1.0 Hello world.

Welcome to WellySwift. Today we're going to talk about the basics of iOS development using Apple's new development language, **Swift**. We'll build a basic budgeting app and you'll learn the process a developer goes through when turning mockups into an app, and get a feel for some of the interactions you should be thinking about when designing or developing an app.

We've made sure the computers we're using today have the following software installed, so if you're trying this at home, here's what you'll need:

- A Mac computer running **OS X 10.10 Yosemite** or **OS X 10.9 Mavericks**
- **Xcode 6**, which you can download here: https://itunes.apple.com/nz/app/xcode/id497799835?mt=12

Today we'll be using **Swift**, a new language released by Apple this year, that builds on the best of older languages to make programming apps "easier, more flexible and more fun" (to quote Apple).

We're going to start by getting familiar with **Xcode**, a tool from Apple for making OS X and iOS apps. 

The app we'll be building is a basic savings app, which lets the user create a savings goal, and track how much they've saved towards it. We'll start with a basic design for the app, and later on in the day you'll have a chance to customise the colours, fonts, layout and imagery that we've used.

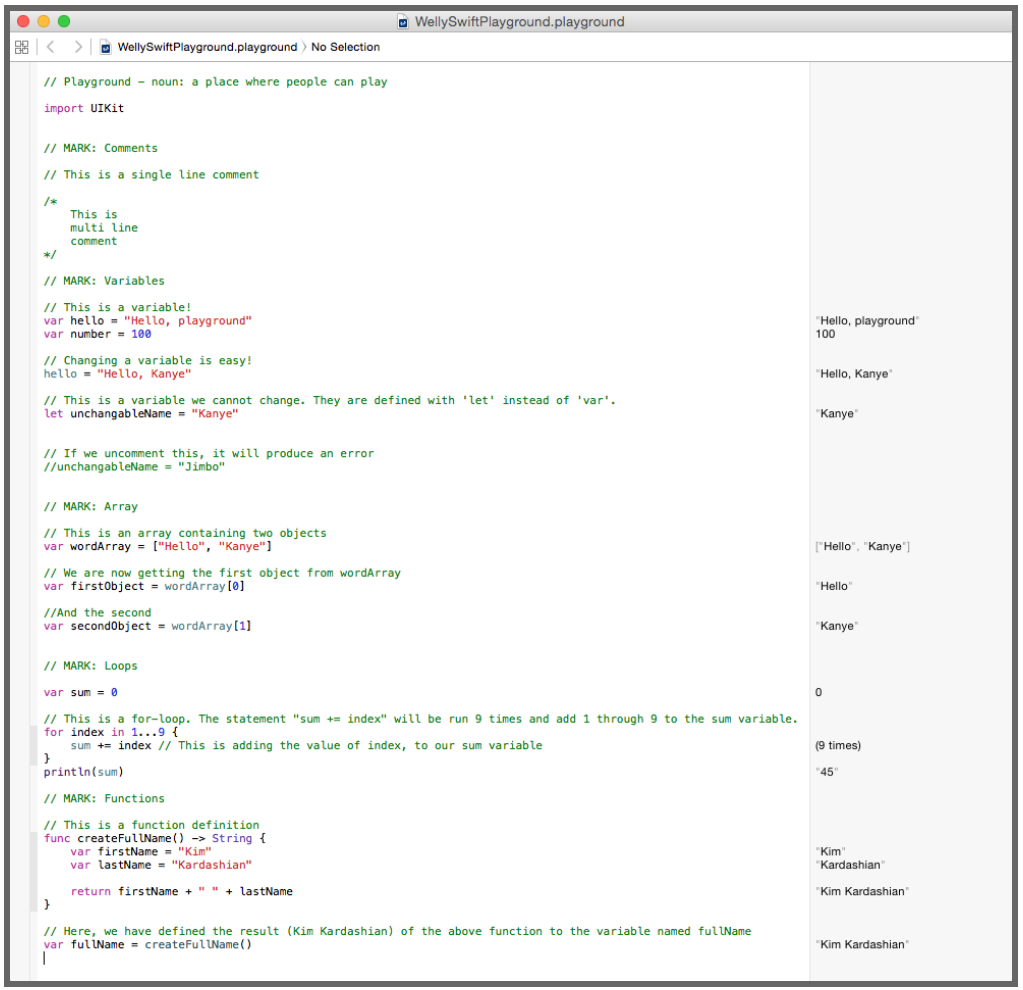
2.0 Getting started with Swift

Let's start by taking a look at a playground file. Playgrounds make writing Swift code super fun and simple. You can type a line of code and see the result appear immediately in the results sidebar.

Open up WellySwiftPlayground.playground in Xcode.

Check out the code we've written for you, and have a play around:

- Try adding a single-line comment
- Add a multi-line comment
- Declare a variable that equals "23"
- Change your variable to equal "25"
- Declare an array with your name and the name of the people next to you
- Write the code to get your name from the array
- Try using "println" to print this message to the console output: "I'm coding in Swift!"
- Try creating a function that returns your full name



3.0 Getting started with Xcode

3.1 Creating a project in Xcode

All iOS apps need a **project**, which organizes your files and resources needed to build the app. Once you've created a project, you can add new source files and begin writing code. For today, we've created a project for you, with the basic structure of the app you'll be building.

You'll see two projects on your Desktop. **WellySwiftApp** is the basic structure for the app you'll be building today, and **WellySwiftAppComplete** is the completed app.

Open up **WellySwiftAppComplete** in Xcode, to see the completed codebase and to practice building and running an app in Xcode.

In the top right-hand side, you'll see different options for viewing your project, including:

- Hide or show the **Navigator** (you'll want to show this, to view the file structure of your app)
- Hide or show the **Debug Area** (this will show you any error messages happening when you run your app)
- Hide or show the **Utilities** (you'll want to show **Utilities**, as this is where you'll find UI components to add to your **storyboard**)

3.2 Building and running your project in the Xcode simulator

Press **Command-B** to build your app. When you build your app, Xcode compiles all the files in your project, to create an executable application (i.e. an app that can be run on a device or simulator). If there are any issues with your code preventing it from running, Xcode will return these as build errors. Otherwise you should see a "**Build succeeded**" message.

Press **Command-R** to build and run your app. This allows you to run the app in a simulator, which we'll be using today. Make sure you have "**iPhone 5s**" selected from the dropdown in the top left of your Xcode screen.

4.0 Storyboards

4.1 Intro to storyboards

Storyboards are an iOS development tool that let you create iOS apps in a really visual way. You can create storyboard scenes for screens within your app. Using storyboards, it's easy to drag and drop UI components onto a storyboard scene.

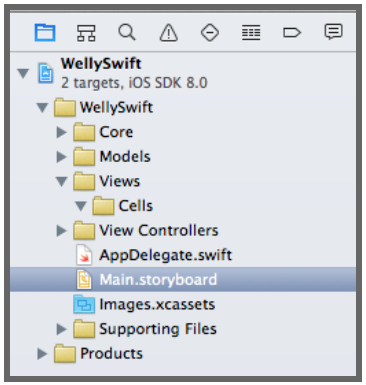
You might want to know the following terms:

- **Storyboard -** A **storyboard** is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A **storyboard** is composed of a sequence of scenes, each of which represents a **view controller** and its **views**.
- **Storyboard scene** - A **scene** represents one screen of content in your app.
- **View** - A **view** is an object which represents an onscreen element. An app screen has many different **views**, such as a delete button, an add button, a back button, a header, icons, etc.
- **View controller** - Your **view controllers** contain the logic that tie your app's **views** together. Each **view controller** organizes and controls a **view**.
- **Segue** - We use **segues** to transition between scenes. A **segue** object represents a transition between two view controllers. You can choose how you want this transition to be made by using one of the preset transitions (**Push**, **Modal** or **Popover**) or you can create your own custom transitions.

Xcode provides a visual editor for **storyboards**, where you can design the user interface of your application by adding **views** such as **buttons**, **table views**, and **text views** onto **scenes**. A **storyboard** also lets you connect a view to its controller object. This lets a view (e.g. a Cancel button) inform its **view controller** of an event (e.g. the Cancel button being pressed), so the **view controller** can make the appropriate thing happen (taking the user to the previous screen).

Using **storyboards** to design the user interface of your application lets you easily visualize the appearance and flow of your UI on one canvas.

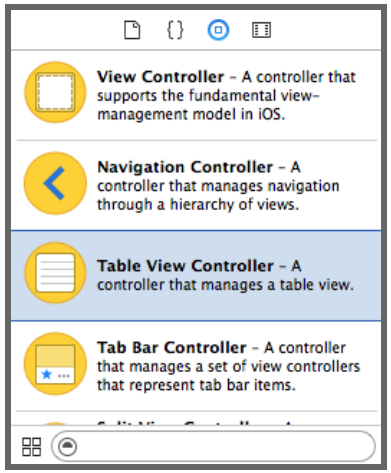
4.2 Finding your way around an Xcode project



Open up **WellySwiftApp** in Xcode. On the left, you'll see the following folders and files:

- **Core** - contains the custom **data objects**, which we've prepared for you. **Data** refers to anything that we store in our application - generally things like numbers and words (strings). In this case we'll be storing a Savings Target which consists of our savings goal (as a dollar number), the current progress (as a dollar number) and a title (as a string). Don't worry too much about this - just remember that **data** is information that we store.
- **Views** - contains the logic for the app's **views**, including the Cell folder, where we'll be putting some of the UI code later today.

- **View Controllers** - As we mentioned above, a **view** (e.g. a button) informs its **view controller** of things like being pressed, and then the **view controller** makes the appropriate things happen.
- **App Delegate** - the starting point for every iOS app, the **app delegate** manages events affecting the app that happen outside of its direct control, like launching and terminating. Your app needs to know when these things happen or are about to happen, so the operating system will automatically notify your **app delegate** by calling on the appropriate function (e.g. **applicationDidFinishLaunching** or **applicationWillTerminate**). The **app delegate** monitors when it's safe to open or close things, terminate the app, etc.
- **Main.storyboard** - This is the default **storyboard** file, where we'll be creating the structure and visual layout of your app.
- **Image.xcassets** - This is where we'll store all images that we want to use in the app. This includes your app icon and launch image, as well as imagery appearing within the app. It's important to make sure this folder contains version of each image for each screen resolution that the app is being built for (e.g. regular, Retina, and the higher resolution required by the iPhone 6).
- **Supporting files** - This contains default files, which we won't worry about today.



We're going to start by opening the **storyboard** file, so select **Main.storyboard** from the left-hand navigation. Make sure you can see the **Utilities** pane on the right hand side.

Near the bottom of the **Utilities** pane, you'll see four libraries which allow you to quickly add items to your **storyboard**:

- **File template library -** This provides templates for adding new files to your project.
- **Code snippet library** - If you're reusing code, you can save it here, to quickly drag and drop snippets of code into your files.
- **Objects library** - This lists all the UI components you can place onto a **storyboard**, and we'll be using these today.
- **Media library** - This is where the images you've placed into your **Images.xcassets** folder will show, allowing you to add them to scenes within your **storyboard**.

4.3 Adding a UI component to a storyboard scene

The first UI component we're going to use today is a **Table View Controller,** so select this from the **Objects library**, or search "Table View Controller" in the search bar at the bottom of the **Utilities** pane. Drag and drop the **Table View Controller** into the centre of your screen to add this to your **storyboard** file. This will be the main screen of the app.

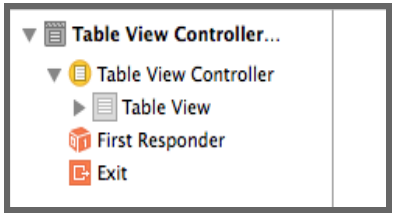
If you build and run your app (press **Command-R,** or push the **Play** button in the top left), you'll see a blank screen, because we haven't yet told Xcode to make this our main **storyboard** scene. Let's do that now.

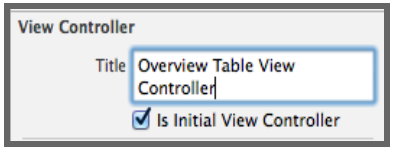
At the top of the **Utilities** pane, you'll see the **Properties** menu, with the following icons:

- **File inspector** - shows information about the **storyboard** file that you've selected
- **Quick Help inspector** - shows basic information about the **storyboard** scene you're currently viewing)
- **Identity inspector** - Lets you view and manage metadata for an object, such as its class name, accessibility information, runtime attributes, label, etc.
- **Attributes inspector** - This lets you configure attributes of the selected interface object. The attributes available are specific to the selected object. For example, some text field attributes include text alignment and color, border type, and editability.
- **Size inspector** - Specify characteristics such as the initial size and position, minimum and maximum sizes, and autosizing rules for an interface object.
- **Connections inspector** - View the outlets and actions for a **view**, make new connections, and break existing connections.

We'll want to name your scene, so we can easily see which screen is which later on. We also want to make it the default scene of your app.

- Make sure the **Document Outline** is visible, by toggling the icon in the bottom left of Xcode. Select the **Table View Controller** from the **Document Outline**.
- Select the **Attributes inspector** from the **Properties** menu.
- Under the **View Controller** section, you can enter a **Title** for the **Table View Controller** we've just created. Call your View Controller "**Overview Table View Controller**" and select the checkbox "**Is Initial View Controller**".
- Push the **Play** button to build and run your app. What has changed?



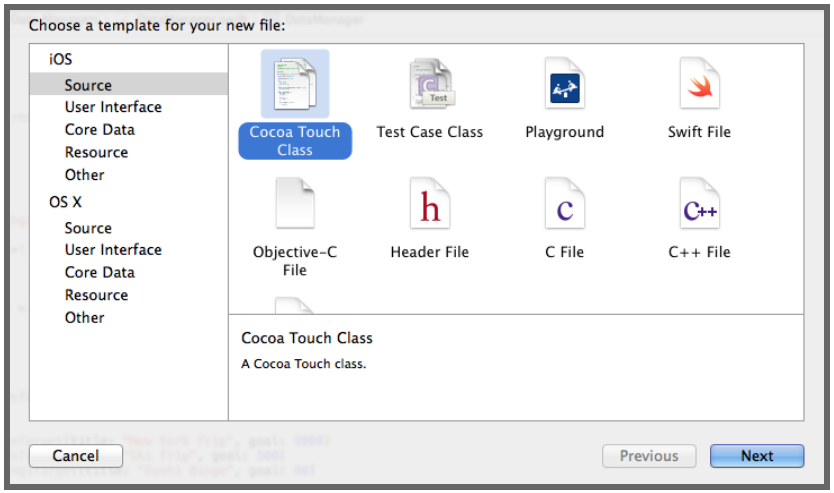


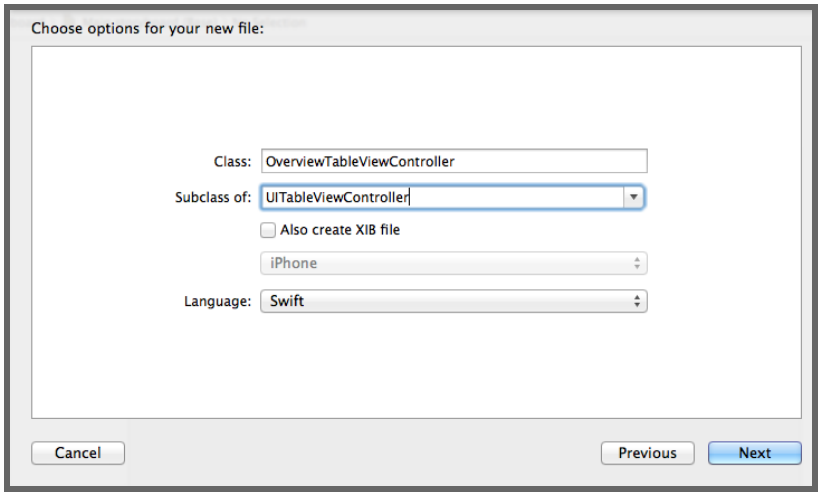
4.4 Showing data in a table view

Now we want to make data show up in our **table view**. Because we haven't yet made the ability to create any savings goals, we've made some pre-built savings data available for you in the **Data Manager** file.

We'll need to create a new Table **View Controller**, so right-click on the **View Controller** folder and select the **New File** option. Xcode gives us templates for creating new files, and in this case, we want to use the template for a **Cocoa Touch Class.** Select **Source** from the **iOS** tab, select **Cocoa Touch Class**, and then push **Next** (making sure the language is set to "Swift").

Let's name this class **OverviewTableViewController.** Make sure the class is a subclass of **UITableViewController.** Click **Next** and **Finish** to save the class.





Now it's time to hook everything up. We'll be connecting your **storyboard** scene "Overview Table View Controller" to the View Controller file you just created.

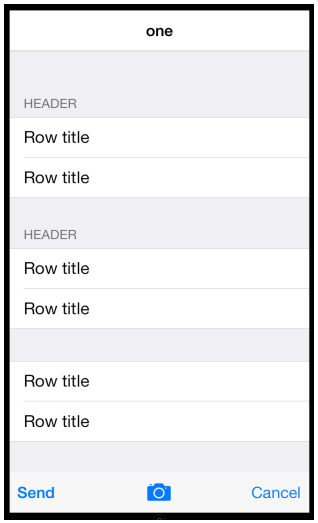
Select the **Overview Table View Controller** scene, and head to the **Identity Inspector** in the **Utilities** pane. We want to update the **Class** field to the class we just created (**OverviewTableViewController**), so type that in.

What did we just do here? We told the table **view controller** in our **storyboard** that it should use the **OverviewTableViewController** class we just created to populate the cells. Now we can modify this class to make things happen (like showing data in the table).

Select the **OverviewTableViewController** from the **View Controllers** folder.

In iOS, you can think of a **Table View** as a type of list. **Tables** are made up of **table view cells,** which display the information in the list. **Table view cells** are organised into sections and rows.

A **table view** can have any number of **sections**, which can have any number of **rows,** containing any number of **cells**. This image shows a **table view** containing three **sections**, with each **section** containing two **rows**.



To tell our app that we only need one section, find the following **numberOfSectionsInTableview** function:

```
override func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        // #warning Potentially incomplete method implementation.
        // Return the number of sections.
        return 0
}
```

Change the function to read "**return 1**" instead of "**return 0**".

Now we need to tell the app the number of **rows** we need. We don't yet know how many **rows** we'll need, so let's just hard-code 5 rows. Update "**return 0**" to "**return 5**" in the following function:

```
override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete method implementation.
        // Return the number of rows in the section.
        return 0
}
```

The function **cellForRowAtIndexPath** (below the **numberOfRowsInSection** function) allows you to customise the table cells. It's currently commented out, but we'll need this, so let's remove the comment marks. Xcode may return an error at this point, so just remove the exclamation marks from the function name to get rid of the error.

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("reuseIdentifier", forIndexPath: indexPath) as
UITableViewCell
        // Configure the cell...
        return cell
}
```

Now we need to make sure our **table view controller** knows which cell to use in this function. We can do this by adding a **reuse identifier**, which will help keep our app fast (by letting the app reuse cells that it has already created). We define a **reuse identifer** in our **storyboard** file on the prototype cell that was created when we added the **OverviewTableViewController** scene.

To add this, select **Main Storyboard**, **Overview Table View Controller**, **Table View**, and then **Table View Cell** from the **Document outline** pane.

Navigate to the **Attributes inspector** within the **Utilities** pane on the righthand side (the third **Utilities** tab).

Let's name this **OverviewTableViewCell**, as each of these cells will depict an overview of a Savings Target object. Type "**overviewTableViewCell**" into the **Identifier** field.

Now we need to tell the **cellForRowAtIndexPath** function the name of the Reuse Identifier, so that it can use the table cells from our **storyboard**. Update "**ReuseIdentifier**" to the name of our ReuseIdentifier (**overviewTableViewCell**). You can check the **storyboard** if you're unsure what you named the Reuse Identifier. The **cellForRowAtIndexPath** function should now look like this:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier
        ("overviewTableViewCell", forIndexPath: indexPath) as UITableViewCell
        // Configure the cell...
        return cell
}
```

Xcode has generated a comment saying "**Configure the cell..**" so let's go ahead and customise the cell by adding a background colour. Add this line of code below the comment:

```
cell.backgroundColor = .redColor()
```

Save and run the app again. What's changed? Why are only 5 cells red?

4.5 Using the DataManager class

The purpose of the **DataManager** class is to manage and provide access to data. We're going to add a new variable that points to our **DataManager** class, which stores our Savings Targets.

Find the following line of code within your **OverviewTableViewController** file:

```
class OverviewTableTableViewController: UITableViewController {
```

Below this line, add the following code to create an instance of the **DataManager** that we've created for you:

```
let overviewDataManager:DataManager! = DataManager()
```

We've just told Xcode to create an instance of **DataManage**r named **overviewDataManager**. The exclamation mark tells Xcode that this needs to be there.

We're going to want as many **cells** in our **table view** as there are **data objects**, or Savings Targets, in our **DataManager**. The table view doesn't know how many Savings Targets we've created, but the **overviewDataManager** does. We need to tell the **table view** to ask the **DataManager** how many rows are needed.

Find the function named **numberOfRowsInSection** and change it to the following:

```
override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete method implementation.
        // Return the number of rows in the section.
        return overviewDataManager.allSavingsTargets.count
}
```

We've just told Xcode to ask the **DataManager** to return the list of Savings Targets, and then count how many there are. That number will now be the number of rows in our table view. If we create a new Savings Target, we'll also get a new row in the table view to display this data.

Save and run your app again. How many red rows are there? Why?

Now let's display our (some information about our) Savings Targets. The first thing we want to show is the name of the Savings Target.

Go to the place in the file where you set the cell background colour. Here we're going to get a Savings Target object from the list and display the name of it in the cell.

Type the following code:

```
var target:SavingsTarget = overviewDataManager.savingsTargetAtPosition(0)
```

We've just told Xcode to return the variable "**target**" of variable type "**SavingsTarget**" from within the **overviewDataManager**. The position refers to the number of the list, and by asking for position 0, we're referring to the first object in our list.

Comment out the "**cell.backgroundcolour**" statement and add the following line of code:

```
cell.textLabel?.text = target.title
```

If that code has an error, just remove the question mark, like so:

```
cell.textLabel.text = target.title
```

The function should now read:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("overviewTableViewCell", forIndexPath: indexPath) as
UITableViewCell
        var target:SavingsTarget = overviewDataManager.savingsTargetAtPosition(0)
        // Configure the cell...
        cell.textLabel?.text = target.title
        return cell
}
```

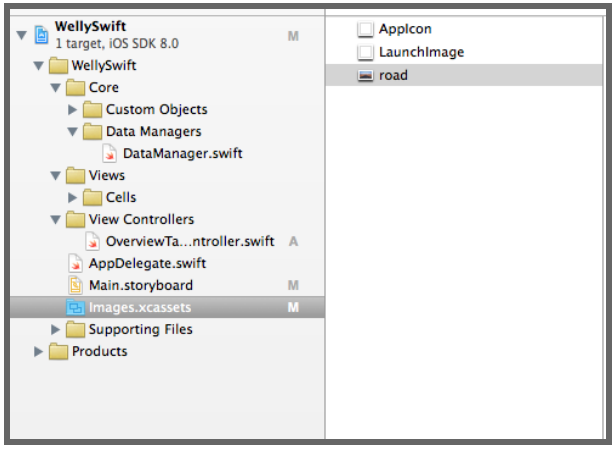
Run your app again. What title is displaying in each row? Why does every row have the same title?

Let's change this so we get the title for each Savings Target in the list and not just the first one.

We're storing our Savings Targets in an array, or group of data objects. The indexPath is represents the section number and the row number of the cell we are configuring. We can use this to retrieve a Savings Target object at a specific position in the list. Llet's replace the "**0**" with "**indexPath.row**", and save and run your app again.

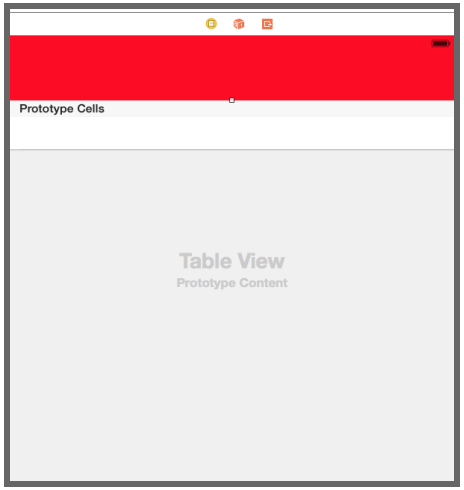
5.0 Styling

5.1 Adding a view



Now we're going to go back into the **storyboard** to start styling the app. Let's start with adding an image to the top. Open the **WellySwift Assets** folder on your desktop, and drag the **road@2x** file into the **Images.xcassets** file (you can find this in the **File Hierarchy** pane on the right hand side of our Xcode window).

Now we're going to add a new **view** to the top of the **table view**. Remember that a **view** is the base UI element used in iOS, and that everything you see displayed on an iOS device is a type of **view**. We'll add this new **view** to the **main.Storyboard** file. Select the **Object library** tab (in the **Utilities** pane) and search for a **View**. Drag and drop this to the top of your **storyboard** scene (just above the prototype cell). You can resize the **view** by dragging the edges up and down.



To change the background color, select the view, and select the **Attributes inspector**. The "**Background**" color dropdown will let you change the colour of the view.

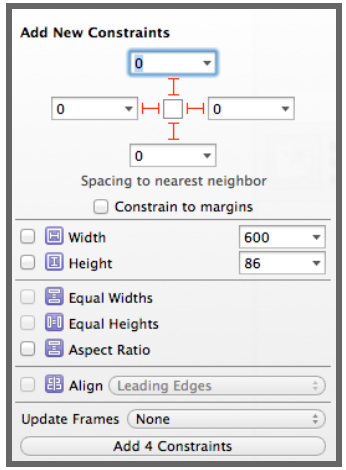
Run and save the app. Is your view displaying in your chosen colour?

Change the background colour of the view back to white, as we're now going to add a background image. Now we'll add an **image view** so we can display an image in the header. Select "**Image View**" in the **Objects library**, and drag this into the centre of the view you just created.

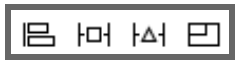
Select the **object view**, and under the **Attributes inspector** and start typing the name of the image into the "Image" field (**road**). Resize the view by dragging the edges up or down until you're happy with how the image is displaying.

Save and run the app. Is the full width of the image displaying?

5.2 The joys of Auto Layout



In order to make the image fit neatly in the view, we're going to use **Auto Layout**. **Auto Layout** makes it easy to support different screen sizes in your apps, by positioning and sizing things relatively. This means we can build a single Storyboard scene that will look the same on an iPhone 4, 5, 6, 6+ and even an iPad.



You'll see four icons in the bottom right of your **storyboard** screen. Make sure the **image view** is selected, and select the second icon "**Pin**". We're going to pin the image view to the top, bottom, left and right of the view containing it.

You can now resize the **view** by dragging the bottom edge up or down. Make sure to resize the **view**, not the **image view**, as the image view will automatically resize to fit (since we told the image view to stick to the edges of the view). Build and run the app.

5.3 Adding a button

We want users of our app to be able to add a new Savings Target, so let's add a button to the view, that will eventually show a new screen when pressed.

Find "Button" in the **Objects library** and drag it onto the bottom right of the image. Select the **button** on your **storyboard**, and edit the font, colour and text using the **Attributes inspector**.
Double click the button to change the text to "Add". In the attributes inspector, we are also able to change the text color. I chose white, because.
Build and run the app. Does anything look different?

You can't yet see the button because it's displaying full width - we haven't told **Auto Layout** to always keep the button in the same relative position. Let's do that now.

Select the button, and then select the second icon "Pin" from the **Auto Layout** menu. Pin the button to the **right** and **bottom** of the containing view. Add a **width** and **height** constraint.

Build and run the app again. Can you see the button?

6.0 Displaying Data

6.1 Adding data labels

In each cell, we want to display the savings goal title, the amount we've saved so far, and a progress bar.

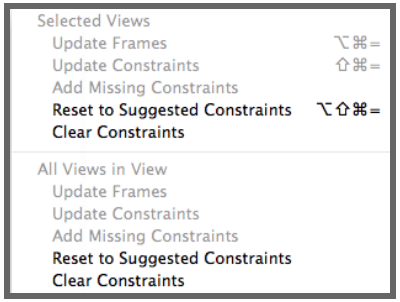
Go to the Main.storyboard file. Find "Label" in the **Objects library**, and drag this into the prototype cell on the far left. Double click on the label to change the text. Let's call this "NYC trip".

Now let's add another label for how much we've saved so far. Drag another label onto the righthand side of the cell, and change the text to "$1900 of $5000". Drag the labels so that they're sitting at the same height.

Go into **OverviewTableViewController.swift** and find the line containing "**CellForRowAtIndexPath**" and delete the following two lines:

```
var target:SavingsTarget = tableViewDataManager.savingsTargetAtPosition(indexPath.row)
cell.textLabel?.text = target.title
```

Go back into the **storyboard**. Let's add some **constraints** to the cells, like we did with the button and the image. Start with the "NYC trip" label, and open up the Pin tab in the **Auto Layout** menu. Turn off "Constrain to margins" and pin the label to the **top** and **left** of the cell. Tick the **height** checkbox, so the label will always display as the same height.



Let's do the same thing to the "$1900 of $5000" label - uncheck "Constrain to margins" and pin the button to the **top** and **right** of the cell, and tick the **height** checkbox.

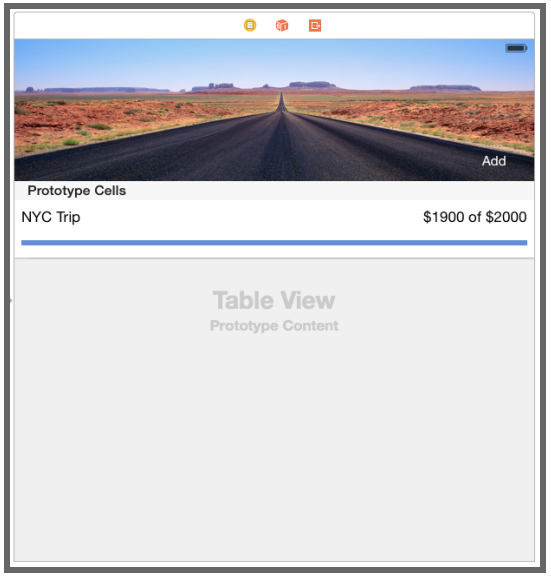
Moving the labels around will break the constraints you've just added. To clear the constraints, select the label, select the third tab from the **Auto Layout** menu, and select the top **Clear Constraints**. This should be under "**Selected Views**". Clicking the bottom one will clear all constraints on all views and you will have to re-add them! You can then drag the labels to a new position, and add the constraints again in the "Pin" tab.

Now lets add a view for the progress bar, to show how close we are to our savings goal. Start by dragging a **view** from the **Objects library** into the cell. Use the **Attributes inspector** in the **Utilities** pane to change the background colour, so we can see it easily. Resize the width of the **view** so it lines up with the sides of the labels. Using the **Size inspector** in the **Utilities** panel, change the height of the progress bar to 6px (or drag to resize).

Now let's add constraints to the progress bar. Select the view you just created, and add constraints using the pin tab in the **Auto Layout** menu. Pin the bar to the **bottom**, **left** and **right** of the containing **view**. Check the **height** box, as we want the **height** of the progress bar to stay the same, but the **width** to depend on the size of the screen.

Build and run your app again. Remember that if things aren't showing up correctly, you can select the **content view** in the **Document outline** pane and then clear all constraints from there. After that, re-add the constraints to your labels and progress bar view.

Now we should have something that looks like this:



6.2 Pulling through data

We want to display actual data though, so lets create a file that will tell our table view cell that we have created in the storyboard, to display our Savings Targets. Right click on **views** > **cells**, and create a new file by selecting **Source** from the **iOS** tab. Select **Cocoa Touch Class**, and then push **Next**. Let's give this class the same name as our **reuse identifier** (**OverviewTableViewCell**). Make sure the class is a subclass of **UITableViewCell.** Click **Next** and **Finish** to save the class.

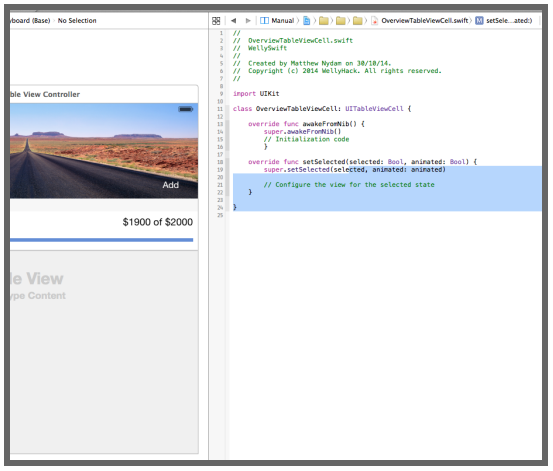
Now we need to tell the newly designed prototype cell to reference this file, by giving it a custom class. Go back to your **storyboard** scene and select the prototype **cell**. Using the **Identity inspector** in the **Utilities** pane, start type the name of the class in the class field (**OverviewTableViewCell**).

In your newly created file, you'll see a function named "**AwakeFromNib**". Change the background colour by adding the line of code below:

```
override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
        backgroundColor = .redColor()
}
```

Build and run the app. If we've linked the file correctly, we should see a change in color.

Because we have code changes to make, we're going to want to look at the **storyboard** and the code file side-by-side. On the top right-hand side of Xcode, select the second icon to show the **Assistant editor**. Select "**automatic**" on the **Assistant editor** and open the file you just created by navigating to: **Manual** > **WellySwift** > **WellySwift** > **Views** > **Cells** > **SavingsTableViewCell.swift**



Click on the "NYC trip" label and hold the control button on your keyboard and drag from the label to the assistant editor pane, just above the **AwakeFromNib** line.

Name the variable "**titleTextLabel**" and select "**connect**". Do the same thing with the Savings label - name the label "**currentProgressTextLabel**" and select "**connect**". The screenshot above is what you will see once you drop the connection in the file.

Delete the line of code that changes the background colour to red. Select "**OverviewTableViewController.swift**" and change this line of code:

```
let cell = tableView.dequeueReusableCellWithIdentifier
("overviewTableViewCell", forIndexPath: indexPath) as UITableViewCell
```

to the following:

```
let cell = tableView.dequeueReusableCellWithIdentifier("overviewTableViewCell", forIndexPath: indexPath) as
OverviewTableViewCell
```

We've just told Xcode that when it finds a cell, it needs to configure them as the cell we just created in **storyboard**.

Change the following code:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("overviewTableViewCell", forIndexPath:
    indexPath) as OverviewTableViewCell
    return cell
}
```

To this code:

```
    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
{
        let cell = tableView.dequeueReusableCellWithIdentifier("overviewTableViewCell", forIndexPath: indexPath) as
OverviewTableViewCell
        var target:SavingsTarget = overviewDataManager.savingsTargetAtPosition(indexPath.row)
        cell.titleTextLabel.text = target.title;
        cell.currentProgressTextLabel.text = target.formattedDisplayValue()
        return cell
    }
```

7.0 CRUD (Creating, Reading, Updating and Deleting) Savings Targets

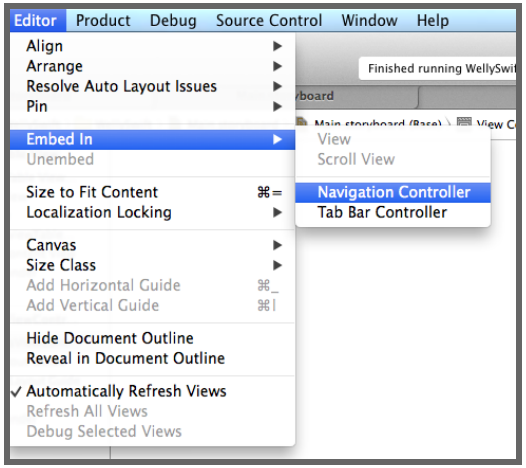
7.1 Using segues to transition between screens

Segues are a way of getting our application to transition between screens. We create **segues** in the **storyboard**, so let's move there now and get cracking.

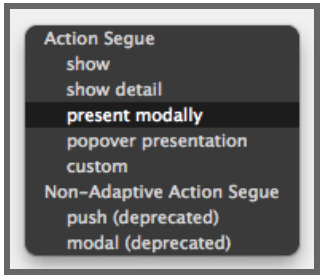
First things first, we need to create a new screen to transition to. We'll start by creating a new view controller. Navigate to **main.Storyboard** and search for "**view controller**" in the **Objects library** in the bottom right of the **Utilities** pane. Drag the view controller onto the **storyboard**, to the right of the original **storyboard** scene. Go into the attributes inspector in the **Utilities** pane to give the new view controller a name. Let's call this one "**AddSavingsTargetViewController**", as this is the screen that will allow users to add a new Savings Target.

We're going to need a **navigation controller**, to allow the user to move between screens in the app and back again, in different contexts. A **navigation controller** manages a stack of view controllers to provide a drill-down interface for hierarchical content. The view hierarchy of a **navigation controller** is made up of **view controllers** managed directly by the **navigation controller**. Each **view controller** manages a distinct view hierarchy, and the **navigation controller** coordinates the navigation between these view hierarchies. For the purposes of this app, you just need to know that we're creating a new **navigation controller** to contain our new **view controller**.

Select your new view controller, and then click on Editor in the main navigation. Under editor, select "Embed in" and then "navigation controller". How has your **storyboard** changed?



You can drag the navigation controller around the screen so it's not in the way. You might want to drag the navigation controller above the new view controller, so you can view the two view controllers side by side.



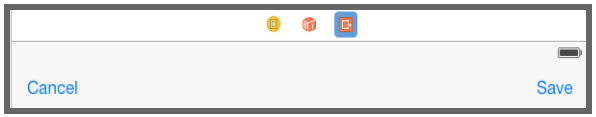
We now need to add a segue, so that when the user pushes the "add" button, the app knows to present this screen. Select the add button, and hold control on your keyboard, while dragging from the add button to the navigation controller. When you release the mouse, you should see the "**Action segue**" popup, from which you should select "**present modally**".

Although a navigation interface consists mostly of your custom content, there are still places where your code must interact directly with the navigation controller object. As well as telling the navigation controller when to display a new view, you are responsible for configuring the navigation bar—the view at the top of the screen that provides context about the user's place in the navigation hierarchy. You can also provide items for a toolbar that is managed by the **navigation controller**. Push command R to run the app. What happens when you press the add button?

Now we need a way for users to get back to the main screen. Let's add a "Cancel" button. Search for "**bar button item**" within the object library, and drag this into the navigation bar at the top of your **AddSavingsTargetViewController**. Add this to the top left of the bar. Select the button, and rename it using the **Attributes inspector**. Let's call the button "Cancel". To change the title of a button quickly, you can just double click it.

We want the cancel button to dismiss the modal screen that has appeared in front of the app's main screen. Navigate to your **OverviewTableViewController** file, and at the bottom you'll find a function named "**prepareForSegue**" near the bottom of the file. Add this code below the function:

```
@IBAction func unwindFromAdd(segue: UIStoryboardSegue) {
}
```



Move back into **main.Storyboard**, press control and drag from the Cancel button to the exit icon at the top of the view controller. This is calling the function we just added to the **OverviewTableViewController** file.

Run the app, and check that your "Add" and "Cancel " buttons display and dismiss the modal screen.

Go into the object library and search "text field". Drag this into the **AddSavingsTargetViewController** and make it nearly full width. Add another two text fields below this one (you can copy and paste the original text field).

We want to let users know what to put into each of these fields, so let's add some placeholder text. Use the **Placeholder** field in the **Attributes inspector** to call these fields "Title", Savings Goal, and "Current Savings Progress".

We'll want to add **Auto Layout** constraints again to make sure these text fields display correctly on different screen sizes. Select the **title** text box, and then select the pin tab from the bottom right of the screen to add **Auto Layout** constraints. Pin the text field to the **top**, **bottom**, **left** and **right**, making sure that "Constrain to margins" is checked. Do the same for the **Savings Goal** text field. For the **Current Savings Progress** text field, let's pin the field to the **top**, **left** and **right**, but not the bottom. Instead, let's give it a set **height** (check the Height box).

Run the app to make sure your text fields are sitting correctly. Can you type into the text fields?

Now that the user can enter content, we want to give them the ability to save. Let's start by adding a save button. Search for **Bar button item** in the **Objects library**, and drag this onto the top right of the navigation bar. Name the button save, using the **Attributes inspector**, or by double-clicking on the button to rename. press control and drag from the Save button to the exit icon at the top of the view controller.

Run the app. Is there any difference in how the **Save** and **Cancel** buttons are behaving?

7.2 Using tags to reference UI components in code

Now we need to tell Xcode the difference between the **Save** and "Cancel" buttons. In order to do this, we'll need to create a new **View Controller** file. In your project files, find the **View Controller** folder, right click on it, and select **New file.** Select **Cocoa Touch Class** from **iOS** > **Source**. Let's name this class **AddSavingsTargetViewController** as this will allow the user to save and edit a new Savings Target. Make sure the class is a subclass of **UI View Controller.** Click **Next** and **Create** to save the class.

To use this class, we need to hook it up to the **View Controller** in our **storyboard**, so move back to the **storyboard** scene containing the **Save** and **Cancel** buttons. Select this **View Controller**, and navigate to the **Identity inspector** in the **Utilities** pane. Type the name of the class we just created into the **Custom Class** field.

Move back into the **AddSavingsTargetViewController.swift** file in the project files. We're going to add a print statement to the **viewDidLoad** function, which will display the data in the console (at the bottom of your screen, below the file display. Add the following line of code to to the **AddSavingsTargetViewController.swift** file, at the end of the **viewDidLoad** function:

```
println("Loaded add Savings Target view controller")
```

You should see a commented out function at the bottom of the file named **prepareForSegue.** We're going to remove the comment marks, and the exclamation mark after **UIStoryboardSegue**, so that the function now looks like this:

```
// MARK: - Navigation
// In a storyboard-based application, you will often want to do a little preparation before navigation
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
// Get the new view controller using segue.destinationViewController.
// Pass the selected object to the new view controller.
}
```

We're now going to move the println statement into the **prepareForSegue** function, and change it to say "Prepare for segue" The function now looks like this:

```
// MARK: - Navigation
// In a storyboard-based application, you will often want to do a little preparation before navigation
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
// Get the new view controller using segue.destinationViewController.
// Pass the selected object to the new view controller.
```

```
println("Prepare for segue")
}
```

Run your app, and press the **Cancel** button. What output do you see in the console?

The **sender** is the UI component that initiated the segue, in this case, either the **Save** or **Cancel** button. Now we need to distinguish which **sender** is initiating, and we'll do this within the **AddSavingsTargetViewController.swift** file. Add the following lines of code before your **println** statement:

```
var barButton:UIBarButtonItem = sender as UIBarButtonItem
```

Let's add tags to our buttons, so we can reference them in code. Go back into your **storyboard** scene, and select the **Cancel** button. In the **Attributes inspector**, make sure the **tag** field says "0".

Select the **Save** button, and make sure the **tag field** in the **Attributes inspector** says "1". Now we'll be able to use these tags to distinguish between the buttons. Go back to your **AddSavingsTargetViewController.swift** file and add the following lines of code to the end of the **prepareForSegue** function:

```
if (barButton.tag == 0) { // Cancel Button Pressed
        println("Cancel Button pressed")
} else { // Saved Button Pressed
        println("Save button pressed")
}
```

Run the app. What displays in your console output when you press the **Save** or **Cancel** buttons?

7.3 Saving a new Savings Target

Now we need to add the ability to save, when the user enters a new Savings Target. To do this, we'll need to pull the data through. We'll be creating **outlets**, so we can reference the text fields within our **AddSavingsTargetViewController.swift** file.

Select the **storyboard** scene with the **Cancel** and **Save** buttons, and make sure the **Assistant Editor** is open (by selecting the icon in the top right of the Xcode screen). Make sure the **AddSavingsTargetViewController.swift** file is displayed in the **Assistant Editor** and if not, select it from the dropdown breadcrumbs menu.

Xcode makes it really easy for us to reference UI components (such as these text fields) within a file. Select the **Title** field, hold down the **Control** button on your keyboard, and drag from the **Title** field to just above the following line of code:

```
override func viewDidLoad()
```

When you release the mouse, you should see a popup, where you can enter **titleTextField** in the **Name** field. You should now see the following line of code in your file:

```
@IBOutlet weak var titleTextField: UITextField!
```

Now we'll do the same thing with the **Savings Goal** field (naming it **savingsGoalTextField**), and the **Current Savings Progress** field (naming it **currentSavingsProgressTextField**).

Now let's reference the **Title** field within the **prepareForSegue** function. Let's print the contents of the **Title** text field to the console, by adding the following line of code below our existing **println** statements:

```
println(titleTextField.text)
```

Run your app, and type something into the "Title" button before pressing **Save** or **Cancel.** What do you see in your console output?

Now that we can reference user-entered data, we need to tell Xcode to create a new Savings Target with this data, whenever the **Save** button is pressed.

The following code will create a new Savings Target from the user-entred fields, making sure that there aren't any empty fields (i.e. that the Savings Target has a title, a goal, and a current savings progress). Add this code above the **prepareForSegue** function:

```
func createSavingsTargetFromForm() -> SavingsTarget {
    var newTitle = titleTextField.text
    var goal = savingsGoalTextField.text.toInt()
    var currentProgress = currentSavingsProgressTextField.text.toInt()
    var newSavingsTarget:SavingsTarget! = SavingsTarget(title: newTitle, goal: goal)
    newSavingsTarget.progress = currentProgress!
    return newSavingsTarget
}

// Make sure all of the fields are there! And that the current progress is less than or equal to the goal
func allFormFieldsAreValid() -> Bool {
    var newTitle:String? = titleTextField.text
```

```
        var goal:Int? = savingsGoalTextField.text.toInt()
        var currentProgress:Int? = currentSavingsProgressTextField.text.toInt()
        if (newTitle != nil && goal != nil && currentProgress != nil) {
            return true
        } else {
            return false
        }
    }
```

Add the following code to the **prepareForSegue** function, under the **println("Save button pressed)** statement. This creates a new Savings Target, if all the form fields are valid (i.e. have content).

```
if (allFormFieldsAreValid()) {
        var target:SavingsTarget = createSavingsTargetFromForm()
}
```

Run the app, and add a new Savings Target. Does the new Savings Target show up on the home screen of the app?

This is because we haven't yet saved it to the **DataManager**, and we haven't told the **Overview Table View Controller** to reload its data. Let's do that now.

We'll need to create a delegate in the **AddSavingsTargetViewController**, which will tell the **Overview Table View Controller** to reload its data whenever we add a new Savings Target.

In your **AddSavingsTargetViewController** function, add the following code below the "import UIKit" line:

```
protocol AddSavingsTargetViewControllerProtocol {
        func addSavingsTargetViewControllerDidDismissWithSavingsTarget(target: SavingsTarget)
}
```

Add the following line of code above the IBOutlet definitions:

```
var delegate:AddSavingsTargetViewControllerProtocol?
```

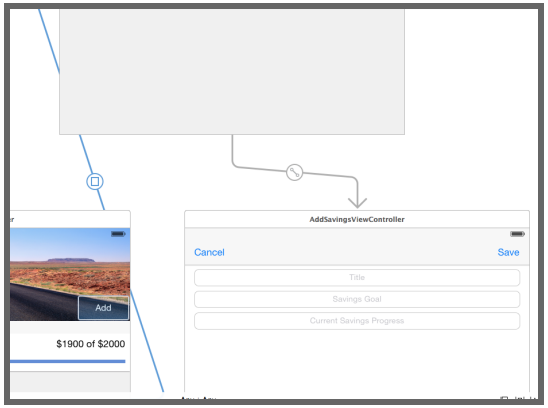
In the same file, add the following line of code inside your if statement in your **prepareForSegue** function:

```
delegate?.addSavingsTargetViewControllerDidDismissWithSavingsTarget(target)
```

The function should now read:

```
// In a storyboard-based application, you will often want to do a little preparation before navigation
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
        // Get the new view controller using segue.destinationViewController.
        // Pass the selected object to the new view controller.
        var barButton:UIBarButtonItem = sender as UIBarButtonItem
        if (barButton.tag == 0) { // Cancel Button Pressed
        println("Cancel Button pressed")
} else { // Saved Button Pressed
        println("Save button pressed")
        if (allFormFieldsAreValid()) {
                var target:SavingsTarget = createSavingsTargetFromForm()
                delegate?.addSavingsTargetViewControllerDidDismissWithSavingsTarget(target)
                }
        }
}
```

A **delegate** is a contract between two classes. Let's assume an object A calls object B to perform an action, and once the action is complete object A should know that B has completed the task and take necessary action. This is achieved with the help of **delegates**:



- A is delegate object of B
- B will have a reference of A
- A will implement the delegate methods of B.
- B will notify A through the delegate methods.

While we've called the delegate from the **prepareForSegue** function, we haven't set a class to be the delegate. We want the **Overview Table View Controller** class to be the delegate, so let's set that now.

Move to the main **storyboard**, and select the segue **Present modally segue to Navigation Controller** from underneath the **OverviewTableViewController** scene in the **Document outline** pane (the **segue**).

In the **Attributes inspector** for the segue, add “addSavingsTargetSegue” to the **Identifier** field. Naming the segue will let us reference this in code.

Let’s go back to our **OverviewTableViewController.swift** file. At the very bottom of the file, you’ll see a commented out function, which we’ll replace with the following:

```
    // In a storyboard-based application, you will often want to do a little preparation before navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
        // Get the new view controller using [segue destinationViewController].
        // Pass the selected object to the new view controller.
        if segue.identifier == "addSavingsTargetSegue"{
            let addSavingsViewController = segue.destinationViewController.topViewController as
AddSavingsTargetViewController
            addSavingsViewController.delegate = self
        }
    }
```

You’ll now see an error appear at the bottom of the file, “Does not confirm to protocol”, because we haven’t yet implemented the function we defined earlier in the **AddSavingsTargetViewController** class, to complete the delegate ‘contract’ between the two classes.

Let’s add a statement to the class definition of the **OverviewTableViewController** to resolve this error. Change this line of code:

```
    class OverviewTableViewController: UITableViewController {
```

To the following:

```
    class OverviewTableViewController: UITableViewController, AddSavingsTargetViewControllerProtocol {
```

We’re still seeing the error, so let’s add the function we defined earlier. Add the following code above the **prepareForSegue** function:

```
    func addSavingsTargetViewControllerDidDismissWithSavingsTarget(target: SavingsTarget) {
        println("I was called")
    }
```

Run the app again, and add a new Savings Target. What appears in your console output?

Now we'll save the user-entered data to the **DataManager**, and tell the **OverviewTableViewController** to reload the data. Update the **addSavingsTargetViewControllerDidDismissWithSavingsTarget** function again, so it reads as:

```
    func addSavingsTargetViewControllerDidDismissWithSavingsTarget(target: SavingsTarget) {
       overviewDataManager.updateOrInsertSavingsTarget(target)
        tableView.reloadData()
    }
```

Run the app again, and add a new Savings Target. Does it appear on the home screen?

7.4 Deleting a Savings Target

Now we're going to add the ability to delete a Savings Target. Find the following function in your **OverviewTableViewController** file:

```
    override func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle,
forRowAtIndexPath indexPath: NSIndexPath) {
```

It will be commented out, so remove the comment marks.

We'll need to add one more line of code to this function:

```
        overviewDataManager.removeSavingsTargetAtPosition(indexPath.row)
```

The function will now read:

```
    override func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle,
forRowAtIndexPath indexPath: NSIndexPath) {
        if editingStyle == .Delete {
            // Delete the row from the data source
            overviewDataManager.removeSavingsTargetAtPosition(indexPath.row)
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
        } else if editingStyle == .Insert {
```

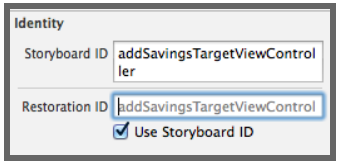
```
            // Create a new instance of the appropriate class, insert it into the array, and add a new row to the table
view
        }
    }
```

7.5 Editing a Savings Target

In order to edit a Savings Target, we'll need to add a new function to our **OverviewTableViewController.** Add the following code underneath the **CellForRowAtIndexPath** function:

```
    override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        let mainStoryboard:UIStoryboard = UIStoryboard(name: "Main", bundle: nil)
        let editSavingsTargetViewController:AddSavingsTargetViewController =
mainStoryboard.instantiateViewControllerWithIdentifier("addSavingsTargetViewController")
            as AddSavingsTargetViewController
        let editNavigation:UINavigationController = UINavigationController(rootViewController:
editSavingsTargetViewController)
        self.presentViewController(editNavigation, animated: true, completion:nil)
    }
```

We're creating a view controller with a specific identifier (**addSavingsTargetViewController**), but we haven't yet given our **addSavingsTargetViewController** this identifier in the **storyboard**, so the function won't know which view controller we're talking about.



Let's do this now, by going back to the **storyboard**, selecting the **addSavingsTargetViewController**, and adding "addSavingsTargetViewController" as the **Storyboard ID** in the **Identity inspector**, and selecting the checkbox "Use Storyboard ID".

Run the app again. What happens when you select a Savings Target on the home screen?

We want to see the Savings Target data on that screen, instead of a blank form, so let's make that display. Move into the **AddSavingsTargetViewController**, and add the following function underneath the **didReceiveMemoryWarning** function:

```
    func configureWithSavingsTarget(target: SavingsTarget) {
        editableSavingsTarget = target
    }
```

Now we're going to add another variable for Savings Target being edited, so add the following line of code above the IBOutlet declarations:

```
    var editableSavingsTarget:SavingsTarget?
```

Add an 'if' statement to the **viewDidLoad** function, so the function reads as:

```
    override func viewDidLoad() {
        super.viewDidLoad()
        if (editableSavingsTarget != nil) {
            titleTextField.text = editableSavingsTarget!.title
            savingsGoalTextField.text = String(editableSavingsTarget!.goal)
            currentSavingsProgressTextField.text = String(editableSavingsTarget!.progress)
        }
    }
```

When we select a Savings Target from the home screen, the **OverviewTableViewController** needs to know which Savings Target to open, so now we need to call the **configureWithSavingsTarget** function we created above.

Select the **OverviewTableViewController** class from the Project, and find the **didSelectRowAtIndexPath** function. We're going to add two lines of code, so the function now reads:

```
override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
       let mainStoryboard:UIStoryboard = UIStoryboard(name: "Main", bundle: nil)
       let editSavingsTargetViewController:AddSavingsTargetViewController =
mainStoryboard.instantiateViewControllerWithIdentifier("addSavingsTargetViewController")
              as AddSavingsTargetViewController

        // New code
        var target:SavingsTarget = overviewDataManager.savingsTargetAtPosition(indexPath.row)
        editSavingsTargetViewController.configureWithSavingsTarget(target)
                editSavingsTargetViewController.delegate = self
```

```
        let editNavigation:UINavigationController = UINavigationController(rootViewController:
editSavingsTargetViewController)
        self.presentViewController(editNavigation, animated: true, completion:nil)
}
```

Run your app and tap on a Savings Target. Do you see the correct data on the Savings Target screen? Try editing the data. Did it work?

7.6 Saving when we edit

Now we want to make sure that when we edit a Savings Target, the target is edited. Currently, it will just create a new one. Not exactly what we want.. We'll do this in the **AddSavingsTargetViewController** class.

Add the following function below the createSavingsTargetFromForm function:

```
    func updateSavingsTarget(target:SavingsTarget) {
        target.title = titleTextField.text!
        target.goal = savingsGoalTextField.text.toInt()!
        target.progress = currentSavingsProgressTextField.text.toInt()!
    }
```

Replace the **prepareForSegue** function with the following function, that tells the app to get a new object from the form, unless we're editing an existing Savings Target, in which case it should replace the old Savings Target with the new data:

```
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        let barButton:UIBarButtonItem? = sender as? UIBarButtonItem
        if (!allFormFieldsAreValid()) {
            println("Please fill out all the fields and make sure they are valid!")
            return
        }
        // Let's send the objects back.
        if (delegate != nil ) {
            // If we are not editing a Savings Target
            if (editableSavingsTarget == nil) {
                editableSavingsTarget = createSavingsTargetFromForm() // Get our new object from our form
```

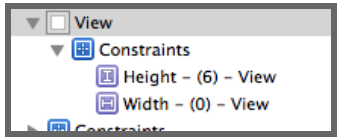
```
            } else {
                updateSavingsTarget(editableSavingsTarget!) // Edit the current Savings Target
            }
            delegate?.addSavingsTargetViewControllerDidDismissWithSavingsTarget(editableSavingsTarget!)
        }
    }
```

Run your app again. Create and edit a Savings Target. Were your changes saved?

8.0 Extension exercise - making a dynamic progress bar

We want our users to know how close they are to their Savings Target, so let's add a dynamic progress bar. Go back to your **storyboard.**

Making sure the progress bar is selected, we want to clear the constraints we gave it earlier.
Now we want to give the bar a width of 0px in the **Size inspector**. Using **Auto Layout,** constrain the new view to the left and bottom margins, and make sure the **width** and **height** boxes are checked.



We want the width of the progress bar to show the user's progress towards their Savings Target, so we'll need to reference the width in code to dynamically update it. Select the **Width** Constraint in the **Document outline** pane, hold down the Control button, and drag from the **Width** Constraint to above the **IBOutlets** declarations in the code. You should see the option to name your new variable, so name this "**widthConstraint**", and select **Connect**.

Now let's create a new variable in **OverviewTableViewCell.swift** that will hold a Savings Target, by writing the following line of code above the **IBOutlet** declarations:

```
required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
}
var currentTarget:SavingsTarget!
```

Now let's add the following function to the bottom of the **OverviewTableViewCell** class:

```
func configureWithSavingsTarget(target: SavingsTarget) {
    currentTarget = target
    titleTextLabel.text = target.title
    currentProgressTextLabel.text = target.formattedDisplayValue();
    updateConstraints()
}
```

Now we need to call that function in our **OverviewTableViewController** by replacing these lines of code:

```
    // Configure the cell...
    cell.titleLabel.text = target.title
    cell.currentProgressLabel.text = target.formattedDisplayValue()
```

With this code:

```
        cell.configureWithSavingsTarget(target)
```

Move back to the **OverviewTableViewCell** and add the following functions to the bottom of the class:

```
   override func prepareForReuse() {
        widthConstraint.constant = 0
        super.prepareForReuse()
    }
    override func updateConstraints() {
        // This will only work correctly if our progress bar is set in by 20px on each side in our storyboard.
        var maxWidth:CGFloat! = contentView.bounds.size.width - 40 // 40 includes our two edge insets.
        var savingsGoalProgressWidth:CGFloat! = currentTarget.getProgressWidth(maxWidth) // Get a width based on our max
width
        widthConstraint.constant = savingsGoalProgressWidth // Set the width of our constraint. This will reflect in the UI
        super.updateConstraints()
    }
```

Run your app, and create a new Savings Target. Can you see your savings progress? Edit your savings to see the progress bar grow.

9.0 Customising your app

If you've already completed Sections 1-8 (you natural, you), here's some ideas for further customising your app.

- New icons and imagery - you could add an icon image for the "Add" button, that contrasts with the header image, and/or add a different header image.
- Customising your labels - you could change the colours, fonts and sizing of the text labels.
- You could change the overview cell layout - make it pixel perfect! (Remember to clear your constraints and add them again if you're moving UI components).
- Styling your dynamic progress bar - you could add rounded corners and/or make the background of the progress bar a more transparent shade of the progress bar colour.

10.0 Where to from here?

If you're interested in learning more about iOS design and development after today, there are plenty of online tutorials to help you on your way!

There's also heaps of Wellington-based groups for mobile designers and devs - we've listed a few below, but there's plenty more on Meetup.com.

Online resources for iOS design:

- https://designcode.io/iosdesign
- http://teamtreehouse.com/library/mobile-app-design-for-ios

Online resources for iOS development:

- http://mathewsanders.com/prototyping-iOS-iPhone-iPad-animations-in-swift/
- https://designcode.io/swift
- https://www.udemy.com/swift-learn-apples-new-programming-language-by-examples/
- http://mhm5000.gitbooks.io/swift-cheat-sheet/content/index.html

Meetup groups:

- Hack Pack (that's us!) - http://www.meetup.com/hackpack/
- Wellington Mobile Dev Group - http://www.meetup.com/wellington-mobile-dev/
- Game Developers of Wellington - http://www.meetup.com/Game-Developers-of-Wellington/
- Playing With Code Wellington - http://www.meetup.com/playing-with-code-wellington/
- Weekend Founders - http://www.meetup.com/Weekend-Founders/