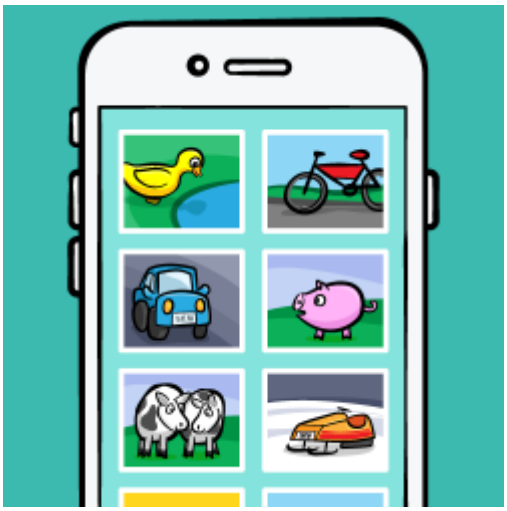
raywenderlich.com

UICollectionView Tutorial: Getting Started

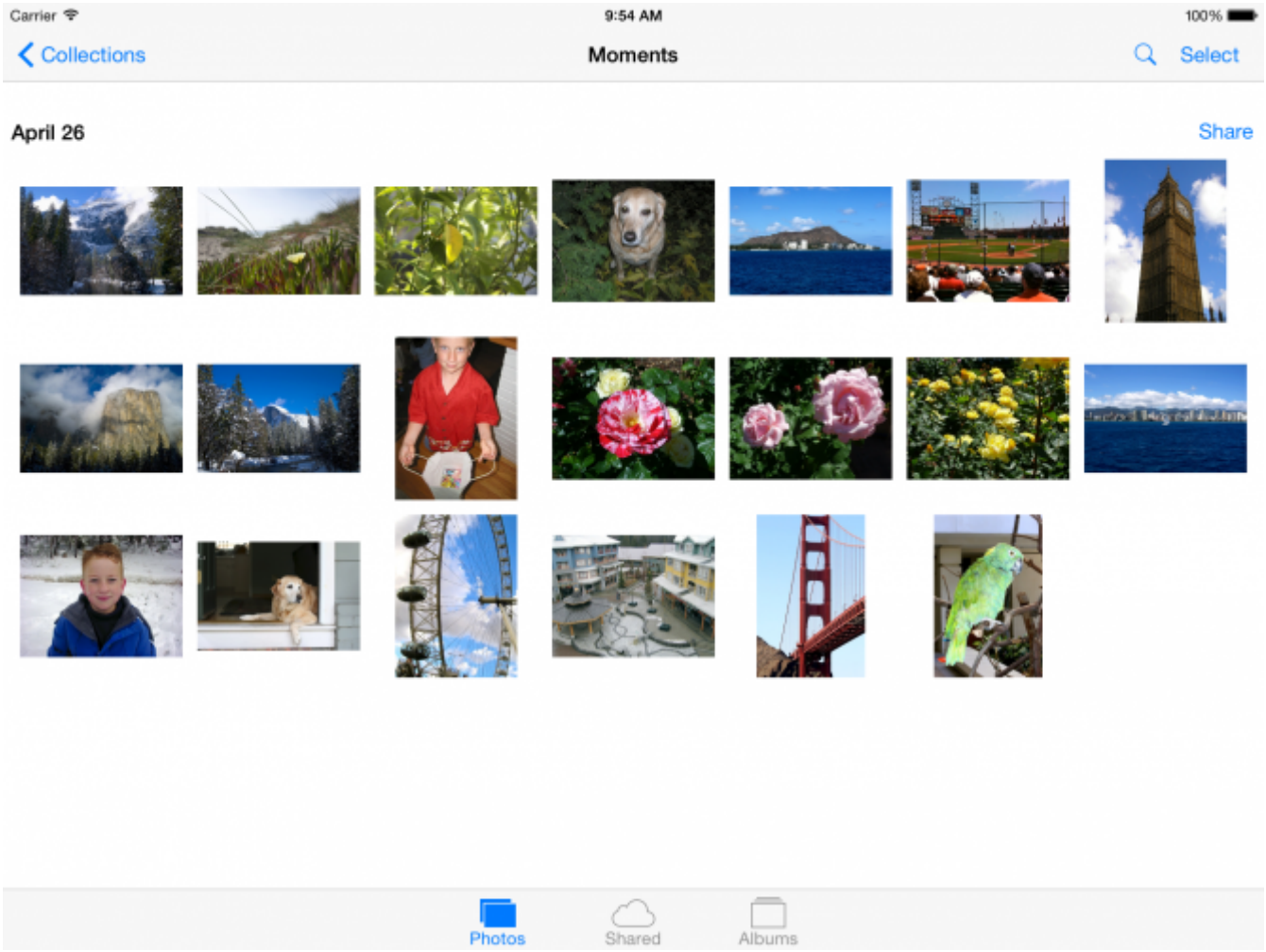
Bradley Johnson on September 5, 2016

Note: This tutorial has been updated for Swift 3, iOS 10, and Xcode 8.

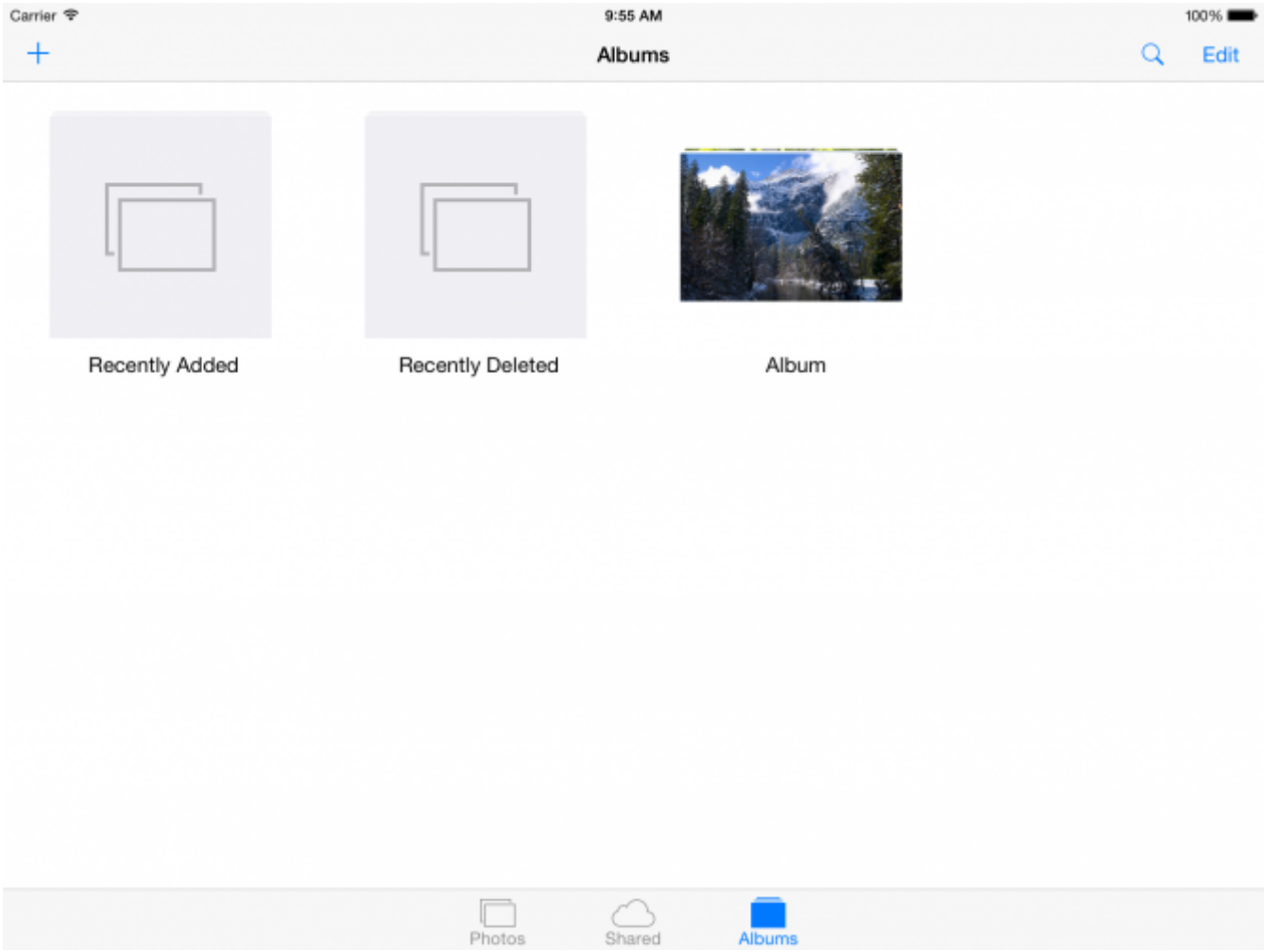
The iOS Photos app has a stylish way of displaying photos via a multitude of layouts. You can view your photos in a nice grid view:



Create your own grid-based photo browsing app with collection views!



Or you can view your albums as stacks:



You can even transition between the two layouts with a cool pinch gesture. “Wow, I want that in my app!”, you may think.

`UICollectionView` makes adding your own custom layouts and layout transitions (like those in the Photos app) simple to build.

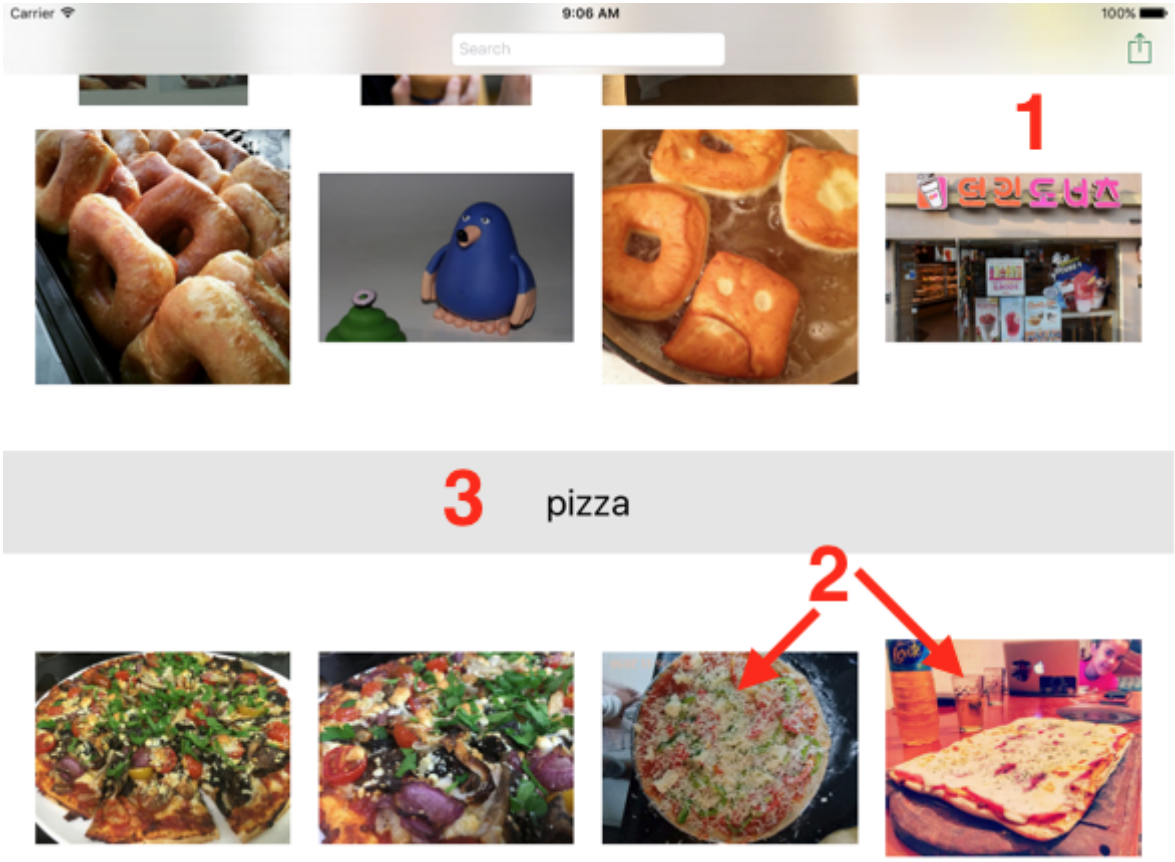
You’re by no means limited to stacks and grids, because collection views are extremely customizable. You can use them to make circle layouts, cover-flow style layouts, Pulse news style layouts – almost anything you can dream up!

The good news is, if you’re familiar with `UITableView`, you’ll have no problem picking up collection views – using them is very similar to the table view data source and delegate pattern.

In this tutorial, you’ll get hands-on experience with `UICollectionView` by creating your own grid-based photo browsing app. By the time you are done with this tutorial, you will know the basics of using collection views and will be ready to start using this amazing technology in your apps!

Anatomy of a UICollectionView

Let’s go right to an example of the finished project. `UICollectionView` contains several key components, as you can see below:



Take a look at these components one-by-one:

1. `UICollectionView` – the main view in which the content is displayed, similar to a `UITableView`. Like a table view, a collection view is a `UIScrollView` subclass.
2. `UICollectionViewCell` – similar to a `UITableViewCell` in a table view. These cells make up the content of the view and are added as subviews to the collection view. Cells can be created programmatically or inside Interface Builder.
3. `Supplementary Views` – if you have extra information you need to display that shouldn't be in the cells but still somewhere within the collection view, you should use supplementary views. These are commonly used for headers or footers.

> **Note**: Collection views can also have **Decoration Views** – if you want to add some extra views to enhance the appearance of the collection view (but don't really contain useful data), you should use decoration views. Background images or other visual embellishments are good examples of decoration views. You won't be using decoration views in this tutorial as it requires you to write a custom layout class.

In addition to the above visual components, a collection view has a layout object which is responsible for the size, position and several other attributes of the content. Layout objects are subclasses of `UICollectionViewLayout`. Layouts can be swapped out during runtime and the collection view can even automatically animate switching from one layout to another!

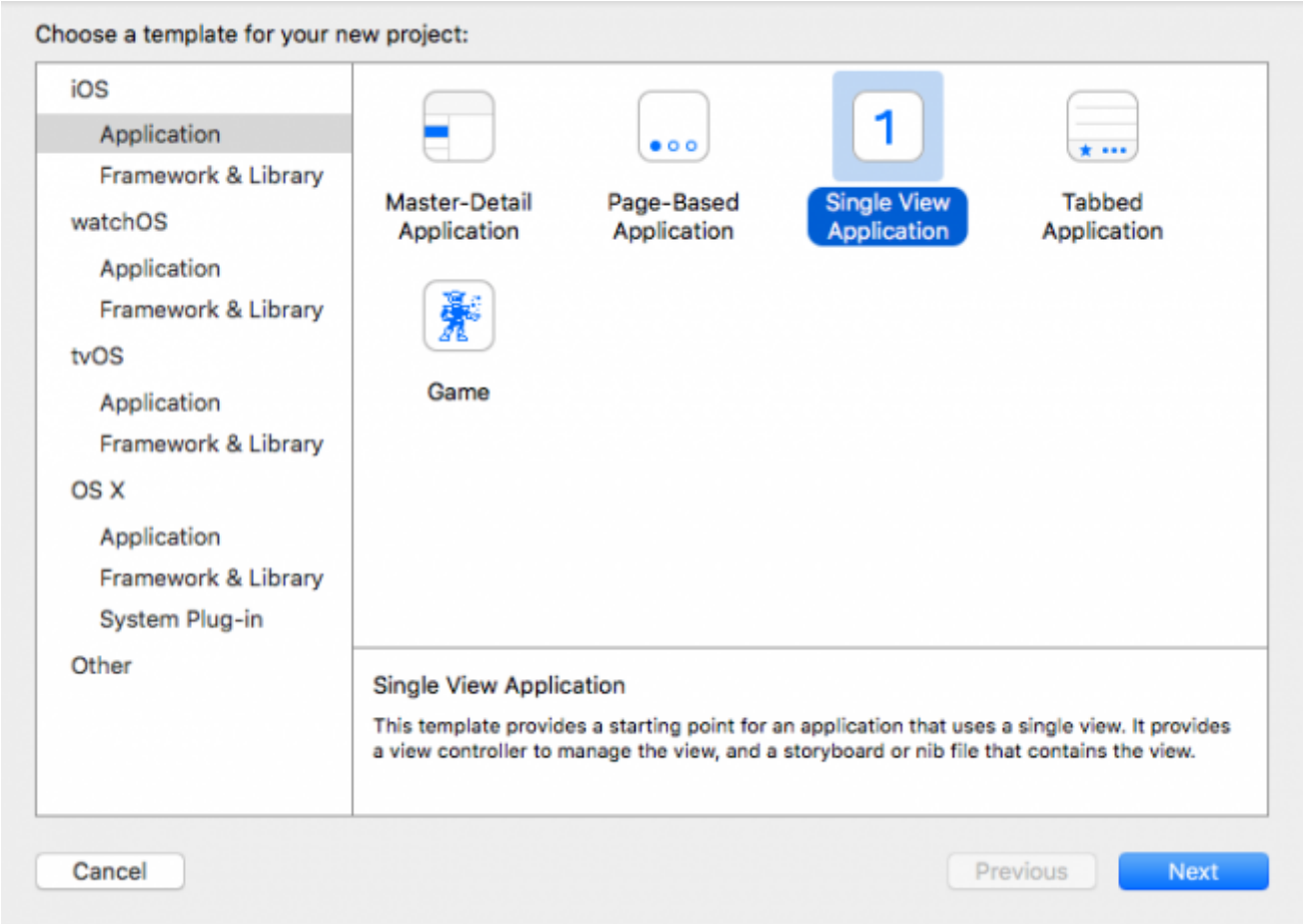
You can subclass `UICollectionViewLayout` to create your own custom layouts, but Apple has graciously provided developers with a basic "flow-based" layout called `UICollectionViewFlowLayout`. It lays elements out one after another based on their size, quite like a grid view. You can use this layout class out of the box, or subclass it to get some interesting behavior and visual effects.

You will learn more about these elements in-depth throughout this tutorial and the next. But for now, it's time for you to get your hands into the mix with a project!

Introducing FlickrSearch

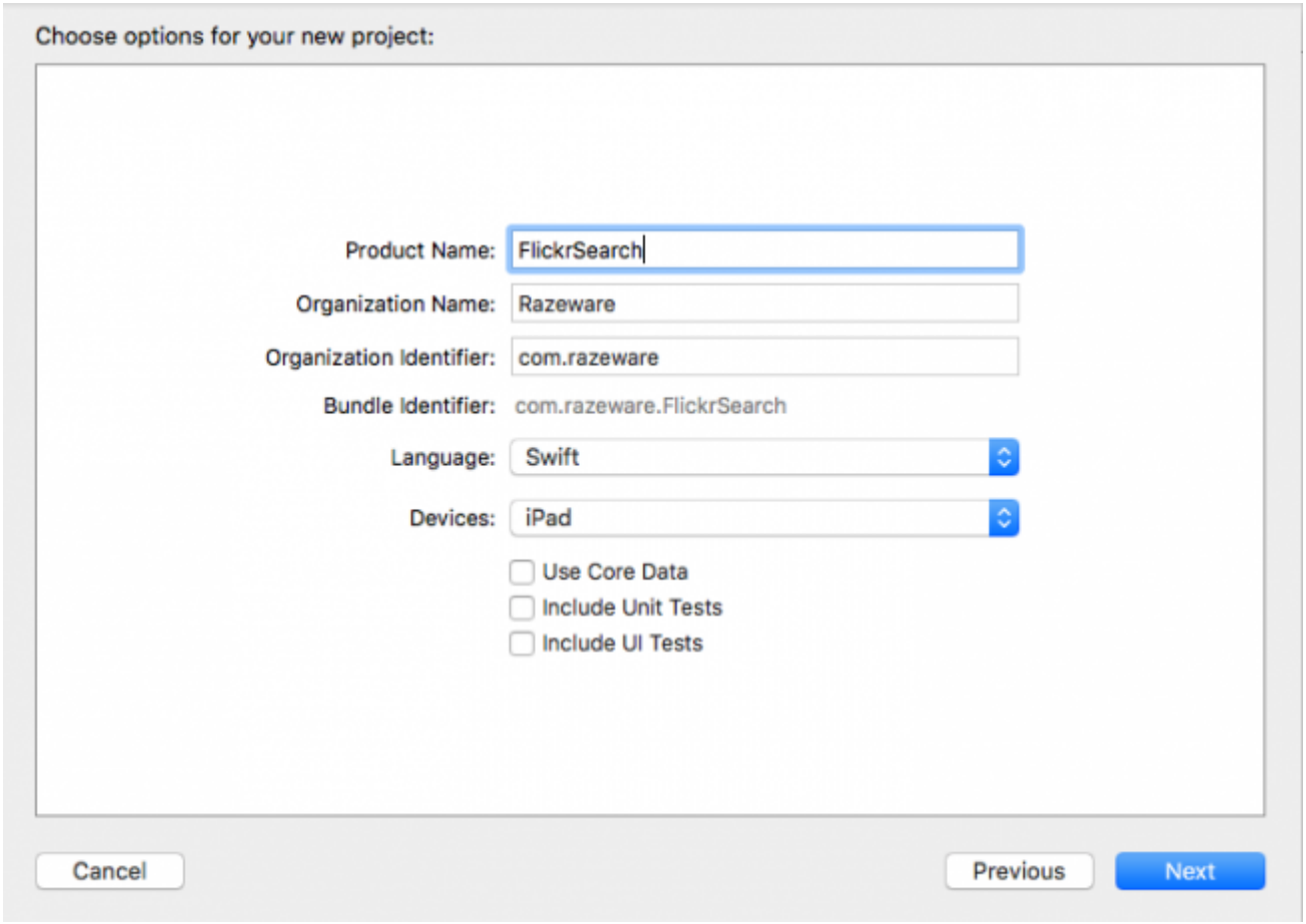
In the rest of this tutorial, you are going to create a cool photo browsing app called **FlickrSearch**. It will allow you to search for a term on the popular photo sharing site Flickr, and it will download and display any matching photos in a grid view, as you saw in the screenshot earlier.

Ready to get started? Fire up Xcode and go to **File\New\Project...** and select the **iOS\Application\Single View Application** template.



This template will provide you with a simple UIViewController and storyboard to start out with, and nothing more. It's a good "almost from scratch" point to start from.

Click **Next** to fill out the information about the application. Set the Product Name to **FlickrSearch**, the device type to **iPad** and the language to **Swift**. Click **Next** to select the project location, and then click **Create**.



The view controller subclass and storyboard that come with the single view application template aren't any use to you – you're going to use a **UICollectionViewController**, which, like a **UITableViewController**, is a specialized view controller subclass designed to host a collection view. Delete **ViewController.swift** and remove the empty view controller from **Main.storyboard**. Now you've got a *really* blank slate :].

Open **AppDelegate.swift** and add a constant to hold a delightful shade I call Wenderlich Green. Add the following underneath the **import UIKit** line:

```
let themeColor = UIColor(red: 0.01, green: 0.41, blue: 0.22, alpha: 1.0)
```

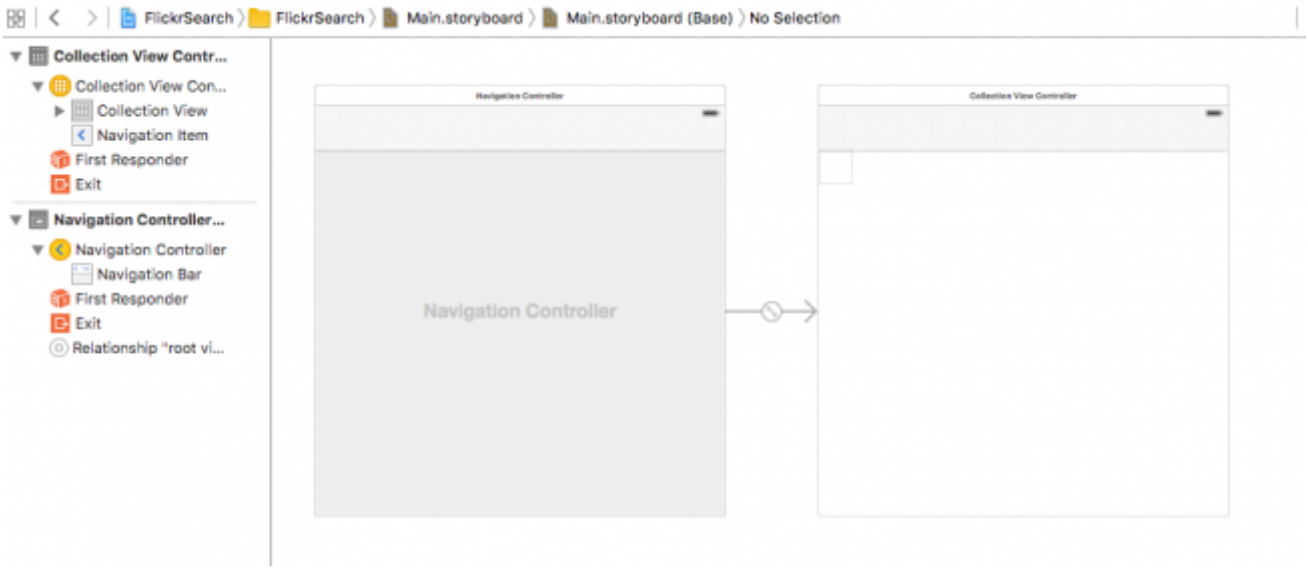
Use Wenderlich Green as a theme color for the whole app. Update **application(_:didFinishLaunchingWithOptions:)** to the following:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
  launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

  window?.tintColor = themeColor
  return true
}
```

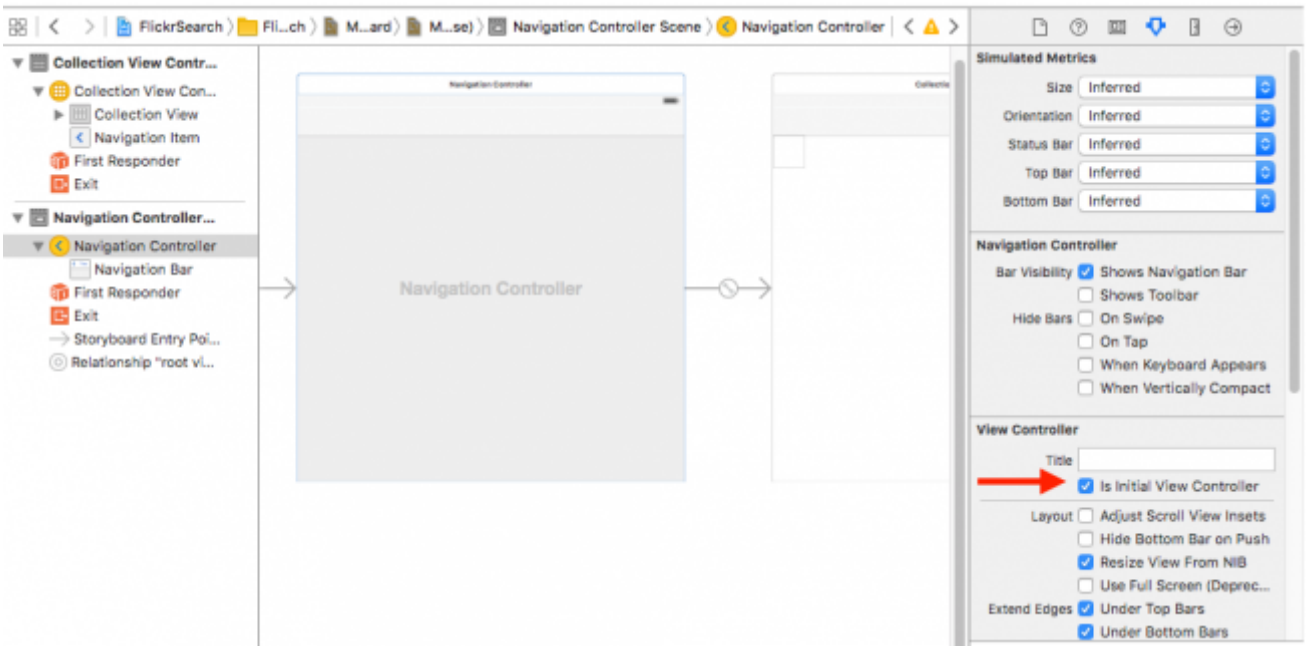
Starting your collection

Open **Main.storyboard** and drag in a **Collection View Controller**. Go to **Editor\Embed in\Navigation Controller** to create a navigation controller and automatically set the collection view controller as the root.

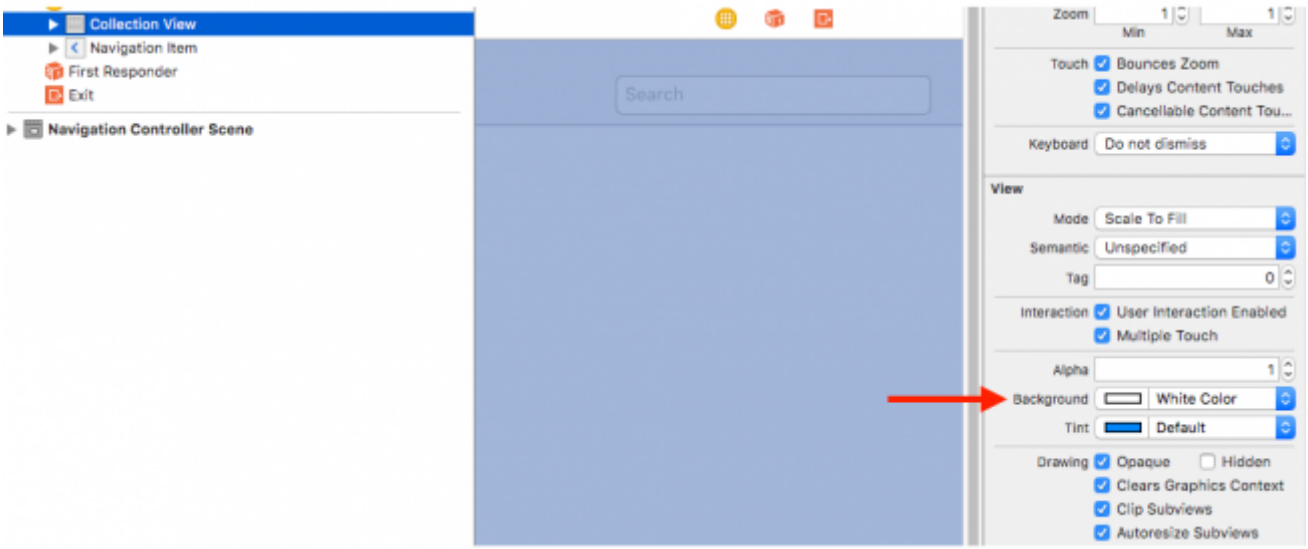
You should now have a layout like this in the storyboard:



Next, select the **Navigation Controller** you installed and make it the initial view controller in the **Attributes Inspector**:



Next, focusing on the collection view controller, select the `UICollectionView` inside and change the background color to **White Color**:

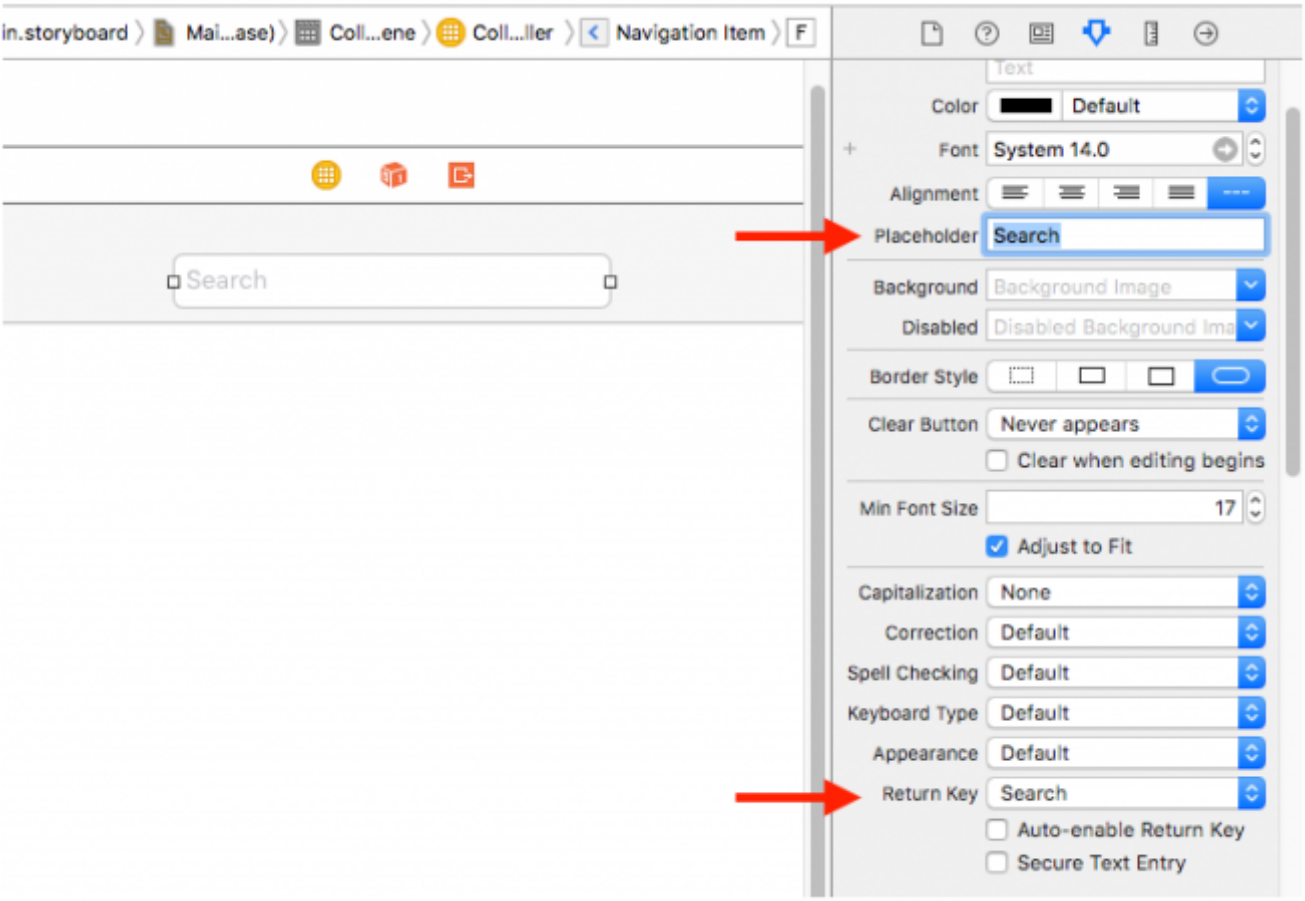


Note: Wondering what the **Scroll Direction** property does? This property is specific to `UICollectionViewFlowLayout`, and defaults to **Vertical**. A vertical flow layout means the layout class will place items from left to right across the top of the view until it reaches the view's right edge, at which point it moves down to the next line. If there are too many elements to fit in the view at once, the user will be able to scroll vertically to see more.

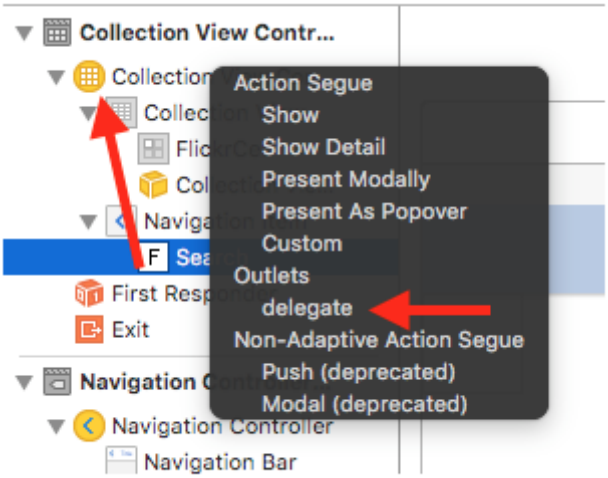
Conversely, a horizontal flow layout places items from top to bottom across the left edge of the view until it reaches the bottom edge. Users would scroll horizontally to see items that don't fit on the screen. In this tutorial, you'll stick with the more common **Vertical** collection view.

Select the single cell in the collection view and set the **Reuse Identifier** to **FlickrCell** using the attributes inspector. This should also be familar from table views – the data source will use this identifier to dequeue or create new cells.

Drag in a text field to the center of the navigation bar above the collection view. This will be where the user enters their search text. Set the **Placeholder Text** of the search field to **Search** and the **Return Key** to **Search** in the attributes inspector:



Next, control-drag from the text field to the collection view controller and choose the **delegate** outlet:



UICollectionViewController does a lot, but you generally need to make a subclass. Do this now. Go to **File\New\File...**, choose **iOS\Source\Cocoa Touch Class** template and click **Next**. Name the new class **FlickrPhotosViewController**, making it a subclass of **UICollectionViewController**. There's a lot of code added by the template, but the best way to understand what this class does is to start from scratch. Open **FlickrPhotosViewController.swift** replace the contents of the file with the below code:

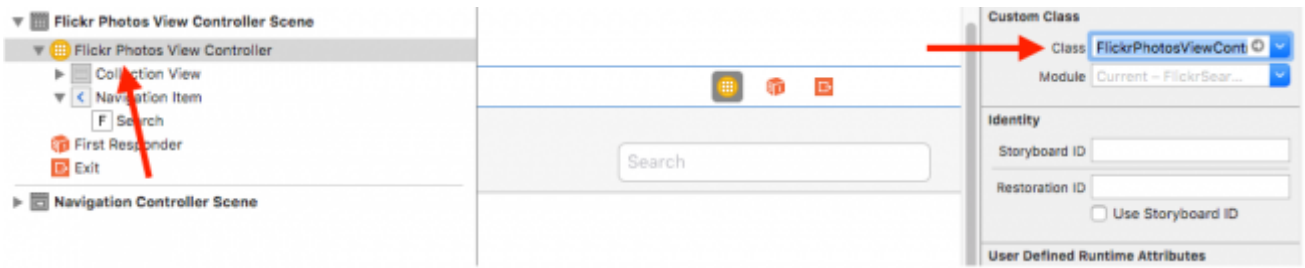
```
import UIKit

final class FlickrPhotosViewController: UICollectionViewController {

  // MARK: - Properties
  fileprivate let reuseIdentifier = "FlickrCell"
}
```

Next, add a constant for section insets (which you'll use later) right below the **reuseIdentifier** constant:

```
fileprivate let sectionInsets = UIEdgeInsets(top: 50.0, left: 20.0, bottom: 50.0, right: 20.0)
```

You'll be filling in the rest of the gaps as you progress through the tutorial.

Go back to **Main.storyboard**, select the collection view controller and in the identity inspector, set the **Class** to **FlickrPhotosViewController** to match your new class:



Fetching Flickr Photos

You first task for this section is to say the section title ten times fast. OK, just kidding.

Flickr is a wonderful image sharing service that has a publicly accessible and dead- simple API for developers to use. With the API you can search for photos, add photos, comment on photos, and much more.

To use the Flickr API, you need an API key. If you are doing a real project, I recommend you sign up for one here: http://www.flickr.com/services/api/keys/apply/.

However, for test projects like this, Flickr has a sample key they rotate out every so often that you can use without having to sign up. Simply perform any search at: http://www.flickr.com/services/api/explore/?method=flickr.photos.search and copy the API key out of the URL at the bottom – it follows the "&api_key=" all the way to the next "&". Paste it somewhere in a text editor for later use.

For example, if the URL is:

```
http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=6593783
efea8e7f6dfc6b70bc03d2afb&format=rest&api_sig=f24f4e98063a9b8ecc8b522b238 d5e2f
```

Then the API key is: 6593783efea8e7f6dfc6b70bc03d2afb

> **Note:** If you use the sample API key, note that it is changed nearly every day. So if you're doing this tutorial over the course of several days, you might find that you have to get a new API key often. For this reason it might be easier to get an API key of your own from Flickr if you think you're going to spend several days on this project.

Since this tutorial is about **`UICollectionView`** and not the Flickr API, a set of classes has been created for you that abstracts the Flickr search code. You can download them here.

Unzip and drag the files into your project, making sure that the box **Copy items into destination group's folder (if needed)** is checked, and click Finish.

The file contains two classes and a struct:

- **`FlickrSearchResults`**: A struct which wraps up a search term and the results found for that search.
- **`FlickrPhoto`**: Data about a photo retrieved from Flickr – its thumbnail, image, and metadata information such as its ID. There are also some methods to build Flickr URLs and some size calculations. **`FlickrSearchResults`** contains an array of these objects.
- **`Flickr`**: Provides a simple block-based API to perform a search and return a **`FlickrSearchResult`**

Feel free to take a look at the code – it's pretty simple and might inspire you to make use of Flickr in your own projects!

Before you can search Flickr, you need to enter an API key. Open **Flickr.swift** and replace the value of **`apiKey`** with the API key you obtained earlier. It should look something like this:

```
let apiKey = "hh7ef5ce0a54b6f5b8fbc36865eb5b32"
```

When you're ready to go, move on to the next section – it's time to do a little prep work before hooking into Flickr.

Preparing Data Structures

You're going to build this project so that after each time you perform a search, it displays a new **section** in the collection view with the results (rather than simply replacing the previous section). In other words, if you search for *"ninjas"* and then *"pirates"*, there will be a section of ninjas and a section of pirates in the collection view. Talk about a recipe for disaster!

To accomplish this, you're going to need a data structure so you can keep the data for each section separate. An array of **`FlickrSearchResults`** will do the trick nicely.

Open **FlickrPhotosViewController.swift** and the following properties below the **sectionInsets** constant:

```
fileprivate var searches = [FlickrSearchResults]()
fileprivate let flickr = Flickr()
```

`searches` is an array that will keep track of all the searches made in the app, and **`flickr`** is a reference to the object that will do the searching for you. Next, add the following **private** extension to the bottom of the file:

```
// MARK: - Private
private extension FlickrPhotosViewController {
  func photoForIndexPath(indexPath: IndexPath) -> FlickrPhoto {
    return searches[(indexPath as NSIndexPath).section].searchResults[(indexPath as
NSIndexPath).row]
  }
}
```

`photoForIndexPath` is a convenience method that will get a specific photo related to an index path in your collection view. You're going to access a photo for a specific index path a lot, and you don't want to repeat code.

Getting Good Results

You are now ready to get your Flickr search on! You want to trigger a search when the user hits **Search** after typing in a query. You already connected the text field's delegate outlet to your collection view controller, now you can do something about it.

Open **FlickrPhotosViewController.swift** and add an extension to hold the text field delegate methods:

```
extension FlickrPhotosViewController : UITextFieldDelegate {
  func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    // 1
    let activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
    textField.addSubview(activityIndicator)
    activityIndicator.frame = textField.bounds
    activityIndicator.startAnimating()

    flickr.searchFlickrForTerm(textField.text!) {
      results, error in


      activityIndicator.removeFromSuperview()


      if let error = error {
        // 2
        print("Error searching : \(error)")
        return
      }

      if let results = results {
        // 3
        print("Found \(results.searchResults.count) matching \(results.searchTerm)")
        self.searches.insert(results, at: 0)

        // 4
        self.collectionView?.reloadData()
      }
    }

    textField.text = nil
    textField.resignFirstResponder()
    return true
  }
}
```

Here is an explanation of the code:

1. After adding an activity view, use the Flickr wrapper class I provided to search Flickr for photos that match the given search term asynchronously. When the search completes, the completion block will be called with a the result set of FlickrPhoto objects, and an error (if there was one).
2. Log any errors to the console. Obviously, in a production application you would want to display these errors to the user.
3. The results get logged and added to the front of the searches array
4. At this stage, you have new data and need to refresh the UI. You're using the `reloadData()` method, which works just like it does in a table view.

Go ahead and run your app. Perform a search in the text box, and you should see a log message in the console indicating the number of search results, similar to this:

```
Found 20 matching bananas
```

Note that the results are limited to 20 by the Flickr class to keep load times down.

Unfortunately, you don't see any photos in your collection view! Just like a table view, a collection view doesn't do much unless you implement the relevant data source and delegate methods.

Feeding the UICollectionView

As you probably already know, when you use a table view you have to set a data source and a delegate in order to provide the data to display and handle events (like row selection).

Similarly, when you use a collection view you have to set a data source and a delegate as well. Their roles are the following:

- The data source (`UICollectionViewDataSource`) returns information about the number of items in the collection view and their views.
- The delegate (`UICollectionViewDelegate`) is notified when events happen such as cells being selected, highlighted, or removed.

`UICollectionViewFlowLayout` also has a delegate protocol – `UICollectionViewDelegateFlowLayout`. It allows you to tweak the behaviour of the layout, configuring things like the cell spacing, scroll direction, and more.

In this section, you're going to implement the required `UICollectionViewDataSource` and `UICollectionViewDelegateFlowLayout` methods on your view controller, so you are all set up to work with your collection view. The `UICollectionViewDelegate` methods aren't needed for this part, but you'll be using them in part 2.

UICollectionViewDataSource

Open **FlickrPhotosViewController.swift**, add the following extension to the file for the `UICollectionViewDataSource` protocol:

```
// MARK: - UICollectionViewDataSource
extension FlickrPhotosViewController {
  //1
  override func numberOfSections(in collectionView: UICollectionView) -> Int {
    return searches.count
  }

  //2
  override func collectionView(_ collectionView: UICollectionView,
                               numberOfItemsInSection section: Int) -> Int {
    return searches[section].searchResults.count
  }

  //3
  override func collectionView(_ collectionView: UICollectionView,
                               cellForItemAt indexPath: IndexPath) -> UICollectionViewCell
{
    let cell = collectionView.dequeueReusableCell(withReuseIdentifier: reuseIdentifier,
                                                  for: indexPath) as!
FlickrPhotoCell
    cell.backgroundColor = UIColor.black
    // Configure the cell
    return cell
  }
}
```

These methods are pretty straightforward:

1. There's one search per section, so the number of sections is the count of the `searches` array.
2. The number of items in a section is the count of the `searchResults` array from the relevant `FlickrSearch` object.
3. This is a placeholder method just to return a blank cell – you'll be populating it later. Note that collection views *require* you to have registered a cell with a reuse identifier, or a runtime error will occur.

Build and run again, and perform a search. You should see 20 new cells, albeit looking a little dull at the moment:

UICollectionViewFlowLayoutDelegate

As I mentioned early in the section, every collection view has an associated layout. You're using the pre-made flow layout for this project, since it's nice and easy to use and gives you the grid-view style you're looking for.

Still in **FlickrPhotosViewController.swift**, add the following constant below your `flickr` constant:

```
fileprivate let itemsPerRow: CGFloat = 3
```

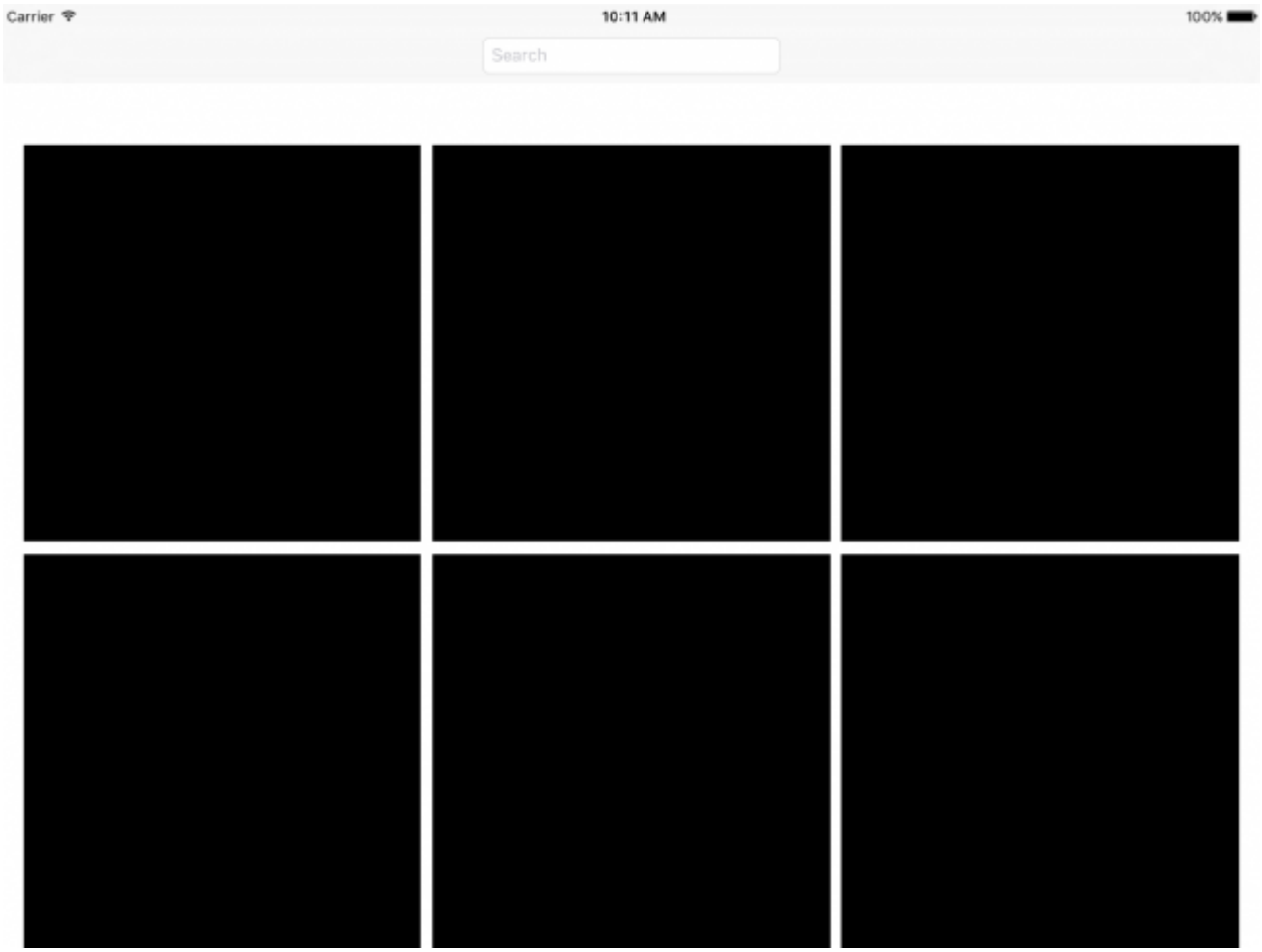
Next, add the `UICollectionViewDelegateFlowLayout` extension to allow the view controller to conform to the flow layout delegate protocol:

```
extension FlickrPhotosViewController : UICollectionViewDelegateFlowLayout {
  //1
  func collectionView(_ collectionView: UICollectionView,
                      layout collectionViewLayout: UICollectionViewLayout,
                             sizeForItemAt indexPath: IndexPath) -> CGSize {
    //2
    let paddingSpace = sectionInsets.left * (itemsPerRow + 1)
    let availableWidth = view.frame.width - paddingSpace
    let widthPerItem = availableWidth / itemsPerRow

    return CGSize(width: widthPerItem, height: widthPerItem)
  }

  //3
  func collectionView(_ collectionView: UICollectionView,
                      layout collectionViewLayout: UICollectionViewLayout,
                             insetForSectionAt section: Int) -> UIEdgeInsets {
    return sectionInsets
  }

  // 4
```

```
  func collectionView(_ collectionView: UICollectionView,
    layout collectionViewLayout: UICollectionViewLayout,
    minimumLineSpacingForSectionAt section: Int) -> CGFloat {
    return sectionInsets.left
  }
}
```

1. `collectionView(_:layout:sizeForItemAt:)` is responsible for telling the layout the size of a given cell
2. Here, you work out the total amount of space taken up by padding. There will be n + 1 evenly sized spaces, where n is the number of items in the row. The space size can be taken from the left section inset. Subtracting this from the view's width and dividing by the number of items in a row gives you the width for each item. You then return the size as a square
3. `collectionView(_:layout:insetForSectionAt:)` returns the spacing between the cells, headers, and footers. A constant is used to store the value.
4. This method controls the spacing between each line in the layout. You want this to match the padding at the left and right.

Build and run again, and perform a search. Behold! Black squares that are bigger than before!

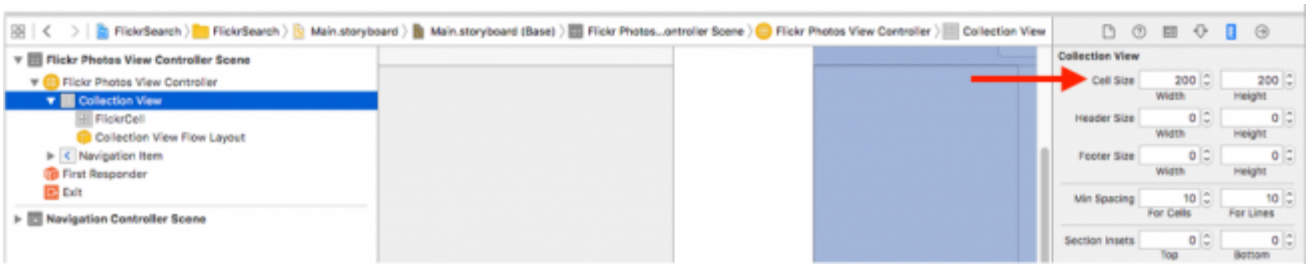


With this infrastructure in place, you are now ready to actually display some photos on screen!

Creating custom UICollectionViewCells

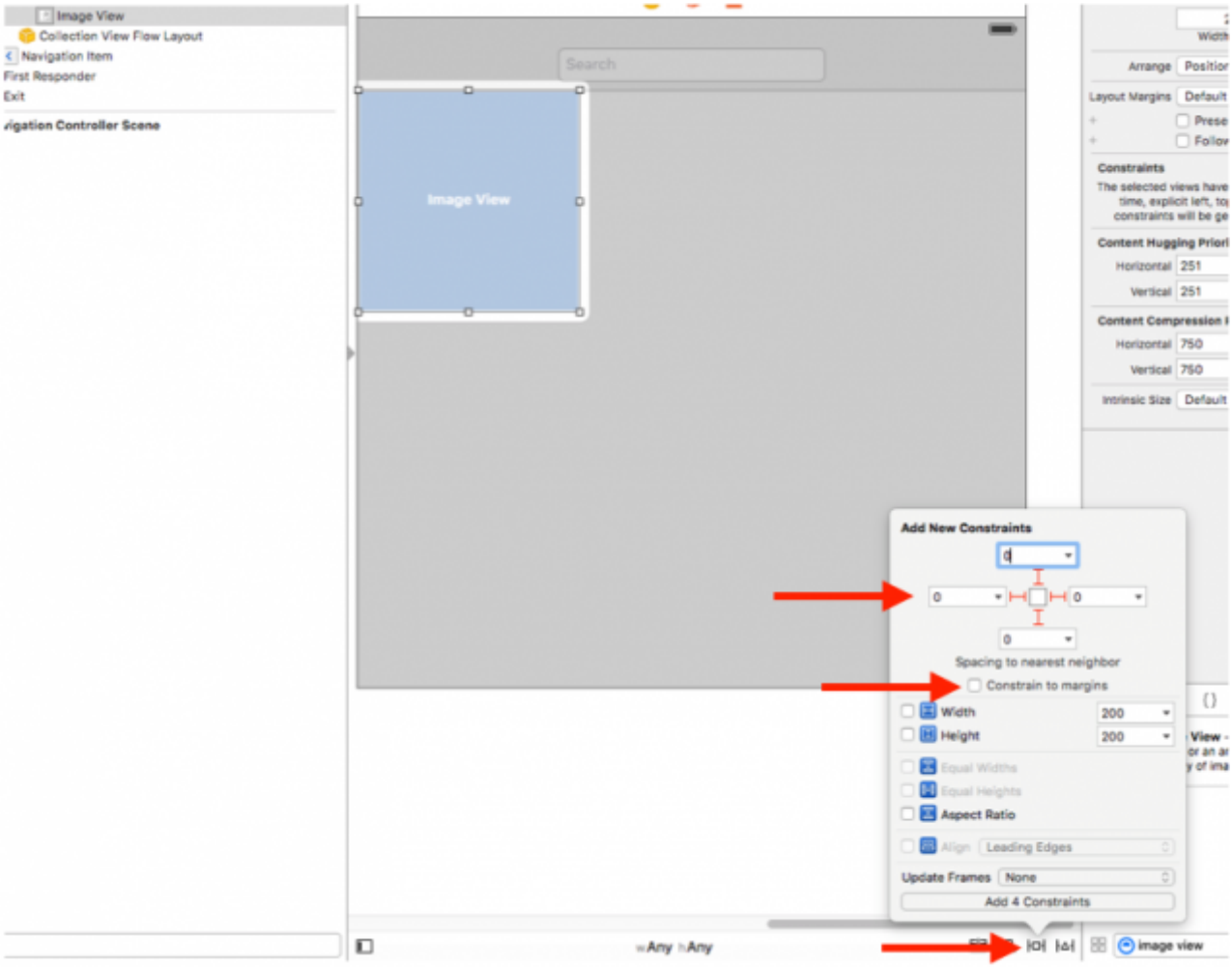
One of the great things about `UICollectionView` is that, like table views, it is easy to set up collection views visually in the Storyboard editor. You can drag and drop collection views into your view controller, and design the layout for your cells right from within the Storyboard editor! Let's see how it works.

Open **Main.storyboard** and select the collection view. Give yourself a bit of room to work by setting the cell size to 200×200 in the size inspector:

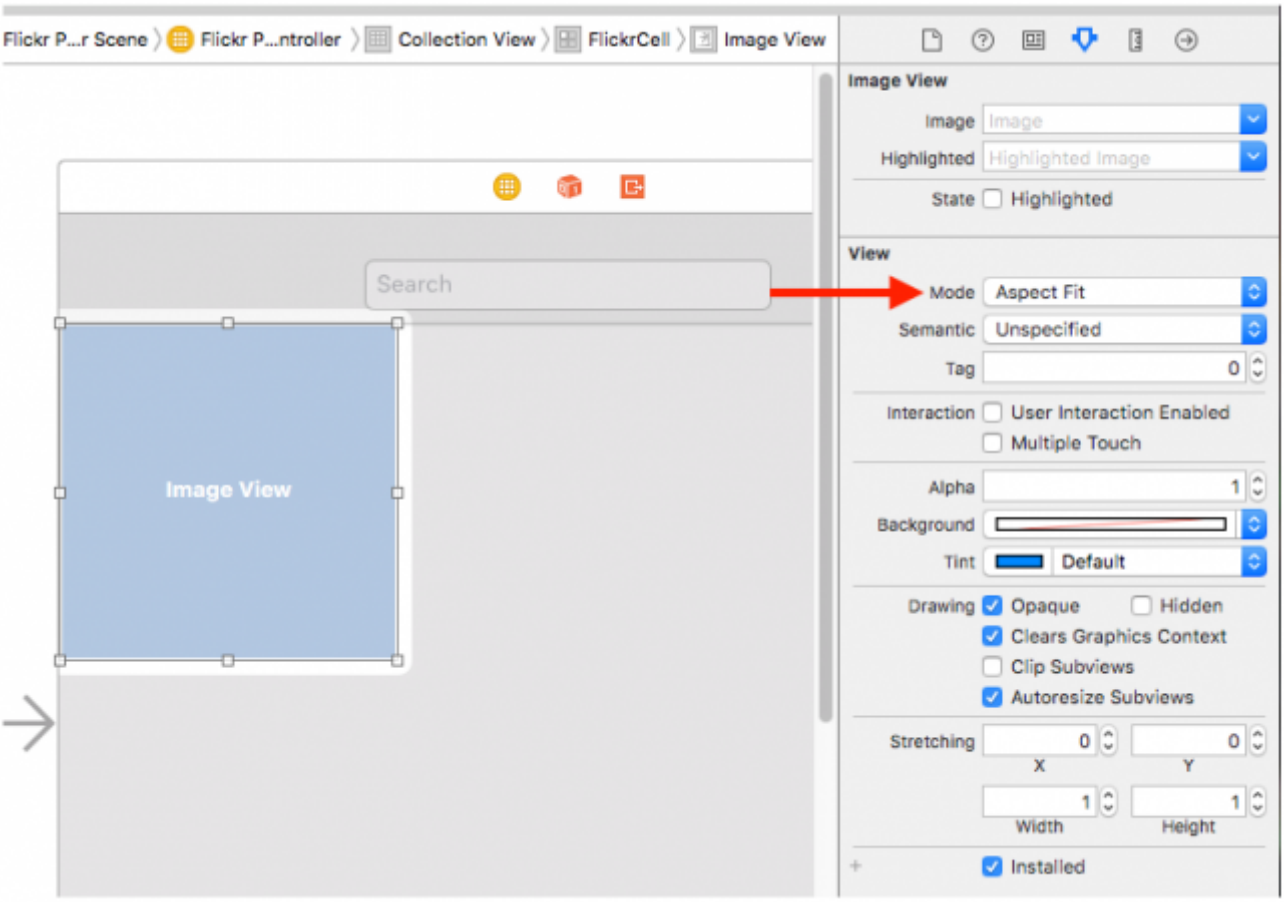


> **Note:** Setting this size doesn't affect the cells in your app, because you've implemented the delegate method to give a size for each cell, which overwrites anything set in the storyboard.

Drag an image view onto the cell and stretch it so it perfectly takes up the entire cell. With the image view still selected, open the pin menu, uncheck **Constrain to margins** and add constraints of 0 points all the way round:



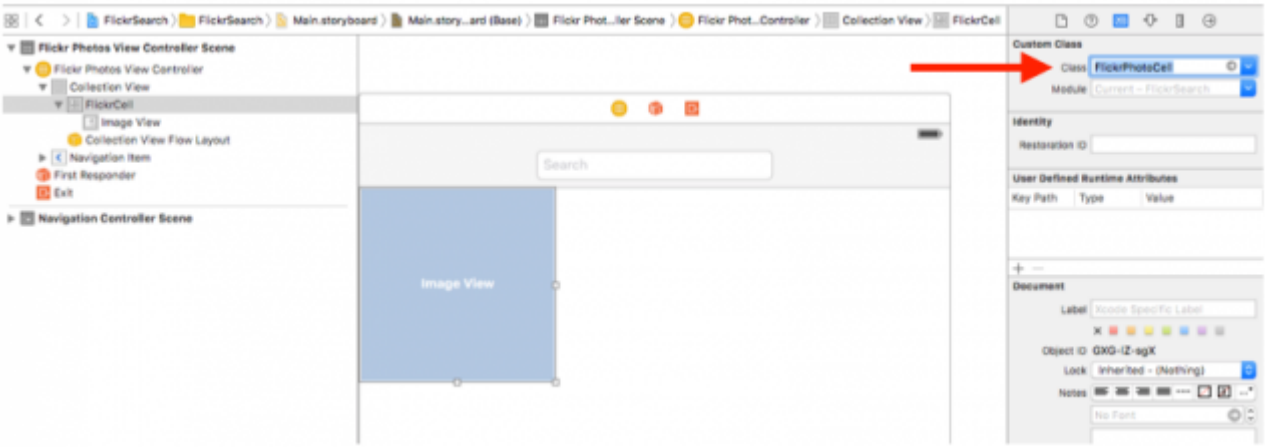
With the image view still selected, change its **Mode** to **Aspect Fit**, so the images are not cropped or stretched in any way:



`UICollectionViewCell` doesn't allow for much customization beyond changing the background color. You will almost always want to create your own subclass, to allow you to easily access any content subviews you have added.

Choose **File\New\File...** and choose **iOS\Source\Cocoa Touch Class** template and click **Next**. Name the new class **FlickrPhotoCell**, making it a subclass of `UICollectionViewCell`.

Open **Main.storyboard** and select the cell. In the identity inspector, set the cell's class to **FlickrPhotoCell**:



Open the Assistant editor, making sure it is displaying **FlickrPhotoCell.swift** and control-drag from the image view to the class to add a new outlet, naming it **imageView**:

Now you have a custom cell class with an image view. It's time to put a photo on it! Open **FlickrPhotosViewController.swift** and replace `collectionView(_:cellForItemAtIndexPath:)` with the following:

```
override func collectionView(collectionView: UICollectionView, cellForItemAt indexPath:
NSIndexPath) -> UICollectionViewCell {
  //1
  let cell = collectionView.dequeueReusableCell(withReuseIdentifier: reuseIdentifier,
              for: indexPath) as! FlickrPhotoCell
  //2
  let flickrPhoto = photoForIndexPath(indexPath)
  cell.backgroundColor = UIColor.white
  //3
  cell.imageView.image = flickrPhoto.thumbnail

  return cell
}
```

This is a little different from the placeholder method you defined earlier.

1. The cell coming back is now a `FlickrPhotoCell`
2. You need to get the `FlickrPhoto` representing the photo to display, using the convenience method from earlier
3. You populate the image view with the thumbnail

Build and run, perform a search and you'll finally see the pictures you've been searching for!

Yes! Success!

> **Note:** If your view doesn't look like this or the photos are acting weird, it likely means your Auto Layout settings aren't correct. If you get stuck, try comparing your settings to the solution for this project.

At this point, you've now got a complete working (and quite cool) example of `UICollectionView` – give yourself a pat on the back! Here's a link to the finished project: FlickrSearch

Where To Go From Here?

But there's more! Stay tuned for part 2 of this tutorial, where you will learn:

- How to add custom headers to collection views
- How to easily move cells by dragging them
- How to implement single cell selection to bring up a detail view
- How to implement multi-cell selection too!

In the meantime, if you have any questions or comments on what you've learned so far, please join the forum discussion below!

Bradley Johnson

I'm a Mobile Developer at Getty Images Inc. in Seattle, Wa. I've been doing iOS for about 4 years now, and just started dabbling in Android as well. Swift is amazing! Go hawks.