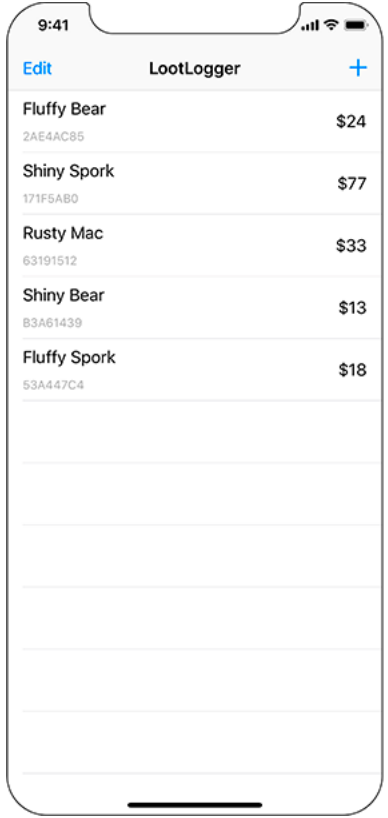
Navigation Controllers

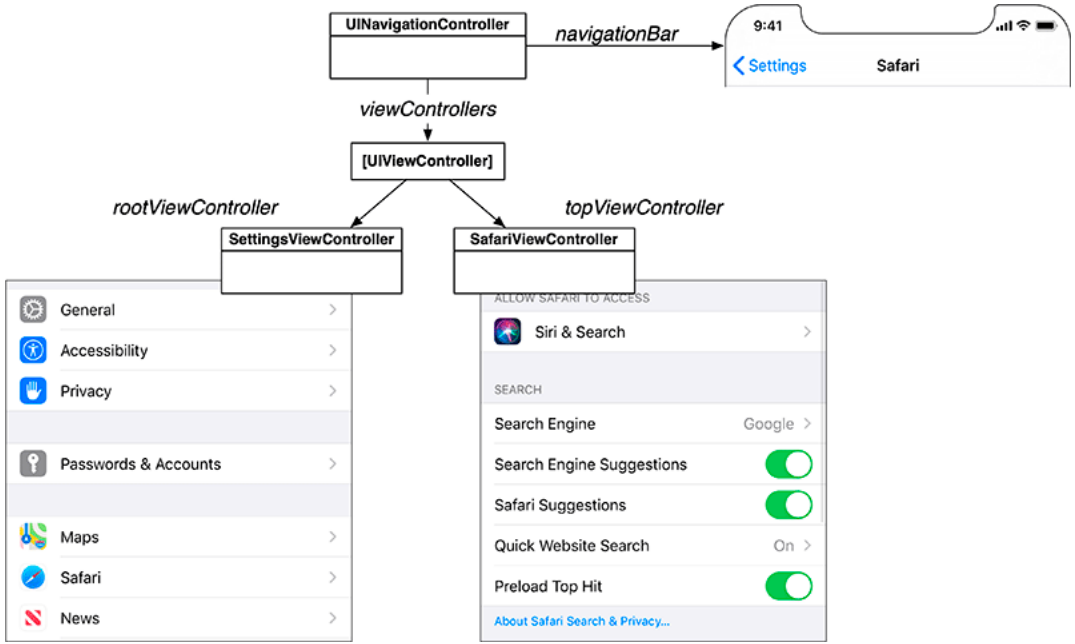
Navigation Controller

drill-down interface in LootLogger
that lets the user see and edit the details of items.



Navigation Controller

A **UINavigationController** maintains an array of view controllers presenting related information in a stack. When a UIViewController is on top of the stack, its view is visible.

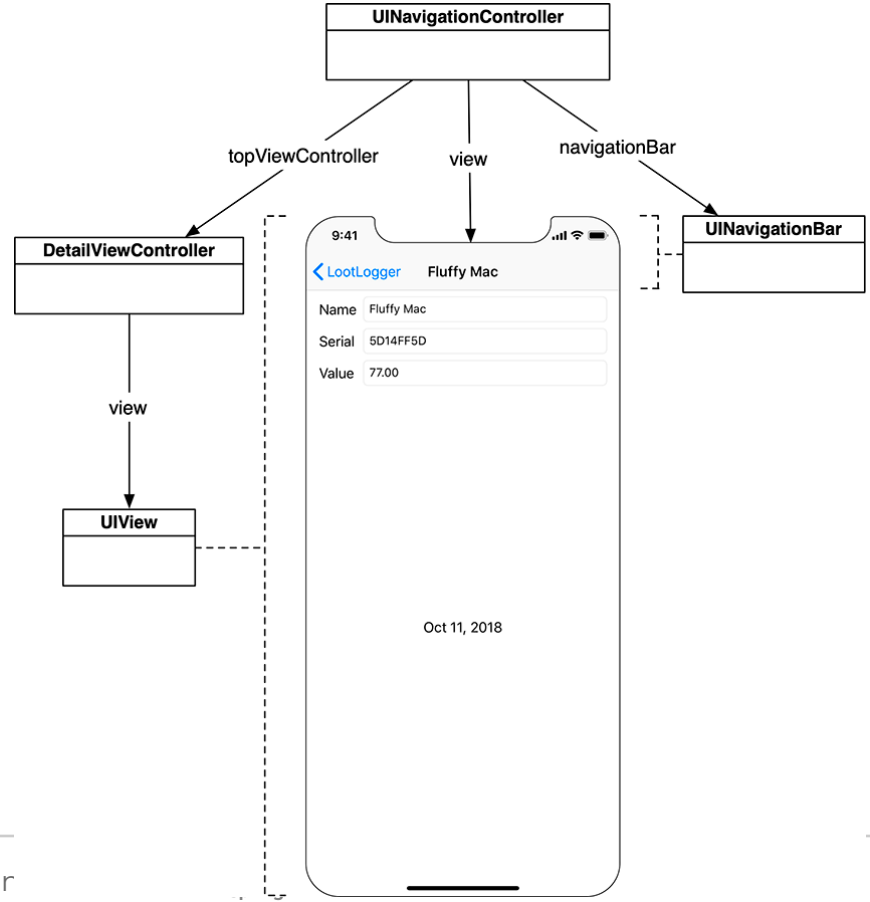


Navigation Controller

More view controllers can be pushed on top of the UINavigationController's stack while the application is running. These view controllers are added to the end of the viewControllers array that corresponds to the top of the stack. UINavigationController's topViewController property keeps a reference to the view controller at the top of the stack.

When a view controller is pushed onto the stack, its view slides onscreen from the right. When the stack is popped (i.e., the last item is removed), the top view controller is removed from the stack and its "view slides off to the right, exposing the view of the next view controller on the stack, which becomes the top view controller.

Navigation Controller



Navigation Controller

UINavigationController in the LootLogger will make the ItemsViewController the UINavigationController's root view controller.

The DetailViewController will be pushed onto the UINavigationController's stack when an Item is selected. This view controller will allow the user to view and edit the properties of an Item selected from the table view of ItemsViewController.

Open LootLogger Project.

The only requirements for using a UINavigationController are that you give it a root view controller and add its view to the window.

Navigation Controller

“Open Main.storyboard and select the Items View Controller. Then, from the Editor menu, choose Embed In → Navigation Controller (this can also be done from the button in the bottom right). 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.

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.

“Build and run the application and … the application crashes.

Navigation Controller

You previously created a contract with the SceneDelegate 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.

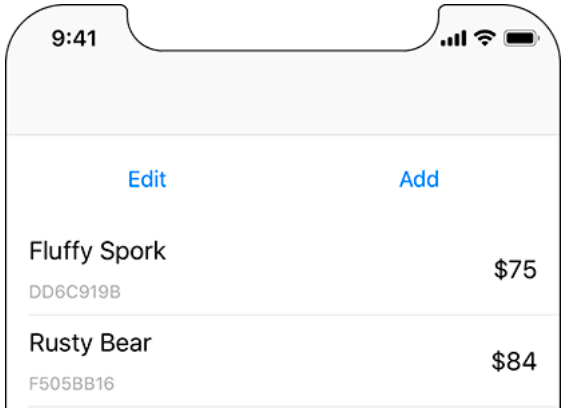
Open SceneDelegate.swift (if Xcode has not opened it for you) and update scene(_:willConnectTo:options:) to reflect the new view controller hierarchy.

Navigation Controller

```
func scene(_ scene: UIScene,
        willConnectTo session: UISceneSession,
        options connectionOptions: UIScene.ConnectionOptions) {
    guard let _ = (scene as? UIWindowScene) else { return }

    // Create an ItemStore
    let itemStore = ItemStore()


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

Navigation Controller

Build and run the application again.



Navigation Controller

With the application still running, create a new item and select that row from the UITableView. Not only are you taken to DetailViewController's view, but you also get a free animation and a Back button in the UINavigationBar. Tap this button to get back to ItemsViewController.

Having a view controller push the next view controller is a common pattern. The root view controller typically creates the next view controller, and the next view controller creates the one after that, and so on. Some applications may have view controllers that can push different view controllers depending on user input. For example, the Photos app pushes a video view controller or an image view controller onto the navigation stack depending on what type of media is selected.

Navigation Controller

Whenever a UINavigationController is about to swap views, it calls two methods: viewWillDisappear(_:) and viewWillAppear(_:). The UIViewController that is about to be popped off the stack has viewWillDisappear(_:) called on it. The UIViewController that will then be on top of the stack has viewWillAppear(_:) call

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
    }
```

Navigation Controller

Now the values of the Item will be updated when the user taps the Back button on the UINavigationBar.

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

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

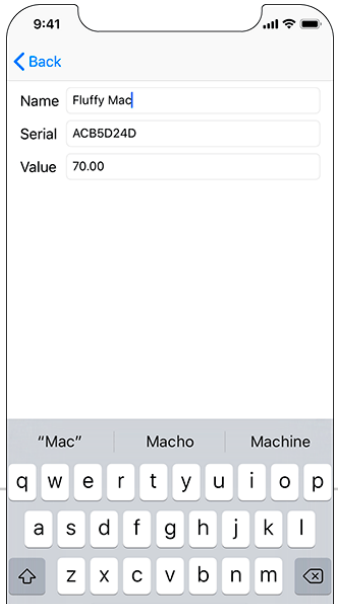
Build and run the app.
Now you can add items, move back and forth between the view controllers that you created, and change the data with ease.

Dismissing the Keyboard

Run the application, add and select an item, and touch the text field with the item's name. When you touch the text field, a keyboard appears onscreen.

The appearance of the keyboard in response to a touch is built into the UITextField class as well as UITextView.

you are going to give the user two ways to dismiss the keyboard: pressing the keyboard's Return key and tapping anywhere else on the detail view controller's view.



Dismissing the Keyboard

If you touch another text field in the application, that text field will become the first responder, and the keyboard will stay onscreen. The keyboard will only give up and go away when no text field (or text view) is the first responder. To dismiss the keyboard, then, you call resignFirstResponder() on the text field that is the first responder.

Dismissing the Keyboard

in DetailViewController.swift, have DetailViewController conform to the UITextFieldDelegate protocol.

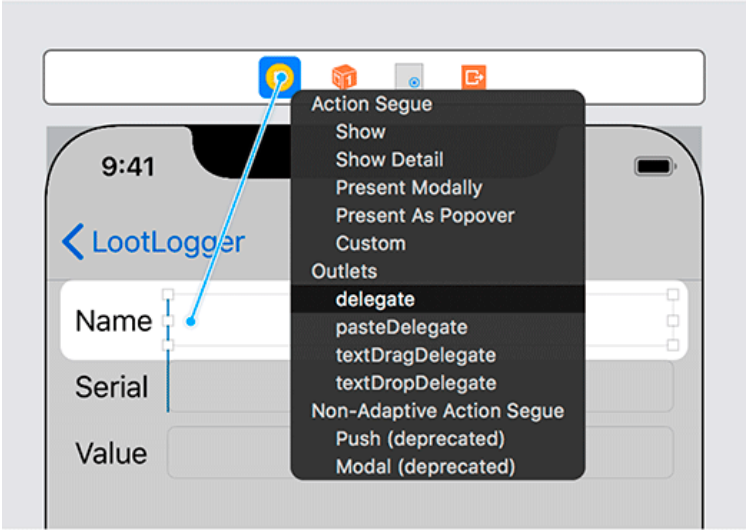
class DetailViewController: UIViewController, **UITextFieldDelegate {**

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

Dismissing the Keyboard

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.



Dismissing the Keyboard

Build and run the application. Add an item and drill down to its detail view. Tap a text field and then press the Return key on the keyboard. “The keyboard will disappear. To get the keyboard back, tap any text field.

Dismissing the Keyboard

To dismiss the keyboard if the user taps anywhere else on DetailViewController's view.
To do this, you are going to use a gesture recognizer when the view is tapped, just as you did in the WorldTrotter app. In the action method, you will call resignFirstResponder() on the text field.

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. (
Make sure you drag onto the VIEW, not on the DateCreated)

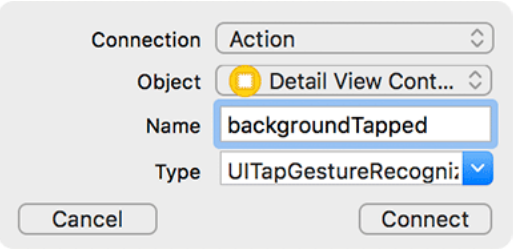
Dismissing the Keyboard

In the project navigator, Option-click DetailViewController.swift to open it in an additional editor.

Control-drag from the tap gesture recognizer in the storyboard to the implementation of DetailViewController.

In the panel 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.



Dismissing the Keyboard

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

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

Calling endEditing(_:) is a convenient way to dismiss the keyboard without having to know (or care) which text field is the first responder. When the view gets this call, it checks whether any text field in its hierarchy is the first responder. If so, then resignFirstResponder() is called on that particular view.

Build and run your application, add an item, and tap it. Tap a text field to show the keyboard. Tap the view outside of a text field, and the keyboard will disappear.

Dismissing the Keyboard

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, with no animation. To dismiss the keyboard more smoothly, update the implementation of viewWillDisappear(_:) in DetailViewController.swift to call endEditing(_:).

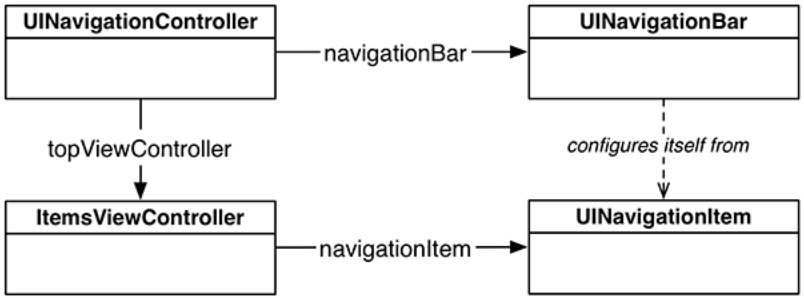
Dismissing the Keyboard

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.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
    }
```

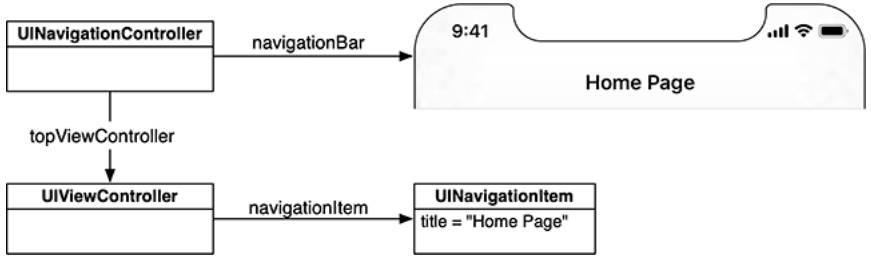
UINavigationBar

Every UIViewController has a navigationItem property of type UINavigationItem. However, unlike UINavigationBar, UINavigationItem is not a subclass of UIView, so it cannot appear on the screen. Instead, the navigation item supplies the navigation bar with the content it needs to draw. When a UIViewController comes to the top of a UINavigationController's stack, the UINavigationBar uses the UIViewController's navigationItem to configure itself.



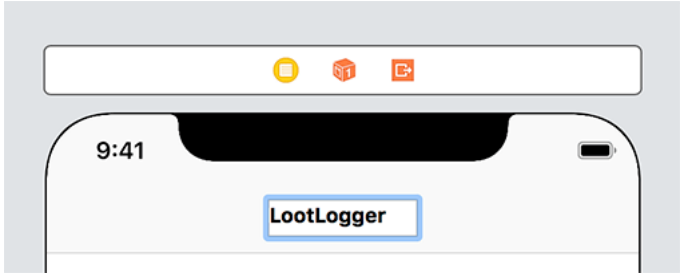
UINavigationBar

By default, a UINavigationItem is empty. At the most basic level, a UINavigationItem has a simple title string. When a UIViewController is moved to the top of the navigation stack and its navigationItem has a valid string for its title property, the navigation bar will display that string



UINavigationBar

Open Main.storyboard. Drag a Navigation Item from the library on top of the Items View Controller. Double-click the center of the navigation bar above the Items View Controller to edit its title. Give it a title of LootLogger.



Build and run the application. Notice the string LootLogger on the navigation bar. Create and tap 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.

UINavigationBar

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
    }
}
```

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

UINavigationBar

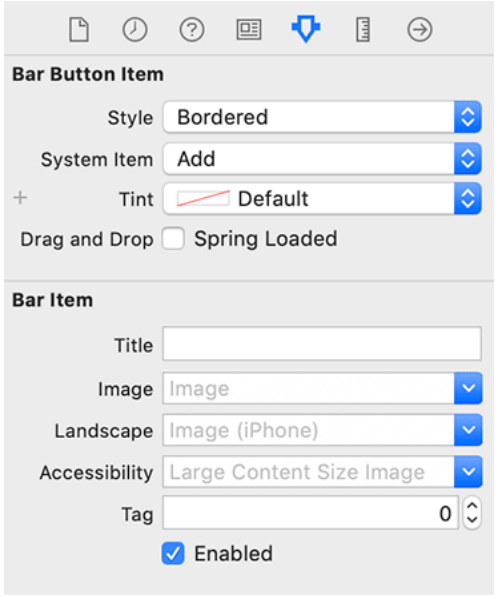
To replace the two buttons that are in the table's header view with two bar button items that will appear in the UINavigationBar when the ItemsViewController is on top of the stack. A bar button item has a target-action pair that works like UIControl's target-action mechanism: When tapped, it sends the action message to the target.

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

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

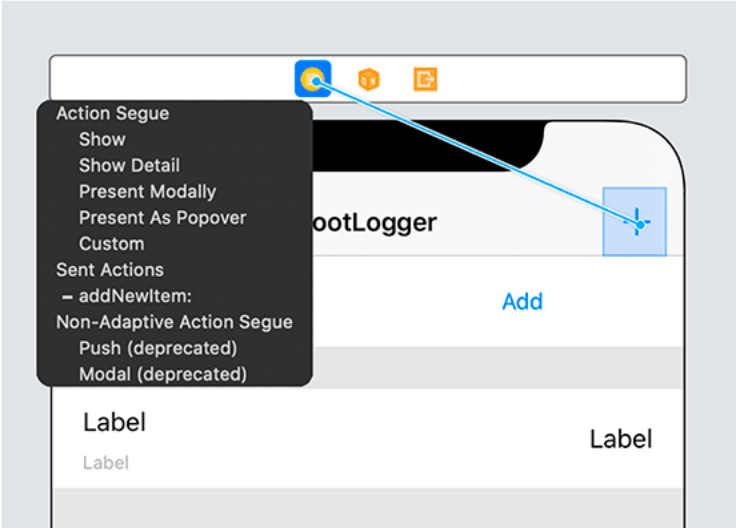
UINavigationBar

open Main.storyboard. Open the object library and drag a Bar Button Item to the right side of the items view controller's navigation bar. Select this bar button item and open its attributes inspector. Change the System Item to Add



UINavigationBar

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



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

UINavigationBar

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
}
```

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.

UINavigationBar

“Open Main.storyboard. Now that LootLogger 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.

UINavigationBar

Finally, in ItemsViewController.swift, 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)
    }
}
```

UINavigationBar

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