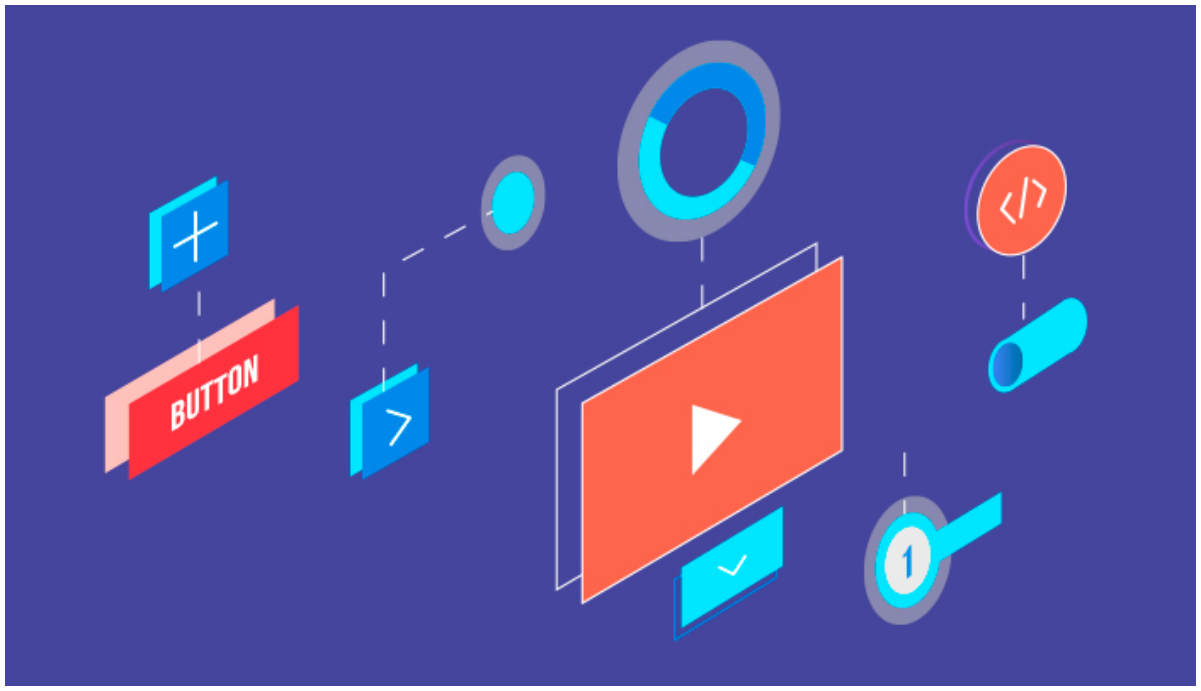
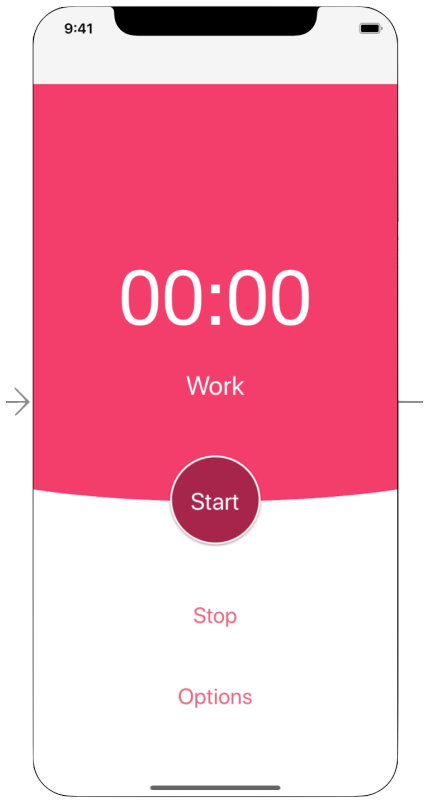
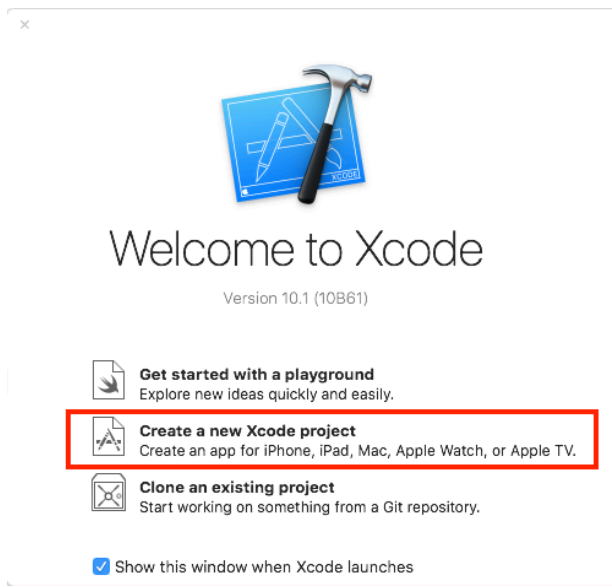
CHAPTER NINE : CREATE MAIN STORYBOARD



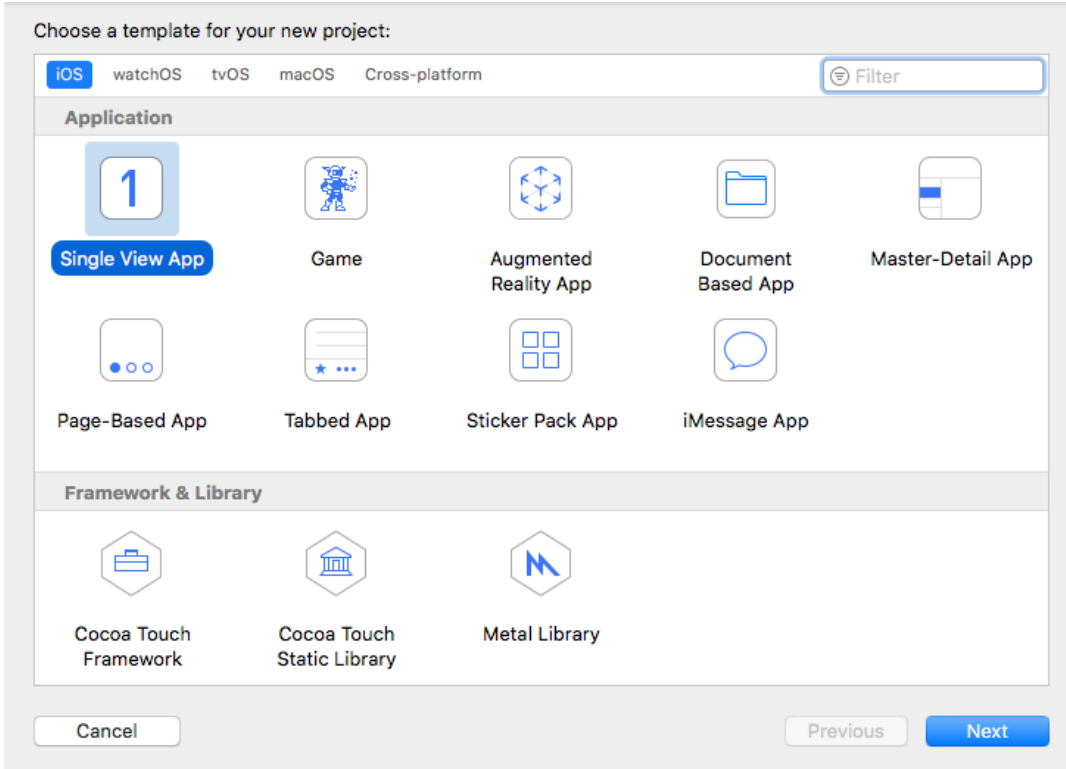


I. CREATE A NEW PROJECT

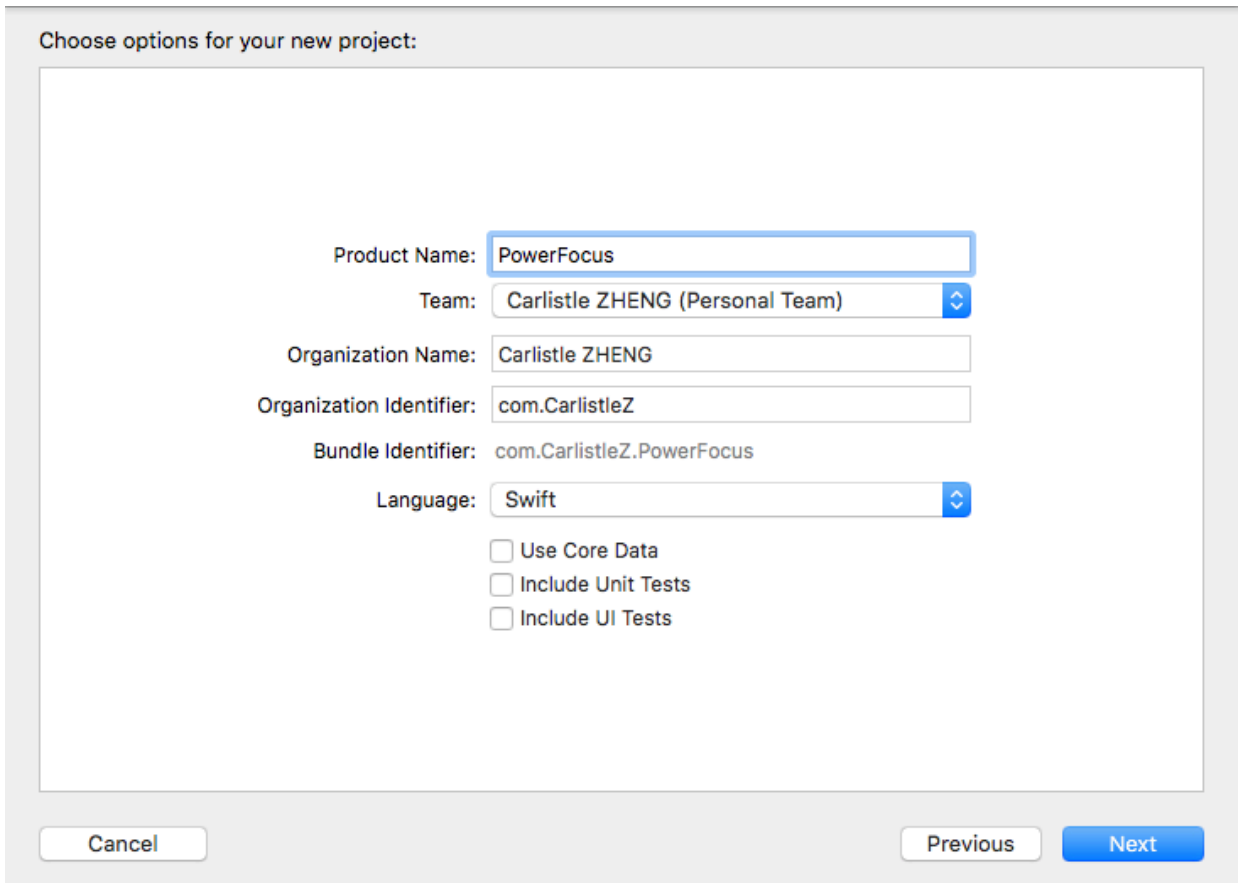
Open Xcode from dock or launchpad. In the welcoming screen, choose Create a new Xcode project.



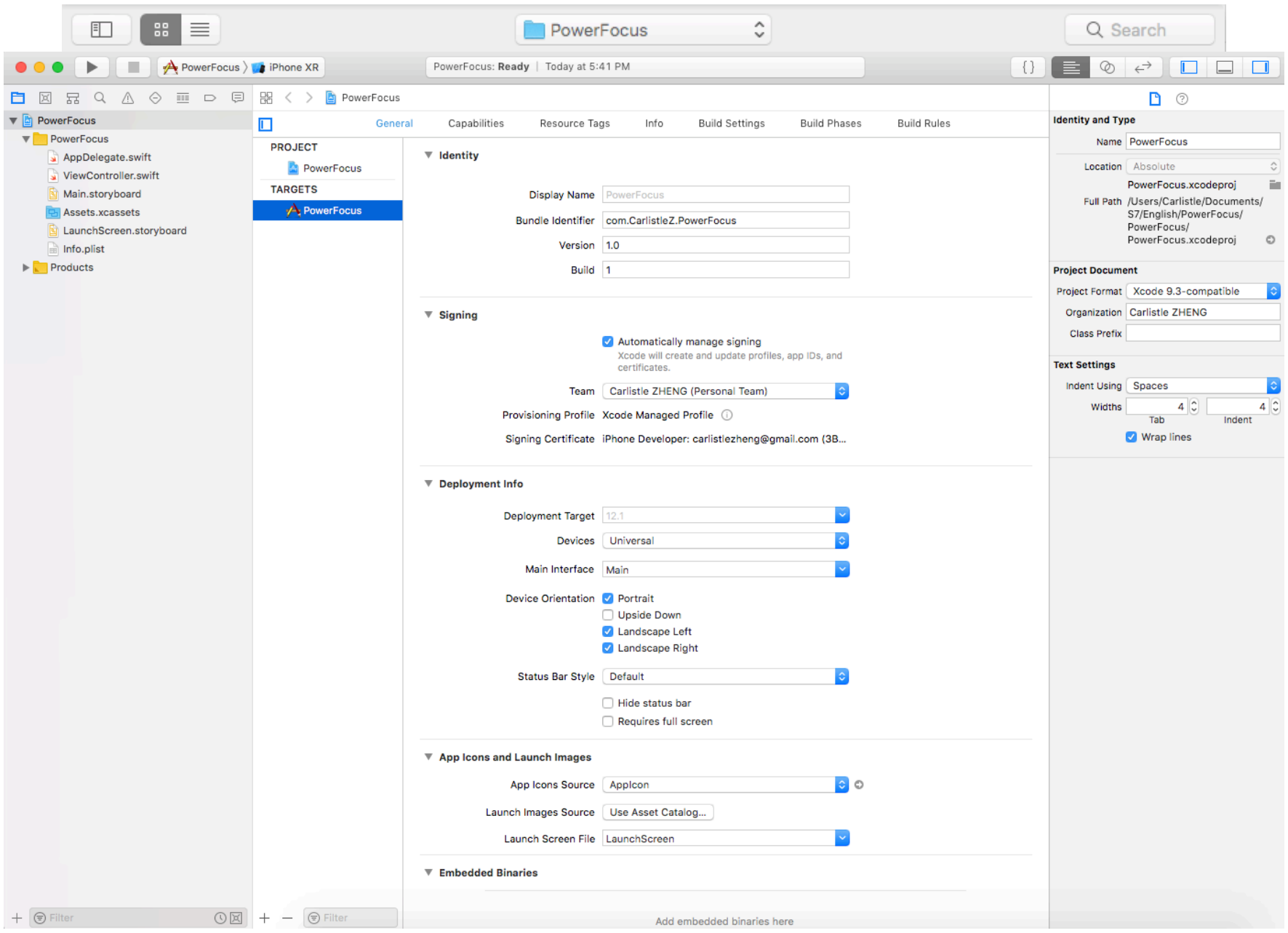
Then under the tab "**iOS**", choose "**Single View App**", then click "**next**".



For the **Product Name**, enter "**PowerFocus**". Here you have to enter an **organization identifier** and an **organization name**, or you won't be able to test it on a real device. But don't worry if you missed it here, you can also set it up after the project is created. For the checkboxes for "Use Core Date", "Include Unit Tests", and "Include UI Tests", leave them unchecked, we don't need these features for now. Then click "Next".



Then choose a root directory for the project. Xcode will create a folder that contains the Xcode info file (.xcodeproj) and the source code. If you want to collaborate with others on this project, you should enable the "Create Git repository on my Mac" option.



Click “Create”, and you will see the newly created project in Xcode like this:

II. SET UP THE STORYBOARD

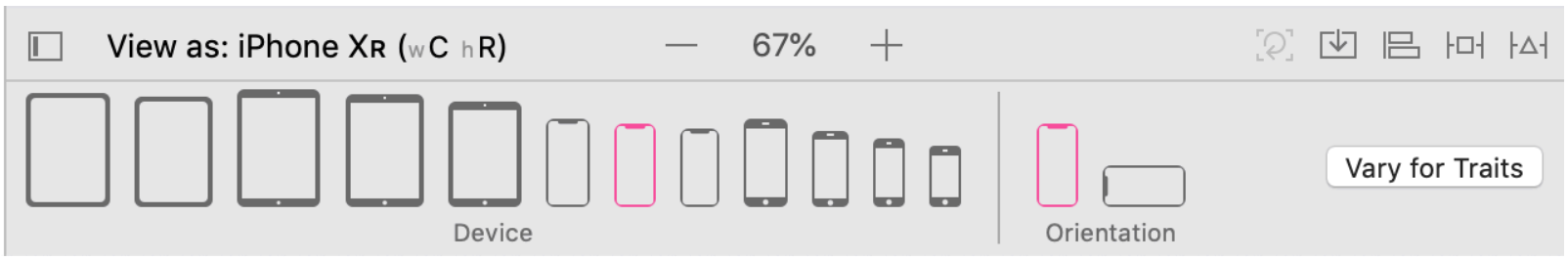
As we have mentioned before in the chapter four, the user interface in iOS consists of storyboards and their interactions. The simple version is that a storyboard is a “screen” (with several exceptions) in your app. The hard part about designing and implementing storyboards is that the UI is hardware-dependent, which means the user interface of your app should look good on all kinds of (iOS) devices or at least on the supported devices.

A storyboard can have different variations depending on the hardware by using traits. However, we should always start simple by creating a functional and well-formed storyboard on one device. In the rest of this chapter, we are going to work on an iPhone XR simulator. I highly suggest you start with an iPhone 8, iPhone X, or iPhone XR because the “plus” and the “max” versions of iPhone can lead to issues because of the zoomed option.

Go to the main editor, and click “View as…”. You will see a list of devices and the orientation option, and trait option. We don’t need to use traits for now, but if you are interested in traits, I highly suggest you look at the official documents:

https://developer.apple.com/documentation/uikit/uitraitcollection

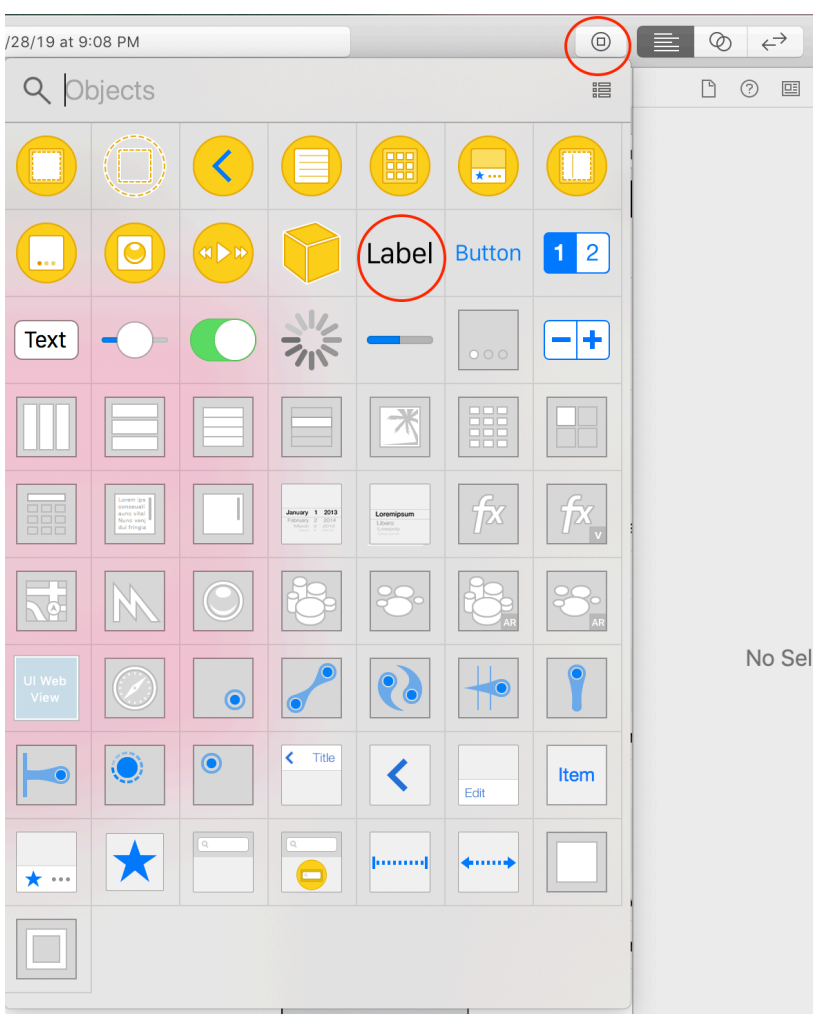
If you want your app to run on an iPhone and an iPad, you will probably need traits in your storyboards. For now, select device “**iPhone XR**” and orientation “**portrait**”.

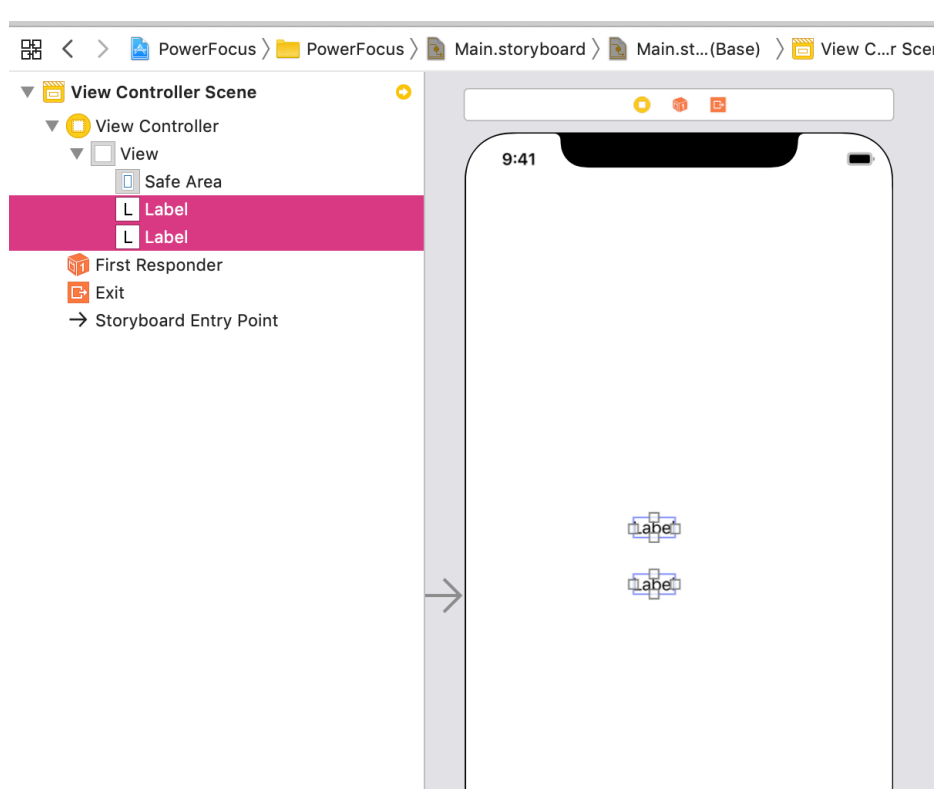


III. ADD WIDGETS TO THE STORYBOARD

In Apple’s UIKit, you have a lot of widget and container options. Using these options are highly recommended because you can have a coherent interface as the system and other applications.

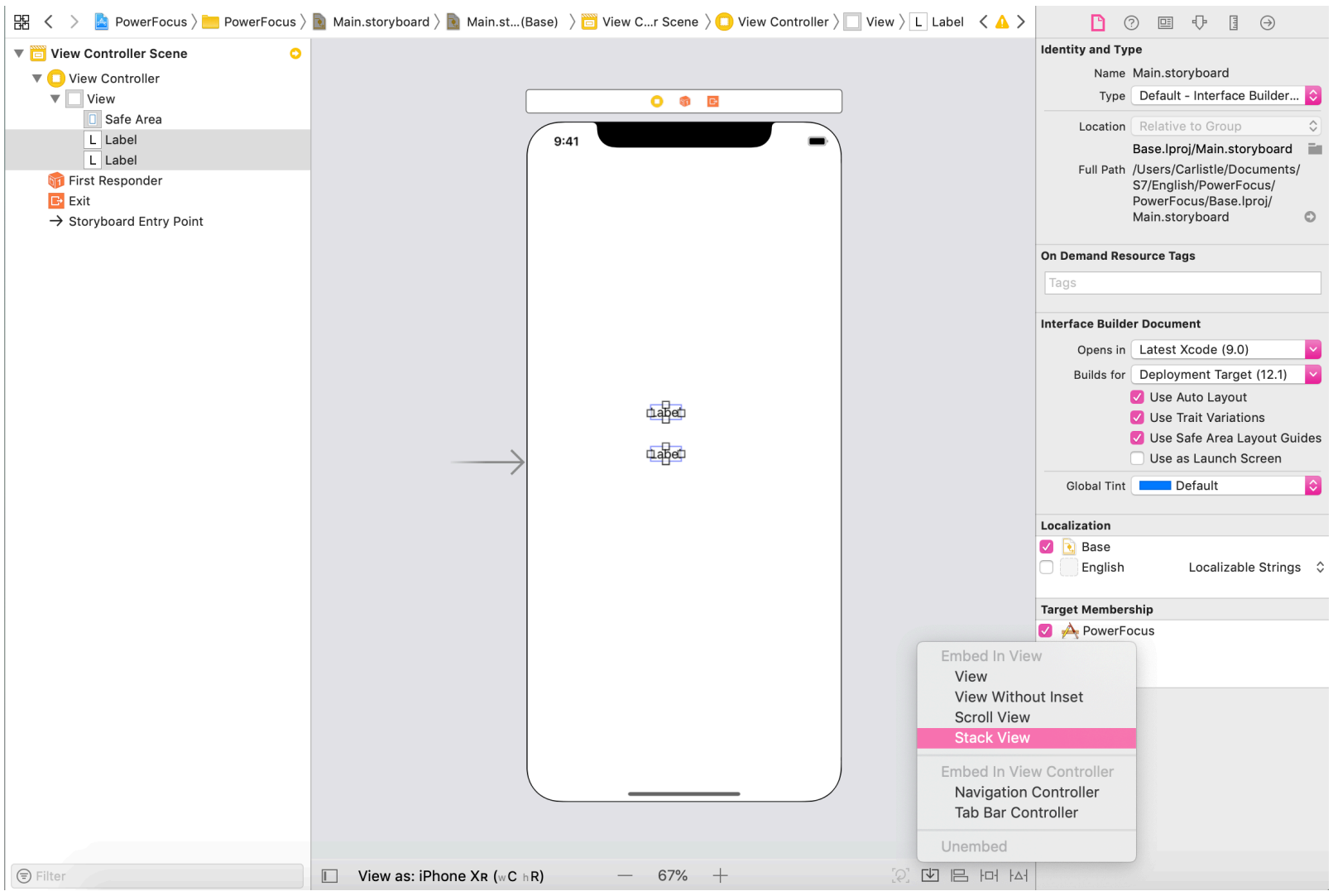
In the upper right corner, click Library (icon), and choose **label**. Drag the label into the storyboard and do it twice. Select both of them by holding the shift and clicking on each of them.





One of the most convenient feature to organize the widgets in storyboards is stack views. It embeds several widgets in to a group of stack, then they are considered as one single object from the outside. A stack view can be horizontal or vertical, and Xcode will choose it automatically for you if you according to the positions of the selected widgets.

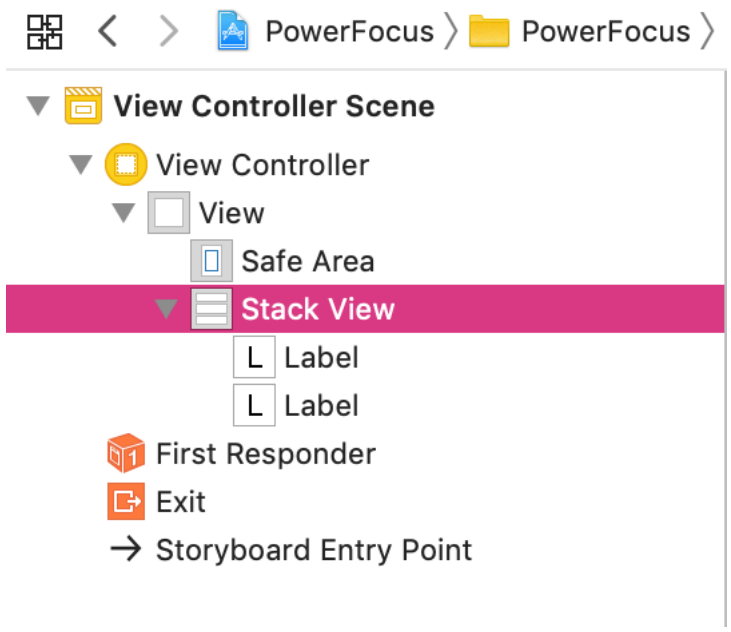
In the lower right corner of the main editor, click "**Embed in**" (icon), and choose "Stack view". Xcode will generate a vertical stack view that contains the labels that you just selected.

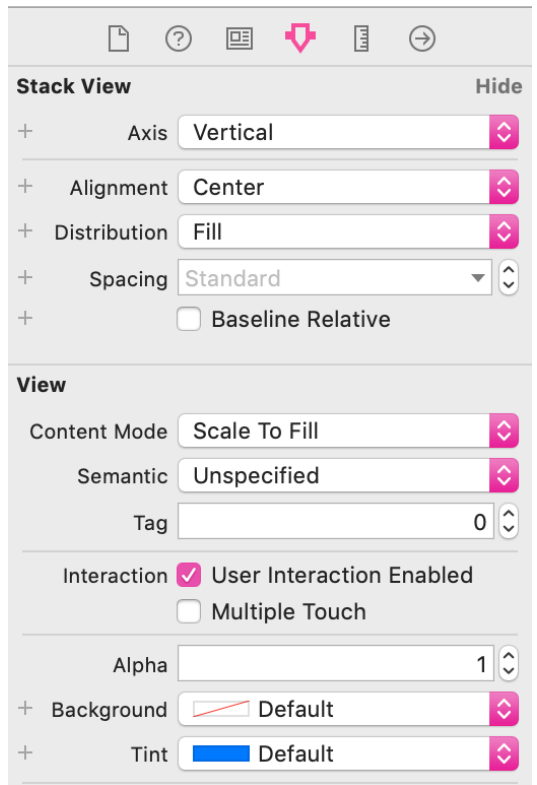


After creating the stack view, you should always check in the document outline if it's created in the right hierarchy. The view hierarchy defines the widgets' ownerships and constraints. In our case, the stack view should be in the view and at the same level as the safe area. Both the buttons should be in the stack view.

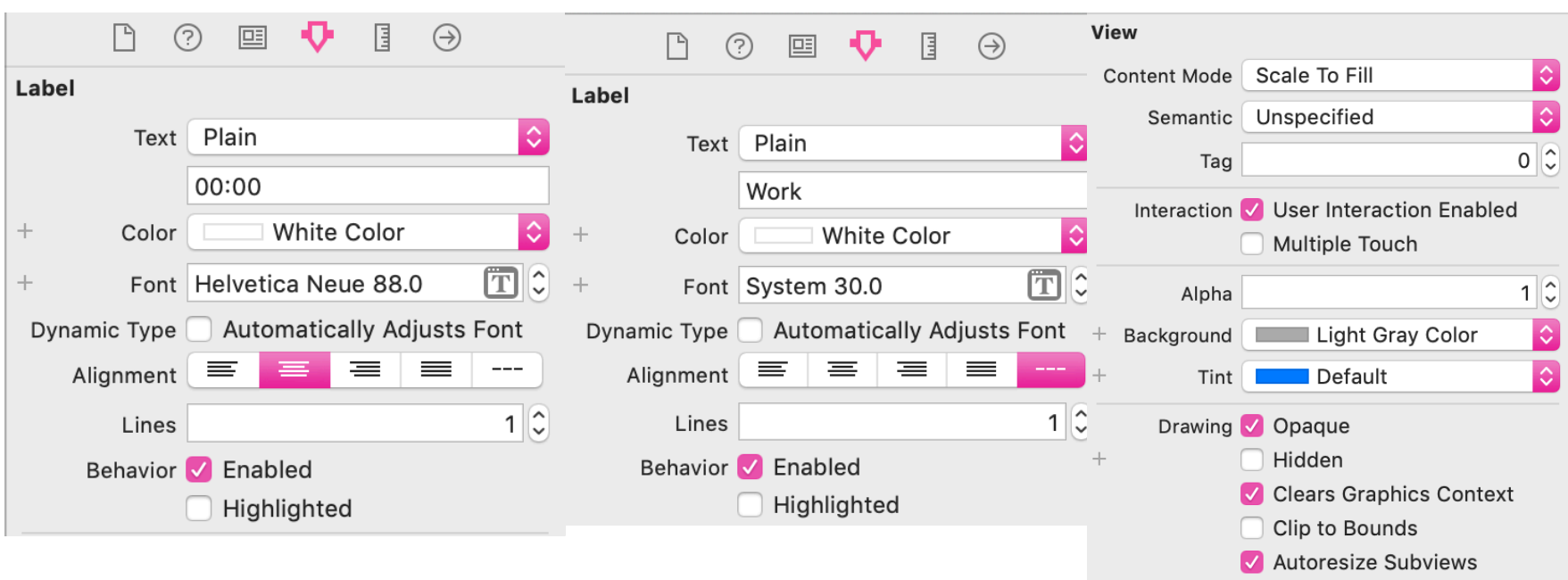
There are two buttons in the stack view, but how are they organized? Stack view allows us to change the way that widget are organized. Select the stack view and go to the attribute inspector on the right. In the section "**Stack View**", you can change the axis direction, alignment, distribution, and spacing. You can also change these attributes in the code during run time. Choose **vertical** axis, **center** alignment, **fill** distribution, and **standard spacing**.

These options are quite straightforward, you can try other options and see how the labels are positioned in the stack view. One thing to notice is the standard spacing, this option allows automatic spacing adjustment according to the user's display settings. Instead of hard-coded spacing, the spacing will be a little bigger in a zoomed display and smaller in a normal one. You can see the spacing difference by changing the device to iPhone XS Max.



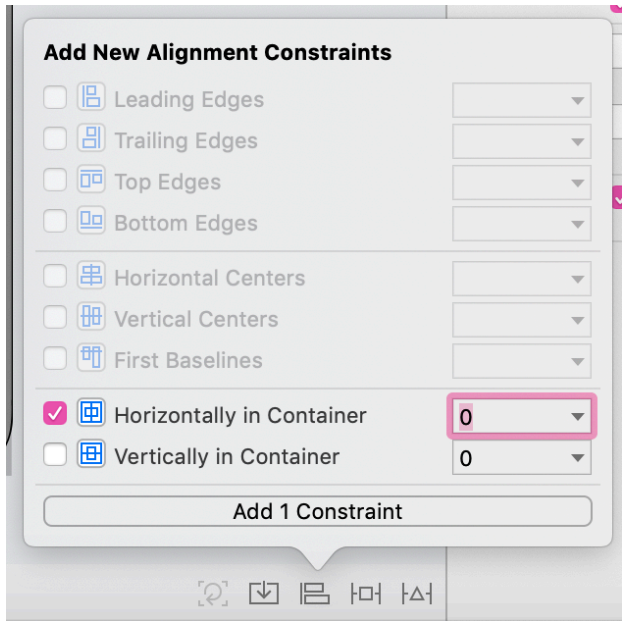


Similar to a stack view, you can also change the settings of a label. Changing the text style is just like other document applications. You can use the following styles for the labels or define you own styles. Because I used white text, in order to see them clearly, I changed the background color of the view to "Light Gray Color"

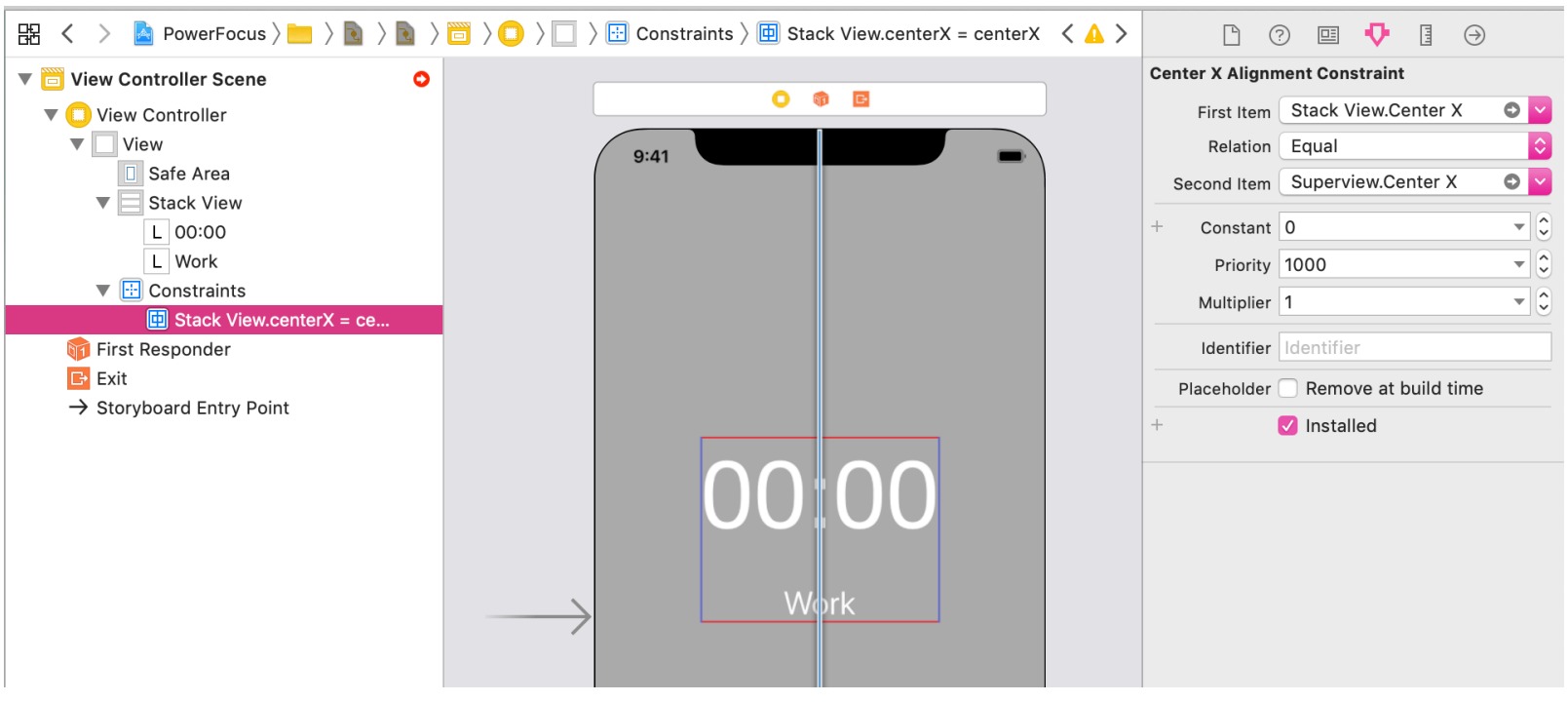


We have just formatted and fixed everything in the stack view, but where should we place the stack view? Because the screens of iOS devices can have many different sizes and resolutions, we need to use constraints to make the layout as hardware-independent as possible.

First we can make the stack view aligned horizontally in container, which is the **View** widget of the View Controller. On the lower right corner of the main editor, click "**Align**" button (icon) and tick "**Horizontally in Center**" with an **offset 0**. Click **Add 1 Constraint** to confirm.

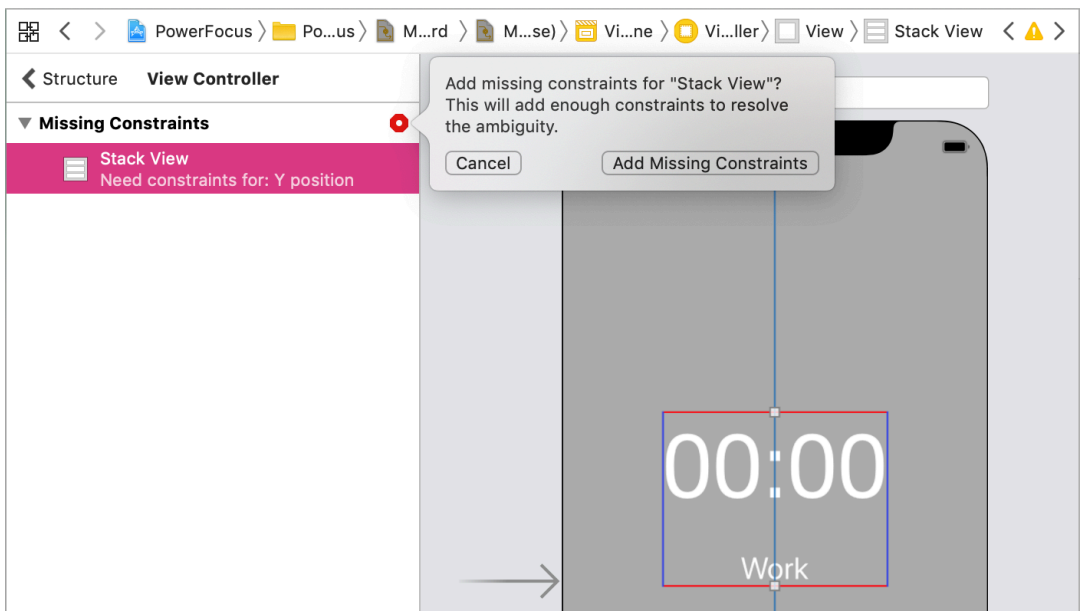


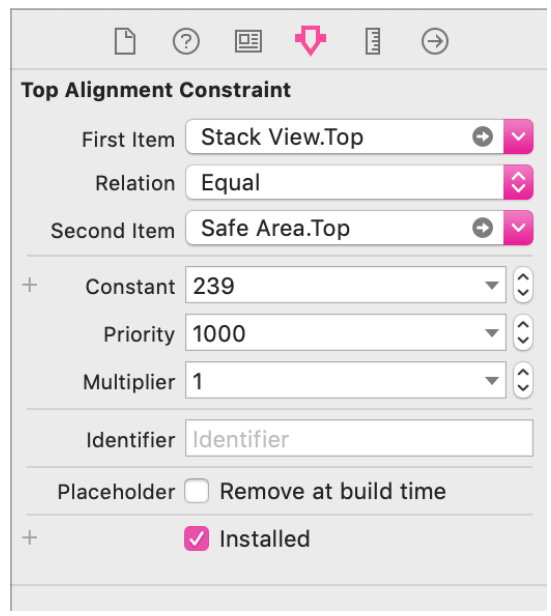
If you have done it correctly, you will see that the stack view is aligned to the center like this:



You can verify this step by looking at the constraints in the document outline. The constraints of a widget is at the **same** level as the widget. The first item and the second item should be the stack view and the superview.

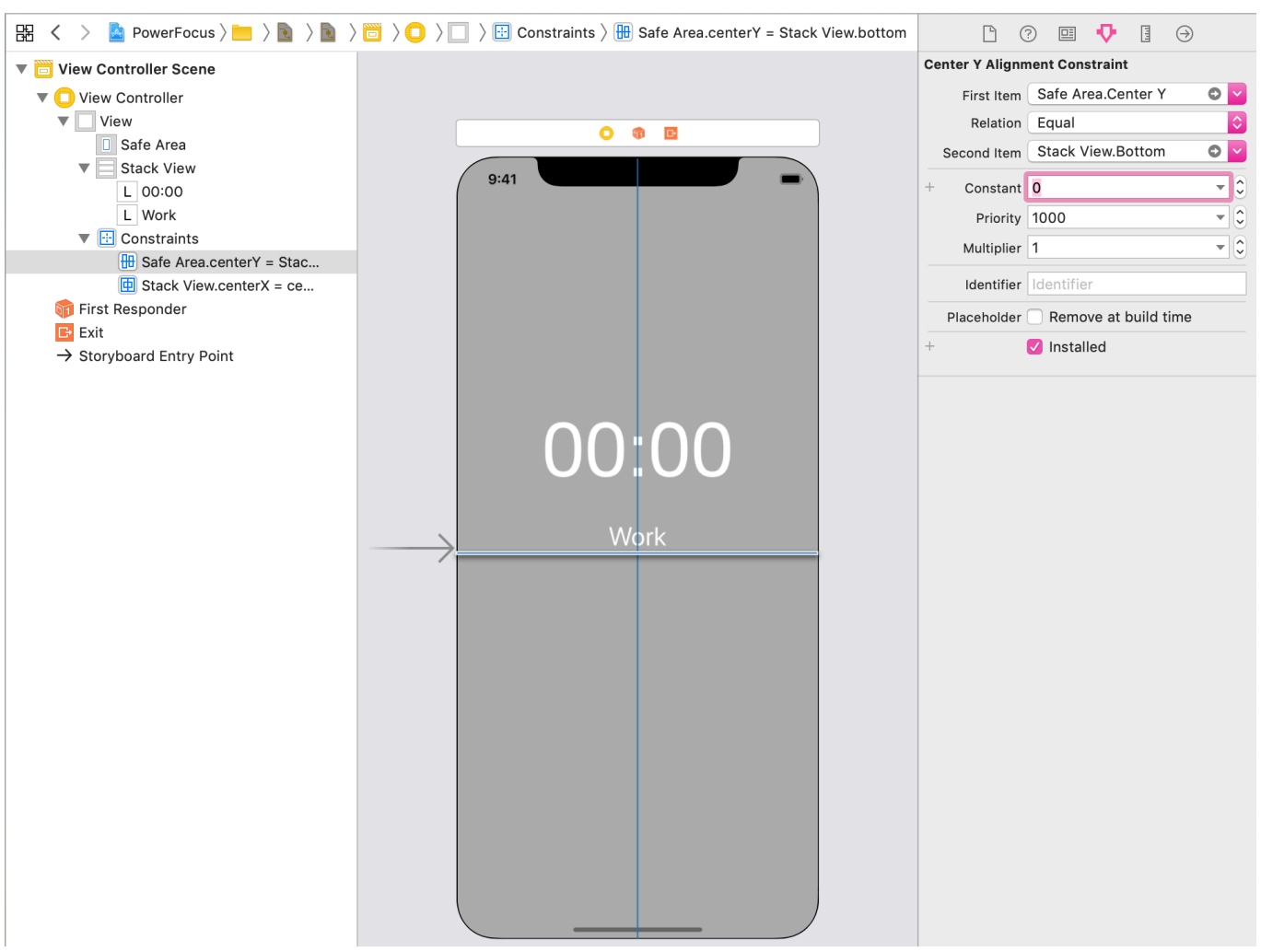
Ops, why is the upper and lower edges of the stack view red? Click on the red arrow to see the error detail. This is because we haven't told Xcode where is the stack view horizontally in the view. Luckily, Xcode gives us a quick fix. Click "**Add Missing Constraints**".





It seems that the stack view is correctly centered after the quick fix. But click on the added constraints and you will see the following properties: **Stack View.Top Equal Safe Area.Top with Constant 239 and Multiplier 1**.

That 239 points constant doesn't look right to you, does it? Even though it is centered in this device, it will take the hard-coded 239 points and apply to every single device. If you switch to an older iPhone with a smaller screen or an iPad with a bigger screen, you will find out that the widget is actually off the center. But you can change the constraint to this:



If you have a hard time imagining how should the constraint formula be, you can always consider the widgets as rectangle cards and the superview as a table. You should ask yourself this question: how should I describe where to put my card on the table?

My solution is: the bottom of the stack view should align to the center of the superview. So the constraint's formula is:

Safe Area.Center Y Equal Stack View.Bottom with Constant 0 and Multiplier 1

This isn't the only solution. You can try your own formula out, and the only requirement is that the layout should look good on all supported screens.

If you have trouble setting everything up, you can download the project from this link:

https://github.com/CarlistleZ/MyiOSTutorial/blob/master/Chapter_Nine_Stack.zip

IV. IMAGE VIEW AND USE ASSETS IN STORYBOARDS

Another important way to make the UI look pretty is to use images. Most of the images in the UI is different from a normal image in your photo library. They belong to an app and nobody could have access except from the app itself. Files like this are called **assets**. An asset can be an image, a piece of audio, video, a text file… They are stored in a special folder (directly under the project root) called **Assets.xcassets**. In this folder, you can put some background images, icons, sound effect audios, JSON files…

An important question to ask yourself before setting a file as an assets is: Is it going to change during run time? In iOS, we have Core Data to store large amount of user data, and property list to store the settings of an app. These files are going to be constantly modified and accessed during the execution. But files like a background image and an icon will not change during runtime, so they are indeed assets. Having many assets can increase your app's size and make it slower to run because all the assets have to be loaded before execution.

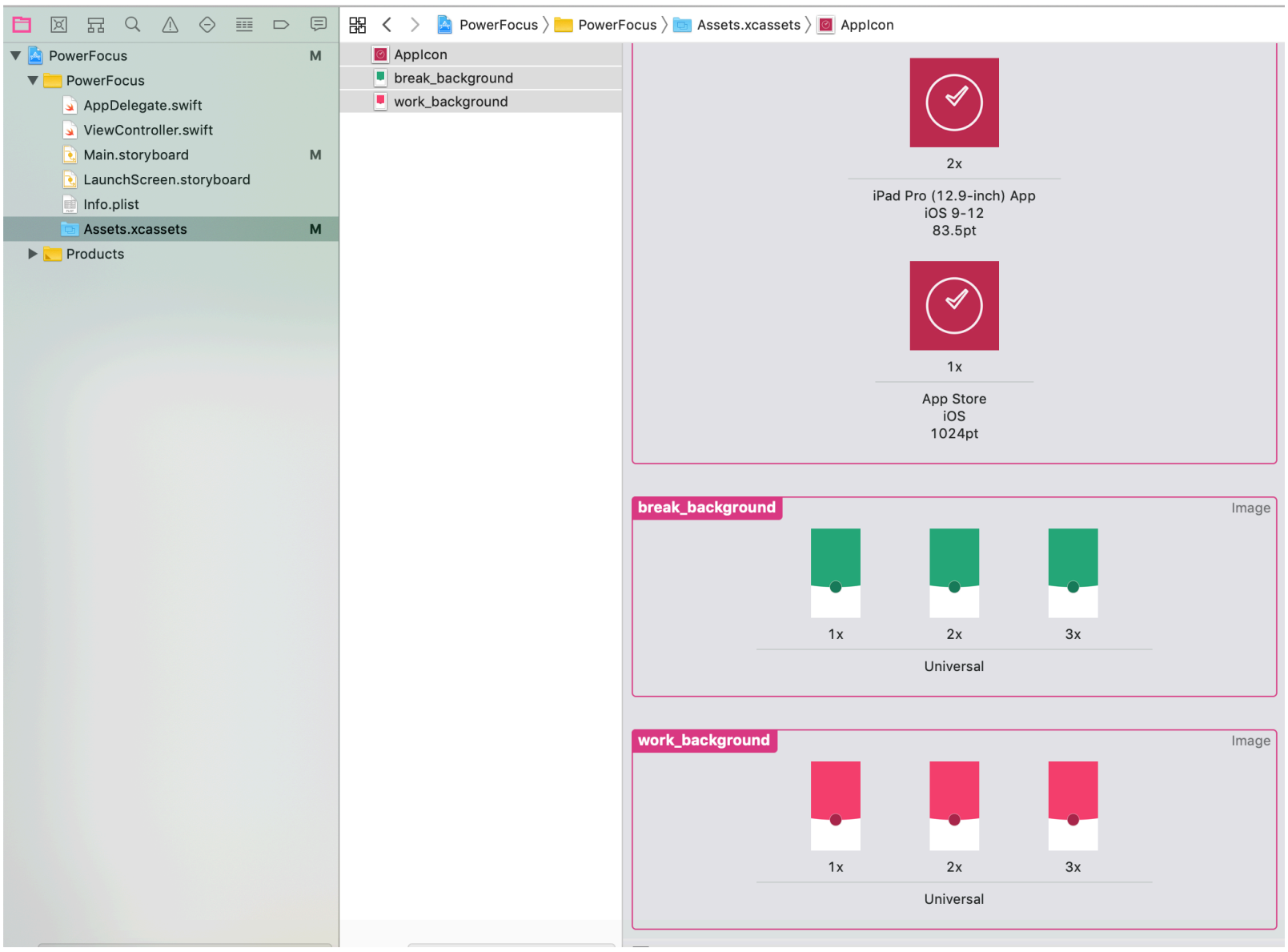
To add an asset, go to the project navigator and open Assets.xcassets. Simply choose the file from Finder and drag it to the list. To download the assets, use the following link:

https://github.com/CarlistleZ/MyiOSTutorial/blob/master/Chapter_Nine_Assets.zip

In the asset folder, you will notice that there are several copies of the same image in different resolutions. This is because there are different sizes of screens and iOS automatically selects the most suitable image size according to the hardware. The naming convention is add "**@2x**" and "**@3x**" for bigger images, and all the image with the same prefix will be considered as one asset.

A very important asset to mention is the Appicon. It has 18 sizes for different use cases and it will be the icon of the app for home screen, settings, spotlight, and notification. You need to set the icons for iPhone, iPad, and App Store so that people with different devices are able to see you app with the correct icon.

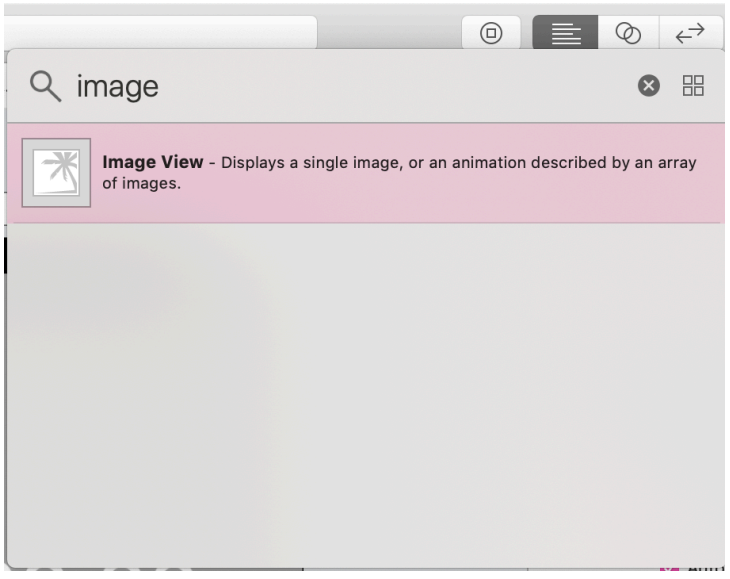
This is the result of the background assets and the app icon.

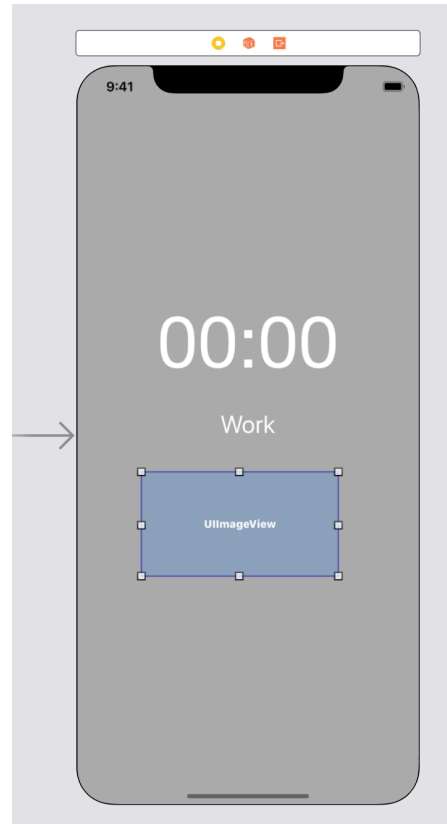


V. IMAGE VIEW AND IMAGE IN STORYBOARDS

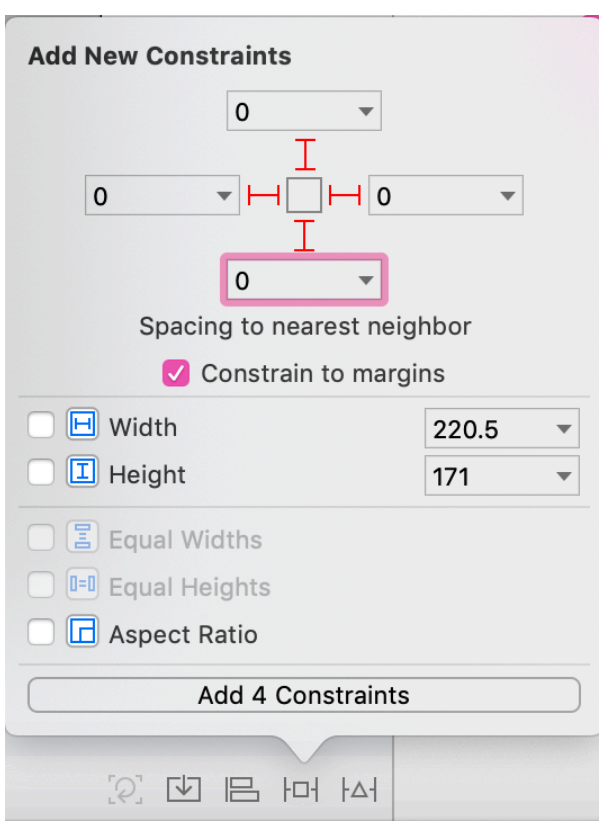
We have provided our app the assets to display, but how can we tell the view to display a certain image and where can we put it? The widget that holds an image is **Image View**.

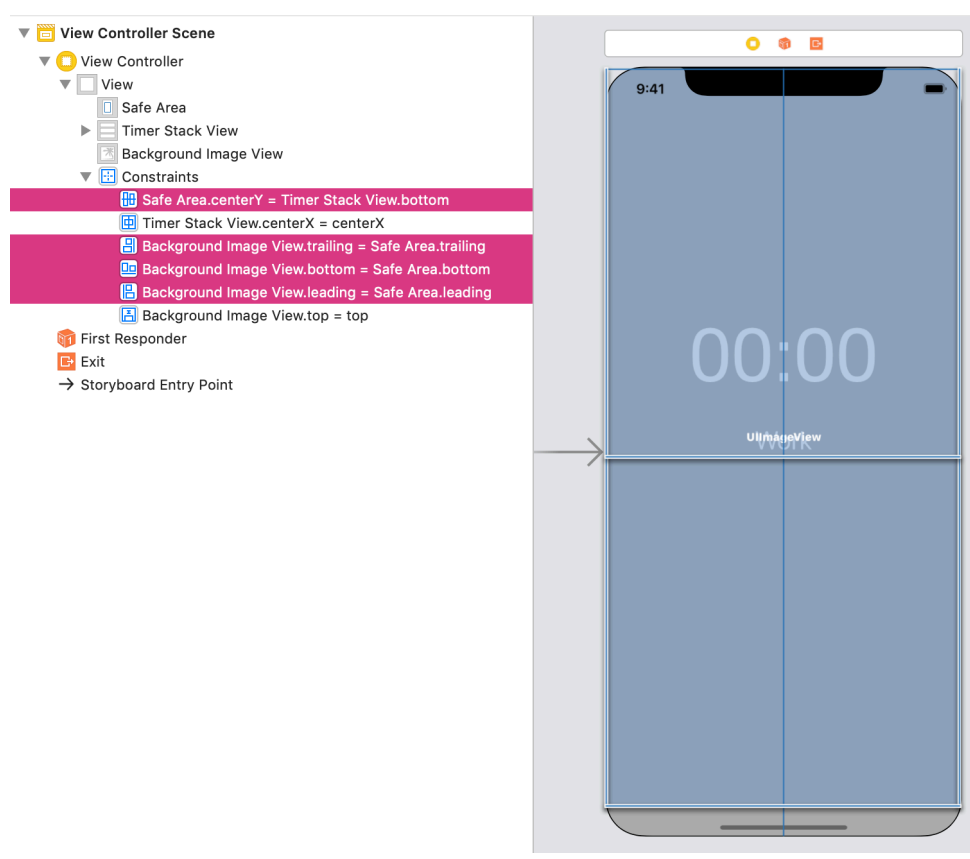
Choose image view from the library and drag it to the storyboard. You will have the following result:



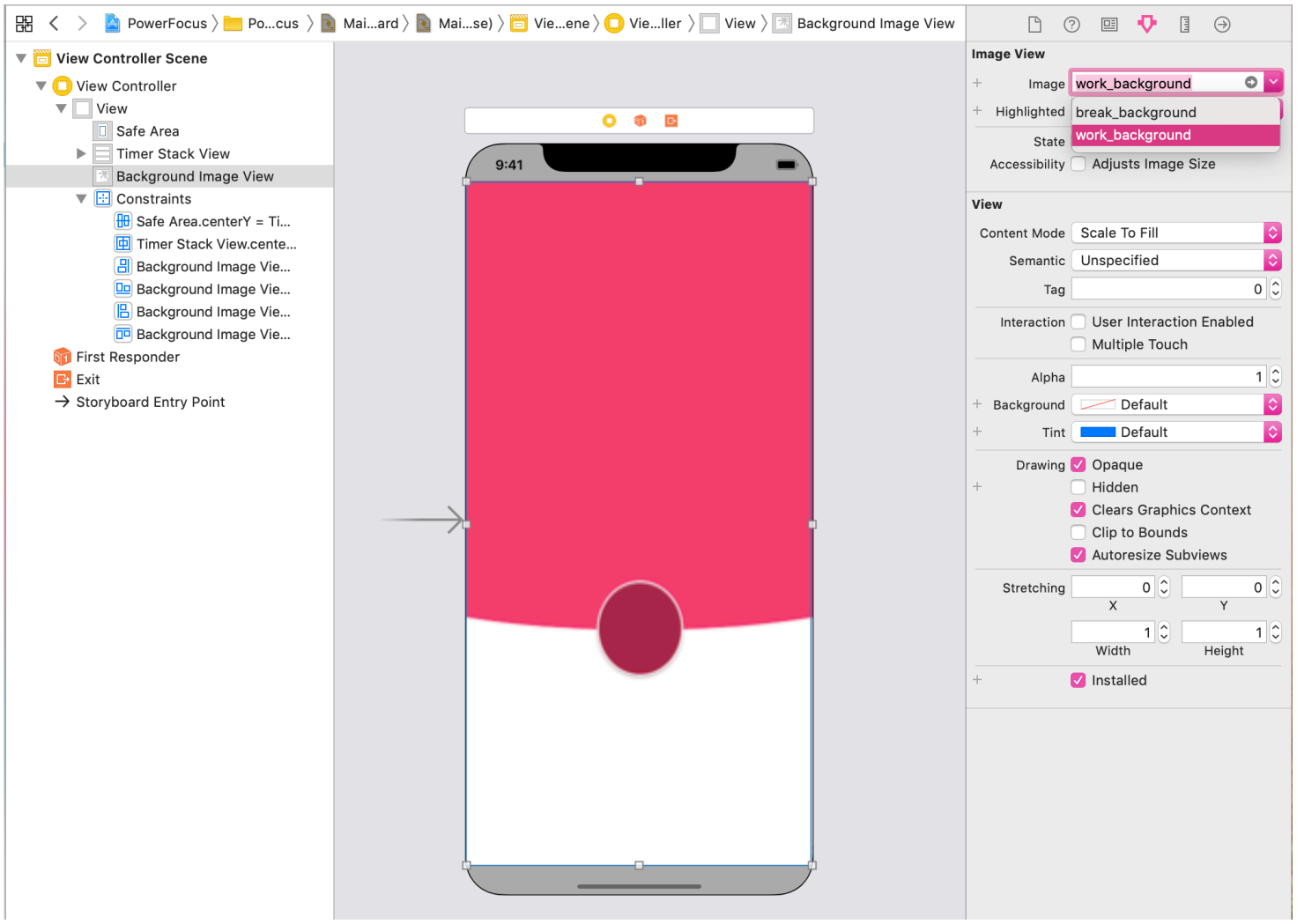


For the background, we want the UIImageView to fill the whole screen. We need to add four constraints to pin its edges to the superview. Another way to add constraints is using "Add New Constraints" button on the lower right corner of the main editor. Choose **(0, 0, 0, 0)** and check **constrain to margins**. Click Add 4 Constraints and you will see 4 more constraints in the view hierarchy. The UIImageView should fill the entire safe area. If not, go to each constraint and change the item "nearest neighbor" to "safe area".



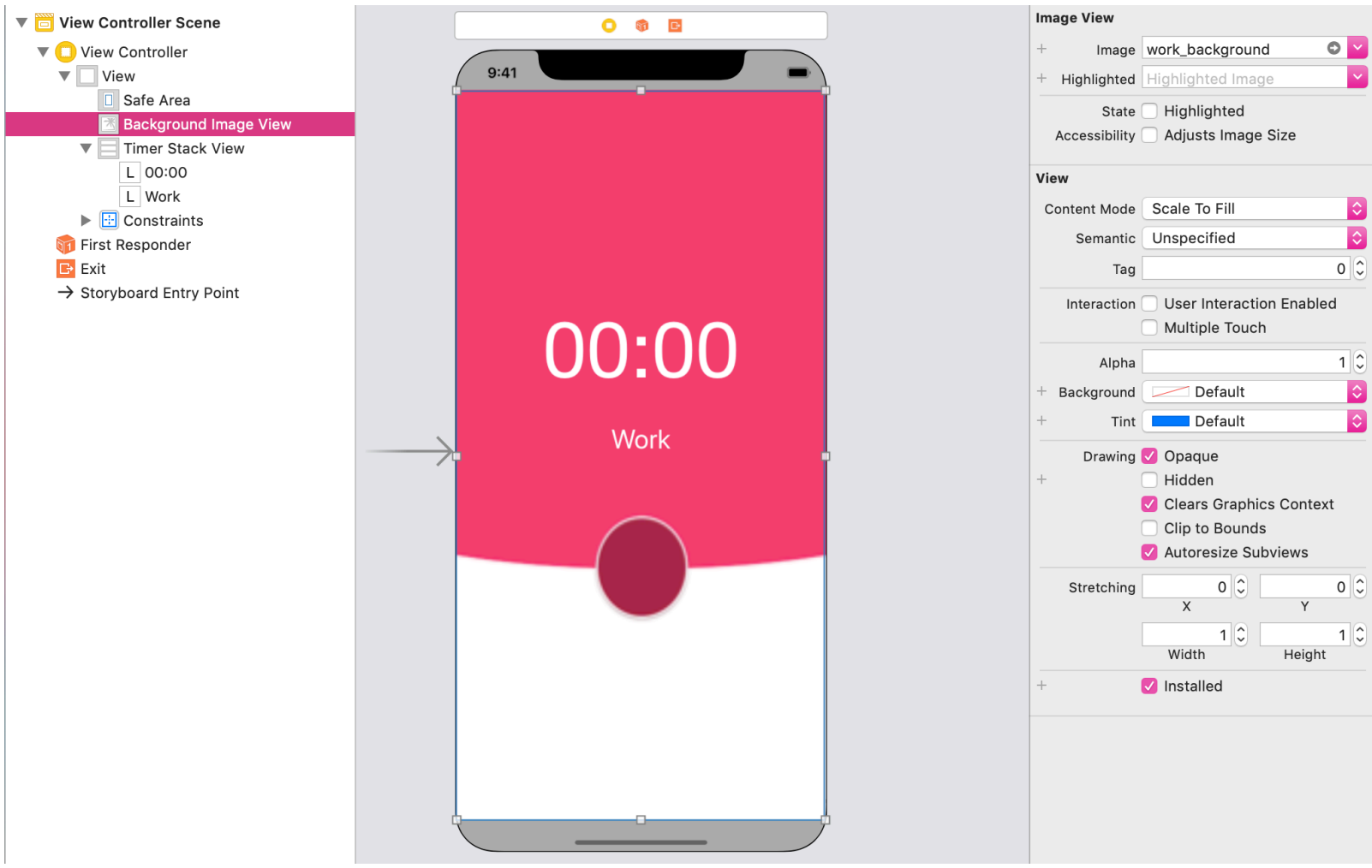


Choose Background Image View and go to the attribute inspector. In the drop list of image, you will find a list of image assets. Choose **work_background**, you will see the following result:



The background image is covering everything else, this is obviously not what we want. It is displayed in this way because the order in the view hierarchy is important. It always renders the widgets from top to bottom. If the current widget is not opaque (alpha is 1), it will cover all the previously rendered widget in its rectangle. The "**Timer Stack View**" is rendered before the "**Background Image View**" and since the image view isn't opaque, we can no longer see the stack view even if it's still there. Its graphic layout won't even be calculated by the system to save CPU resource. This is a common reason why a widget appears to be "missing".

To solve this, simply drag the image view above the stack view. Since the background of the stack view is transparent, you will see both of them at the same time.

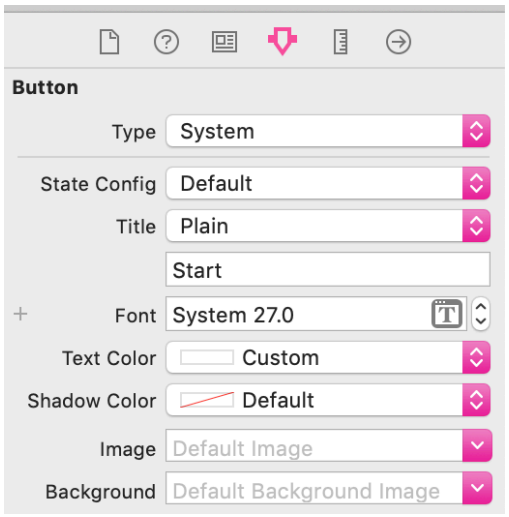


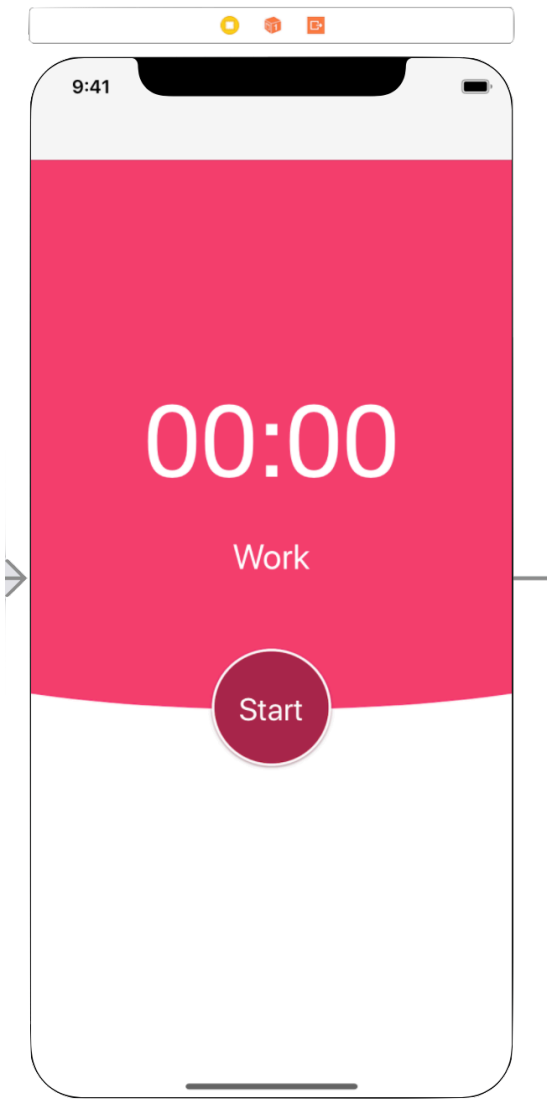
Congratulations, now you know how to create storyboards for an iOS app! You can try the following exercises to test your self. If you are stuck somewhere, go back and read the corresponding section again. If you are really stuck, you can check out the solution at the end of this chapter.

EXERCISE #1

Add a start button to the storyboard. This button isn't in any stack, but it has to be displayed in a proper position. Use constraints correctly and test your solution in as many screens as possible.

The suggested style and the preview of this button is:

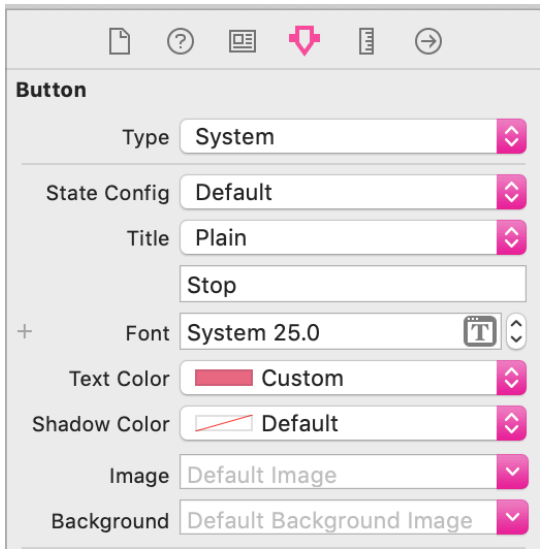


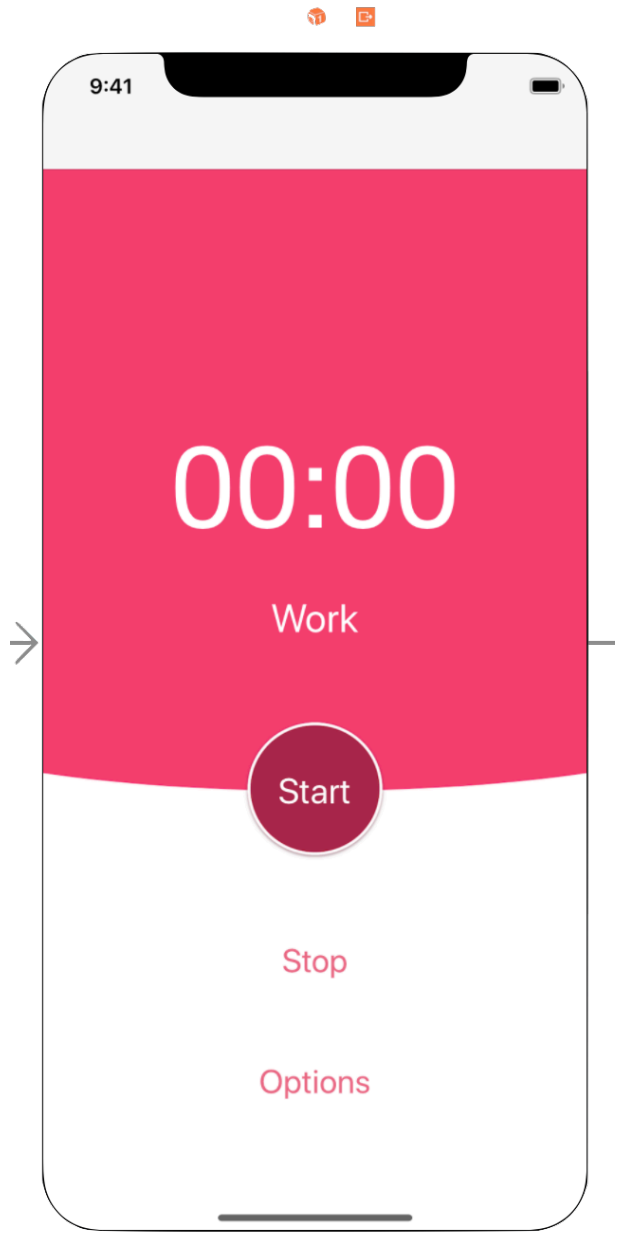


EXERCISE #2

Add a stop button and an option button to the storyboard. These two buttons are in a stack, which has to be displayed in a proper position. Use constraints correctly and test your solution in as many screens as possible. The elements in the stack have to be well organized (centered, proper spacing, proper distribution…)

Here are some suggestions for the style of the buttons and the preview.


Button
Type
System
State Config
Default
Title
Plain
Stop
Font
System 25.0
Text Color
Custom
Shadow Color
Default
Image
Default Image
Background
Default Background Image


9:41
00:00
Work
Start
Stop
Options

SUMMARY

In this chapter, we have worked on these topics:

- set up a new project
- set up a new storyboard
- add widgets and configure widgets from storyboard
- use stack view and constraints to organize widgets
- add image assets and app icon
- view hierarchy

If any of these is unclear to you, please make sure to go back and read the related part or parts before moving on the the next chapter.

For reference, you can download the complete Xcode project from

https://github.com/CarlistleZ/MyiOSTutorial/blob/master/Chapter_Nine_PowerFocus.zip