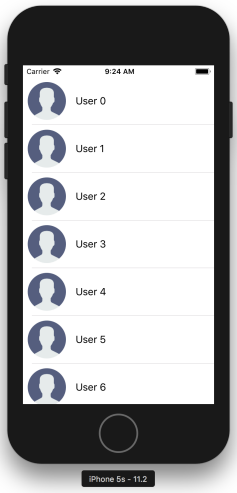
- Implement a Table View;
- Use a navigation Controller;
- Learn to integrate CocoaPods in a project.

1. Table Views

Table views on iOS display a single column of vertically scrolling content, divided into rows. Each row in the table contains one piece of your app's content. For example, the Contacts app displays the name of each contact in a separate row, and the main page of the Settings app displays the available groups of settings. You can configure a table to display a single long list of rows, or you can group related rows into sections to make navigating the content easier. Other examples where tables are used: news feed, a list of users (messeger, wapp) or a restaurant's menu.

An individual item of the table view is called a cell. A cell is a UITableViewCell object that acts like a template for the rows of the table. Cells are views, and they can contain any subviews that you need to display your content. You can add labels, image views, and other views to their content area and arrange those views using constraints.

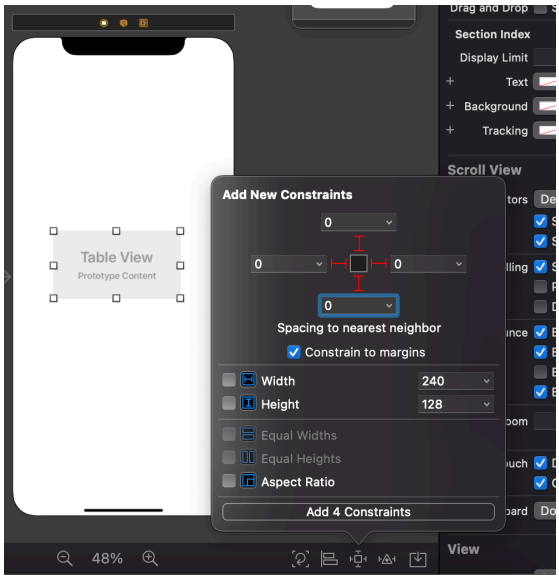
Let's start out with a really basic example of a table view, here's what we want to do:



First, open up Xcode, create **a new project** (**Xcode12**: Create a new Xcode project -> select iOS for Multiplatform, then select App)**.** I named my project **TableViewDemo**.

CONFIGURE A TABLE VIEW

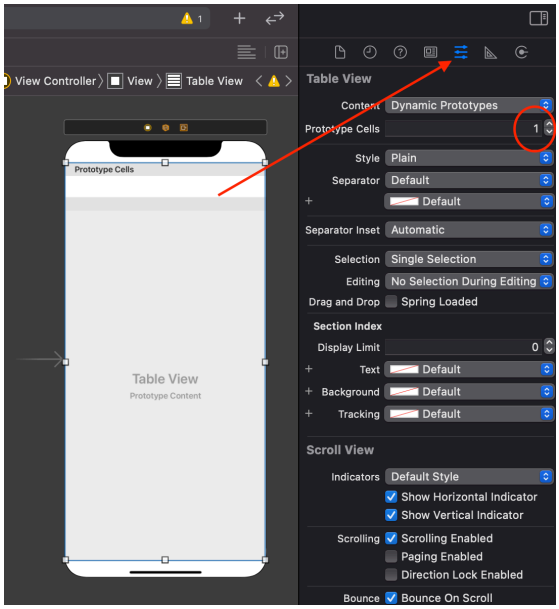
Open up the storyboard and place a new **table view** in our view controller. We want the table view to fill the screen, so add these 4 constraints (top, left, bottom and right)



CONFIGURE A CELL WITH CUSTOM VIEWS

Next, we need to add a cell to the table view. To do that, select the table view, go into the Attributes Inspector and set the number of **Prototype Cells** to **1**.

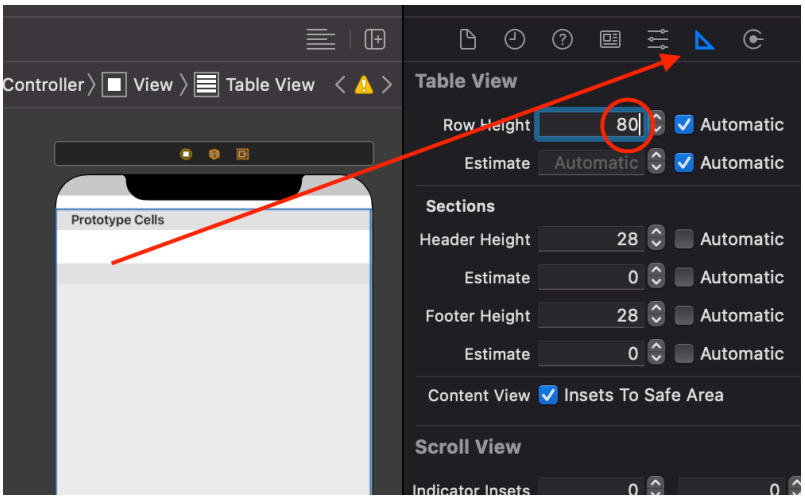
A **prototype cell** acts like a template for your cell's appearance. It includes the views you want to display and their arrangement within the content area of the cell.



Next, go into the Size Inspector, and set the **Row Height** to **80.**

UITableView provides default sizes for rows, but you can override the default height by assigning a custom value to the table view's rowHeight property. Always use this property when the height of all of your rows is the same.

If the row heights are not all the same, or can change dynamically, provide the heights using the **tableView(_:heightForRowAt:)** method of your delegate object. When you implement this method, you must provide values for every row in your table.



There you go, that's our new custom cell.

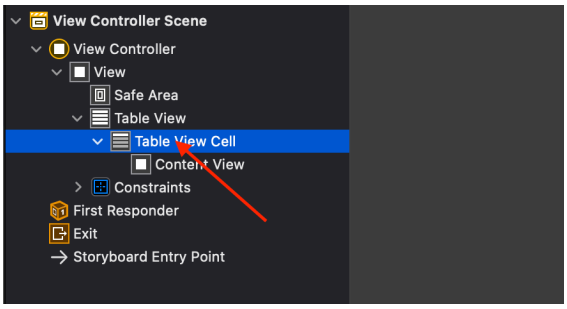
For custom cells, you need to define a **UITableViewCell** subclass to access your cell's views. Then add outlets to your subclass and connect those outlets to the corresponding views in your prototype cell.

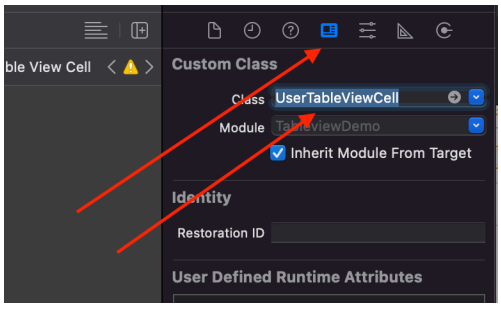
Go to **File** / **New** / **File…**

Select **Cocoa Touch Class**.

Make sure it's type is **UITableViewCell**. And name it **UserTableViewCell**.

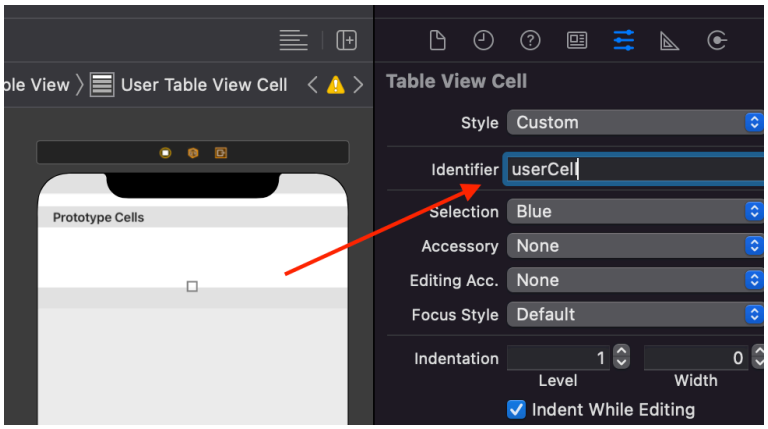
In the storyboard, select the cell, go to the **Identity Inspector** and set it's **Class** to **UserTableViewCell.**





With the cell still selected, go to the **Attributes Inspector** and set it's **Identifier** to **userCell**. Each cell in a table view must have a unique reuse identifier. Reuse identifiers facilitate the creation and recycling of your table's cells.

When you need a cell object at runtime, call the table view's **dequeueReusableCell(withIdentifier:for:)** method, passing the reuse identifier for the cell you want. The table view maintains an internal queue of already-created cells. If the queue contains a cell of the requested type, the table view returns that cell. If not, it creates a new cell using the prototype cell in your storyboard. Reusing cells improves performance by minimizing memory allocations during critical times, such as during scrolling.

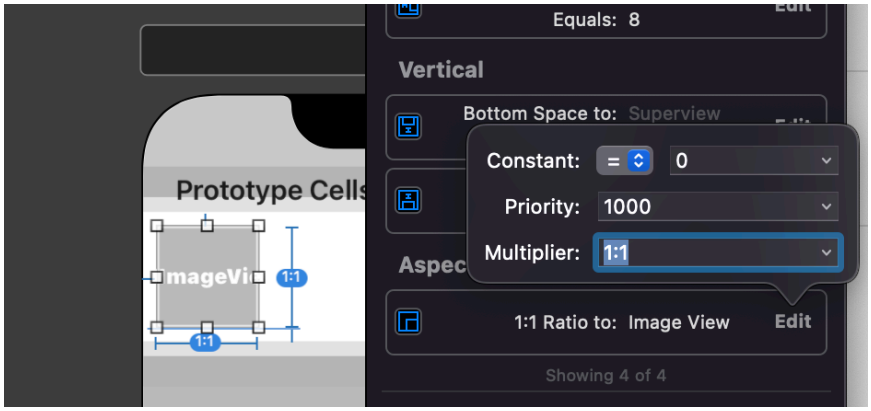


Next, download this image that we're going to use. Save it to your downloads folder.

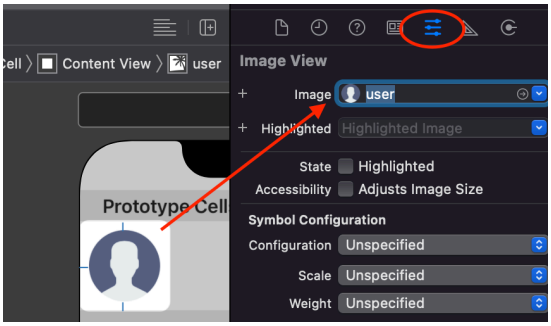
Then, back to Xcode, open up your **Assets.xcassets** folder. And drag the image from the downloads folder into your assets. Make sure it's named **user**.

Back to the storyboard, add an **Image View** to the cell. These 4 constraints (top, left and bottom and Aspect Ratio) should be work fine:

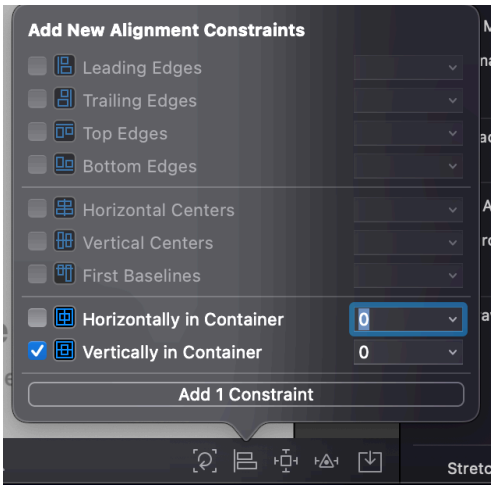
We want the image view to be a perfect square. Make sure it's selected, go into **Size Inspector**, find the **Ratio to: Image View** constraint, press **Edit** and set its **Multiplier** to **1:1**.

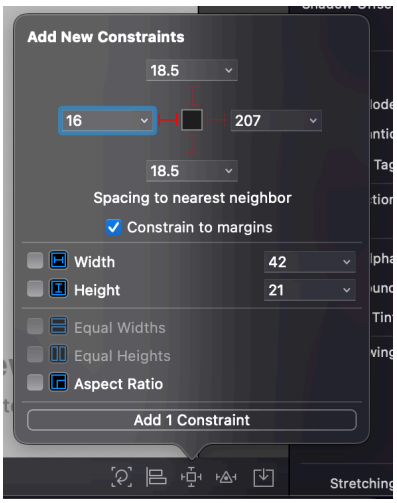


Next, go to the **Attributes Inspector** and set it's **Image** to **user.**

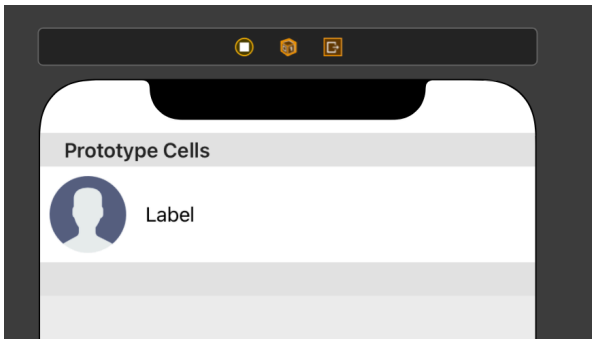


We're done with the image. Let's add a label. **Center it vertically** and add a **left constraint** of **16**.



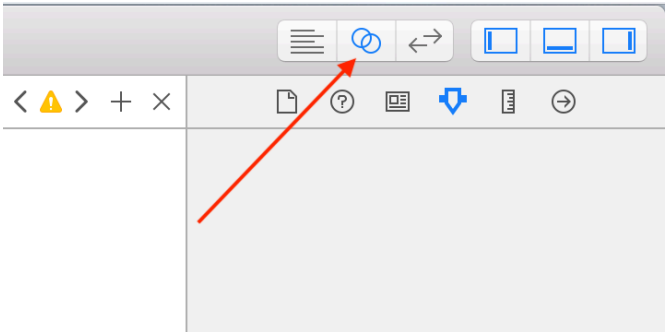


You should have something like this on your screen:

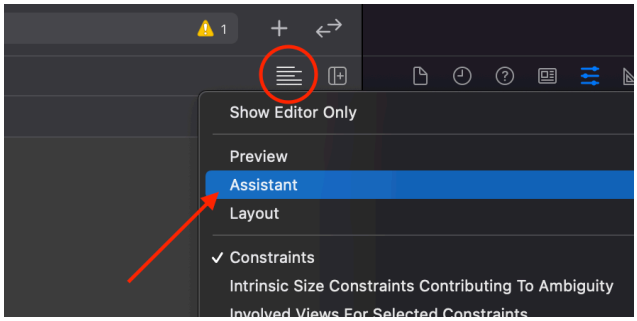


Next, we need an outlet for that label. Click the **Show Assistant Editor** button.

Xcode 10:

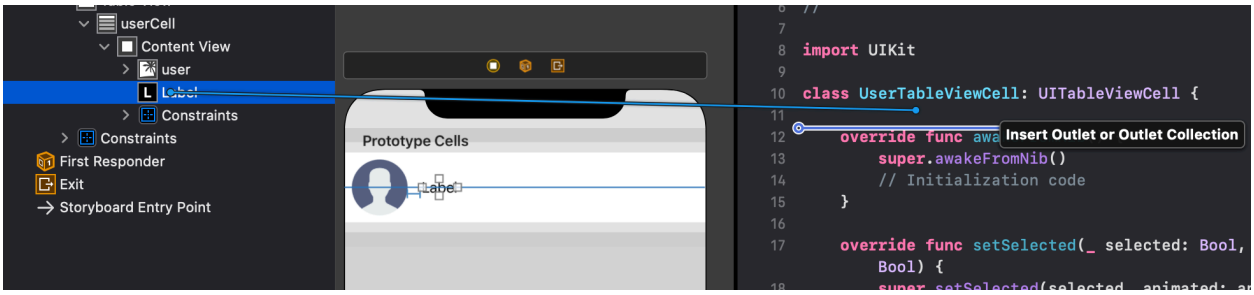


Xcode 12:



In the code editor, select your **UserTableViewCell.swift** file.

Create an outlet by **control** + **dragging** your label into the UserTableViewCell code. Name the outlet **userLabel**.



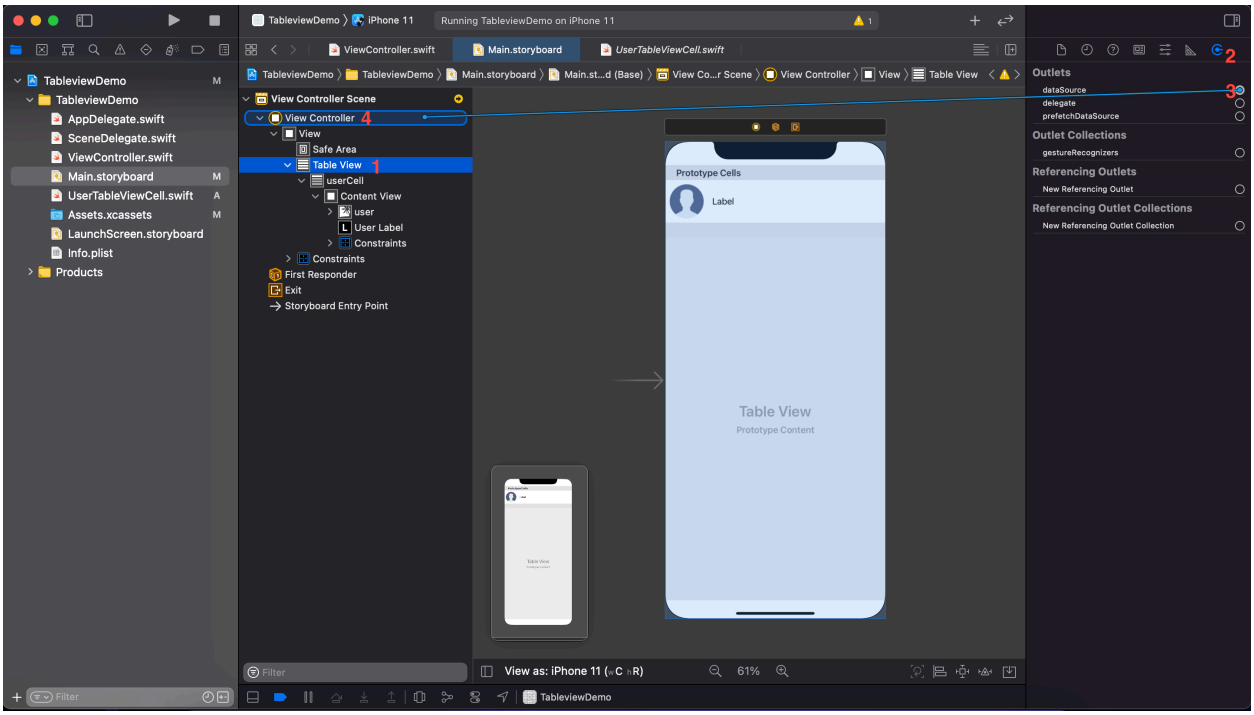
That's it for our cell. Let's run the app and see if worked.

You probably can't see any cells. That's because we didn't set the number of cell and we also didn't configure the cells in any way. So those are the two things that we still need to do.

FILLING A TABLE WITH DATA

First off, in the storyboard, we need to tell Xcode that the View Controller is the object that is going to be responsible for the number of cells and customizing the cells.

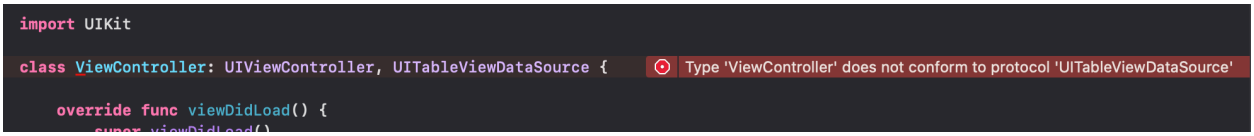
To do that, select the **table view** (1), go into the **Connections Inspector** (2), drag from the **dataSource** (3) circle to the **View Controller** (4).



Now, the table view knows that it relies on the View Controller to act as its datasource.

Let's configure the View Controller. Go into the **ViewController.swift** file.

We need the View Controller to implement the **UITableViewDataSource** protocol. We will talk protocols in another lesson. For now, all you need to do is add this in your **ViewController.swift** file.



Xcode is letting you know that the **UITableViewDataSource** protocol it's not correctly implemented. To do that we need to add a couple of functions.

Before it appears onscreen, a table view asks you to specify the total number of rows and sections. Your data source object provides this information using two methods:

```
func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return 20
}
```

Let's return 20 rows for now.

Before a table view appears onscreen, it asks its data source object to provide cells for the rows in or near the visible portion of the table. Your data source object's **tableView(_:cellForRowAt:)** method must respond quickly. Implement this method with the following pattern:

1. Call the table view's **dequeueReusableCell(withIdentifier:for:)** method to retrieve a cell object.

2. Configure the cell's views with your app's custom data.

3. Return the cell to the table view.

All we want to do here is update the **userLabel** to say ***User 0***, ***User 1***, ***User 2***. As shown in the initial screen shot of the final table view.

Here's the function:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let userCell = tableView.dequeueReusableCell(withIdentifier: "userCell") as! UserTableViewCell

    userCell.userLabel.text = "User \(indexPath.row)"

    return userCell
}
```

All we've done is retrieved the cell from the table view, updated the text of the cell and then returned the customized cell.

Run the app. That's it!

Let's check out a more complicated example. You can close this project

I created a demo project, it's hosted publicly on bitbucket.
Go back to the terminal and change the directory to ***~/Desktop***

Now download my project by typing this in the terminal window:
git clone https://cristitest3@bitbucket.org/cristitest3/restaurantdemo.git

Git will create a new folder on the desktop and download the project in that folder. The folder is called ***restaurantdemo.*** Go ahead and open it, and open the project in Xcode. Run the app.

You should be able to see a table view with dishes and a slider on top to filter the maximum price of the dishes.

The Table View has two different sections. Each section has a title and list of dishes.

The number of the sections is set here:

```
func numberOfSections(in tableView: UITableView) -> Int {
    return filteredSections.count
}
```

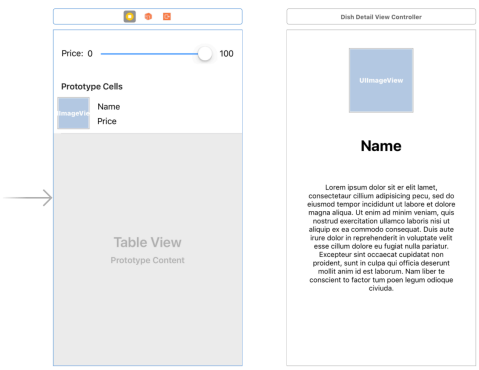
The title of each section is set here:

```
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    return filteredSections[section].name
}
```

The number of dishes in each section is set here: (This is the same function we used earlier and just returned **20** rows.)

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return filteredSections[section].dishes.count
}
```

You can see that nothing happens when you press on a cell. We want to go to that dishes details page.
Check the storyboard, you should be able to see the other screen in there.



First thing that we need to do is add a Navigation Controller. A **navigation controller** manages transitions backward and forward through a series of view controllers. The set of view controllers managed by a particular navigation controller is called its **navigation stack**. The first item added to the stack becomes the **root view controller** and is never popped off (removed from) the navigation stack.

Select the first view controller and then, in the top menu go to **Editor** / **Embed In** / **Navigation Controller.**

This will add a new navigation controller to the storyboard and set the root view controller of the as our Dish List View Controller.

Now, go to the **DishLishViewController.swift** file and find the **didSelectRow** function.

Here's the new code for that function:

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    tableView.deselectRow(at: indexPath, animated: true)

    let dishSection = filteredSections[indexPath.section]
    let selectedDish = dishSection.dishes[indexPath.row]

    let dishDetailViewController = storyboard?.instantiateViewController(withIdentifier: "dishDetail") as! DishDetailViewController
    dishDetailViewController.dish = selectedDish

    navigationController?.pushViewController(dishDetailViewController, animated: true)
}
```

First we figure out that the selected dish is.
Next we create a new DishDetailViewController, set it's dish variable and push it.

Try it out.

2. CocoaPods

What is CocoaPods?

From google: *CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 44 thousand libraries and is used in over 3 million apps.*

Here's how we're going to use it in our project:

I want you to change the slider from a *maximum price* slider to a *range slider* so you can select both the lower value and an upper value for the price. The default SDK does not offer a control for this. We can either create one ourselves or we can you a framework for this and include it in our project.

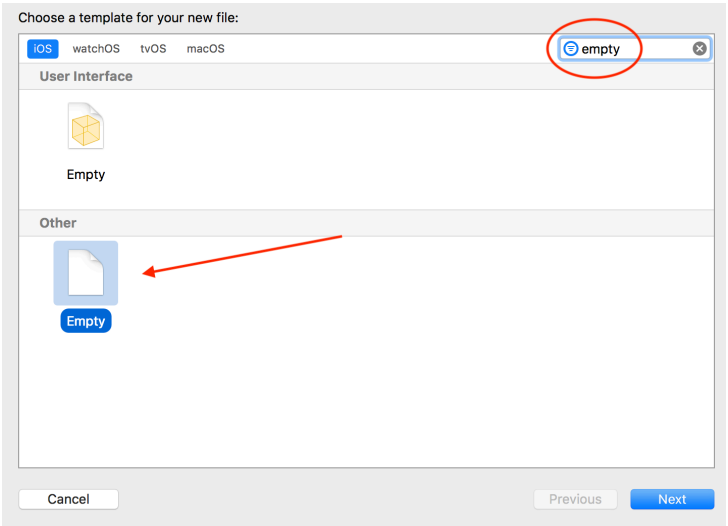
Go to https://cocoapods.org and search for ***range slider***. You will see multiple controls show up. Let's go with SwiftRangeSlider.

First we need to install CocoaPods in our project.

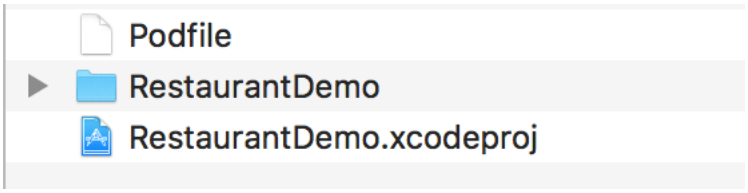
$ sudo gem install cocoapods

Create a Podfile.

Option 1: Create a new empty file in Xcode, name it Podfile (no extension) and save it in the root of your project.



It should be here:



Option 2:
Open a terminal window, and $ cd into your project directory.
Run *$ pod init*.

Open your Podfile. The first line should specify the platform and version supported.

Now, back in Xcode, the contents of the Podfile should be this:

```
use_frameworks!
target 'RestaurantDemo' do
      pod 'SwiftRangeSlider', :git => 'https://github.com/AlexeyMakonin/SwiftRangeSlider.git'
end
```

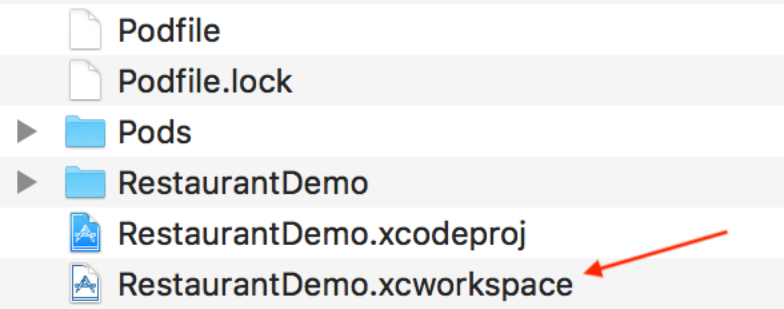
The **target** should be the name of our project, **SwiftRangeSlider** is the library we want to add and **1.1** is the version number. At the top, the **use_frameworks!** is needed because we are using swift.

You can close Xcode now.

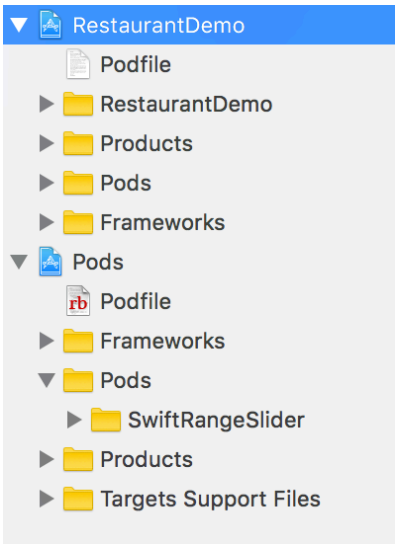
Go back to the terminal and run this command:

pod install

After it's done installing, you'll see it added a few new files.
Go ahead and open your workspace (not the project):

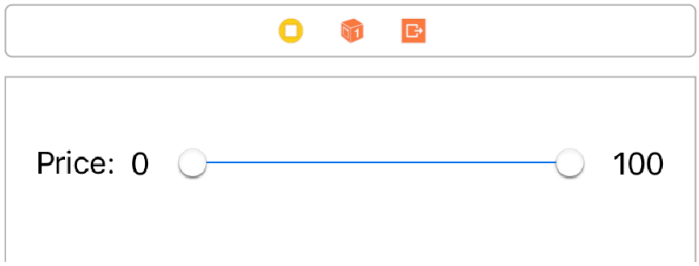


You'll see that the workspace contains both the **RestaurantDemo** project and a new project named **Pods**, that has the **SwiftRangeSlider** library in it.

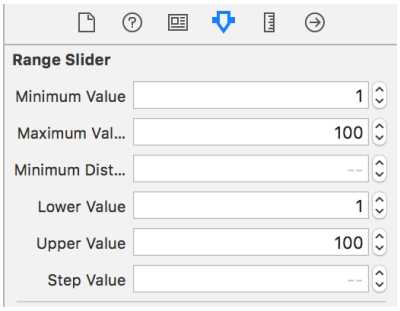


Try and switch the UISlider with a RangeSlider yourself.
You will to:

1. Replace the view
    - Delete the UISlider
    - Add a new UIView, set it's class to RangeSlider
    - Add some constraints to the new range slider



2. Configure the attributes of the range slider



3. Update the **sliderValueChanged** function
    - Go into DishListViewController and import the **SwiftRangeSlider** module
    - In the **sliderValueChanged** function declaration, change UISlider with RangeSlider
    - Replace **newPrice** with two new variables: **minPrice** and **maxPrice**, update the rest of the function to use those two variables

```
@IBAction func sliderValueChanged(_ slider: RangeSlider) {
    let minPrice = Int(slider.lowerValue)
    let maxPrice = Int(slider.upperValue)
```

    - Drag from the slider to the function