Homework 9: Stock Market Search IOS App

1. Objectives

- Become familiar with Swift language, Xcode and IOS App development.
- Practice the Model-View-Controller design pattern.
- Build a good-looking IOS app.
- Add social networking features using the Facebook SDK for IOS.
- Manage and use third-party libraries by CocoaPods

2. Background

2.1 Xcode

Xcode is an integrated development environment (IDE) containing a suite of software development tools developed by Apple for developing software for OS X, iOS, watchOS and tvOS. First released in 2003, the latest stable release is version 9.0 and is available via the Mac App Store free of charge for OS X High Sierra users.

Features:

- Swift 4 support
- Playgrounds
- Interface Builder
- Device simulator and testing
- User Interface Testing
- Code Coverage

The Official homepage of the Xcode is located at:

https://developer.apple.com/xcode/

2.2 iOS

iOS (originally iPhone OS) is a mobile operating system created and developed by Apple Inc. and distributed exclusively for Apple hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod touch. It is the second most popular mobile operating system in the world by sales, after Android.

The Official iOS home page is located at:

http://www.apple.com/ios/

The Official iOS Developer homepage is located at:

https://developer.apple.com/ios/

2.3 Swift

Swift is a general-purpose, multi-paradigm, compiled programming language created for iOS, OS X, watchOS, tvOS and Linux development by Apple Inc. Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C code written for Apple products. Swift is intended to be more resilient to erroneous code ("safer") than Objective-C and also more concise. It is built with the LLVM compiler framework included in Xcode 6 and later and uses the Objective-C runtime, which allows C, Objective-C, C++ and Swift code to run within a single program.

The Official Swift homepage is located at:

https://developer.apple.com/swift/

2.4 Facebook

Facebook provides developers with an API called the Facebook Platform. Facebook Connect is the next iteration of Platform, which provides a set of APIs' that enable Facebook members to log onto third-party websites, applications and mobile devices with their Facebook identity. While logged in, users can connect with friends via these media and post information and updates to their Facebook profile.

Below are a few links for Facebook Connect:

https://developers.facebook.com/

https://developers.facebook.com/docs/ios

2.5 Amazon Web Services (AWS)

AWS is Amazon's implementation of cloud computing. Included in AWS is Amazon Elastic Compute Cloud (EC2), which delivers scalable, pay-as-you-go compute capacity in the cloud, and AWS Elastic Beanstalk, an even easier way to quickly deploy and manage applications in the AWS cloud. You simply upload your application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. Elastic Beanstalk is built using familiar software stacks such as the Apache HTTP Server, PHP, and Python, Passenger for Ruby, IIS for .NET, and Apache Tomcat for Java.

The Amazon Web Services homepage is available at: http://aws.amazon.com/

2.6 Google App Engine (GAE)

Google App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go. App Engine applications automatically scale based on incoming traffic. Load balancing, micro services, authorization, SQL and noSQL databases, memcache, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

To learn more about GAE support for PHP visit this page:

https://cloud.google.com/appengine/docs/php/

To learn more about GAE support for Node.js visit this page:

https://cloud.google.com/appengine/docs/flexible/Node.js/

3. Prerequisites

This homework requires a number of software components outlined in this section.

3.1 Download and install the latest version of Xcode

To develop the iOS app described in this document, using the latest technologies, you need a Mac computer (macOS Sierra 10.12.4 or later) running the latest version of Xcode. Xcode includes all the features you need to design, develop, and debug an app. Xcode also contains the iOS SDK, which extends Xcode to include the tools, compilers, and frameworks you need specifically for iOS development.

Download the latest version of Xcode on your Mac for free from the App Store.

To download the latest version of Xcode

- Open the App Store app on your Mac (by default it's in the Dock).
- In the search field in the top-right corner, type Xcode and press the Return key.
- The Xcode app shows up as the first search result.
- Click Get and then click Install App.
- Enter your Apple ID and password when prompted.
- Xcode is downloaded into your /Applications directory.

You may use any other IDE other than Xcode, but you will be on your own if problems spring up.

3.2 Add your account to Xcode

When you add your Apple ID to the Xcode Accounts preferences, Xcode displays all the teams you belong to. Xcode also shows your role on the team and details about your signing identities and provisioning profiles that you'll create later in this document. If you don't belong to the Apple Developer Program, a personal team appears.

Here is detailed documentation:

https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppStoreDistributionTutorial/AddingYourAccounttoXcode/AddingYourAccounttoXcode.html

3.3 Install CocoaPods

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over ten thousand libraries and can help you scale your projects elegantly. You can install dependencies using it, we will need to install many third party modules and frameworks using it.

CocoaPods is built with Ruby and is installable with the default Ruby available on OS X. We recommend you use the default Ruby. Using the default Ruby install can require you to use sudo when installing gems.

Run the command below in your Mac terminal:

```
$ sudo gem install cocoapods
```

Once you have created you Xcode project, you can start to integrate cocoapods into your project.

Further guides on how to integrate cocoapods are available at: https://cocoapods.org/

4. High Level Design

This homework is a mobile app version of HW8. In this exercise, you will develop an iOS Mobile application, which allows users to search for stock information, save some stock symbols as favorites, and post to Facebook timeline. You should reuse the backend service (PHP/node.js script) you developed in HW8, with no changes needed.

The main "scene" of this app is like the one shown in Figure 1, where the user can enter the stock ticker symbol and select from a list of matching stock symbols using "autocomplete." A "stock quote" on a matched stock symbol can be performed.

Once the user has entered some characters in the edit box and selected a matching result from the autocomplete list, he/she would click on Get Quote, at which point validation must be done to check that the entered data is not empty.

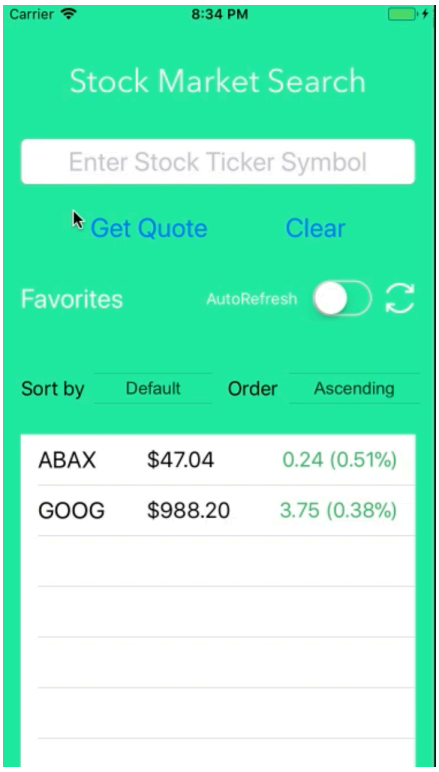

Figure 1. The Stock Market Search App

Once the validation is successful, the software would request the stock details using our PHP/Node.js script hosted on AWS/GAE, which would return the result in JSON format. Then we need to parse the JSON and display results similar to HW8. See the architecture overview of this homework in Figure 2.

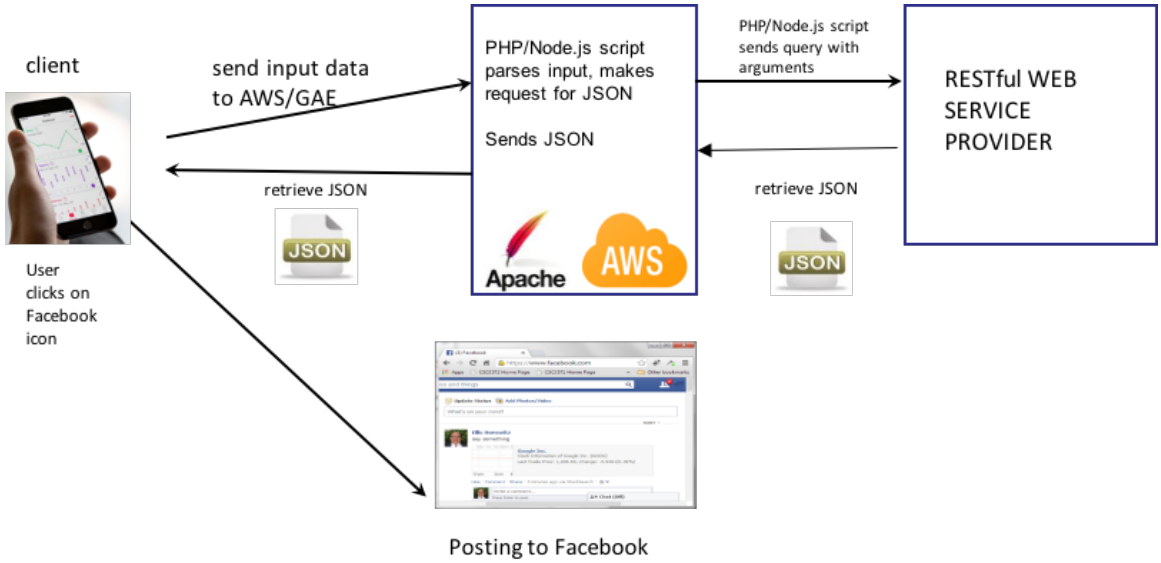

Figure 2. Architecture Overview

5. Implementation

5.1 Search Form

You must replicate the Search Form, as shown in Figure 1.

The interface consists of the following:

- An 'UITextField' component allowing the user to enter the stock symbol.
- A **Get Quote** button to get the stock quote, after validation. If the validations are successful, then the stock details would be fetched from the server. However, if the validations are unsuccessful, appropriate messages would be displayed and no further requests would be made to the server.
- A **Clear** button to clear the input field.

5.1.1 Auto-Complete

The user can enter part of the stock symbol to get stock name information from our PHP/Node.js script. Based on the user input, the auto-complete functionality would display the all the matching companies and symbols (see Figure 3). The auto-complete dropdown only displays when the user has typed in at least one character and the maximum number of results displayed in the auto-complete dropdown is 5.

To get the data used for auto-complete suggestions, you need to make http requests to your PHP/Node.js script which is located in the Google App Engine/Amazon Web Services.

If the user selects one of the results from the auto-complete dropdown, the content of the result (symbol with the company name) should be copied to the input field and the auto-complete dropdown then disappears.

If the user taps on an area other than the auto-complete form, the dropdown should be hidden.

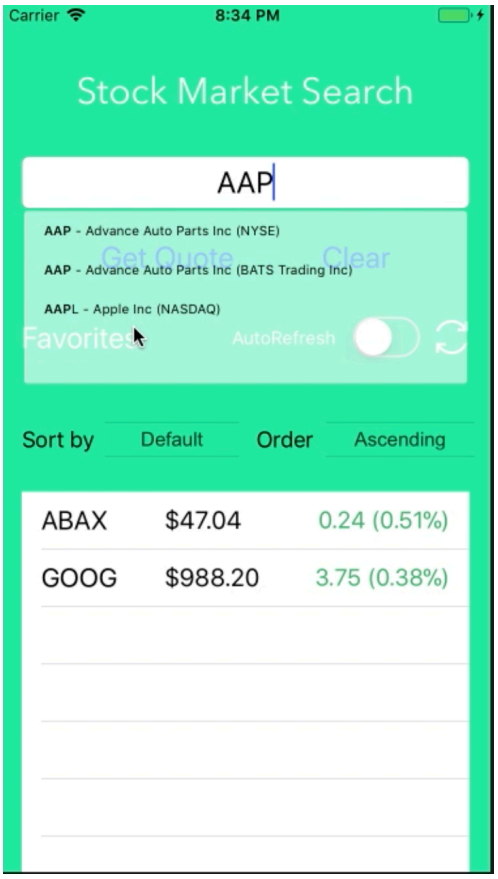

Figure 3: AutoComplete Suggestions

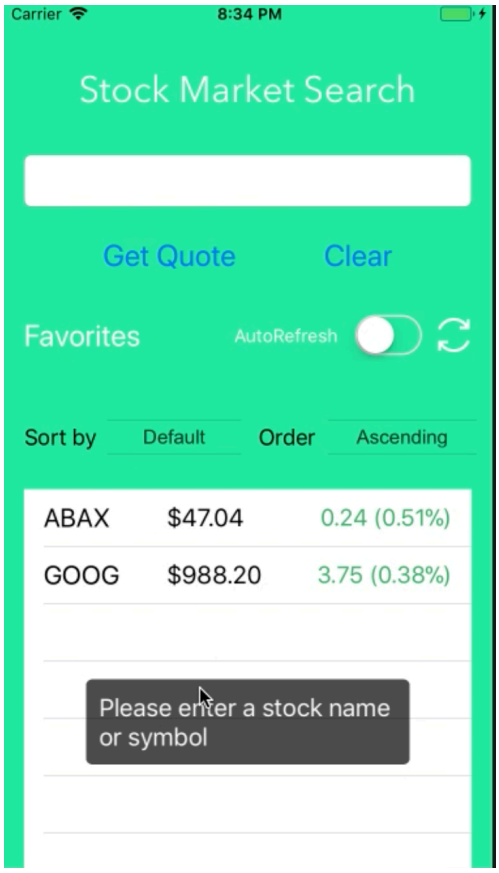

Figure 4: Validations

5.1.2 Validations

The validation for empty symbol entry needs to be implemented. If the user does not enter anything in the 'UITextField' or just enters some empty spaces, when he presses the **Get Quote** button you need to display an appropriate message to indicate the error, as shown in Figure 4.

5.1.3 Get Quote Execution

Once the validation is successful, you should execute an HTTP request to the PHP/Node.js script which is hosted on AWS/GAE (the cloud-based code you completed in Homework 8), and then navigate to the details page about the requested stock.

5.2 Favorites List

The Favorites list interface consists of the following:

- An Automatic Refresh switch, labeled **AutoRefresh**
- A Refresh button
- Two "UIPickerView" controlling the order of the list
- The Favorite "UITableView" showing a list of favorite stocks.

The stocks in the user's Favorites list would be displayed in a list as per Figure 5.

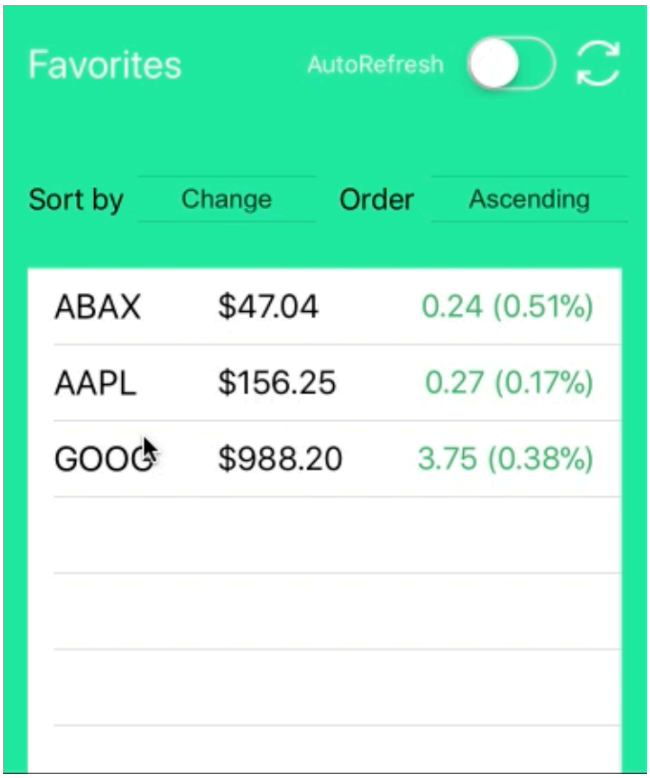

Figure 5: Favorite Stocks

Here are some important points about this feature:

- Display symbol, stock price and change (change percent) on each row.
- Sort by Default/Symbol/Price/Change/Change(%) in Ascending/Descending order
- See Homework 8 about the behavior of the **AutoRefresh** switch and refresh button
- Whenever the favorite list appears or re-appears, the price and change (change percent) data need to be updated. But the symbols should always be stored in "local device" storage, as shown in Figure 6.
- Display an 'activity indicator' while loading data from server, as shown in Figure 6.
- Display a proper error message if failing to update one or more stocks in the favorite list.
- Select a row to search that stock and navigate to the stock detail page.
- Swipe across a row and display a **Delete** button. Then user can then remove that stock from his/her Favorites list, as shown in Figure 7.

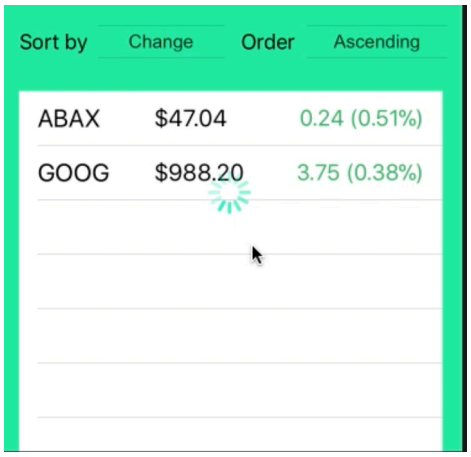

Figure 6: Display an activity indicator while loading data

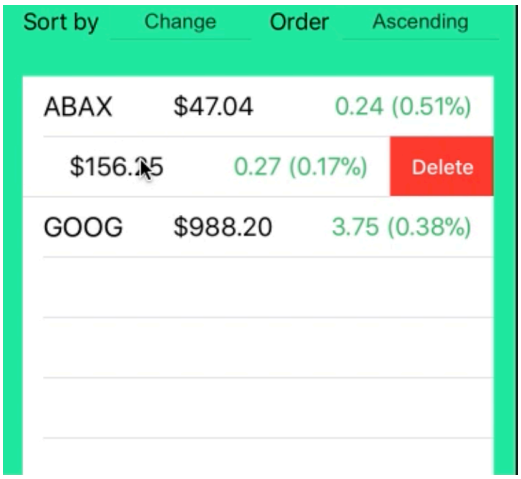

Figure 7: Swipe cross a row to delete

5.3 Stock Details

When the user clicks the **Get Quote** button, your app should display a "big spinner" before you are ready to show the stock details view, as shown in Figure 8. The Stock Details section should be designed as per Figure 9.

The stock detail section should have 3 views:

- Current Stock
- Historical Charts
- News Feeds

You can use a "segmented control" to navigate between 3 views above.

The back button in the header should navigate back to the Search Form.

The Stock Details would be starting with the 'Current' view as loaded by default. Furthermore, the stock details would have a table showing all the stock values. The list of the items in the stock details would be implemented using a 'UITableView'. The following stock values shpould be displayed: Stock Symbol, Last Price, Change, Timestamp, Open, Close, Day's Range, Volume. **The meaning of these values is the same as in Homework 8.** The Favorite button (star) should have the same behavior as in Homework 8. The function of the **Facebook** button will be discussed in section 5.6.

Figure 8: Display a big spinner while loading

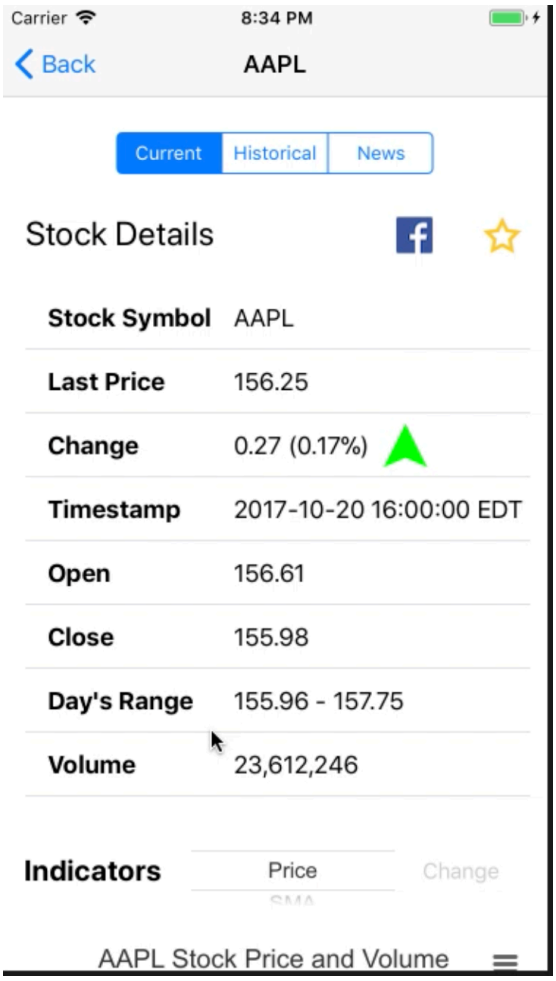

Figure 9: Stock Details

Below the list of stock details, you need to show the indicator choices and the high chart image, as shown in Figure 10. The user will need to scroll down to see these areas. A picker view and a button labeled **Change** are used to choose another indicator and change the high chart image, as shown in Figure 11. Here are some points:

- Include all the indicators used in Homework 8 and the chart is about the price/volume at the beginning.
- The **Change** button is only enabled if a different indicator is selected.
- You should use a "UIWebView" to display the chart and reuse some of your HTML and JavaScript code from previous homeworks. **But you should figure out a way to communicate between your Swift code and the JS code asynchronously**. It's **NOT** allowed to block the app while waiting for the chart to be shown in the UIWebView.
- Whenever the chart in the UIWebView is in a loading state, you should hide the previous chart (if there's any) and also display an activity indicator, as shown in Figure 12.
- Display a proper message if there is any failure inn retrieving a chart.

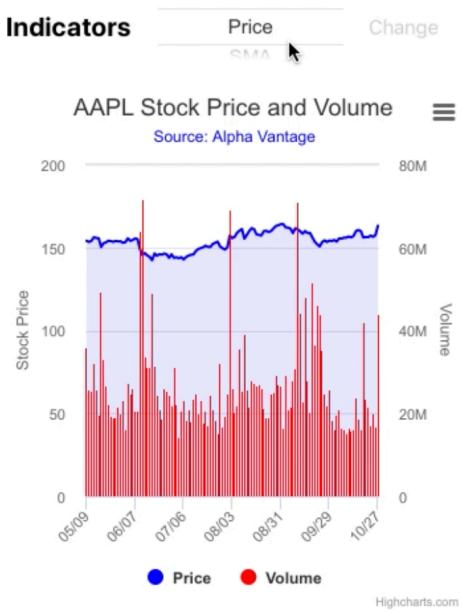

Figure 10: Price/Volume Chart

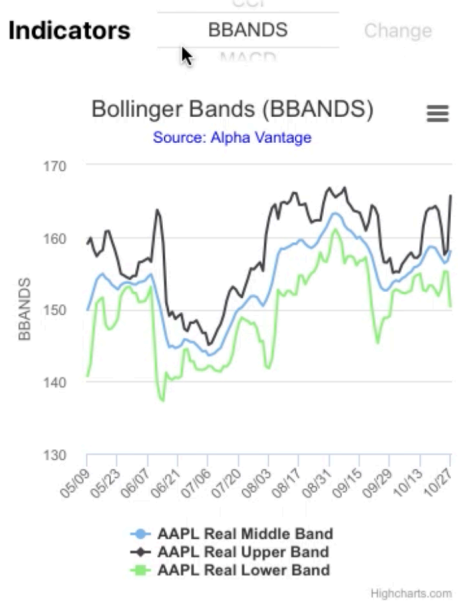

Figure 11: STOCH Chart

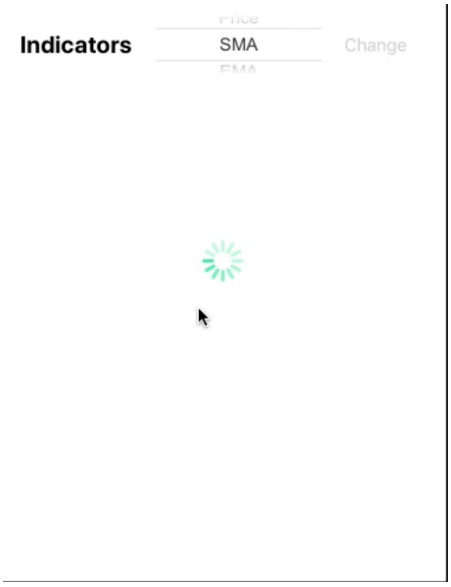

Figure 12: Display an activity indicator while loading the chart

5.4 Historical Charts

This tab represents the historical charts showing the value of the stock in the past market sessions. It has to be rendered as per Figure 13.

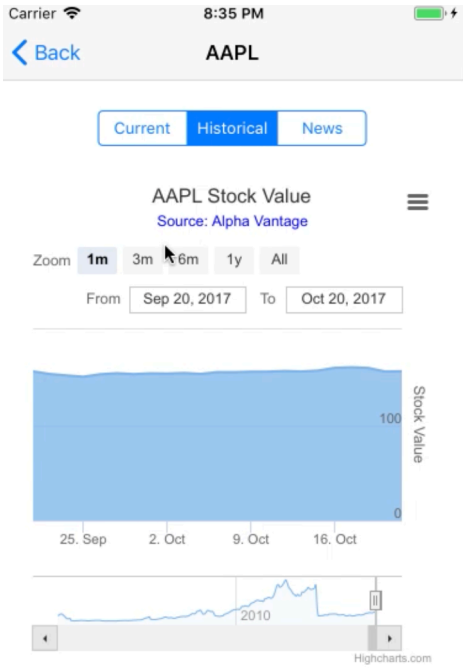

Figure 13: Historical Charts

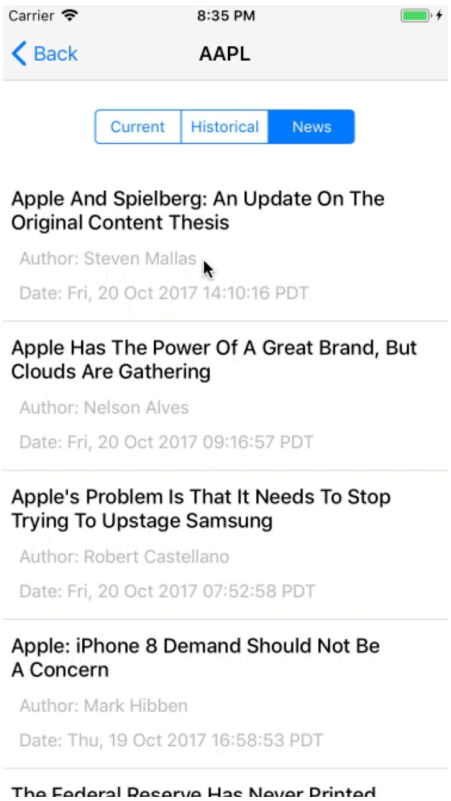

Figure 14: News Feed

This chart should also be implemented with a UIWebView. Similar requirements as with Indicator charts: remember to display the activity indicator while loading the chart and display a proper message if an error happens. Again, the communication between Swift code and JS code should be asynchronous.

5.5 News Feed

This page represents news articles related to the current stock. This would be rendered as per Figure 14.

The news articles need to be implemented in a 'UITableView'. Display title, author and date for each article. The app should open the article in the browser window (Mobile Safari) if one row is selected.

5.6 Facebook Share

The "Facebook" icon should be provided to post the current stock chart shown in the app on Facebook (similar to Homework 8), as shown in Figure 15.

When the user posts information on Facebook, a success message should be displayed on the screen. When the user cancels the Facebook dialog, a corresponding message should be displayed on the screen. See the video for more details.

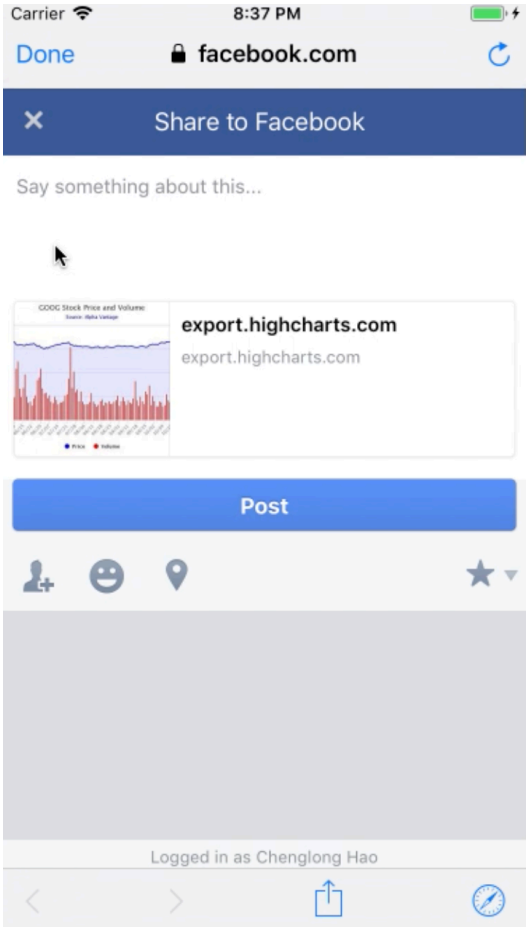

Figure 15. Facebook Share

5.7 Error handling

If for any reason (non-existing stock ticker symbols, API failure, etc.) an error occurs, an appropriate error messages should be displayed for each type of error as shown in Figures 16-19. However, the message text strings don't need to be exactly the same.

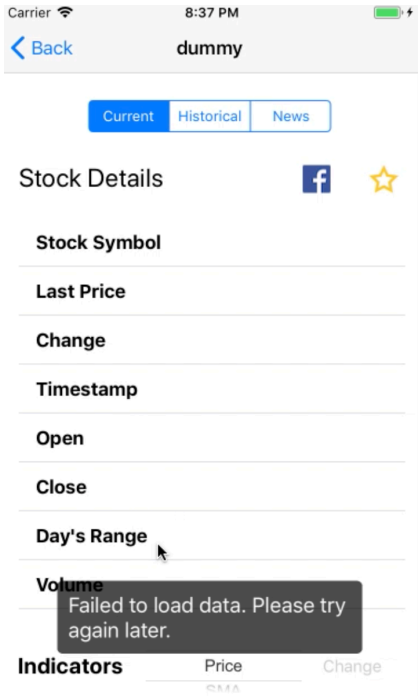

Figure 16. Display an error message if failing to load table data

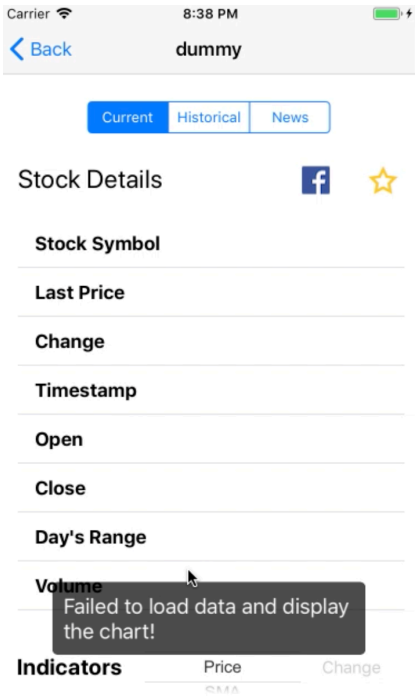

Figure 17. Display an error message if failing to load the stock chart

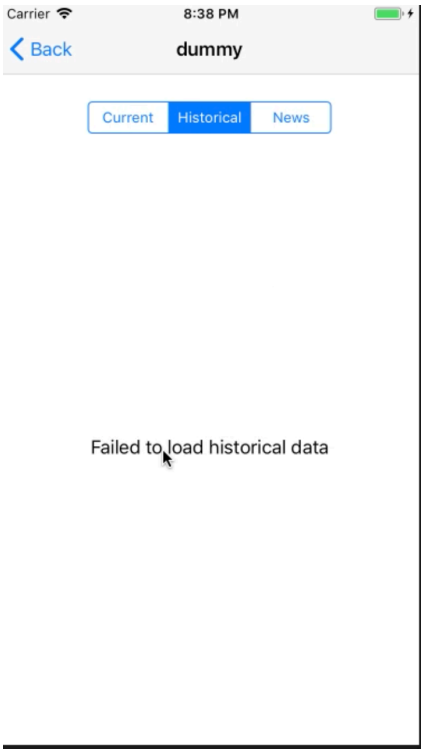

Figure 18. Show the error message in the center if failing to load the historical page

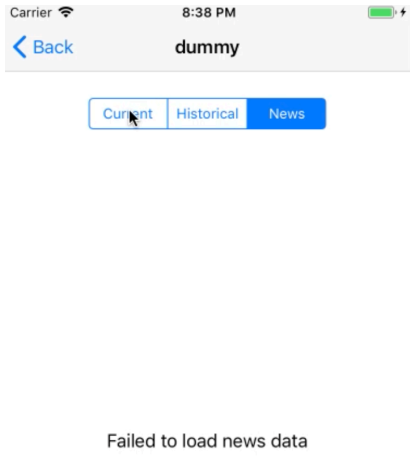

Figure 19. Show the error message in the center if failing to load the news page

5.8 Additional Specification

Find an app icon for your app, as shown at the end of the video. The icon doesn't need to be the same as that in the video. You can try to find one by yourself on https://icons8.com or somewhere else.

For things not specified in the document, grading guidelines or video, you can make your own decisions. But keep in mind about the following points:

- Always display a proper message and don't crash if an error happens.
- You can only make HTTP requests to your backend (PHP/Node.js script on AWS/GAE).
- All HTTP requests should be asynchronous.

6. Implementation Hints

6.1 Favorites list

- Use the 'UserDefaults' to store the favorited entity.
- Use a "Timer" for auto-refresh.
- Make the table view reload data when the date is updated.

6.2 UI Design

The most difficult part in UI design may be navigation among the three views: current, historical and news. The easiest way that we can think of is to use segues. In that way, you may have to request data every time are switching to another view and this is ok. But you may also have to write duplicate code in that case and have to take care of the navigation controller. A better way is to "embed" three child view controllers in the same parent view controller by using "container views". However, it's all about implementation details. You can always free to make decisions by yourself.

Another thing is about the "current" page. Basically, you need to add everything (Stock detail, chart image and indicators) to that view, and implement it as a "Scroll view", so that it's scrollable.

6.3 More about table view

Several table views are used in this app. But here they are a little more complicated than the examples in the class, because most of the table views are inside a view controller other than the TableViewController. So this means you should do more configurations about the table view by yourself. You may have to learn something about "delegate" and "dataSource" of a table view. In a TableViewController, the delegate and datasource of the tableview is the TableViewController. In a simple ViewController, however, you have to set the ViewController as the delegate and datasource of your tableview. It's very similar when you use the picker view.

About the events selecting a row or removing a row in a table view, you can try to find solutions online and there should be many good answers.

6.4 Web View

You can find how to use UIWebView easily online. Basically we load an HTML page in the web view and can execute a JavaScript method with parameters from Swift and get the return value of that method. So here are a few questions you will need to find the answer for (we give some suggestion to get you started):

- It takes time for the web view to finish loading the html page, which means you can only execute JS method after that. How can you know it has finished loading?
- How can you communicate between Swift and JS? By calling a JS method with parameters in Swift and get the return value.
- How do you communicate asynchronously when the JavaScript is making an async request? One way is to set a timer and ask the JavaScript whether it gets results every X seconds until yes.

6.5 Third-party modules

6.5.1 Auto-complete module

You can install and use third party modules by CocoaPods to implement this AutoComplete feature, such as “SearchTextField”. You can also explore and use other similar modules.

6.5.2 HTTP request module

As learned in the class, you may use “Alamofire” and “AlamofireSwiftyJSON” to make HTTP requests and parse JSON.

6.5.3 Show messages and Spinners

Install “EasyToast” by CocoaPods to display messages in your app. For the bigger spinner, install “SwiftSpinner” by CocoaPods. For the small spinner, use the “activity indicator” in Xcode.

6.5.4 Facebook iOS SDK

There are lots of documents online. If you have problems importing the SDK or using it, try to search online and solve them. There should be enough useful docs and discussions online.

7. Material You Need to Submit

Unlike other exercises, you will have to “demo” your submission “in person” during a special grading session. Details and logistics for the demo will be provided in class, in the Announcement page and in Piazza.

You should also ZIP all your source code (without image files and third-part modules) and SUBMIT the resulting ZIP file.

IMPORTANT:

All videos are part of the homework description. All discussions and explanations in Piazza related to this homework are part of the homework description and will be accounted into grading. So please review all Piazza threads before finishing the assignment.