HW 10: Navigation Controllers

In this assignment you will use a UINavigationController to add a drill-down interface to Homeowner that lets the user see and edit the details of an Item. These details will be presented by the DetailViewController that you created in the previous assignment.

1. Ensure you commit and push, then submit your commit ID for HW 9 before moving on with this lab. Also, ensure you create a new repository for this lab, remove your remote for HW 9, add a new remote for HW 10, then do a push (also don't forget to add collaborators).

2. Open Main.storyboard and select the Items View Controller. Then, from the Editor menu, choose Embed In → Navigation Controller. This will set the ItemsViewController to be the root view controller of a UINavigationController. It will also update the storyboard to set the Navigation Controller as the initial view controller.

3. Your Detail View Controller interface may have misplaced views now that it is contained within a navigation controller. If it does, select the stack view and click the Update Frames button in the Auto Layout constraint menu.

4. Build and run the application and … the application crashes. What is happening? You previously created a contract with the AppDelegate that an instance of ItemsViewController would be the rootViewController of the window. You have now broken this contract by embedding the ItemsViewController in a UINavigationController. You need to update the contract. Open AppDelegate.swift and update application(_:didFinishLaunchingWithOptions:) to reflect the new view controller hierarchy.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey : Any]?) -> Bool {
    // Override point for customization after application launch.

    // Create an ItemStore
    let itemStore = ItemStore()

    // Access the ItemsViewController and set its item store
    ~~let itemsController = window!.rootViewController as! ItemsViewController~~
    let navController = window!.rootViewController as! UINavigationController
    let itemsController = navController.topViewController as! ItemsViewController
    itemsController.itemStore = itemStore

    return true
}
```

5. Build and run the application again. Homeowner works again and has a very nice, if totally empty, UINavigationBar at the top of the screen.

6. In DetailViewController.swift, implement viewWillDisappear(_:).

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // "Save" changes to item
    item.name = nameField.text ?? ""
    item.serialNumber = serialNumberField.text

    if let valueText = valueField.text,
        let value = numberFormatter.number(from: valueText) {
        item.valueInDollars = value.intValue
    } else {
        item.valueInDollars = 0
    }
}
```

7. In ItemsViewController.swift, override viewWillAppear(_:) to reload the table view.

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    tableView.reloadData()
}
```

8. Build and run your application once again. Now you can move back and forth between the view controllers that you created and change the data with ease.

9. To have the text field resign in response to the Return key being pressed, you are going to implement the UITextFieldDelegate method textFieldShouldReturn(_:). This method is called whenever the Return key is pressed. First, in DetailViewController.swift, have DetailViewController conform to the UITextFieldDelegate protocol.

   class DetailViewController: UIViewController**, UITextFieldDelegate** {

10. Next, implement textFieldShouldReturn(_:) to call resignFirstResponder() on the text field that is passed in.

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
}
```

11. Open Main.storyboard and connect the delegate property of each text field to the Detail View Controller (Control-drag from each UITextField to the Detail View Controller and choose delegate).

12. Build and run the application. Tap a text field and then press the Return key on the keyboard. The keyboard will disappear. To get the keyboard back, touch any text field.

13. Open Main.storyboard and find Tap Gesture Recognizer in the object library. Drag this object onto the background view for the Detail View Controller. You will see a reference to this gesture recognizer in the scene dock.

14. In the project navigator, Option-click DetailViewController.swift to open it in the assistant editor. Control-drag from the tap gesture recognizer in the storyboard to the implementation of DetailViewController.

15. In the pop-up that appears, select Action from the Connection menu. Name the action backgroundTapped. For the Type, choose UITapGestureRecognizer. Click Connect and the stub for the action method will appear in DetailViewController.swift.

16. Update the method to call endEditing(_:) on the view of DetailViewController.

```
@IBAction func backgroundTapped(_ sender: UITapGestureRecognizer) {
    view.endEditing(true)
}
```

17. Build and run your application. Tap on a text field to show the keyboard. Tap on the view outside of a text field and the keyboard will disappear.

18. There is one final case where you need to dismiss the keyboard. When the user taps the Back button, viewWillDisappear(_:) is called on the DetailViewController before it is popped off the stack, and the keyboard disappears instantly. To dismiss the keyboard more smoothly, update the implementation of viewWillDisappear(_:) in DetailViewController.swift to call endEditing(_:).

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // Clear first responder
    view.endEditing(true)

    // "Save" changes to item
    item.name = nameField.text ?? ""
    item.serialNumber = serialNumberField.text
    if let valueText = valueField.text,
        let value = numberFormatter.number(from: valueText) {
        item.valueInDollars = value.integerValue
    } else {
        item.valueInDollars = 0
    }
}
```

19. Open Main.storyboard. Double-click on the center of the navigation bar above the Items View Controller to edit its title. Give it a title of “Homeowner”.

20. Build and run the application. Notice the string Homeowner on the navigation bar. Create and tap on a row and notice that the navigation bar no longer has a title. It would be nice to have the DetailViewController’s navigation item title be the name of the Item it is displaying. Because the title will depend on the Item that is being displayed, you need to set the title of the navigationItem dynamically in code.

21. In DetailViewController.swift, add a property observer to the item property that updates the title of the navigationItem.

```
var item: Item! {
   didSet {
      navigationItem.title = item.name
   }
}
```

22. Build and run the application. Create and tap a row and you will see that the title of the navigation bar is the name of the Item you selected.

23. In ItemsViewController.swift, update the method signature for addNewItem(_:).

```
~~@IBAction func addNewItem(_ sender: UIButton) {~~
@IBAction func addNewItem(_ sender: UIBarButtonItem) {
   ...
}
```

24. Now open Main.storyboard and then open the object library. Drag a Bar Button Item to the right side of Items View Controller’s navigation bar. Select this bar button item and open its attributes inspector. Change the System Item to Add.

25. Control-drag from this bar button item to the Items View Controller and select addNewItem.

26. Build and run the application. Tap the + button and a new row will appear in the table.

27. In ItemsViewController.swift, override the init(coder:) method to set the left bar button item.

```
required init?(coder aDecoder: NSCoder) {
   super.init(coder: aDecoder)

   navigationItem.leftBarButtonItem = editButtonItem
}
```

28. Build and run the application, add some items, and tap the Edit button. The UITableView enters editing mode! The editButtonItem property creates a UIBarButtonItem with the title Edit. Even better, this button comes with a target-action pair: It calls the method setEditing(_:animated:) on its UIViewController when tapped.

29. Open Main.storyboard. Now that Homepwner has a fully functional navigation bar, you can get rid of the header view and the associated code. Select the header view on the table view and press Delete.

30. Also, the UINavigationController will handle updating the insets for the table view. In ItemsViewController.swift, delete the following code.

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Get the height of the status bar
    let statusBarHeight = UIApplication.shared.statusBarFrame.height

    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0, right: 0)
    tableView.contentInset = insets
    tableView.scrollIndicatorInsets = insets

    tableView.rowHeight = UITableViewAutomaticDimension
    tableView.estimatedRowHeight = 65
}
```

31. Remove the toggleEditingMode(_:) method.

```
@IBAction func toggleEditingMode(_ sender: UIButton) {
    // If you are currently in editing mode...
    if isEditing {
        // Change text of button to inform user of state
        sender.setTitle("Edit", for: .normal)

        // Turn off editing mode
        setEditing(false, animated: true)
    } else {
        // Change text of button to inform user of state
        sender.setTitle("Done", for: .normal)

        // Enter editing mode
        setEditing(true, animated: true)
    }
}
```

32. Build and run again. The old Edit and Add buttons are gone, leaving you with a lovely UINavigationBar.

Final App