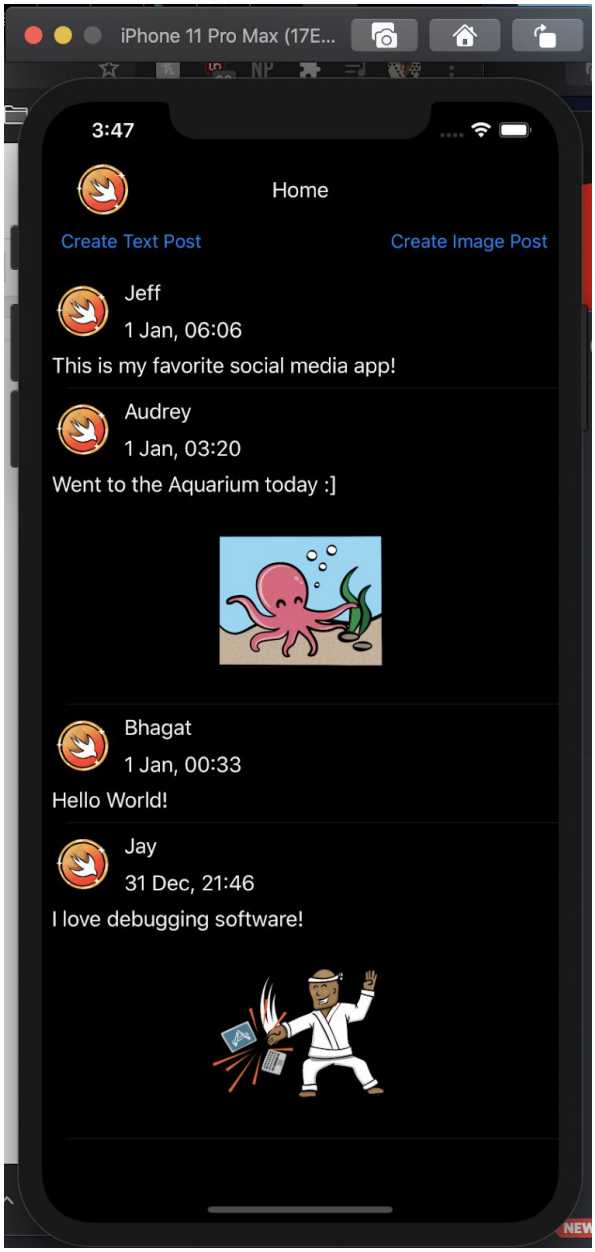
Birdie - A Twitter-like app with tableviews

A recording of the final product in action: https://share.getcloudapp.com/12uNJ2X6



Part 1: Successfully completing these tasks will earn you **"Good Job!"**

1. **Populate the Tableview**
    a. Open the **Model** folder. Take a look at **MediaPostsHandler.swift**. **MediaPostsHandler** is a Singleton class that will hold an array of **MediaPost** objects. A singleton means that any time you reference **MediaPostsHandler.shared.mediaPosts**, you will get the same memory

address every time. This is a benefit because you don't want to run into a situation where one version of the array has 4 posts, but another could have 5. You definitely don't want to make EVERY class a singleton/shared instance, but for objects used in multiple places, it's a good idea.

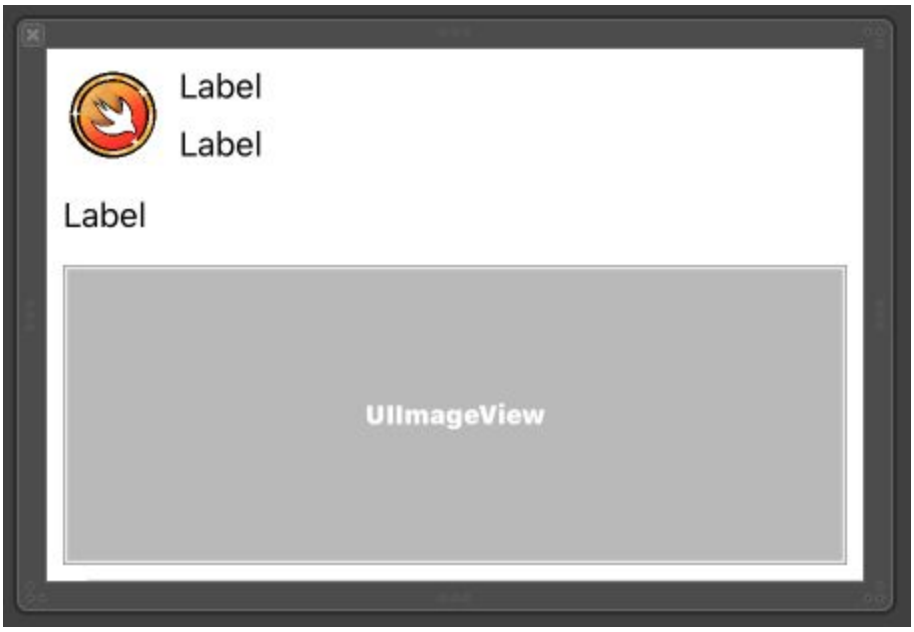
i. In a real app, **getPosts()** would most likely connect to the internet, pull down information, then make these posts that you can put in a tableview. We simulate that by having it make a few sample posts and add them to the array.

ii. **mediaPosts** is an array of items that conform to the protocol **MediaPost,** found in **MediaPostProtocol.swift**. That way posts with images (**ImagePostModel**) and without images (**TestPostModel**) can still be in the array together.

b. Make 2 custom cells

i. You can do this either in the storyboard or with xibs, whichever way you prefer. Xibs might be a little easier to use autolayout in. Make sure that your text body labels can handle more than one line of text!

ii. 

iii. 

c. Go back to **ViewController.swift.** Conform your viewcontroller to **UITableVIewDelegate** and **UITableViewDataSource**. Don't forget that the datasource is coming from **MediaPostsHandler,** so the array of posts is referenced like so: **MediaPostsHandler.shared.mediaPosts**.
    i. The tricky part of setting the cells up in **cellForRowAtIndexPath** is that even with the protocol, you're still working with two different structs and 2 different cells. You can use **downcasting** the way that it's used in this article to help you write if statements to get each type of struct from each item in the array.
        1. Hint: If you ever get a crash because an IBOutlet in your cell is nil when it shouldn't be, make sure that you registered the cell to the tableview.
    ii. To display the timestamp as a string, you will need to use DateFormatter.

2. **Add a text post to the tableview**
    a. When you tap the "Create Text Post" button, make a UIAlert pop up that asks for the user's Username and the text of the post.
    b. You can add the new text post with **MediaPostsHandler.shared.addTextPost.**
    c. When you create a text post, you can just have the timestamp be **Date().** Writing **Date()** makes a timestamp with the time the code is run, so it would be the timestamp for "now".
    d. Don't forget to reload the tableview so your changes appear.

Part 2: Successfully completing these tasks will earn you **"Above and Beyond"**

1. **Add an image post to the tableview**
    a. When you tap the "Create Image Post" button, use **UIImagePickerControllerDelegate** to bring up the user's camera roll so they can select an image. Here's a short tutorial: https://www.raywenderlich.com/3716-ios-101-with-swift-2/lessons/11
        i. Note: If you're using a physical device instead of a simulator, you will need to allow the app to access images in the info.plist.
    b. After selecting the image, make a UIAlert pop up that asks for the user's Username and the text of the post.
        i. Hint: You can set the selected image as a variable at the top so you can access it in any function.
        ii. Hint: make sure that after you make the image post, if you store it as a property at the top and don't clear it after making the post, make sure the app doesn't think a text post made after the image post has a selected image.
    c. You can add the new image post with **MediaPostsHandler.shared.addImagePost.**
    d. Don't forget to reload the tableview so your changes appear.

2. **Move the logic in cellForRowAt into a ViewModel.**
    a. Model-View-ViewModel (MVVM) is a design pattern similar to MVC, and it shares the same goal of MVC, which is to abstract logic *out* of your viewcontroller so that you aren't working with huge, confusing files.
    b. Here's a tutorial I wrote about view models that can help get you started.
    c. The **MediaPostsViewModel** should have a function that takes information like a **MediaPost**, determines which custom tableview cell to use, then returns a tableviewcell that **cellForRowAt** can use.

Good luck! :]