18

BUILDING GRID LAYOUT USING COLLECTION VIEW

BUILDING GRID LAYOUT USING COLLECTION VIEW

If you have no idea about what grid-like layout is, just take a look at the built-in Photos app. The app presents photos in grid format. Before Apple introduced UICollectionView, you had to write a lot of code or make use of third-party libraries to build a similar layout. UICollectionView, in my opinion, is one of the most spectacular APIs in the iOS SDK. It not only simplifies the way you arrange visual elements in a grid layout, it even lets developers customize the layout (e.g. circular, cover flow style layout) without changing the data.

In this chapter, we will build a simple app to display a collection of recipe photos in grid layout. Here is what you're going to learn:

- Introduction to UICollectionView
- How to Use UICollectionView to build a simple Grid-based layout
- Customizing the Collection Cell Background

Let's get started.

GETTING STARTED WITH UICOLLECTIONVIEW AND UICOLLECTIONVIEWCONTROLLER

UICollectionView operates similar to UITableView. While UITableView manages a collection of data items and displays them on screen in a single-column layout, the UICollectionView class offers developers the flexibility to present items using customizable layouts. You can present items in multi-column grids, tiled layout, circular layout, etc.

By default, the SDK comes with a UICollectionViewFlowLayout class that organizes items into a grid with optional header and footer views for each section. Later, we'll use the layout class to build the demo app.

The UICollectionView is composed of several components:

Cells – instances of a UICollectionViewCell. Like UITableViewCell, a cell represents a single item in the data collection. The cells are the main elements organized by the associated layout. If a UICollectionViewFlowLayout is used, the cells are arranged in a grid-like format.

Supplementary views – Optional. It's usually used for implementing the header or footer views of sections.

Decoration views – think of it as another type of supplementary view but for decoration purpose only. The decoration view is unrelated to the data collection. We simply create decoration views to enhance the visual appearance of the collection view.

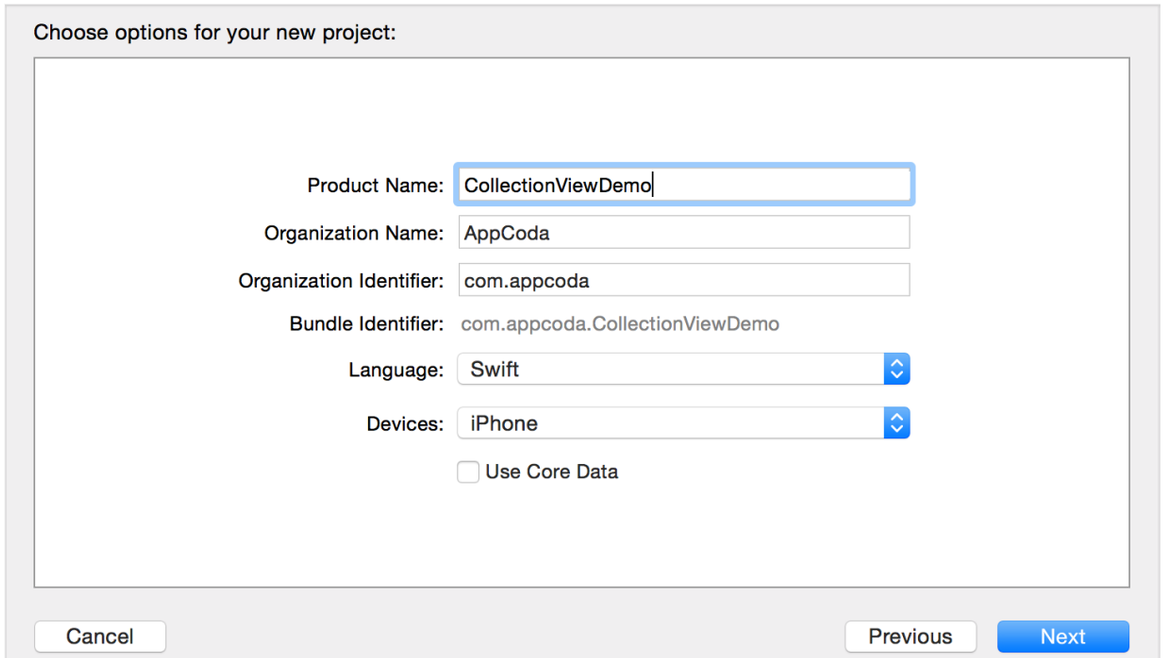
Like the table view, you have two ways to implement a collection view. You can add a collection view (UICollectionView) to your user interface and provide an object that conforms to the UICollectionViewDataSource protocol. The object is responsible for providing and managing the data associated with the collection view.

Alternatively, you can add a collection view controller from the Object Library to your storyboard. The collection view controller has a view controller with a collection view built-in and provides a default implementation of the UICollectionViewDataSource protocol.

For the demo project, we will use the latter approach.

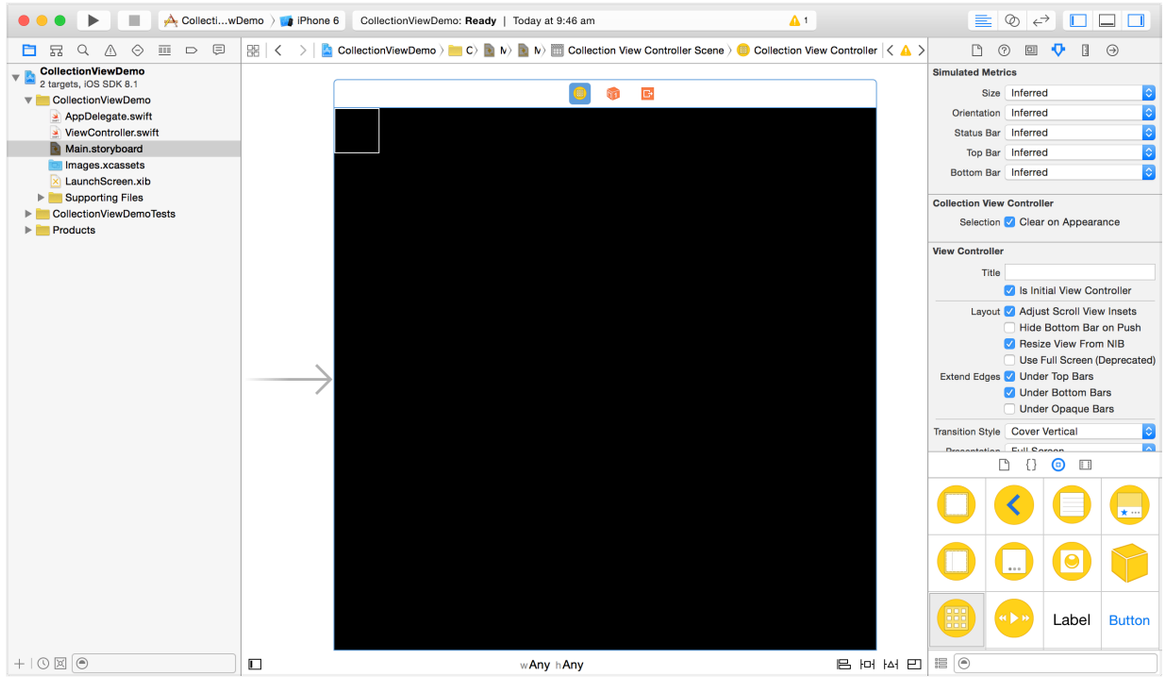
CREATING A NEW PROJECT

First, fire up Xcode and create a new project using the Single View Application template. Name the project “CollectionViewDemo”, set the device to iPhone and make sure you select Swift for the programming language.

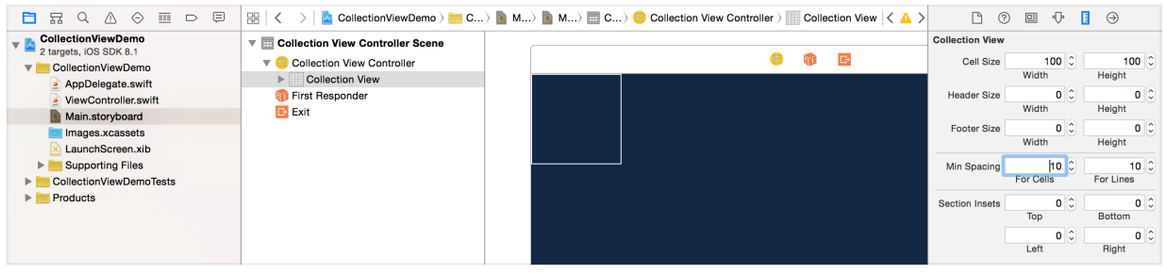


Once you’ve created the project, open main.Storyboard in the project navigator. Delete the default view controller and drag a Collection View Controller from the Object Library to the storyboard. The controller already has a collection view built-in. You should see a collection view cell in the controller, which is similar to the prototype cell of a table view.

Under the Attributes inspector, set the collection view controller as the initial view controller.



Next, open the Document Outline and select the collection view. Under the Size inspector, change the width and height of the cell to 100 points. Also change the min spacing of both “for cells” and “for lines” to 10 points.

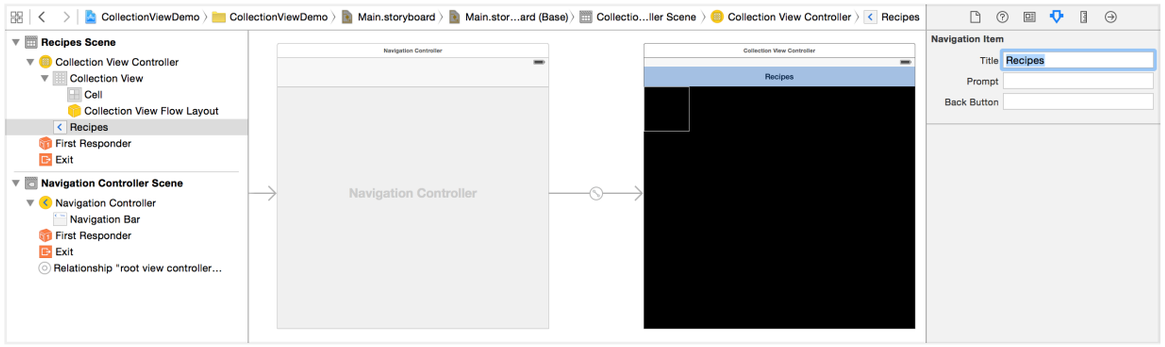


The “for cells” value defines the minimum spacing between items in the same row, while the “for lines” value defines the minimum spacing between successive rows.

Next, select the collection view cell and set the identifier to “Cell” in the Attribute Inspector. This looks familiar, right?

Add an image view to the cell. Xcode automatically resizes the image view and makes it fit to the cell.

Lastly, embed the collection view controller in a navigation controller. Go up to the Xcode menu, select Editor > Embed In > Navigation Controller. Set the title of the navigation bar to Recipes.

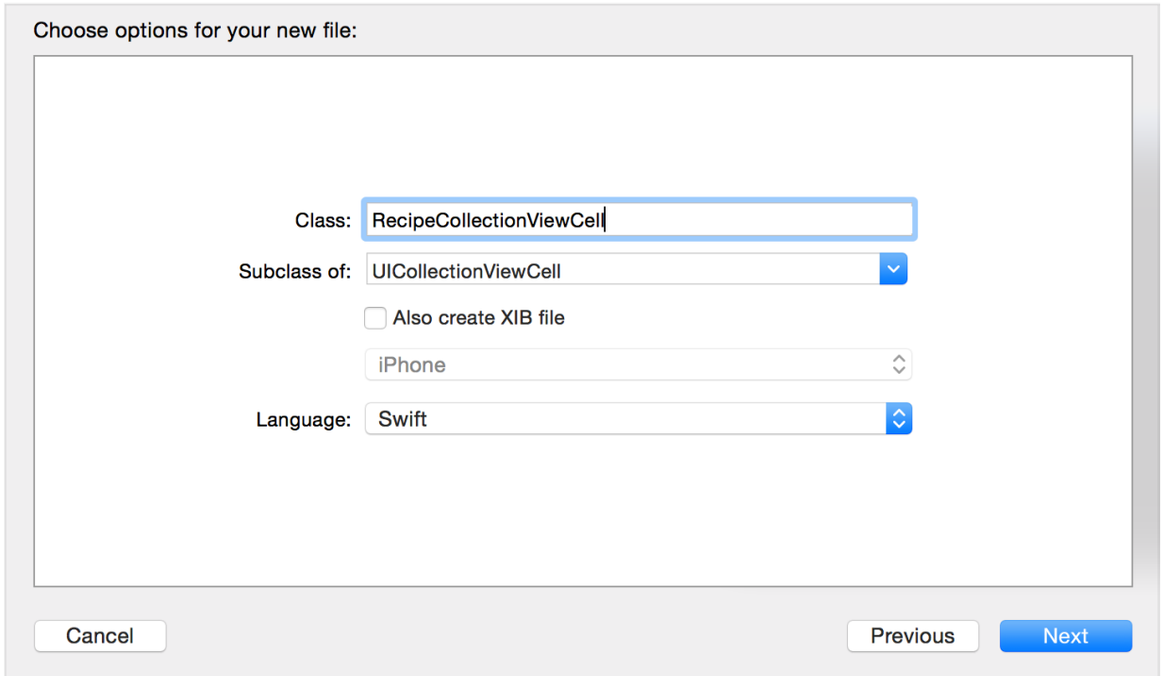


That's it. We have completed the user interface design. The next step is to create the custom classes for the collection view controller and the collection view cell.

CREATING CUSTOM CLASSES

In the project navigator, delete ViewController.swift file that was generated by Xcode. We do not need it because we will create our own classes.

Right click the CollectionViewDemo folder and select "New File...". Create a new class using the Cocoa Touch Class template. Name the class RecipeCollectionViewCell and set the subclass to UICollectionViewCell.

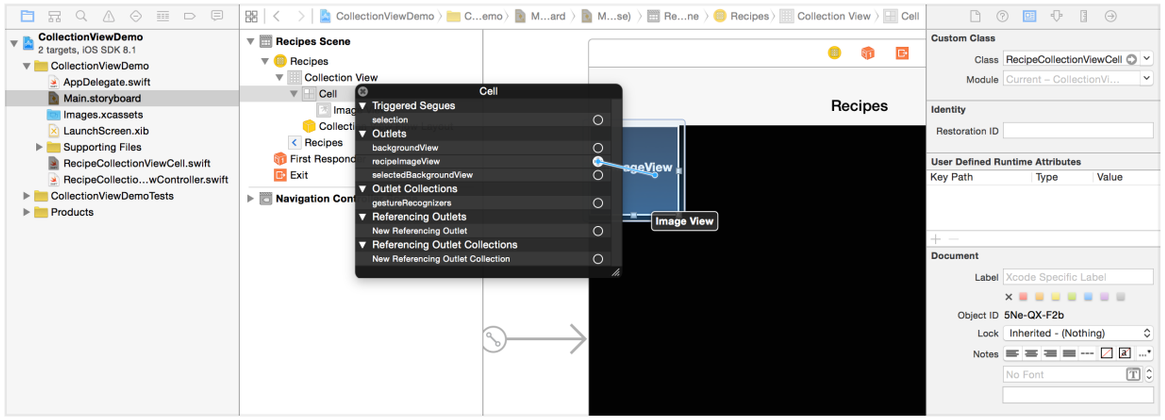


Repeat the process to create another class. Name the new class RecipeCollectionViewController and set the subclass to UICollectionViewController.

Now open RecipeCollectionViewCell.swift file and insert the following line of code to declare an outlet variable for the image view. Your class should look like this:

```
class RecipeCollectionViewCell: UICollectionViewCell {
    @IBOutlet weak var recipeImageView:UIImageView!
}
```

Go back to storyboard and select the collection view cell. Under the Identity inspector, change the custom class to RecipeCollectionViewCell. Then right click the cell and connect the outlet variable with the image view.



Next, select the collection view controller (Recipes). Under the Identity inspector, set the custom class to RecipeCollectionViewController.

IMPLEMENTING THE COLLECTION VIEW CONTROLLER

As mentioned before, UICollectionView operates in similar to the UITableView. To populate data in a table view, all you have to do is implement two methods defined in the UITableViewDataSource protocol. Like the UITableView, the UICollectionViewDataSource protocol defines a number of data source methods to interact with the collection view. You have to implement at least two methods:

- func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int
- func collectionView(_ collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell

First, download this image pack (https://www.dropbox.com/s/971c43jqbu9w4uj/RecipePhotoImagePack.zip?dl=0), unzip it and add all the images to the image asset.

Declare a recipeImages array in the RecipeCollectionViewController class:

```
var recipeImages = ["angry_birds_cake", "creme_brelee",
"egg_benedict", "full_breakfast", "green_tea",
"ham_and_cheese_panini", "ham_and_egg_sandwich", "hamburger",
"instant_noodle_with_egg.jpg", "japanese_noodle_with_pork",
"mushroom_risotto", "noodle_with_bbq_pork", "starbucks_coffee",
"thai_shrimp_cake", "vegetable_curry", "white_chocolate_donut"]
```

By default, Xcode 6 generates a statement in the *viewDidLoad* method to register a collection view cell for reuse purpose. Since we already use a prototype cell in storyboard, this line of code is no longer required. Remove it from the *viewDidLoad* method:

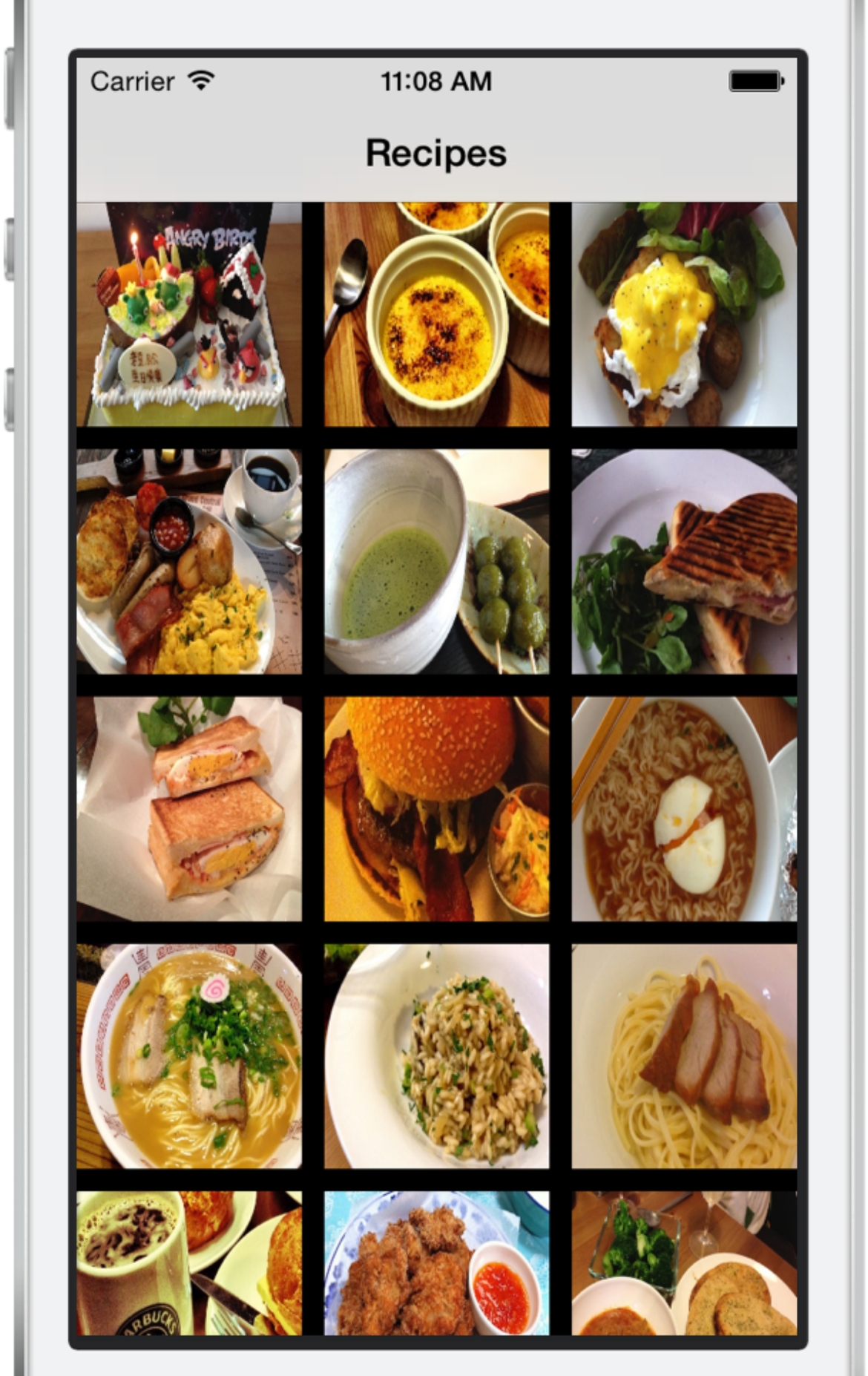
```
self.collectionView.registerClass(UICollectionViewCell.self,
forCellWithReuseIdentifier: reuseIdentifier)
```

Similar to what you have done when implementing a table view, update these data source methods of the UICollectionViewDataSource protocol to the following:

```
override func numberOfSectionsInCollectionView(collectionView:
UICollectionView) -> Int {
    return 1
}

override func collectionView(collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {
    return recipeImages.count
}

override func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell
{
```




```
    let cell =
collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifi
er, forIndexPath: indexPath) as! RecipeCollectionViewCell

    // Configure the cell
    cell.recipeImageView.image = UIImage(named:
recipeImages[indexPath.row])

    return cell
}
```

The *numberOfSectionsInCollectionView* method returns the total number of sections. In this case, we only have one section in the collection view. The *numberOfItemsInSection* method returns the number of recipe images.

The *cellForItemAtIndexPath* method manages the data for the collection view cells. We first define a cell identifier and then request the collectionView to dequeue a reusable cell using that reuse identifier. The *dequeueReusableCellWithReuseIdentifier* method will either automatically create a cell or return a cell from the re-use queue. The cell returned is downcast to a type of RecipeCollectionViewCell.

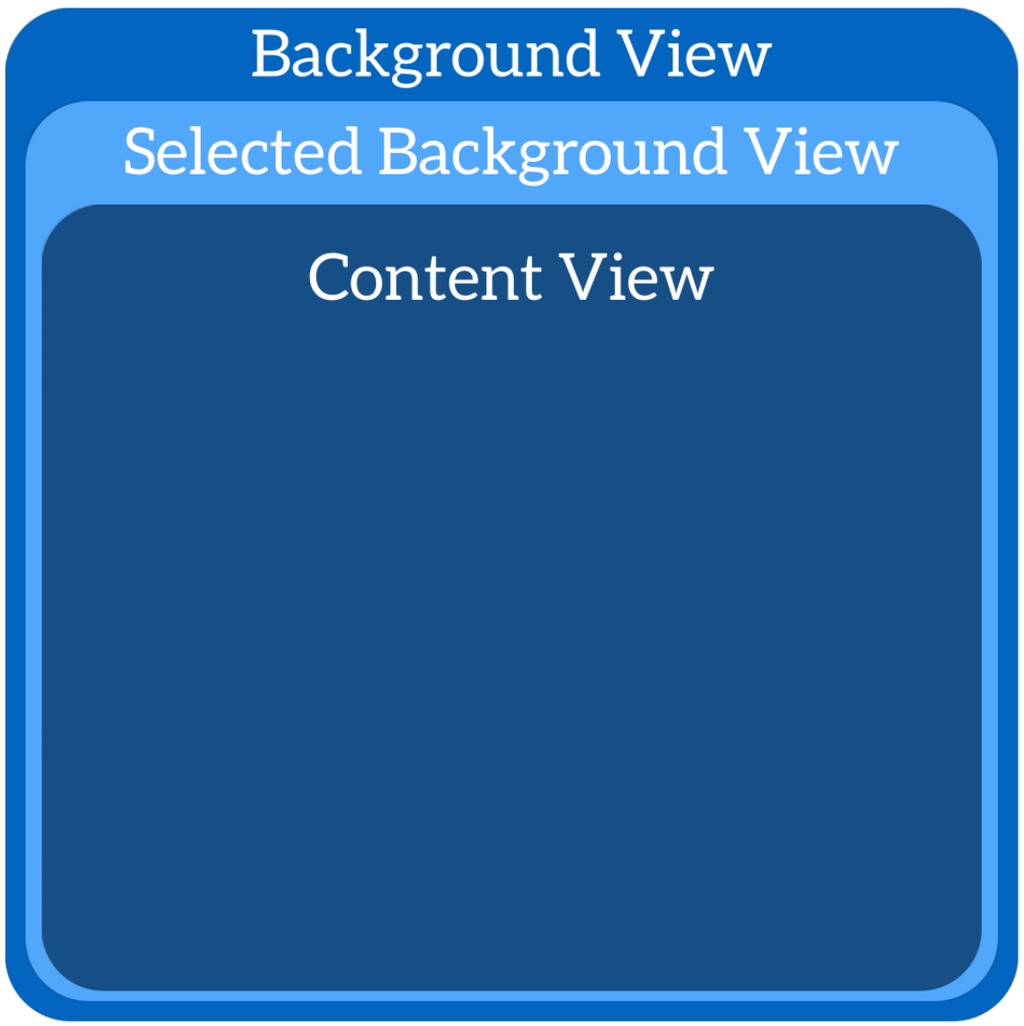
Finally, we get the corresponding recipe image and assign it to the image view to display.

Now compile and run the app. You should have a grid-based photo app.

CUSTOMIZING THE COLLECTION CELL BACKGROUND

Cool, right? With a few lines of code, you can create a grid-based photo app. What if you want to add a picture frame to the photos? Like other UI elements, UICollectionViewCell lets developers easily customize its background.

A collection view cell is comprised of three different views including background, selected background and content view:



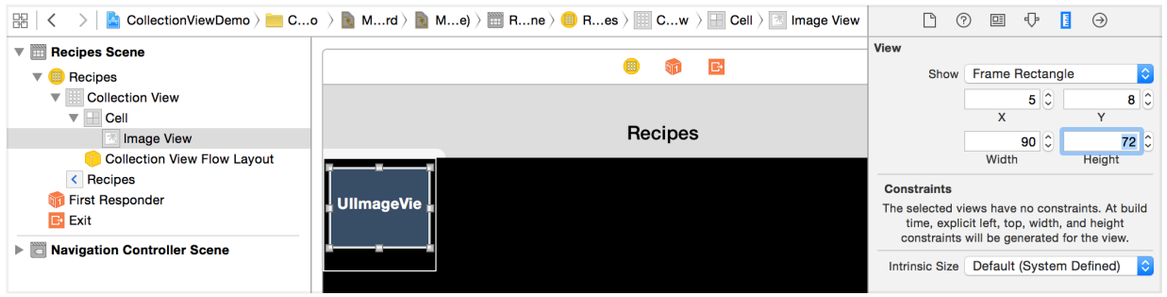
Background View – background view of the cell

Selected Background View – the background view when the cell is selected. When the user selects the cell, this selected background view will be layered above the background view.

Content View – obviously, it's the cell content.

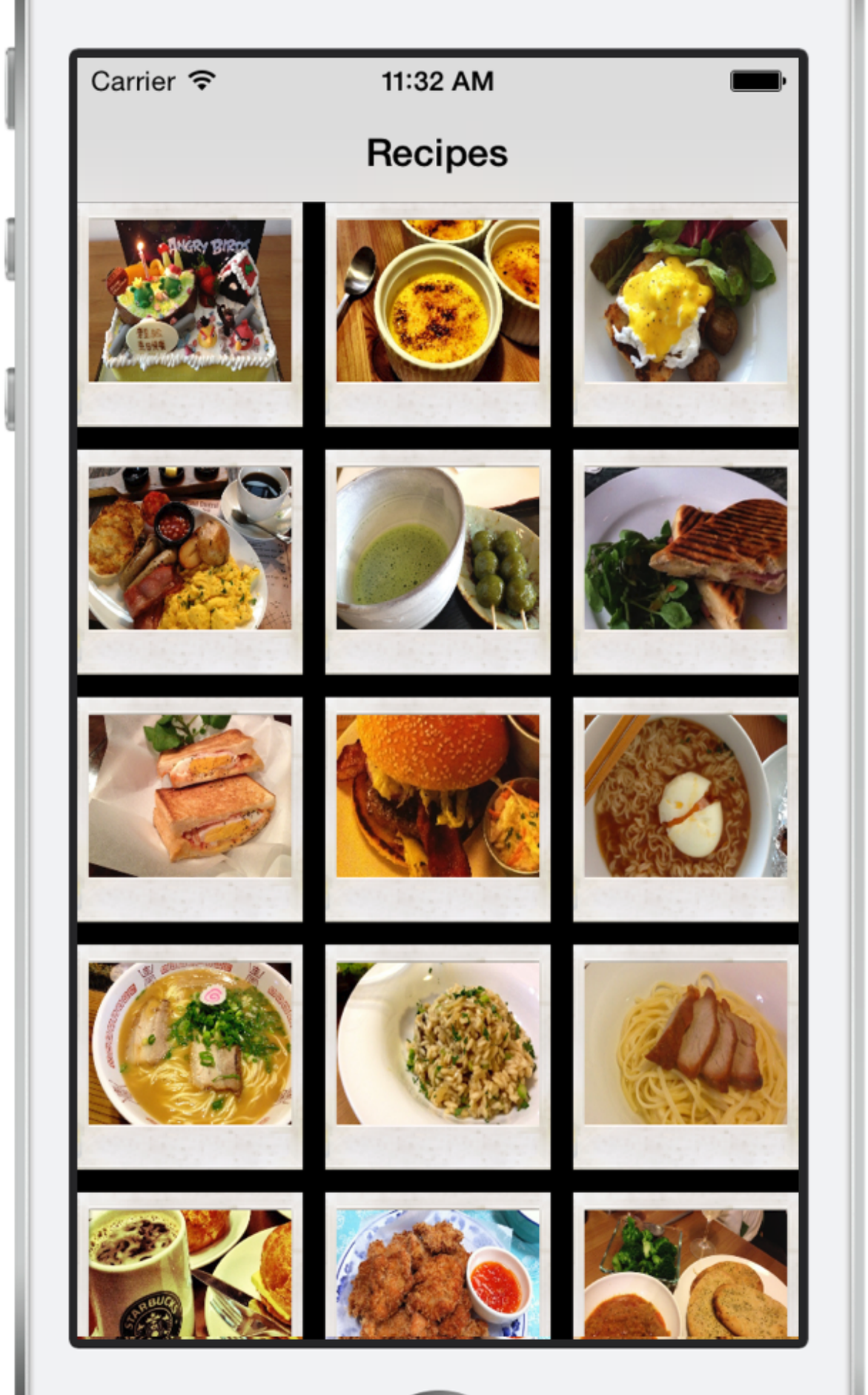
We have used the content view to display the recipe image. What we are going to do is use the background view to display a picture frame. In the image pack you downloaded earlier, it includes a file named photo_frame.png, which is the picture frame. The size of the frame is 100 by 100 pixel.

In order to frame the recipe photo, we'll first resize the image view of the cell and change its position. Go to the Main.storyboard and select the image view. Under the Size inspector, set X to 5 and Y to 8. The width and height should be changed to 90 and 72 pixels respectively.



Next, go back to the RecipeCollectionViewController.swift file. In the *cellForItemAtIndexPath* method, insert the following line of code before "return cell":

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



We simply load the photo frame image and set it as the background view of the collection view cell. Now compile and run the app again. Your app now displays a photo frame for each of the cell item.

For your reference, you can download the Xcode project from https://www.dropbox.com/s/o1anhhgdxmoxz1p/CollectionViewDemo.zip?dl=0.