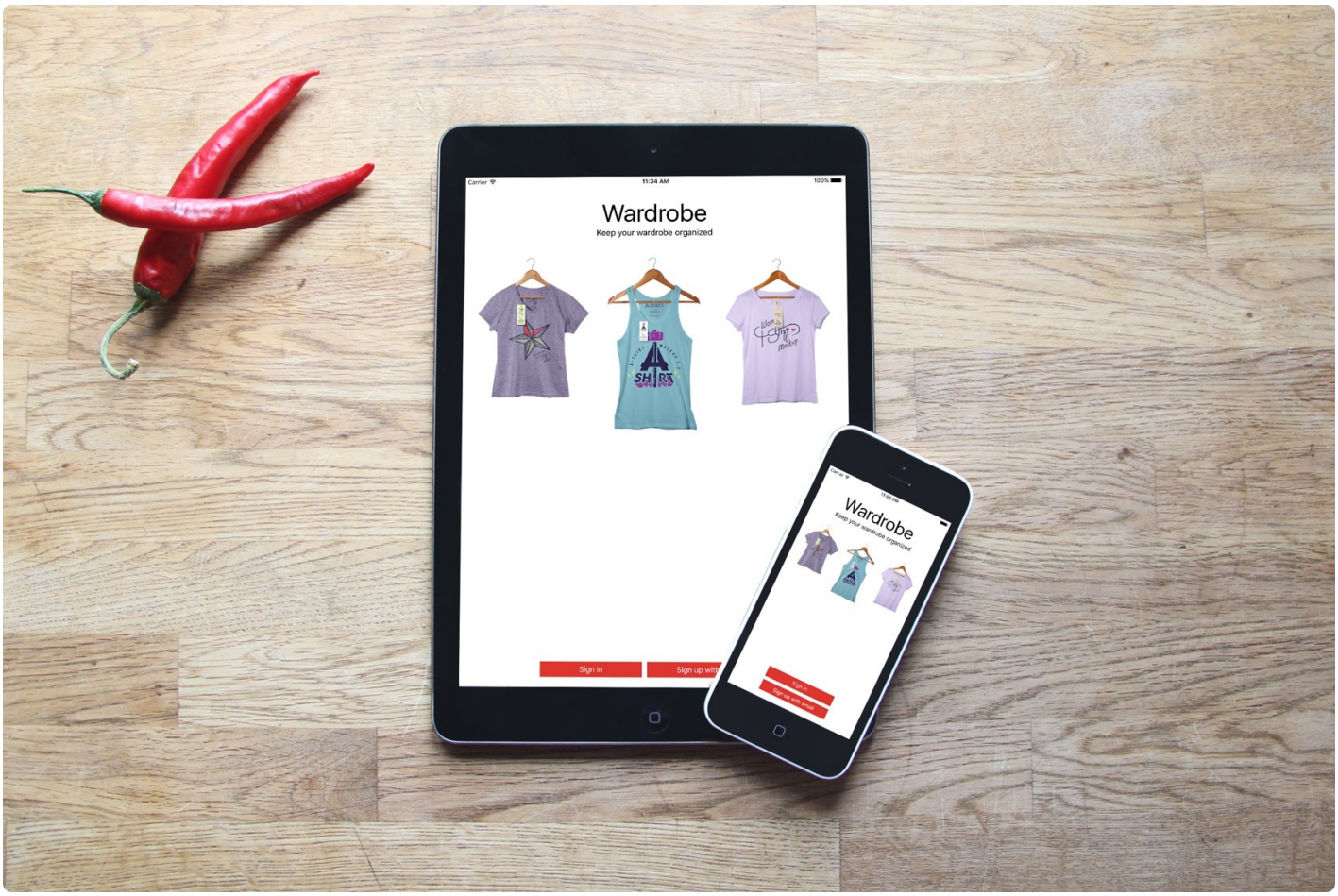
Chapter 6
Designing UI Using Stack Views



> To the user, the interface is the product.
>
> - Aza Raskin

iOS 9 comes with a lot of new features that make our developers' life simpler. The introduction of Stack Views is a great example. I have given you a brief overview of auto layout. The examples that we have worked on were pretty easy. However, as your app UI becomes more complex, you will find it more difficult to define the layout constraints that apply perfectly for all iOS devices. This is one of the reasons why Apple introduces Stack views in the latest version of Xcode and iOS.

In this chapter, we will continue to focus on discussing UI design with Interface Builder. I will teach you how to build a more comprehensive UI, which you may come across in a real-world application. You will learn how to:

1. Use stack views to lay out user interfaces.
2. Use image views to display images.
3. Manage images using the built-in asset catalog.
4. Adapt stack views using Size Classes.

On top of the above, we will explore more about auto layout. You'll be amazed how much you can get done without writing a line of code.

What is a Stack View

First things first, what is a stack view? The stack view provides a streamlined interface for laying out a collection of views in either a column or a row. In Keynote or Microsoft Powerpoint, you can group multiple objects together so that they can be moved or resized as a single object. Stack views offer a very similar feature. You can embed multiple UI objects into one by using stack views. Best of all, for views embedded in a stack view, you no longer need to define auto layout constraints.

```
Quick note: For views embedded in a stack view, they are usually known as
arranged views.
```

The stack view manages the layout of its subviews and automatically applies layout constrants for you. That means, the subviews are ready to adapt to different screen sizes. Furthermore, you can embed a stack view in another stack view to build more complex user interfaces. Sounds cool, right?

Don’t get me wrong. It doesn’t mean you do not need to deal with auto layout. You still need to define the layout constrants for the stack views. It just saves you time from creating constraints for every UI element and makes it super easy to add/remove views from the layout.

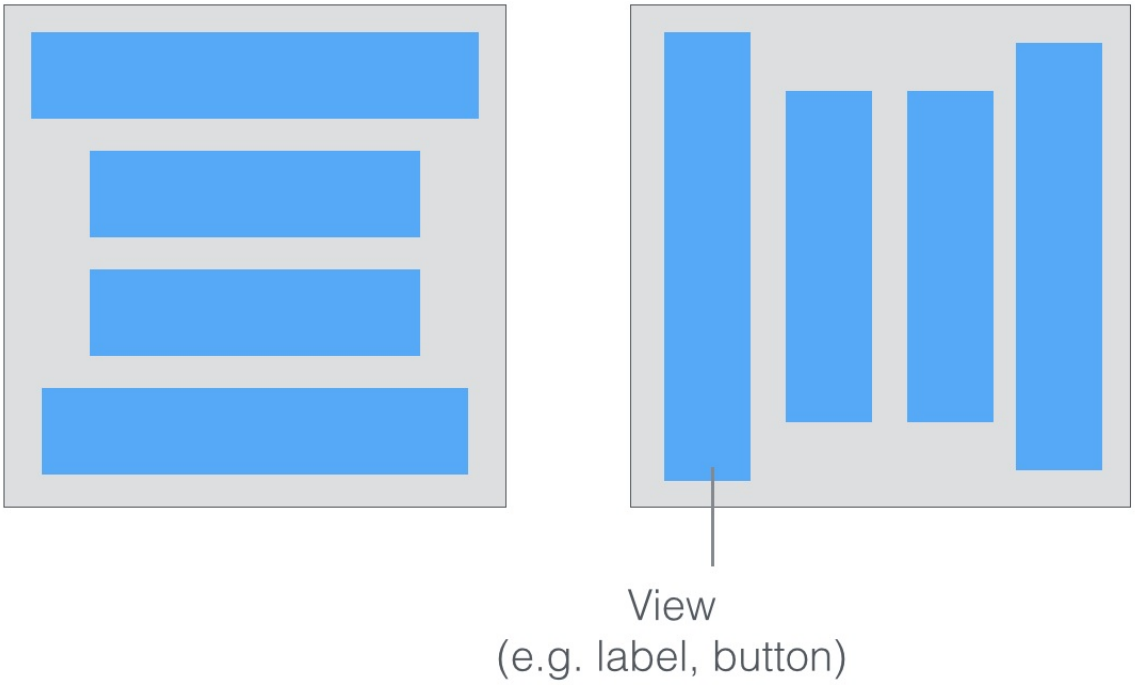


Figure 6-1. Horizontal Stack View (left) / Vertical Stack View (right)

Xcode 7 provides two ways to use stack view. You can drag a Stack View (horizontal / vertical) from the Object library, and put it right into the storyboard. You then drag and drop view objects such as labels, buttons, image views into the stack view. Alternatively, you can use the Stack option in the auto layout bar. For this approach, you simply select two or more view objects, and then choose the Stack option. Interface Builder will embed the objects into a stack view and resize it automatically. If you still have no ideas about how to use a stack view, no worries. We'll go through both approaches in this chapter. Just read on and you'll understand what I mean in a minute.

The Sample App

Let's first take a look at the demo app we're going to build. I will show you how to layout a user interface like this using stack views:

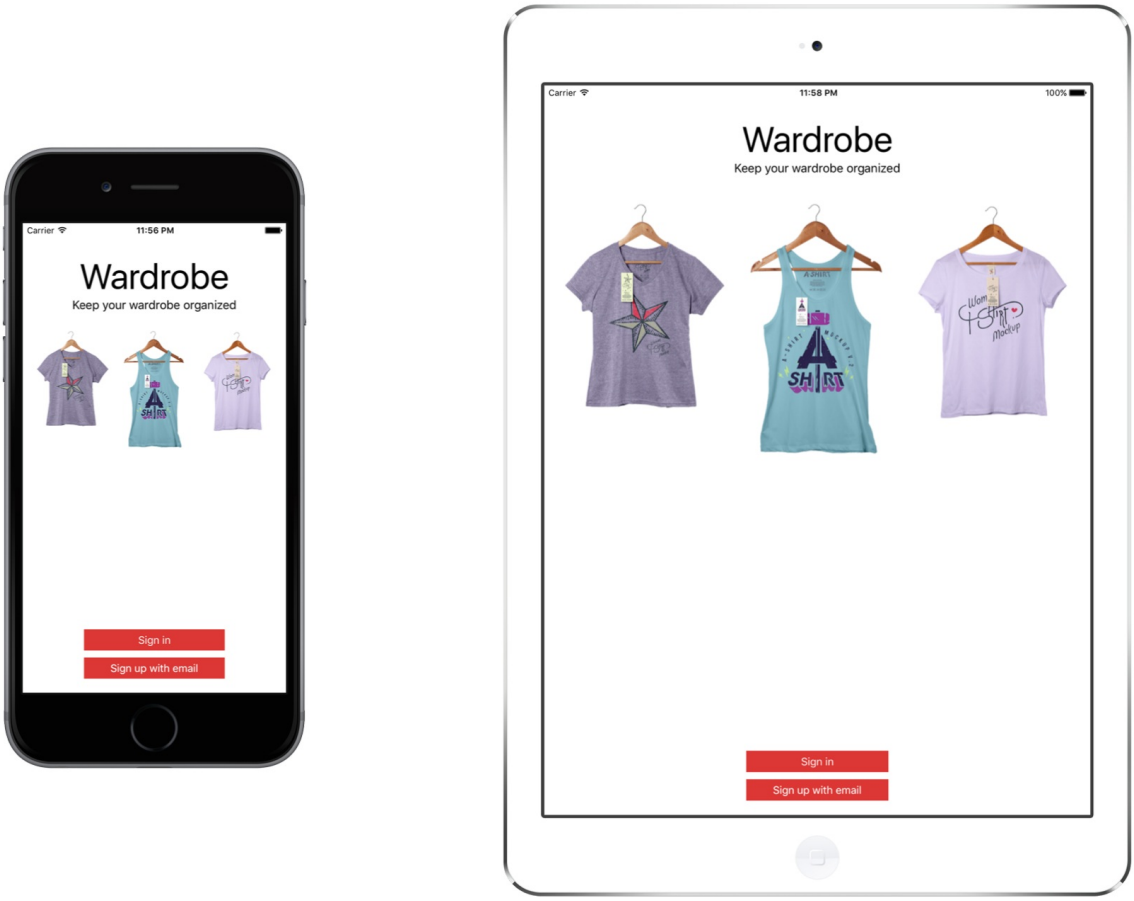


Figure 6-2. The sample app

You can create the same UI without using stack views. But as you will see, stack views completely change the way how you layout user interfaces. Again, there is no coding in this chapter. We will just focus on using Interface Builder to layout the user interface.

Creating a New Project

Now fire up Xcode and create a new Xcode project. Choose Application (under iOS) > "Single View Application" and click "Next". You can simply fill in the project options as follows:

- **Product Name: StackViewDemo** – This is the name of your app.
- **Organization Name: AppCoda** – It's the name of your organization.
- **Organization Identifier: com.appcoda** – It's actually the domain name written the other way round. If you have a domain, you can use your own domain * name. Otherwise,

you may use "com.appcoda" or just fill in "edu.self".

- **Bundle Identifier: com.appcoda.StackViewDemo** - It's a unique identifier of your app, which is used during app submission. You do not need to fill in this option. Xcode automatically generates it for you.
- **Language: Swift** – We'll use Swift to develop the project.
- **Devices: Universal** – Select "Universal" for this project. A universal app is a single app that is optimized for iPhone, iPod touch, and iPad devices. For this demo, we'll design a user interface that works on all devices.
- **Use Core Data: [unchecked]** – Do not select this option. You do not need Core Data for this simple project.
- **Include Unit Tests: [unchecked]** – Do not select this option. You do not need unit tests for this simple project.
- **Include UI Tests: [unchecked]** – Do not select this option. You do not need UI tests for this simple project.

Click "Next" to continue. Xcode then asks you where to save the StackViewDemo project. Pick a folder on your Mac. Click "Create" to continue.

Adding Images to the Xcode Project

As you see, the sample app includes three images. The question is how can you bundle images in Xcode projects?

In each Xcode project, it includes an asset catalog (i.e. Assets.xcassets) for managing images and icons that are used by your app. Go to the project navigator and select the `Assets.xcassets` folder. By default, it is empty with a blank AppIcon set. We are not going to talk about app icons in this chapter, but will revisit it after building a real-world app.

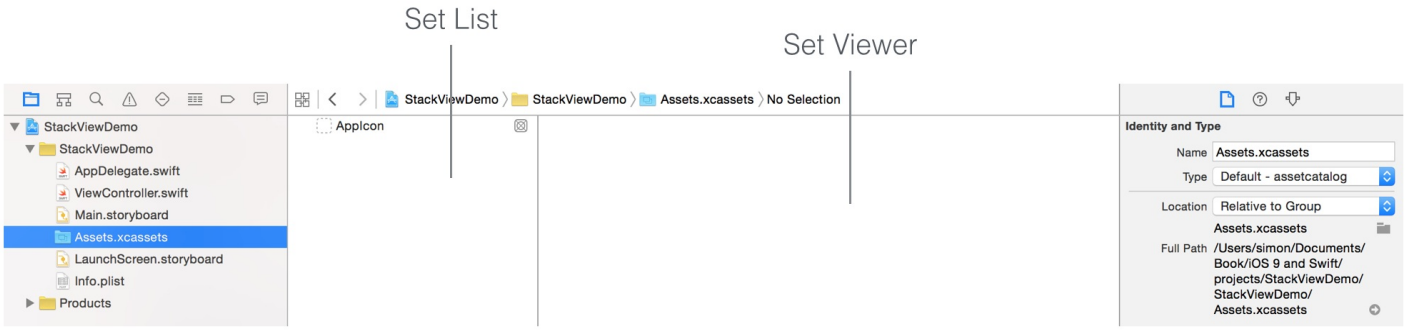


Figure 6-3. Asset Catalog

Now download this image set (https://www.dropbox.com/s/cl1det7u2jlzwxz/tshirts.zip?dl=0) and unzip it on your Mac. The zipped archive contains a total of 9 image files, but it actually includes three different t-shirt images. Each one of them comes with three different resolutions. Here is an example:

- t-shirt-1.png
- t-shirt-1@2x.png
- t-shirt-1@3x.png

When developing an iOS app, it is recommended to prepare three versions of an image. The one with @3x suffix, which has the highest resolution, is for iPhone 6 Plus. The one with @2x suffix is for iPhone 4/4s/5/5s/6, while the one without the @ suffix is for older devices with non-Retina display.

```
Credit: The images are provided by pixeden.com.
```

To add the images to the asset catalog, all you need to do is drag the images from Finder, and drop them into the set list or set viewer.

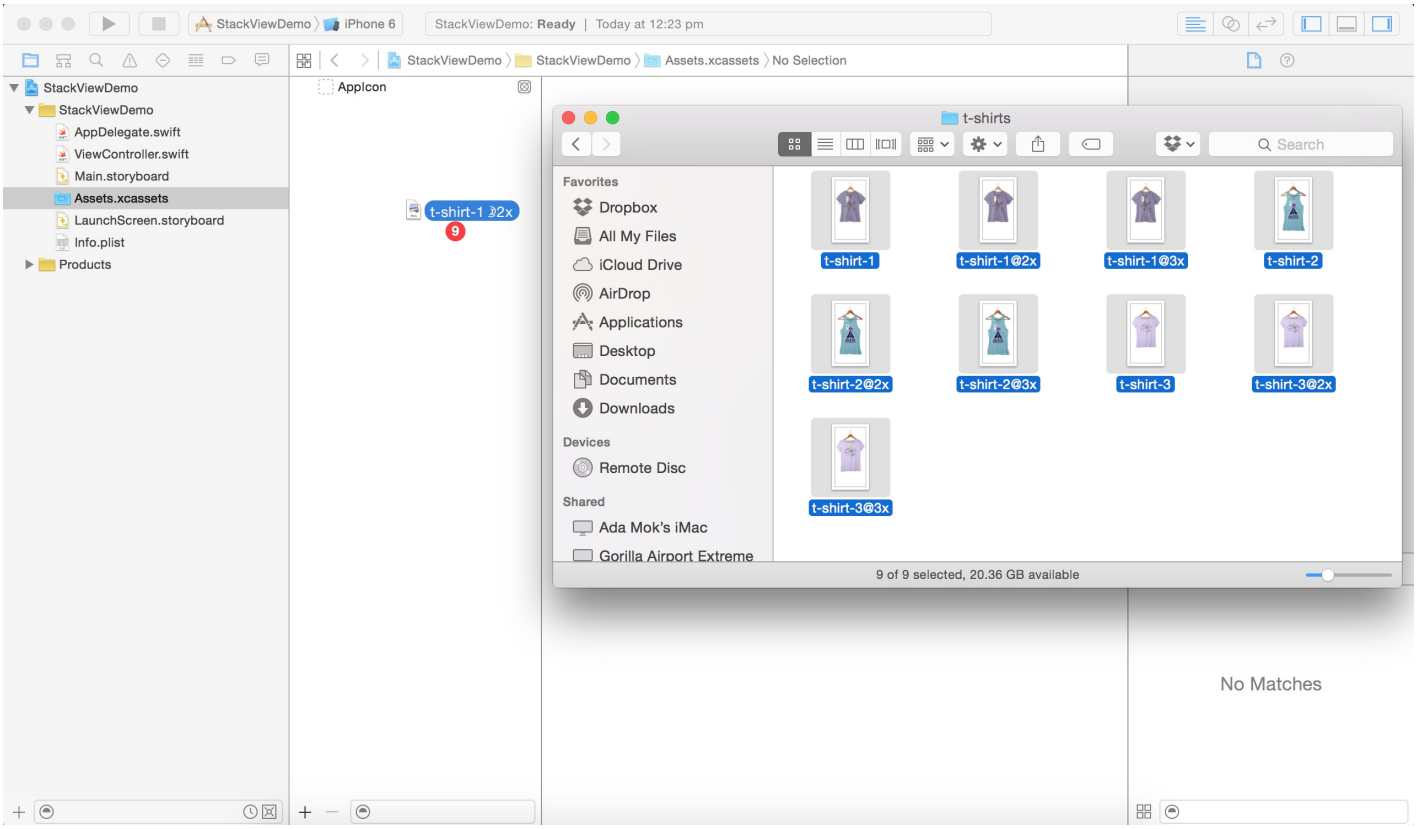


Figure 6-4. Adding images to the asset catalog

Once you add the images to the asset catalog, the set view automatically organizes the images into different wells. Later, to use the image, you just need to use the set name of a particular image (e.g. t-shirt-1). You can omit the file extension and you don't have to worry about which version (@2x/@3x) of the image to use. All these are handled by iOS accordingly.

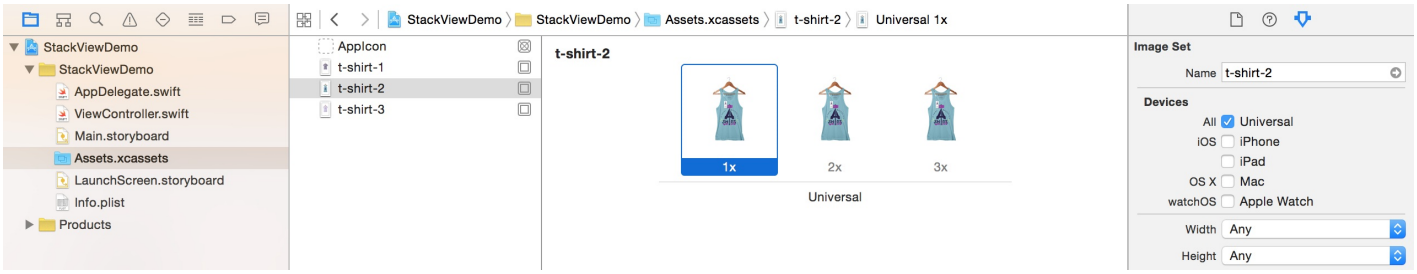


Figure 6-5. Images are automatically organized

Layout the Title Labels with Stack Views

Now that you've bundled the necessary images in the project, let's move onto stack views. First, open `Main.storyboard`. We'll start with the layout of these two articles.

Figure 6-6. Title and subtitle labels for the demo app

Stack view can arrange multiple views (known as arranged views) in both vertical and horizontal layouts. So first, you have to decide whether you want to use a vertical or horizontal stack view. The title and subtitle labels are arranged vertically. Obviously, we will use a vertical stack view.

From the Object library, drag a Vertical Stack View object to the view controller in the storyboard.

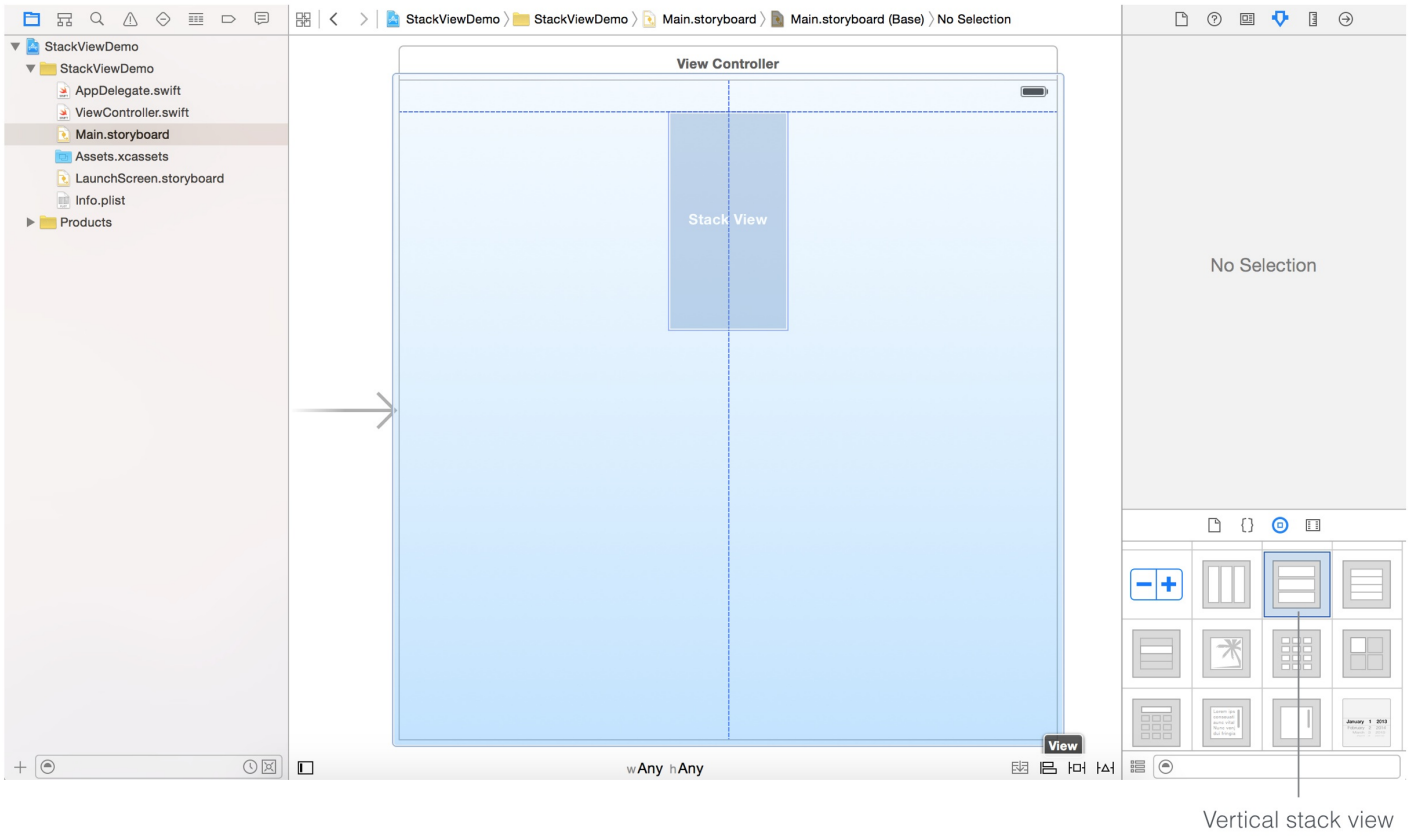


Figure 6-7. Adding a vertical stack view to the view controller

Next, drag a label from the Object library and put it right into the stack view. Once you drop the label into the stack view, the stack view automatically embeds it and resizes itself to fit the label. Double click the label and change the title to "Wardrobe". In the Attributes inspector, increase the font size to 50 points.

Now, drag another label from the Object library to the stack view. As soon as you release the label, the stack view embeds the new label and arranges the two labels vertically like this:

Figure 6-8. Embedding two labels in a vertical stack view

Edit the title of the new label and change it to "Keep your wardrobe organized".

Currently, the "Wardrobe" label is aligned to the left. Once you have a stack view, you can alter a number of the stack view's properties to change its appearance. Select the stack view and you can find its properties in the Attributes inspector.

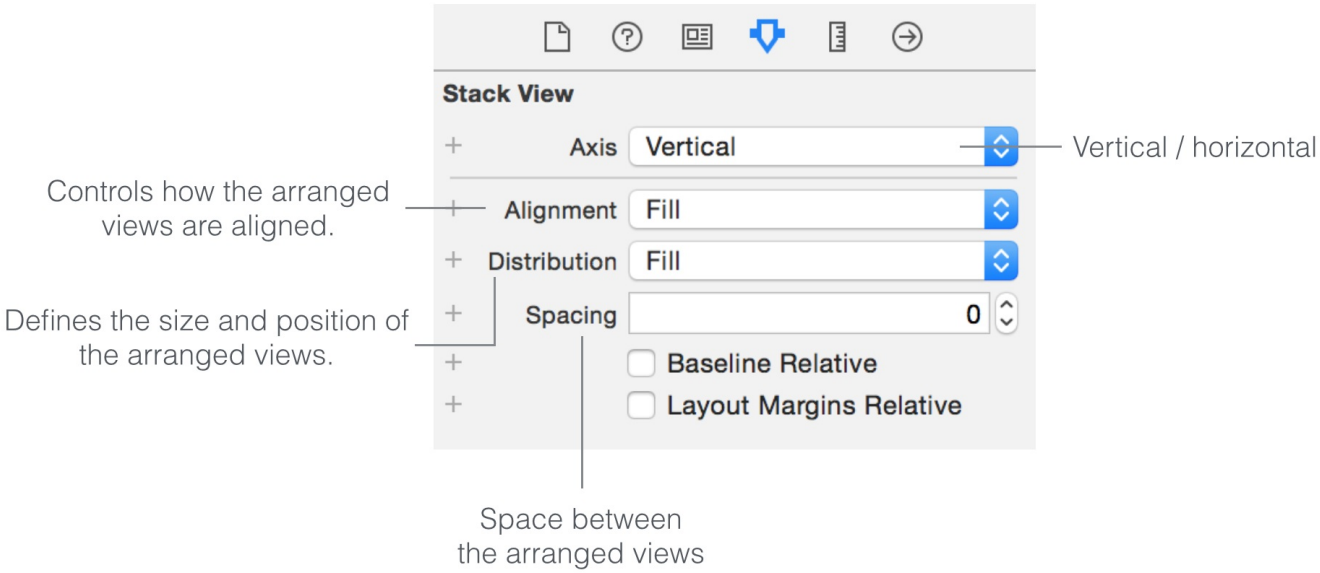


Figure 6-9. Sample properties of a stack view

As a sidenote, if you have any problem selecting the stack view, you can hold the shift key and right click the stack view. Interface Builder will then show you a shortcut menu for selection.

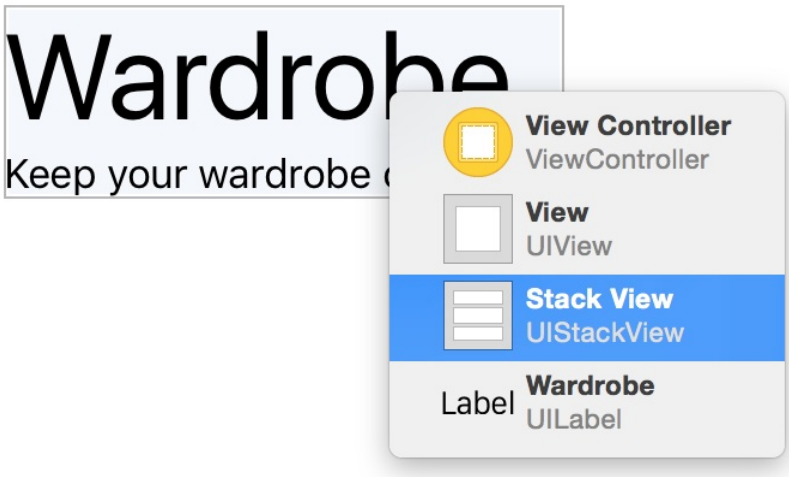


Figure 6-10. Sample properties of a stack view

Let's briefly talk about each property of the stack view. The *axis* option indicates whether the arranged views should be layout vertically or horizontally. By changing it from vertical to horizontal, you can turn the existing stack view to a horizontal stack view. The *alignment* option controls how the arranged views are aligned. For example, if it is set to Leading, the stack view aligns the leading edge (i.e. left) of its arranged views along its leading edge. The *distribution* option defines the size and position of the arranged views. By default, it is set to Fill. In this case, the stack view tries its best to fit all subview in its available space. If it is set to Fill Equally, the vertical stack view distributes both labels equally so that they are all the same size along the vertical axis. Figure 6-11 shows some sample layouts of different properties.

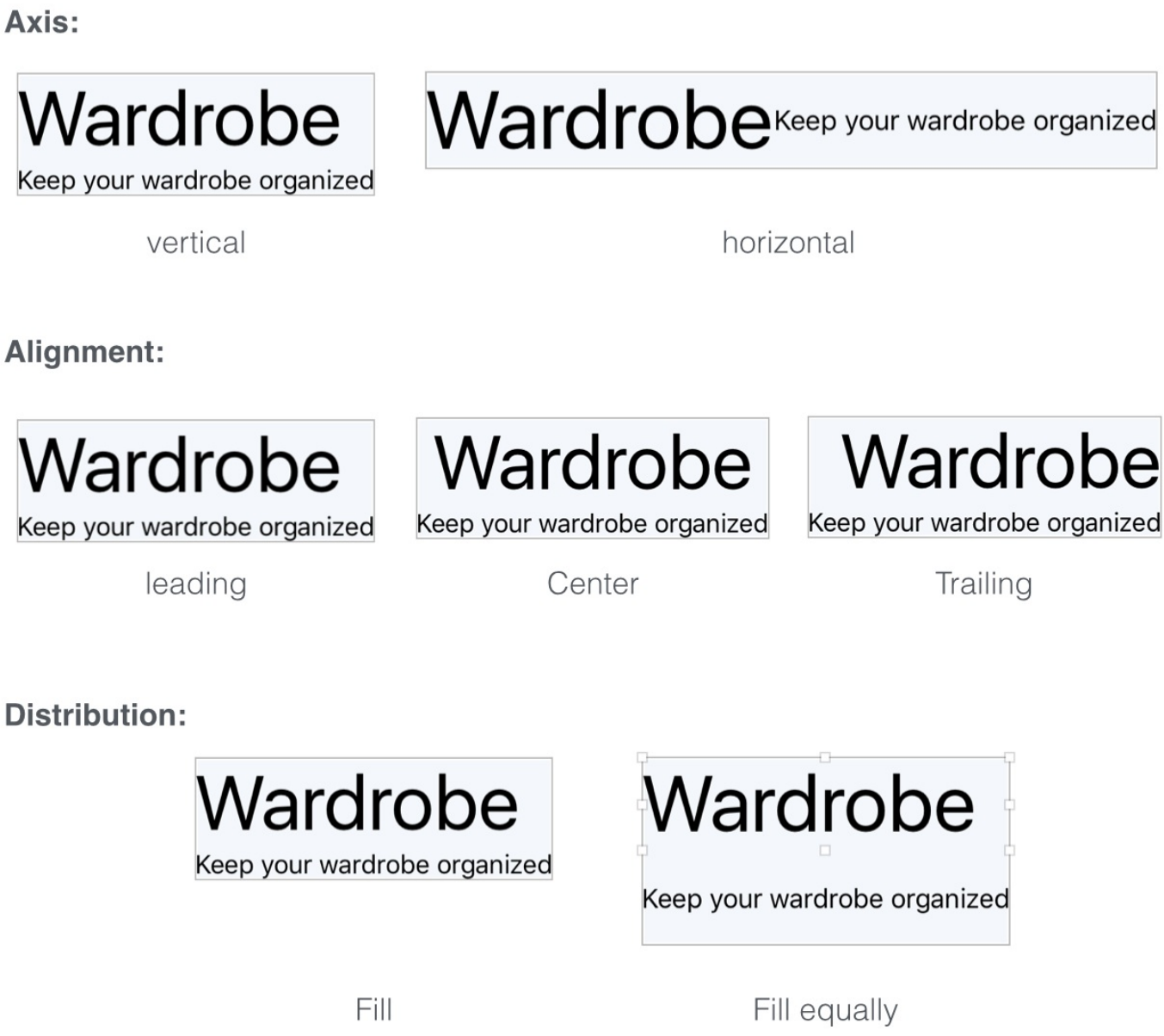


Figure 6-11. A quick demo of the stack view's properties

For our demo app, we just need to change the alignment option from *Fill* to *Center* and keep other options intact. This should center both labels. Before moving onto the next section, make sure you position the stack view correctly. You can select the stack view and go to the Size inspector. Ensure you set the value of X to 183 and Y to 46.

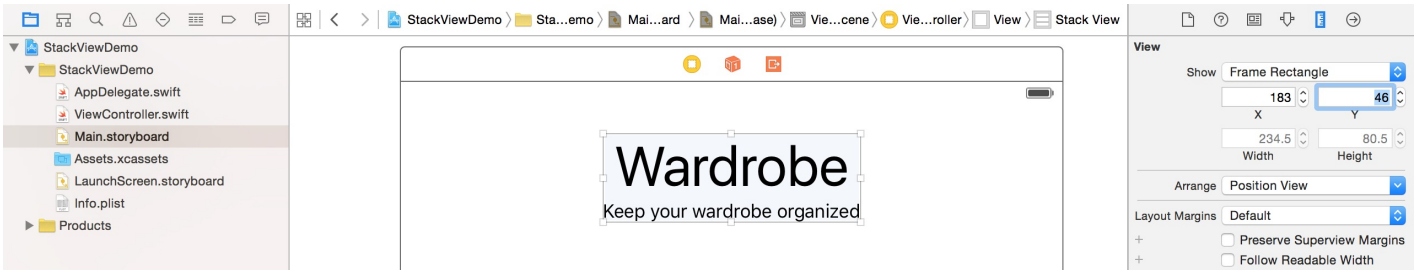


Figure 6-12. Verify the position of the stack view

Layout the Images Using Horizontal Stack Views

Now we're going to lay out the T-shirt images. As the images are arranged horizontally, we use horizontal stack views instead. Drag a horizontal stack view from the Object library to the storyboard. In iOS, we use image views to display images. From the Object library, look for the image view object, and drag it into the stack view we just added. Repeat the procedures to add two more image views. As you add more image views to the stack view, it automatically arranges the image views horizontally.

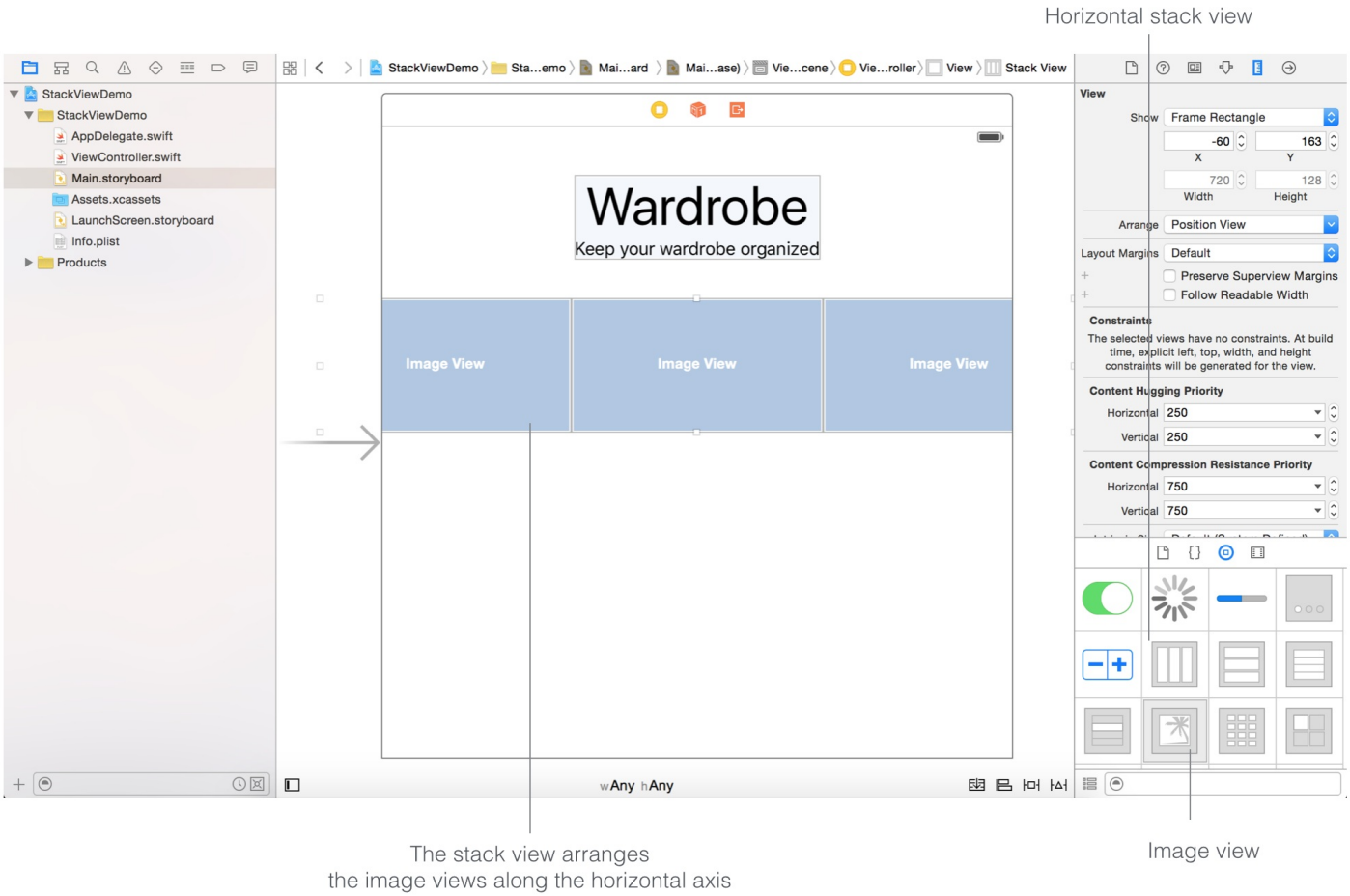


Figure 6-13. The stack view arranges the image view horizontally

The image view is not assigned with any image yet. Select the image view on the left and go to the Attributes inspector. Recalled that we have added our T-shirt images to the project, the *image* option automatically loads the image names from the asset catalog for your selection. Click the dropdown menu of the image option and choose "t-shirt-1". Repeat the same steps for the rest of the image views but set the image to "t-shirt-2" or "t-shirt-3" instead. Your layout should look like this:

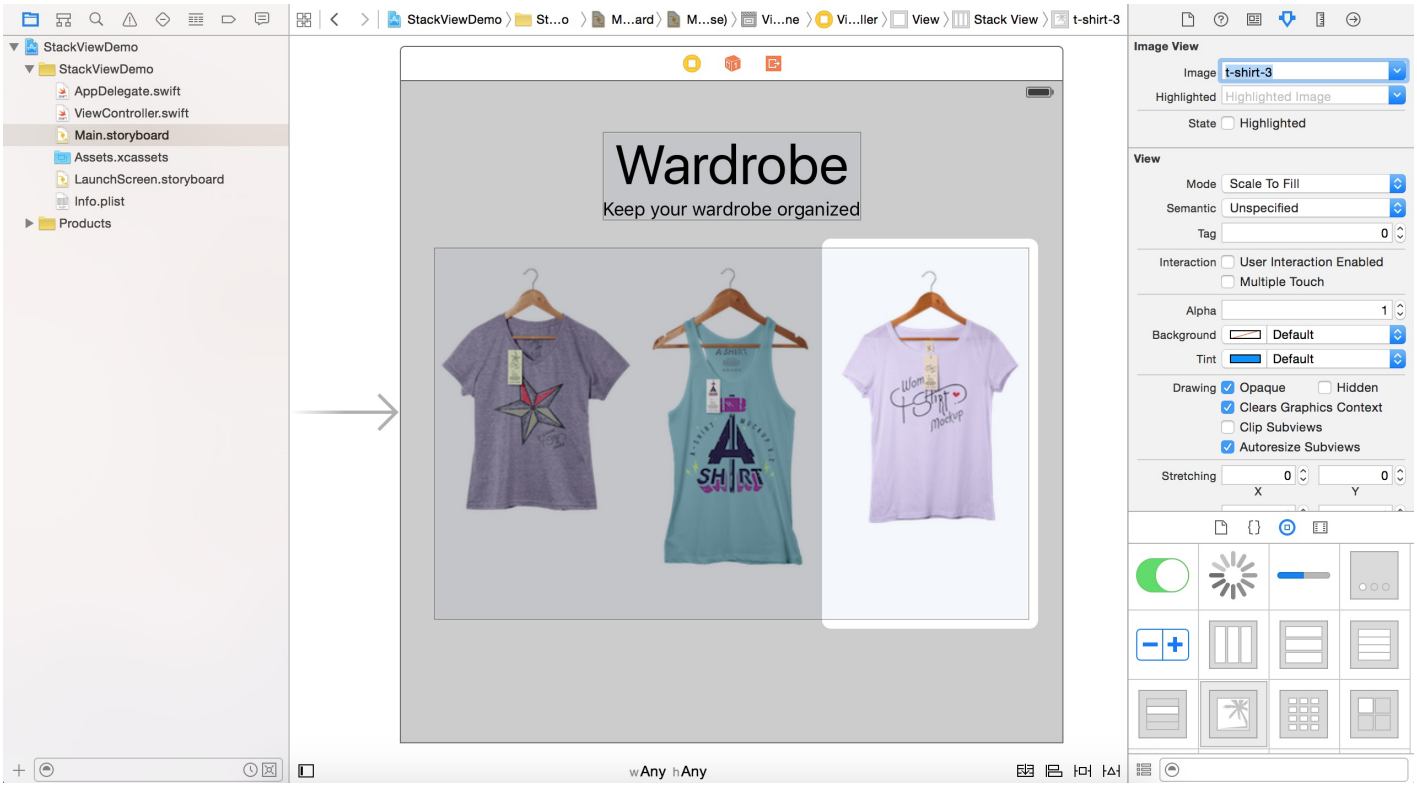


Figure 6-14. Assign the images to the image views

Make sure you position the stack view at X=20 and Y=142. Lastly, select the stack view and change its distribution from *Fill* to *Fill Equally*. Also, set the spacing to 10. This will add some spacing between the image views.

```
Quick note: It seems that the stack view has already distributed the image
views equally. Why do we need to change the distribution to Fill Equally?
Remember that you're now designing the UI on a freeform canvas. If you do not
explicitly set the distribution to Fill Equally, iOS will layout the image
views using Fill distribution policy. It may not look good on other screen
sizes.
```

The UI looks good now, right? If you run the app or preview the UI in Interface Builder, it doesn't look as expected. The reason is that we haven't defined the layout constraints for the stack views. As I mentioned at the very beginning of the chapter, stack views only save you from defining the layout constraints of the arranged views. You still need to create layout constraints for the stack views.

Adding Layout Constraints for the Stack Views

For the stack view with titles, we will define the following layout constraints:

- Set a spacing constaint between itself and the top layout guide.
- Center it horizontally, with respect to the container view.

To do that, control-drag vertically from the stack view to the container view. Now hold the *shift* key, and choose both "Vertical Spacing to Top Layout Guide" and "Center Horizontally in Container". The selected constraints are indicated by a checkmark. Hit Return to add the constraints.

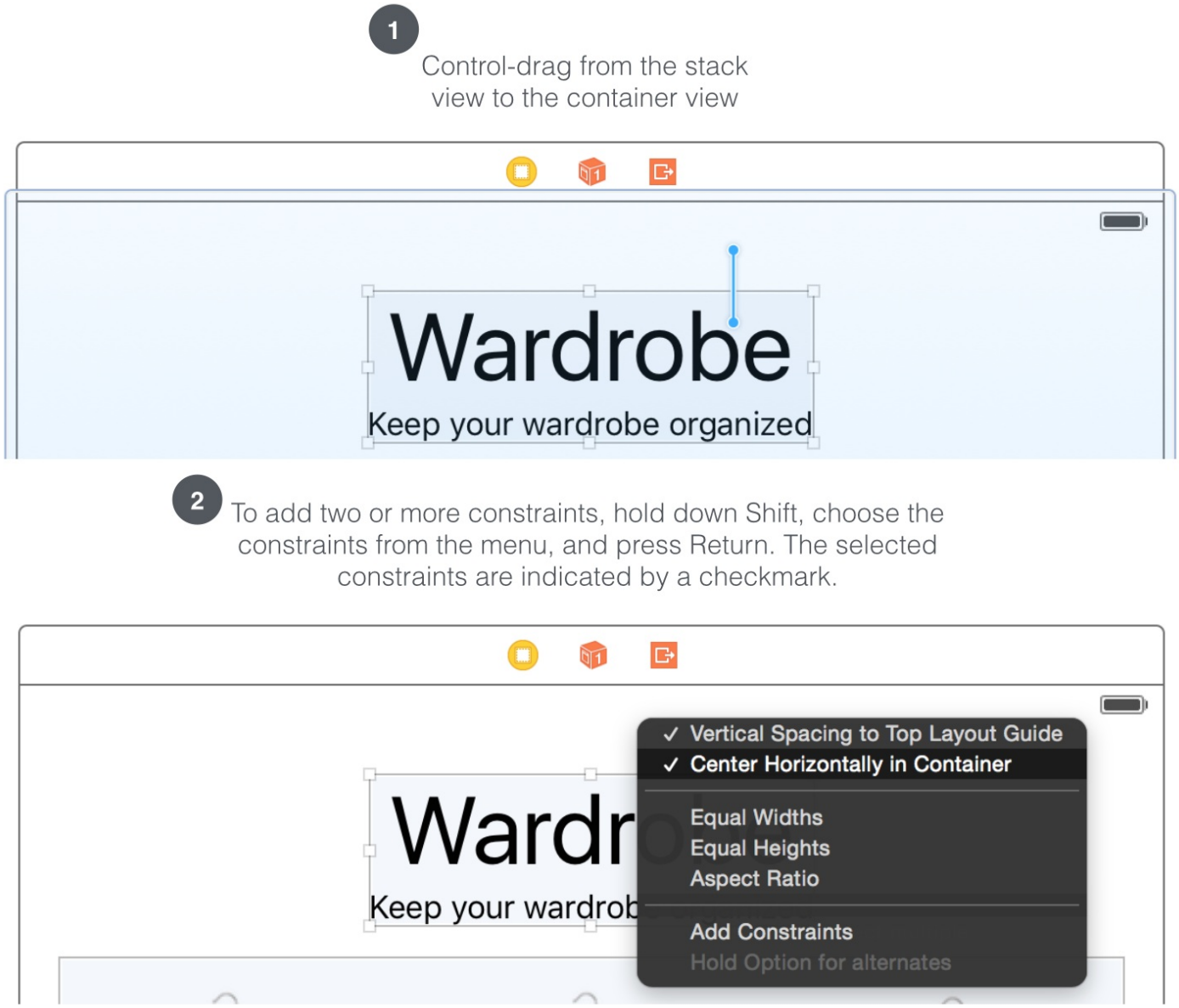


Figure 6-15. Adding multiple constraints using Control-drag

For the stack view containing the image views, we will define the following layout constraints:

- Set a spacing constraint between itself and the other stack view, so that there is a space between them.
- Set a spacing constraint between left side of the stack view and the left margin of the view, so that there is no space (i.e. 0 point) between them.
- Set a spacing constraint between right side of the stack view and the right margin of the

view, so that there is no space (i.e. 0 point) between them.

Now click the Pin button in the layout button. Set the space constraints of the top, left and right side to 15.5, 0 and 0 respectively. When the constraint is enabled, it is indicated by a solid red bar. Then click the "Add 3 Constraints" button to add the constraints.

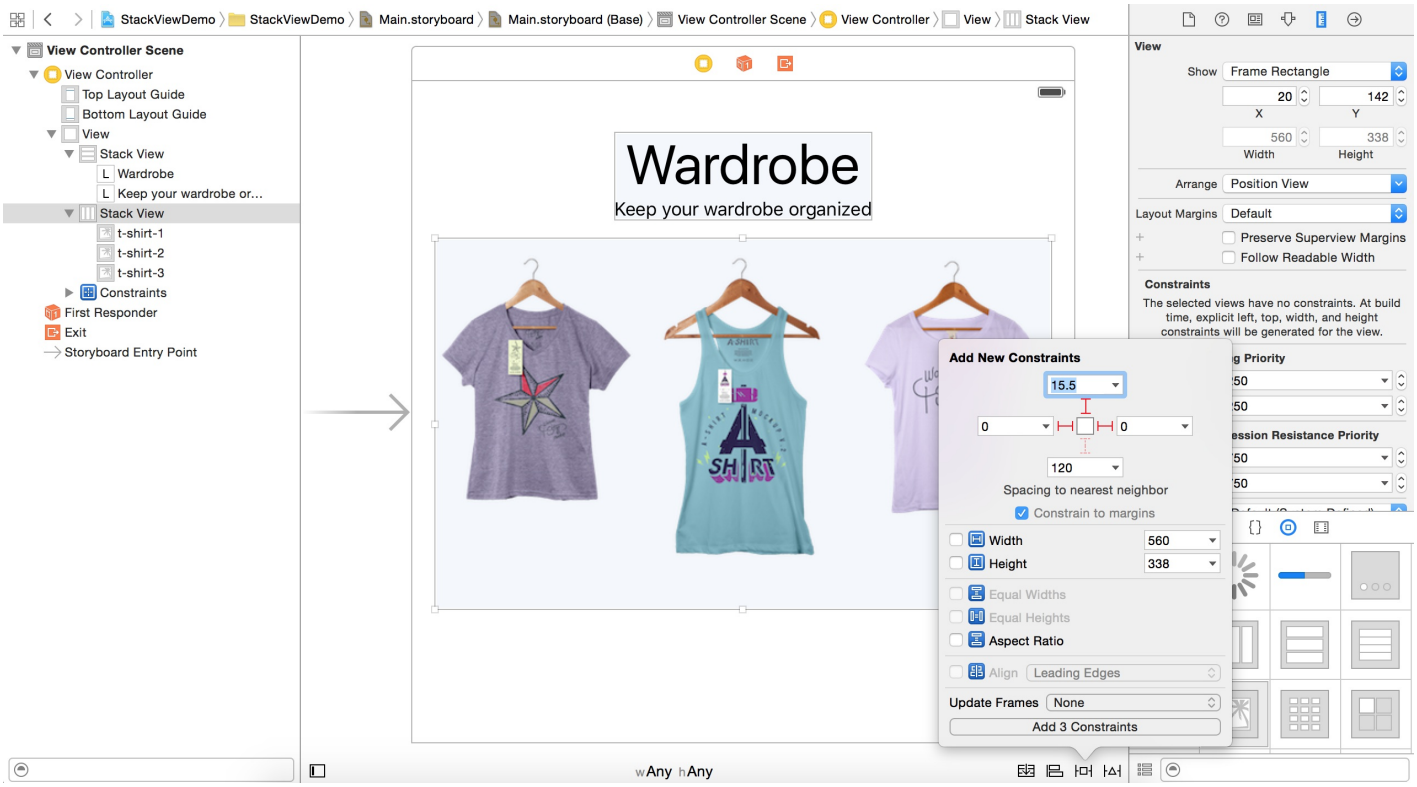


Figure 6-16. Adding spacing constraints using Pin button

If you run the app in the simulator, the layout looks better than the previous version. However, all images appear vertically stretched.

Figure 6-17. Images appear all vertically stretched

Currently, the height of the stack view is fixed, while the width can be varied. Because the screen width of the actual iPhone is smaller than the canvas, this is why the images appear all vertically stretched.

To fix the issue, we have to define one more constraint and ask the stack view to retain the aspect ratio, regardless of the screen sizes. In the document outline view, control-drag on the stack view (containing the image views). In the shortcut menu, select "Aspect Ratio".

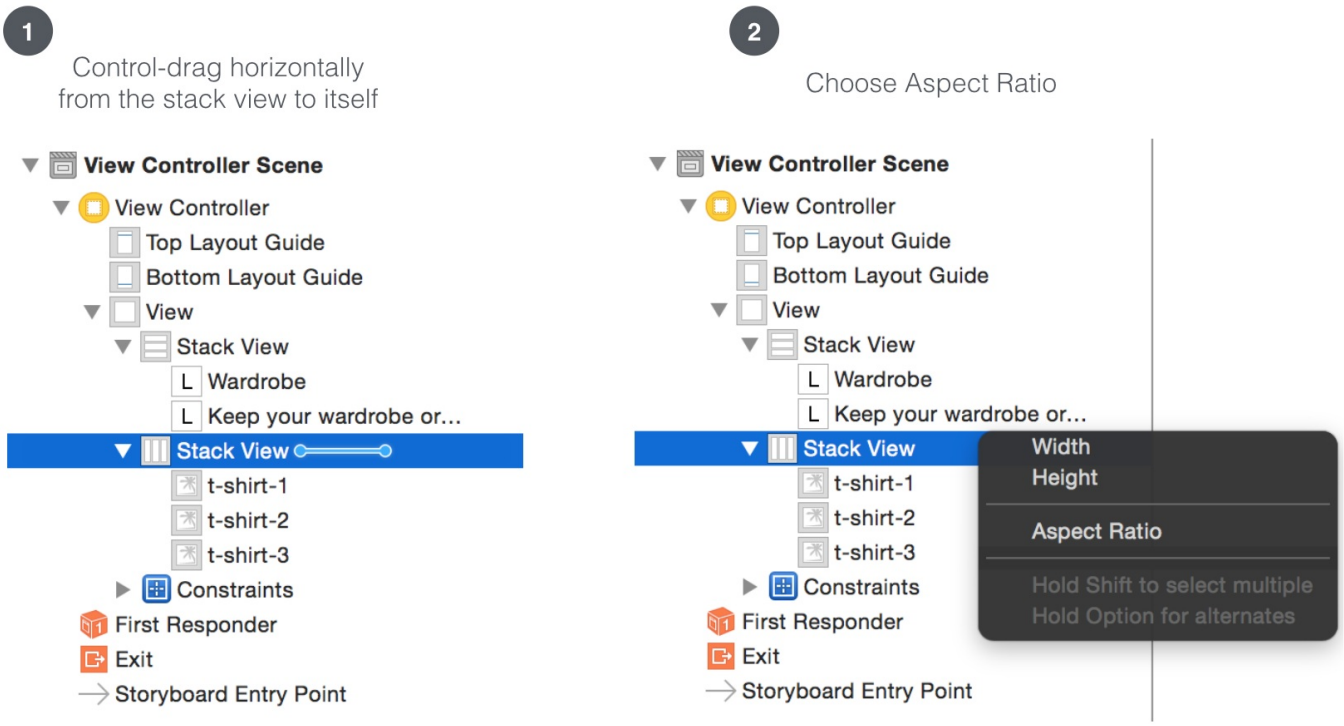


Figure 6-18. Control-drag from the stack view to itself to add a constraint

Run the project again. The layout should now look great.

Layout the Buttons Using Stack View

We haven't finished yet. Let's continue to layout the two buttons at the bottom of the screen. At the very beginning of this tutorial, I mentioned that there are two ways to use stack views. Earlier, you added a stack view from the object library. Now I will show you another approach.

First, drag a button from the Object library to the view. Double click the button to name it "Sign in". In the Attributes inspector, change its background color to *red* and text color to *white*. In the Size inspector, set the width to 200. Next, drag another button to the view and name it "Sign up with email". In the Attributes inspector, change its background color to *red* and text color to *white*. In the Size inspector, set the width to 200.

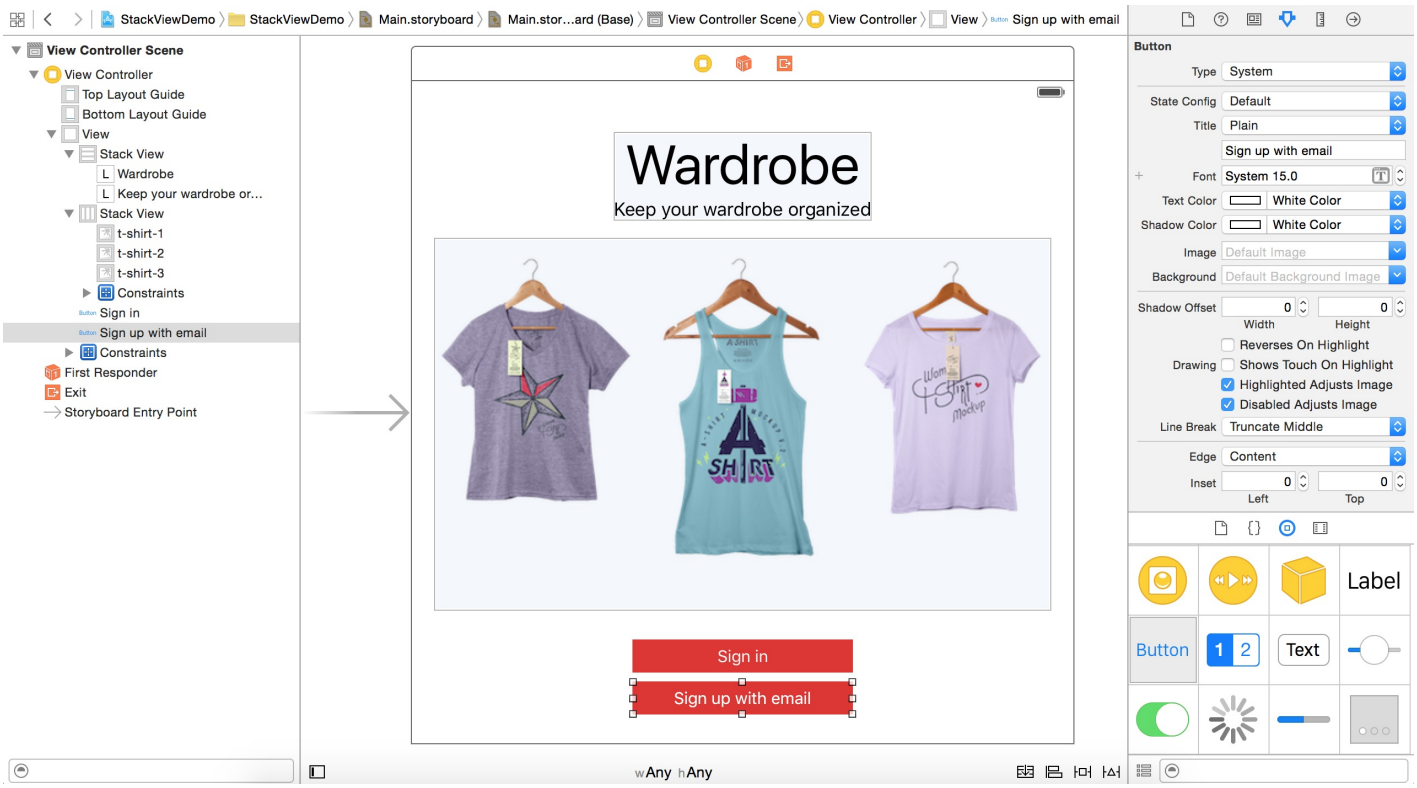


Figure 6-19. Adding two buttons to the view

Again, you do not need to set the layout constraints for these labels. Let the stack view to do the magic for you. Hold the command key to select both buttons, and then click the Stack button in the layout bar. Interface Builder automatically embeds both buttons in a vertical stack view. To add a space between the buttons, select the stack view. Under the Attributes inspector, set the value of*spacing* to 10. Try to center the stack view horizontally and

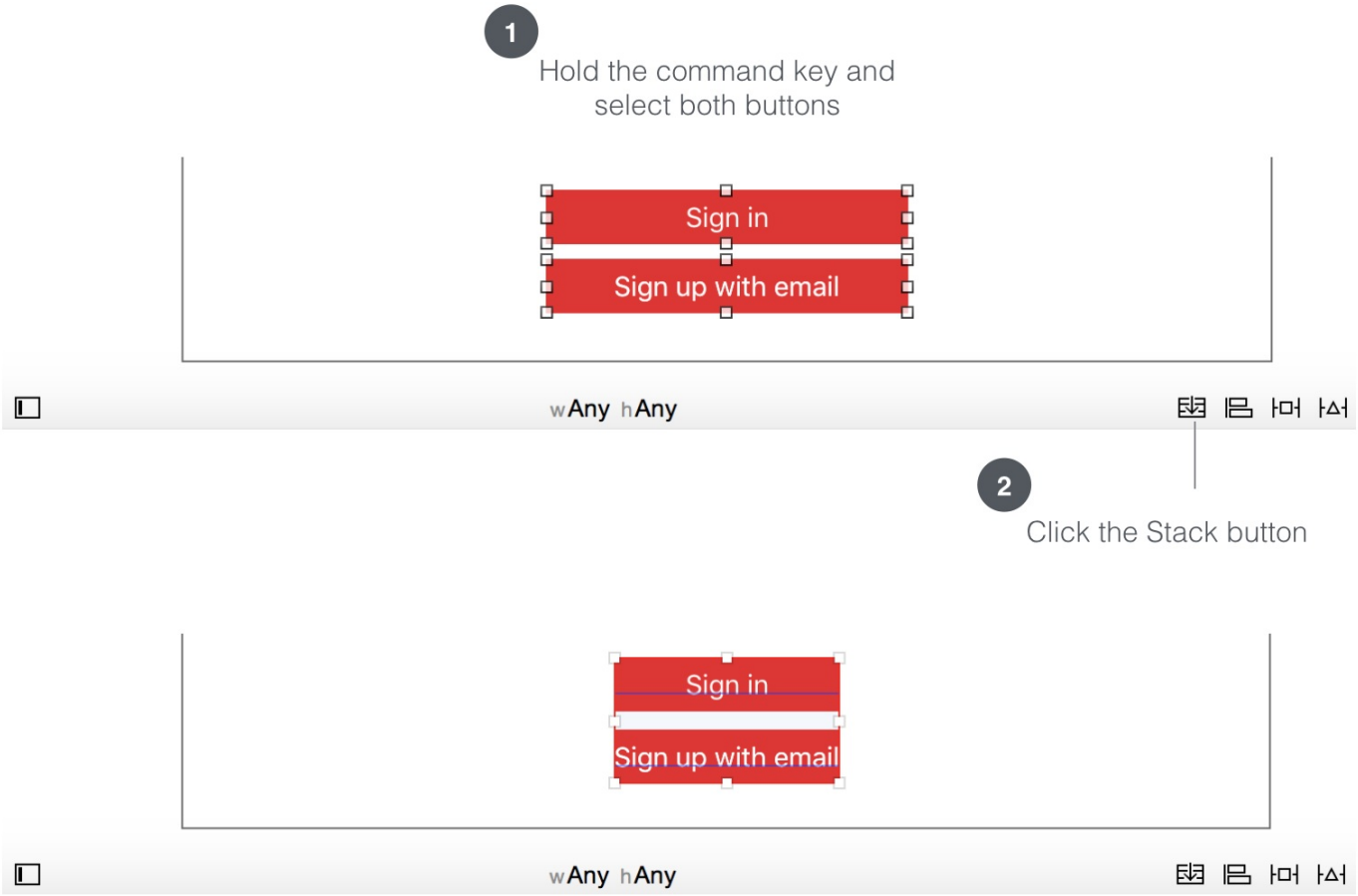


Figure 6-20. Embedding the buttons in a stack view

Similarly, we have to define layout constraints for this stack view so that it is positioned close to the bottom of the view. Here are the layout constraints we're going to define:

- Center the stack view horizontally, with respect to the container view.
- Set a spacing constraint so that there is a space between the stack view and the Bottom Layout Guide.

You can follow the steps as illustrated in figure 6-21 to add the constraints.

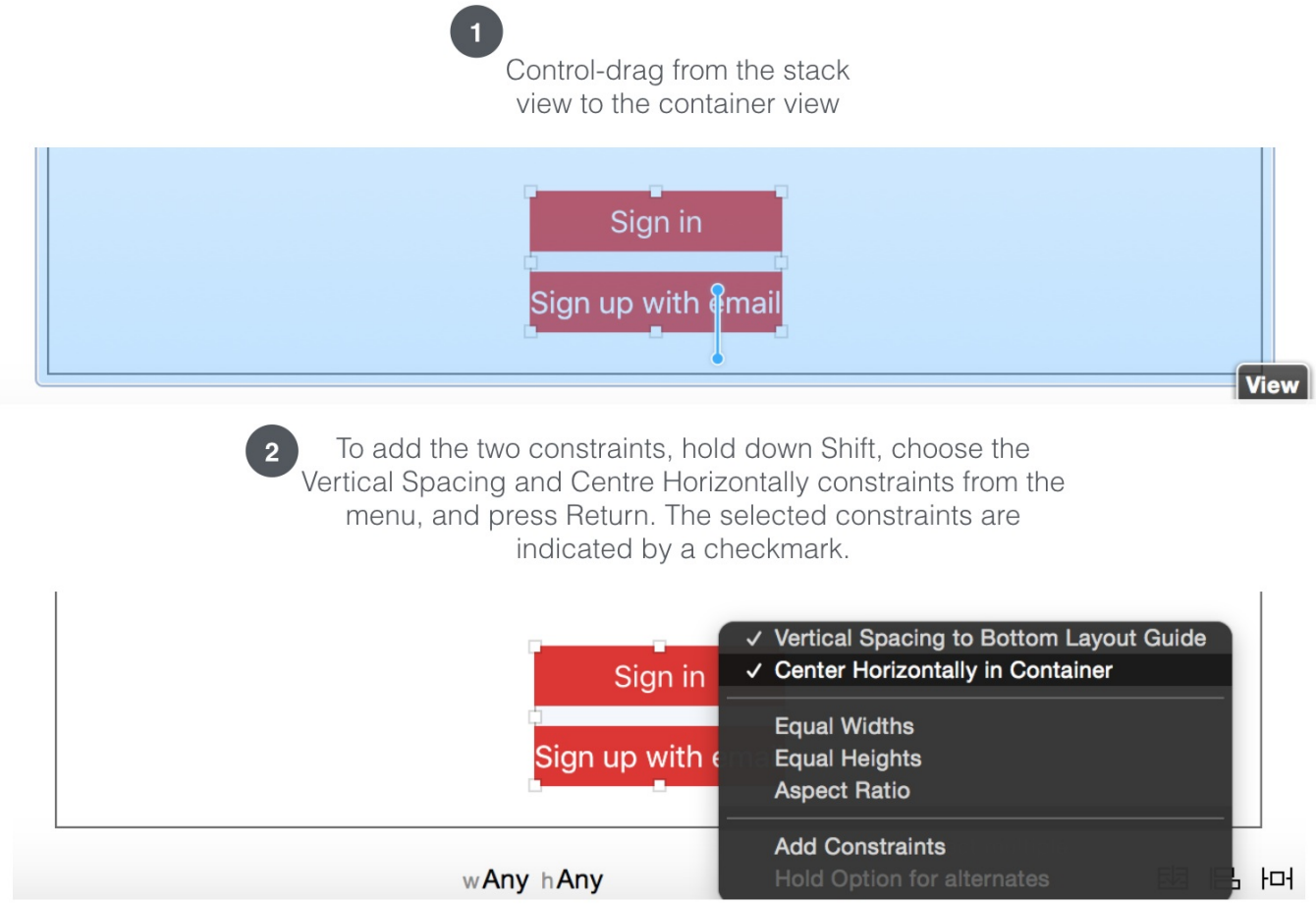


Figure 6-21. Adding constraints to the "Button" stack view

One thing you may notice is that the button is narrower than expected. As you embed the buttons in the stack view, they are resized. If you want to set the width to 200, you will need to add a width constraint to the stack view. In the document view, control-drag from the stack view to itself and choose *Width* to add the constraint.

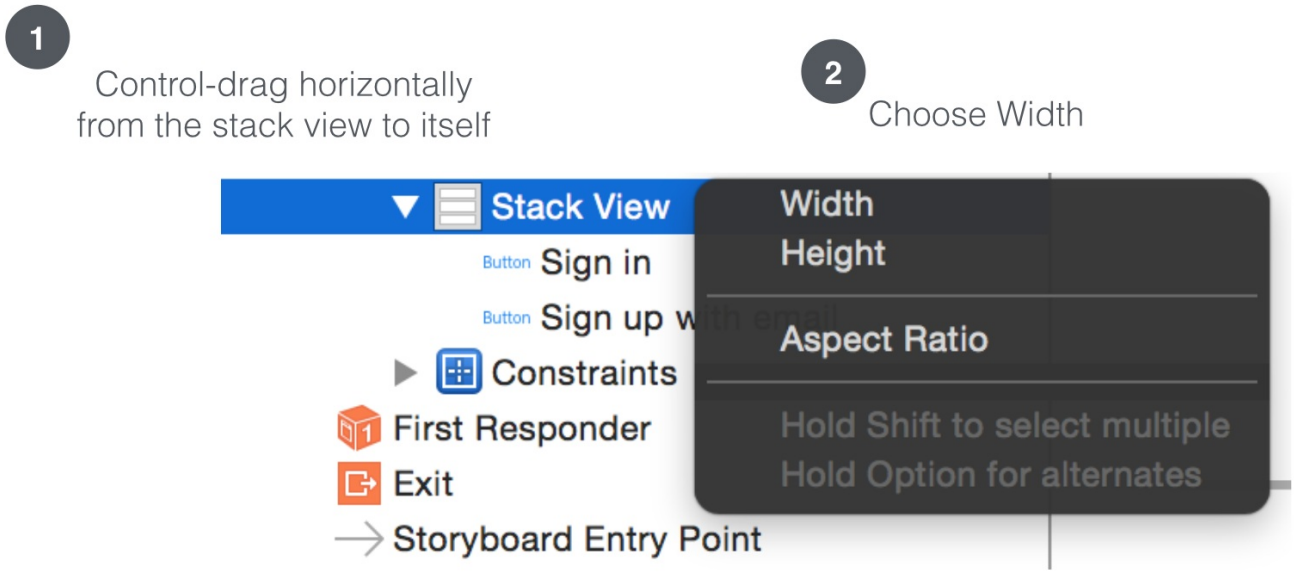


Figure 6-22. Adding the width constraint

To change the width to 200 points, select the width constraint in the document outline view. Then go to the Size inspector, and change the constant to 200. The stack view now has a fixed width of 200 points.

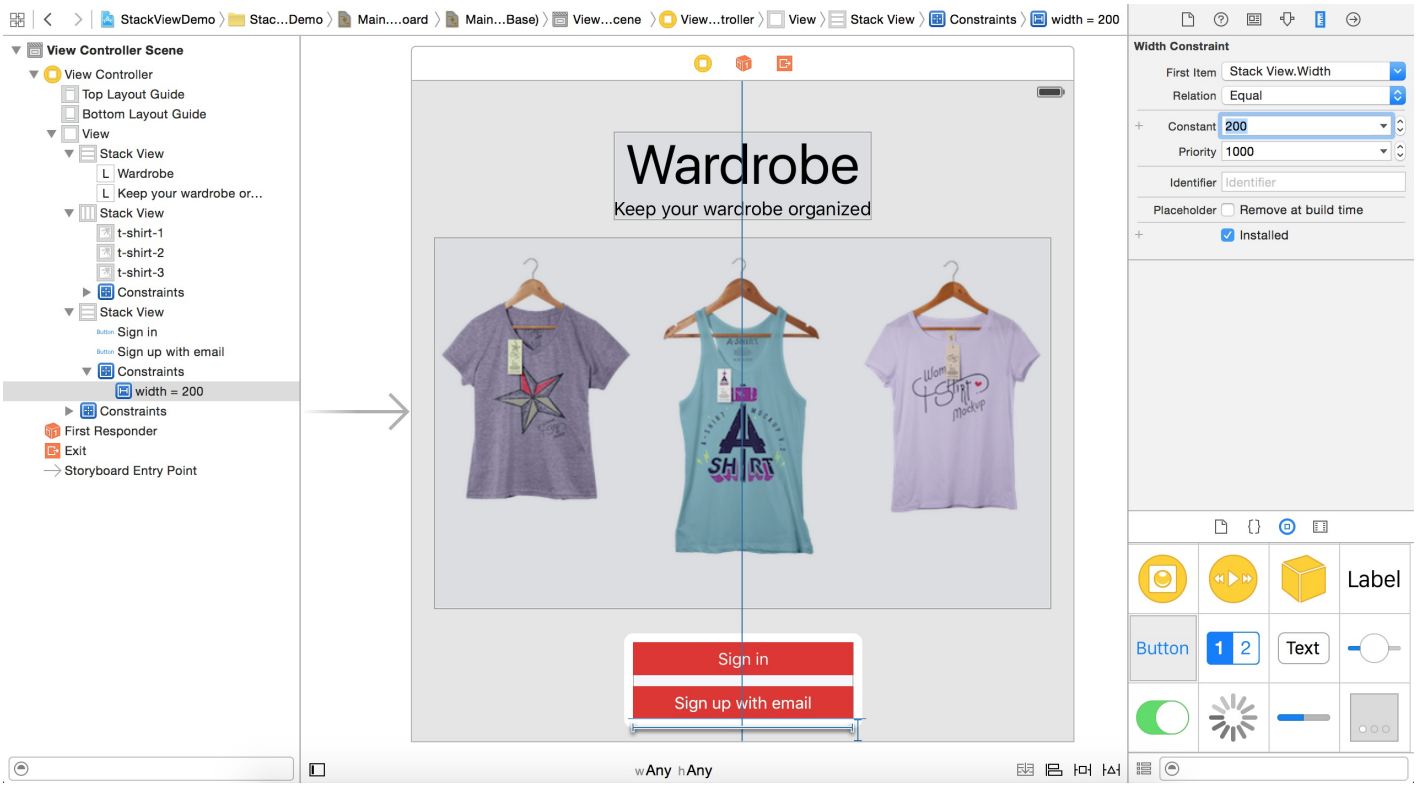


Figure 6-23. Editing the width constraint

It's time to test the app again. Run the project on iPhone (or iPad). Your UI should look like figure 6-24.

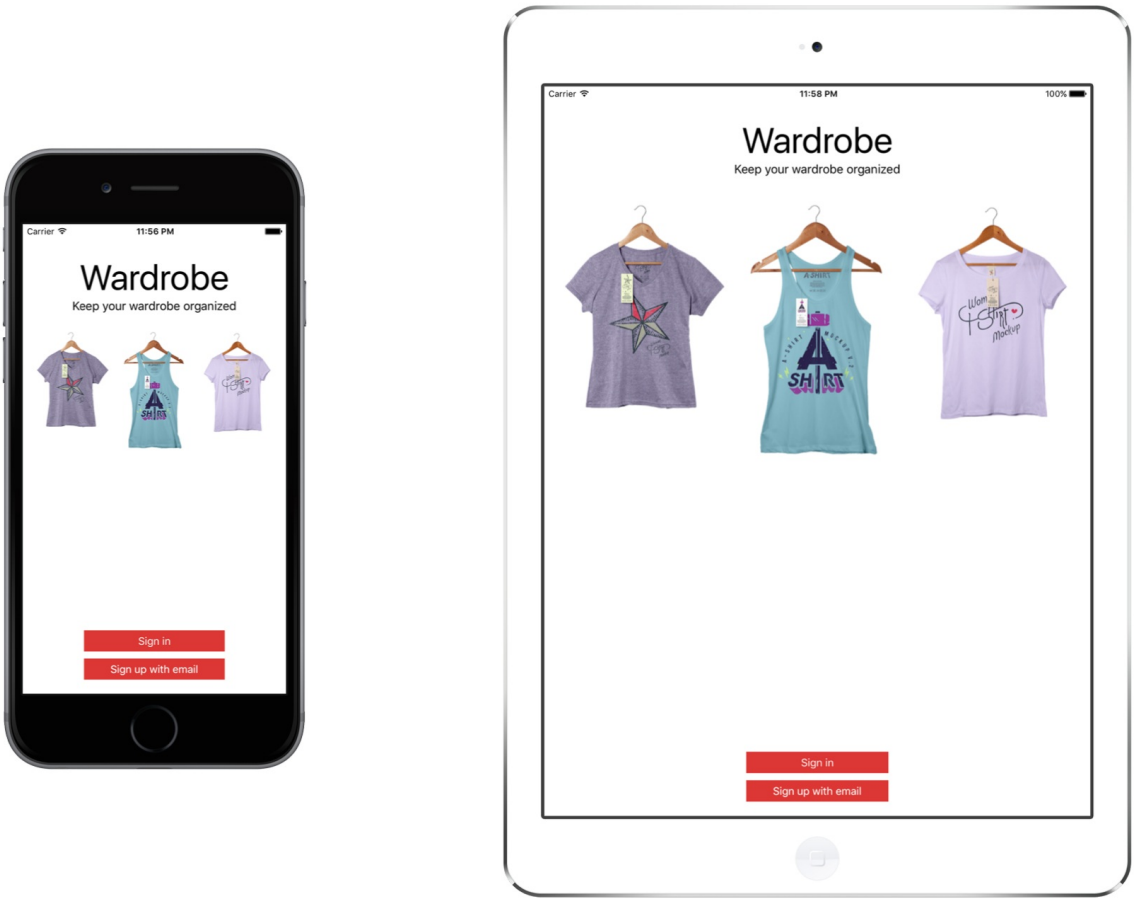


Figure 6-24. The app UI on both iPhone 6 and iPad Air

Adapting Stack Views Using Size Classes

If you turn the iPhone simulator sideway, the UI in landscape orientation looks like this:

Figure 6-25. The app UI in landscape orientation

It works pretty great. To make it look even better, I want to change the spacing between the Top Layout Guide and the titles, as well as, the spacing between the titles and the images. But please note that these changes only apply to iPhones in landscape orientation.

As you know, we can adjust the spacing by changing the constant of the spacing constraints. The real question is: how can you apply the change for iPhones in landscape orientation only?

This leads to a new UI design concept known as *Adaptive Layout*, introduced since the release of iOS 8. With adaptive layout, your apps can adapt their UI to a particular device and device orientation. To achieve adaptive layout, Apple introduced a new concept, called **Size Classes**. This is probably the most important aspect which makes adaptive layout possible. Size classes are an abstraction of how a device is categorized depending on its screen size and orientation. You can use both size classes and auto layout together to design adaptive user interfaces.

A size class identifies a relative amount of display space for both vertical (height) and horizontal (width) dimensions. There are two types of size classes: *regular* and *compact*. A regular size class denotes a large amount of screen space, while a compact size class denotes a smaller amount of screen space. By describing each display dimension using a size class, this

will result in four abstract devices: *Regular width-Regular Height*, *Regular width-Compact Height*, *Compact width-Regular Height* and *Compact width-Compact Height*.

The table below shows the iOS devices and their corresponding size classes:

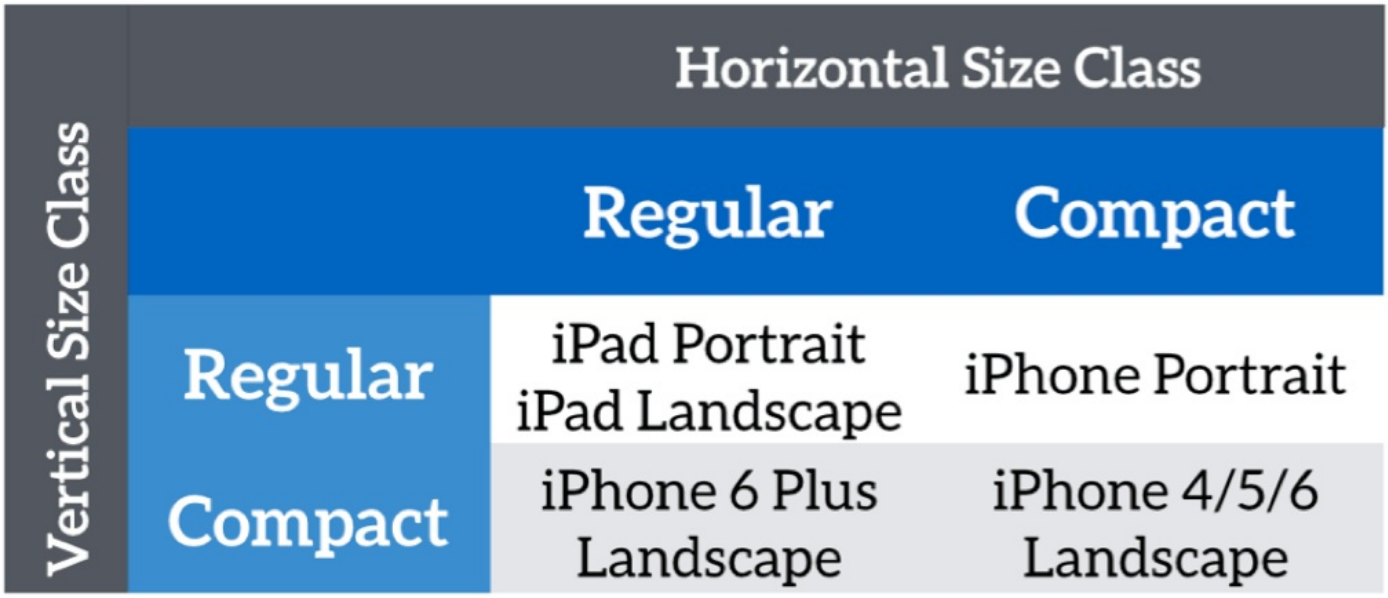


Figure 6-26. Size Classes

To characterize a display environment, you must specify both a horizontal size class and vertical size class. For instance, an iPad has a regular horizontal (width) size class and a regular vertical (height) size class. In our case, we want to provide layout specializations for iPhones in landscape orientation. In other words, here are the two size classes we have to deal with:

- Compact width-Compact height
- Regular width-Compact height

Now go back to the `Main.storyboard`. Select the spacing constraint between the Top Layout Guide and the stack view (holding the title labels). In the Size inspector, click the + button next to the *Constant* option. Select Any Width > Compact height, and then set the value of this size class to *0*.

Quick note: Here, any width includes both compact and regular width.

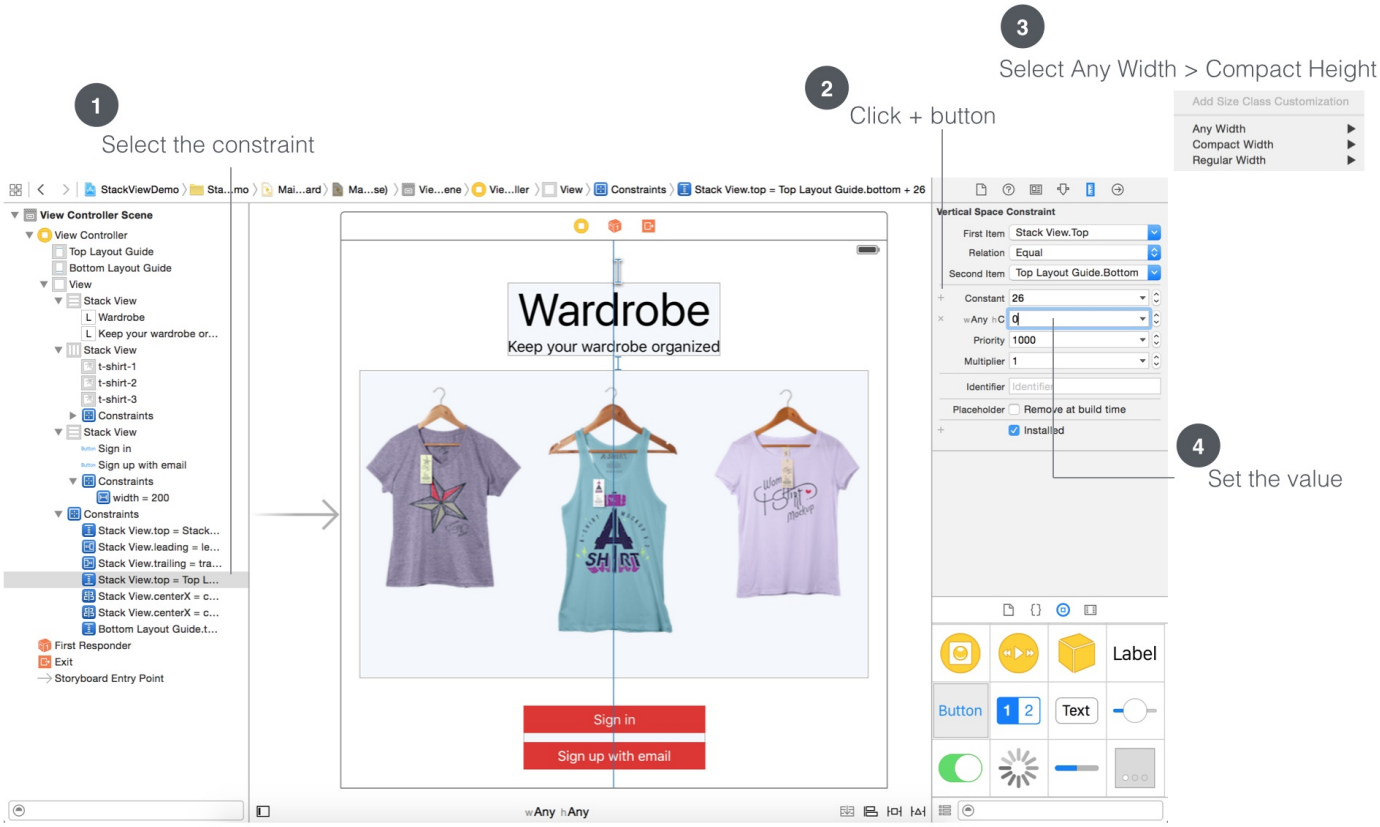


Figure 6-27. Add a Size-Class-specific constraint

By doing that, the space between the stack view and Top Layout Guide will be minimized when your iPhone is turned sideway. Run the project on various iOS devices and see the result.

You can apply the same procedure if you want to adjust the spacing between the titles and the images. I'll leave it as an exercise for you.

Figure 6-28. Wardrobe app in landscape orientation

Your Exercise

To help you better understand how size classes work, let's have another simple exercise. You are required to create another layout specialization for iPad devices. Instead of using a vertical stack view to layout the Sign in/Sign up buttons, please lay out the buttons horizontally on iPad.

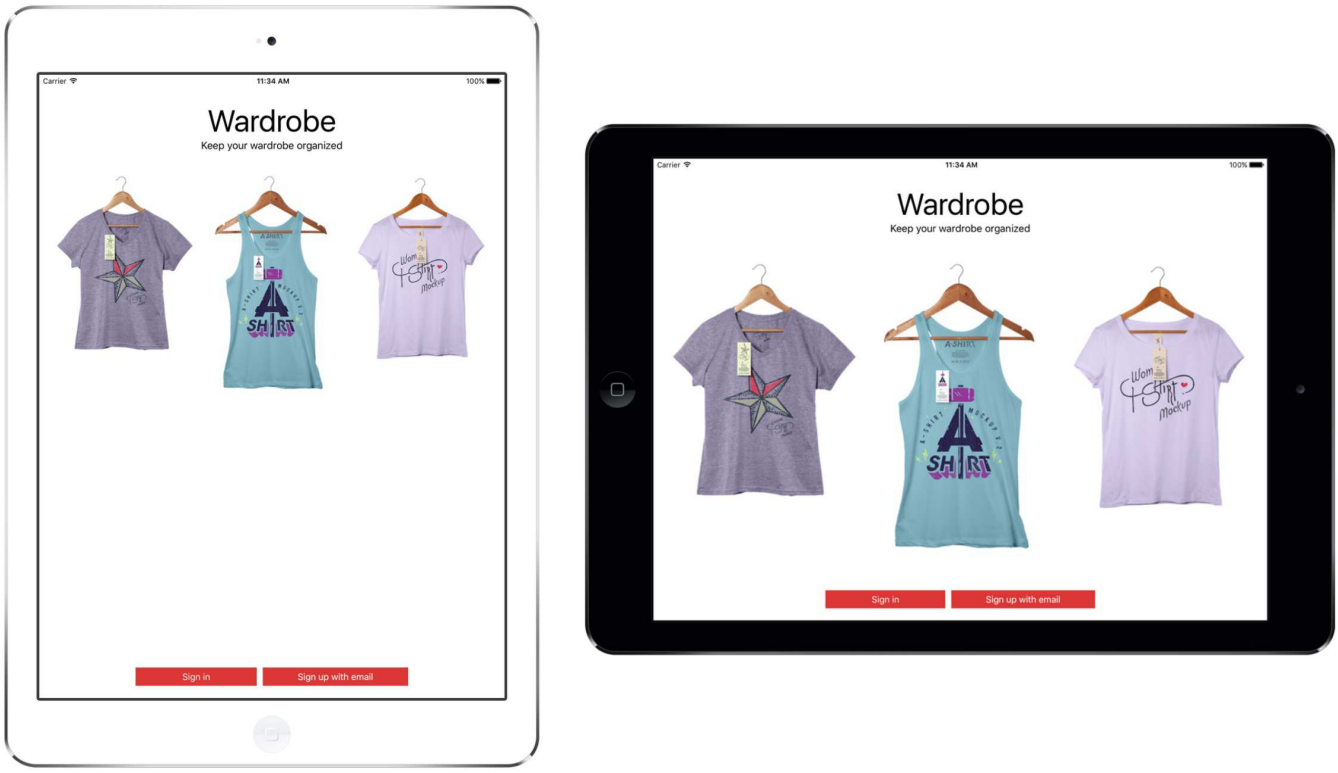


Figure 6-29. Sign in/Sign up buttons are arranged in horizontal direction (iPad only)

Hint: Refer to the size classes table to determine which size class to use, and change the axis of the stack view for that size class to horizontal. You will also need to customize the width constraint of the stack view.

Summary

In this chapter, I have given you an introduction to stack view and a demo on how to layout your UI using this new component. Stack views streamline the way you build user interfaces on iOS, with very minimal constraints. One question you may have is that when should you use stack views? Apple's engineers recommended developers to adopt stack views first, and then only when you need to actually use raw constraints. So throughout this book, we will design the user interfaces using stack views.