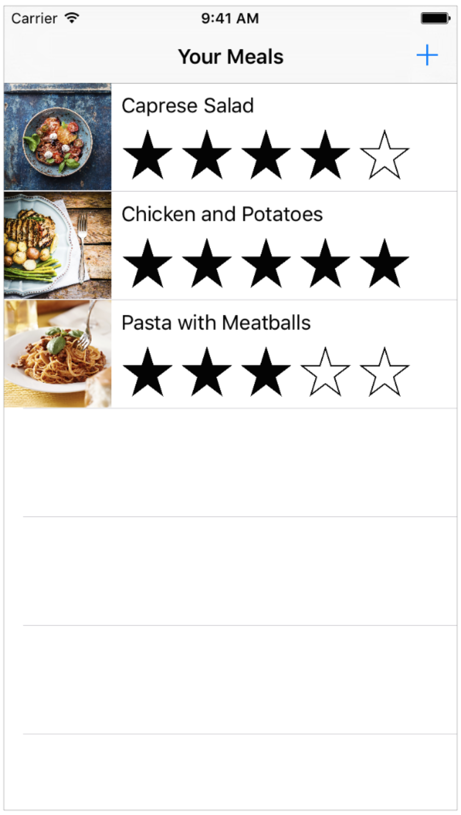
Implement Navigation

On This Page

In this lesson, you use navigation controllers and segues to create the navigation flow of the FoodTracker app. At the end of the lesson, you'll have a complete navigation scheme and interaction flow for the app. When you're finished, your app will look something like this:



Learning Objectives

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

- Embed an existing view controller within a navigation controller in a storyboard
- Create segues between view controllers
- Edit the attributes of a segue in a storyboard using the Attributes inspector
- Pass data between view controllers using the `prepareForSegue(_:sender:)` method
- Perform an unwind segue
- Use stack views to create robust, flexible layouts

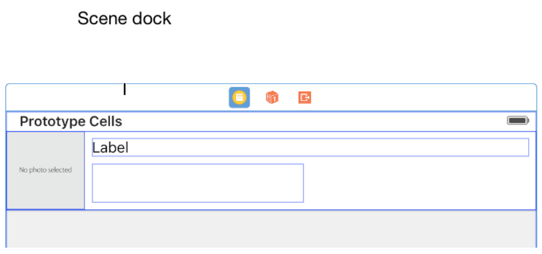
Add a Segue to Navigate Forward

With data displaying as expected, it's time to provide a way to navigate from the meal list to the meal scene. Transitions between scenes are called segues.

Before creating a segue, you need to configure your scenes. First, you'll put your table view controller inside of a navigation controller. A navigation controller manages transitions backward and forward through a series of view controllers. The set of view controllers managed by a particular navigation controller is called its navigation stack. The first item added to the stack becomes the root view controller and is never popped off (removed from) the navigation stack.

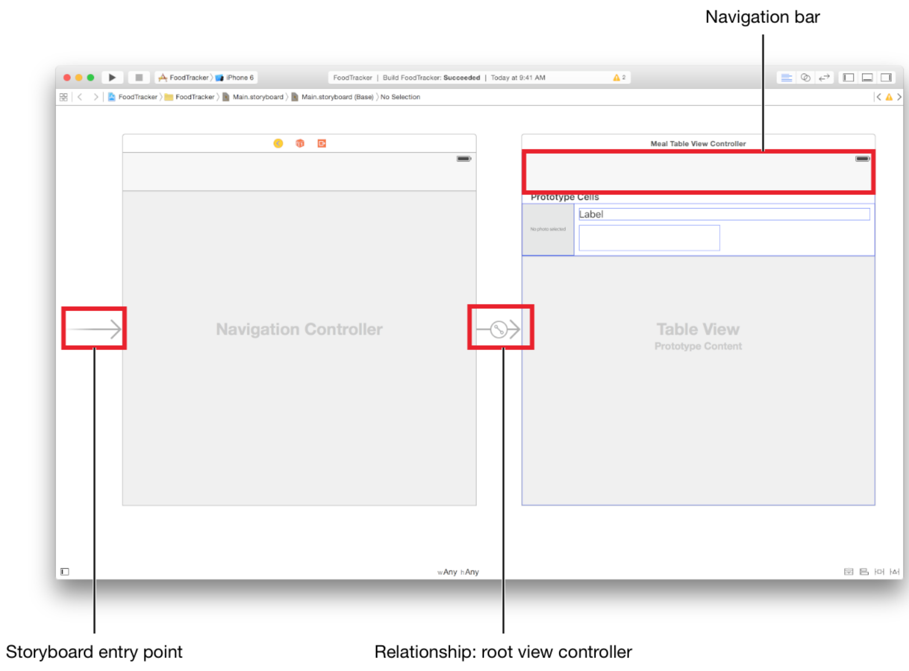
To add a navigation controller to your meal list scene

1. Open your storyboard, `Main.storyboard`.
2. Select the table view controller by clicking on its scene dock.



3. With the table view controller selected, choose Editor > Embed In > Navigation Controller.

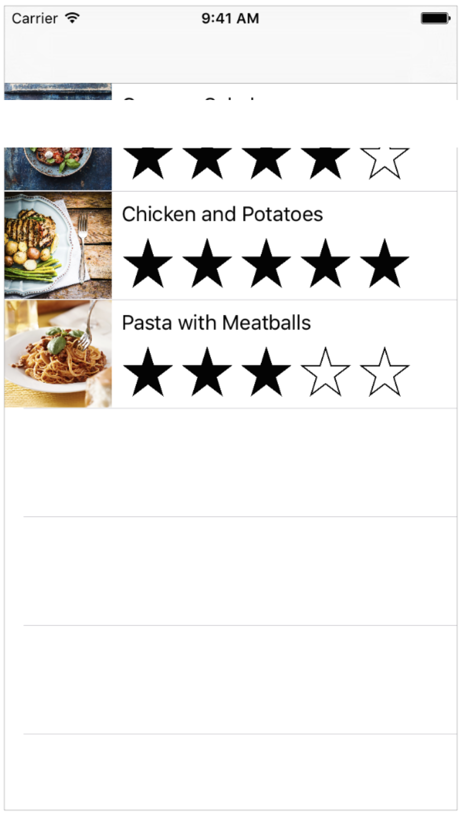
   Xcode adds a new navigation controller to your storyboard, sets the storyboard entry point to it, and creates a relationship between the new navigation controller and your existing table view controller.



On the canvas, the icon connecting the controllers is the root view controller relationship. The table view controller is the navigation controller's root view controller. The storyboard entry point is set to the navigation controller because the navigation controller is now a container for the table view controller.

You might notice that your table view has a bar on top of it now. This is a navigation bar. Every controller on the navigation stack gets a navigation bar, which can contain controls for backward and forward navigation. Next, you'll add a button to this navigation bar to transition to the meal scene.

Checkpoint: Run your app. Above your table view you should now see extra space. This is the navigation bar provided by the navigation controller. The navigation bar extends its background to the top of the status bar, so the status bar doesn't overlap with your content anymore.



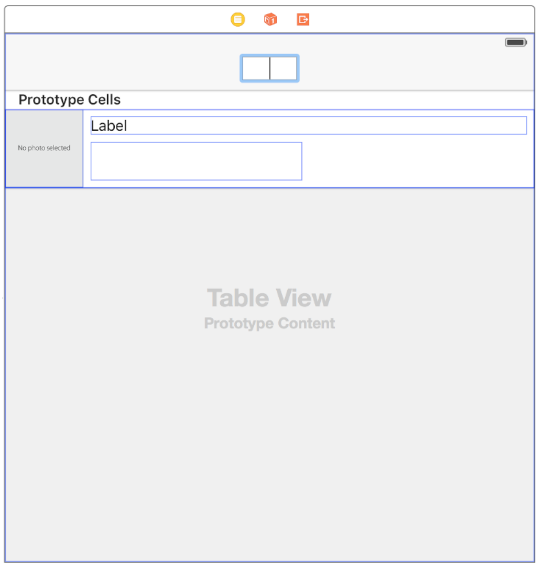
On This Page

Configure the Navigation Bar for the Scenes

Now, you'll add a title (to the meal list) and a button (to add additional meals) to the navigation bar. Navigation bars get their title from the view controller that the navigation controller currently displays—they don't themselves have a title. You set the title using the navigation item of your meal list (the table view controller) rather than setting it directly on the navigation bar.

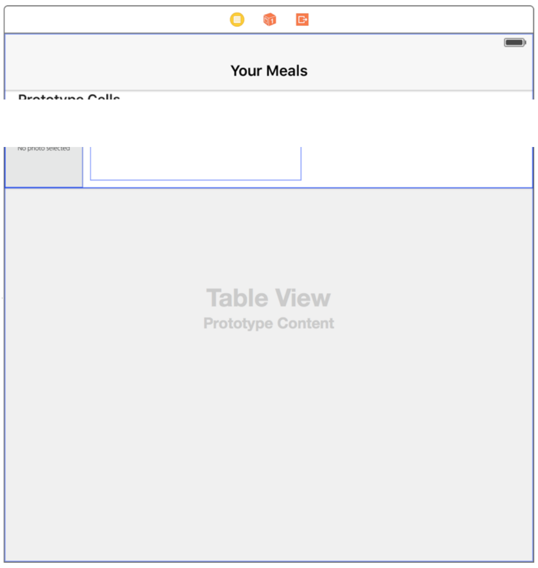
To configure the navigation bar in the meal list

1. Double-click the navigation bar in the meal list scene.



A cursor appears, letting you enter text.

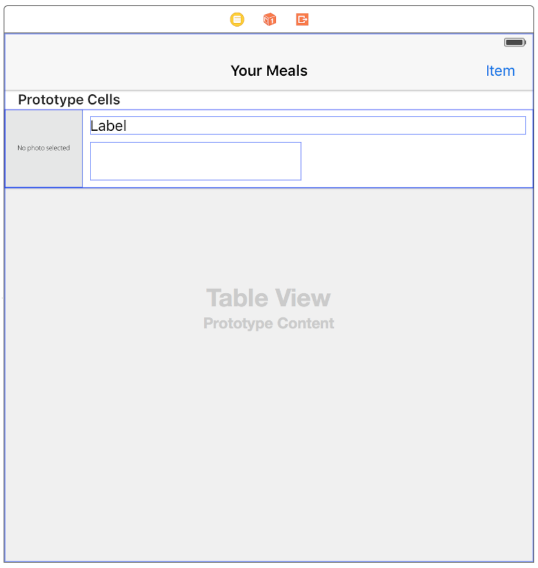
2. Type `Your Meals` and press Return to save.



On This Page

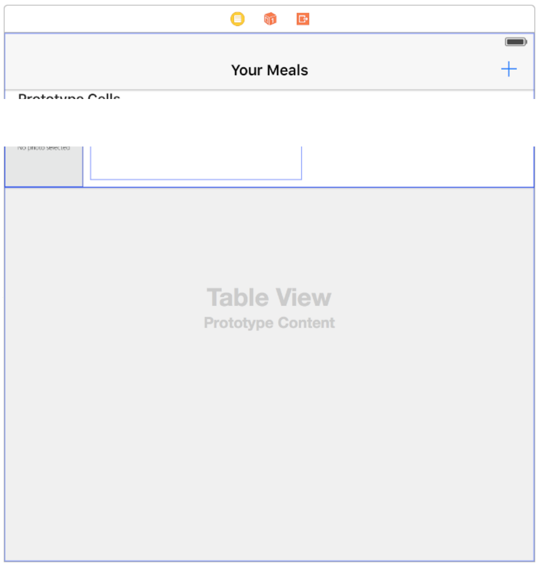
3. Open the Object library. (Choose View > Utilities > Show Object Library.)
4. In the Object library, find a Bar Button Item object.
5. Drag a Bar Button Item object from the list to the far right of the navigation bar in the meal list scene.

   A button called Item appears where you dragged the bar button item.



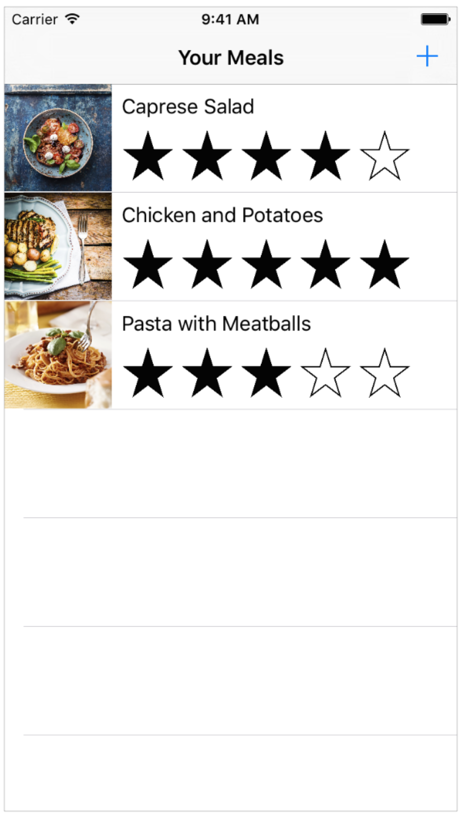
6. Select the bar button item and open the Attributes inspector.
7. In the Attributes inspector, choose Add from the pop-up menu next to the System Item option.

   The button changes to an Add button (+).



On This Page

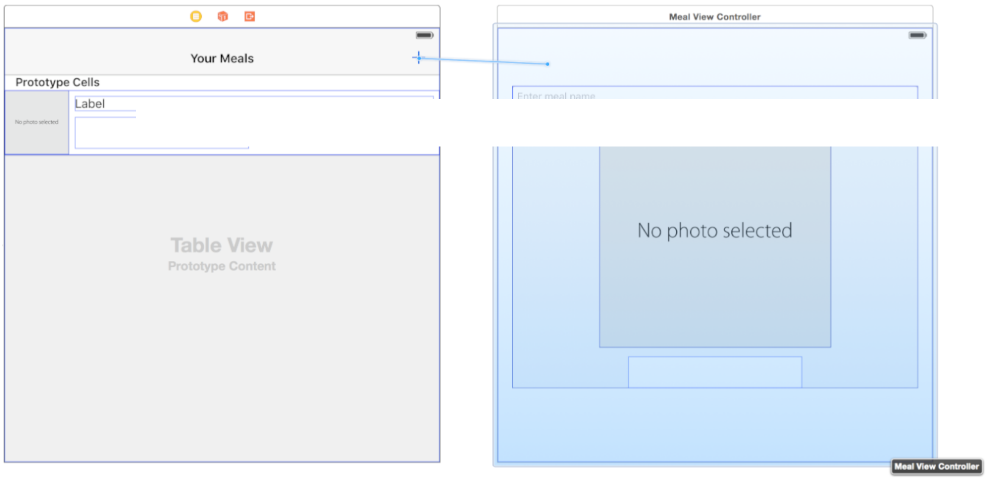
Checkpoint: Run your app. The navigation bar should now have a title and display an Add button (+). The button doesn't do anything yet. You'll fix that next.



You want the Add button (+) to bring up the meal scene, so you'll do this by having the button trigger a segue (or transition) to that scene.

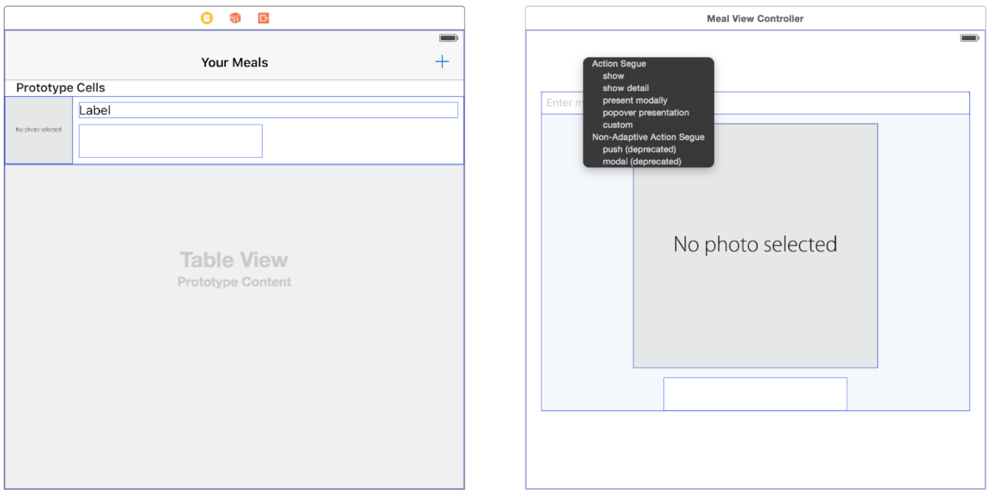
To configure the Add button in the meal scene

1. On the canvas, select the Add button (+).
2. Control-drag from the button to the meal scene.



On This Page

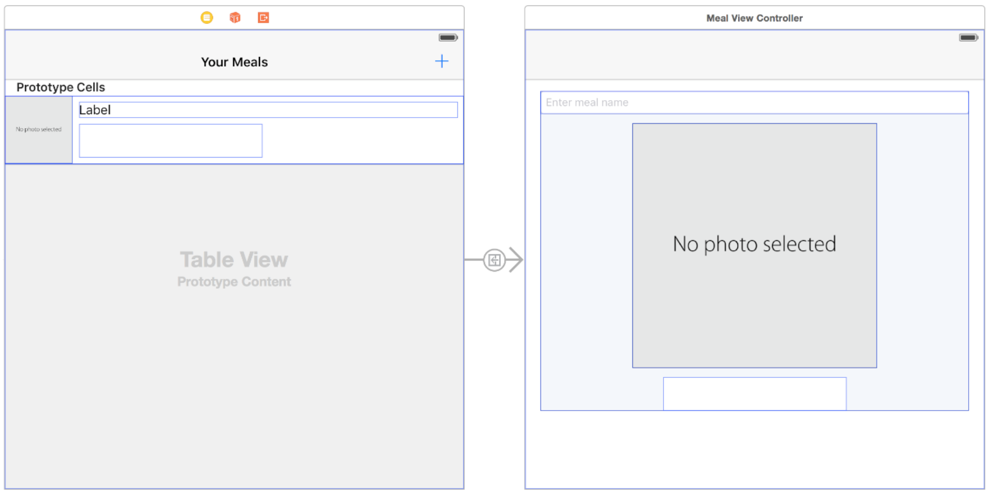
A shortcut menu titled Action Segue appears in the location where the drag ended.



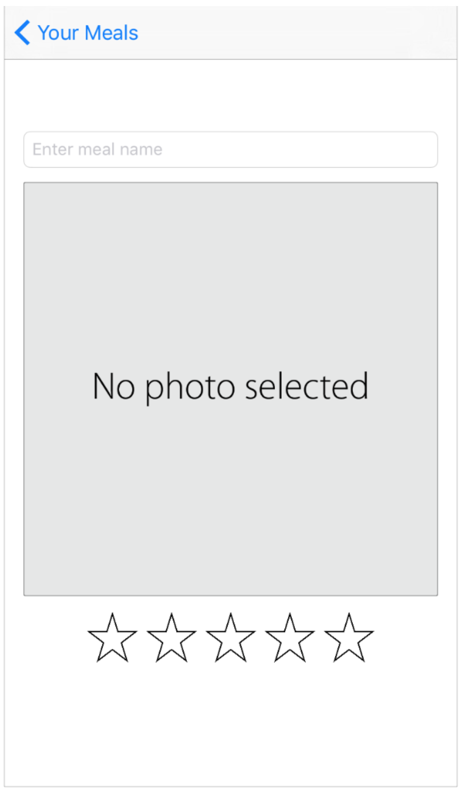
The Action Segue menu allows you to choose what type of segue to use to transition from the meal list to the new meal view controller when the user taps the Add button.

3. Choose show from the Action Segue menu.

Xcode sets up the show segue and configures the meal scene to be displayed in a navigation controller—you see the navigation bar in Interface Builder.



Checkpoint: Run your app. You can click the Add button and navigate to the meal scene from the meal list scene. Because you're using a navigation controller with a show segue, the backward navigation is handled for you, and a back button automatically appears in the meal scene. This means you can click the back button in the meal scene to get back to the meal list.

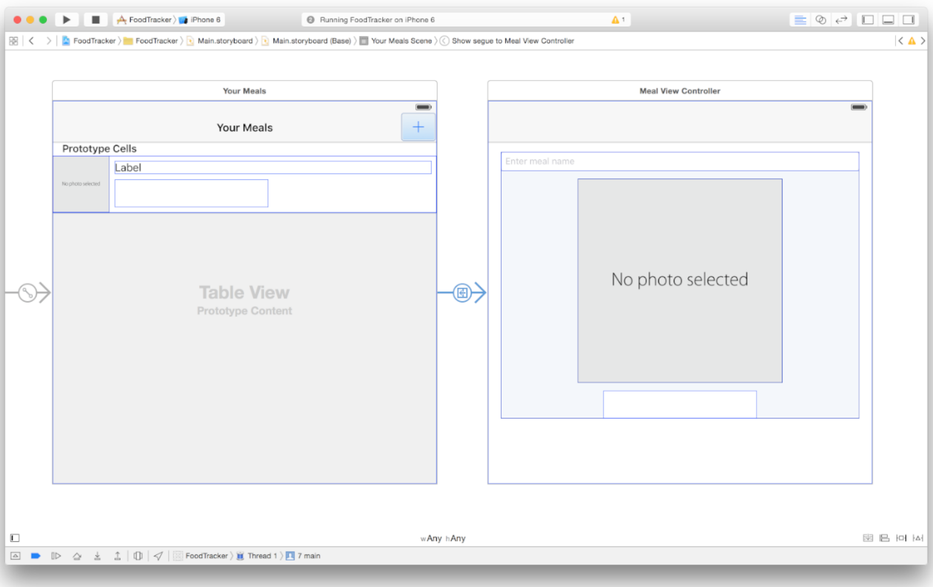


The push-style navigation you get by using the show segue is working just as it's supposed to—but it's not quite what you want when adding items. Push navigation is designed for a drill-down interface, where you're providing more information about whatever the user selected. Adding an item, on the other hand, is a modal operation—the user performs an action that's complete and self-contained, and then returns from that scene to the main navigation. The appropriate method of presentation for this type of scene is a modal segue.

Instead of deleting the existing segue and creating a new one, simply change the segue's style in the Attributes inspector. As is the case with most selectable elements in a storyboard, you can use the Attributes inspector to edit a segue's attributes.

To change the segue style

1. Select the segue from the meal list scene to the meal scene.



2. In the Attributes inspector, choose Present Modally from the pop-up menu next to the Segue option.
3. In the Attributes inspector, type `AddItem` in the Identifier field. Press Return.

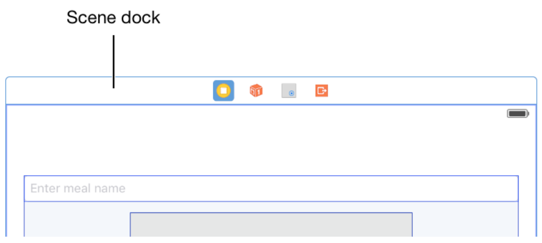
   Later, you'll need this identifier to identify the segue.

On This Page

A modal view controller doesn't get added to the navigation stack, so it doesn't get a navigation bar from the meal list's navigation controller. However, you want to keep the navigation bar to provide the user with visual continuity. To give the meal scene a navigation bar when presented modally, embed it in its own navigation controller.

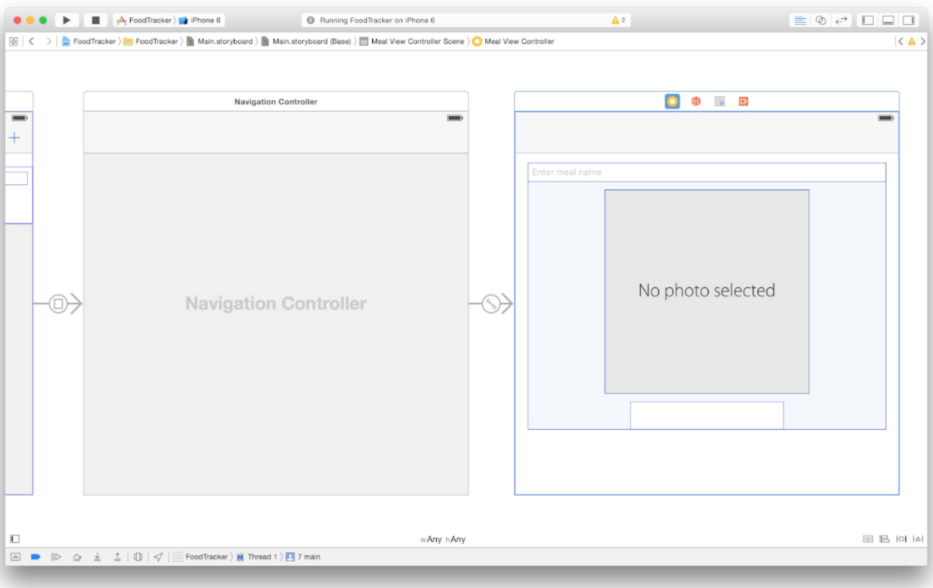
To add a navigation controller to the meal scene

1. Select the meal scene by clicking on its scene dock.



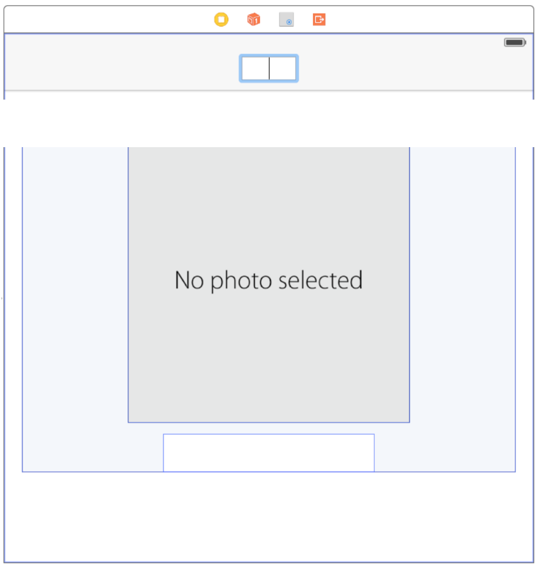
2. With the view controller selected, choose Editor > Embed In > Navigation Controller.

As before, Xcode adds a navigation controller and shows the navigation bar at the top of the meal scene. Next, configure this bar to add a title to this scene as well as two buttons, Cancel and Save. Later, you'll link these buttons to actions.



To configure the navigation bar in the meal scene

1. Double-click the navigation bar in the meal scene.

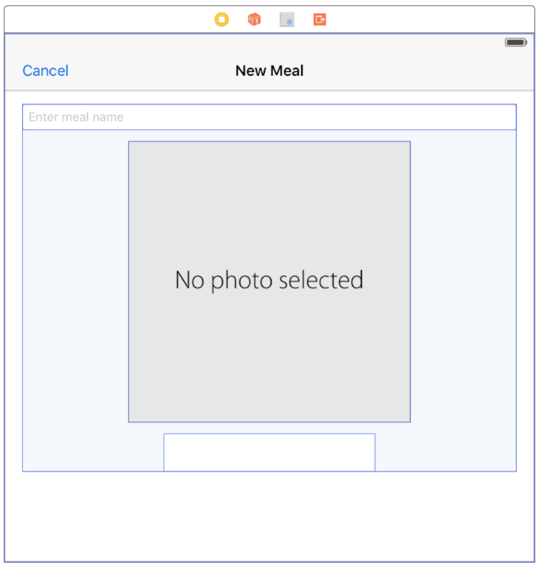


On This Page

A cursor appears, letting you enter text.

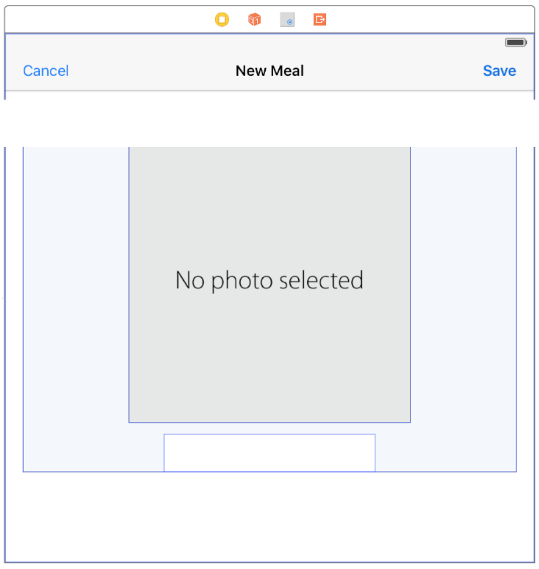
2. Type `New Meal` and press Return to save.
3. Drag a Bar Button Item object from the Object library to the far left of the navigation bar in the meal scene.
4. In the Attributes inspector, for System Item, select Cancel.

   The button text changes to Cancel.



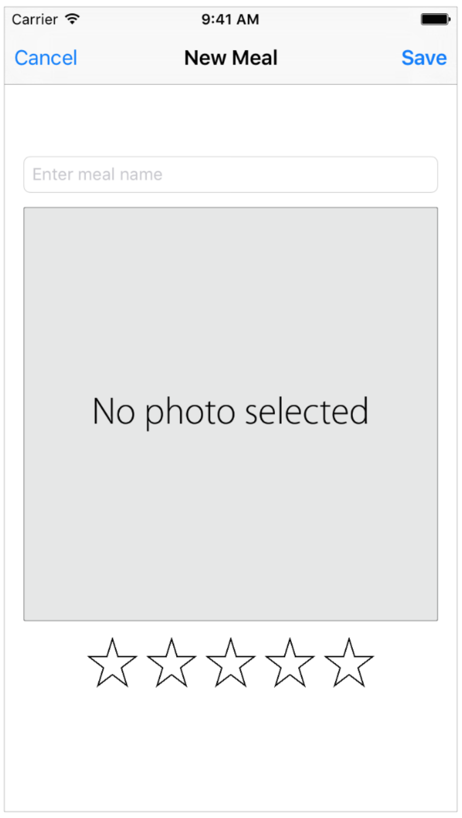
5. Drag another Bar Button Item object from the Object library to the far right of the navigation bar in the meal scene.
6. In the Attributes inspector, for System Item, select Save.

   The button text changes to Save.



On This Page

Checkpoint: Run your app. Click the Add button. You still see the meal scene, but there's no longer a button to navigate back to the meal list—instead, you see the two buttons you added, Cancel and Save. Those buttons aren't linked to any actions yet, so you can click them, but they don't do anything. You'll configure the buttons to save or cancel adding a new meal and to bring the user back to the meal list soon.



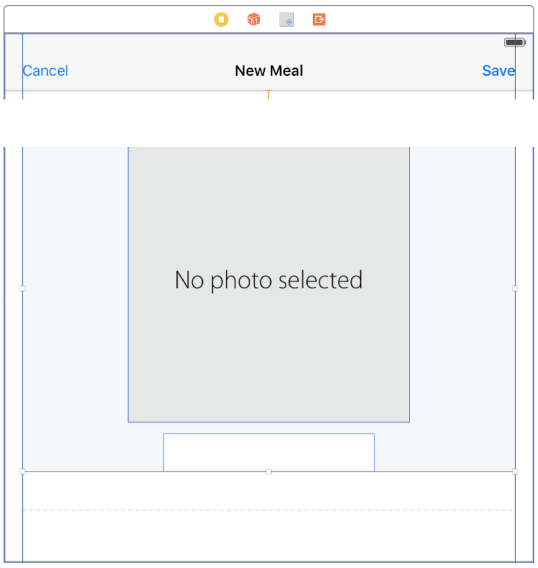
Finalize the UI with Auto Layout

It's been a while since you built the original UI, and a lot of things have changed since then. At this point, you won't be making any more changes to your layout, so it's time to make sure everything looks great with Auto Layout.

To do this, you'll make a few simple adjustments to your stack view.

To update the layout of the stack view

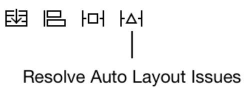
1. In the meal scene, click the light blue area to select the stack view.



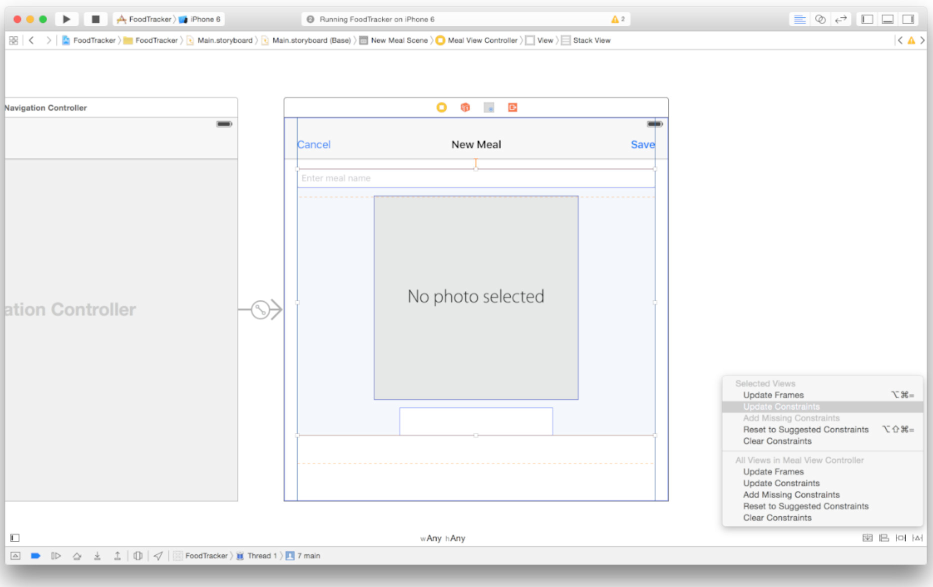
On This Page

Alternatively, select Stack View in the outline view.

2. On the bottom right of the canvas, open the Resolve Auto Layout Issues menu.

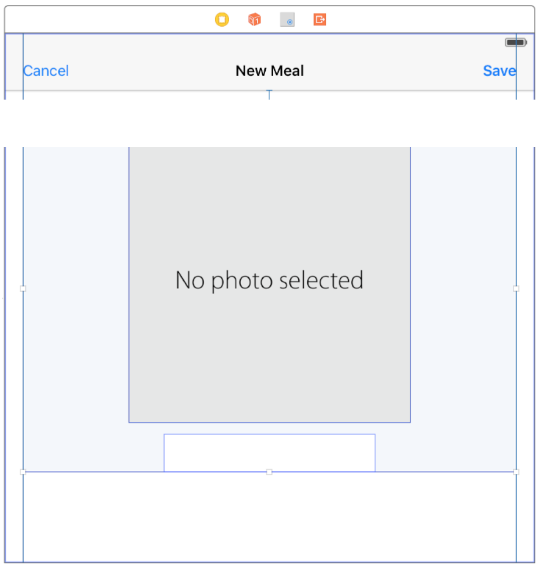


3. Under Selected Views, choose Update Constraints.

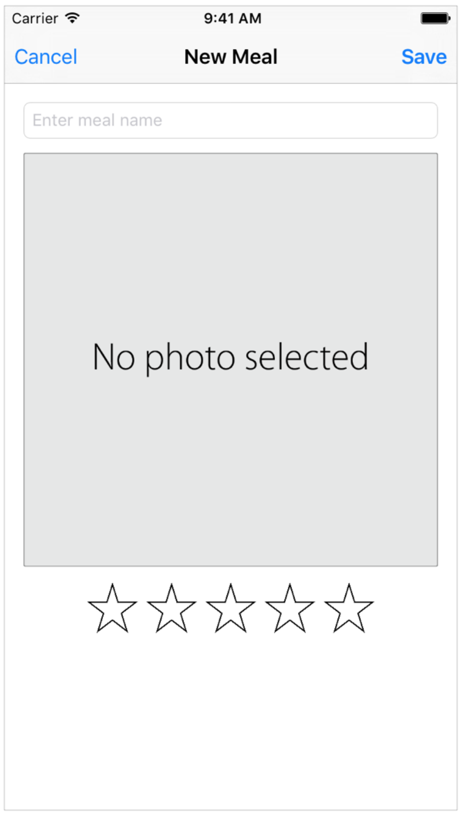


The elements stay in the same position, but the stack view is now pinned to the navigation bar instead of the top margin of the view.

The meal scene constraints and UI should look something like this:



Checkpoint: Run your app. The text field, image view, and rating control should appear closer to the navigation bar.



Store New Meals in the Meal List

The next step in creating the FoodTracker app's functionality is implementing the ability for a user to add a new meal. Specifically, when a user enters a meal name, rating, and photo in the meal scene and taps the Save button, you want `MealViewController` to configure a `Meal` object with the appropriate information and pass it back to `MealTableViewController` to display in the meal list.

Start by adding a `Meal` property to `MealViewController`.

To add a Meal property to MealViewController

1. Open `MealViewController.swift`.
2. Below the `ratingControl` outlet in `MealViewController.swift`, add the following property:

```
/*
 This value is either passed by `MealTableViewController` in
  `prepareForSegue(_:sender:)`

 */
var meal: Meal?
```

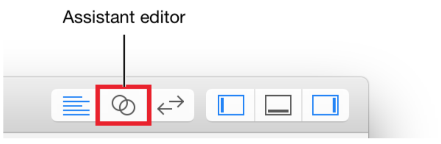
On This Page

This declares a property on `MealViewController` that is an optional `Meal`, which means that at any point, it may be `nil`.

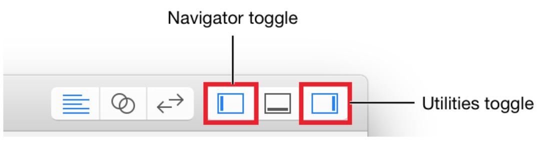
You care about configuring and passing the `Meal` only if the Save button was tapped. To be able to determine when this happens, add the Save button as an outlet in `MealViewController.swift`.

To connect the Save button to the MealViewController code

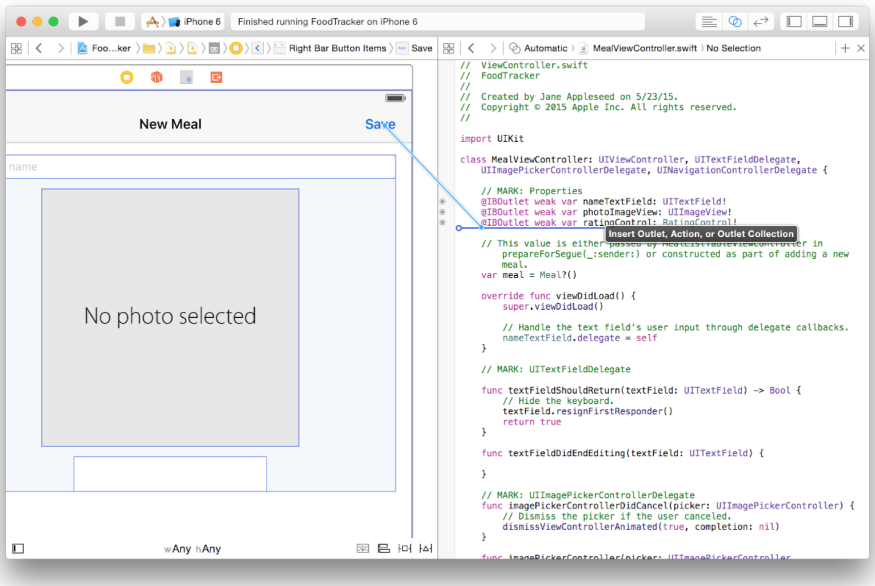
1. Open your storyboard.
2. Click the Assistant button in the Xcode toolbar to open the assistant editor.



3. If you want more space to work, collapse the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.



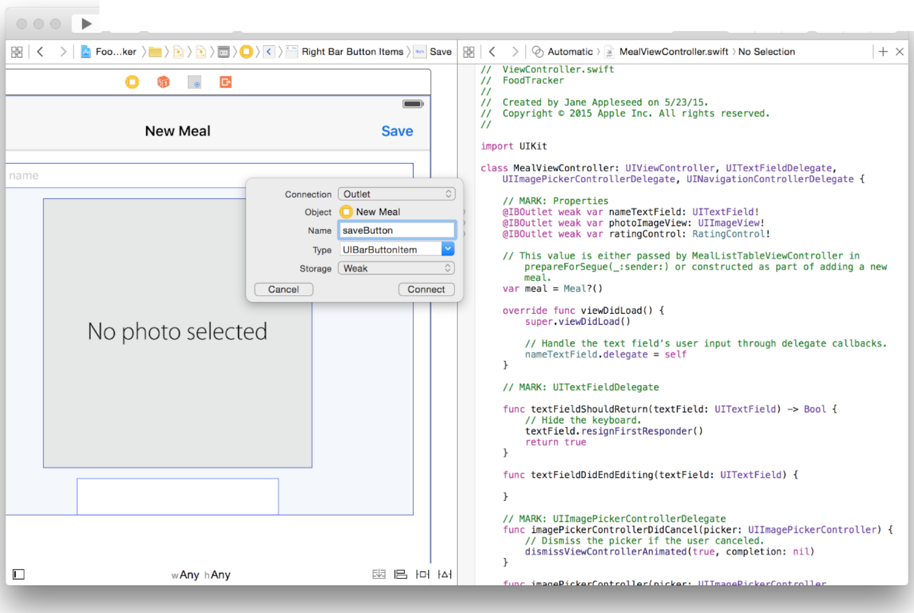
4. In your storyboard, select the Save button.
5. Control-drag from the Save button on your canvas to the code display in the editor on the right, stopping the drag at the line just below your `ratingControl` property in `MealViewController.swift`.



```
//  ViewController.swift
//  FoodTracker
//
//  Created by Jane Appleseed on 5/23/15.
//  Copyright © 2015 Apple Inc. All rights reserved.
//

import UIKit

class MealViewController: UIViewController, UITextFieldDelegate,
    UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    // MARK: Properties
    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var photoImageView: UIImageView!
    @IBOutlet weak var ratingControl: RatingControl!

    // This value is either passed by MealListTableViewController in
        prepareForSegue(_:sender:) or constructed as part of adding a new
        meal.
    var meal = Meal?()

    override func viewDidLoad() {
        super.viewDidLoad()

        // Handle the text field's user input through delegate callbacks.
        nameTextField.delegate = self
    }

    // MARK: UITextFieldDelegate

    func textFieldShouldReturn(textField: UITextField) -> Bool {
        // Hide the keyboard.
        textField.resignFirstResponder()
        return true
    }

    func textFieldDidEndEditing(textField: UITextField) {

    }

    // MARK: UIImagePickerControllerDelegate
    func imagePickerControllerDidCancel(picker: UIImagePickerController) {
        // Dismiss the picker if the user canceled.
        dismissViewControllerAnimated(true, completion: nil)
    }

    func imagePickerController(picker: UIImagePickerController
```

6. In the dialog that appears, for Name, type `saveButton`.

Leave the rest of the options as they are. Your dialog should look like this:



Connection: Outlet
Object: New Meal
Name: saveButton
Type: UIBarButtonItem
Storage: Weak
Cancel | Connect

```
//  ViewController.swift
//  FoodTracker
//
//  Created by Jane Appleseed on 5/23/15.
//  Copyright © 2015 Apple Inc. All rights reserved.
//

import UIKit

class MealViewController: UIViewController, UITextFieldDelegate,
    UIImagePickerControllerDelegate, UINavigationControllerDelegate {

    // MARK: Properties
    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var photoImageView: UIImageView!
    @IBOutlet weak var ratingControl: RatingControl!

    // This value is either passed by MealListTableViewController in
        prepareForSegue(_:sender:) or constructed as part of adding a new
        meal.
    var meal = Meal?()

    override func viewDidLoad() {
        super.viewDidLoad()

        // Handle the text field's user input through delegate callbacks.
        nameTextField.delegate = self
    }

    // MARK: UITextFieldDelegate

    func textFieldShouldReturn(textField: UITextField) -> Bool {
        // Hide the keyboard.
        textField.resignFirstResponder()
        return true
    }

    func textFieldDidEndEditing(textField: UITextField) {

    }

    // MARK: UIImagePickerControllerDelegate
    func imagePickerControllerDidCancel(picker: UIImagePickerController) {
        // Dismiss the picker if the user canceled.
        dismissViewControllerAnimated(true, completion: nil)
    }

    func imagePickerController(picker: UIImagePickerController
```

7. Click Connect.

You now have a way to identify the Save button.

Create an Unwind Segue

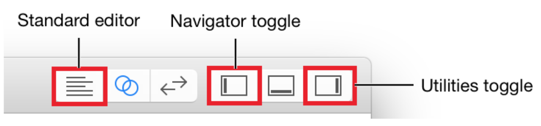
The task now is to pass the `Meal` object to `MealTableViewController` when a user taps the Save button and discard it when a user taps the Cancel button, switching from displaying the meal scene to displaying the meal list in either case.

To accomplish this, you'll use an unwind segue. An unwind segue moves backward through one or more segues to return the user to an existing instance of a view controller. You use unwind segues to implement reverse navigation.

Whenever a segue gets triggered, it provides a place for you to add your own code that gets executed. This method is called `prepareForSegue(_:sender:)`, and it gives you a chance to store data and do any necessary cleanup on the source view controller (the view controller that the segue is coming from). You'll implement this method in `MealViewController` to do exactly that.

To implement the prepareForSegue(_:sender:) method on MealViewController

1. Return to the standard editor by clicking the Standard button.



2. Open `MealViewController.swift`.
3. In `MealViewController.swift`, above the `// MARK: Actions` section, add the following:

```
// MARK: Navigation
```

   This is a comment to help you (and anybody else who reads your code) know that this method is related to the navigation flow of your app.

4. Below the comment, add this method skeleton:

```
// This method lets you configure a view controller before it's presented.
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
}
```

5. In the `prepareForSegue(_:sender:)` method, add the following `if` statement:

   ```
   if saveButton === sender {
   }
   ```

   This code uses the identity operator (`===`) to check that the object referenced by the `saveButton` outlet is the same object instance as `sender`. If it is, the `if` statement is executed.

6. In the `if` statement, add the following code:

   ```
   let name = nameTextField.text ?? ""
   let photo = photoImageView.image
   let rating = ratingControl.rating
   ```

   This code creates constants from the current text field text, selected image, and rating in the scene.

   Notice the nil coalescing operator (`??`) in the `name` line. The nil coalescing operator is used to return the value of an optional if the optional has a value, or return a default value otherwise. Here, the operator unwraps the optional `String` returned by `nameTextField.text` (which is optional because there may or may not be text in the text field), and returns that value if it's a valid string. But if it's `nil`, the operator the returns the empty string (`""`) instead.

7. In the `if` statement, add the following code:

   ```
   // Set the meal to be passed to MealTableViewController after the unwind segue.
   meal = Meal(name: name, photo: photo, rating: rating)
   ```

   This code configures the `meal` property with the appropriate values before segue executes.

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

```
// This method lets you configure a view controller before it's presented.
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if saveButton === sender {
        let name = nameTextField.text ?? ""
        let photo = photoImageView.image
        let rating = ratingControl.rating

        // Set the meal to be passed to MealTableViewController after the unwind
  segue.
        meal = Meal(name: name, photo: photo, rating: rating)
    }
}
```

The next step in creating the unwind segue is to add an action method to the destination view controller (the view controller that the segue is going to). This method must be marked with the `IBAction` attribute and take a segue (`UIStoryboardSegue`) as a parameter. Because you want to unwind back to the meal list scene, you need to add an action method with this format to `MealTableViewController.swift`.

In this method, you'll write the logic to add the new meal (that's passed from `MealViewController`, the source view controller) to the meal list data and add a new row to the table view in the meal list scene.

To add an action method to MealTableViewController

1. Open `MealTableViewController.swift`.
2. In `MealTableViewController.swift`, before the last curly brace (`}`), add the following:

   ```
   @IBAction func unwindToMealList(sender: UIStoryboardSegue) {
   }
   ```

3. In the `unwindToMealList(_:)` action method, add the following `if` statement:

```
if let sourceViewController = sender.sourceViewController as?
  MealViewController, meal = sourceViewController.meal {
}
```

On This Page

There's quite a lot happening in the condition for this `if` statement.

This code uses the optional type cast operator (`as?`) to try to downcast the source view controller of the segue to type `MealViewController`. You need to downcast because `sender.sourceViewController` is of type `UIViewController`, but you need to work with `MealViewController`.

The operator returns an optional value, which will be `nil` if the downcast wasn't possible. If the downcast succeeds, the code assigns that view controller to the local constant `sourceViewController`, and checks to see if the meal property on `sourceViewController` is `nil`. If the `meal` property is non-`nil`, the code assigns the value of that property to the local constant `meal` and executes the `if` statement.

If either the downcast fails or the `meal` property on `sourceViewController` is `nil`, the condition evaluates to `false` and the `if` statement doesn't get executed.

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

```
// Add a new meal.
let newIndexPath = NSIndexPath(forRow: meals.count, inSection: 0)
```

This code computes the location in the table view where the new table view cell representing the new meal will be inserted, and stores it in a local constant called `newIndexPath`.

5. In the `if` statement, below the previous line of code, add the following code:

```
meals.append(meal)
```

This adds the new meal to the existing list of meals in the data model.

6. In the `if` statement, below the previous line of code, add the following code:

```
tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Bottom)
```

This animates the addition of a new row to the table view for the cell that contains information about the new meal. The `.Bottom` animation option shows the inserted row slide in from the bottom.

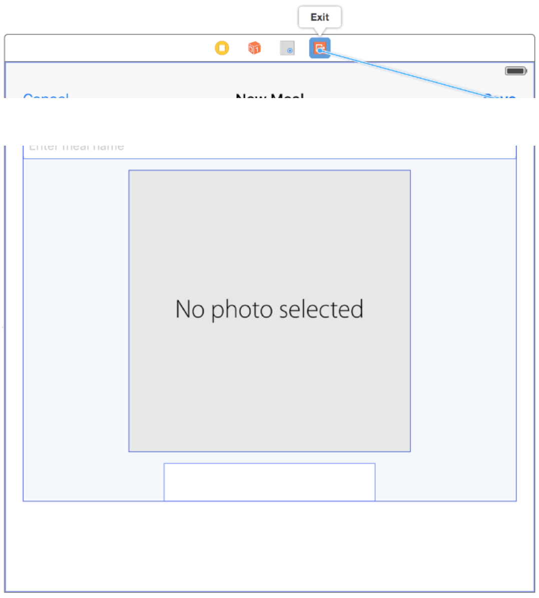
You'll finish a more advanced implementation of this method in a little while, but for now, the `unwindToMealList(_:)` action method should look like this:

```
@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)
    }
}
```

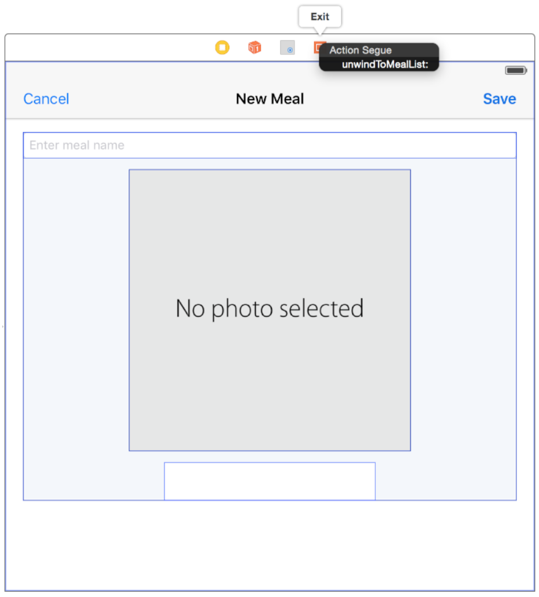
Now you need to create the actual unwind segue to trigger this action method.

To link the Save button to the unwindToMealList action method

1. Open your storyboard.
2. On the canvas, Control-drag from the Save button to the Exit item at the top of the meal scene.



A menu appears in the location where the drag ended.



3. Choose `unwindToMealList:` from the shortcut menu.

   Now, when users tap the Save button, they navigate back to the meal list scene, during which process the `unwindToMealList(_:)` action method is called.

Checkpoint: Run your app. Now when you click the Add button (+), create a new meal, and click Save, you should see the new meal in your meal list.

> IMPORTANT
>
> If you don't see the `unwindToMealList` method in the shortcut menu, make sure that the method has the right signature.
>
> ```
> @IBAction func unwindToMealList(sender: UIStoryboardSegue)
> ```

Disable Saving When the User Doesn't Enter an Item Name

What happens if a user tries to save a meal with no name? Because the `meal` property on `MealViewController` is an optional and you set your initializer up to fail if there's no name, the `Meal` object doesn't get created and added to the meal list—which is what you expect to happen. But you can take this a step further and keep users from accidentally trying to add meals without a name by disabling the Save button while they're typing a meal name, and checking that they've specified a valid name before letting them dismiss the keyboard.

To disable the S

On This Page

1. In `MealViewController.swift`, find the `// MARK: UITextFieldDelegate` section.

   You can jump to it quickly using the functions menu, which appears if you click the name of the file at the top of the editor area.

2. In this section, add another `UITextFieldDelegate` method:

```
func textFieldDidBeginEditing(textField: UITextField) {
    // Disable the Save button while editing.
    saveButton.enabled = false
}
```

   The `textFieldDidBeginEditing` method gets called when an editing session begins, or when the keyboard gets displayed. This code disables the Save button while the user is editing the text field.

3. Below the `textFieldDidBeginEditing(_:)` method, add another method:

```
func checkValidMealName() {
    // Disable the Save button if the text field is empty.
    let text = nameTextField.text ?? ""
    saveButton.enabled = !text.isEmpty
}
```

   This is a helper method to disable the Save button if the text field is empty.

4. Find the `textFieldDidEndEditing(_:)` method:

```
func textFieldDidEndEditing(textField: UITextField) {
}
```

   The implementation should be empty at this point.

5. Add these lines of code:

```
checkValidMealName()
navigationItem.title = textField.text
```

   The first line calls `checkValidMealName()` to check if the text field has text in it, which enables the Save button if it does. The second line sets the title of the scene to that text.

6. Find the `viewDidLoad()` method.

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

    // Handle the text field's user input through delegate callbacks.
    nameTextField.delegate = self
}
```

7. Add a call to `checkValidMealName()` at the implementation to make sure the Save button is disabled until a user enters a valid name:

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

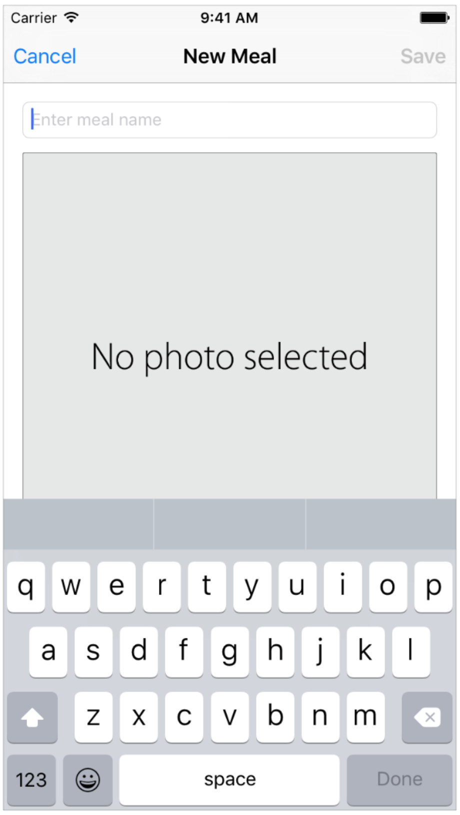
Your `viewDidLoad()` method should look like this:

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

    // Handle the text field's user input through delegate callbacks.
    nameTextField.delegate = self

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

And your `textFieldDidEndEditing(_:)` method should look like this:

```
func textFieldDidEndEditing(textField: UITextField) {
    checkValidMealName()
    navigationItem.title = textField.text
}
```

Checkpoint: Run your app. Now when you click the Add button (+), the Save button is disabled until you enter a valid (nonempty) meal name and dismiss the keyboard.

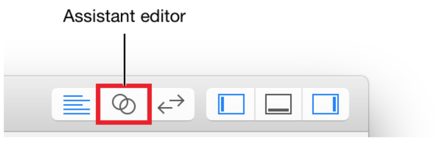


Cancel a New Meal Addition

A user might decide to cancel the addition of a new meal, and return to the meal list without saving anything. For this, you'll implement the behavior of the Cancel button.

To create and implement a cancel action method

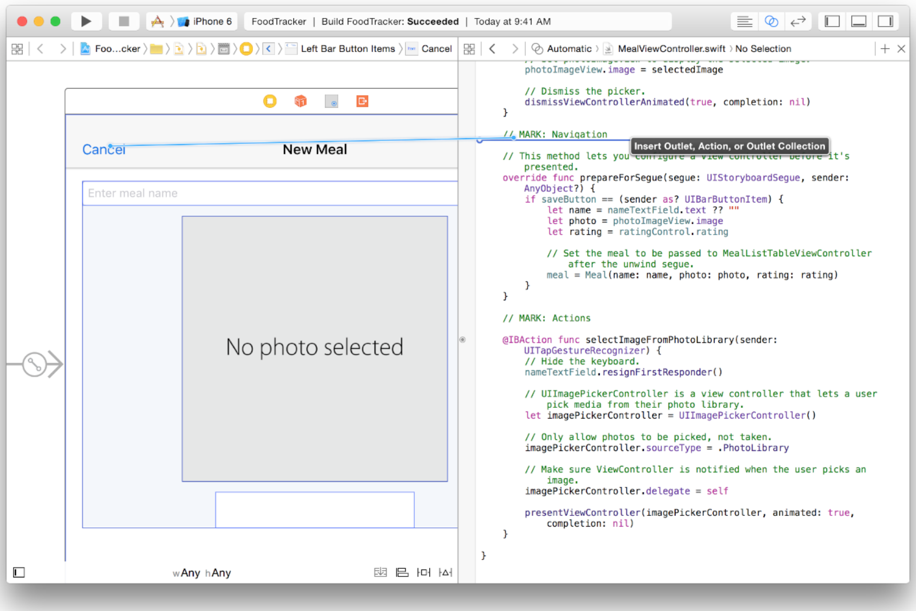
1. Open your storyboard.
2. Click the Assistant button in the Xcode toolbar to open the assistant editor.



3. In your storyboard, select the Cancel button.

4. Control-drag from the Cancel button on your canvas to the code display in the editor on the right, stopping the drag at the line just below the `// MARK: Navigation` comment in `MealViewController.swift`.

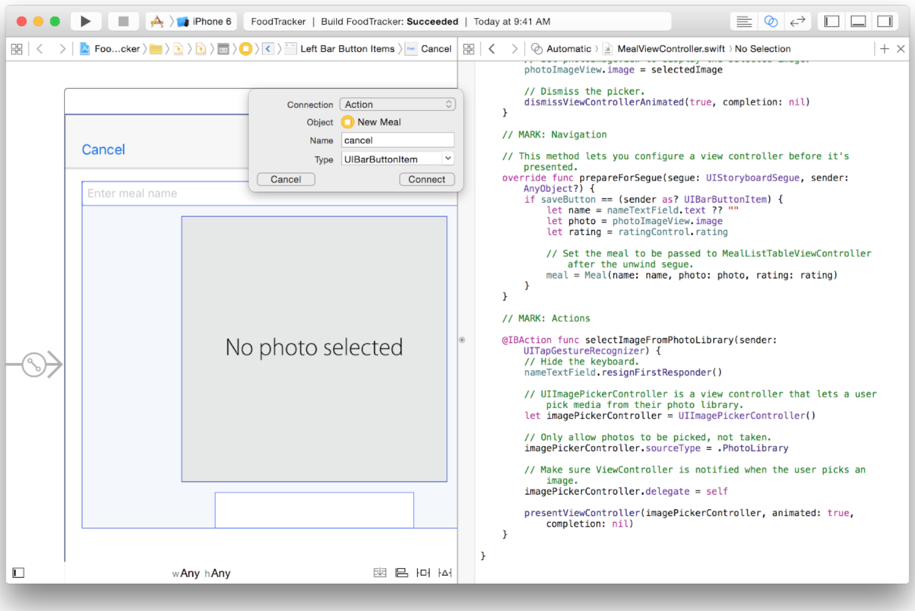
On This Page


iPhone 6
FoodTracker | Build FoodTracker: Succeeded | Today at 9:41 AM
Foo...cker 〉 Left Bar Button Items 〉 Cancel
Automatic 〉 MealViewController.swift 〉 No Selection
Cancel
New Meal
Enter meal name
No photo selected
Insert Outlet, Action, or Outlet Collection
wAny hAny

```
        photoImageView.image = selectedImage

        // Dismiss the picker.
        dismissViewControllerAnimated(true, completion: nil)
    }

    // MARK: Navigation

    // This method lets you configure a view controller before it's
        presented.
    override func prepareForSegue(segue: UIStoryboardSegue, sender:
        AnyObject?) {
        if saveButton == (sender as? UIBarButtonItem) {
            let name = nameTextField.text ?? ""
            let photo = photoImageView.image
            let rating = ratingControl.rating

            // Set the meal to be passed to MealListTableViewController
                after the unwind segue.
            meal = Meal(name: name, photo: photo, rating: rating)
        }
    }

    // MARK: Actions

    @IBAction func selectImageFromPhotoLibrary(sender:
        UITapGestureRecognizer) {
        // Hide the keyboard.
        nameTextField.resignFirstResponder()

        // UIImagePickerController is a view controller that lets a user
            pick media from their photo library.
        let imagePickerController = UIImagePickerController()

        // Only allow photos to be picked, not taken.
        imagePickerController.sourceType = .PhotoLibrary

        // Make sure ViewController is notified when the user picks an
            image.
        imagePickerController.delegate = self

        presentViewController(imagePickerController, animated: true,
            completion: nil)
    }

}
```

5. In the dialog that appears, for Connection, select Action.
6. For Name, type `cancel`.
7. For Type, select `UIBarButtonItem`.

   Leave the rest of the options as they are. Your dialog should look like this:


iPhone 6
FoodTracker | Build FoodTracker: Succeeded | Today at 9:41 AM
Foo...cker 〉 Left Bar Button Items 〉 Cancel
Automatic 〉 MealViewController.swift 〉 No Selection
Cancel
Enter meal name
No photo selected
Cancel
Connect
wAny hAny

Connection: Action
Object: New Meal
Name: cancel
Type: UIBarButtonItem

```
        photoImageView.image = selectedImage

        // Dismiss the picker.
        dismissViewControllerAnimated(true, completion: nil)
    }

    // MARK: Navigation

    // This method lets you configure a view controller before it's
        presented.
    override func prepareForSegue(segue: UIStoryboardSegue, sender:
        AnyObject?) {
        if saveButton == (sender as? UIBarButtonItem) {
            let name = nameTextField.text ?? ""
            let photo = photoImageView.image
            let rating = ratingControl.rating

            // Set the meal to be passed to MealListTableViewController
                after the unwind segue.
            meal = Meal(name: name, photo: photo, rating: rating)
        }
    }

    // MARK: Actions

    @IBAction func selectImageFromPhotoLibrary(sender:
        UITapGestureRecognizer) {
        // Hide the keyboard.
        nameTextField.resignFirstResponder()

        // UIImagePickerController is a view controller that lets a user
            pick media from their photo library.
        let imagePickerController = UIImagePickerController()

        // Only allow photos to be picked, not taken.
        imagePickerController.sourceType = .PhotoLibrary

        // Make sure ViewController is notified when the user picks an
            image.
        imagePickerController.delegate = self

        presentViewController(imagePickerController, animated: true,
            completion: nil)
    }

}
```

8. Click Connect.

   Xcode adds the necessary code to `MealViewController.swift` to set up the action.

```
@IBAction func cancel(sender: UIBarButtonItem) {
}
```

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

```
dismissViewControllerAnimated(true, completion: nil)
```

This code
dismissec ,

On This Page

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

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

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.

NOTE

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

Download File