CHAPTER 19

INTERACTING WITH COLLECTION VIEW

HANDLING CELL SELECTIONS IN COLLECTION VIEW

In the previous chapter I covered the basics of UICollectionView, so you should already know how to display items using that view object. However, you may not know how to interact with the collection view cells.

As mentioned before, a collection view works pretty much like a table view. To give you a better idea, I will cover how to interact with collection items in order to handle single and multiple item selections. To provide you with a working example of how an item selection works, we'll continue to improve the collection view demo app. Here is what we're going to implement:

- When a user taps a recipe photo, the app will bring up a modal view and display the photo in a larger size.
- We'll also implement Facebook sharing in the app in order to demonstrate multiple item selections. Users will be allowed to select multiple photos and share them on Facebook.

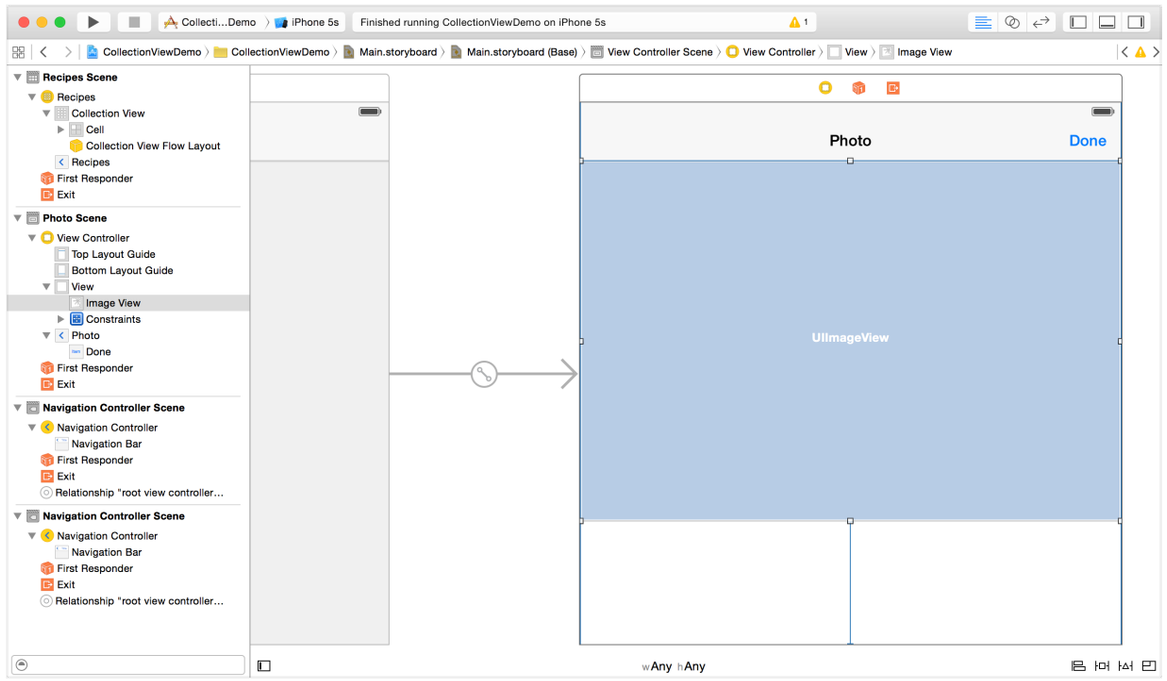
First, we'll improve the collection view demo app (https://www.dropbox.com/s/o1anhhgdxmoxz1p/CollectionViewDemo.zip?dl=0) to handle single selection. When the user taps any of the recipe photos, the app will bring up a modal view to display the photo in a higher resolution.

To begin, let's design the view controller that will display the recipe photo. Go to Main.storyboard, drag a View Controller from the Object library. Then add an Image View to it and set the width and height to 600 and 400 respectively. Under the Attributes inspector, change the mode of the image view to *Aspect Fill* and enable *Clip Subviews.*

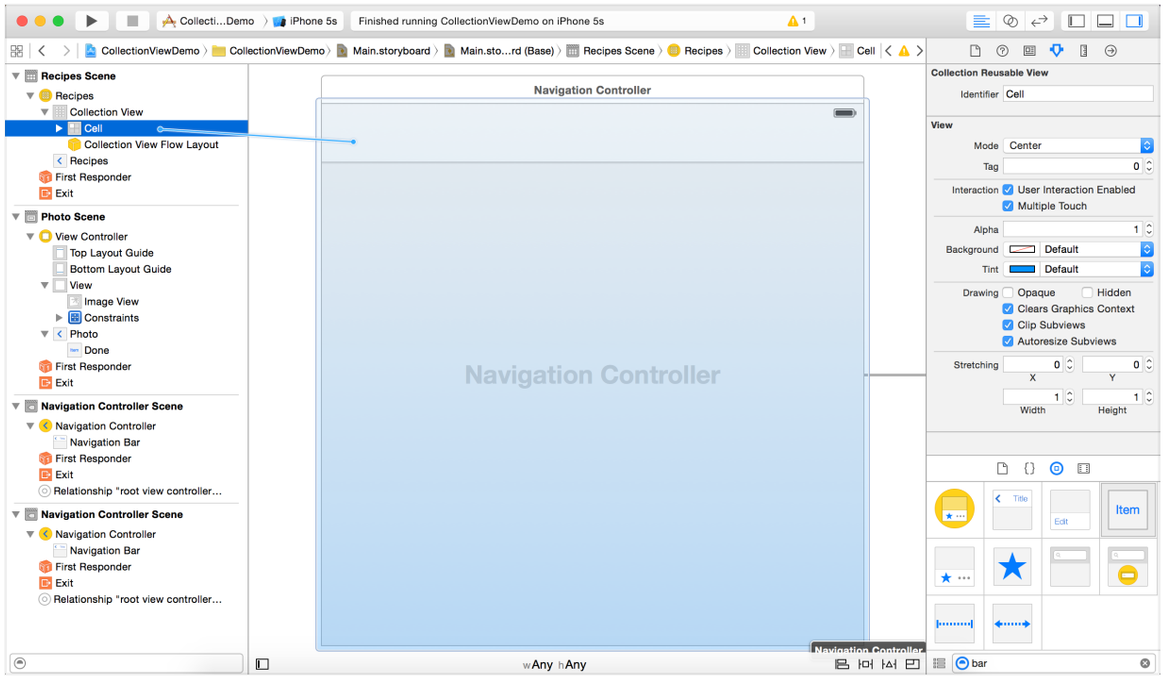
Lastly, embed it in a navigation controller and add a Bar Button Item to the navigation bar. Change the title of the navigation bar to "Photo". Under the Attributes Inspector, set the identifier of the button item to "Done".

Remember to define auto layout constraints for the image view. Simply select it and click the Issues button of the auto layout menu, followed by choosing the *Add Missing Constraints* option.

Your view controller should look similar to the screenshot below.



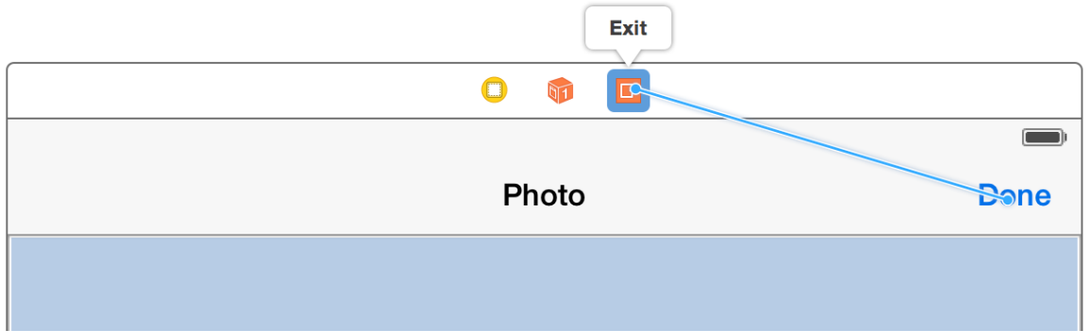
Since we want to display the view controller when the user taps any of the recipe photos in the collection view, we will connect the collection view with the view controller using a segue. Press the control button, click on the Cell of the collection view in the Document Outline and drag it towards the navigation controller we just added. Select “Present Modally” for the style and set the segue identifier to “showRecipePhoto.”



In RecipeCollectionViewController.swift, insert the following unwind segue method:

```
@IBAction func unwindToHome(segue: UIStoryboardSegue) {
}
```

Go back to the storyboard. Control-drag from the Done button to the Exit icon and select *unwindToHome:* segue when prompted.



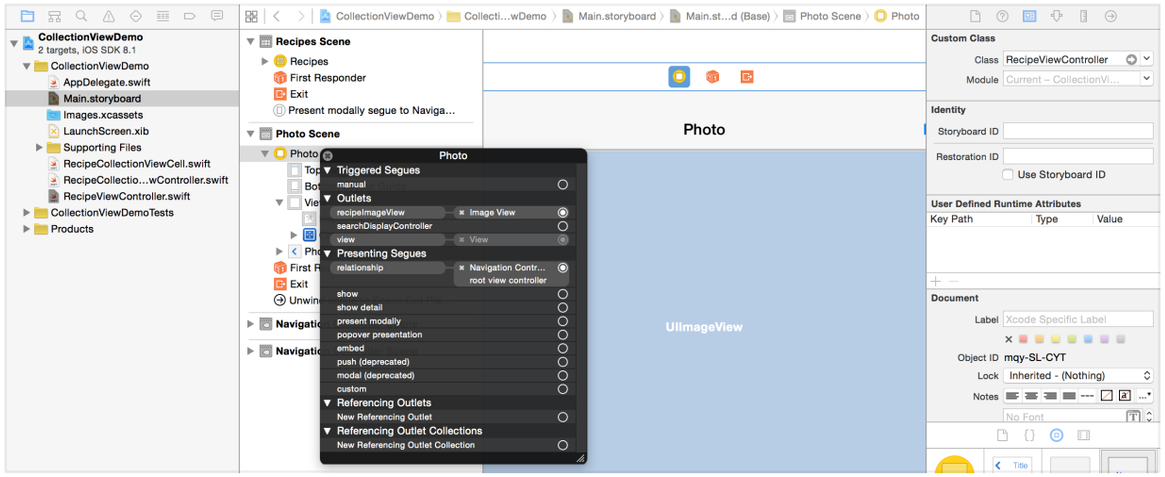
If you compile and run the app, you’ll end up with an empty view when selecting any of the recipe photos. Tapping the Done button will dismiss the view.

Since we haven’t written the code, the modal view controller knows nothing about the selected recipe photo. Create a new class and name it *RecipeViewController*. Make it a subclass of UIViewController.

Once created, add an outlet variable in the RecipeViewController class:

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

In Storyboard, select the view controller we just created and set the custom class to RecipeViewController. Then establish a connection between the image view and the recipeImageView outlet variable.



DATA PASSING

In order to let other controllers pass the image name, we'll add a *recipeImageName* property in the RecipeViewController class. Declare the property like this:

```
var recipeImageName:String = " "
```

To load the specified recipe image in the image view, change the *viewDidLoad* method of RecipeViewController.swift to the following:

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

    recipeImageView.image = UIImage(named: recipeImageName)
}
```

Now the RecipeViewController class should be able to load the recipe image in the image view - but there's still one thing left. How can we identify the selected item of the collection view and pass the image name to the RecipeViewController?

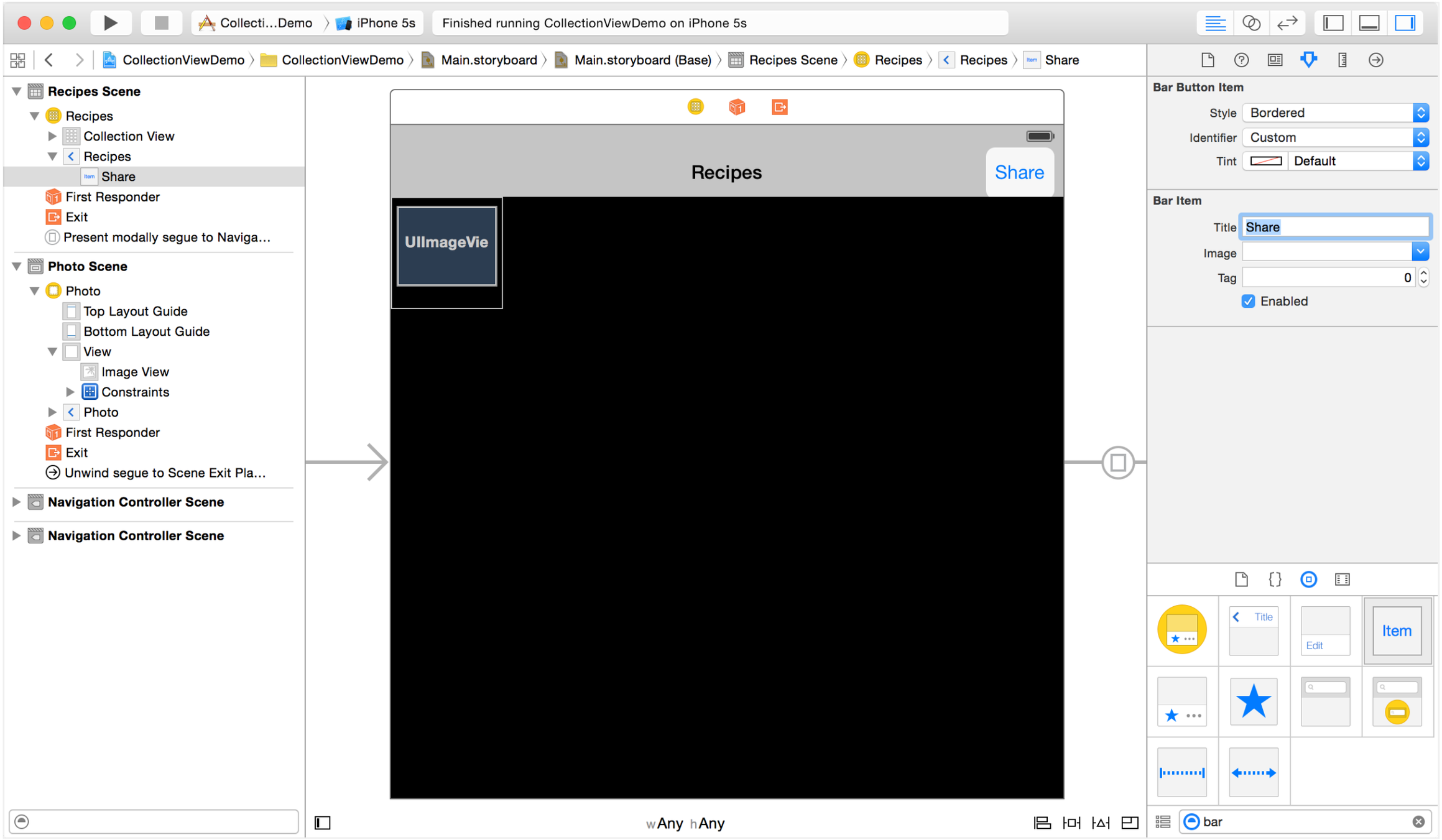
If you understand how data passing works via a segue, you know we must implement the *prepareForSegue* method in the RecipeCollectionViewController class. Select RecipeCollectionViewController.swift and add the following code:

```
override func prepareForSegue(segue: UIStoryboardSegue, sender:
AnyObject?) {
    if segue.identifier == "showRecipePhoto" {
        var indexPaths = collectionView?.indexPathsForSelectedItems() as!
[NSIndexPath]
        var destinationViewController = segue.destinationViewController as!
UINavigationController
        var recipeViewController =
destinationViewController.viewControllers[0] as! RecipeViewController
        recipeViewController.recipeImageName =
recipeImages[indexPaths[0].row]
        collectionView?.deselectItemAtIndexPath(indexPaths[0], animated:
false)
    }
}
```

Just like UITableView, the UICollectionView class provides the *indexPathsForSelectedItems* method that returns the index paths of the selected items. You may wonder why multiple index paths are returned. The reason is that UICollectionView supports multiple selections. Each of the index paths corresponds to a selected item. For this demo, we only have single item selection. Therefore, we pick the first index path and retrieve the selected image.

When a user taps a collection cell in the collection view, the cell changes to the highlighted state and then to the selected state. The last line of code is to deselect the selected item once the image is displayed in the modal view controller.

Now, let's build and run the app. After the app is launched, tap any of the recipes and you should see a modal view showing the selected recipe image.



HANDLING MULTIPLE SELECTIONS

UICollectionView supports both single and multiple selections. By default, the app only allows users to select a single item. The *allowsMultipleSelection* property of the UICollectionView class controls whether multiple items can be selected simultaneously. To enable multiple selections, the trick is to set the property to *true*.

To give you a better idea of how multiple selections work, we'll continue to tweak the demo app. Users are allowed to select multiple recipes and share them on Facebook in the following ways:

- A user taps the "Share" button in the navigation bar. Once the sharing starts, the button title is automatically changed to "Upload."
- The user selects the recipe photos to share.
- After the selection, the user taps the "Upload" button. The app will bring up a dialog for sharing the photos on Facebook.

We'll first add the "Share" button in the navigation bar of Recipe Collection View Controller. Go to Storyboard, drag a Bar Button Item from the Object library and put it in the navigation bar of Recipe Collection View Controller.

In RecipeCollectionViewController.swift, insert an outlet variable for the Share button:

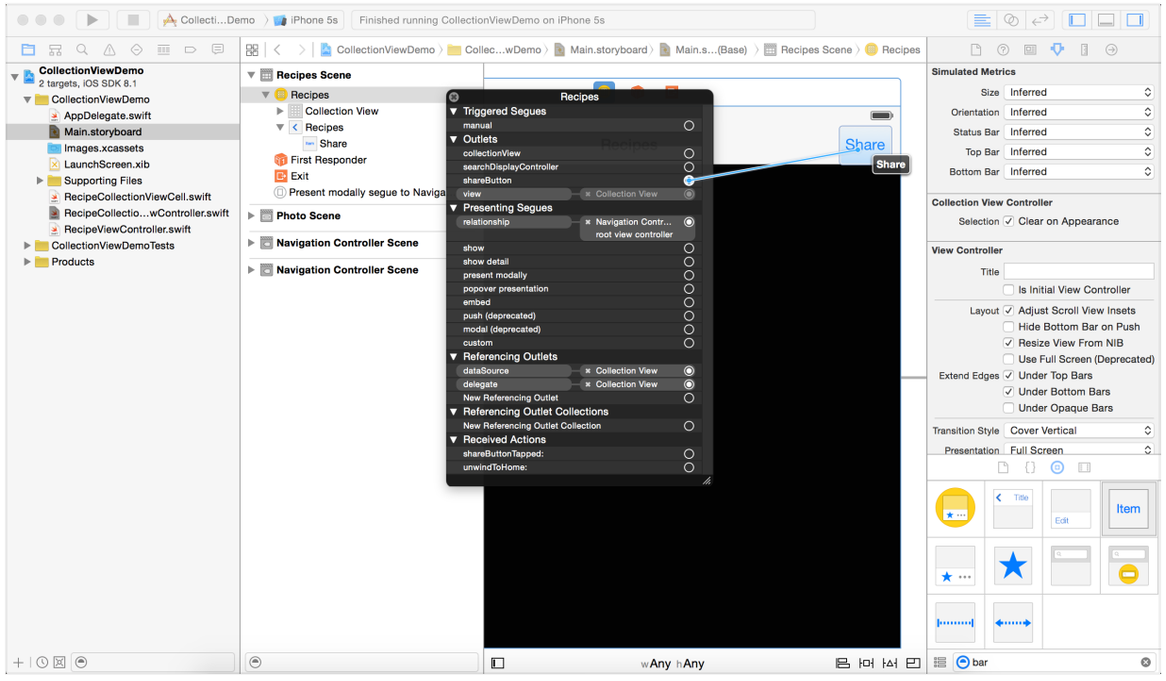
```
@IBOutlet weak var shareButton:UIBarButtonItem!
```

Also, add an action method:

```
@IBAction func shareButtonTapped(sender: AnyObject) {


}
```

As usual, go to Storyboard. Establish a connection between the Share button and RecipeCollectionViewController.swift.



The demo app now offers two modes: single selection and multiple selections. When a user taps the Share button, the app goes into multiple selection mode. This allows users to select multiple photos for sharing. To support multiple selection mode, we'll add two variables in RecipeCollectionViewController.swift:

- *shareEnabled* – this is a boolean variable to indicate the selection mode. If it's set to *true*, it indicates the Share button was tapped and multiple selection is enabled.
- *selectedRecipes* – this is an array to store the selected recipes

Your code should look like this:

```
var shareEnabled = false
var selectedRecipes:[String] = []
```

The UICollectionViewDelegate protocol defines methods that allow you to manage the selection and highlight items in a collection view. When a user selects an item, the *didSelectItemAtIndexPath* method will be called. We'll implement this method and add the selected items into the *selectedRecipes* array. Insert the following method in RecipeCollectionViewController.swift:

```
override func collectionView(collectionView: UICollectionView,
didSelectItemAtIndexPath indexPath: NSIndexPath) {

    if shareEnabled {
        // Determine the selected items by using the indexPath
        var selectedRecipe = recipeImages[indexPath.row]
        // Add the selected item into the array
        selectedRecipes.append(selectedRecipe)
    }
}
```

The UICollectionViewCell class provides a property named *selectedBackgroundView* for setting the background view of a selected item. To indicate a selected item, we'll change the background image of a collection cell. I've included the photo-frame-selected.png in the project template. If you didn't use the template, you can download the image from https://www.dropbox.com/s/owpr02wk24aq7n7/photo-frame-selected.png?dl=0 and add it to the image asset. Edit the *cellForItemAtIndexPath* method and insert the following line of code:

```
cell.selectedBackgroundView = UIImageView(image: UIImage(named:
"photo-frame-selected"))
```

Now when a user selects a recipe item, the selected cell will be highlighted.

Not only do we have to handle item selection, we also need to account for deselection. A user may deselect an item from the collection view. When an item is deselected, it should be removed from the *selectedRecipes* array. Insert the following code in RecipeCollectionViewController.swift:

```
override func collectionView(collectionView: UICollectionView,
didDeselectItemAtIndexPath indexPath: NSIndexPath) {

    if shareEnabled {
        var deSelectedRecipe = recipeImages[indexPath.row]
        if var index = find(recipeImages, deSelectedRecipe) {
            recipeImages.removeAtIndex(index)
        }
    }
}
```

Next, we'll move onto the implementation of the *shareButtonTouched* method. The method is called when a user taps the Share button. Update the method to the following code:

```
@IBAction func shareButtonTapped(sender: AnyObject) {
    if shareEnabled {

        // Post selected photos to Facebook
        if selectedRecipes.count > 0 {
            if
SLComposeViewController.isAvailableForServiceType(SLServiceTypeFace
book) {
                let facebookComposer =
SLComposeViewController(forServiceType: SLServiceTypeFacebook)
                facebookComposer.setInitialText("Check out my recipes!")

                for recipePhoto in selectedRecipes {
                    facebookComposer.addImage(UIImage(named:
recipePhoto))
                }

                presentViewController(facebookComposer, animated: true,
completion: nil)

            }
        }

        // Deselect all selected items
        for indexPath in collectionView?.indexPathsForSelectedItems() as!
[NSIndexPath]  {
            collectionView?.deselectItemAtIndexPath(indexPath, animated:
false)
        }

        // Remove all items from selectedRecipes array
        selectedRecipes.removeAll(keepCapacity: true)

        // Change the sharing mode to NO
        shareEnabled = false
        collectionView?.allowsMultipleSelection = false
        shareButton.title = "Share"
        shareButton.style = UIBarButtonItemStyle.Plain

    } else {

        // Change shareEnabled to YES and change the button text to
Upload
        shareEnabled = true
        collectionView?.allowsMultipleSelection = true
        shareButton.title = "Upload"
        shareButton.style = UIBarButtonItemStyle.Done

    }
}
```

Let's take a look at the above code line by line.

When the app is in sharing mode (i.e. shareEnabled is set to true), after the user taps the "Upload" button, we'll bring up the Facebook composer. The iOS SDK allows you to integrate the social sharing feature in your app via the SLComposeViewController class. The SLComposeViewController comes with some built-in methods to compose a post for various social networking services such as Facebook and Twitter. Here we use some of its built-in methods to upload multiple photos to Facebook. We first use the *isAvailableForServiceType* method to check if Facebook is configured on the current iOS device. If it's configured, we create a Facebook composer and attach the selected images using the *addImage* method. Finally, we call up the *presentViewController* method to display the composer on screen.

If you do not understand how SLComposeViewController works, refer back to Chapter 5 for details.

After uploading the photos to Facebook, we deselect the selected items and remove them from the selectedRecipes array. Lastly, we switch back to single selection mode and change the button title.

If sharing mode was originally disabled (i.e. the value of shareEnabled is set to false), we'll put the app into sharing mode and enable multiple selections. To enable multiple selections, all you need to do is set the *allowsMultipleSelection* property to *true*. Finally, we change the title of the button to "Upload."

Because SLComposeViewController is a class provided by the Social framework, remember to import the Social framework at the very top of RecipeCollectionViewController.swift:

```
import Social
```

The app is almost ready. However, if you run the app now, you will end up with a bug. After switching to sharing mode, the modal view still appears when you select any of the recipe photos - the result is not what we expected. The segue is invoked every time a collection view cell is tapped. Obviously, we don't want to trigger the segue when the app is in sharing mode. We only want to trigger the segue when it's in single selection mode.

The *shouldPerformSegueWithIdentifier* method allows you to control the performance of a segue. Insert the following code snippet in RecipeCollectionViewController.swift:

```
override func shouldPerformSegueWithIdentifier(identifier: String?, sender: AnyObject?) -> Bool {
    if identifier == "showRecipePhoto" {
        if shareEnabled {
            return false
        }
    }

    return true
}
```

READY TO TEST YOUR APP

Great! Now compile and run the app. Tap the Share button, select a few recipe photos and tap the Upload button to share them on Facebook. If the app can't bring up the Facebook composer, you probably forgot to set up your Facebook account in the Simulator. Go back to the home screen, select Settings > Facebook. Sign in with your Facebook account. Once configured, re-run the app and try it out again.

For your reference, you can download the Xcode project from https://www.dropbox.com/s/nr478zimu0l50q7/CollectionViewSelection.zip?dl=0.