Implement Edit and Delete Behavior

On This Page

In this lesson, you focus on adding behavior that allows the user to edit and delete meals in the FoodTracker app.

Learning Objectives

At the end of the lesson, you'll be able to:

- Differentiate between push and modal navigation
- Dismiss view controllers based on their presentation style
- Understand when to use different type cast operators for downcasting
- Leverage optional binding to check for complex conditions
- Use segue identifiers to determine which segue is occurring

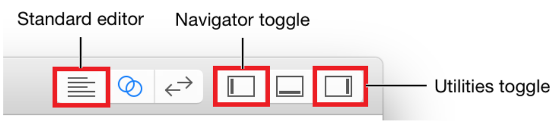
Allow Editing of Existing Meals

Currently, the FoodTracker app gives users the ability to add a new meal to a list of meals. Next, you'll want to give users the ability to edit an existing meal.

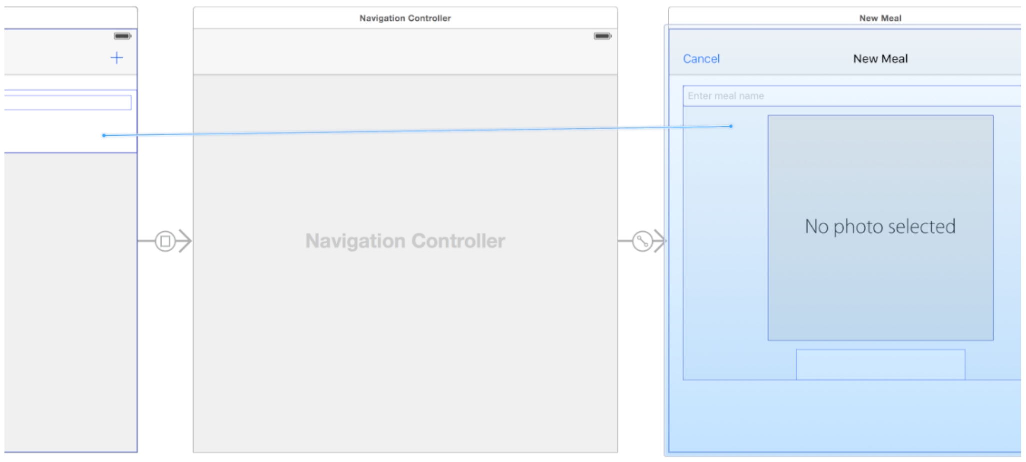
You'll enable the user to tap on a meal cell to pull up a version of the meal scene that's prepopulated with information about a meal. The user makes changes and taps the Save button, which updates the information and overwrites the previous entry in the meal list.

To configure the table view cell

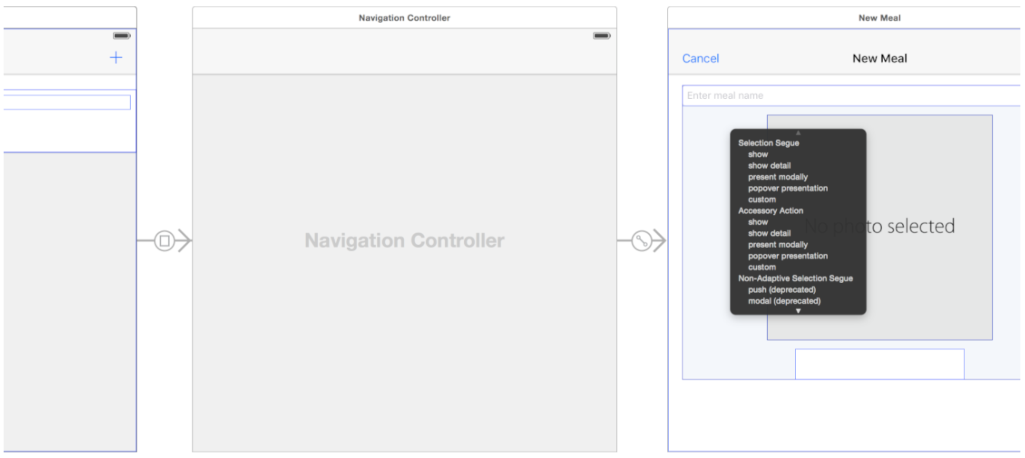
1. If the assistant editor is open, return to the standard editor by clicking the Standard button.



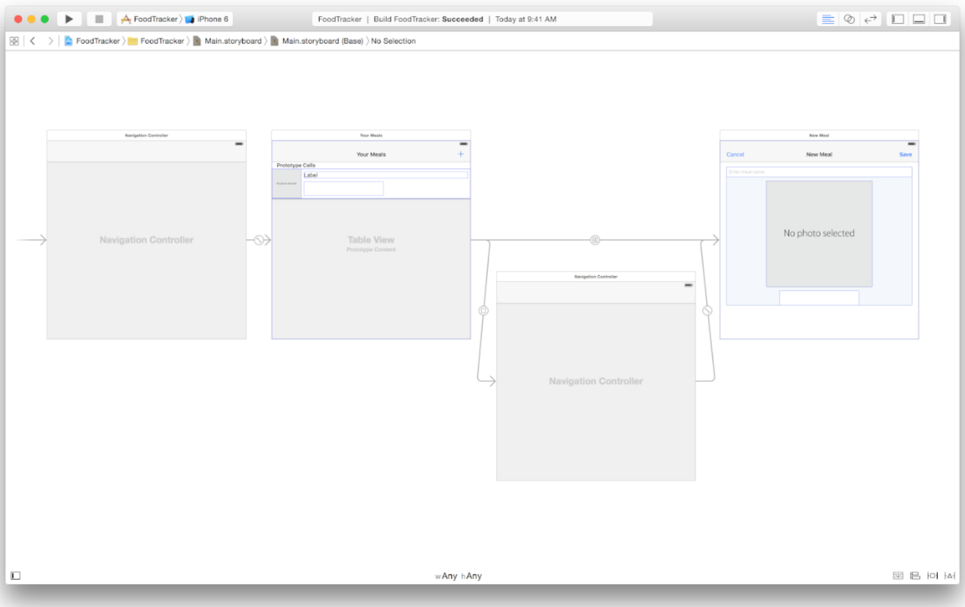
2. Open your storyboard, `Main.storyboard`.
3. On the canvas, select the table view cell.
4. Control-drag from the table view cell to the meal scene.



A shortcut menu titled Selection Segue appears in the location where the drag ended.

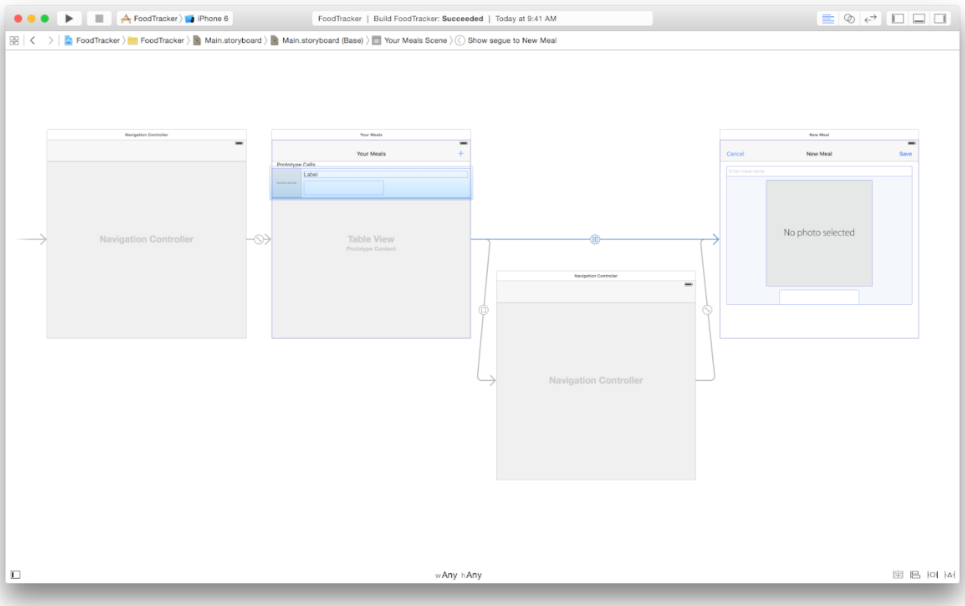


5. Choose show from the Selection Segue menu.
6. Drag the navigation controller between the meal list and the meal scene down so you can see the new segue.

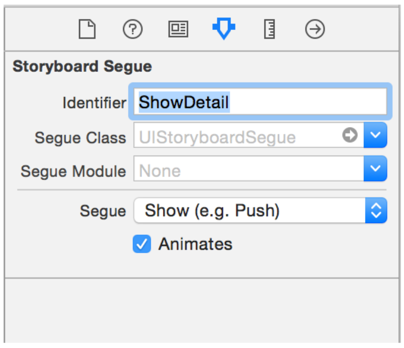


If you want, you can zoom out using Command-Minus (–).

7. On the canvas, select the newly added segue.


FoodTracker 〉 iPhone 6
FoodTracker | Build FoodTracker: Succeeded | Today at 9:41 AM
FoodTracker 〉 FoodTracker 〉 Main.storyboard 〉 Main.storyboard (Base) 〉 Your Meals Scene 〉 Show segue to New Meal
Navigation Controller
Your Meals
Table View
Prototype Content
New Meal
Cancel
Save
No photo selected
Navigation Controller
wAny hAny

8. In the Attributes inspector, type `ShowDetail` in the Identifier field. Press Return.


Storyboard Segue

Identifier: ShowDetail

Segue Class: UIStoryboardSegue

Segue Module: None

Segue: Show (e.g. Push)

✓ Animates

When this segue is triggered, it pushes the view controller for the meal scene onto the same navigation stack as the meal list scene.

Checkpoint: Run your app. In the meal list scene, you should be able to tap a table view cell to navigate to the meal scene, but the content in the scene is blank. When you tap on an existing cell in the table view, you want to edit an existing meal, not create a new one.

You now have two segues that go to the same scene, so you need a way to identify when the user is trying to add a new meal or edit an existing one.

Recall from earlier that the `prepareForSegue(_:sender:)` method is called before any segue gets executed. You can use this method to identify which segue is occurring, and display the appropriate information in the meal scene. You'll differentiate the segues based on the identifiers you assigned to them earlier: AddItem (modal segue) and ShowDetail (show segue).

To identify which segue is occurring

1. Open `MealTableViewController.swift`.
2. In `MealTableViewController.swift`, find and uncomment the `prepareForSegue(_:sender:)` method. (To uncomment the method, remove the `/*` and `*/` characters surrounding it.)

   After you do that, the template implementation looks like this:

```
// MARK: - Navigation

// In a storyboard-based application, you will often want to do a little
  preparation before navigation
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

    // Get the new view controller using segue.destinationViewController.
    // Pass the selected object to the new view controller.
}
```

Because `MealTableViewController` is a subclass of `UITableViewController`, the template implementation comes with a skeleton for `prepareForSegue(_:sender:)`.

3. Delete the two lines of comments, and replace them with this `if` statement and `else` clause:

   ```
   if segue.identifier == "ShowDetail" {
   }
   else if segue.identifier == "AddItem" {
   }
   ```

   This code compares the segue identifiers against the identifier strings assigned to them earlier.

4. In the first `if` statement (which gets executed if the meal is being edited), add the following code:

   ```
   let mealDetailViewController = segue.destinationViewController as!
     MealViewController
   ```

   This code tries to downcast the destination view controller of the segue to a `MealViewController` using the forced type cast operator (`as!`). Notice that this operation has an exclamation mark (`!`) instead of a question mark (`?`) at the end, like you've seen so far with type cast operators. This means that the operator performs a forced type cast. If the cast is successful, the local constant `mealDetailViewController` gets assigned the value of `segue.destinationViewController` cast as type `MealViewController`. If the cast is unsuccessful, the app should crash at runtime.

   Only use a forced cast if you're absolutely certain that the cast will succeed—and that if it fails, something has gone wrong in the app and it should crash. Otherwise, downcast using `as?`.

5. Below the previous line, add another `if` statement (nested inside the first one):

   ```
   // Get the cell that generated this segue.
   if let selectedMealCell = sender as? MealTableViewCell {
   }
   ```

   This code tries to downcast `sender` to a `MealCell` using the optional type cast operator (`as?`). If the cast is successful, the local constant `selectedMealCell` gets assigned the value of `sender` cast as type `MealTableViewCell`, and the `if` statement proceeds to execute. If the cast is unsuccessful, the expression evaluates to `nil` and the `if` statement isn't executed.

6. Inside the `if` statement, add this code:

   ```
   let indexPath = tableView.indexPathForCell(selectedMealCell)!
   let selectedMeal = meals[indexPath.row]
   mealDetailViewController.meal = selectedMeal
   ```

   This code fetches the `Meal` object corresponding to the selected cell in the table view. It then assigns that `Meal` object to the `meal` property of the destination view controller, an instance of `MealViewController`. (You'll configure `MealViewController` to display the information from its `meal` property when it loads.)

7. Inside the `else` clause you added earlier, add this `print` statement:

   ```
   print("Adding new meal.")
   ```

   Although you don't need to do anything in this method if you're adding a new meal instead of editing an existing one, it's useful to log what's going on.

Your `prepareForSegue(_:sender:)` method should look something like this:

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowDetail" {
        let mealDetailViewController = segue.destinationViewController as!
```

```
  MealViewController

        // Get the cell that generated this segue.
        if let selectedMealCell = sender as? MealTableViewCell {
            let indexPath = tableView.indexPathForCell(selectedMealCell)!
            let selectedMeal = meals[indexPath.row]
            mealDetailViewController.meal = selectedMeal
        }
    }
    else if segue.identifier == "AddItem" {
        print("Adding new meal.")
    }
}
```

Now that you have the logic implemented, you need to do a small amount of work in `MealViewController.swift` to make sure the UI updates correctly. Specifically, when an instance of `MealViewController` (the meal scene) gets created, its views should be populated with data from its `meal` property, if that data exists. Recall that the appropriate place to do this type of setup work is in the `viewDidLoad()` method.

To update the implementation of viewDidLoad

1. Open `MealViewController.swift`.
2. In `MealViewController.swift`, find the `viewDidLoad()` method.

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

       // Handle the text field’s user input via delegate callbacks.
       nameTextField.delegate = self

       // Enable the Save button only if the text field has a valid Meal name.
       checkValidMealName()
   }
   ```

3. Below the `nameTextField.delegate` line, add this code:

   ```
   // Set up views if editing an existing Meal.
   if let meal = meal {
       navigationItem.title = meal.name
       nameTextField.text   = meal.name
       photoImageView.image = meal.photo
       ratingControl.rating = meal.rating
   }
   ```

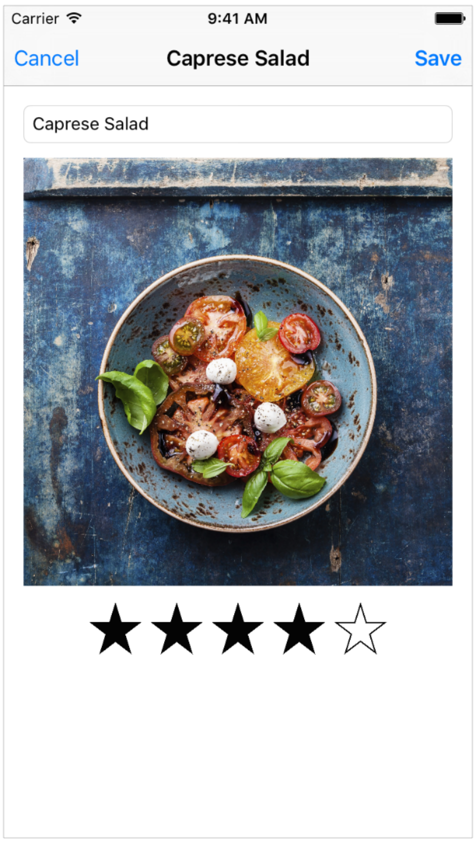
   This code sets each of the views in `MealViewController` to display data from the `meal` property if the meal property is non-`nil`, which happens only when an existing meal is being edited.

Your `viewDidLoad()` method should look something like this:

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

    // Handle the text field’s user input via delegate callbacks.
    nameTextField.delegate = self

    // Set up views if editing an existing Meal.
    if let meal = meal {
        navigationItem.title = meal.name
        nameTextField.text   = meal.name
        photoImageView.image = meal.photo
```

```
            ratingControl.rating = meal.rating
        }

        // Enable the Save button only if the text field has a valid Meal name.
        checkValidMealName()
    }
```

Checkpoint: Run your app. You should be able to click a table view cell to navigate to the meal scene, and see it prepopulated with data about the meal. But if you click Save, instead of overwriting the existing meal, the app adds a new meal. You'll work on getting the right behavior for this next.



To overwrite an existing meal in the meal list, you'll need to update the `unwindToMealList(_:)` action method to handle the two different cases. In one case, you need to add a new meal, and in the other, you need to replace an existing one. Recall that this method is only called when a user taps the Save button, so you don't need to account for the Cancel button in this method.

To update the implementation of unwindToMealList(_:) to add or replace meals

1. Open `MealTableViewController.swift`.
2. In `MealTableViewController.swift`, find the `unwindToMealList(_:)` method.

```
@IBAction func unwindToMealList(sender: UIStoryboardSegue) {
    if let sourceViewController = sender.sourceViewController as?
  MealViewController, meal = sourceViewController.meal {
        // Add a new meal.
        let newIndexPath = NSIndexPath(forRow: meals.count, inSection: 0)
        meals.append(meal)
        tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation:
  .Bottom)
    }
}
```

3. At the beginning of the first `if` statement, add this `if` statement:

```
if let selectedIndexPath = tableView.indexPathForSelectedRow {
}
```

This code checks whether a row in the table view is selected. If it is, that means a user tapped one of the table views cells to edit a meal. In other words, this `if` statement gets executed if an existing meal is being edited.

4. In this `if` statement, add the following code:

```
// Update an existing meal.
meals[selectedIndexPath.row] = meal
tableView.reloadRowsAtIndexPaths([selectedIndexPath], withRowAnimation: .None)
```

The first line updates the appropriate entry in `meals` to store the updated meal information. The second line reloads the appropriate row in the table view to display the changed data.

5. After the `if` statement, add an `else` clause and wrap it around the last four lines in the method, like this:

```
else {
    // Add a new meal.
    let newIndexPath = NSIndexPath(forRow: meals.count, inSection: 0)
    meals.append(meal)
    tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Bottom)
}
```

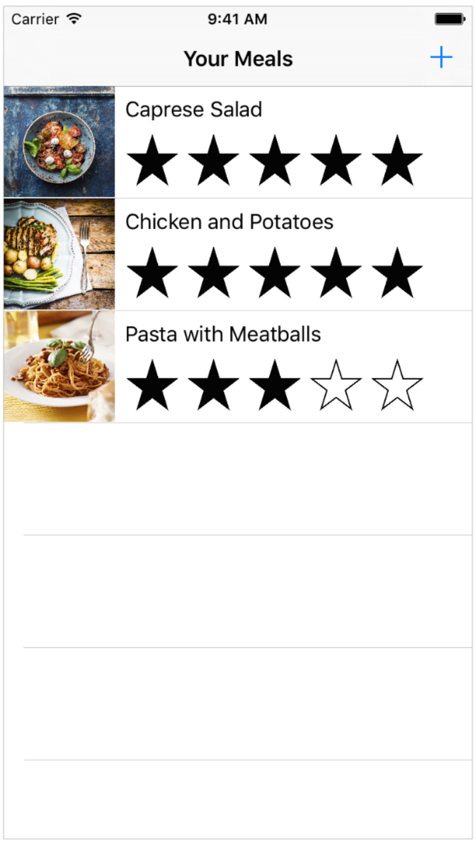
You can make sure the lines in the `else` clause are indented properly by selecting all of them and pressing Control-I.

The `else` clause executes when there's no selected row in the table view, which means a user tapped the Add button to get to the meal scene. In other words, this `else` statement gets executed if a new meal is being added.

Your `unwindToMealList(_:)` action method should look like this:

```
@IBAction func unwindToMealList(sender: UIStoryboardSegue) {
    if let sourceViewController = sender.sourceViewController as? MealViewController, meal = sourceViewController.meal {
        if let selectedIndexPath = tableView.indexPathForSelectedRow {
            // Update an existing meal.
            meals[selectedIndexPath.row] = meal
            tableView.reloadRowsAtIndexPaths([selectedIndexPath], withRowAnimation: .None)
        }
        else {
            // Add a new meal.
            let newIndexPath = NSIndexPath(forRow: meals.count, inSection: 0)
            meals.append(meal)
            tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Bottom)
        }
    }
}
```

Checkpoint: Run your app. You should be able to click a table view cell to navigate to the meal scene, and see it prepopulated with data about the meal. If you click Save, the changes you made should overwrite the existing meal in the list.



Cancel an Edit to an Existing Meal

A user might decide not to keep edits to a meal, and want to return to the meal list without saving any changes. For this, you'll update the behavior of the Cancel button to dismiss the scene appropriately.

The type of dismissal depends on the type of presentation. You'll implement a check that determines how the current scene was presented when the user taps the Cancel button. If it was presented modally (using the Add button), it'll be dismissed using `dismissViewControllerAnimated(_:completion:)`. If it was presented with push navigation (using a table view cell), it will be dismissed by the navigation controller that presented it.

To change the implementation of the cancel action

1. Open `MealViewController.swift`.
2. In `MealViewController.swift`, find the `cancel(_:)` action method.

```
@IBAction func cancel(sender: UIBarButtonItem) {
    dismissViewControllerAnimated(true, completion: nil)
}
```

   This implementation is only using the `dismissViewControllerAnimated` to dismiss the meal scene because you've only had to account for the Add button so far.

3. In the `cancel(_:)` action method, before the existing line of code, add the following code:

```
// Depending on style of presentation (modal or push presentation), this view
  controller needs to be dismissed in two different ways.
let isPresentingInAddMealMode = presentingViewController is
  UINavigationController
```

   This creates a Boolean value that indicates whether the view controller that presented this scene is of type `UINavigationController`. As the constant name `isPresentingInAddMealMode` indicates, this means that the meal scene was presented using the Add button. This is because the meal scene is embedded in its own navigation controller when it's presented in this manner, which means that navigation controller is what presents it.

4. After the line you just added, add the following `if` statement, and move the line that calls `dismissViewControllerAnimated` inside of it:

```
if isPresentingInAddMealMode {
    dismissViewControllerAnimated(true, completion: nil)
}
```

   Whereas before, the call to `dismissViewControllerAnimated` happened anytime the `cancel(_:)` method got called, it now only happens when `isPresentingInAddMealMode` is `true`.

5. Right after the `if` statement, add this `else` clause:

```
else {
    navigationController!.popViewControllerAnimated(true)
}
```

   This is now an `if` statement with an `else` clause that executes the code within the `if` statement only when `isPresentingInAddMealMode` is `true`, and executes the code within the `else` clause otherwise.

   The `else` clause gets executed when the meal scene was pushed onto the navigation stack on top of the meal list scene. The code within the `else` clause executes a method called `popViewControllerAnimated`, which pops the current view controller (meal scene) off the navigation stack of `navigationController` and performs an animation of the transition.

Your `cancel(_:)` action method should look like this:

```
@IBAction func cancel(sender: UIBarButtonItem) {
    // Depending on style of presentation (modal or push presentation), this view
  controller needs to be dismissed in two different ways.
    let isPresentingInAddMealMode = presentingViewController is
  UINavigationController
    
    if isPresentingInAddMealMode {
        dismissViewControllerAnimated(true, completion: nil)
    }
    else {
        navigationController!.popViewControllerAnimated(true)
    }
}
```

Checkpoint: Run your app. Now when you click the Add button (+) and click Cancel instead of Save, you should navigate back to the meal list without adding a new meal.

Support Deleting Meals

Next, you'll want to give users the ability to delete a meal from the meal list. You need a way to let users put the table view into an editing mode from which they can delete cells. You accomplish this by adding an Edit button to the table view's navigation bar.

To add an Edit button to the table view

1. Open `MealTableViewController.swift`.
2. In `MealTableViewController.swift`, find the `viewDidLoad()` method.

```
override func viewDidLoad() {
    super.viewDidLoad()
    
    // Load the sample data.
    loadSampleMeals()
}
```

3. Below the `super.viewDidLoad()` line, add the following line of code:

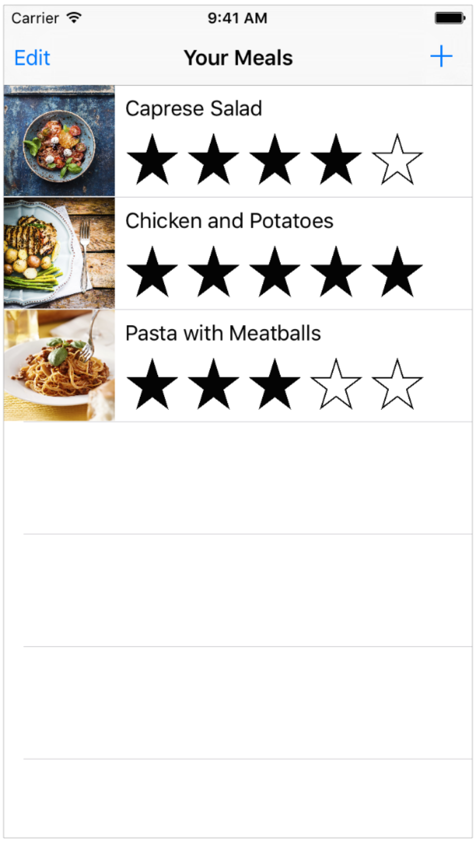
```
// Use the edit button item provided by the table view controller.
navigationItem.leftBarButtonItem = editButtonItem()
```

This creates a special type of bar button item that has editing behavior built into it. It then adds this button to the left side of the navigation bar in the meal list scene.

Your `viewDidLoad()` method should look something like this:

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

    // Use the edit button item provided by the table view controller.
    navigationItem.leftBarButtonItem = editButtonItem()

    // Load the sample data.
    loadSampleMeals()
}
```

Checkpoint: Run your app. Notice there's an Edit button on the left of the table view's navigation bar. If you click the Edit button, the table view goes into editing mode—but you won't be able to delete cells yet, because you haven't implemented that.



To perform any sort of editing on a table view, you need to implement one of its delegate methods, `tableView(_:commitEditingStyle:forRowAtIndexPath:)`. This delegate method is in charge of managing the table rows when it's in editing mode.

You also need to uncomment the implementation of `tableView(_:canEditRowAtIndexPath:)` to support editing.

To delete a meal

1. In `MealTableViewController.swift`, find and uncomment the `tableView(_:commitEditingStyle:forRowAtIndexPath:)` method. (To uncomment the method, remove the `/*` and `*/` characters surrounding it.)

After you do that, the template implementation looks like this:

```
// Override to support editing the table view.
override func tableView(tableView: UITableView, commitEditingStyle
  editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath:
  NSIndexPath) {
    if editingStyle == .Delete {
        // Delete the row from the data source
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    } else if editingStyle == .Insert {
        // Create a new instance of the appropriate class, insert it into the
  array, and add a new row to the table view
    }
}
```

2. Below the comment that says `// Delete the row from the data source`, add the following line of code:

   ```
   meals.removeAtIndex(indexPath.row)
   ```

   This code removes the `Meal` object to be deleted from `meals`. The line after it, which is part of the template implementation, deletes the corresponding row from the table view.

3. In `MealTableViewController.swift`, find and uncomment the `tableView(_:canEditRowAtIndexPath:)` method.

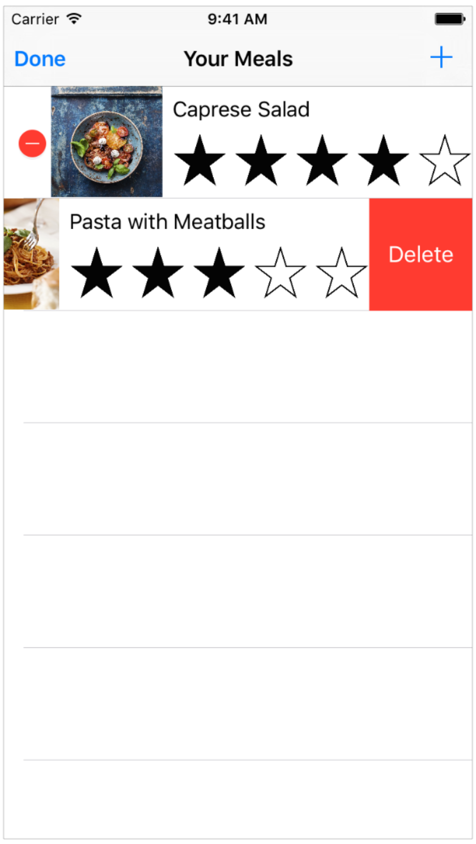
   After you do that, the template implementation looks like this:

```
// Override to support conditional editing of the table view.
override func tableView(tableView: UITableView, canEditRowAtIndexPath
  indexPath: NSIndexPath) -> Bool {
    // Return false if you do not want the specified item to be editable.
    return true
}
```

Your `tableView(_:commitEditingStyle:forRowAtIndexPath:)` method should look like this:

```
// Override to support editing the table view.
override func tableView(tableView: UITableView, commitEditingStyle editingStyle:
  UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {
    if editingStyle == .Delete {
        // Delete the row from the data source
        meals.removeAtIndex(indexPath.row)
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    } else if editingStyle == .Insert {
        // Create a new instance of the appropriate class, insert it into the
  array, and add a new row to the table view
    }
}
```

Checkpoint: Run your app. If you click the Edit button, the table view goes into editing mode. You can choose a cell to delete by clicking the indicator on the left, and confirm that you want to delete it by pressing the Delete button in that cell. Alternatively, swipe left on a cell to expose the Delete button quickly; this behavior is built into table views. When you click the Delete button for a cell, the cell is removed from the list.



NOTE

To see the completed sample project for this lesson, download the file and view it in Xcode.

Download File