Homework 9
Stock Trading App

Prof. Marco Papa

Developed and Designed by
Rushabh Kapadia

1. Objectives

- Become familiar with Java, JSON, Android Lifecycle and Android Studio for Android app development.
- Build a good-looking Android app.
- Learn the essentials of Google's Material design rules for designing Android apps
- Learn to use the tiingo APIs and the Android SDK.
- Get familiar with third party libraries like Picasso, Glide and Volley.

The objective is to create an Android application as specified in the document below and in the video.

2. Background

2.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android application development, based on IntelliJ IDEA - a powerful Java IDE. 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.

More information about Android Studio can be found 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 Nov 2018, Android was the number 1 mobile OS, in unit sales, surpassing iOS, while iOS was still the most profitable platform.

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/

2.3 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/

2.4 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/

2.5 Microsoft Azure

The Azure cloud platform is more than 200 products and cloud services designed to help you bring new solutions to life—to solve today's challenges and create the future. Build, run, and manage applications across multiple clouds, on-premises, and at the edge, with the tools and frameworks of your choice.

To learn more about the Azure services, visit this page:

https://azure.microsoft.com/en-us/solutions/

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

https://docs.microsoft.com/en-us/azure/devops/pipelines/targets/webapp?view=azure-devops&tabs=yaml#deploy-a-javascript-nodejs-app

3. Prerequisites

This homework requires the use of the following components:

- Download and install Android Studio. Technically, you may use any IDE other than Android Studio such as Eclipse, but the latest SDKs may not be supported with Eclipse. We will not be providing any help on problems arising due to your choice of alternate IDEs.
- You must use the **emulator**. Everything should just work out of the box.
- If you are new to Android Development, Hints are going to be your best friends!

4. High Level Design

This homework is an extended mobile app version of Homework 6 and Homework 8.
In this exercise, you will develop an Android application, which allows users to search for different stock symbols/tickers and look at the detailed information about them. Additionally, the users can trade with virtual money and create a portfolio. Users can also favorite stock symbols to track their stock prices. The App contains 2 screens: Home screen and the Detailed Stock Information screen. However, the App has multiple features on each of these screens.

This homework contains 5 API calls. There are 4 calls to the tiingo APIs for company description, stock prices, autocomplete and chart data points, and the additional newsapi endpoint. Each of these 5 API calls are the same as Homework #8. So, you can use the same Node.js backend as HW8 with an additional call added as a new endpoint. In case you need to change something in Node, make sure you do not break your Angular assignment (or deploy a separate copy) as the grading for homework will not be finished at least until 1 week later.

You must use Java strictly. Kotlin is not allowed.

PS: This app has been designed and implemented in a Pixel 2XL emulator by using SDK API 29. It is highly recommended that you use the same virtual device and API to ensure consistency.

Demo will be on an emulator using Zoom Remote Control, no personal devices allowed, see the rules:

https://csci571.com/courseinfo.html#homeworks

5. Implementation

5.1 App Icon and Splash Screen

In order to get the app icon/image, please see the **hints** section.
The app begins with a welcome screen (Figure 2) which displays the icon provided in the hint above.

This screen is called Splash Screen and can be implemented using many different methods. The simplest is to create a resource file for the launcher screen and add it as a style to AppTheme.Launcher (see **hints**).

This image is also the app icon as shown in Figure 1

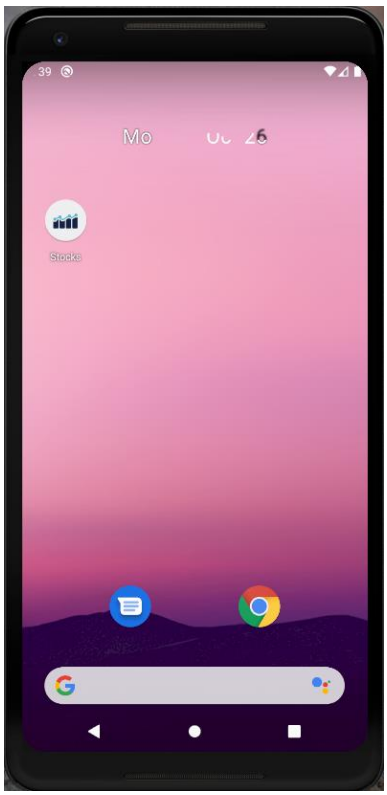


Figure 1: App Icon

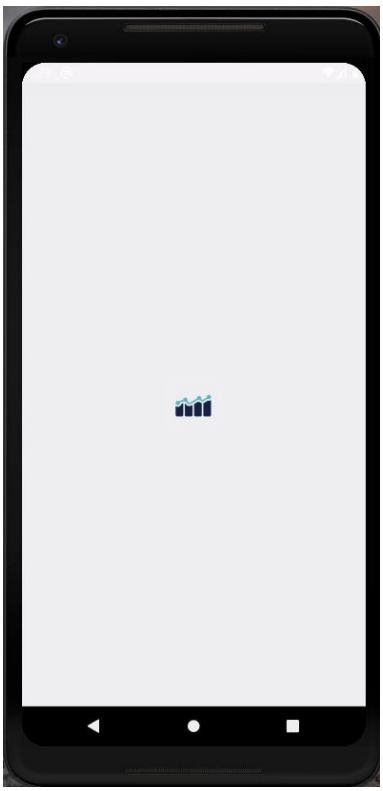


Figure 2: Splash Screen

5.2 Home screen

When you open the app, there will be an initial spinner while the data is being fetched using volley as shown in Figure 3. The home screen will have a toolbar at the top with Stocks title and the search icon. Below that, it will show the current date as shown in Figure 3.

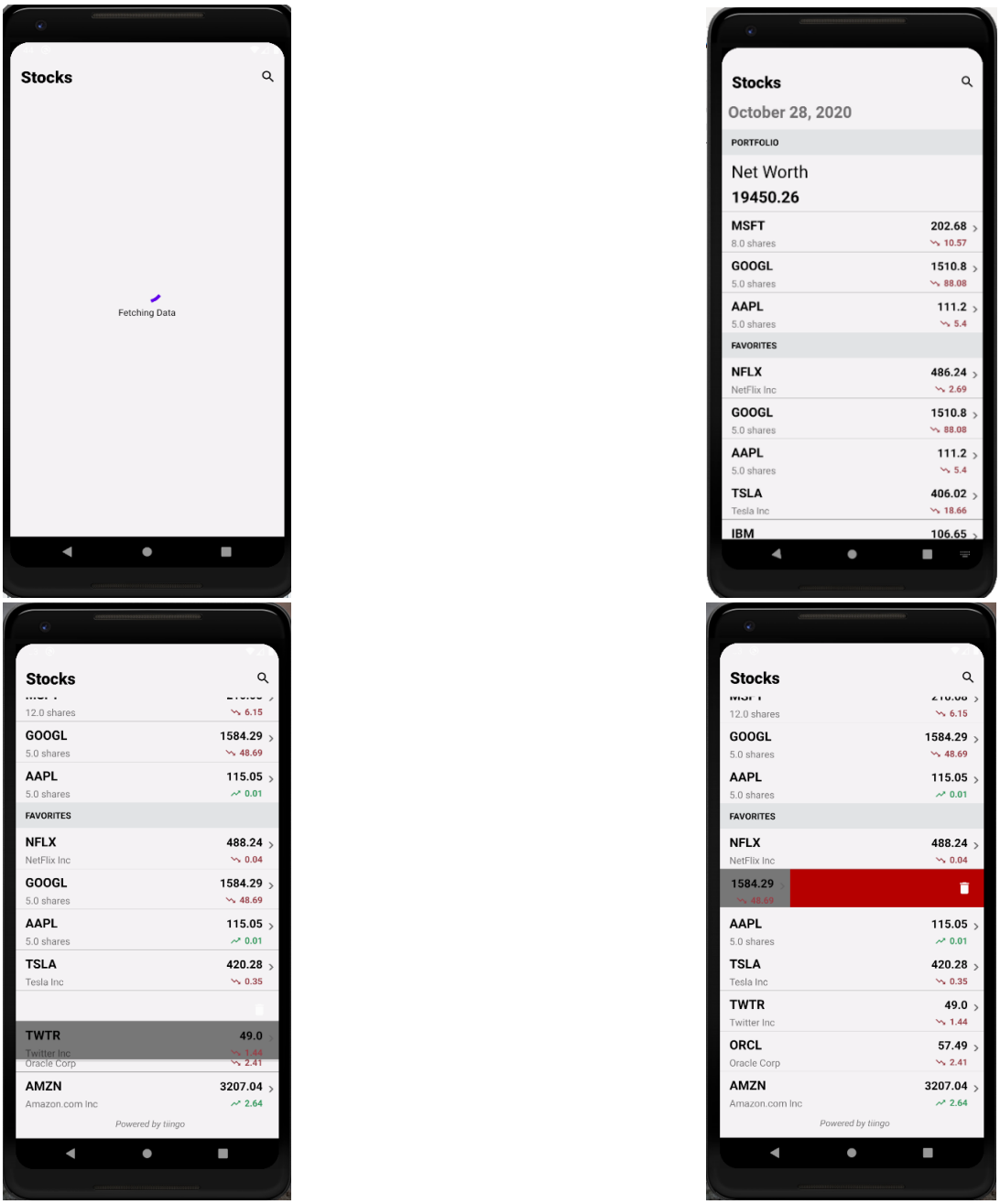


Figure 3: Home screen

There are 2 sections on the home screen:

- ***Portfolio Section*** - This section will show the total net worth of the user, which is calculated as the sum of number of shares of a stock multiplied by the current price, plus uninvested cash. This is followed by the list of stocks in the user portfolio with their current price, change in price and total shares owned information.

- ***Favorites Section*** - This section will show all the stocks that have been favorited by the user to allow the user to easily check the prices of stocks in their watchlist. The stock symbol, current price, change in price and company name should be displayed as shown

in Figure 3. In case the favorited stock is present in the user portfolio, instead of the company name, the stocks owned should be displayed.

Additionally, the symbol next to the change in price value should either be trending down or up based on the change price value. For example, if the change in price is positive, a trending up symbol should be displayed and the symbol as well as the change price value should have green color. In case the change in price is zero i.e. no change, no symbol should be displayed, and the change price value should be grey.

Each stock listing also has a button on the extreme right, next to the current price field. On clicking the button or the stock listing, the detailed information screen will open for the selected stock.

The home screen view should support multiple functionalities like:

- The **swipe to delete functionality** allows the user to remove/delete the stock from the favorite section. On removing a stock from the favorite section, the stock should be removed from the favorite stocks in the local storage and the view.
- The **drag and reorder functionality** allows the user to reorder the stocks in either section. The user should be able to long press the stock listing and drag it to the new position. The list should be updated accordingly to ensure the new order going forward. Note: The user cannot drag the stock from the favorite section into the portfolio section. The stock can only be dragged and dropped in the same section.

At the bottom of the 2 sections, we have a 'Powered by tiingo' text in italic. On clicking this text, the App should open the Tiingo homepage in chrome. (The URL should be: https://www.tiingo.com/)

The field mappings for the stock information is the same as Homework #8.

Note: The price information for each stock should be updated every 15 seconds, same as Homework #8. Additionally, the progress spinner should only be shown whenever the home screen is opened initially or when navigating back from the detailed stock screen. While refreshing the stock price every 15 seconds, the spinner SHOULD NOT be displayed.

The home screen has been implemented by using a **RecyclerView** with the **SectionedRecyclerViewAdapter.** Each of the stock listings has been implemented using **ConstraintLayout, TextView, ImageView.**

The **Search** button on the toolbar opens the search bar to type the stock symbol to search. The search bar uses the **autocomplete** functionality

5.2.1 Search Functionality

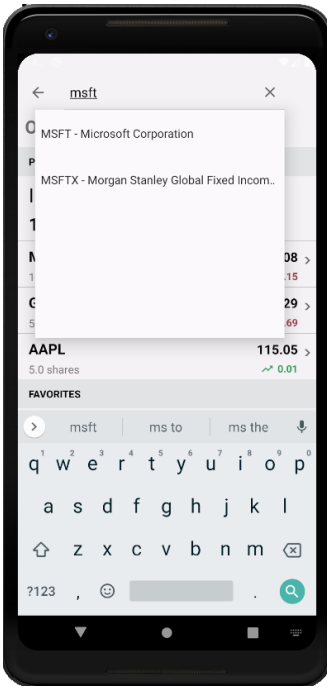


Figure 4: Search Functionality

- On top right side, there will be a search button which opens a textbox where the user can enter a keyword to search for a stock symbol. See hints for icon.
- The user is provided with suggestions of keywords using the Tiingo Autocomplete API. (Same as homework #8)
- When the user taps on a suggestion, it is filled inside the search box and clicking enter/next takes the user to the detailed information screen.
- Before you get the data from your backend server, a progress bar should display on the screen as indicated in the detailed information section.
- The user can only search for valid stock symbols in the search bar. The search should redirect to the detailed information screen only if the user selected one of the autocomplete suggestions to fill the search field.
- In the Autosuggest, only make an API call after the user enters 3 characters. Example: “am” should not display any suggestions but “ama” should have suggestions. Refer to the video for better understanding.

This component involves 2 things:

- Implementing a searchable – see hints
- Implementing autocomplete – see hints

5.3 Detailed Stock Information Screen

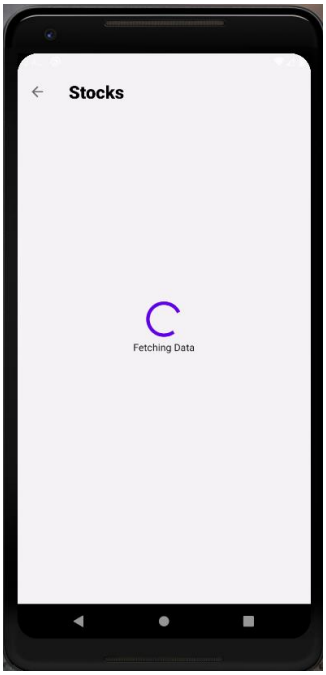


Figure 4: Loading

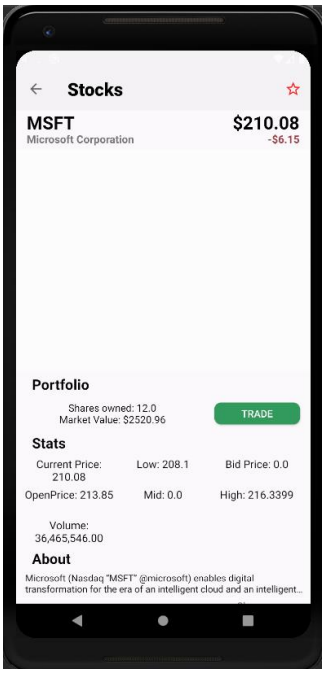


Figure 5: Detailed

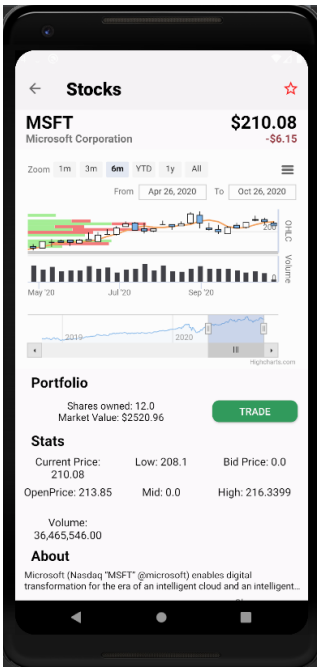


Figure 6: Detailed

On clicking the **Goto** button on any stock listing or searching for a stock symbol, the loading spinner symbol should be displayed while the details are being fetched (see Figure 4). Once the data has been fetched except the chart (since the chart takes longer to load), the spinner should disappear and information regarding the stock should be available to the user (Figure 5 and Figure 6).

The top action bar should have the 'Stock' title and the back button to go back to the home screen (which has the filter values that were used for the current search if triggered by using the search functionality). The action bar should also contain a favorite icon to add or remove the stock from favorites. The favorite icon will either be filled or bordered based on whether the stock is favorited or not. Adding/Removing the stock from favorites should also display a toast message as shown in the video.

Below the action bar, there should be 4 fields: stocks symbol, current price with '$' sign, company name and the change price with '$' sign (the text color should be green, red or grey based on the change price value being positive, negative or zero respectively). The App then has a **WebView** element which is blank till the chart loads. (More details later in this section)

The **Portfolio section** allows the user to trade the shares of the stock. It contains a left section which shows the market value of the stock in the user portfolio and the number of shares the user owns. The right section contains the trade button. Initially, when the user starts the app for the first time, they will not have any stocks/shares in the portfolio and an initial pre-loaded amount of **$20,000** to trade on the app. This amount can change based on the trading done by the user. (For example, if the user sells shares at a loss, it can become less than 20,000 and so forth)

If the user does not own any shares of the given stock, the left section will have the message as shown in Figure 7, else the left section will look like Figure 6.

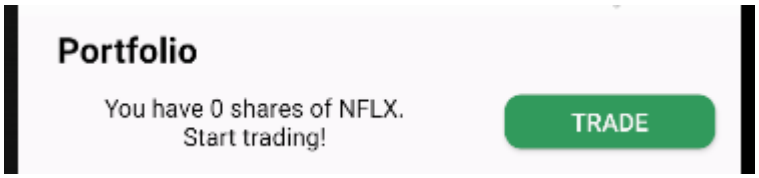


Figure 7: Portfolio section

The **Stats section** displays the trading statistics for the given stock in a grid. The grid has 7 fields namely: Current price, Low, Bid price, Open price, Mid, High and Volume. If any of these fields are missing in the JSON, set them as 0.0. The *GridView element* is to be used for this section.

The **About section** displays the description of the company. If the description is longer than 2 lines, ellipsize the end of the 2nd line and display a 'Show more…' button. On clicking this button, the complete description becomes visible and the button text changes to 'Show less'. (Figure 8) If the description is less than 2 lines, do not display the button.

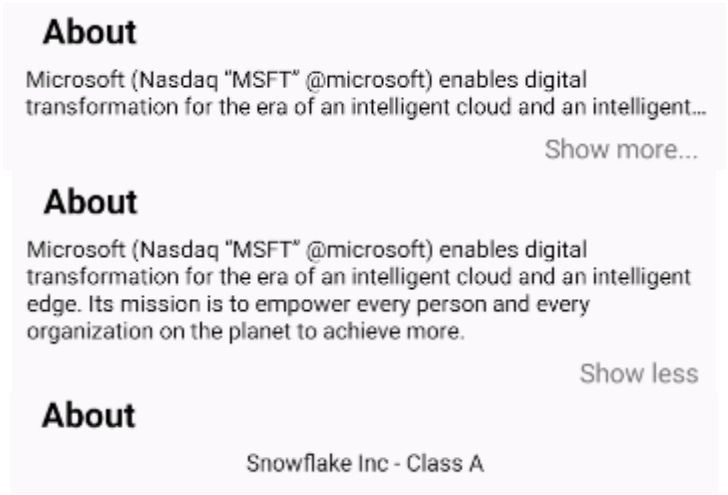


Figure 8: About section

The API response used is the same **as HW8.**

The **News section** displays the news articles related to the given stock symbol. The first article has a different format/layout than the rest of the articles in the list. On clicking the news article, the original article is opened in chrome using the article URL. On long press, a dialog box opens with options to share on twitter and open in chrome. (Figure 9) Refer the hints section for help in setting up twitter share and open in chrome functionalities. For each article, the information displayed is Article source, Article title, Article image and the time ago when the article was published. The time ago should support 'days ago' and 'minutes ago' by calculating the difference between the timestamp the article was published and the current timestamp. (see the hints)
The news section uses **RecyclerView** and **ArticleDialog** elements.

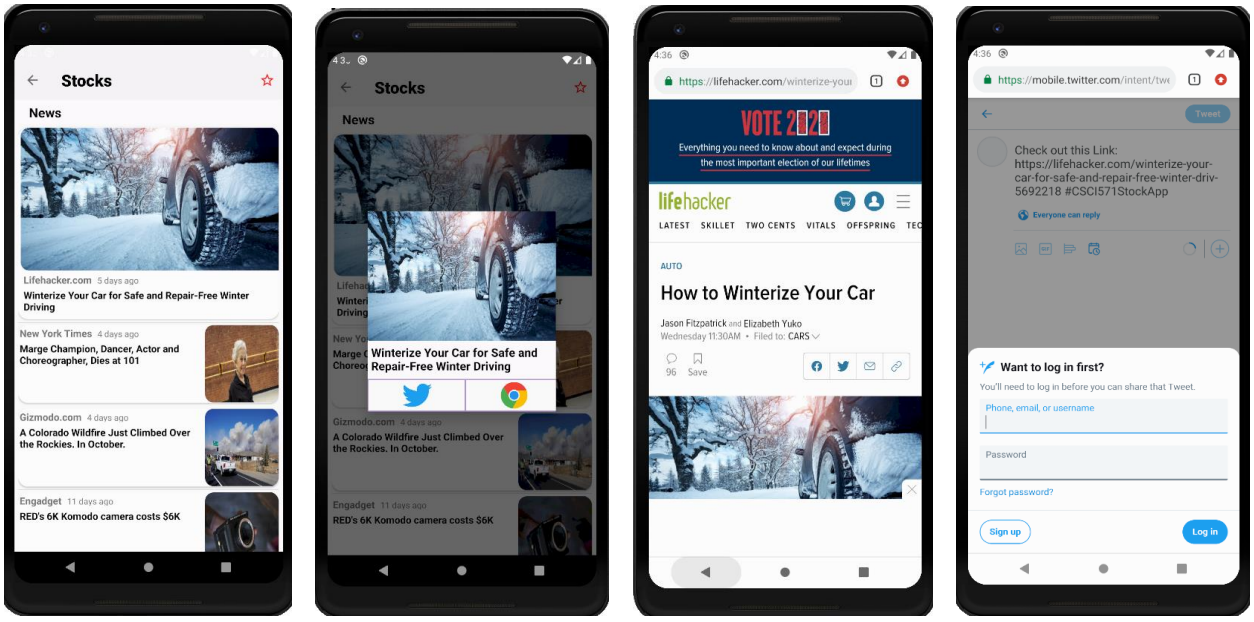
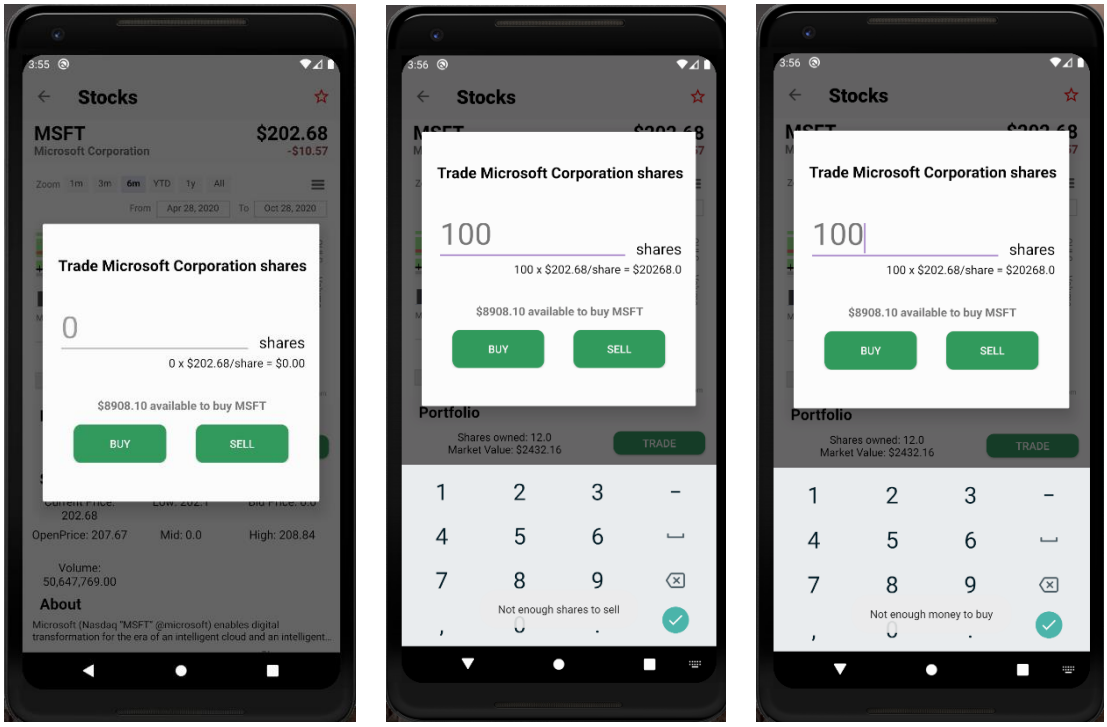


Figure 9: News section

From every twitter button, on clicking the button, the article should be shared by opening a browser with Twitter Intent. Use the following link to implement the Twitter Intent: https://developer.twitter.com/en/docs/twitter-for-websites/tweet-button/guides/web-intent



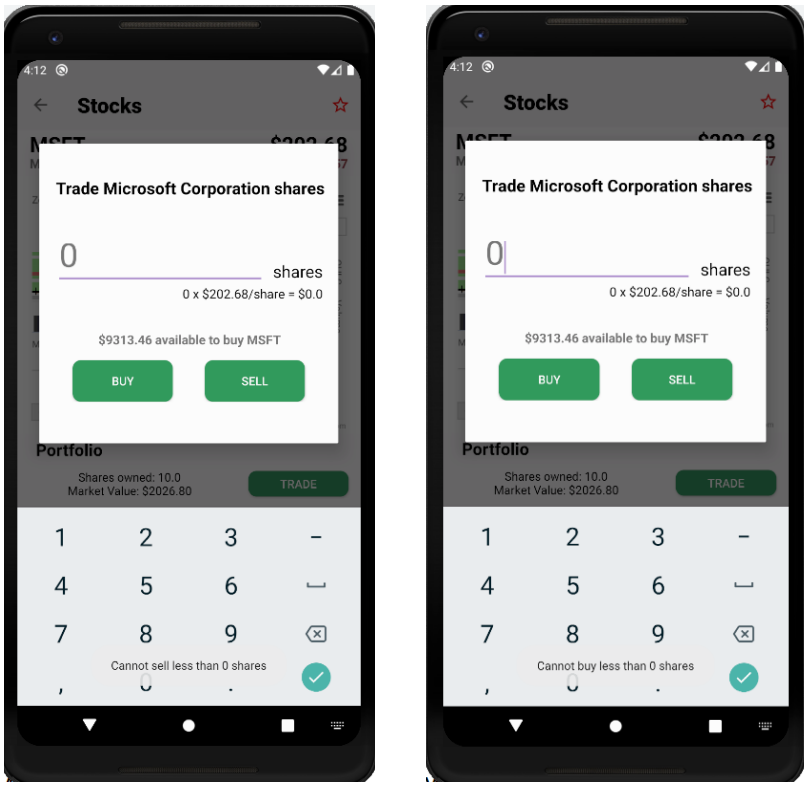


Figure 10: Trade Dialog

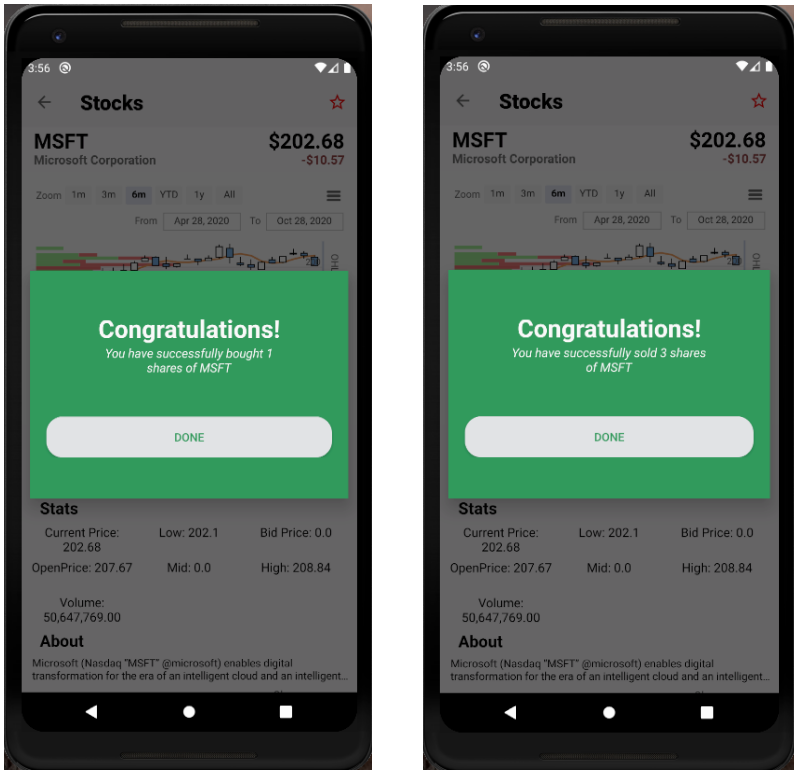


Figure 11: Trade Success Message

The Trade button in the **Portfolio section** opens a new dialog box for trading (Figure 10). The dialog shows an input box which only accepts numeric input. Below the input field, there is a calculation text box which updates based on the numeric input to display the final price of the trade. The trade dialog also displays the current available amount to trade for the user. The user can either buy or sell the shares. Based on the trade, the amount available to trade will be updated accordingly. There are 5 error conditions to be checked before executing the trade and displaying the trade successful dialog (Figure 11). The error conditions are:

- ***Users try to sell more shares than they own*** - The trade dialog box should remain open and a toast message with text 'Not enough shares to sell' should be displayed.
- ***User tries to buy more shares than money available -*** The trade dialog box should remain open and a toast message with text 'Not enough money to buy' should be displayed.
- ***User tries to sell zero or negative shares -*** The trade dialog box should remain open and a toast message with text 'Cannot sell less than 0 shares' should be displayed.
- ***User tries to buy zero or negative shares -*** The trade dialog box should remain open and a toast message with text 'Cannot buy less than 0 shares' should be displayed.
- ***User enters invalid input like text or punctuations -*** The trade dialog box should remain open and a toast message with text 'Please enter valid amount' should be displayed.

5.3.1 HighCharts in Android

The **Chart section** in the detailed stock information screen uses a *WebView* element to load the HighCharts stock chart (same as homework #8). It provides the same features as Homework #8 and the JavaScript code for loading the chart can be reused. To load the chart, the App will load a local HTML file with the necessary JavaScript to request the data from the NodeJS server and display the chart when the data is fetched (See the hints).

Note: All the Images needed to display the section heading icons as well as the 3 tabs are provided in the zip file for Homework #9.

5.4 Progress bar

Every time the user has to wait before they can see the data, you must display a progress bar as shown in Figure 4. The progress bar is to be present across the **Home screen, Detailed Stock screen** and just says "Fetching Data…". One idea would be to display the progress bar by default (where needed) and then hide it as soon as the data is received/view is prepared. See hints for progress bar related ideas and styling.

Note: Based on your implementation, you might need to put progress bars on different places. During the demo, there should NOT be any screen with default/dummy/placeholder values or an empty screen.

5.5 Summary of detailing and error handling

1. Make sure there are no conditions under which the app crashes
2. Make sure all icons and texts are correctly positioned as in the video/screenshots
3. Make sure the screens and toasts are correctly displayed.
4. Make sure there is a progress bar every time there is nothing to show
5. Make sure the styling for different features matches the video/screenshots.
5. All API calls are to be made using Node.js backend.

5.6 Additional Info

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

- Always display a proper message and do not crash if an error happens.
- You can only make HTTP/HTTPS requests to your backend Node.js on GAE/AWS/Azure.
- All HTTP requests should be asynchronous and should not block the main UI thread. You can use third party libraries like Volley to achieve this in a simple manner.

6. Implementation Hints

6.1 Icons

The images used in this homework are available in the zip file mentioned in the Piazza post for Homework #9.
You can choose to work with xml/png/jpg versions. We recommend using xml as it is easy to modify colors by setting the Fill Colors.

6.2 Third party libraries

Sometimes using 3rd party libraries can make your implementation much easier and quicker. Some libraries you may have to use are:

6.2.1 Volley HTTP requests

Volley can be helpful with asynchronous http request to load data. You can also use Volley network ImageView to load photos in Google tab. You can learn more about them here:
https://developer.android.com/training/volley/index.html

6.2.2 Picasso

Picasso is a powerful image downloading and caching library for Android.
http://square.github.io/picasso/

If you decide to use RecycleView to display the photos with Picasso Please use version 2.5.2 since the latest version does not support RecycleView well.
https://github.com/codepath/android_guides/wiki/Displaying-Images-with-the-Picasso-Library

6.2.3 Glide

Glide is also a powerful image downloading and caching library for Android. It is similar to Picasso. You can also use Glide to load photos in Google tab.
https://bumptech.github.io/glide/

6.3 Working with action bars and menus

https://developer.android.com/training/appbar/setting-up
https://stackoverflow.com/questions/38195522/what-is-oncreateoptionsmenumenu-menu

6.4 Displaying Progress Bars

https://stackoverflow.com/a/28561589
https://stackoverflow.com/questions/5337613/how-to-change-color-in-circular-progress-bar

6.5 Implementing Splash Screen

There are many ways to implement a splash screen. This blog highlights almost all of them with examples:
https://android.jlelse.eu/the-complete-android-splash-screen-guide-c7db82bce565

6.6 Adding the App Icon

https://dev.to/sfarias051/how-to-create-adaptive-icons-for-android-using-android-studio-459h

6.7 Adding ellipsis to long strings

https://stackoverflow.com/questions/6393487/how-can-i-show-ellipses-on-my-textview-if-it-is-greater-than-the-1-line

6.8 Adding a button to ActionBar

https://developer.android.com/training/appbar/actions
https://stackoverflow.com/questions/12070744/add-back-button-to-action-bar

6.9 Implementing a Recycler view in android

https://developer.android.com/guide/topics/ui/layout/recyclerview
https://stackoverflow.com/a/40217754
https://abhiandroid.com/materialdesign/recyclerview-gridview.html

6.10 Adding Toasts

https://stackoverflow.com/questions/3500197/how-to-display-toast-in-android

6.11 Passing variables to intent

https://stackoverflow.com/questions/2405120/how-to-start-an-intent-by-passing-some-parameters-to-it

6.12 Formatting Text using HTML in TextView

https://stackoverflow.com/a/27462961

6.13 Open Link in browser

https://www.tutorialkart.com/kotlin-android/android-open-url-in-browser-activity/

6.14 Back press behavior on Back button

https://stackoverflow.com/a/27807976

6.15 SearchBar and AutoCompleteTextView

To implement the search functionality, these pages will help:
https://www.youtube.com/watch?v=9OWmnYPX1uc
https://developer.android.com/guide/topics/search/search-dialog

Working with the AutoCompleteTextView to show the suggestions might be a little challenging. This tutorial goes over how it is done to help implement it.
https://www.truiton.com/2018/06/android-autocompletetextview-suggestions-from-webservicecall/

In order to link your Search Bar with autocomplete suggestions, these links might help:
https://www.dev2qa.com/android-actionbar-searchview-autocomplete-example/

6.16 To implement favorites and portfolio sections

https://developer.android.com/reference/android/content/SharedPreferences

https://www.journaldev.com/9412/android-shared-preferences-example-tutorial

6.17 Implementing Dialogs

https://mkyong.com/android/android-custom-dialog-example/

https://androidexample.com/Custom_Dialog_-_Android_Example/index.php?view=article_discription&aid=88&aaid=111

https://medium.com/@suragch/creating-a-custom-alertdialog-bae919d2efa5

6.18 Sectioned Adapter for RecyclerView

https://github.com/luizgrp/SectionedRecyclerViewAdapter

https://github.com/luizgrp/SectionedRecyclerViewAdapter/tree/master/app/src/main/java/io/github/luizgrp/sectionedrecyclerviewadapter/demo/example1

https://github.com/luizgrp/SectionedRecyclerViewAdapter/blob/master/app/src/main/res/layout/section_ex1_header.xml

https://github.com/luizgrp/SectionedRecyclerViewAdapter/blob/master/app/src/main/res/layout/section_ex1_item.xml

https://github.com/luizgrp/SectionedRecyclerViewAdapter/blob/master/app/src/main/res/layout/fragment_ex1.xml

6.19 Rounded buttons/Images

https://stackoverflow.com/a/13872391

6.20 GridView

https://abhiandroid.com/ui/gridview#:~:text=In%20android%20GridView%20is%20a,item%20by%20clicking%20on%20it.

6.21 Using Recyclerview inside ScrollView

https://stackoverflow.com/questions/27083091/recyclerview-inside-scrollview-is-not-working

6.22 Loading local HTML files into WebView

https://stackoverflow.com/questions/27104185/webview-load-html-from-assets-directory-and-send-data-to

https://stackoverflow.com/questions/17180491/highcharts-wont-load-in-android-webview-on-android-4-x

https://stackoverflow.com/questions/27709995/load-a-javascript-method-from-android-webview?rq=1

https://stackoverflow.com/questions/18302603/where-to-place-the-assets-folder-in-android-studio
https://stackoverflow.com/a/4029605

6.23 Swiping to delete feature in RecyclerView

https://www.journaldev.com/23164/android-recyclerview-swipe-to-delete-undo#code

6.24 Drag and Reorder feature in RecyclerView

https://www.journaldev.com/23208/android-recyclerview-drag-and-drop

7. Files to Submit

You should also ZIP all your source code (the java/ and res/ directories excluding the vector drawables that were downloaded) and submit the resulting ZIP file **if requested by the course staff.**

Unlike other semesters, you will have to demo your submission using **Zoom Remote Control** during a special grading session. Details and logistics for the demo will be provided in class, on the Announcement page and on Piazza. **Demo is done on a Laptop/Notebook/MacBook or Windows PC using the emulator, and not a physical mobile device.**

****IMPORTANT****

All videos are part of the homework description. All discussions and explanations on 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.