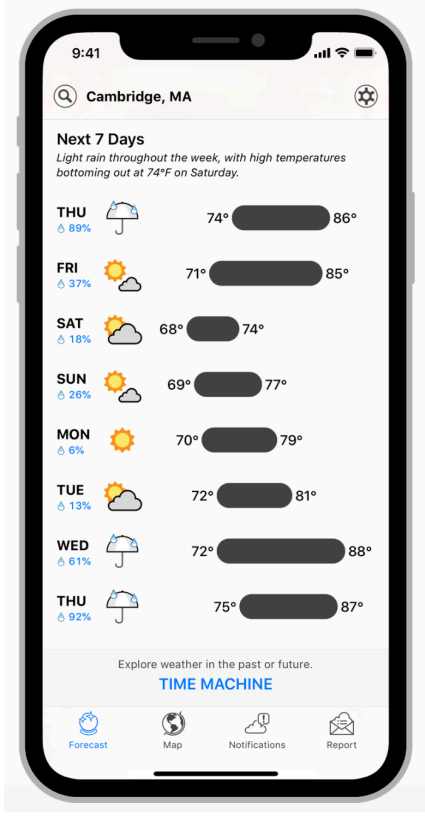
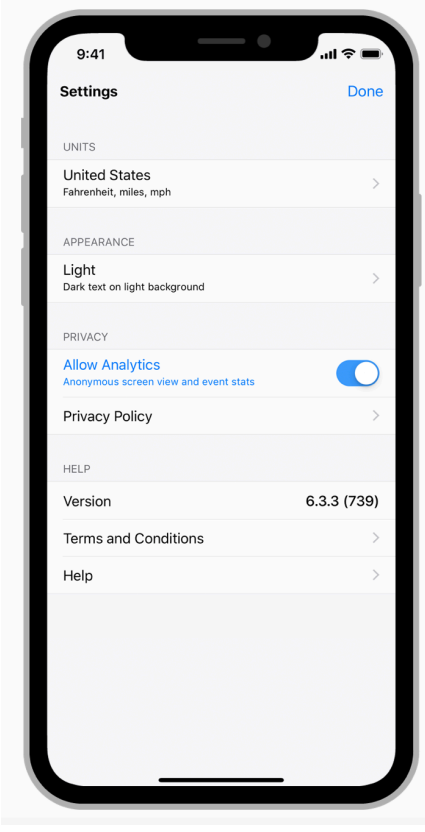
CHAPTER FIFTEEN: SETTINGS STATIC TABLE VIEW

I. DESIGN SETTINGS

App setting page is one of the most important pages of a mobile application. For iOS applications, all UI/UX designs should follow Apple's Human Interface Guidelines. Here's what it says about the settings:

Some apps may need to provide a way to make setup or configuration choices, but most apps can avoid or delay doing so. Successful apps work well for most people right away, while also offering some convenient ways to adjust the experience. When you design your app to function the way most people expect, you decrease the need for settings.





Infer what you can from the system. If you need information about the user, device, or environment, query the system for it whenever possible instead of asking the user. For example, instead of asking someone to enter their zip code so you can present local options, ask permission to use their current location. Gracefully fall back to manual entry if the user denies access to their information.

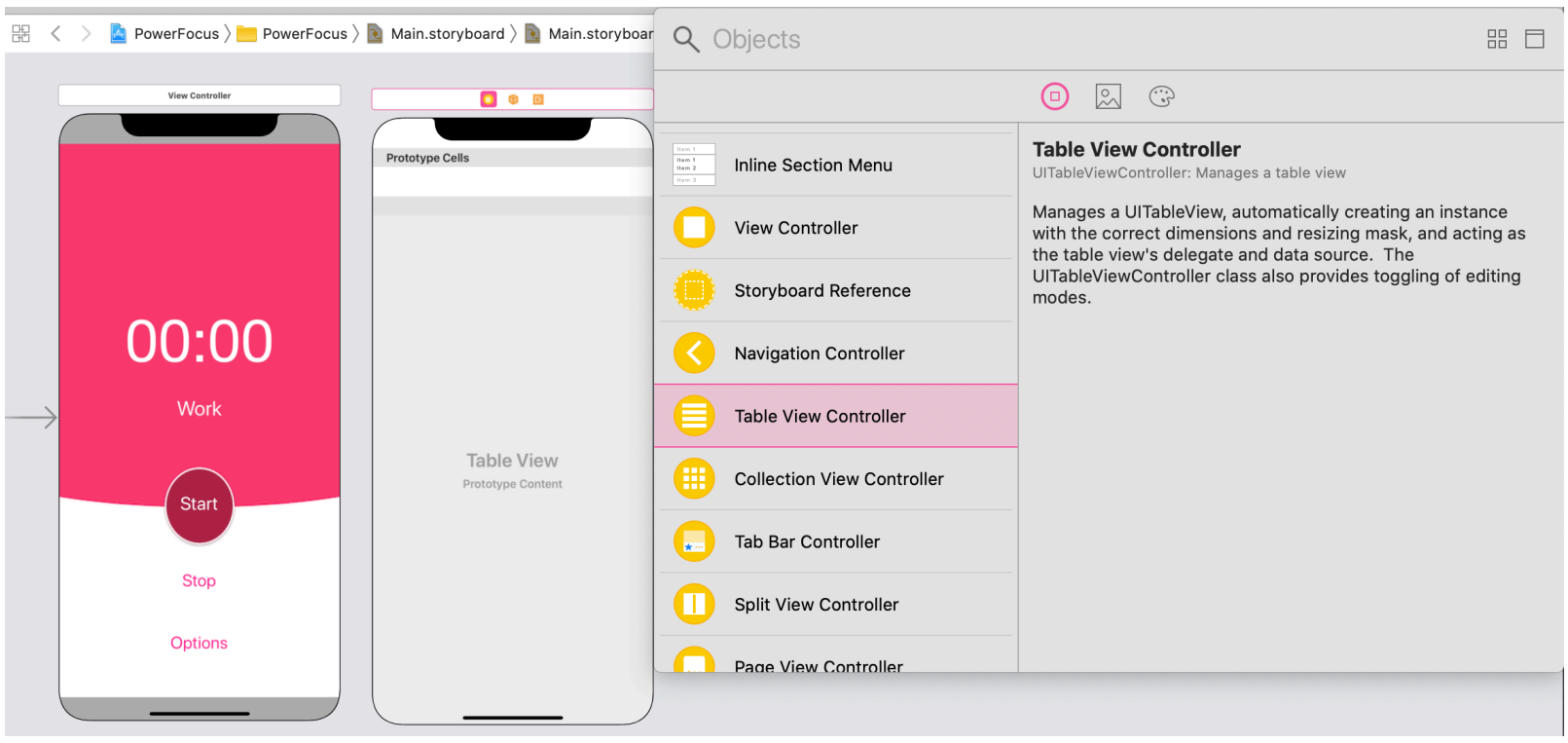
Thoughtfully prioritize configuration options within your app. Your app's main screen is a good place for options that are essential or that change frequently. Secondary screens are better for options that change only occasionally.

Expose infrequently changed configuration options in Settings. The Settings app is a central location for making configuration changes throughout the system, but people must leave your app to get there. It's far more convenient to adjust settings directly within your app. If you must provide settings that rarely require change, see Implementing an iOS Settings Bundle in Preferences and Settings Programming Guide for developer guidance.

Provide shortcuts to Settings when appropriate. If your app includes text that directs users to Settings, such as "Go to Settings > MyApp > Privacy > Location Services," provide a button that opens that location automatically. For developer guidance, see openSettingsURLString in UIApplication.

II. CREATE A STATIC TABLE VIEW

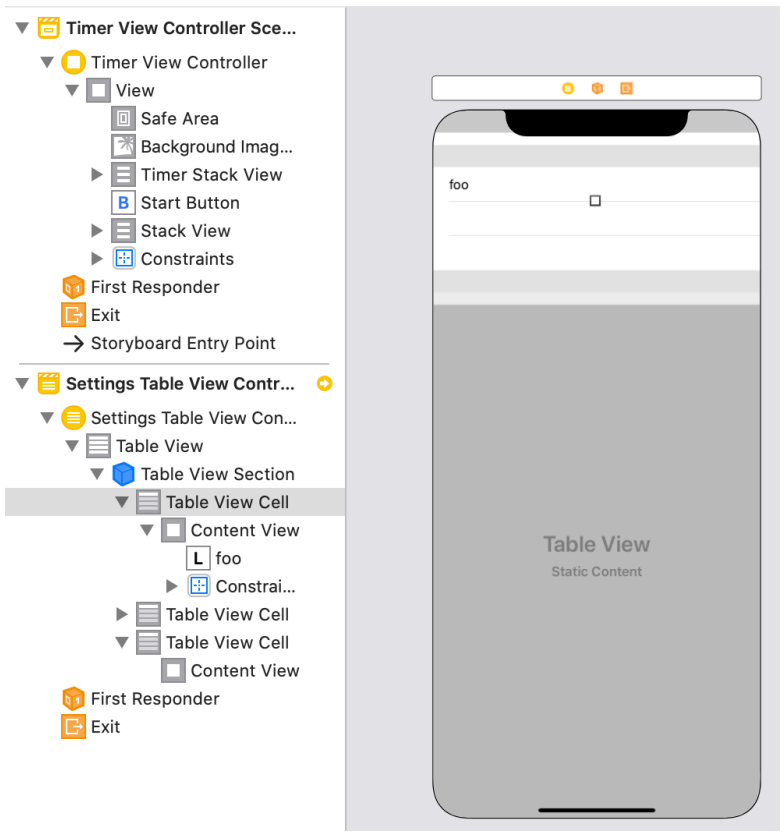
The settings screen should have its own view controller scene, and it should be a table view controller. So click Main.storyboard and click "+" library button in the upper right corner. In the drop down menu, select "**Table View Controller**" and drag it next to the timer View Controller.

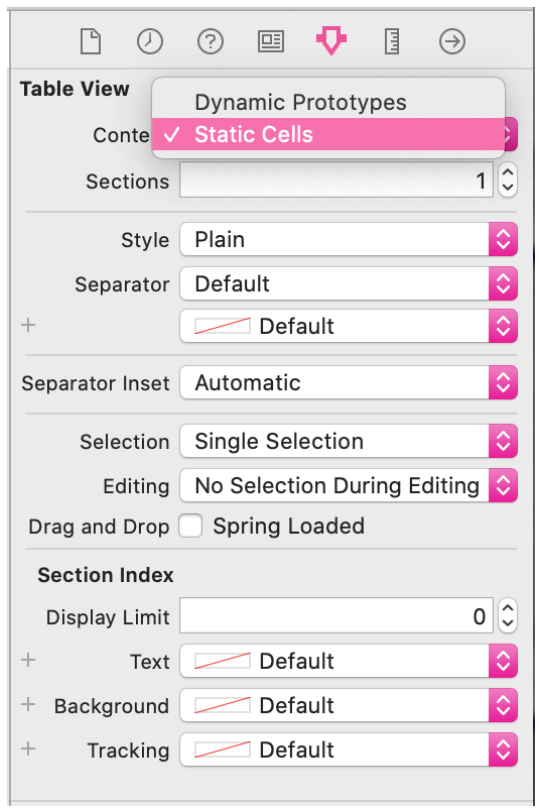


Do you still remember the difference between a dynamic table view and a static one? In a dynamic table view, the number of sections and rows can change when its data source changes. For example, a dynamic table view for an array of strings can change its number of rows according to the size of the string array.

However, for the cases where the number of rows can be predicted and doesn't change, it is much more efficient to use a static one. In our case, the options of settings are always the same, so the table view should be a static one.

To change the table view's content type, go to the "**Attributes inspector**", and change the content type to "Static Cells". The table view should look like this:



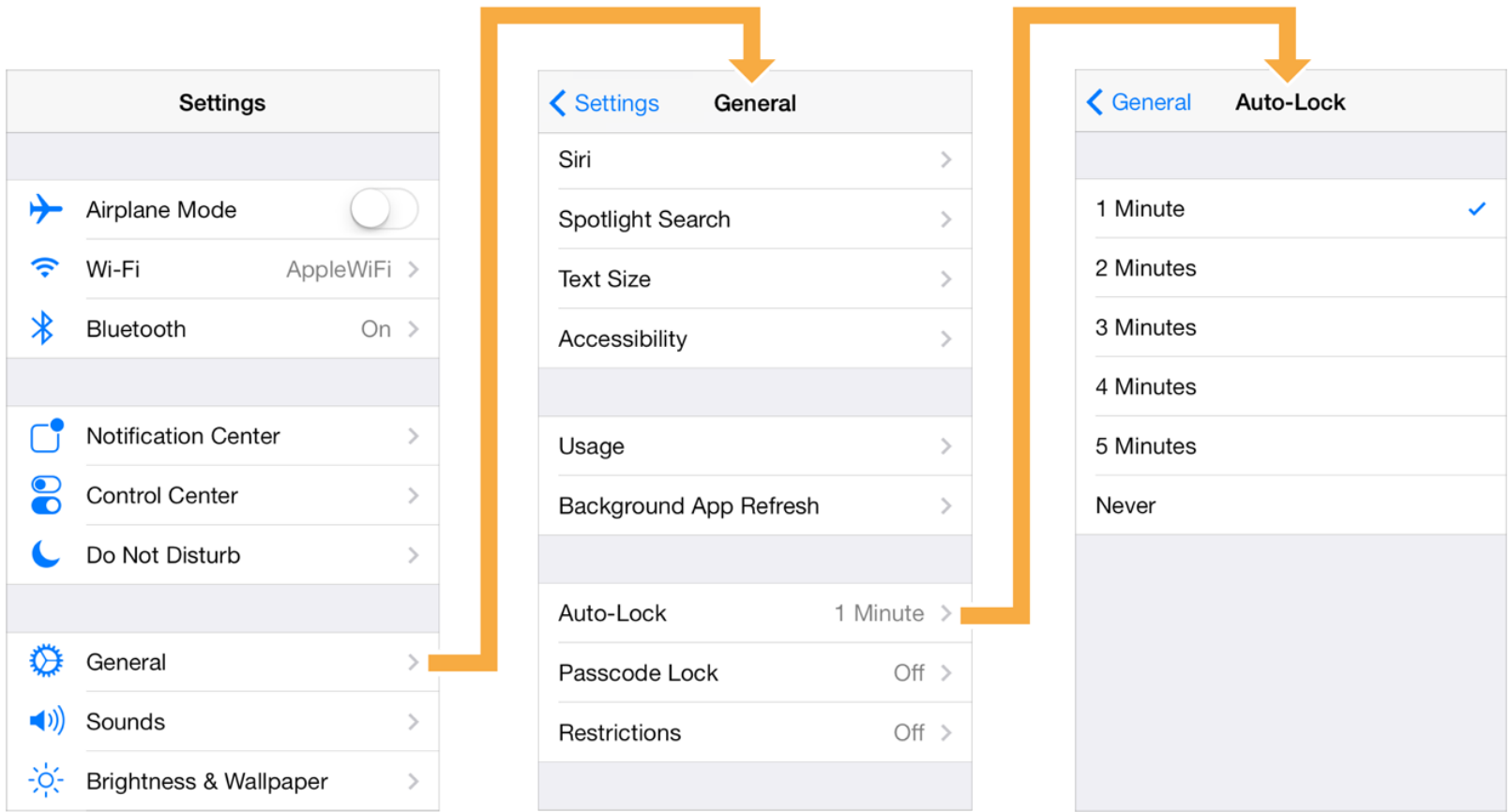


Add a text view of "foo" to the first cell of the first section to make sure it displays the table view correctly. For constraints, use "vertically in container" and "standard leading space".

One last thing to do in the storyboard is to make the settings table view controller display and dismiss properly. When the "Options" button is clicked, the timer view controller should stay behind the settings view controller. So we could embed both of them in the navigation controller. You can image a navigation controllers as "a stack of cards", with each card being a view controller. Here's a quick overview of navigation controller:

A navigation controller is a container view controller that manages one or more child view controllers in a navigation interface. In this type of interface, only one child view controller is visible at a time. Selecting an item in the view controller pushes a new view controller onscreen using an animation, thereby hiding the previous view controller. Tapping the back

button in the navigation bar at the top of the interface removes the top view controller, thereby revealing the view controller underneath.

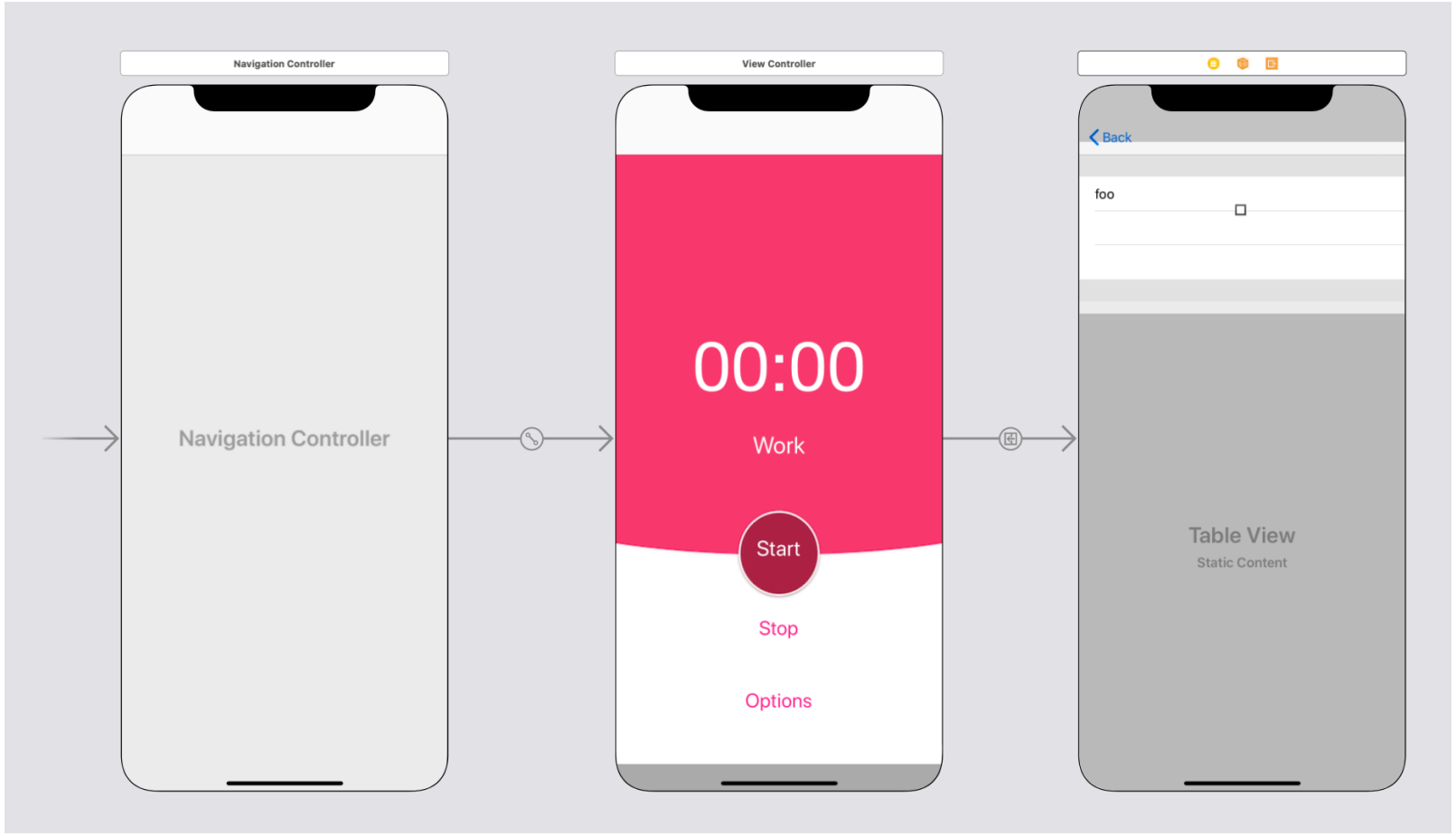


Use a navigation interface to mimic the organization of hierarchical data managed by your app. At each level of the hierarchy, you provide an appropriate screen (managed by a custom view controller) to display the content at that level. Figure 1 shows an example of the navigation interface presented by the Settings application in iOS Simulator. The first screen presents the user with the list of applications that contain preferences. Selecting an application reveals individual settings and groups of settings for that application. Selecting a group yields more settings and so on. For all but the root view, the navigation controller provides a back button to allow the user to move back up the hierarchy.

In the document outline pane, select “Timer View Controller”. Then choose **Editor > Embed In > Navigation Controller**.

When the “options” button is clicked, the navigation controller should display the settings table view controller. So we should add a segue the options button in the TimerViewController. Control drag from the Options Button to the setting view controller and choose “**show**”.

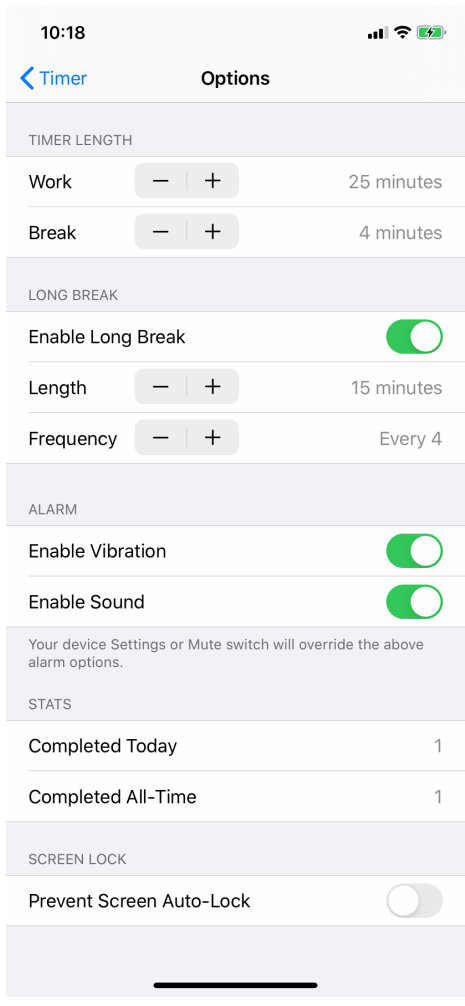
The view controllers should look like this when they are all set up:



Now you can test it out in your iPhone or in a simulator, the settings should appear and dismiss properly.

III. CUSTOMIZE STATIC TABLE VIEW

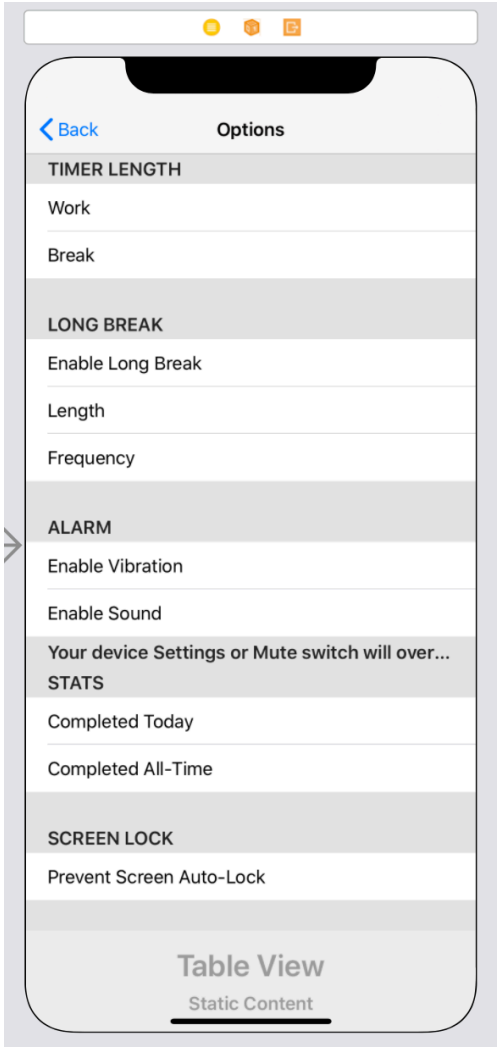
In the section II, we have set up the settings table view. Now it's time to customize it! Here's a preview of the options page:



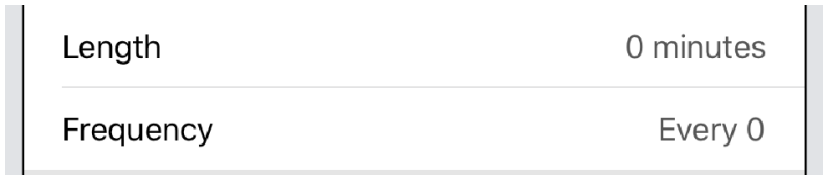
There are five static sections. Select the settings table view and go to the Attributes inspector. Change the number of sections from 1 to 5. You can change the number of rows by selecting the section in the document outline and change the number of rows in the **Attributes inspector > Table View Section > Rows**. For sections 1, 3, and 4, change it to 2. For section 5, change it to 1.

In the **Attributes inspector > Table View Section > Header**, change the headers to "TIMER LENGTH", "LONG BREAK", "ALARM", "STATS", and "SCREEN LOCK". For section "ALARM", add the footer "Your device Settings or Mute switch will override the above alarm options".

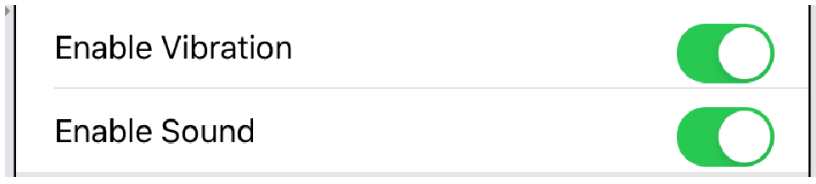
Just like the way we added a label "foo" for testing, add the correct labels to all table view cells. The table view should look like this:



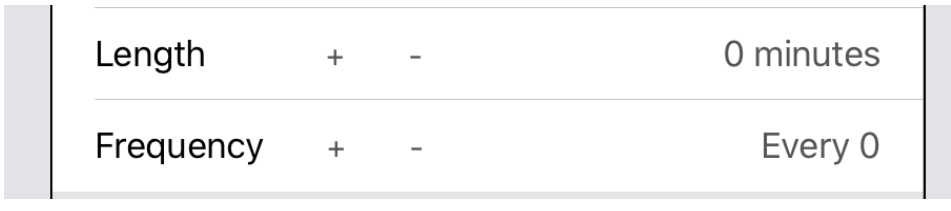
For "Work", "Break", "Length", "Frequency", "Completed Today", and "Completed All-Time", add a second label to the cell that holds the value of the setting item. Set the text color to "dark gray" and add constraints: "Vertically in container" and "Standard tailing space".



For “Enable Long Break”, “Enable Vibration”, “Enable Sound”, and “Prevent Screen Auto-Lock”, add a Switch to the cell and add constraints: “Vertically in container” and “Standard tailing space”.



For “Work”, “Break”, “Length”, and “Frequency”, we need to add a plus and a minus button. Add 2 buttons each to their container views and change the text color to “Dark Gray Color”. Then click “Embed In” in the lower right corner and choose “Embed in stack view”. In the attribute inspector, choose “standard spacing”. For the stack view, add constraints: “Vertically in container” and “leading space 60”.



IV. VIEW CONTROLLER SETUP

Just like what we did with the timer view, the settings view needs a view controller. Go to **File > New > File** and select **Cocoa Touch Class**. Change the class name to SettingsTableViewController, subclass of UITableViewController. Click next and put the file under the group PowerFocus. Then click create to confirm. You will get a file like this:

```
import UIKit

class SettingsTableViewController: UITableViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between presentations
        // self.clearsSelectionOnViewWillAppear = false

        // Uncomment the following line to display an Edit button in the navigation bar for this
view controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        return 0
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // #warning Incomplete implementation, return the number of rows
```

```
        return 0
    }
    . . .
}
```

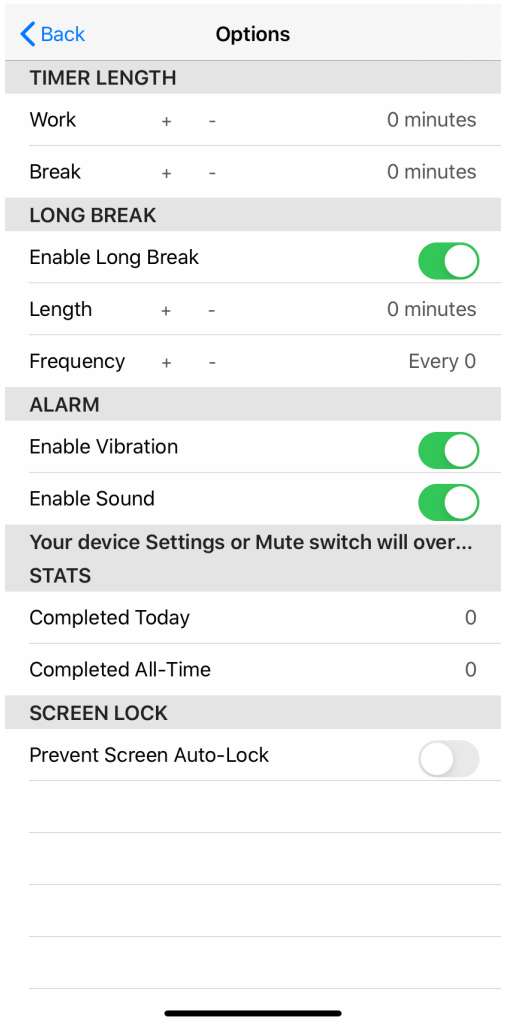
Go to Main.storyboard and select Settings Table View Controller. In **Identity inspector**, change the class to SettingsTableViewController.

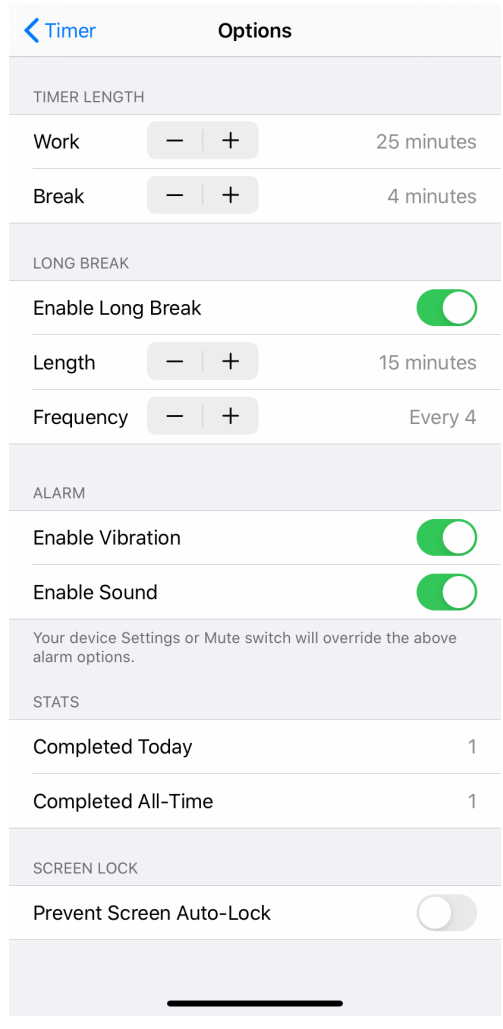
Since the table view is a static one, we don't need the methods func numberOfSections(in tableView: UITableView) -> Int and func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int. You can remove them from the view controller, along with other unnecessary commented methods for static table view.

Now launch the app again and make sure there's no crash because of the added SettingsTableViewController.

EXERCISE #1

If you compare the current UI to the expected UI, you will notice that the style of the headers and footer is not correct. The headers and footers of a table view can be modified in its view controller. So go to SettingsTableViewController and make the changes.





To change the header/footer of a table view, you can override these functions:

```
 func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView?
func tableView(_ tableView: UITableView, viewForFooterInSection section: Int) -> UIView?
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String?
func tableView(_ tableView: UITableView, titleForFooterInSection section: Int) -> String?
```

In those methods, you can create a view by program like this:

```
        let headerView = UIView(frame: CGRect(x: 0, y: 0, width: tableView.bounds.size.width,
height: 50))
        headerView.backgroundColor = .clear
        let label = UILabel(frame: CGRect(x: 0, y: 0, width: 200, height: 40))
        label.text = "some text"
        label.textAlignment = .left
        label.textColor = .darkGray
        label.font = label.font.withSize(15.0)
        headerView.addSubview(label)
```

You should change the background color, the font size and the text color to match the expected style.

EXERCISE #2

After the modifications of Exercise #1, the UI still doesn't look perfect. Here are some possible improvements:

1. Set the height of section headers to 35 (or to a value that looks good)
2. Set the height of section footers to 80 (or to a value that looks good)
3. Set the isScrollEnabled property of the table view to false to disable scrolling
4. Hide the separators after the defined sections

Hint:

1. To create a color from RGB values:

```
let color = UIColor(red: 241.0/255.0, green: 241.0/255.0, blue: 246.0/255.0, alpha: 1.0)
```

2. A simple way to hide separators in a table view:

```
tableView.tableFooterView = UIView(frame: CGRect.zero)
```

SUMMARY

In this chapter, we have worked on these topics:

- design a setting page
- create and link a static table view in storyboard
- customize settings table view
- set up table view controller

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.

If you can't figure out the exercises on your own, you can check the answers at

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

For reference, you can download the complete Xcode project from

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