Lucas Palmer, 3B BMath (AMATH, CS, CM)
Undergraduate Research Assistant, Fall 2016
Supervisor: Justin Wan

Read the following points before you start programming:

1.) The application makes use of Sugar ORM database to store its data. Sugar ORM is currently not compatible with Instant Run (a tool in Android Studio for quick compilation). As a result, when you initially try to install the application on a phone or an emulator, you might get database errors. To resolve these issues, disable Instant Run in Android Studio (File->Settings (or Preferences)->Build, Execution, Deployment->Instant Run, then uncheck enable...). You can re-enable Instant Run after you successfully install the application on the device.

2.) I would recommend visiting https://satyan.github.io/sugar/creation.html and reading up on the features of the Sugar ORM database. All the database models are stored in the "portfoliorebalancing/model" directory in the application files. This database does allow single relationships. For an example of how to create relationships between database models, see the SimulationStrategy.java model (which is related to the Simulation.java model).

3.) The code for the Android application is on Github. Ask Justin Wan to add your Github profile as a "Collaborator" to the project. You will then be able to commit to the project using your own Github username and password. To get the code, do the following:

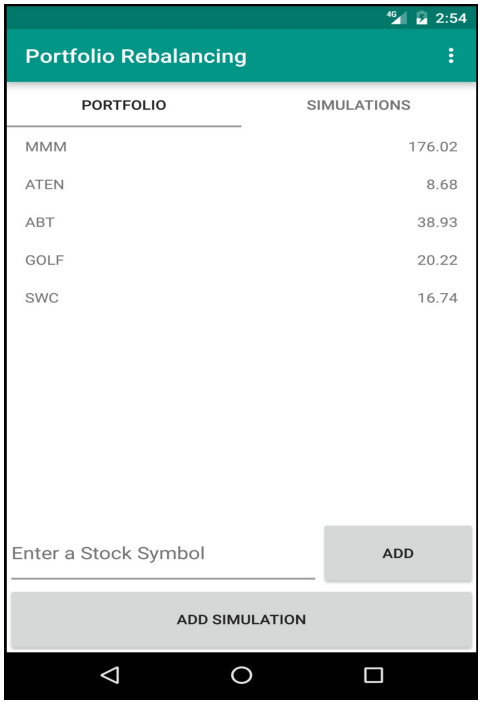
i.) In terminal, go to the directory where you want to store the root directory of the project.
ii.) Run "*git clone https://github.com/portfoliorebalancing/portfolio-rebalancing-android*". This creates a directory called "portfolio-rebalancing-android" which contains all the code.
iii.) Install Android Studio.
iv.) Open the project in Android Studio by going to File->Open and then selecting the "portfolio-rebalancing-android" directory that you just cloned from Github.
v.) If you are prompted with "Unregistered VCS root detected," click "Add root".

It would be helpful to learn the basics of Git source control before you begin. However, all you really need to know is this:

i.) Once you are satisfied with your changes and want to commit them, navigate to the project directory ("portfolio-rebalancing-android") in terminal.
ii.) Run "*git status*" to check that your files are correctly marked as changed.
iii.) Run "*git add .*" to add your changes.
iv.) Run "*git commit -m "An informative Message Here""* to commit your changes.
v.) Run "*git push origin master*" to push your changes to the master branch of the project.

The purpose of this URA was to get the portfolio rebalancing Android application as close to production quality as possible. This not only involved adding new features, but using current trends in Android programming to make the user interface look as polished and intuitive as possible. Listed below are that main things that I did to accomplish this.

Before:



After:

1.) Adding a Custom Theme to the Application
The theme is what specifies visual qualities (such as primary colour, background colour, and accent colour) that will be used throughout the application to create uniformity. Previously, the application used Android's default theme. I created a custom theme using a blend of blue and white to give the application its own distinct appearance.

2.) Using Icons Instead of Text
Previously, the application used text to describe each tab in the main activity as well as the functionality of buttons. Popular Android applications today use informative icons rather than text to create a more visually appealing user experience. https://material.io/icons/ is a great place to get free icons.

3.) Improving the Display of Items in a List
In several places in the application, items such as stocks and simulations are displayed in a list. Rather than using a generic view for each item, I created my own custom view that is more colourful and visually pleasing.

4.) Managing Async Tasks
An Async Task is used to perform a task on a background thread rather than the UI thread. They are used because performing large tasks on the UI thread can cause the UI to become

frozen/unresponsive. Android requires that some tasks (such as accessing data from a URL) must be done in an Async Task.

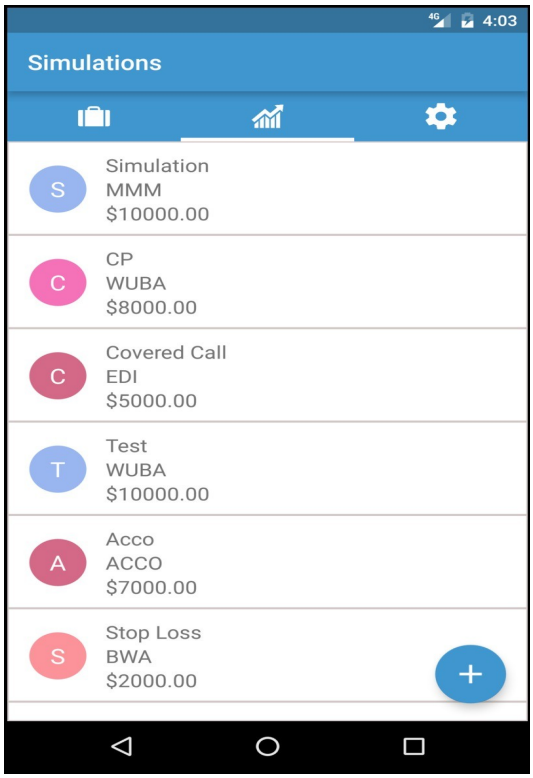
The application fetches stock data from various APIs via URL. It is important that the Async Tasks used to fetch this data are managed properly. Consider the following example:

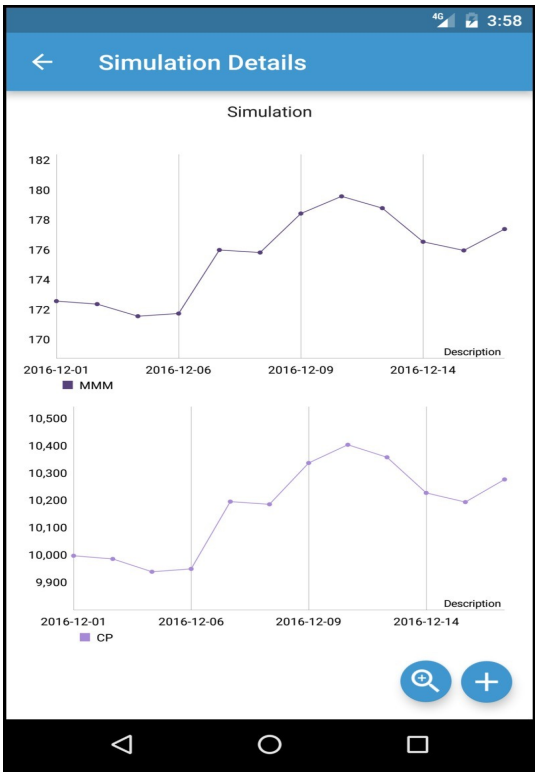
Suppose I create a simulation with many different strategies and a start date very far in the past. When I click on such a simulation to view the simulation graphs, an Async Task will be run to fetch all the stock data and the portfolio value will be calculated for each strategy. This could potentially take a long time if the simulation is big enough. Thus, when I exit the activity that displays the graphs, I need to make sure that I cancel the Async Task if it is still running. Otherwise, the task could remain running in the background and prevent other Async Tasks from being run until it is finished.

For an example of a simple way to manage Async Tasks, see AsyncTaskActivity.java and the classes that inherit from this class.

5.) Making Use of the Floating Action Button

The floating action button is an Android widget that has become increasingly common in Android programming. You can find it in the Gmail, LinkedIn, YouTube, Facebook, and just about any other Android application on the market today. Floating action buttons are used when an Android component has one (or two) main functionalities. For example, in the portfolio tab pictured below, there is one main functionality: adding a stock to your portfolio. So it makes sense to use a single floating action button (with a '+' icon) for the user to access this functionality. In the simulation details activity pictured below, there are two main functionalities: adding a strategy to the simulation and viewing an enlarged graph of your portfolio simulation. Hence, I used two floating action buttons here.





6.) Minimizing the Need for User Text Input

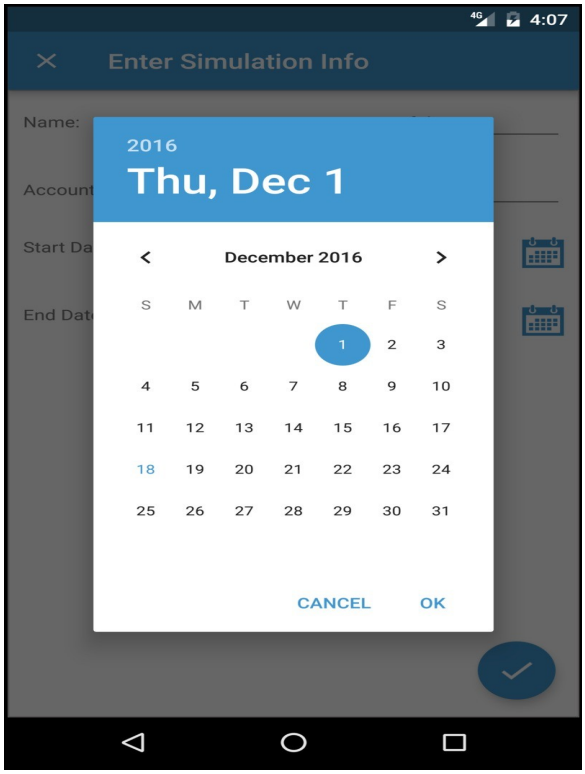
Inputting text into an Android application using a phone keyboard can be slow and messy. As a result, Android applications today only make the user input text when it is absolutely necessary. The following two improvement were made to improve user experience in this area:

i.) Adding a Calendar Date Picker to the Simulation Workflow

When a user is creating a new simulation, they must select the start and end date of the simulation. Previously, the user had to enter text in a very specific format (ie. 01-02-2016). The app then parsed this text and stored it in the application model. There are two problems with this approach: i.) the user must manually input the date and ii.) storing the date as a string means that it is hard to manipulate (ie. it is hard to compare two dates). So instead, a calendar widget was added so that the user can easily select a date without entering text. This also allows us to make use of the built in java Calendar and Date objects, which are very easy to manipulate.

ii.) Adding a Stock Picker

Previously, when a user wanted to add a stock to their portfolio, they had to manually enter the text of the stock symbol and then try and add it to their portfolio. If the stock symbol was invalid, the application would not do anything. As a result, it was unclear whether the application was still getting stock data from the Internet or whether an error had occurred. Instead, I added a Stock Picker Activity into the application. This activity loads all possible stock symbols into a list so that the user can add an arbitrary number of stocks simply by navigating the list and making selections.



7.) Completely Redesigning the Simulation Workflow

Previously, when the user wanted to create a new simulation, they were taken through a series of dialogues. Each dialog would take input from the user, store this input, and then activate the next dialog in the workflow until all necessary information had been gathered from the user. There are several reasons why using dialogues was undesirable for this workflow:

i.) Dialogues are useful for getting simple input from the user. For example, if you simply want to ask the user a yes/no question, using an Android dialogue is ideal. However, creating a simulation requires more complex input. The user is required to select the start/end dates of the simulation as well as choose a stock from their portfolio. As a result, dialogues can get cluttered and become hard to use.

ii.) Android dialogues are hard to customize. Because dialogues are built-in Android widgets, they come with behavior that is sometimes not desirable. There are usually ways to work around certain default behaviour, but this becomes messy and prone to bugs.

Luckily, there is another programming technique that neglects these disadvantages and comes with its own advantages. Instead of going from dialogue to dialogue, do the following:

i.) Create a new activity whose sole purpose is to take the user through the workflow.

ii.) Make use of Android Fragments to display the workflow. Each fragment will represent a different part of the workflow.

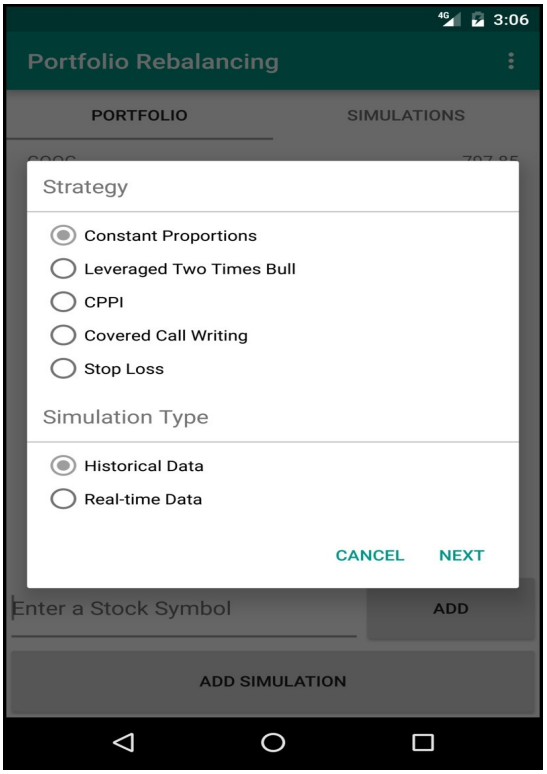
iii.) When the user is done entering input into a fragment, switch to the next fragment in the workflow.

There are several advantages to using this technique:

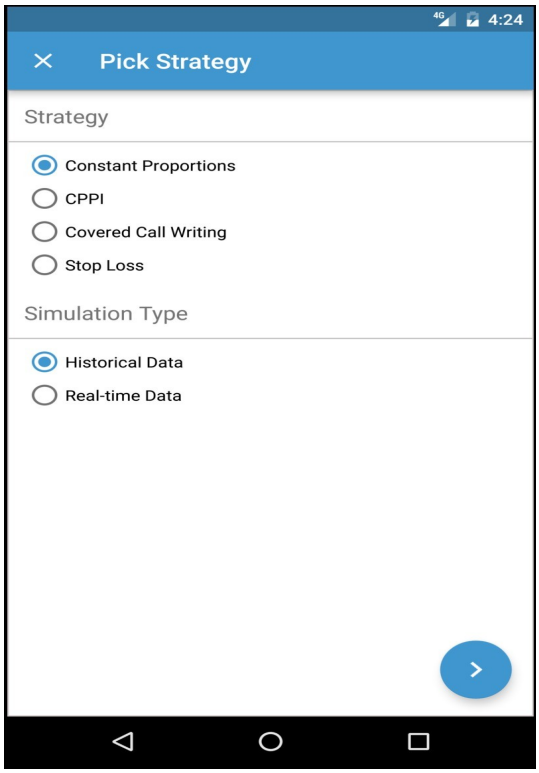
i.) By housing the entire workflow in a single Android Activity, you can keep track of all user input in a single location. This prevents you from having to pass data from one component to the next.

ii.) Fragments, unlike dialogues, are easily customizable. They are very capable of handling more complex user input.

For two good examples of housing a workflow in a single activity, see AddStrategyActivity.java and SimulationSelectorActivity.java.



a.) Previous workflow.



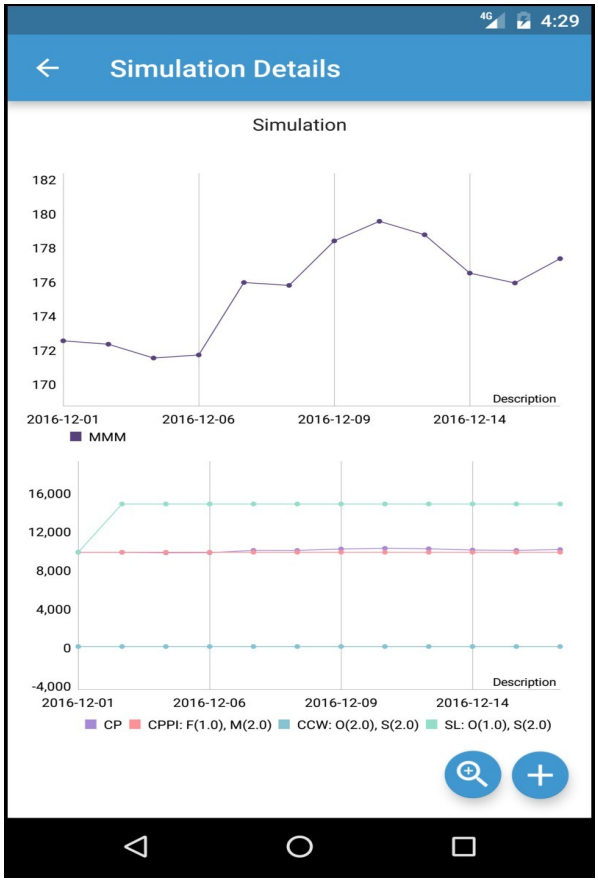
b.) New workflow.

8.) Changing the Simulation Model to Have Only One Stock

Previously, when creating a simulation, all the stocks in a user's portfolio were added to the simulation. When there are multiple stocks in a simulation, rebalancing the portfolio becomes increasing complex. So for now, the user will only be allowed to choose a single stock in a simulation.

9.) Adding Multiple Strategies to a Simulation

Previously, a simulation only had one strategy. However, it would be very useful to view multiple strategies on the same graph in order to compare them. As a result, I added a workflow for adding new strategies to an existing simulation. This will allow users to compare different rebalancing strategies while keeping other simulation data (ie. stock, start date, end date, and initial value) the same.



10.) Adding Custom Application Settings
To create a better user experience, lots of Android applications have their own settings/preferences that the user can change. I added the following settings:

i.) Stock Ordering in Portfolio
When the user views the stocks in their portfolio, they can choose whether to view them in the order in which they were added, in alphabetical order by name, in alphabetical order by symbol, or in increasing order by price.

ii.) Stock Ordering in Selector
When the user views the stocks in the stock picker, they can choose whether to view them in alphabetical order by name, in alphabetical order by symbol, or in increasing order by price.

iii.) Stock Market
The stock picker can either display stocks from the New York Stock Exchange or NASDAQ. Choose your preferred stock market here.

iv.) Simulation Ordering in Portfolio
When the user views their simulations, they can choose whether to view them in the order in which they were added, in alphabetical order by name, in alphabetical order by symbol, or in increasing order by initial account balance.

v.) Preferred Strategy
When the user is creating a new simulation, their preferred strategy will be selected by default.

vi.) Preferred Type
When the user to creating a new simulation, their preferred type will be selected by default.

For now, the settings are located in the third tab of the main activity. In the future, if more tabs need to be added to the main activity and there is not enough room, the settings activity can be relocated to the Android Options Menu.