COMP41550/47390: CocoaTouch

Individual Assignment 4: TweeterTags

February 15, 2017

Table of Contents

Objectives

In this assignment, you will build an App to fetch tweets given a hashtag or a user name as a searching keyword. In Part1, the user interface will allow for the input of a hashtag query and display Twitter's search results in a tableview, using custom cells to display content of tweets (user avatar, screen name, list of hashtags, urls, images or users mentioned). In Part 2, you will enhance your App to allow the user to navigate the content of a tweet and display linked images.

Important notes

- You will need a Twitter account to complete this assignment.
- First ensure that your account information is properly set for the provided TwitterAPI class to work.
- Twitter accounts are tightly integrated with iOS and can be configured with your Twitter credentials on your device from the iOS Settings App (this is also possible in the iOS Simulator).

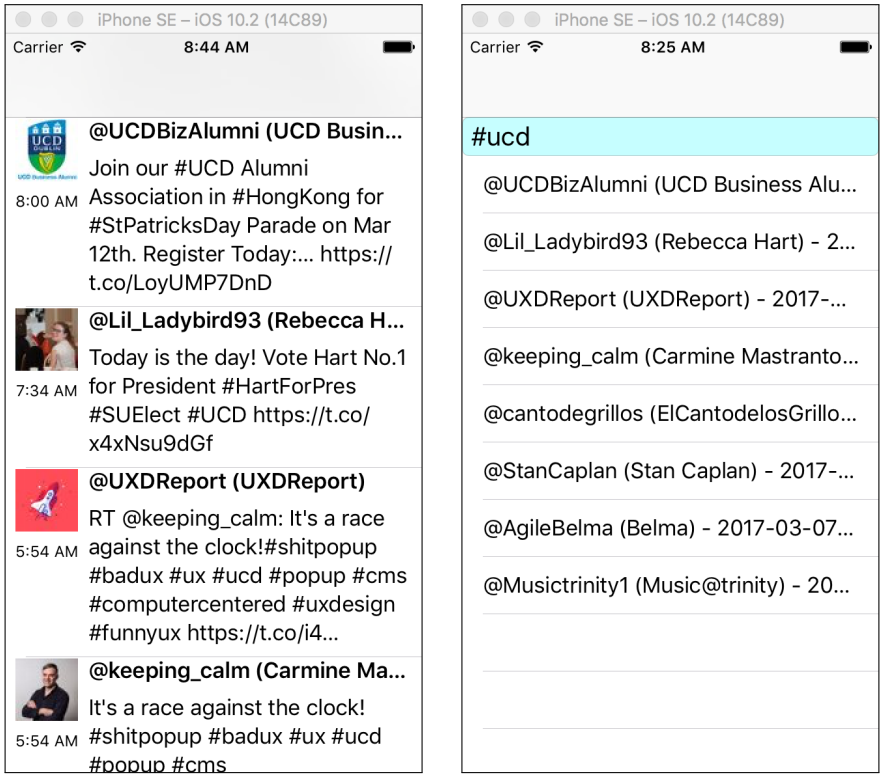
Part 1

Build an App to retrieve tweets from Twitter for a given hashtag and display the results in a tableview with custom cell view as shown in Figure 1a.

Step 1: here is a rough outline of the tasks required to complete this part successfully

1. Create a new single view application for iOS called `TweeterTags`.
2. Import the provided `TwitterAPI.swift` class.
3. Delete and remove the default `ViewController.swift` from your project.
4. Create a new `UITableViewController` subclass called `TweetsTVC`.
5. Edit storyboard, delete the existing view controller and replace it by a new `UITableViewController`. Set its custom class to `TweetsTVC`. Embed your new table view controller in a navigation view controller.
6. Add your model to the code of your `TweetsTVC` class: `var tweets = [[TwitterTweet]]()`.
7. Add a new optional property called `twitterQueryText` and set default value to `"#ucd"`.
8. Add a private action method called `refresh()` to fetch new tweets using the text of `twitterQueryText` and to subsequently update the data model `tweets`. Ensuring that you do not block the UI while calling `TwitterRequest()`.

9. Add the appropriate code to the `didSet()` property observer of `twitterQueryText`, this is to ensure that your tableview is updated each time the data model is changed.
10. In your tableview controller method `viewDidLoad()`, ensure your data model is updated by calling the `refresh()` method.
11. Implement all required tableview data source delegate methods. Do not forget to set your cell identifier in the storyboard's scene.



(a) (b)

Figure 1: TweeterTags App – Screenshots of Part 1

Build & Test

For simplicity, use the iOS Simulator. Your app should build and run with no warnings with a UI similar the Figure 1b.

Step 2

1. Add a new textfield in your tableview above your tweet prototype cell. It will be used to input the Twitter query. Set the font to System 20.0, the placeholder text to `"Twitter Query"`, the keyboard type to `Twitter`.
2. Wire-up your textfield to an outlet called `twitterQueryTextField` in your custom `TweetsTVC` tableview controller.
3. Ensure your `TweetsTVC` class conforms to the `UITextFieldDelegate` and implement the delegation method `textFieldShouldReturn(_ textField: UITextField)` to dismiss the keyboard and update the `twitterQueryText` property.
4. Create a new custom table view cell called `TweetTVCell`. Edit your tableview controller scene in your storyboard, set the prototype cell class to your custom class.

5. Customise your prototype cell with and `UIImageView` for the user avatar, and `UILabel`s for the user screen name, the tweet content and the tweet date as show in Figure 2.
6. Add required layout constraints so that tweets are displayed correctly in your custom cell. Note, you can set the number of lines to 0 if you want a text label to automatically wrap to the required number of lines and fit the entire text.
7. You will also need to initialise the tableView's `estimatedRowHeight` to the value of the tableView's `rowHeight`, and then set the tableView's row height to the `UITableViewAutomaticDimension` constant so that autolayout can automatically adjust the cell height to its content. This should be done in the `viewDidLoad()` method of your tableview controller.
8. Finally, add the relevant code to your tableview controller (data source) and to your custom cell class (configure cell). Build and test your app that should run with no warnings and show a UI similar to that of Figure 1a.

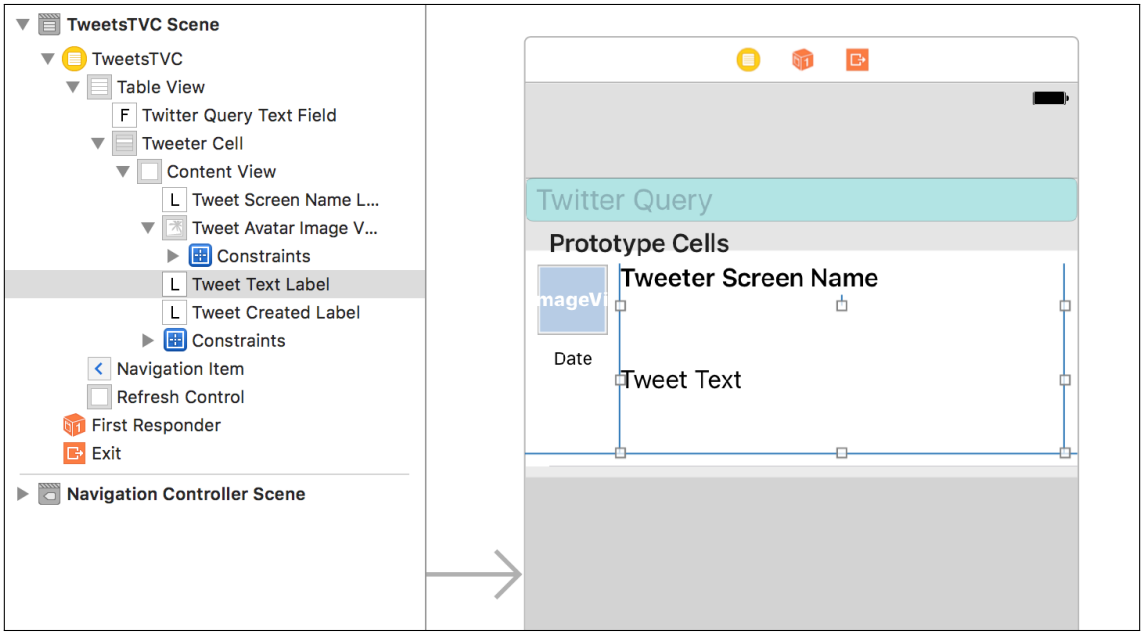
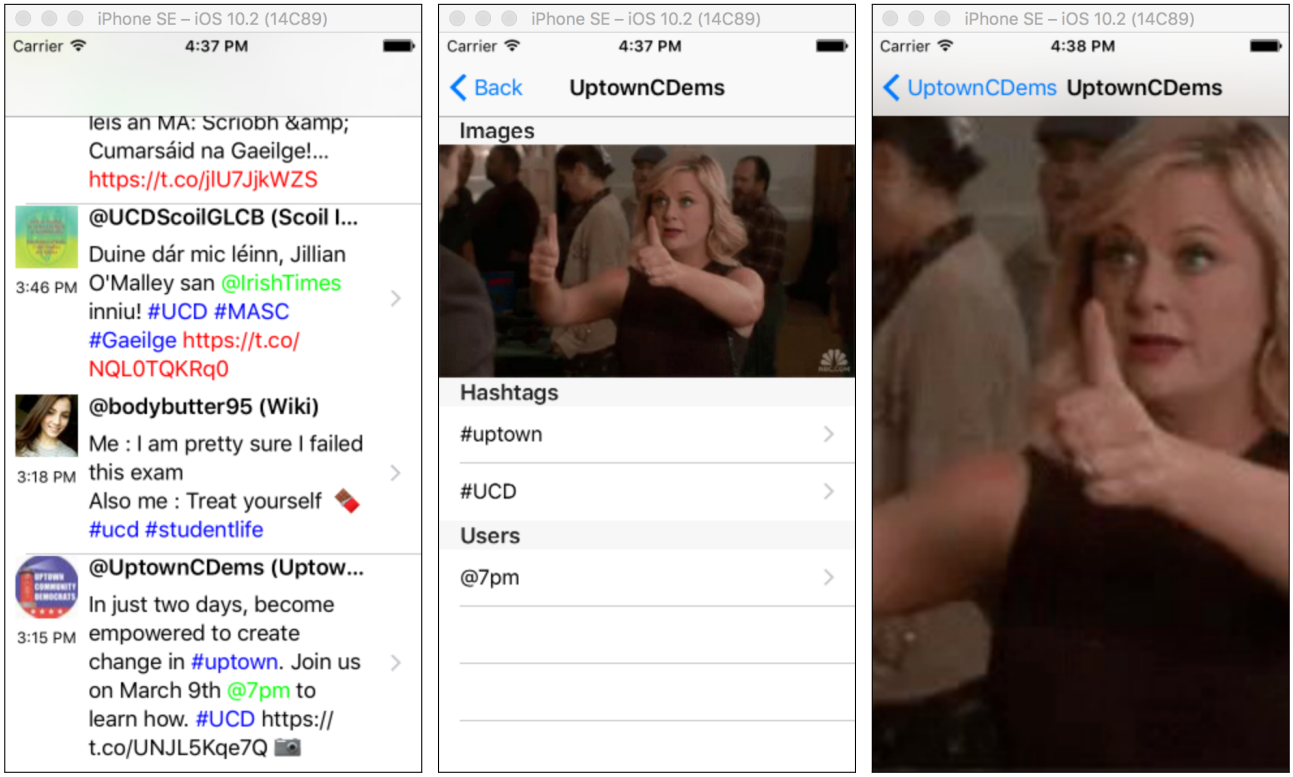


Figure 2: Custom Table View Cell

Part 2: Enhance your app with navigation features to explore tweets' content (mentions)

1. Enhance Part1 to highlight tweet's mentions (hashtags, urls, screen names) in different colours. Note that mentions are extracted from tweets in the TwitterAPI class and show up as arrays `[TwitterMention]` in the `TwitterTweet` class of the supplied TwitterAPI code. Your UI should look like Figure 3a.
2. When a user clicks on a tweet, segue to a new `UITableViewController` that should be configured with four sections for each of the mention types (images, URLs, hashtags and users). The first section will display any images attached to the tweet, (one per row, images are found in the media property of the `TwitterTweet` class). The last three sections show the items described in Step 1 (one per row).
3. In your tableview controller, implement the appropriate tableview datasource delegate so that mentions have an appropriate header. If a section has no items in it, there should be no header visible for that section, as shown Figure 3b.
4. Add a segue to show the Twitter search results when the user selects an entry in the hashtag or user sections of the `MentionsTVC`. Note, you should be fetching for hashtags or users (not just a



(a) (b) (c)

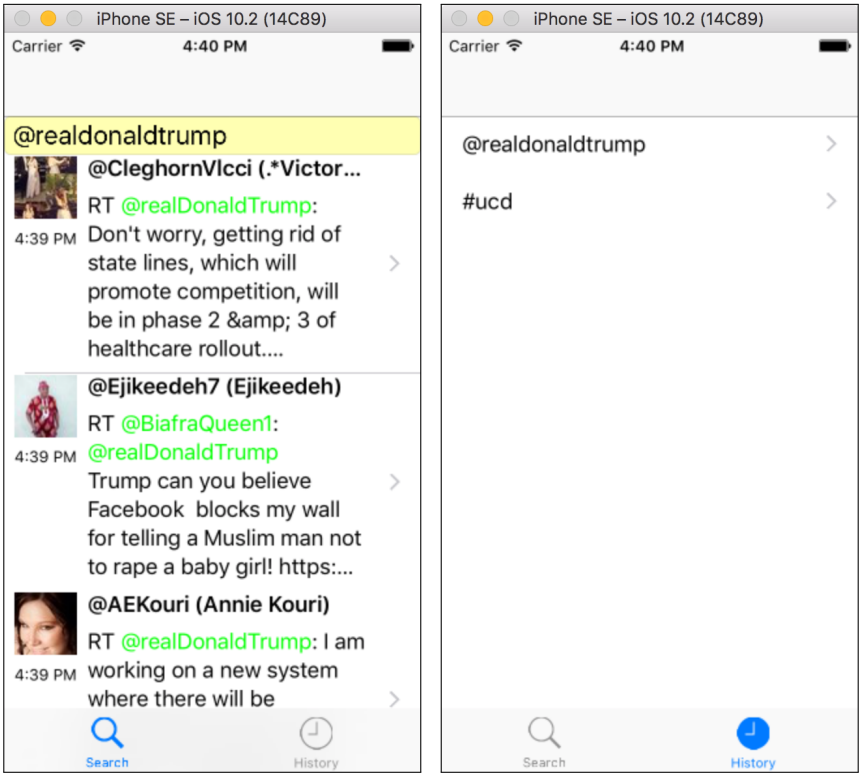
Figure 3: TweeterTags App – Screenshots of some Part 2 steps

string of the hashtag or user names, e.g. search for `"#ucd"`, not `"ucd"`). The view controller you are segueing to must work identically to your main `TweetsTVC`. If the user clicks on a mentioned url, you should open up that url in Safari (see note below).

5. When the user selects an image in the `MentionsTVC`, segue to a new `ImageVC` which lets the user scroll around and zoom in on the image. Note, when the image first appears in the `ImageVC`, it should display zoomed (in its normal aspect ratio) to show as much of the image as possible but with no whitespace around it as show in the Figure 3c.
6. Add persistence to your App by keeping track of the 100 most recent Twitter searches performed by the user. Add a `UITabBarController` to your application with a tab for searching and a second tab showing these most recent search terms in a new tableview controller called `RecentsTVC`. When a user clicks on a search term in the second tab, segue to show the most recent tweets matching that search term. Store these most recent searches permanently in `UserDefaults` so that your application doesn't forget them if it is restarted. Your final UI should like the figure Figure 4.

Notes

- You only to use the `fetchTweets()` method of the `TwitterRequest` class in the provided TwitterAPI class, along with some public properties of the `TwitterTweet`, `TwitterUser` and `TwitterMedia` classes.
- You App must not block the main thread of your application, this particularly important when making network requests. Note, `fetchTweets()` executes its handler off the main thread.
- Ensure your App is responsive and displays correctly in both portrait and landscape on any iPhone.
- Remember that your App should always build and run without warning.



(a) (b)

Figure 4: TweeterTags App – Screenshots of Final App

- If you have an `URL` named url, you can open it in Safari from you App by calling `UIApplication` class method `UIApplication.shared.open(...)`.
- You should give a solid thought as to what the public API of your new controller should be. Make everything else private. Same applies to any UIKit subclass e.g. `UITableViewCell`.
- Think about what should be the appropriate title of your MVC. It will be displayed in the navigation bar of a navigation controller.
- When complete, your storyboard should be similar to Figure 5.
- Security changed in iOS9 now requires that add the following key to our `info.plist` file so that you can download data over `http`:

```
<key>NSAppTransportSecurity</key>
<dict>
        <key>NSAllowsArbitraryLoads</key>
        <true/>
</dict>
```

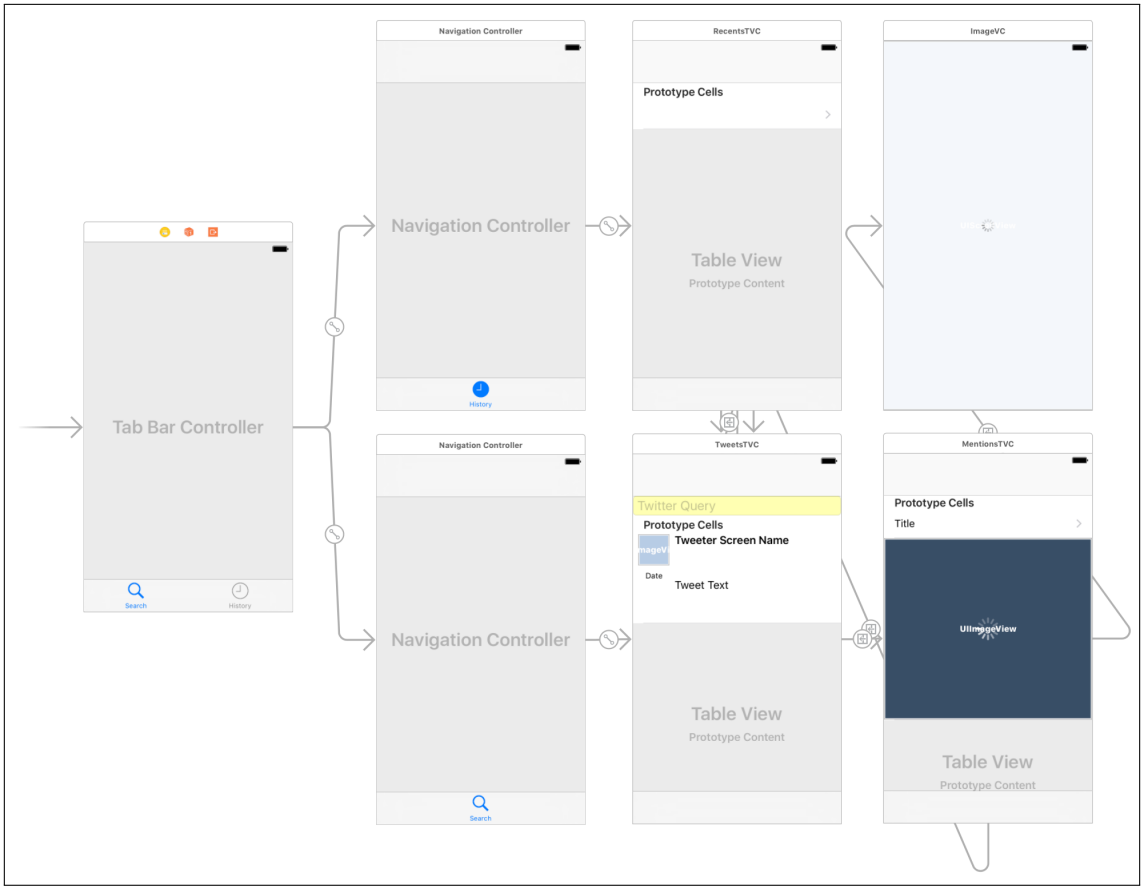


Figure 5: Storyboard of the TweeterTags App

Submission

For this assignment, testing will be done by running the project on your device. We will be looking at the following:

- Submit short reports called `readme-part<#>` with appropriate code segments and screenshots summarising your solution along with zip archives of your Xcode project called `<Firstname>_<Surname>_<StudentNumber>_part<#>.zip` for each parts.
- Your project should build without errors or warnings.
- Your project should run without crashing.
- Each of parts will be considered to verify that you've completed the assignment correctly.
- Your project should have a clean user interface, UI elements arranged logically, aligned neatly, etc.
- We will be verifying that all fundamental concepts are understood.
- Ensure your code is easy to read and not visually sloppy (indentation, etc...). Make good use of object-oriented design principles (avoid code duplication, use inheritance appropriately, etc...)
- Your solution should elegant and your code is easy for someone to read (right amount of comments, appropriate variable/method names, good solution structure, self documenting methods, etc...)

— END OF ASSIGNMENT 4 —