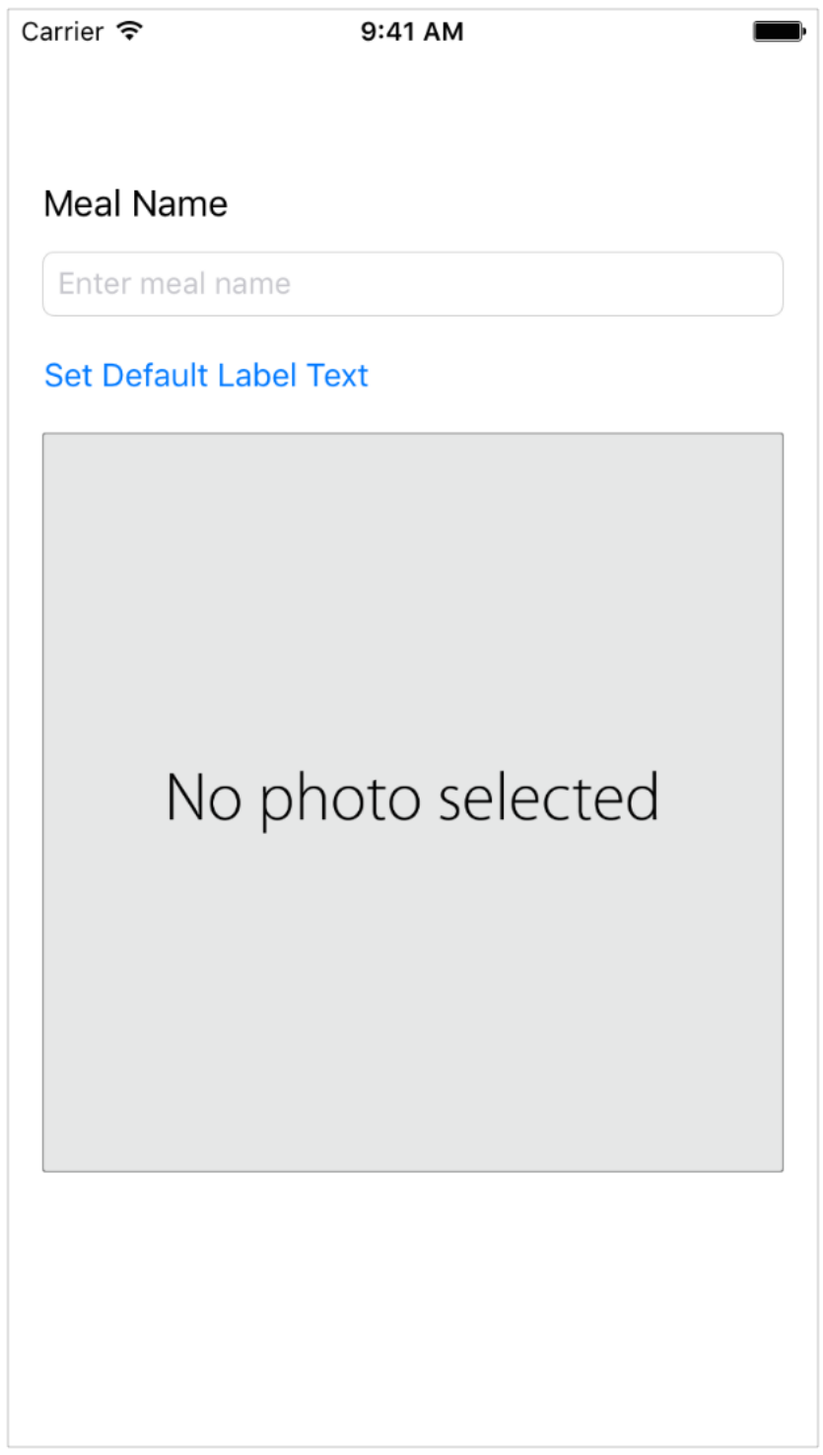
Work with View Controllers

In this lesson, you'll continue to work on the UI for the meal scene in the FoodTracker app. You'll rearrange the existing UI elements and work with an image picker to add a photo to the UI. When you're finished, your app will look something like this:



Learning Objectives

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

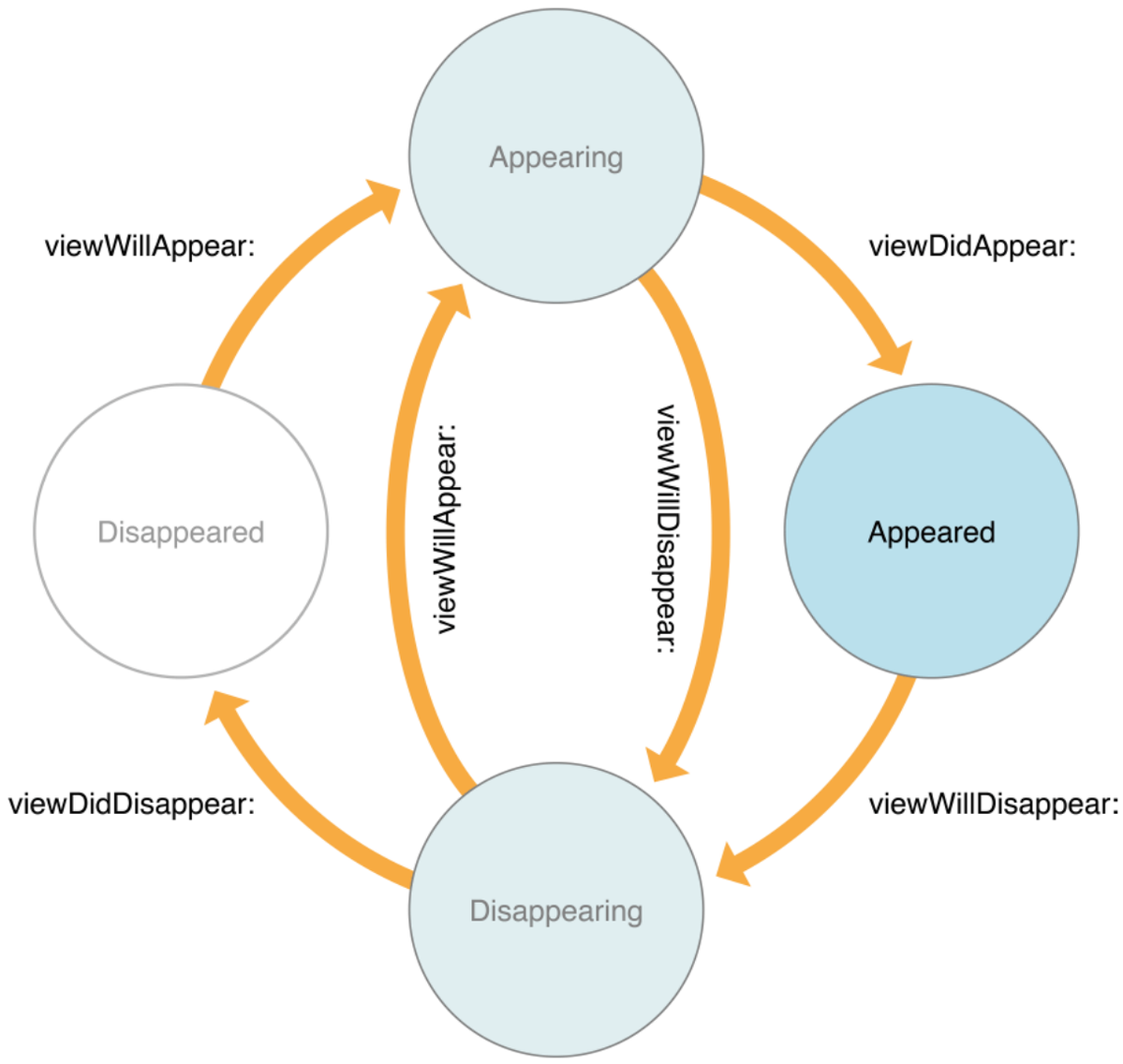
- Understand the view controller life cycle and when its callbacks occur, such as `viewDidLoad`, `viewWillAppear` and `viewDidAppear`
- Pass data between view controllers
- Dismiss a view controller
- Use gesture recognizers as an additional level of generating events
- Anticipate object behavior based on the `UIView`/`UIControl` class hierarchy
- Use the asset catalog to add image assets to a project

Understand the View Controller Lifecycle

So far, the FoodTracker app has a single scene, whose UI is managed by a single view controller. As you build more complex apps, you'll be dealing with more scenes, and will need to manage loading and unloading views as they're moved on and off the screen.

An object of the `UIViewController` class (and its subclasses) comes with a set of methods that manage its view hierarchy. iOS automatically calls these methods at appropriate times when a view controller transitions between states. When you create a view controller subclass (like the `ViewController` class you've been working with), it inherits the methods defined in `UIViewController` and lets you add your own custom behavior for each method. It's important to understand when these methods get called, so you can set up or

tear down the views you're displaying at the appropriate step in the process—something you'll need to do later in the lessons.



`UIViewController` methods get called as follows:

- `viewDidLoad()`—Called when the view controller's content view (the top of its view hierarchy) is created and loaded from a storyboard. This method is intended for initial setup. However, because views may be purged due to limited resources in an app, there is no guarantee that it will be called only once.
- `viewWillAppear()`—Intended for any operations that you want always to occur before the view becomes visible. Because a view's visibility may be toggled or obscured by other views, this method is always called immediately before the content view appears onscreen.
- `viewDidAppear()`—Intended for any operations that you want to occur as soon as the view becomes visible, such as fetching data or showing an animation. Because a view's visibility may be toggled or obscured by other views, this method is always called immediately after the content view appears onscreen.

A complementary set of teardown methods exists, as shown in the state transition diagram above.

You'll be using some of these methods in the FoodTracker app to load and display view data at the right time. In fact, if you recall, you've already written some code in the `viewDidLoad()` method of `ViewController`:

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

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

This style of app design where view controllers serve as the communication pipeline between your views and data model is known as MVC (Model-View-Controller). In this pattern, models keep track of your app's data, views display your user interface and make up the content of an app, and controllers manage your views. By responding to user actions and populating views with content from the data model, controllers serve as a gateway for communication between the model and views. MVC is central to a good design for any iOS app, and so far, the FoodTracker app has been built along MVC principles.

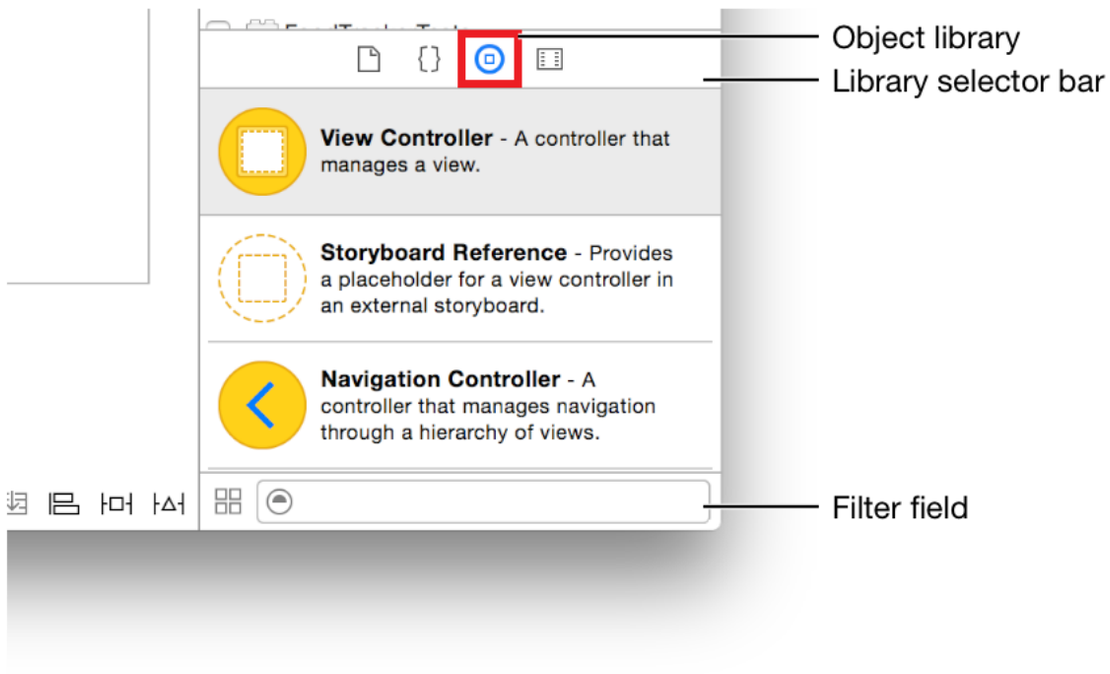
As you keep the MVC pattern in mind for rest of the app's design, it's time to take your basic UI to the next level, and create a final layout for the meal scene.

Add a Meal Photo

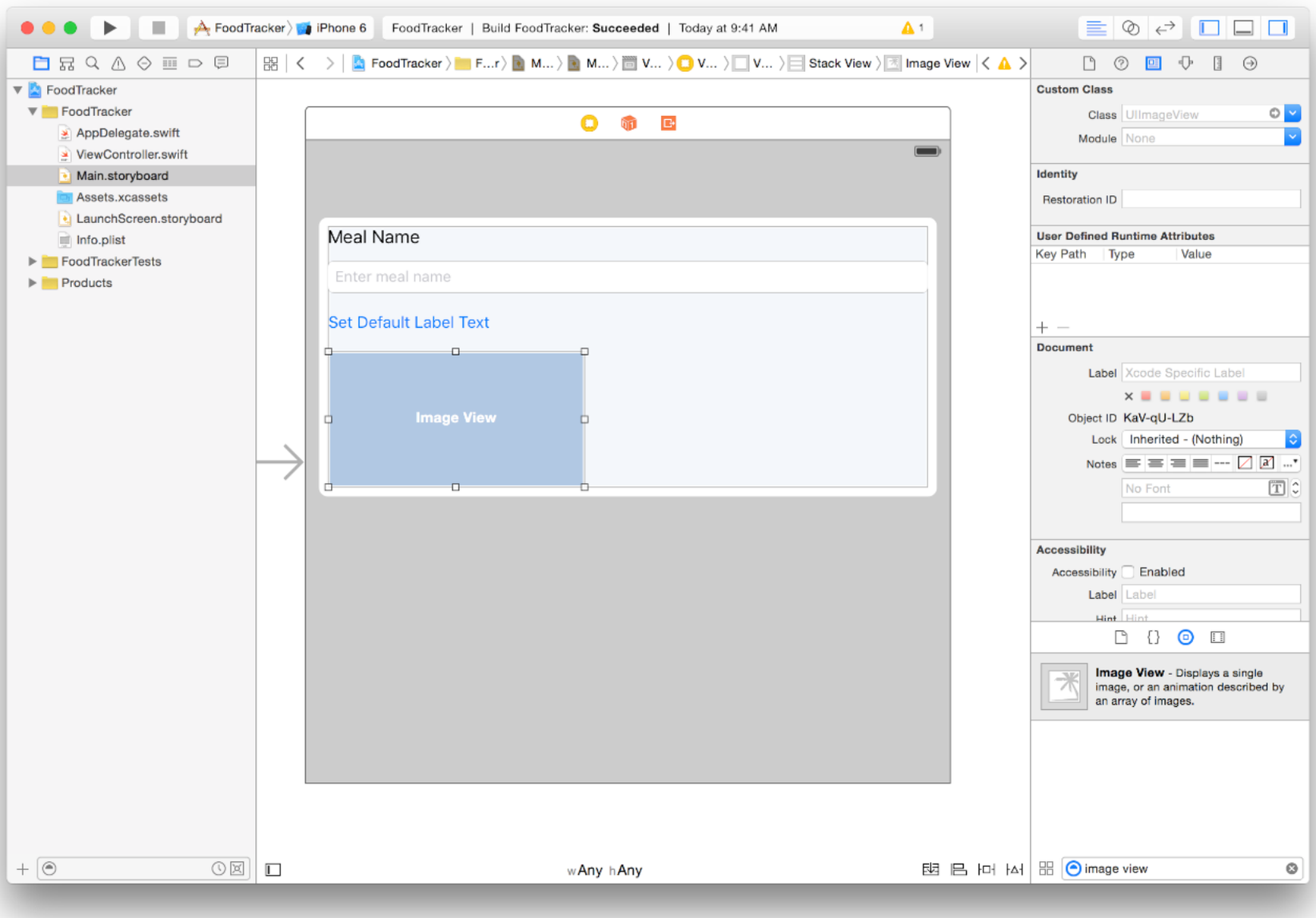
The next step in finishing the meal scene UI is adding a way to display a photo of a particular meal. For this, you'll use an image view (`UIImageView`), a UI element that displays a picture.

To add an image view to your scene

1. Open your storyboard, `Main.storyboard`.
2. Open the Object library in the utility area. (Alternatively, choose View > Utilities > Show Object Library.)

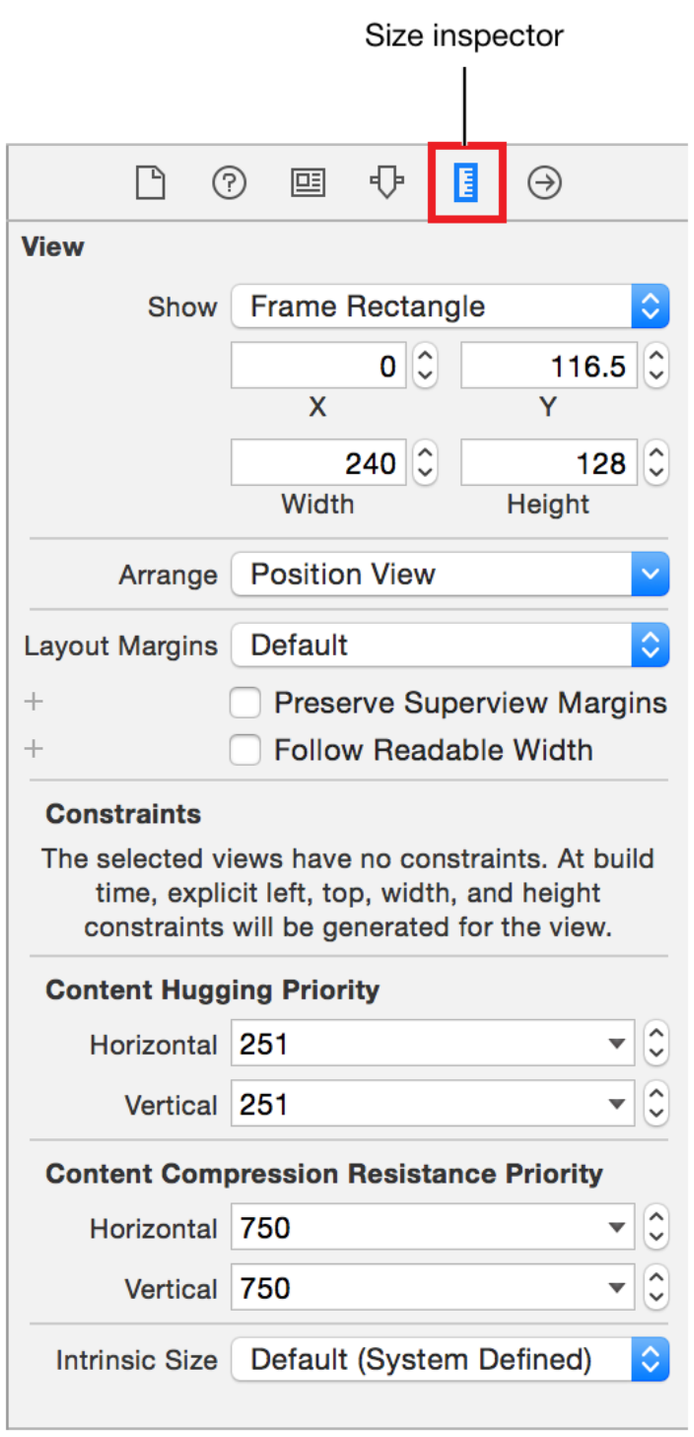


3. In the Object library, type `image view` in the filter field to find the Image View object quickly.
4. Drag an Image View object from the Object library to your scene so that it's in the stack view below the button.



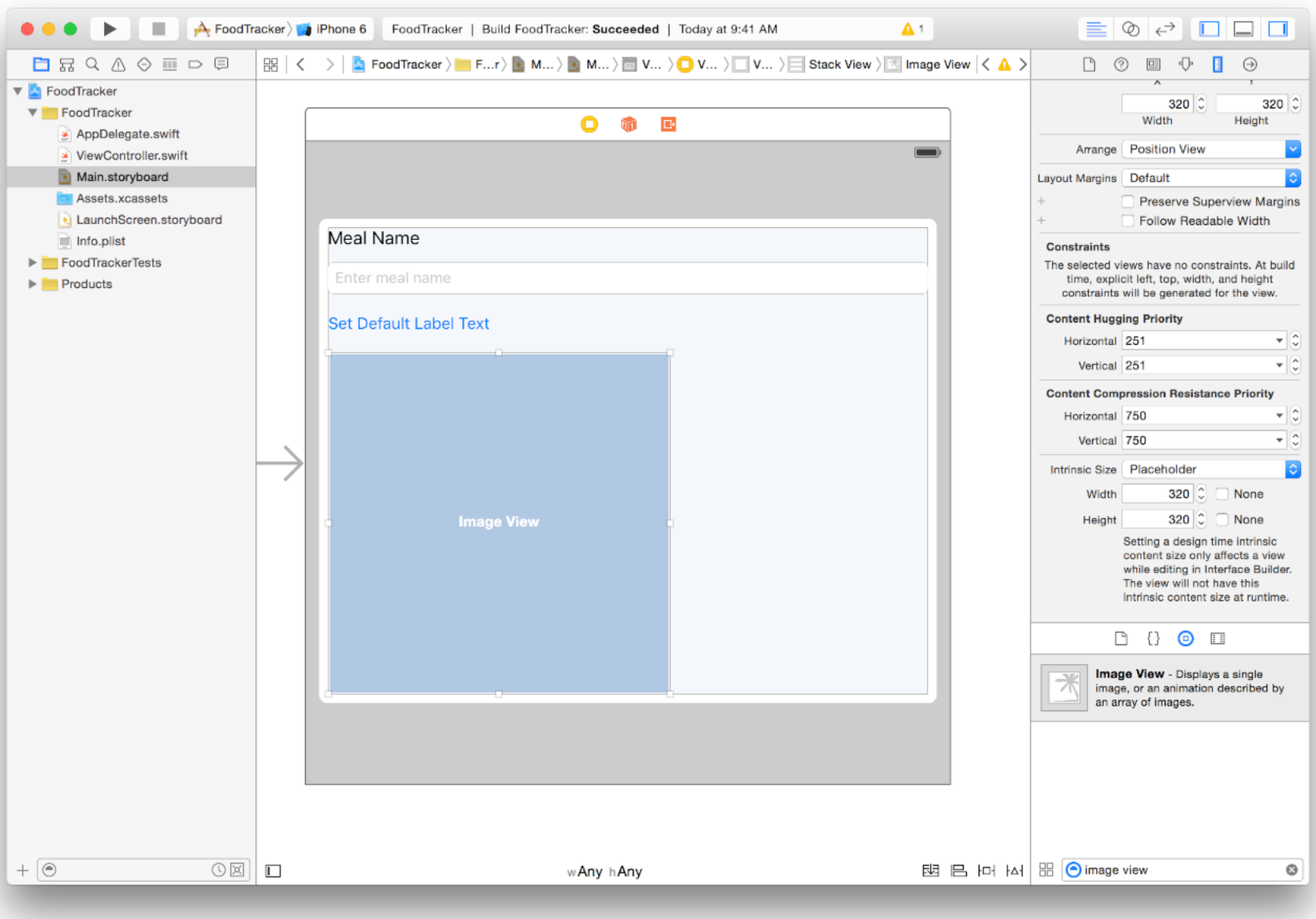
5. With the image view selected, open the Size inspector in the utility area.

   Recall that the Size inspector appears when you select the fifth button from the left in the inspector selector bar. It lets you edit the size and position of an object in your storyboard.



6. In the Intrinsic Size field, select Placeholder. (This field is at the bottom of the Size inspector, so you'll need to scroll down to it.)
7. Type `320` in both the Width and Height fields. Press Return.

   An empty image view doesn't have an intrinsic content size. You're giving your image view a placeholder size so you can specify the appropriate constraints in your interface.

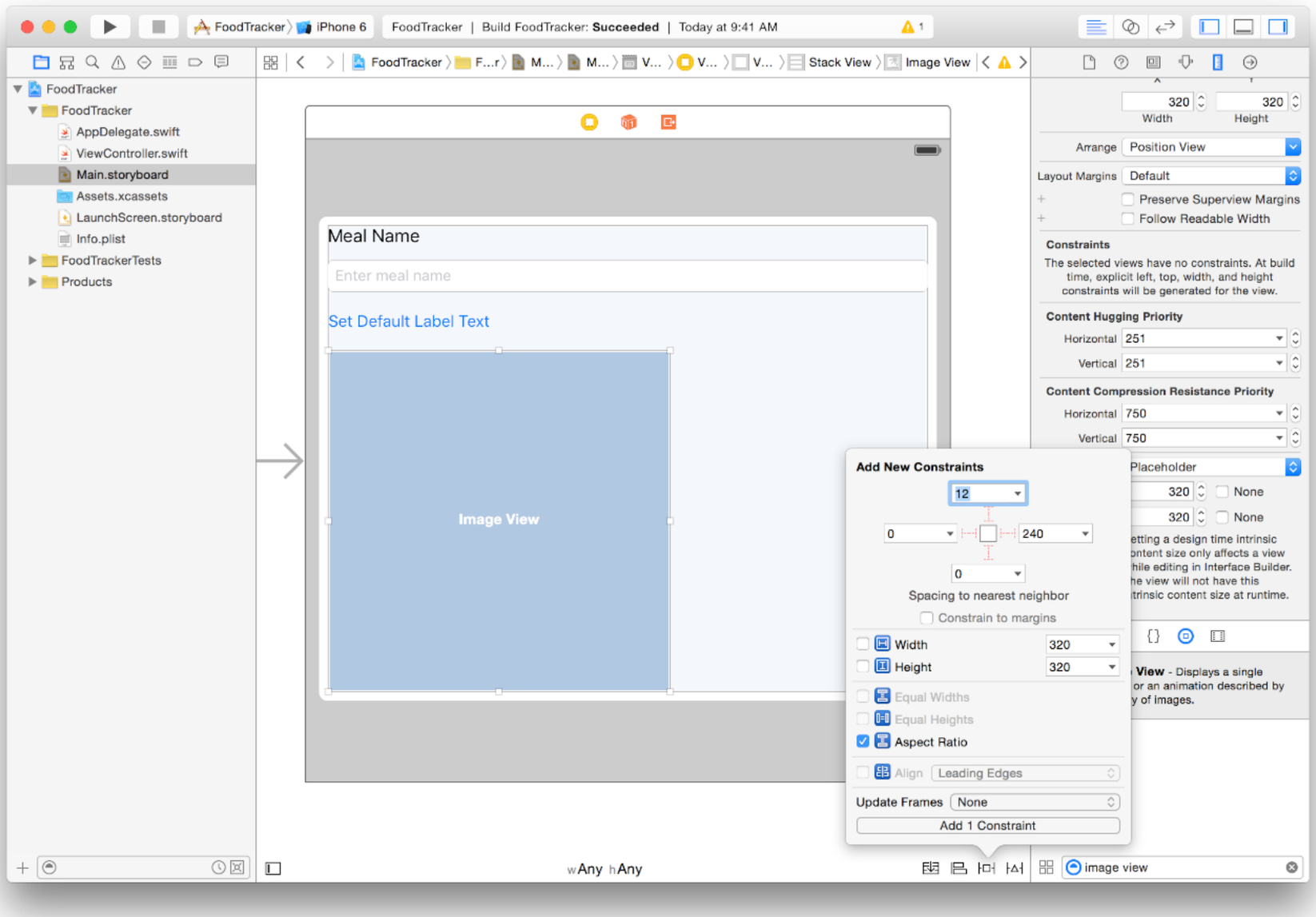


8. On the bottom right of the canvas, open the Pin menu.

Pin

9. Select the checkbox next to Aspect Ratio.

   The Pin menu should look something like this:



10. In the Pin menu, click the Add 1 Constraints button.

    Your image view now has a 1:1 aspect ratio, so it will always show a square.

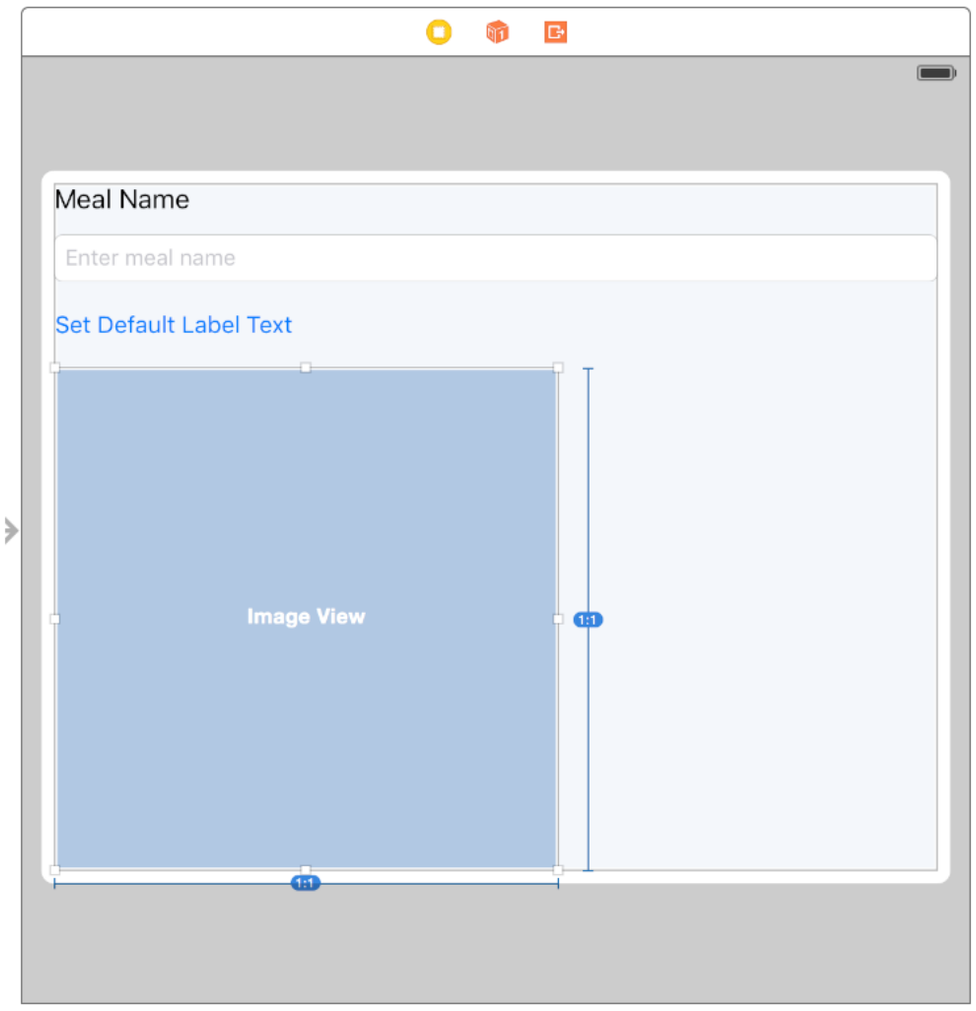
11. With the image view selected, open the Attributes inspector.

    Recall that the Attributes inspector appears when you select the fourth button from the left in the inspector selector bar. It lets you edit the properties of an object in your storyboard.

12. In the Attributes inspector, find the field labeled Interaction and select the User Interaction Enabled checkbox.

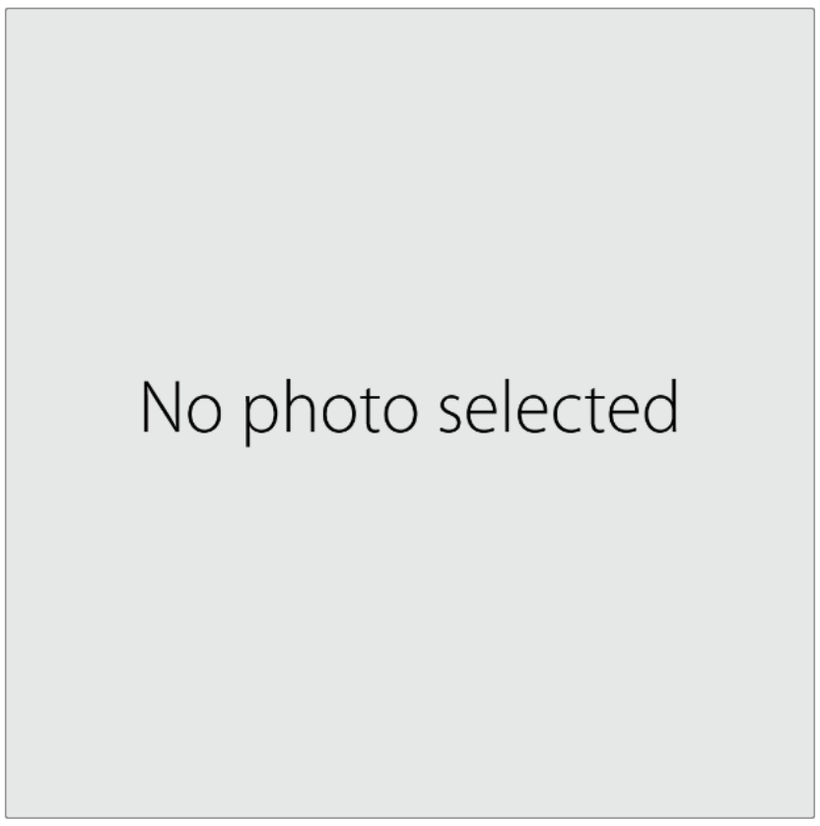
    You'll need this feature later to let users interact with the image view.

Your UI should look like this:



Display a Default Photo

Users need an indication that they can interact with the image view to select a photo. To accomplish this, add a default placeholder image that communicates to users that they can choose a photo.

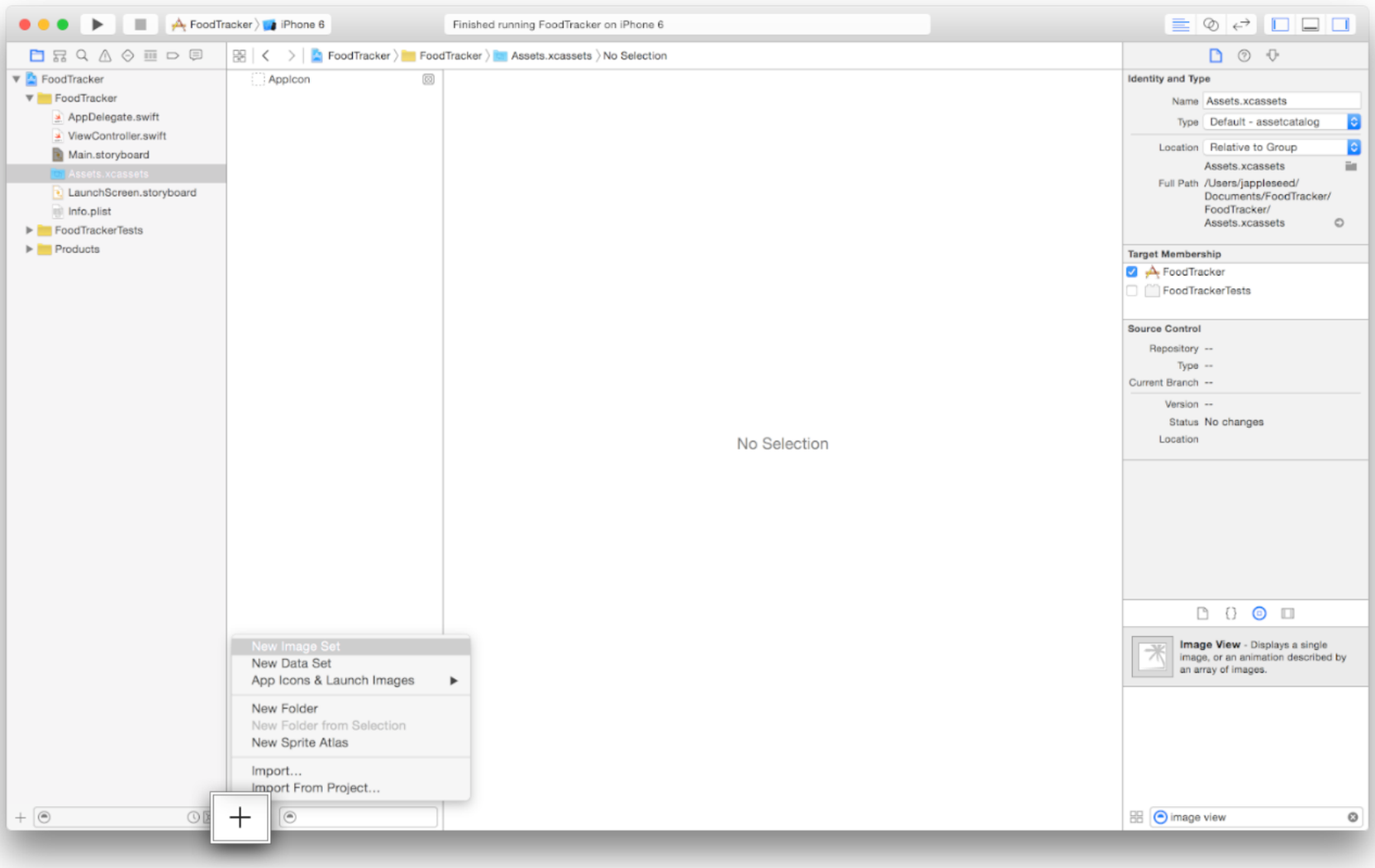


You can find the image shown above within the `Images/` folder of the downloadable file at the end of this lesson, or use your own image.

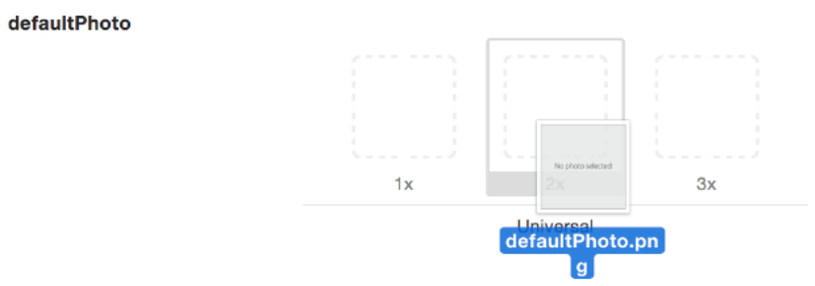
To add an image to your project

1. In the project navigator, select `Assets.xcassets` to view the asset catalog.

   The asset catalog is a place to store and organize your image assets for an app.
2. In the bottom left corner, click the plus (+) button and select New Image Set from the pop-up menu.



3. Double-click the image set name and rename it to `defaultPhoto`.
4. On your computer, select the image you want to add.
5. Drag and drop the image into the `2x` slot in the image set.



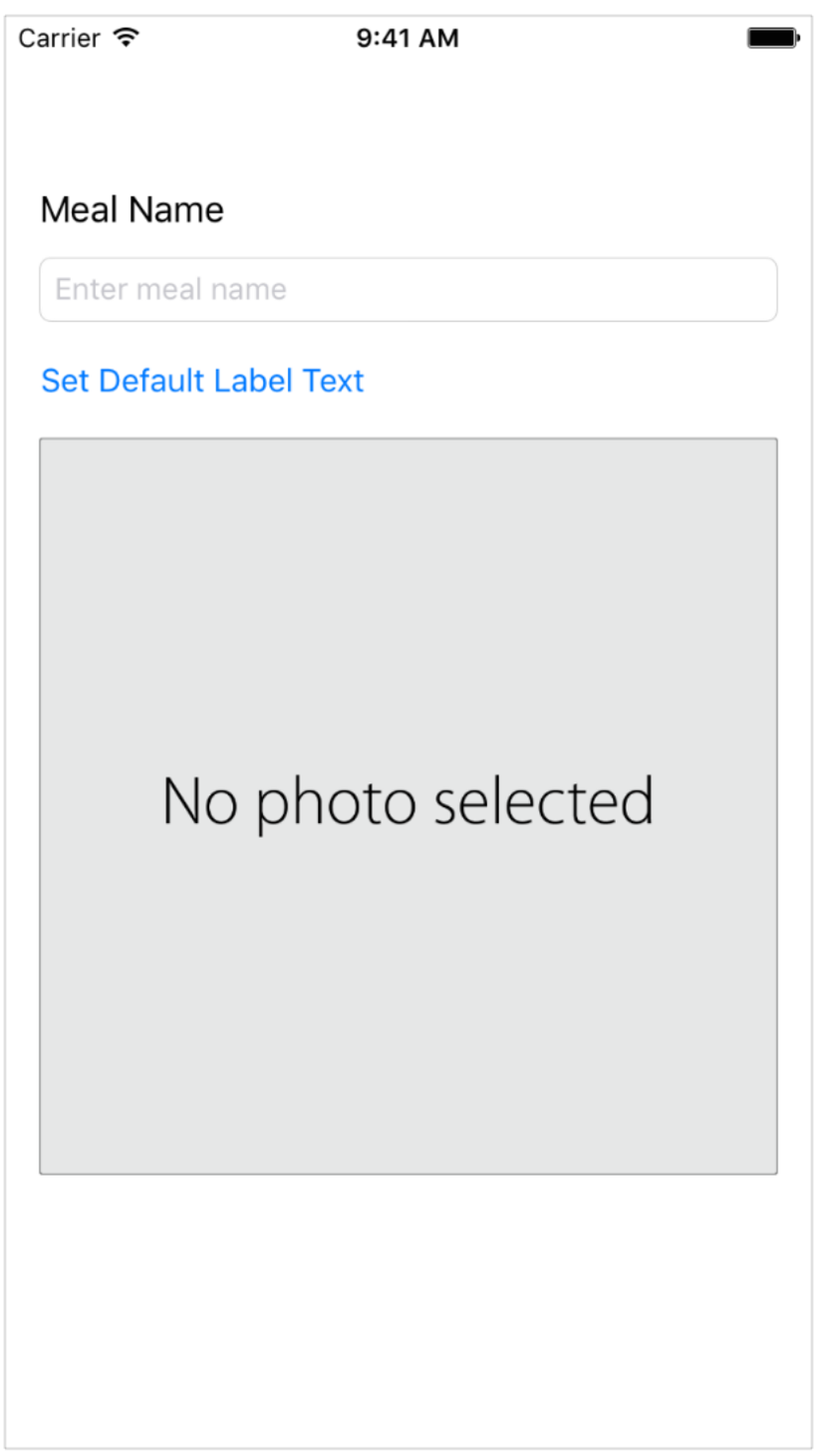
> `2x` is the display resolution for the iPhone 6 Simulator that you're using in these lessons, so the image will look best at this resolution.

With the default placeholder image added to your project, set the image view to display it.

To display a default image in the image view

1. Open your storyboard.
2. In your storyboard, select the image view.
3. With the image view selected, open the Attributes inspector in the utility area.
4. In the Attributes inspector, find the field labeled Image and select `defaultPhoto`.

Checkpoint: Run your app. The default image displays in the image view.

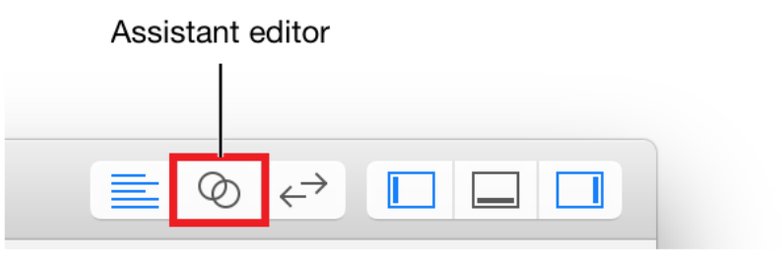


Connect the Image View to Code

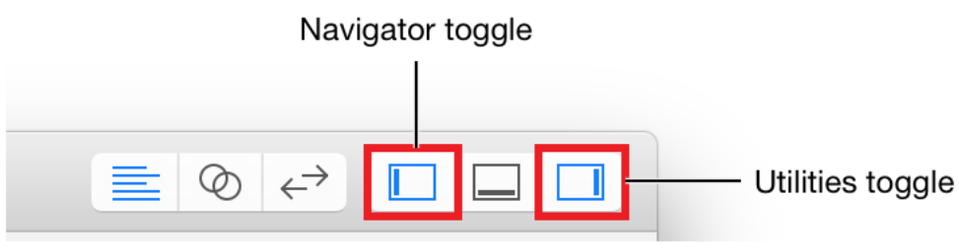
Now, you need to implement the functionality necessary to change the image in this image view at runtime. You'll want to be able to change the image from within the code. For this, you first need to connect the image view to the code in `ViewController.swift`.

To connect the image view to the ViewController.swift code

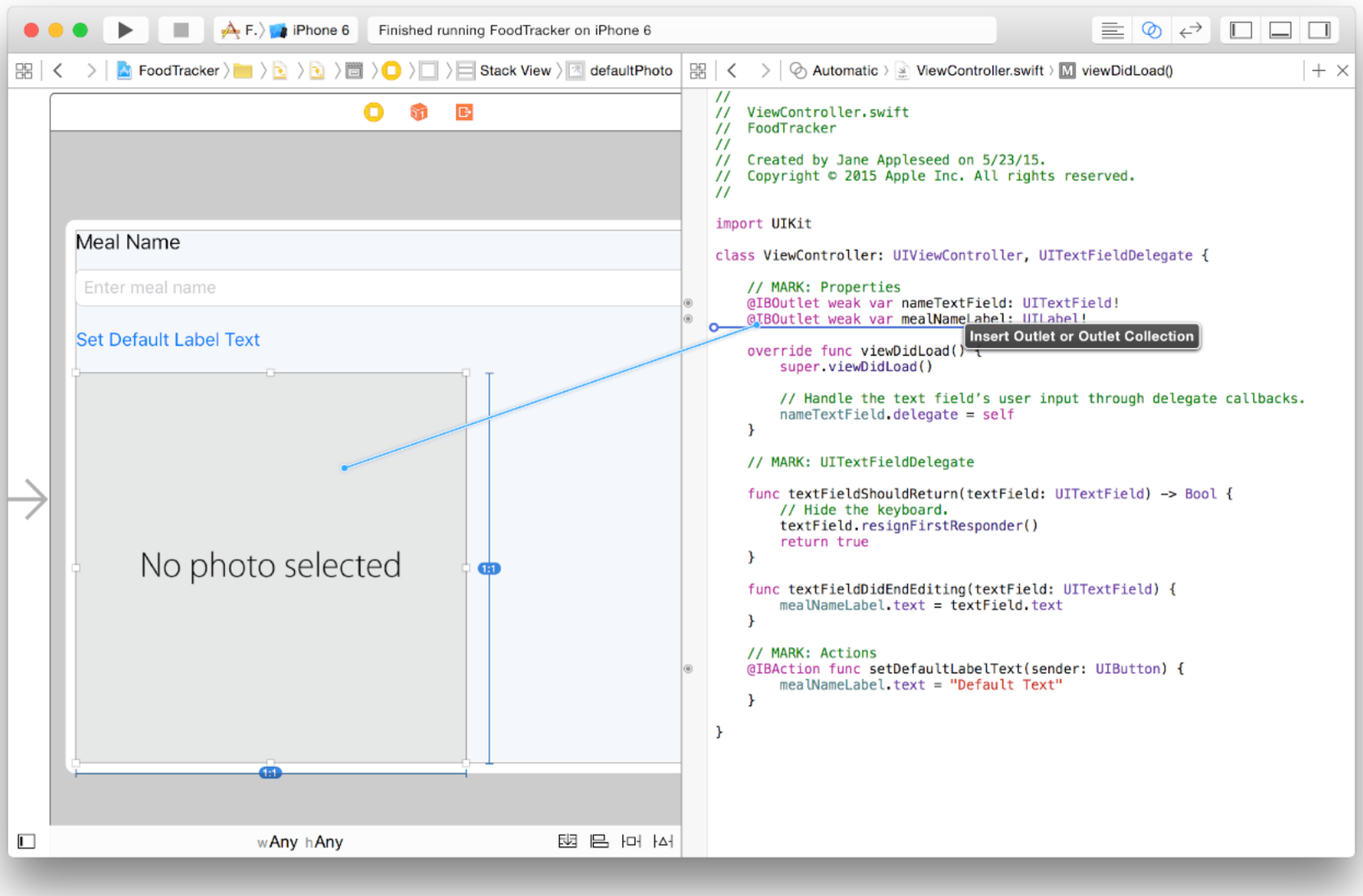
1. Click the Assistant button in the Xcode toolbar near the top right corner of Xcode to open the assistant editor.



2. 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.

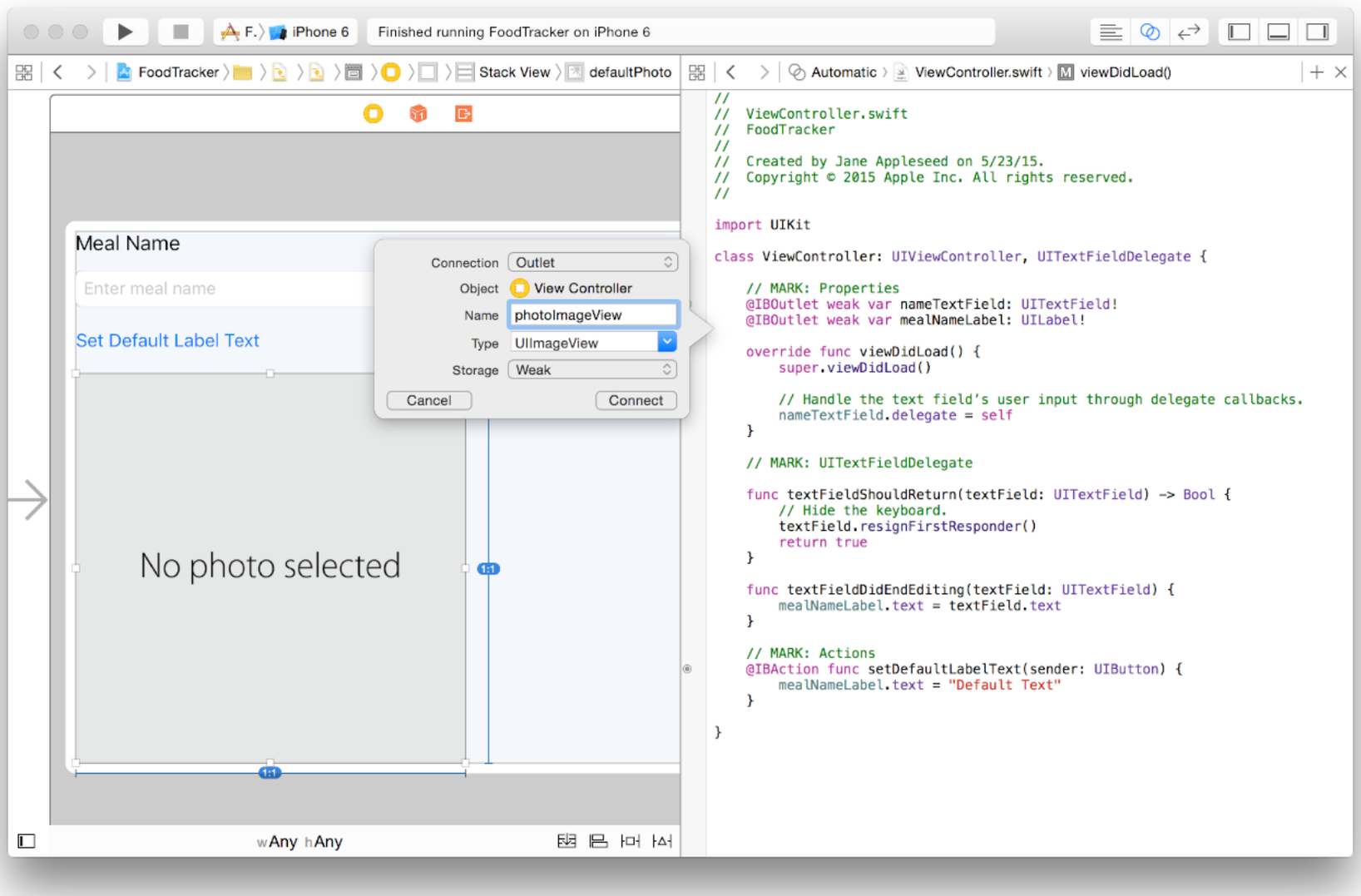


   You can also collapse the outline view.
3. In your storyboard, select the image view.
4. Control-drag from the image view on your canvas to the code display in the editor on the right, stopping the drag at the line just below the existing outlets in `ViewController.swift`.



5. In the dialog that appears, for Name, type `photoImageView`.

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



6. Click Connect.

   Xcode adds the necessary code to `ViewController.swift` to store a pointer to the image view and configures the storyboard to set up that connection.

   ```
   @IBOutlet weak var photoImageView: UIImageView!
   ```

You can now access the image view from code to change its image, but how do you know when to change the image? You need to give users a way to indicate that they want to change the image—for example, by tapping the image view. Then, you'll define an action method to change the image when a tap occurs.

There's a nuanced distinction between views and controls, which are specialized versions of views that respond to user actions in a specific way. A view displays content, whereas a control is used to modify it in

some way. A control (`UIControl`) is a subclass of `UIView`. In fact, you've already worked with both views (labels, image views) and controls (text fields, buttons) in your interface.

Create a Gesture Recognizer

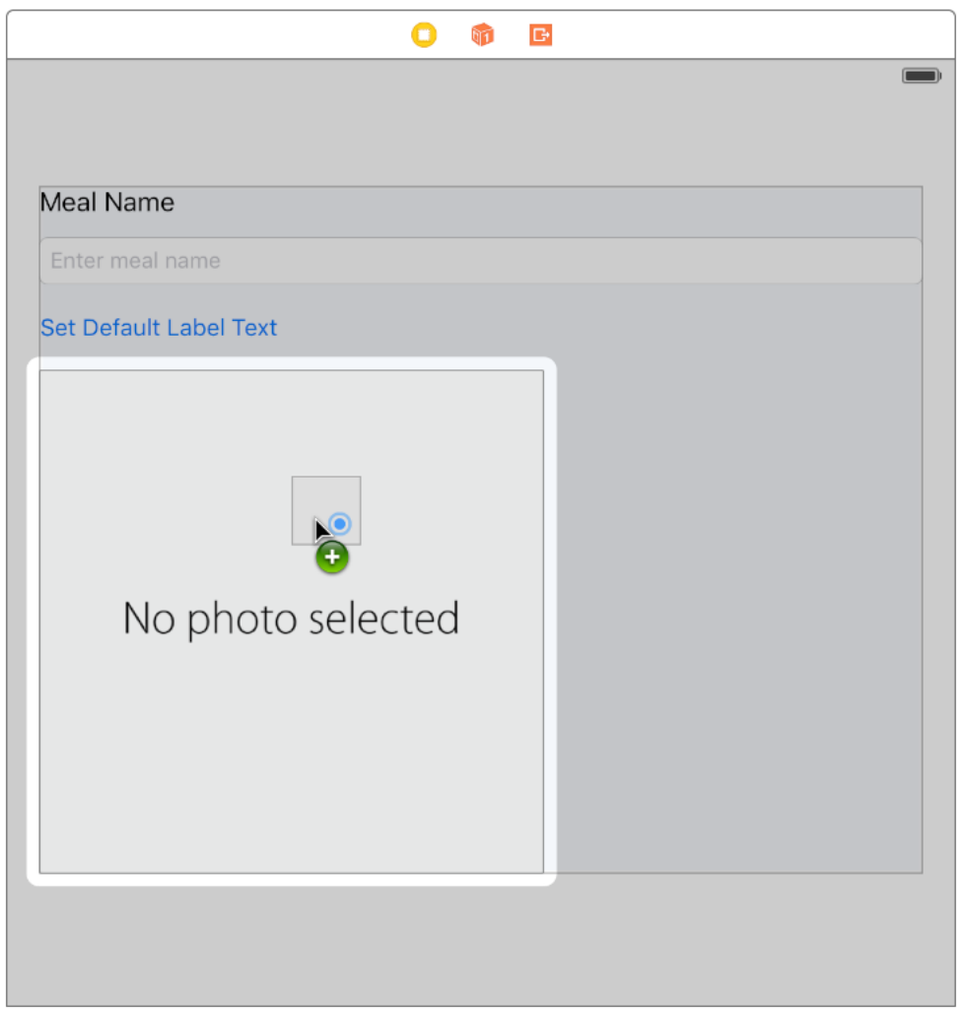
An image view isn't a control, so it's not designed to respond to input in the same way that a control—such as a button—responds. For example, you can't simply create an action method that's triggered when a user taps on an image view. (If you try to Control-drag from the image view to your code, you'll notice that you can't select Action in the Connection field.)

Fortunately, it's quite easy to give a view the same capabilities as a control by adding a gesture recognizer to it. Gesture recognizers are objects that you attach to a view that allow the view to respond to actions the way a control does. Gesture recognizers interpret touches to determine whether they correspond to a specific gesture, such as a swipe, pinch, or rotation. You can write an action method that occurs when a gesture recognizer recognizes its assigned gesture, which is exactly what you need to do for the image view.

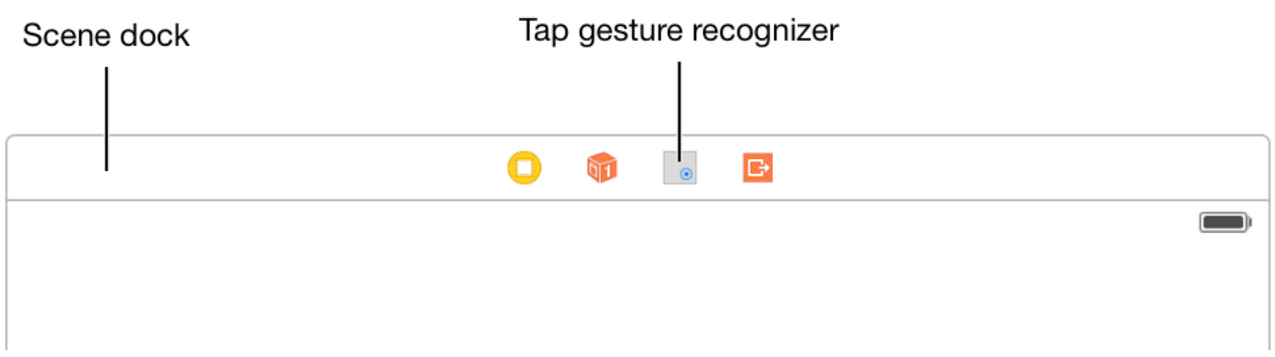
Attach a tap gesture recognizer (`UITapGestureRecognizer`) to the image view, which will recognize when a user has tapped the image view. You can do this easily in your storyboard.

To add a tap gesture recognizer to your image view

1. Open the Object library. (To open it quickly, choose View > Utilities > Show Object Library.)
2. In the Object library, type `tap gesture` in the filter field to find the Tap Gesture Recognizer object quickly.
3. Drag a Tap Gesture Recognizer object from the Object library to your scene, and place it on top of the image view.



The Tap Gesture Recognizer appears in the meal scene dock.

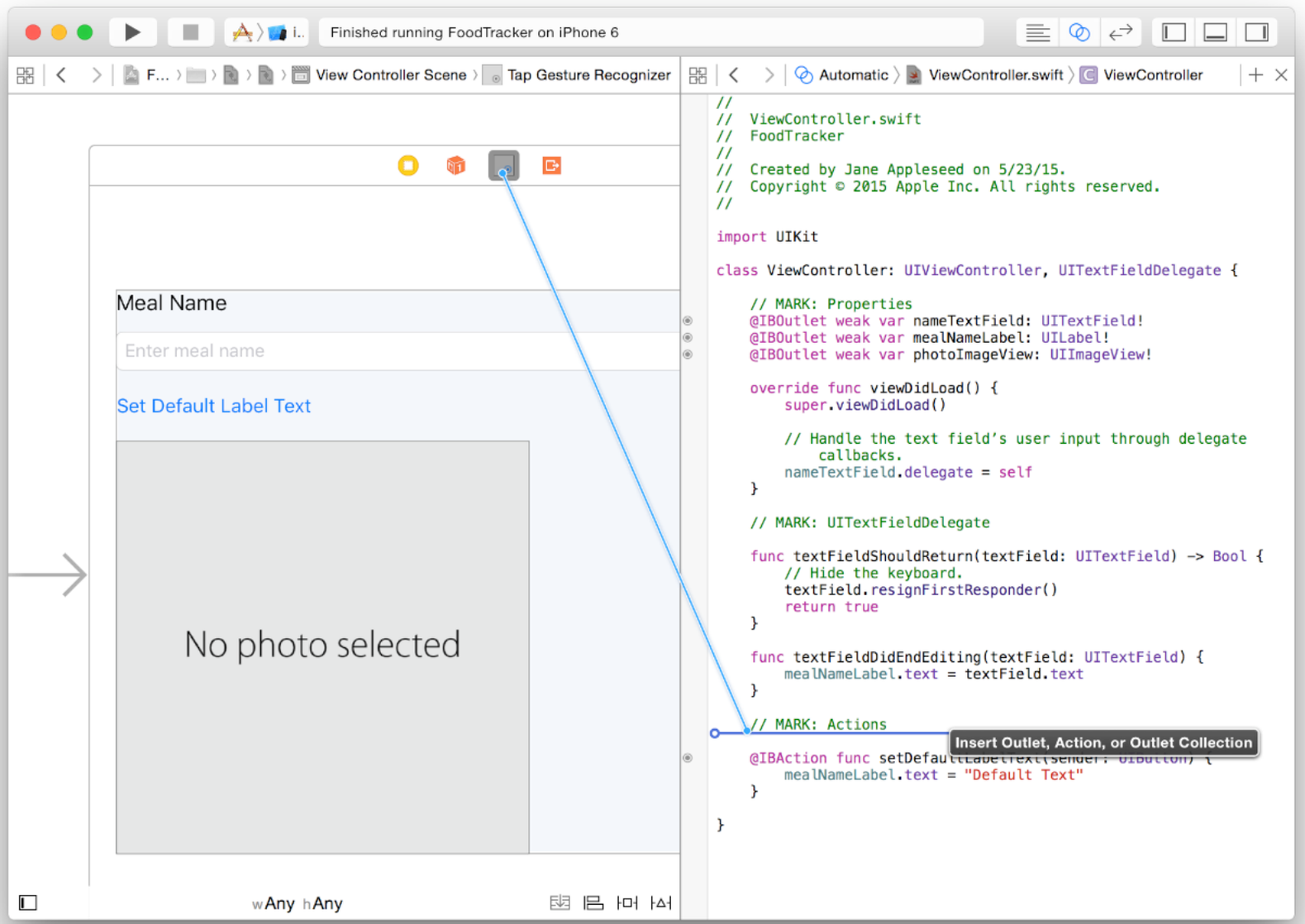


Connect the Gesture Recognizer to Code

Now, connect that gesture recognizer to an action method in your code.

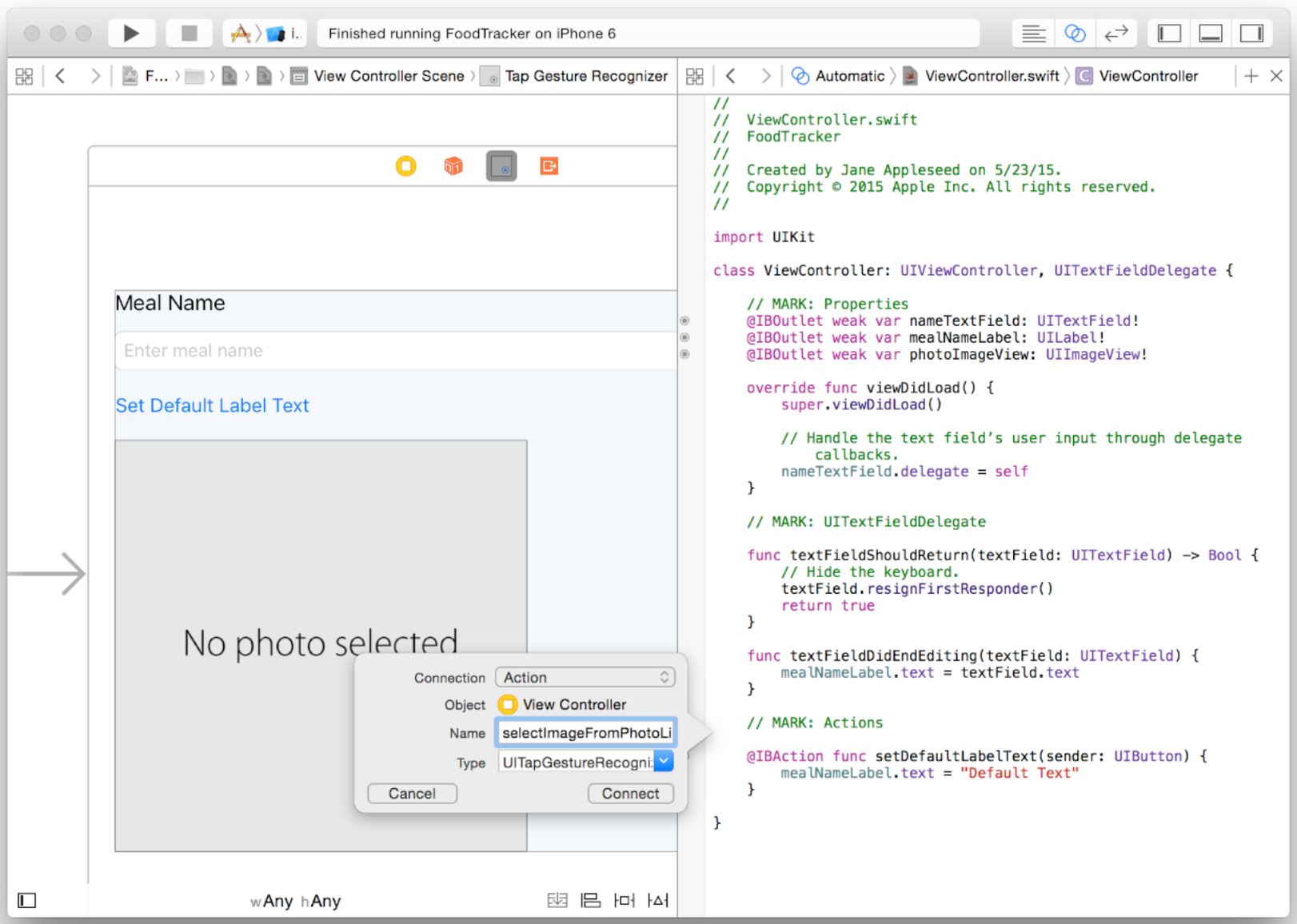
To connect the gesture recognizer to the ViewController.swift code

1. Control-drag from the gesture recognizer in the scene dock to the code display in the editor on the right, stopping the drag at the line below the `// MARK: Actions` comment in `ViewController.swift`.



2. In the dialog that appears, for Connection, select Action.
3. For Name, type `selectImageFromPhotoLibrary`.
4. For Type, select `UITapGestureRecognizer`.

   Your dialog should look like this:



5. Click Connect.

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

```
@IBAction func selectImageFromPhotoLibrary(sender: UITapGestureRecognizer) {
}
```

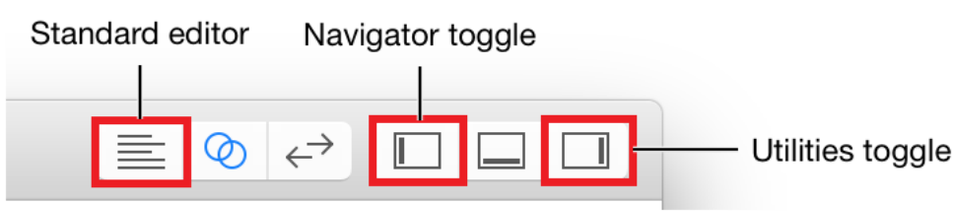
Create an Image Picker to Respond to User Taps

So what needs to happen when a user taps the image view? Presumably, users should be able to choose a photo from a collection of photos, or take one of their own. Fortunately, the `UIImagePickerController` class has all of this behavior built into it. An image picker controller manages a UI for taking pictures and for choosing saved images to use in your app. And just as you need a text field delegate when you work with a text field, you need an image picker controller delegate to work with an image picker controller. The name of that delegate protocol is `UIImagePickerControllerDelegate`, and the object that you'll define as the image picker controller's delegate is `ViewController`.

First, `ViewController` needs to adopt the `UIImagePickerControllerDelegate` protocol. Because `ViewController` will be in charge of presenting the image picker controller, it also needs to adopt the `UINavigationControllerDelegate` protocol, which simply lets `ViewController` take on some basic navigation responsibilities.

To adopt the UIImagePickerControllerDelegate and UINavigationControllerDelegate protocols

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



   Expand the project navigator and utility area by clicking the Navigator and Utilities buttons in the Xcode toolbar.
2. In the project navigator, select `ViewController.swift`.
3. In `ViewController.swift`, find the `class` line, which should look like this:

```
class ViewController: UIViewController, UITextFieldDelegate {
```

4. After `UITextFieldDelegate`, add a comma (`,`) and `UIImagePickerControllerDelegate` to adopt the protocol.

```
class ViewController: UIViewController, UITextFieldDelegate,
  UIImagePickerControllerDelegate {
```

5. After `UIImagePickerControllerDelegate`, add a comma (`,`) and `UINavigationControllerDelegate` to adopt the protocol.

```
class ViewController: UIViewController, UITextFieldDelegate,
  UIImagePickerControllerDelegate, UINavigationControllerDelegate {
```

At this point, you can go back to the action method you defined, `selectImageFromPhotoLibrary(_:)`, and finish its implementation.

To implement the selectImageFromPhotoLibrary(_:) action method

1. In `ViewController.swift`, find the `selectImageFromPhotoLibrary(_:)` action method you added earlier.

   It should look like this:

```
@IBAction func selectImageFromPhotoLibrary(sender: UITapGestureRecognizer) {
```

```
}
```

2. In the method implementation, between the curly braces (`{}`), add this code:

```
// Hide the keyboard.
nameTextField.resignFirstResponder()
```

   This code ensures that if the user taps the image view while typing in the text field, the keyboard is dismissed properly.

3. Add this code to create an image picker controller:

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

4. Add this code:

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

   This line of code sets the image picker controller's source, or the place where it gets its images. The `.PhotoLibrary` option uses Simulator's camera roll.

   The type of `imagePickerController.sourceType` is known to be `UIImagePickerControllerSourceType`, which is an enumeration. This means you can write its value as the abbreviated form `.PhotoLibrary` instead of `UIImagePickerControllerSourceType.PhotoLibrary`. Recall that you can use the abbreviated form anytime the enumeration value's type is already known.

5. Add this code to set the image picker controller's delegate to `ViewController`:

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

6. Below the previous line, add this line of code:

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

   `presentViewController(_:animated:completion:)` is a method being called on `ViewController`. Although it's not written explicitly, this method is executed on an implicit `self` object. The method asks `ViewController` to present the view controller defined by `imagePickerController`. Passing `true` to the `animated` parameter animates the presentation of the image picker controller. The `completion` parameter refers to a completion handler, a piece of code that executes after this method completes. Because you don't need to do anything else, you indicate that you don't need to execute a completion handler by passing in nil.

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

```
@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)
```

```
}
```

After an image picker controller is presented, its behavior is handed off to its delegate. To give users the ability to select a picture, you'll need to implement two of the delegate methods defined in `UIImagePickerControllerDelegate`:

```
func imagePickerControllerDidCancel(picker: UIImagePickerController)
func imagePickerController(picker: UIImagePickerController,
  didFinishPickingMediaWithInfo info: [String : AnyObject])
```

The first of these, `imagePickerControllerDidCancel(_:)`, gets called when a user taps the image picker's Cancel button. This method gives you a chance to dismiss the `UIImagePickerController` (and optionally, do any necessary cleanup). Implement `imagePickerControllerDidCancel(_:)` to do that.

To implement the imagePickerControllerDidCancel(_:) method

1. In `ViewController.swift`, right above the `// MARK: Actions` section, add the following:

   ```
   // MARK: UIImagePickerControllerDelegate
   ```

   This is a comment to help you (and anybody else who reads your code) navigate through your code and identify that this section applies to the image picker implementation.

2. Below the comment you just added, add the following method:

   ```
   func imagePickerControllerDidCancel(picker: UIImagePickerController) {
   }
   ```

3. In this method, add the following line of code:

   ```
   // Dismiss the picker if the user canceled.
   dismissViewControllerAnimated(true, completion: nil)
   ```

   This code animates the dismissal of the image picker controller.

Your `imagePickerControllerDidCancel(_:)` method should look like this:

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

The second `UIImagePickerControllerDelegate` method that you need to implement, `imagePickerController(_:didFinishPickingMediaWithInfo:)`, gets called when a user selects a photo. This method gives you a chance to do something with the image or images that a user selected from the picker. In your case, you'll take the selected image and display it in your UI.

To implement the imagePickerController(_:didFinishPickingMediaWithInfo:) method

1. Below the `imagePickerControllerDidCancel(_:)` method, add the following method:

   ```
   func imagePickerController(picker: UIImagePickerController,
     didFinishPickingMediaWithInfo info: [String : AnyObject]) {
   }
   ```

2. In this method, add the following line of code:

   ```
   // The info dictionary contains multiple representations of the image, and
     this uses the original.
   let selectedImage = info[UIImagePickerControllerOriginalImage] as! UIImage
   ```

   The `info` dictionary contains the original image that was selected in the picker, and the edited version of that image, if one exists. To keep things simple, you'll use the original, unedited image for the meal photo, which is what this line of code stores into the `selectedImage` constant.

3. Add this line of code to set the selected image in the image view outlet that you created earlier:

```
// Set photoImageView to display the selected image.
photoImageView.image = selectedImage
```

4. Add the following line of code to dismiss the image picker:

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

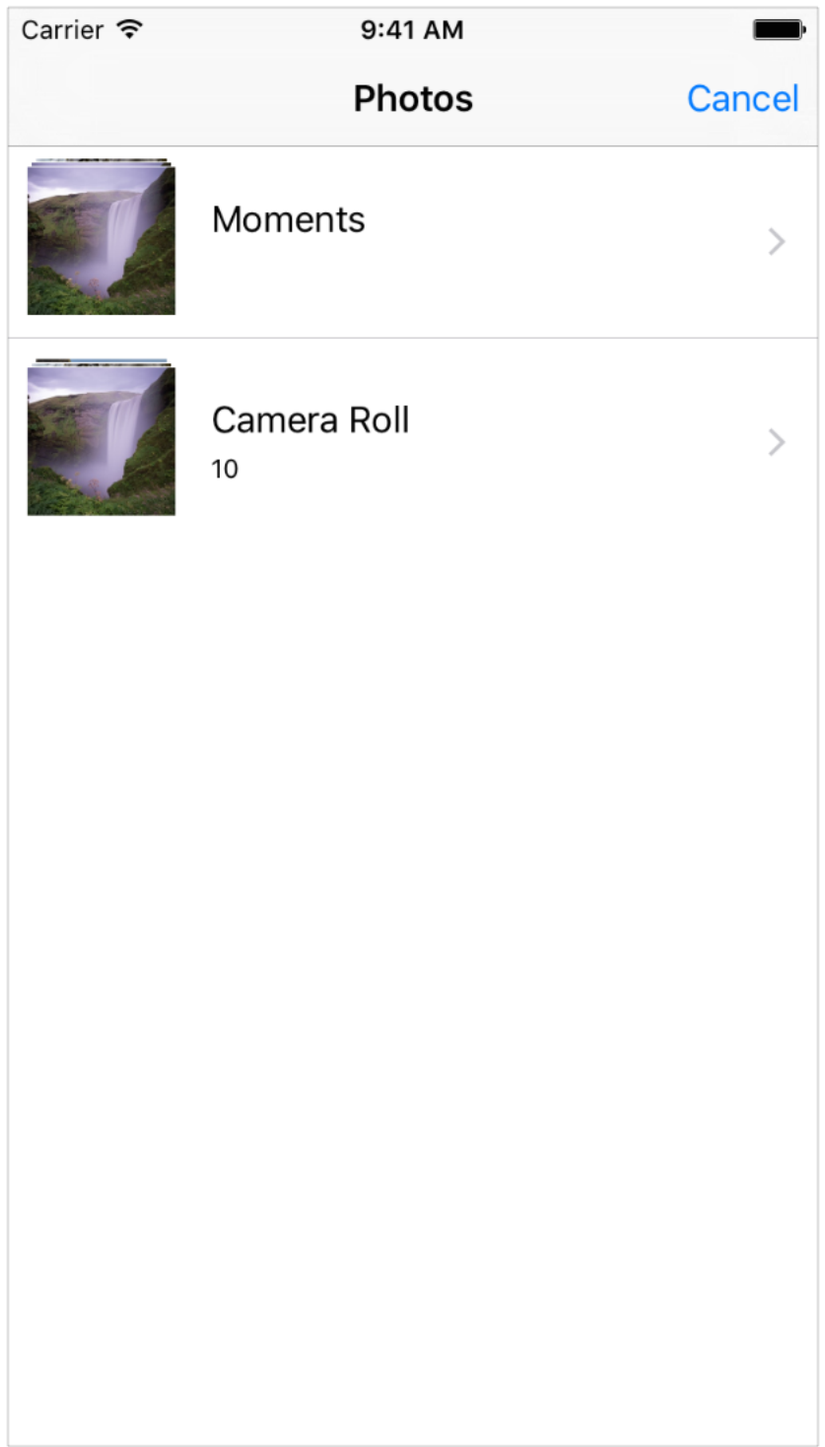
Your `imagePickerController(_:didFinishPickingMediaWithInfo)` method should look like this:

```
func imagePickerController(picker: UIImagePickerController,
  didFinishPickingMediaWithInfo info: [String : AnyObject]) {
    // The info dictionary contains multiple representations of the image, and this
  uses the original.
    let selectedImage = info[UIImagePickerControllerOriginalImage] as! UIImage

    // Set photoImageView to display the selected image.
    photoImageView.image = selectedImage

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

Checkpoint: Run your app. You should be able to click the image view to pull up an image picker. You'll need to click OK on the alert that asks for permission to give the FoodTracker app access to Photos. Then, you can click the Cancel button to dismiss the picker, or open Camera Roll and click an image to select it and set it as the image in the image view.

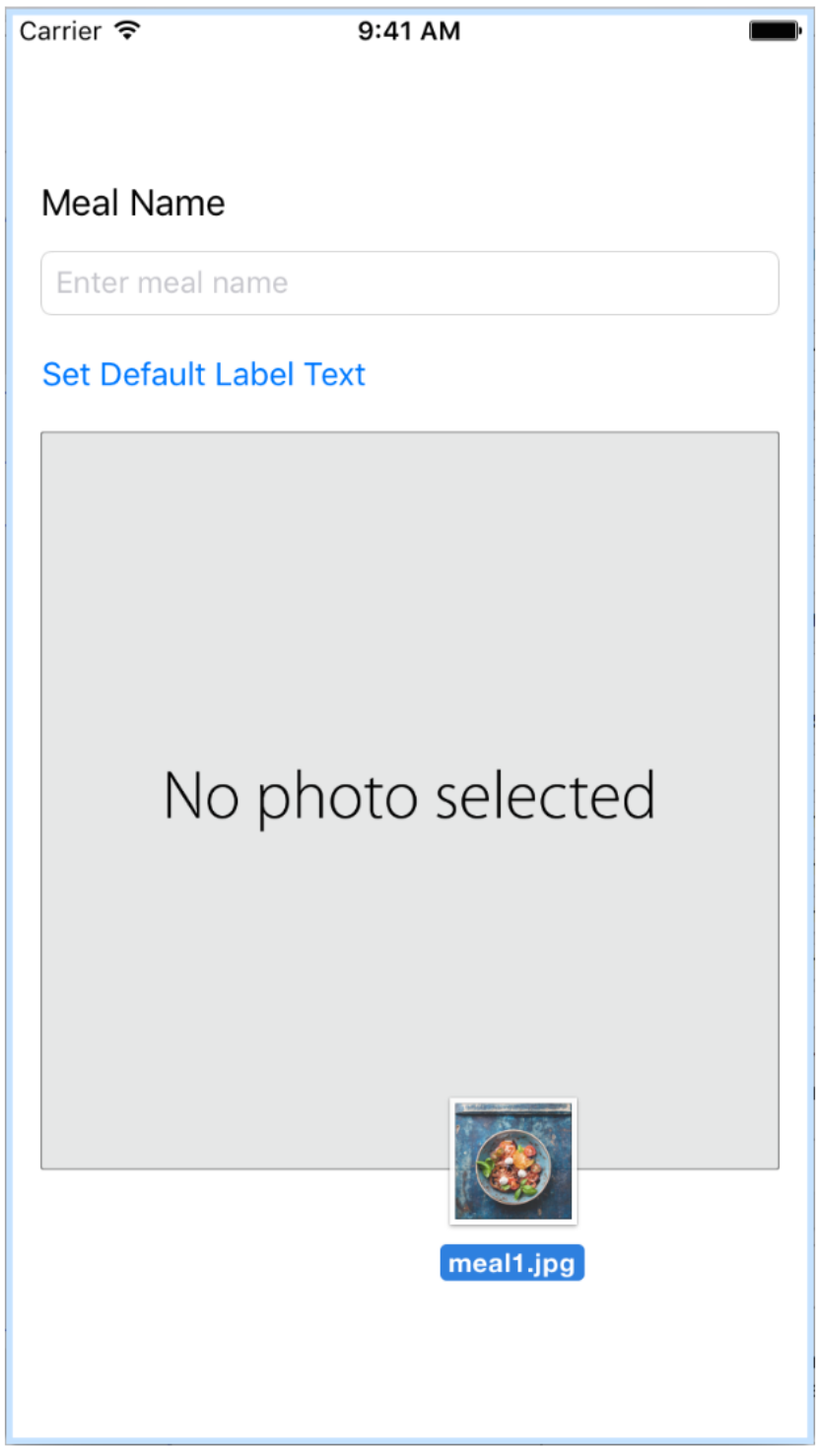


If you look through the photos available in Simulator, you'll notice that it doesn't include any photos of food. You can add your own images directly into Simulator to test the FoodTracker app with appropriate sample content. You can find a sample image within the `Images/` folder of the downloadable file at the end of this

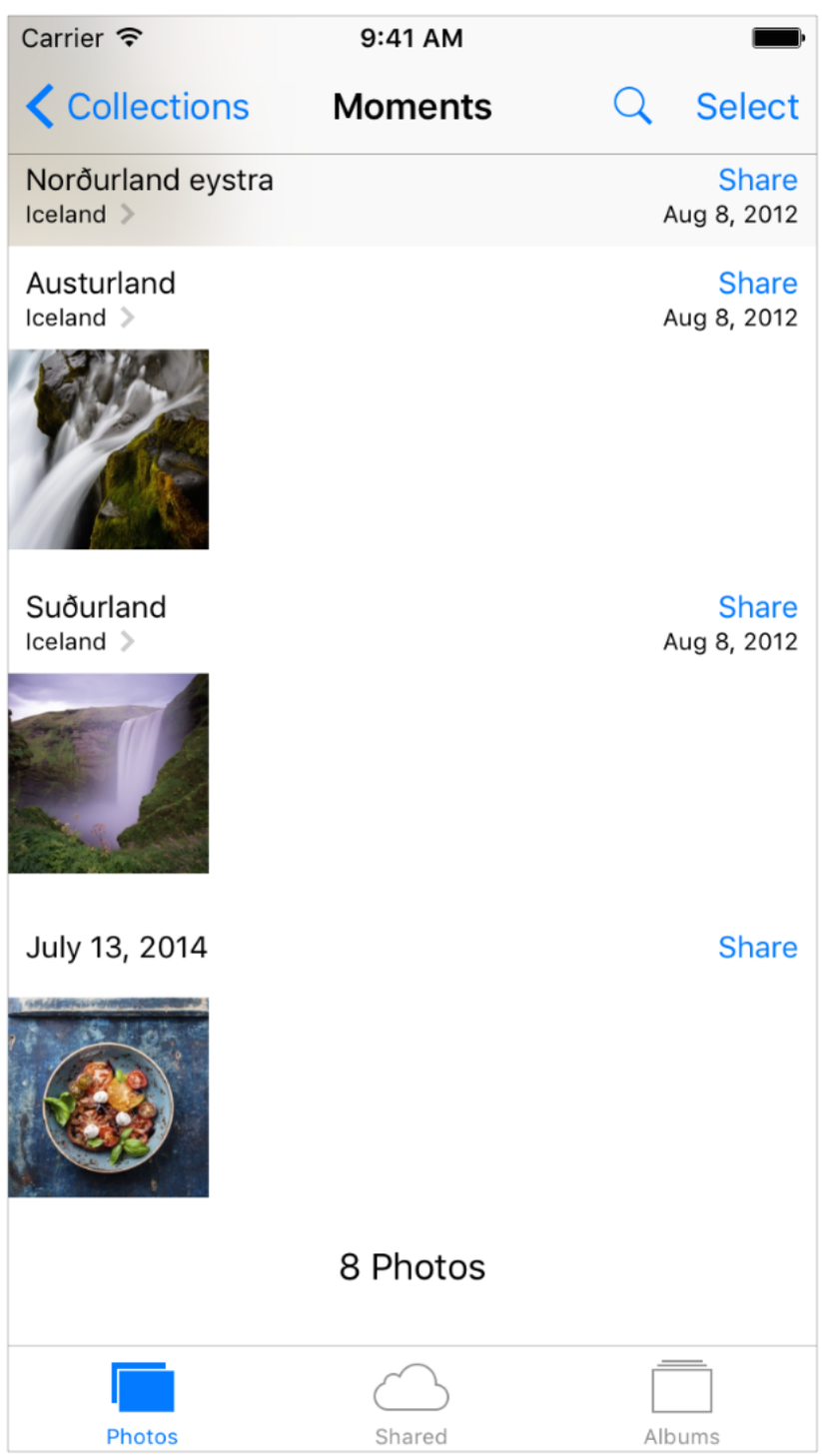
lesson, or use your own image.

To add images to Simulator

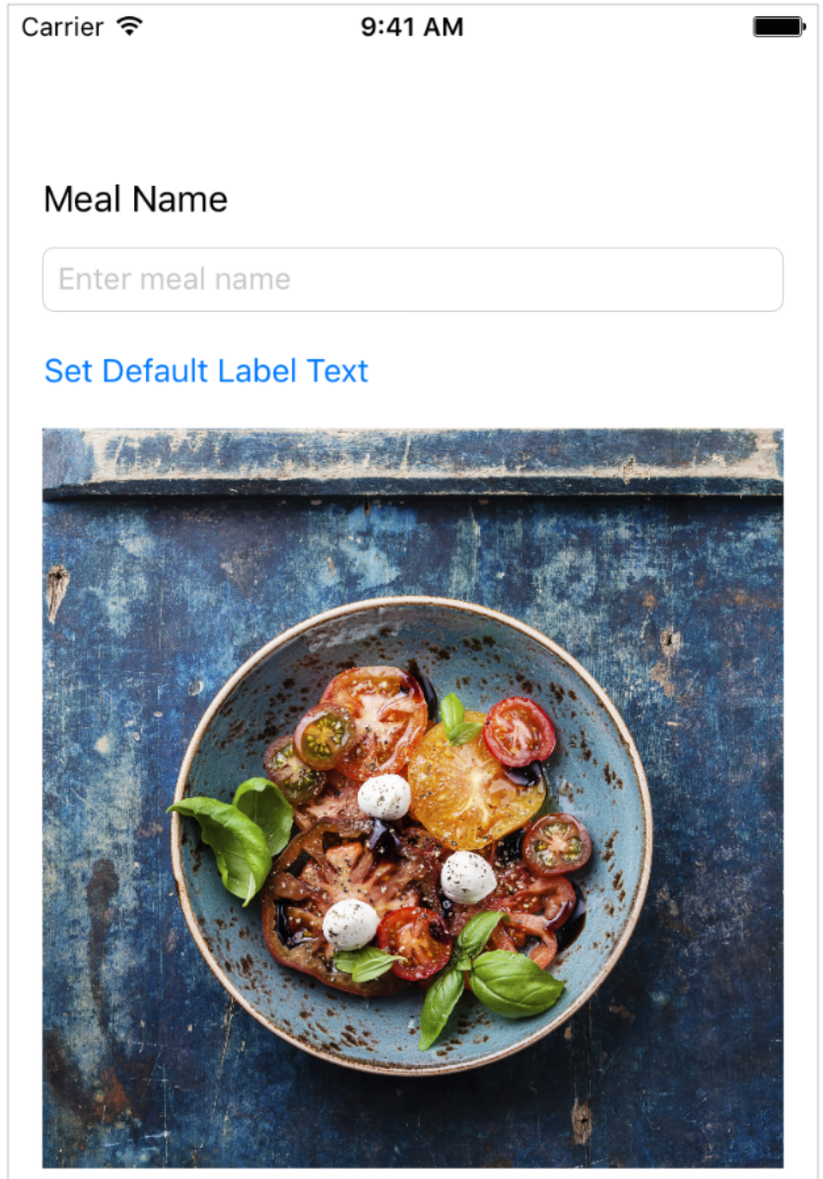
1. If necessary, run your app in Simulator.
2. On your computer, select the images you want to add.
3. Drag and drop the images into Simulator.



Simulator opens the Photos app and shows the images you added.



Checkpoint: Run your app. You should be able to tap the image view to pull up an image picker. Open Camera Roll, and click one of the images you added to Simulator to select it and set it as the image in the image view.



NOTE

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

Download File