Project 3: Social Media with UIActivityViewController

Overview

Brief: Let users share to Facebook and Twitter by modifying Project 1.
Learn: UIBarButtonItem, UIActivityViewController, the Social framework, NSURL.

- UIActivityViewController explained
- Twitter and Facebook: SLComposeViewController
- Wrap up

UIActivityViewController explained

Sharing things using iOS uses a standard, powerful component that other apps can plug into. As a result, it should be your first port of call when adding sharing to an app. This component is called UIActivityViewController: you tell it what kind of data you want to share, and it figures out how best to share it.

As we're working with images, UIActivityViewController will automatically gives us functionality to share by iMessage, by email and by Twitter and Facebook, as well as saving the image to the photo library, assigning it to contact, printing it out via AirPrint, and more. It even hooks into AirDrop and the iOS 8 extensions system so that other apps can read the image straight from us.

Best of all, it takes just a handful of lines of code to make it all work. But before we touch UIActivityViewController, we first need to give users a way to trigger sharing, and the way we're going to use is to add a bar button item.

Project 1, if you recall, used a UINavigationController to let users move between two screens. By default, a UINavigationController has a bar across the top, called a UINavigationBar, and as developers we can add buttons to this navigation bar that call our methods.

Let's create one of those buttons now. First, take a copy of your existing Project1 folder (the whole thing), and rename it to be Project3. Now launch it in Xcode, open the file DetailViewController.swift, and find the viewDidLoad() method. It's mostly empty right now – it just calls super.viewDidLoad() then configureView(). Directly beneath configureView(), I want you to add this code:

```
navigationItem.rightBarButtonItem =
UIBarButtonItem(barButtonSystemItem: .Action, target: self,
action: #selector(shareTapped))
```

This is easily split into two parts: on the left we're assigning to the rightBarButtonItem of our view controller's navigationItem. This is navigation item is used by the navigation bar so that it can show relevant information. In this case, we're setting the right bar button item, which is a button that appears on the right of the navigation bar when this view controller is visible.

On the right we create a new instance of the UIBarButtonItem data type, setting it up with three parameters: a system item, a target, and an action. The system item we specify is .Action, but you can type UIBarButtonSystemItem. To have code completion tell you the many other options available. The .Action system item displays an arrow coming out of a box, signalling the user can do something when it's tapped.

The target and action parameters go hand in hand, because combined they tell the UIBarButtonItem what method should be called. The action parameter is saying "when you're tapped, call the shareTapped() method", and the target parameter tells the button that the method belongs to the current view controller – self.

The part in #selector bears explaining a bit more, because it's new and unusual syntax. What it does is tell the Swift compiler that a method called "shareTapped" will exist, and should be triggered when the button is tapped. Swift will check this for you: if we had written "shareTaped" by accident – missing the second P – Xcode will refuse to build our app until we fix the typo.

If you don't like the look of the various system bar button items available, you can create with one with your own title or image instead. However, it's generally preferred to use the system items where possible because users already know what they do.

With the bar button created, it's time to create the shareTapped() method. Are you ready for this huge, complicated amount of code? Here goes! Put this just after the viewWillDisappear() method:

```
func shareTapped() {
    let vc = UIActivityViewController(activityItems:
[detailImageView.image!], applicationActivities: [])
    vc.popoverPresentationController?.barButtonItem =
navigationItem.rightBarButtonItem
    presentViewController(vc, animated: true, completion:
nil)
}
```

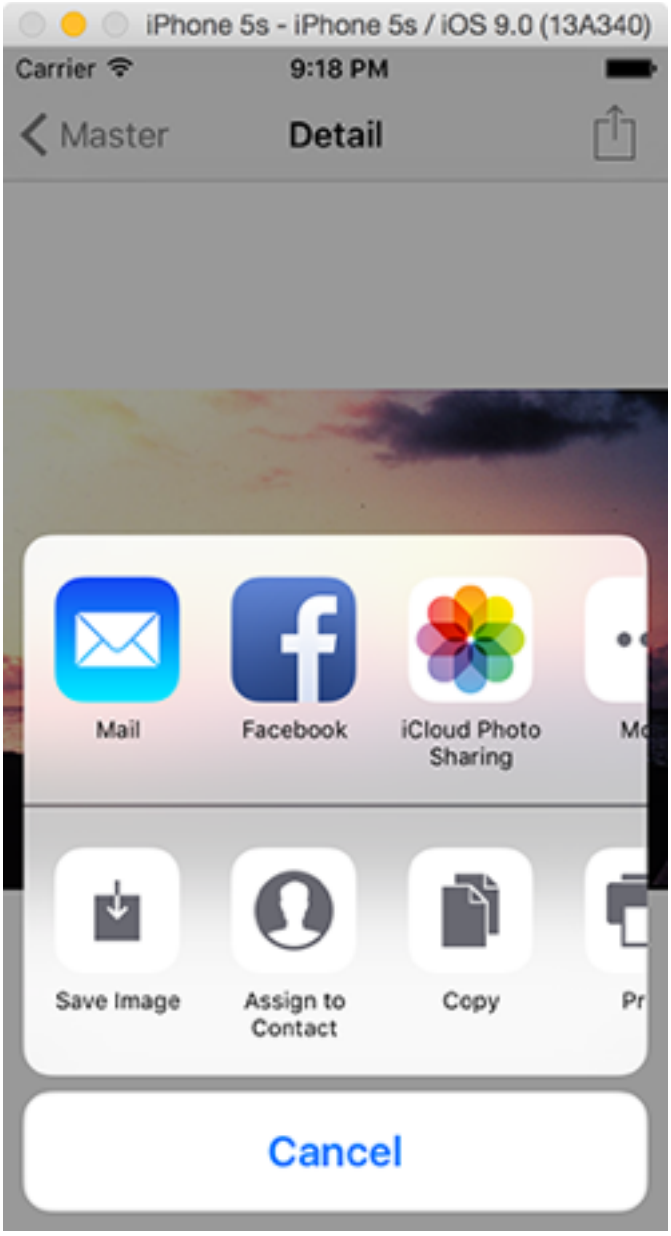
That's it. With those three lines of code, `shareTapped()` can send photos via `AirDrop`, post to `Twitter`, and much more. You have to admit, `iOS` can be pretty amazing sometimes!

The third line is old; we already learned about `presentViewController()` in `Project 2`. However, lines 1 and 2 are new, so let me explain what they do: line 1 creates a `UIActivityViewController`, which is the `iOS` method of sharing content with other apps and services, and line 2 tells `iOS` where the activity view controller should be anchored – where it should appear from.

On `iPhone`, activity view controllers automatically take up the full screen, but on `iPad` they appear as a popover that allows the user to see what they were working on below. This line of code tells `iOS` to anchor the activity view controller to the right bar button item (our share button), but this only has an effect on `iPad` – on `iPhone` it's ignored.

Let's focus on how activity view controllers are created. As you can see in the code, you pass in two items: an array of items you want to share, and an array of any of your own app's services you want to make sure are in the list. We're passing an empty array into the second parameter, because our app doesn't have any services to offer. But if you were to extend this app to have something like `“Other pictures like this”`, for example, then you would include that functionality here.

`So, the real focus is on the first parameter: we're passing in [detailImageView.image!]`. If you recall, the image was being displayed in a `UIImageView` called `detailImageView`, and `UIImageView` has an optional property called image, which holds a `UIImage`. But it's optional, so there may be an image or there may not. And `UIActivityViewController` doesn't want maybe or maybe not, it wants facts.



Fortunately, we know for a fact that our image view has an image, because we set it! In fact, that's the whole point of this view controller. So we use `detailImageView.image!` with that exclamation mark on the end to force unwrap the optional. That then gets put into an array by itself, and send to `UIActivityViewController`.

And… that's it. No, really. We're done: your app now supports sharing. If you want to try pushing your skills further, try modifying `shareTapped()` so that it shows a message if no image was selected. This is only possible on `iPad`, but it's worth catching.

Twitter and Facebook: SLComposeViewController

OK, so I would feel guilty if I didn't spend a little more time with you showing you other ways to share things, in particular there's built-in support for `Facebook` and `Twitter` sharing in `iOS` and both are quite easy to use.

iOS includes a framework called “Social”, which is designed to post to social networks like Facebook and Twitter. We can use both of these in our app to share the image the user is looking at, and it has the added benefit that the user is immediately prompted to enter their tweet / Facebook post – there's no initial view controller there asking them how they want to share.

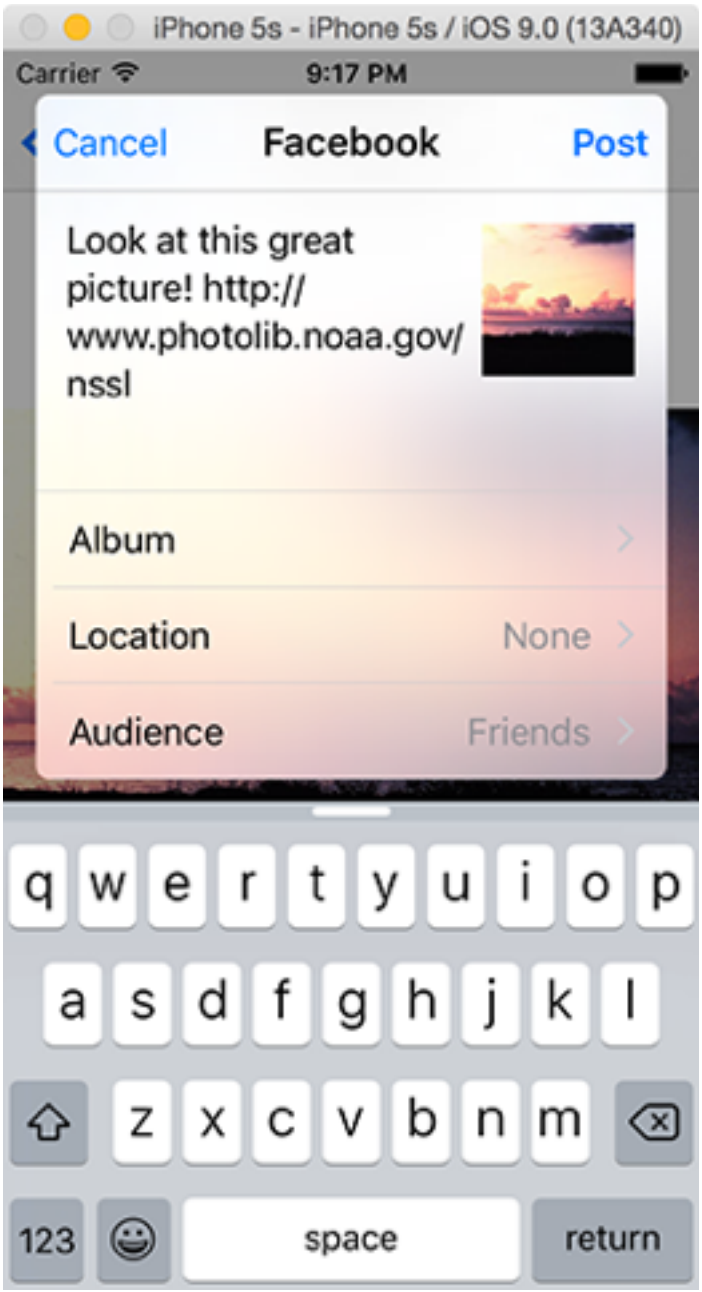
Happily, using the Social framework to post to social media is also straightforward, and has the advantage of simple method calls that are self-describing. In fact, I'm just going to go ahead and show you the code, and see what you think:

```
let vc = SLComposeViewController(forServiceType:
SLServiceTypeFacebook)
vc.setInitialText("Look at this great picture!")
vc.addImage(detailImageView.image!)
vc.addURL(NSURL(string: "http://www.photolib.noaa.gov/nssl"))
presentViewController(vc, animated: true, completion: nil)
```

Apart from the SLComposeViewController component, which as you can see is created with the Facebook service type, the only other new thing in there is NSURL. This is a new data type, and one that might seem a little redundant at first: it stores and processes URLs like www.yoursite.com.

Now, clearly to you and me a URL is a text string, so it seems strange to have a dedicated class when a plain old string would do. However, iOS uses URLs for more things than just websites. For example, you can get a file URL to a local file, or you can get a URL to a document securely stored in iCloud. And even if it were just about website URLs, even then there's things like “is it HTTP or HTTPS?” and “is there a username and password in the URL?”

We're going to use NSURL again in the next project, but right now its use is quite simple: we're attaching the URL to the `National Severe Storm Laboratory` so that people can browse there for more photos. The `NSURL(string:)` method converts the string "`http://www.photolib.noaa.gov/nssl`" into a full NSURL instance, which can then be passed to `addURL()`.



Now that code is in place, you will find it doesn't actually compile. This is because the `Social framework` is separate from the default `iOS` frameworks, so it isn't included as standard. If you want `to import the Social framework` (and you will want to if you ever want your code to compile again!), just scroll to the top of the view controller and look for the line `import UIKit`. Now put this line directly above:

```
import Social
```

`UIKit` is the name for `Apple's iOS development framework`, so this includes both `UIKit and the Social framework`.

Finally, if you want to use `Twitter` instead, just specify `SLServiceTypeTwitter` for the service type; the rest of the code stays the same.

Wrap up

This was a deliberately short technique project taking an existing app and making it better. I hope you didn't get too bored, and hope even more that some of the new material sunk in because we covered UIBarButtonItem, UIActivityViewController, the Social framework, and NSURL.

I hope you can see how trivial it is to add social media to your apps, and it can make a huge difference to helping spread the word about your work once your apps are on the App Store. I hope this project has also shown you how easy it is to go back to previous projects and improve them with only a little extra effort.