Homework 9: Stock Search Android App with Facebook Post – A Mobile Phone Exercise

1. Objectives

- Become familiar with Android Studio, Android App development and Facebook SDK for Android.
- Add social networking features using the Facebook SDK.
- Use Android Adapters to show custom UI.
- Build a good-looking Android app using the Android SDK.

2. Background

2.1 Android Studio

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA (https://www.jetbrains.com/idea/). On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems
- ProGuard and app-signing capabilities
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

The home page of the Android Studio is located at:

http://developer.android.com/tools/studio/index.html

2.2. Android

Android is a mobile operating system initially developed by Android Inc., a firm purchased by Google in 2005. Android is based upon a modified version of the Linux kernel. As of December 2013, Android was the number 1 mobile OS, in unit sales, surpassing iOS, while iOS was still the most profitable platform.

The Android operating system software stack consists of Java applications running on a Java based object oriented application framework on top of Java core libraries running on the Dalvik virtual machine featuring JIT compilation.

The Official Android home page is located at:

http://www.android.com/

The Official Android Developer home page is located at:

http://developer.android.com/index.html

2.3 Facebook

Facebook is a social networking service launched in February 2004, owned and operated by Facebook, Inc. Users can add friends and send them messages, and update their personal profiles to notify friends about themselves and what they are doing.

Users can additionally post news feeds to their profiles, and these feeds may include images, besides text messages.

The Facebook homepage is available at:

http://www.facebook.com

Facebook provides developers with an application-programming interface, called the Facebook Platform.

2.4 Markit on Demand

Markit on Demand API provides detailed description about the stock information of company as well as historical stock values. You can refer to the API description on the following link:

http://dev.markitondemand.com/MODApis/

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 7.5 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 the 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 the use of the following components:

1. This homework requires the use of the following components:

   A. **Download and install Android Studio**. You may use any other IDE other than Android Studio such as Eclipse, but you will be on your own if problems spring up.
   B. First you need to install Java on your local machine. You can download JDK 8 from - http://www.oracle.com/technetwork/java/javase/downloads/index.html. For windows users, after installing the JDK, you need to add environment variables for JDK.
      - Properties -> Advanced -> Environment Variables -> System variables -> New Variable

        Name: JAVA_HOME, Variable Value: <Full path to the JDK>

      - Typically, this full path looks something like C:\Program Files\Java\jdk1.8.0.

        Then modify the PATH variable as follows on Microsoft Windows:
        C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Java\jdk1.8.0\bin
        This path may vary depending on your installation.

      - Note: The PATH environment variable is a series of directories separated by semicolons (;) and is not case-sensitive. Microsoft Windows looks for programs in the PATH directories in order, from left to right. You should only have one bin directory for a JDK in the path at a time. Those following the first instance are ignored. If you are not sure where to add the path, add it to the right of the value of the PATH variable. The new path takes effect in each new command window you open after setting the PATH variable.

      - Reboot your computer and type "java -version" in the terminal to see whether your JDK has been installed correctly.
      - Note: on a Mac with macOS, the path will be set up slightly differently. See the Java and Android Studio documentation.

Set up the Android Studio environment so that you can run any sample android app on your phone/tablet/virtual device from it. Then you can start with this homework app. You will need to enable "Developer Options" and "USB debugging" if you are using an actual device. There are endless resources a simple search away on how to setup your Android Studio.

2. You also need to create a Facebook Developer application as you did for your homework 8. Follow the following steps to get started:

    a. **Download SDK**: Download the latest Facebook Android SDK Link: https://developers.facebook.com/docs/android

    b. Instructions to **import in Android Studio**: https://developers.facebook.com/docs/android/getting-started

    c. Create a **new app on Facebook** developer: https://developers.facebook.com/apps/

    d. Specify **App Info** related to the HW9 android application you are developing.

    e. **Key Hashes**: Specify Android key hash for the development environment using the commands mentioned.

    f. Track App Installs and App Opens: Not required.

    g. Next Steps: Utilize **Login** (optional) and **Share** tutorials to achieve the functionality required for the exercise. Note: In your Facebook application settings, you should go to the "Status & Review" section and choose "Yes" for the question "*Do you want to make this app and all its live features available to the general public*?" as you did for homework 8.

4. High Level Design

This homework is a mobile app version of HW8. In this exercise, you will develop an Android Mobile application, which allows users to search for stock information, save some stock symbols as favorites, and post to the 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 retrieved.

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.

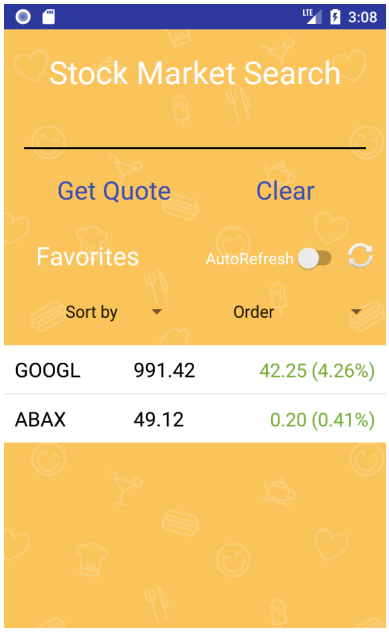

Figure 1. Search Form

Once the validation is successful, we would get the stock details using our PHP script hosted on Amazon Web Services/Google App Engine, which would return the result in JSON format. We would display the stock details in a ListView component in the 'Current' tab. Furthermore, our PHP script would be responsible for rendering the HighCharts in the 'Current' and 'Historical' tabs and also rending the news articles in the 'News' tab.

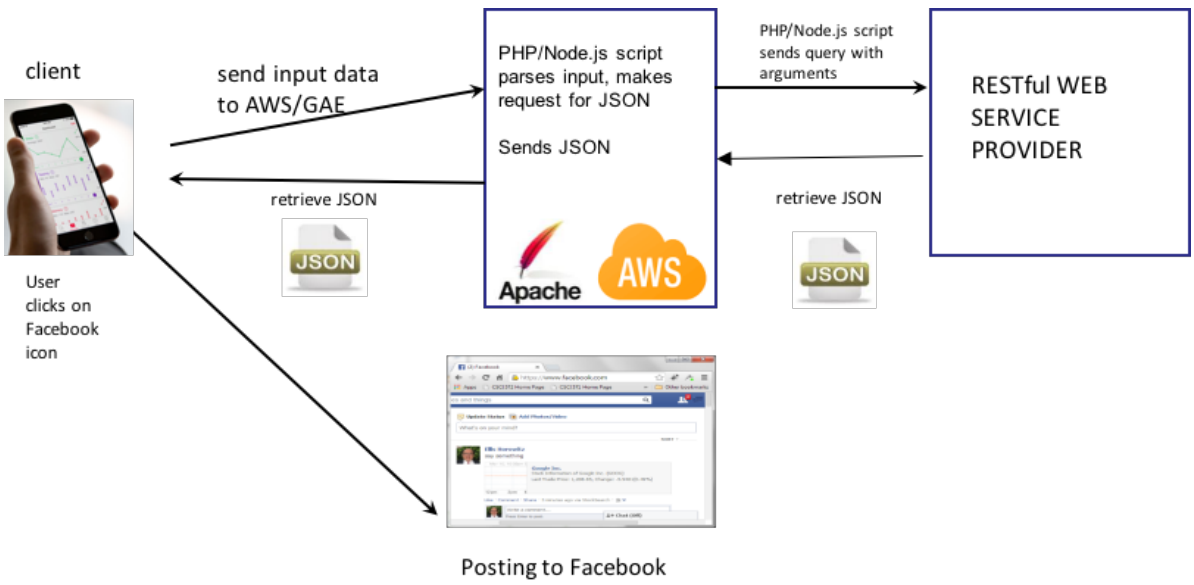

Figure 2. Architecture Overview

5. Implementation

5.1 Search Form

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

The interface consists of the following:

- An 'AutoCompleteTextView' component allowing the user to enter the company name or symbol.
- Two TextViews implemented as buttons for interaction in the Search Form.
  - A button 'Clear' to clear the 'AutoCompleteTextView' component.
  - A button 'Get Quote to get the quote, after validation.
- A Switch implemented as an AutoRefresh element
- Next to the switch, an Android icon to refresh on-click.
- A Spinner listing options to sort the list.
- A Spinner listing options to order the list.
- The Favorite ListView showing the list of favorite stocks.
- The Favorite List starts with an empty favorite list.

The form has two buttons:

a) **Get Quote**: Validations are first performed, when the button is clicked. If the validations are successful, then the stock details would be fetched from the server (either hosted on AWS or GAE). However, if the validations are unsuccessful, appropriate messages would be displayed and no further requests would be made to the server.
b) **Clear**: This button would clear the 'AutoCompleteTextView' and clear any validations error, if present.

5.1.1 AutoComplete

The user can enter the stock name or symbol in the text view to get the stock information from our PHP script. Based on the user input, the AutoComplete would display the all the matching companies and symbols (see Figure 2) by making a HTTP request. The auto-complete dropdown is shown only when the user has typed in at least one character and the maximum number of results displayed in the auto-complete dropdown is 5. This needs to be implemented using AutoCompleteTextView.

To get the data used for auto-complete suggestions, you need to make http requests to your PHP/Node.js script which is in AWS/GAE.

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 autocomplete dropdown then disappears.

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

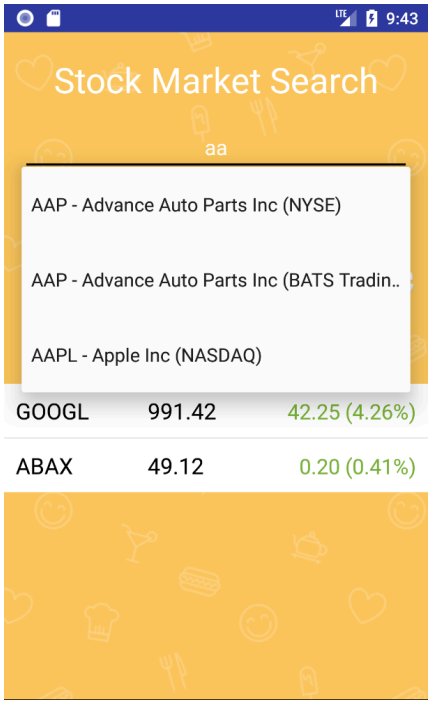

Figure 3: AutoComplete Suggestions

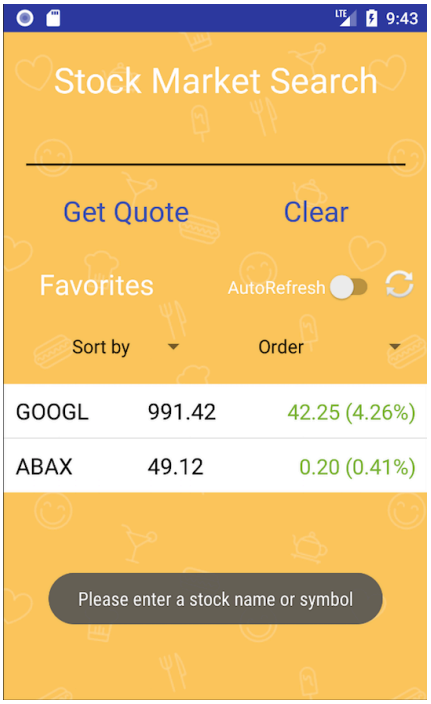

Figure 4: Validation

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 an appropriate message to indicate the error should be displayed, 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 Favorite List

The Favorites list interface consists of the following:

- An Automatic Refresh switch, labeled AutoRefresh
- A Refresh button
- Two "Spinners" controlling the order of the list
- The Favorite "Custom ListView" showing a list of favorite stocks

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

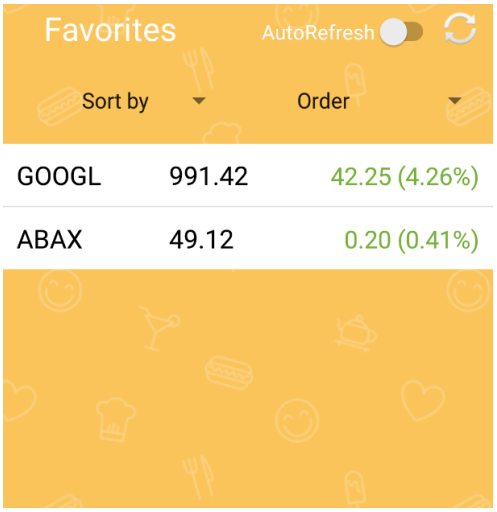

Figure 5: Favorite Stocks

Here are some important points about this feature:

- Display **symbol**, **stock price** and **change (change percent)** in 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.
- Long press a row and display a Context Menu to Delete list item. Then user can then remove that stock from the Favorites list, as shown in Figure 7.

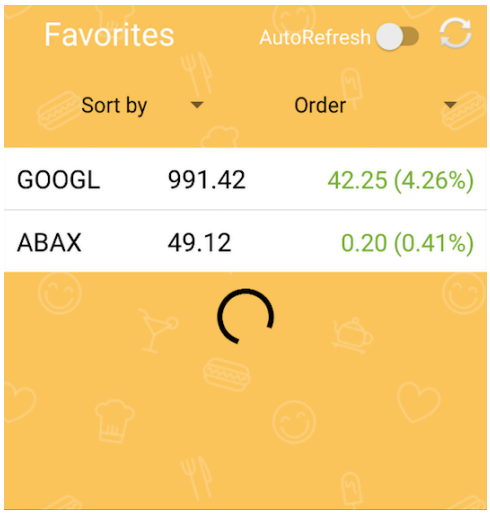

Figure 6: Display an indicator while loading data

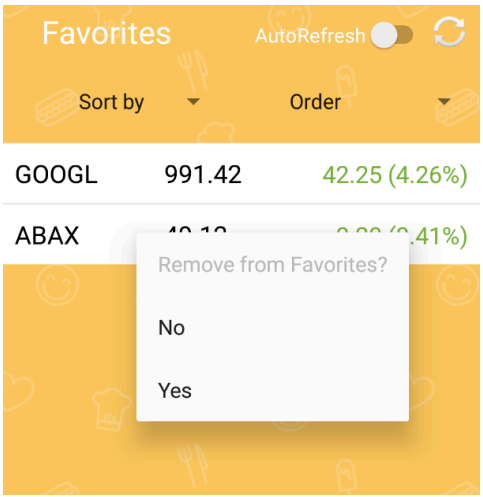

Figure 7: Long press an item to generate context menu. Yes = delete, No = do nothing

5.3 Stock Details

When the user clicks the Get Quote button, your app should display a loading image 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 "Tabbed Activity" 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 list showing all the stock values. The list of the items in the stock details would be implemented using a 'ListView'. The following stock values should 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.

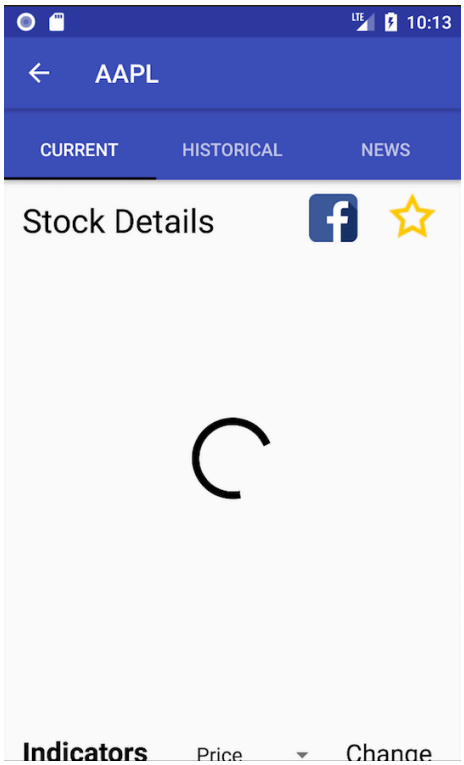

Figure 8: Display an image while loading

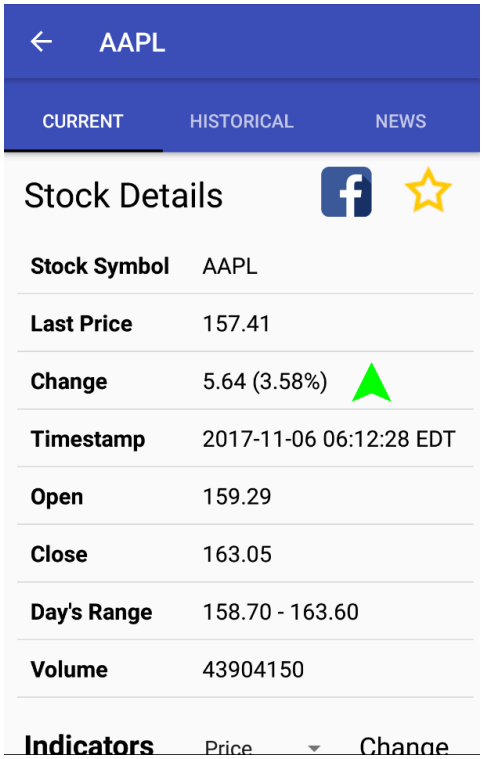

Figure 9: Stock details

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 Spinner and a TextView/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 "WebView" 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 Android code and the JS code asynchronously. It's NOT allowed to block the app while waiting for the chart to be shown in the WebView.
- Whenever the chart in the WebView is in a loading state, you should hide the previous chart (if there's any) and display a loading icon.
- Display a proper message if there is any failure in retrieving a chart.

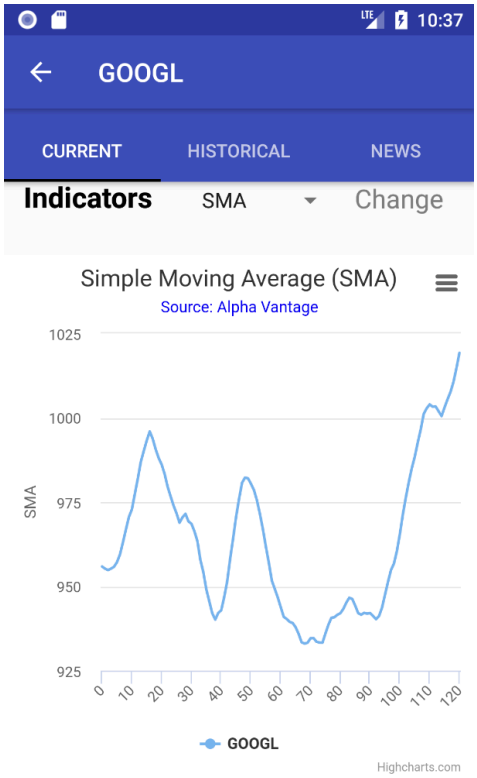

Figure 10: SMA Chart

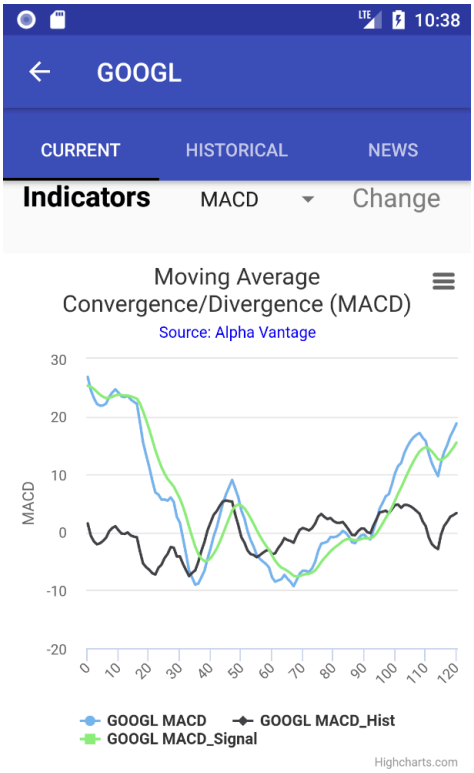

Figure 11: MACD chart

5.4 Historical Charts

This tab represents the historical charts showing the value of the stock in the past. It has to be rendered as per Figure 12.

Figure 12: Historical Charts

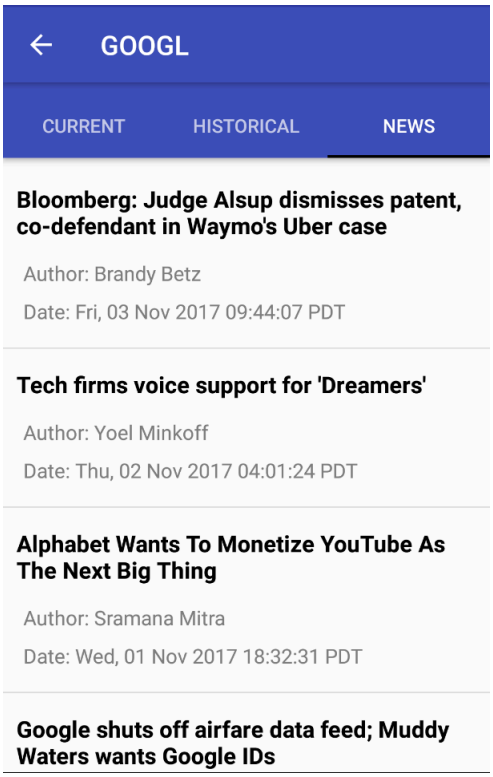

Figure 13: News Feed

This chart should also be implemented with a WebView. 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 Android 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 13. The news articles need to be implemented in a 'Custom ListView'. Display title, author and date for each article. The app should open the article in the browser window (Mobile browser, default is Chrome) 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 14 and Figure 15. Either of these UIs would work.

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.

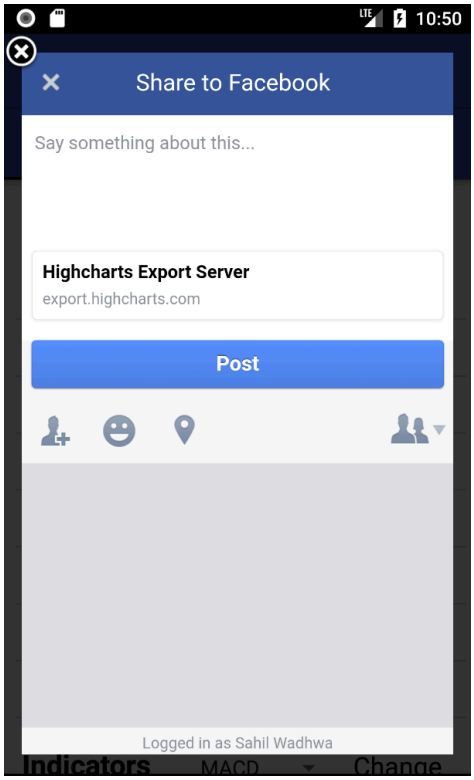

Figure 14. Facebook share

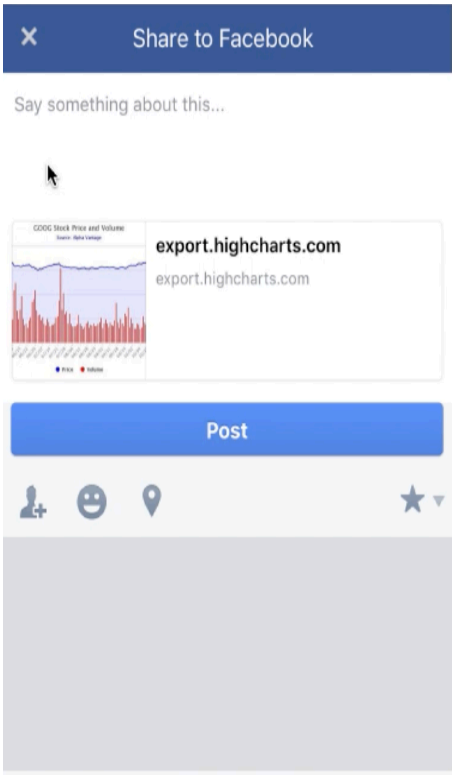

Figure 15. Facebook share

5.7 Error Share

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 16-18. However, the message text strings don't need to be exactly the same.

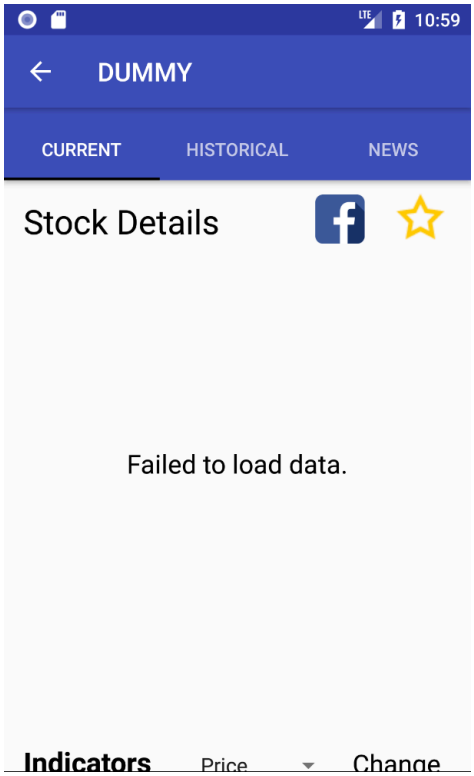

Figure 16. Display an error message if failing to load data in "Current" tab

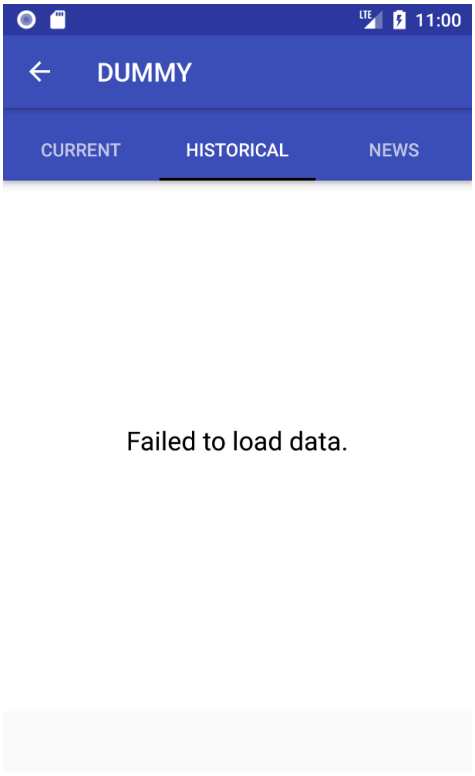

Figure 17. Display an error message if failing to load data in "Historical" tab

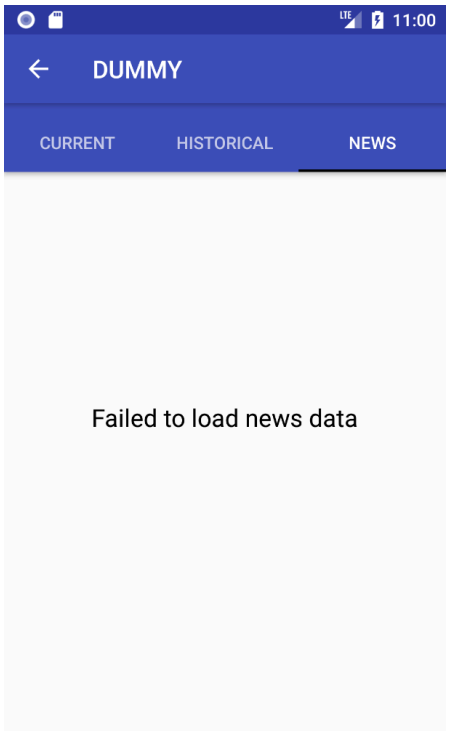

Figure 18. Display an error message if failing to load data in "News" tab

5.8 Additional Specification

Use the app icon provided for your app, as shown in the video.

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 'Shared Preferences" to store the favorited entity.
- Use a "Timer" or "Handler" for auto-refresh.
- Prefer using Array Adapter rather than Base Adapter for Custom ListView.
- Make the Favorites Custom ListView read data from Shared Preferences. Whenever this activity is loaded, view gets created again and so the updated Shared Preferences data will be used always.
- For sorting and ordering, make use of Java's Comparator function.

6.2 UI Design

Use a Tabbed Activity to create navigation between three tabs of details section: current, historical and news. Each tab should be implemented as a Fragment. Please note that Fragment has a different life cycle and so access to the context and UI components will happen in different life cycle events than the main Activity. You will request data inside each fragment. Here a Tabbed Activity will be the parent Activity with 3 child fragments attached to it. However, it's all about implementation details. You are always free to make decisions by yourself as long as the applications works.

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.

One issue that mostly troubles everyone is the timing of receiving data from volley and then updating the view. Since rendering of a view happens much faster than an http call, after receiving response from an API, you will have to refresh the UI. Make use of notifyDataSetChanged() in your adapter implementation.

6.3 More about table view

Several Custom ListViews are used in this app. But here they are a little more complicated than the examples in the class, because most of the custom listviews are inside a fragment than inside an Activity. So this means you should create proper models and layouts in XML for each Custom ListView. You may have to learn about creating custom listviews from Volley. About the

events selecting a row or removing a row in a list view, you can try to implement setOnItemClickListener to handle such events.

6.4 Web View

You can search how to use Android WebView. Basically, we load an HTML page in the web view and can execute an Android method from JavaScript code and vice-versa using JavascriptInterface. Read more about it online.

6.5 Third-party modules

6.5.1 Auto-complete module

You can make use of ArrayAdapter and set this adapter to the view. No third party libraries needed.

6.5.2 HTTP request module

As learnt in the class, you may use "Volley" to make HTTP requests and parse JSON.
For making calls and parsing XML responses, make use of XMLPullPrser.

6.5.3 Show messages and Spinners

Make use of "Toast" to display messages in your app. For loading icon, make use of ProgressBar in your app.

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.

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 your Android source directory and SUBMIT the resulting ZIP file (without image files and third-part modules). Make sure that the source path does not include the .apk binary 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.