COMP3900 20T3 Report

Project Title: Investment Simulator

System Name: Hermes

Group: comp3900-h13a-anonymous

Submission Date: 16/11/2020

Team:

Riley Vozzo
zID: z5161841
Email: z5161841@unsw.edu.au
Role: Scrum master, Developer

Haodong Lu
zID: z5183944
Email: z5183944@ad.unsw.edu.au
Role: Developer

Haoyang Ren
zID: z5183825
Email: z5183825@ad.unsw.edu.au
Role: Developer

Daniel Quin
zID: z5183850
Email: z5183850@ad.unsw.edu.au
Role: Developer

Saloni Goda
zID: z5215272
Email: z5215272@ad.unsw.edu.au
Role: Developer

Table of Contents

1. Overview

Hermes (named after the Greek God of wealth and trade) is an easy to use investment simulator designed for novice investors with limited stock market experience. The app uses real world market data, meaning the skills learnt on the simulator can transfer over into the real world. While Hermes is designed for beginners taking their first steps in the world of stock investment, experienced investors may also use the app to devise a strategy that works for them.

Hermes provides the following functionalities, which will be expanded upon in much greater detail later in this report. Users are able to:

- Search for stocks.
- Add/remove stocks from their watchlist.
- View interactive graphs showing historical stock prices.
- Set watch-prices triggers on stocks in their watchlist
- Forecast future prices of stocks in their watchlist
- Buy and sell stocks in a real time market.
- View their total virtual portfolio.
- View their transactions history.
- View their statistics about their trading data.
- Compete against other users via a global leader board

1.1 System Architecture

Interface Layer
The user interface is managed by HTML, CSS, and JavaScript to create our responsive application. JavaScript is a fast, simple client-side scripting tool that is very versatile and flexible. It is platform independent and works on any browser. To visualise the data, we are using libraries like plotly and Chart.js which have interactive, responsive charts that make the user experience more engaging.

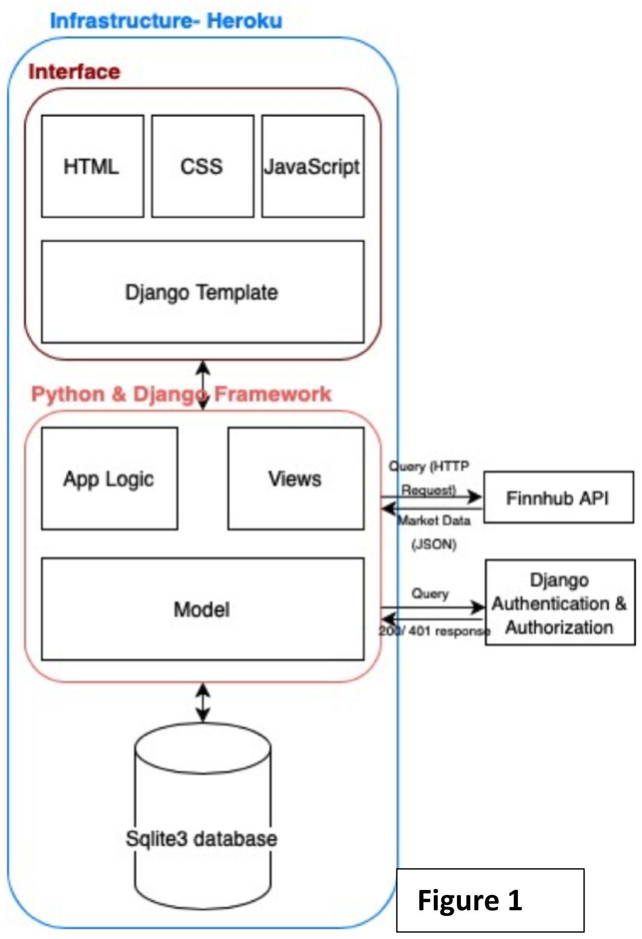


Figure 1

Business Layer
Hermes is using Python is the backend language and Django as the framework for the web application. We chose Python due to its support for data analysis libraries like pandas, which proved to be very useful in the prediction and graphing functionality of the app. Django was chosen because it is both a sophisticated and easy to use, and is far better than other python web frameworks (e.g. Flask).

API layer
Hermes' main source of market data is the Finnhub API. This API is organized around REST. It has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs. Hermes uses Finnhub extensively gain real-time market prices, historical prices and company profiles. (3) (4)

Database Layer
The database Hermes uses is SQL light, which is an embedded application rather than a client-server database system. We made this decision because Sqlite3 integrates well with Django, which has a sqlite3 functionality out of the box.

Infrastructure Layer
The software application is being run on Heroku, which is a Platform as a Service (PaaS). The cloud service helps us to handle the application without building and maintaining on-site infrastructure. (5)

1.2 Django Web Framework

Django was chosen as our framework of choice because of its speed of implementation, inbuilt security protocols and extensibility. A simple Django project can be quickly deployed that consists of a frontend with HTML, CSS, Javascript, a backend with Python, an sqlite3 database and an admin dashboard with user and database information.

Intuitive User Interface

The frontend was designed to be aesthetically pleasing and intuitive. A variety of features have been implemented on the frontend to be easy for the user to play the game, including real-time alerts, history of all transactions, purchases and alerts, leader-board, and being able to filter purchases.

Django Architecture

Django follows the architecture of a Model-Template-Controller (MTC). Model is the data structure behind the entire project and we have used an sqlite3 database in this case. Initially, we thought of using AWS DynamoDB as the database however we found that the built-in sqlite3 database was easier to integrate and improved speed as it did not need to connect to AWS servers. The view is the user interface, where we have used HTML, CSS and Vanilla javascript to make it a fast and lightweight application. The Template connects the View and Model together and passes data from the Model to the View and vice versa.

Benefits of Django

Django has 5 main benefits - versatility, security, scalability, maintainability and portability.

- Versatility - Django is versatile and allows for all kinds of functionality. As we had various different functionality such as aggregating user data for the leader-board, displaying charts and graphs and a real-time alert service, Django’s extensibility was very useful for this project.
- Security - Security protocols such as storing password hashes, storing a key in cookies and session data in database and protection against SQL injection, cross-site scripting attacks are inbuilt in Django.
- Scalability - Django’s architecture is a component-based shared-nothing architecture which allows for components to be replaced or modified without affecting the other overall framework. This also eliminates single points of failure and allows the system to scale as there is no central resource bottleneck.
- Maintainability - Django is written using object-oriented design patterns that facilitate the development of code that is maintainable and reusable. It uses the Don't Repeat Yourself (DRY) concept, in particular, so that there is no needless repetition, minimising the amount of code.
- Portability - Django is written in Python, which runs on several platforms, including Windows, Mac OS and Linux. As our group had different operating systems and we needed to run the code on CSE machines, the portability of Django was essential for this project.

2. Functionalities

This section will discuss the various functionalities offered by Hermes, including technical detail. For more information on how to use these functionalities within the application, see the User Manual section of this report (Section 5).

2.1 Searching

Functionality

The search functionality allows investors to search for a stock using a "stock code" (also known as stock “symbol”), with the result including the stock name, latest available unit price for the stock and the percentage change in the stock unit price when compared to the latest available stock unit price to the previous day's known stock unit price.

Technical detail
The *search* method in *API* class in file *api/search.py* fulfils the requirements of this functionality. Given a valid stock code, sends requests to the appropriate Finnhub API paths specified in the api documentation. The method gets the stock name, latest available unit price for the stock and the percentage change in the stock unit price compared to the latest available stock unit price of the previous day. It then stores these results in a dictionary to send to the frontend.

If the given stock code is invalid, it will raise a FinnhubSearchError telling the frontend "The stock code you searched was invalid". If the Finnhub API cannot be connected or the API is too busy to respond, it will also raise a FinnhubSearchError saying "Error with external API. Please try again". (3)(4)

2.2 Watchlist

Functionality

Investors can add/remove stocks to a personal watchlist, which lists details including stock codes, names, date added to watchlist, current price, change since last closing price, and any untriggered watch prices. Users can also view an interactive graph showing historical stock prices for any stock in their watchlist from the day the stock was added (or of the past 7 days if the stock was added recently).

Technical detail

Stocks are added/removed from a user's watchlist by creating an entry in the *WatchListItem* model of the inbuilt Django database. The methods which are used to maintain the watchlist are located in modules/watchlist.py.
Generating the interactive graph uses the *get_historical* function in file api/historical.py, which takes a stock code and a timestamp (corresponding to the initial time) as arguments. Note that if given time is too recent, a week before the current time is used as the initial time (calculated using datetime.timedelta). The function gets historical data from the

Finnhub API, and uses the pandas module to read the csv file response. After reading the csv, pandas will use a local dataframe to store all the data. Since the time read from API is in unix timestamp for, we used the datetime module to convert it to a datetime. Next, we use the plotly module to generate the candlestick graph with the features stated above and save it in “simulator/templates/simulator/graph.html”

2.3 Watchprices

Functionality

Users can set/remove watch prices from stocks in their watchlist by entering a price and either a 'Buy' or 'Sell' action. 'Buy' watch prices trigger when the stock's current price is lower than the set price, whereas 'Sell' prices trigger when the current price is higher than the set price. This is because, as an investor, your aim is to buy stocks when the price is low, and sell stocks when the price is high.

Technical detail

Watch prices are managing by a number of methods in the *Watchprice* in file *modules/watchprice.py*. The *set* method takes a stock code, watch-price and action as arguments, uses them to create a new entry in *WatchListAlert* table, and start a new thread with the *check* method. The *check* method will then run in its own thread and check the new watch-price every 10 seconds. If the action is “buy”, the alert will be triggered when the stock's current price is greater than the set watch-price. If the action is "sell" the alert will be triggered when the stock's current price is less than the set watch-price. When an alert is triggered, the datetime is recorded in the table entry, and the corresponding the thread is stopped.
The *remove* method takes a user id and alert id as arguments, find the corresponding entry in the *WatchListAlert* table, and deletes that entry. When an entry is deleted, the *check* method will detect this and stop the corresponding thread.
The *start_up* method is used to restart the threads for all untriggered watch-prices when the server is rebooted.

2.4 Predict Future Prices

Functionality

The prediction functionality allows investors predict future prices for a stock, after adding it to the watchlist. This allows the user to understand the trends of current stock by viewing the chart and graph generated by this function, without needing to look into the previous trading data. It establishes a way of visualizing digital data and provides clear indication on the flowing trend of a stock.

Technical Detail

The stock price prediction algorithm is located within the *predict* method in modules/prediction.py. The algorithm leverages the linear regression capabilities of the scikit-learn module, attempting to find a linear relationship between historical stock data and prices to predict future prices. The FinnHub API is used to retrieve historical data of a stock, with the opening/closing prices being used as main factors, and the highest/lowest price used as a support factors within the model. The algorithm uses these factors to train four models separately, and combines them to generate the final predicted price. The price is calculated by taking the average of opening and closing predicted prices and subtracting the square error. (7)
The prediction algorithm returns a list of predicted prices for upcoming dates, which are graphed in the frontend using the Chart.js JavaScript library. (8)

2.5 Buy and Sell Stocks

Functionality

Users are able to simulate buying and selling real world stocks by placing buy and selling orders for a given number of units at the current market price. When a stock is purchased it is added to the users virtual portfolio, provided they have enough funds in their account balance. When a stock is sold, it is removed from the users virtual portfolio.

Technical Detail

Logic for the buy/sell functionality can be found in the modules/buy_sell.py file. When a stock is purchase, an entry is added to the *Purchases* model within the Django database, with the *orignialUnitBought* field set to the number of units of that stock purchased, and the *price* field set to the purchase price. If another purchase is made of that same stock, a new *Purchases* entry is created, as the *orignialUnitBought* and *price* fields will vary.

Stocks are sold on a FIFO basis, with units corresponding to earlier purchases being sold first. The sell order is only processed if the total number of owned units are the stock across all purchases is greater than or equal to the number of units the user is trying to sell. When a sell order is made, the *unitSold* field in the oldest Purchases entry is increased until it is equal to the *orignialUnitBought* field. If the fields are equal and there are still more units to sell, the next oldest Purchases entry is used. This will continue until all the required units are sold. The reason this method is used, as opposed to updating a single table entry corresponding to all units of a given stock, is because the purchase price will vary between units purchased. This purchase price is very important when calculating profit and loss statistics (Section 2.6).

2.6 View Portfolio, Transactions and Portfolio Statistics

Functionality

Users can view a pages containing information about all the stock units they currently own, both on an individual purchase level, and when aggregated by stock. Users can also view the total profit all loss they would make if all units they currently own of a stock are sold at the current market price (i.e. Current worth – amount paid), as well as the profit they would make if all stock units in their entire portfolio are sold.
Users can also view a complete history of the buy and sell transactions they have made since signing up with Hermes.

Technical Detail

Logic for the purchases/portfolio functionality can be found in the modules/purchases.py file. User purchase and portfolio information is gathered from the Purchases table in the Django database, as well as from the Finnhub API (in the case of current price information). The profit/loss for an individual purchase is calculated by first deducing the number of units unsold in that purchase (i.e. *orignialUnitBought – unitSold*), then using that number in the following equation:

Profit = unsold_units * (current_price – purchase_price)

The total profit/loss for all purchases of a stock is calculated by summing up all the profits from the individual purchases of all the individual purchases of that stock. Finally, the total portfolio profit/loss is calculated by summing up the total profit/loss across all stocks in the portfolio.

2.7 Leaderboard

Functionality

Users can compete against other users which have an account with Hermes by viewing a leader board ranking the all users by total worth.

Technical Detail
Logic for the leader-board functionality can be found in the modules/purchases.py file. User information is retrieved for the User and Profile tables in the Django database. Users are ranked by total worth, which is calculated with the following formulas:

Portfolio_worth = sum(unsold_units * current_price) *of each stock in the users portfolio*

Total_worth = current_balance + Portfolio_worth

3. Implementation challenges

The following section will draw attention to any implementation challenges and non-trivial/sophisticated aspects of the Hermes application.

3.1 Threading

The watch-price functionality discussed in Section 2.3 required a non-trivial knowledge of sequential programming and concurrency. A challenge faced when implementing this was that initially the threads were constantly running with no break time, causing too many calls to the Finnhub API and eventually API throttling. To combat this, we set the threads to run every 10 seconds, reducing the number of times we call the API. This decision sacrifices some of the timeliness of watch-price alerts for increased application performance. (6)

3.2 Prediction Algorithm

The stock price prediction algorithm is an novel addition the team made to the investment simulator, as it is both non-trivial and not required by the initial Investment Simulator specification. The prediction algorithm went through a variety of iterations. Bayesian regression was used first, but it turned out that this method required multiple attributes to generate a related model, and produced an overfitted result when stock price was the only attribute provided. After going through the basic manual of scikit learn modular (7), basic linear regression was found to be the best fit as a prediction model for this project.

Another challenge encountered with developing the algorithm was that the data returned from the Finnhub API did not integrate smoothly with the scikit prediction function. The Api returns a convoluted data structure of price information, which require difficult pre-processing before it could be forged into a single list.

3.3 Database

The applications database design and development is another non-trivial aspect of the application, which came with its own development challenges. An ER diagram of the application can be seen in Figure 2.

Initially, the team planned to use Sqlite3 as the SQLite implementation. However, after the first iteration, we found that the Django framework supports its own database models, which behave more coherently with both the backend and frontend. Thus, after the first progression demo, the team migrated our SQLite database to the Django model, which proved to be a relatively difficult and time consuming process.

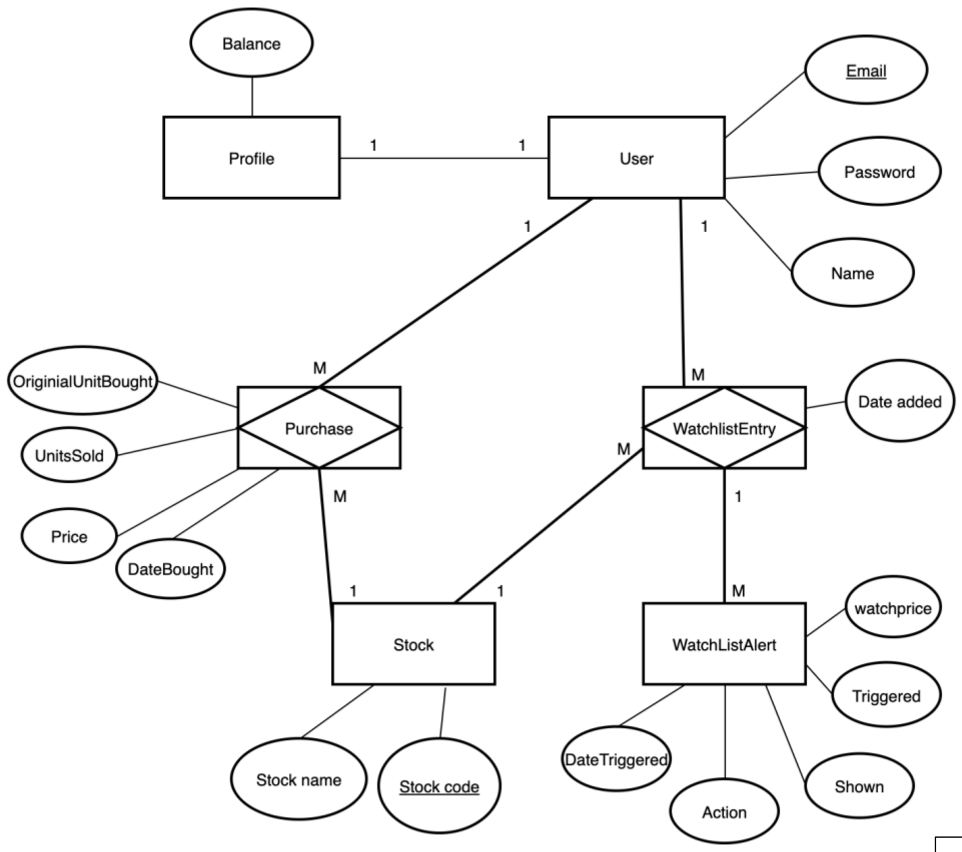


Figure 2

3.4 Deployment

Deploying the application on the public internet was not a requirement in the initial Investment Simulator specification, and thus can be treated as a non-trivial enhancement of our application. One challenge the team faced when it came to deploying the application was in regards to dependency management. From the start if the project, the team used pipenv as their virtual environment system, however Heroku does not recognise pipenv's method or managing dependencies. Thus, before with deployed, we first needed to generate a requirements.txt file containing all our pipenv dependencies, and install them in the Heroku server the old fashion way.

4. Third Party Tools and Libraries

Tool/Library	Description	Licensing
Finnhub.io (3)(4)	Used to retrieve both historical and current market data.	The Finnhub api has several paid methods, however only free methods are used by Hermes. Thus, the team does not have to worry about any licensing terms impacting the application
Pandas (9)	Python module used to read and manage csv files containing market data	Free to use
Plotly (1)	Python module used to generate candlestick graphs for stocks in html format	Free to use
Charts.js (8)	Javascript library used to make interactive, responsive charts	Free to use
scikit-learn (7)	Python module used for its regression capabilities in the prediction algorithm	Free to use
Heroku platform (5)	Underlying infrastructure used to deploy the application on the internet	Hermes is deployed using the free tier, meaning the team doesn't have to worry about any licencing issues. However, since the free tier does not include any scaling capabilities, performance may reduce if the site receives too much traffic.

5. User Manual

5.1 Setup and Configuration

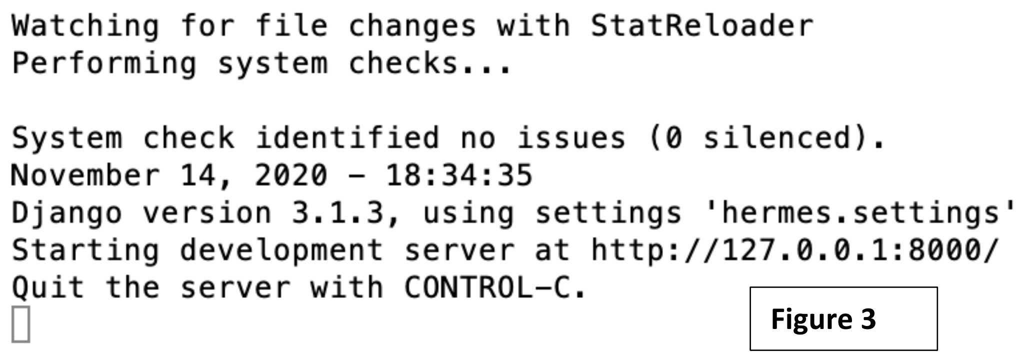
Since the application was deployed on the internet, using the application is as easy entering the following url: invest-simu.herokuapp.com
(*Note that since the Heroku server used to deploy the app is located in USA, all times are in that time zone*.)

However, the application can be set up on the UNSW CSE machines by following the steps below:

Steps

1. Unzip and save the project. Navigate into the unzipped directory via the command line. (i.e. You should be in the same directory as manage.py
2. Install pipenv with the command
    - $ pip3 install pipenv
3. Install project dependencies with
    - $ python3 -m pipenv install
4. Activate the projects virtual environment with 'python3 -m pipenv shell'
5. Setup database with the following 2 commands:
    - $ python3 manage.py makemigrations
    - $ python3 manage.py migrate
6. Run the server with the command
    - $ python3 manage.py runserver

   You should get the following output:



```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 14, 2020 - 18:34:35
Django version 3.1.3, using settings 'hermes.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 3

   Note: If you get the error message **'Django Server Error: port is already in use'**, you can change the port by passing in a new port via the command line. E.g.
    - $ python3 manage.py runserver 8080

7. Copy the http address into the url above to use the web application. I.e. http://127.0.0.1:8000/

5.2 How to use Hermes

5.2.1 User Authentication

To log in from the login page (Figure 4), enter a valid email and password, and click the 'Login' button. When logged in, you will be taken to the home page (Figure 6) where you are presented with a welcome message.

Whenever you are logged in, a 'Logout' button will be available on in the top right of your screen. If this button is pressed, you will be logged out and taken to the login page.

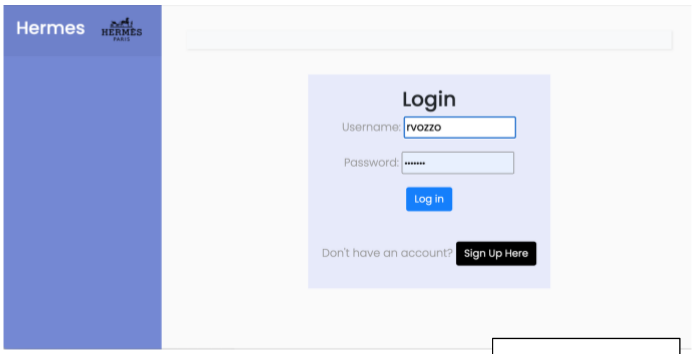


Figure 4

To create an account, click the 'Create Account' button on the login page. You will be taken to the sign up page (Figure 5) and prompted to enter the details needed to create an account. After valid details are entered, click the 'Sign Up' button, to be taken to the home page. Note that all new users start with $10,000 in their account balance when they create a new account, and that when logged in, your current balance is always visible in the sidebar.

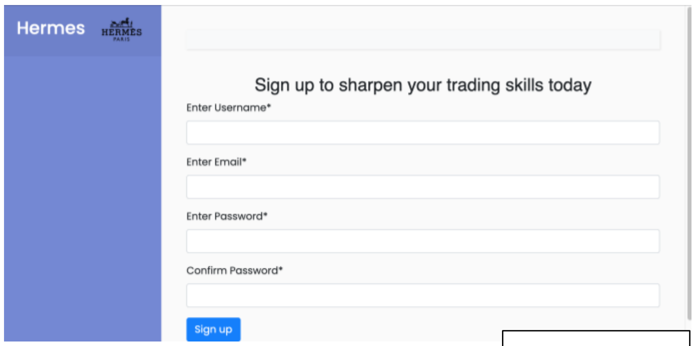
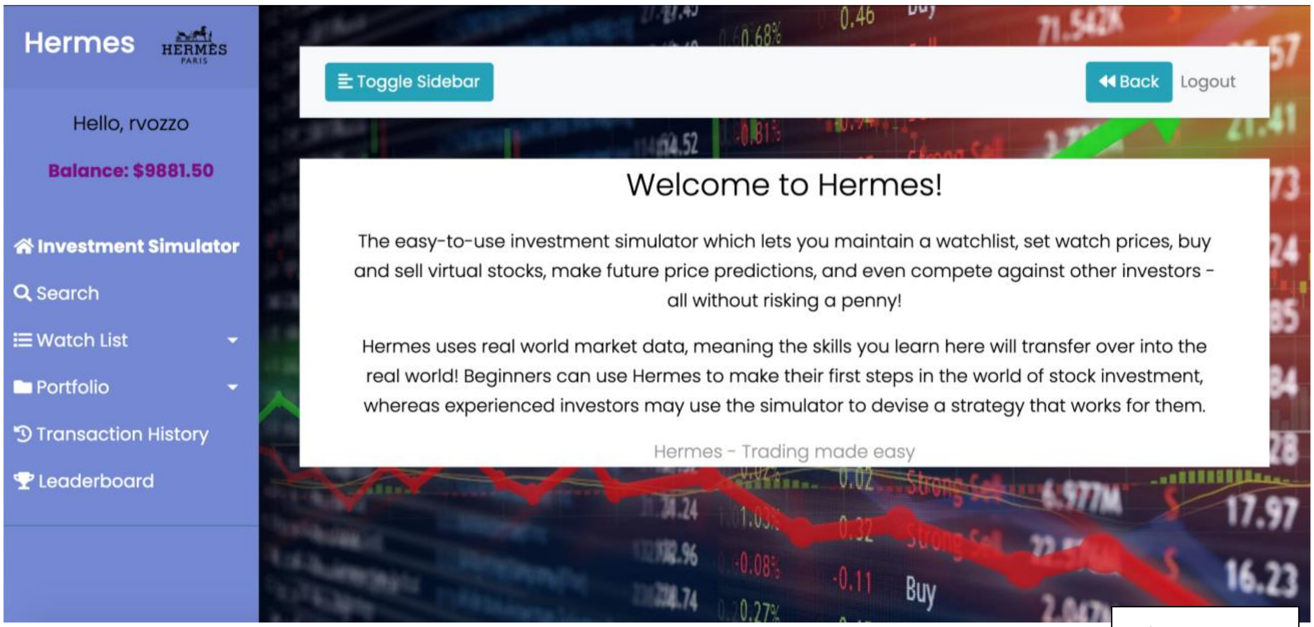


Figure 5



Welcome to Hermes!

The easy-to-use investment simulator which lets you maintain a watchlist, set watch prices, buy and sell virtual stocks, make future price predictions, and even compete against other investors - all without risking a penny!

Hermes uses real world market data, meaning the skills you learn here will transfer over into the real world! Beginners can use Hermes to make their first steps in the world of stock investment, whereas experienced investors may use the simulator to devise a strategy that works for them.

Hermes - Trading made easy

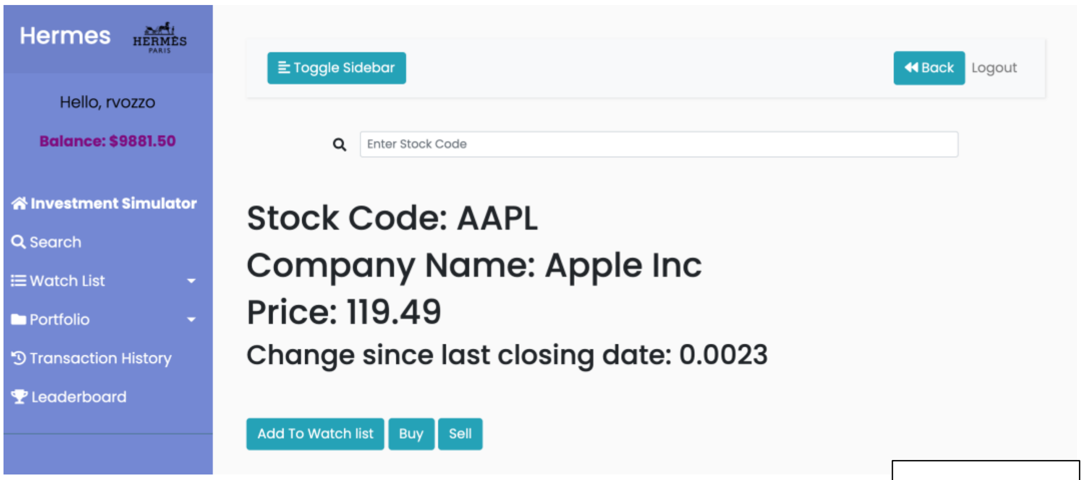
Figure 6

5.2.2 Search and add to Watchlist

To search a stock, click the 'Search' tab in the sidebar. You will be taken to a page with a search bar for entering a stock code. When a valid stock code is searched, some details about the stock are displayed including the stock name, current price, and percent change since the last closing price (Figure 7). You will also be presented with a 'Add To Watchlist', 'Buy' and 'Sell' button.

Clicking 'Add To Watchlist' will add the stock to your watchlist (provided it is not already in there), and direct you to the watchlist page. This page can also be accessed by clicking the 'Watchlist -> My Watchlist' tab in the sidebar.

Note that the 'Buy' and 'Sell' buttons and functionality will be discussed in a later section of this manual. (section)


Hello, rvozzo

Balance: $9881.50

Stock Code: AAPL

Company Name: Apple Inc

Price: 119.49

Change since last closing date: 0.0023

Figure 7

5.2.3 Watchlist

From the 'My Watchlist' page (Figure 9), you can view information about stocks in your watchlist, including stock codes, names, date added to watchlist, current price, change since last closing price, and any untriggered watch prices. Each row will also have a 'Actions' dropdown, with an number of available actions:

- Remove: Removes the stock from the watchlist.
- Graph: Displays an interactive graph of historical prices since the stock was added to the watchlist. Note that if the 'Graph' option is clicked on the same day that the stock was added, the graph will show one week's worth of historical data (see Section 5.2.4)
- Buy: Lets you buy that stock (see Section 5.2.6)
- Predict: generated 2 graphs of forecasted future prices (one for 1 week in the future, and one for a month in the future). (Figure 8)

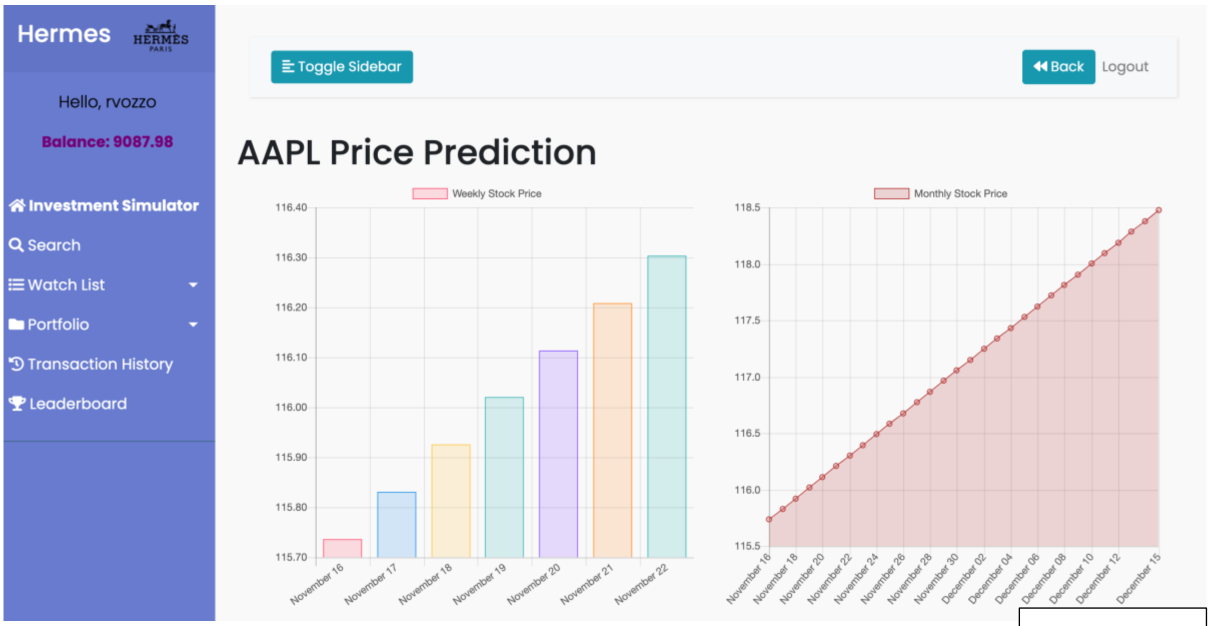
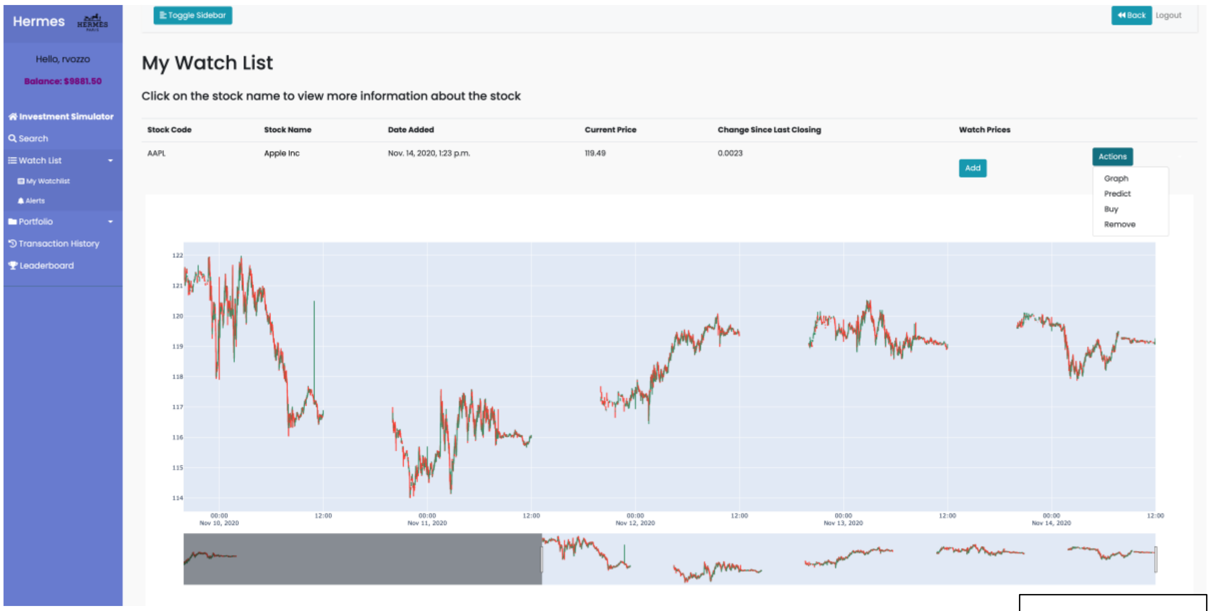


Figure 8

5.2.4 Historical Price Graph

Clicking on the 'Graph' option from the 'Action' dropdown in the 'My Watchlist' and 'My Portfolio' page will display a fully interactive graph of historical prices for the corresponding stock (Figure 9). Users can zoom in on the graph by with the scroll bar underneath the graph, or by dragging their cursor on the graph itself. Hovering your mouse over the graph will also give you quote information about that particular point in time, such as the open, high, low and close prices.

Note that the historical price graph can only be displayed for one stock at a time. Clicking the 'Graph' option on another stock will reload the page with a new graph for that particular stock



My Watch List

Click on the stock name to view more information about the stock

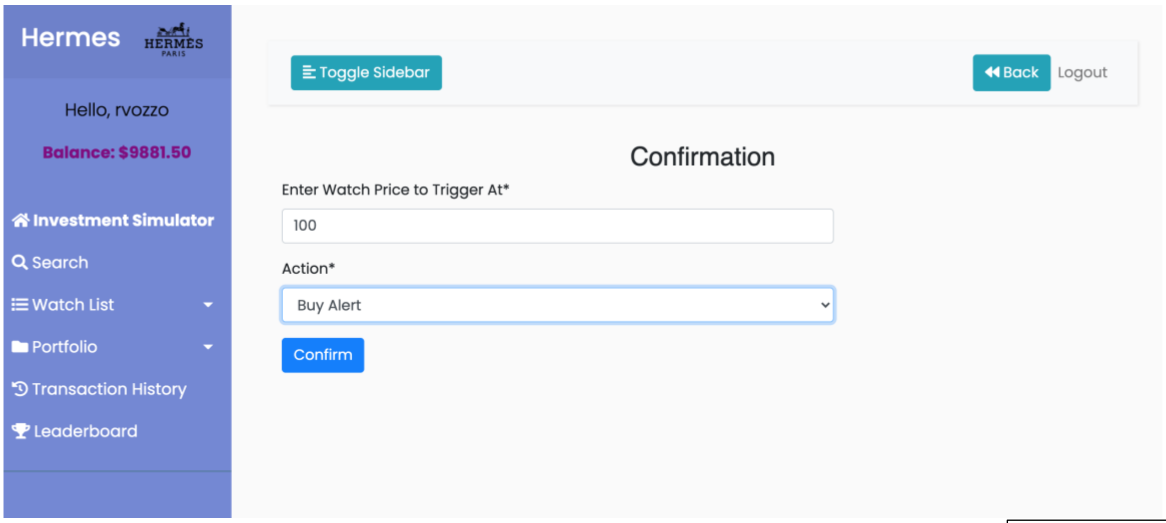
Stock Code	Stock Name	Date Added	Current Price	Change Since Last Closing	Watch Prices
AAPL	Apple Inc	Nov. 14, 2020, 1:23 p.m.	119.49	0.0023	Add

Figure 9

5.2.5 Watch Prices

Set watch-prices on a stock from the 'My Watchlist' page by clicking the corresponding 'Add' button in the watch price column. You will be taken to a form where you can enter a price, and an action, either 'Buy' or 'Sell' (Figure 10). 'Buy' watch prices trigger when the stock's current price is lower than the set price, whereas 'Sell' prices trigger when the current price is higher than the set price. This is because, as an investor, your aim is to buy stocks when the price is low, and sell stocks when the price is high.

All currently set watch-prices can be viewed next to their corresponding stock in the 'My Watchlist' page, as well as from the 'Watchlist -> Alerts' tab. When a watch-price is triggered. When a watch-price is triggered, the user will get a notification the next time they access the 'My Watchlist' screen (Figure 11), and they can see the time it was triggered from the 'Alerts' page (Figure 12). Watch prices can be unset by clicking the remove button next to the watch price. Note that watch prices are polled every 10 seconds, meaning they may to be triggered immediately.



Hello, rvozzo

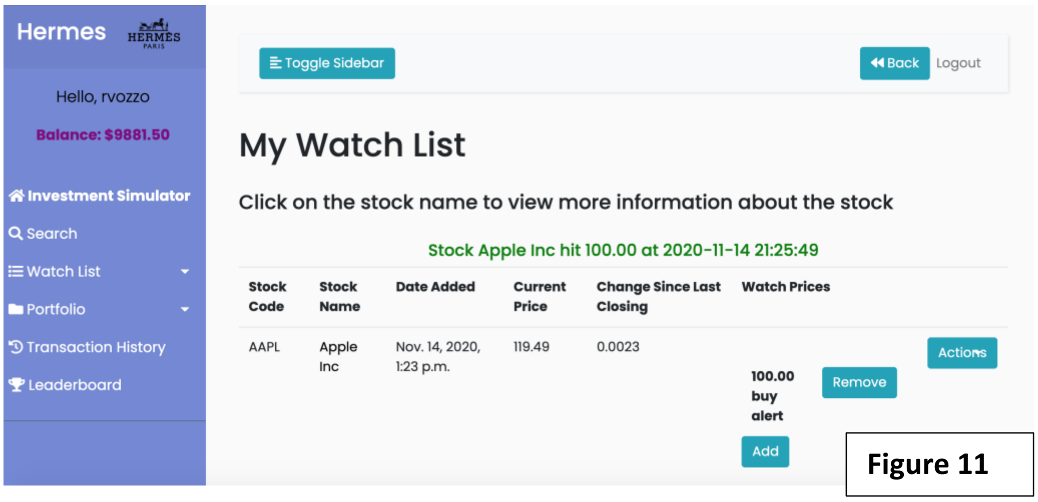
Balance: $9881.50

Confirmation

Enter Watch Price to Trigger At*: 100

Action*: Buy Alert

Figure 10



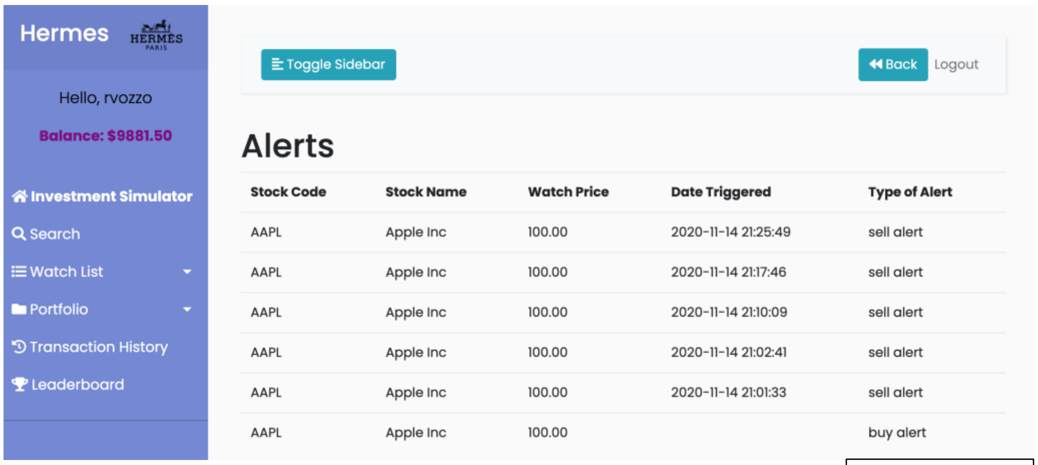
My Watch List

Click on the stock name to view more information about the stock

Stock Apple Inc hit 100.00 at 2020-11-14 21:25:49

Stock Code	Stock Name	Date Added	Current Price	Change Since Last Closing	Watch Prices		
AAPL	Apple Inc	Nov. 14, 2020, 1:23 p.m.	119.49	0.0023	100.00 buy alert	Remove	Actions
					Add		

Figure 11



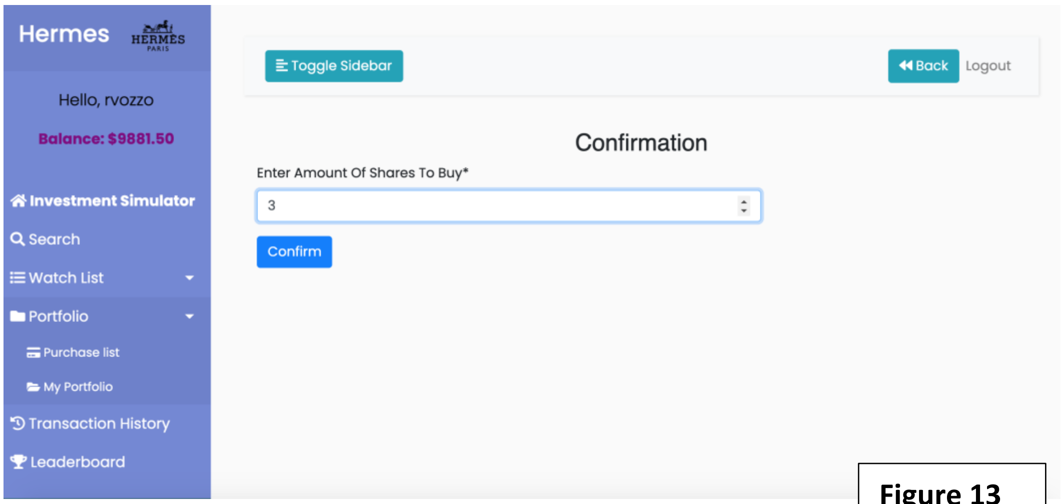
Alerts

Stock Code	Stock Name	Watch Price	Date Triggered	Type of Alert
AAPL	Apple Inc	100.00	2020-11-14 21:25:49	sell alert
AAPL	Apple Inc	100.00	2020-11-14 21:17:46	sell alert
AAPL	Apple Inc	100.00	2020-11-14 21:10:09	sell alert
AAPL	Apple Inc	100.00	2020-11-14 21:02:41	sell alert
AAPL	Apple Inc	100.00	2020-11-14 21:01:33	sell alert
AAPL	Apple Inc	100.00		buy alert

Figure 12

5.2.6 Buy Stock

Stocks can be purchased from a number of pages in the system, including the 'Search', 'My Watchlist' and 'My Portfolio' page. Clicking on the 'Buy' button on these pages will take you to a form where you can enter the number of units you would like to buy and confirm (Figure 13). Provided that you have enough funds in your account, the stock units will be purchased and your account balance will be updated accordingly. You will also be taken to the 'Purchase List' page, where you will see your new purchase.



Confirmation

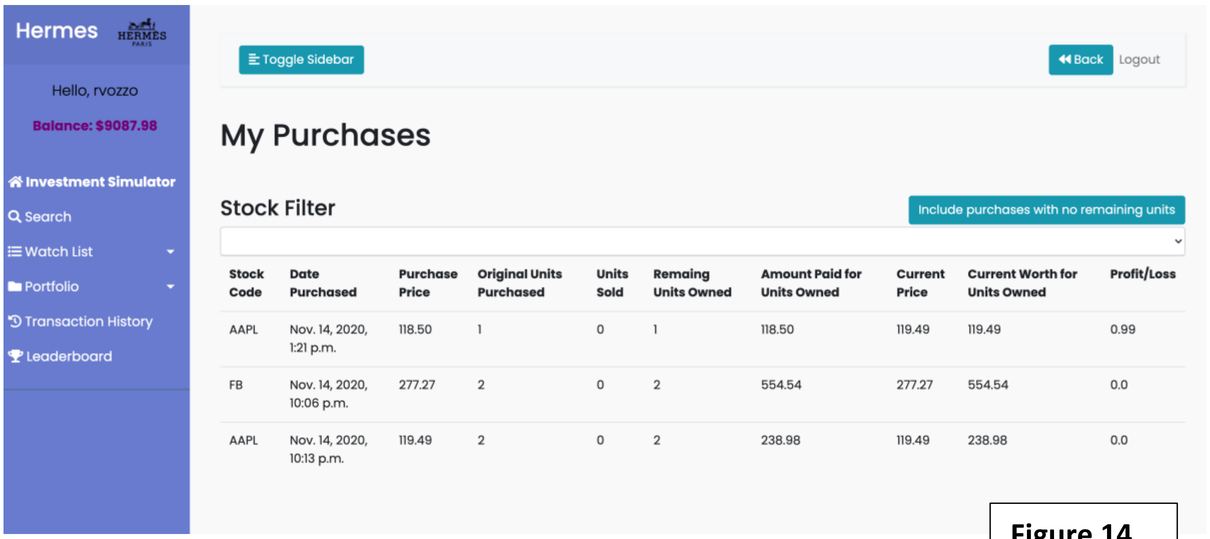
Enter Amount Of Shares To Buy*

3

Figure 13

5.2.7 Purchase List

Clicking on the 'Portfolio -> Purchase List' tab will take you to a page listing information about all the individual stock purchases that you have made, including the date the stock was purchased, the purchase price, number of units purchased, total paid, current price, total current worth, and profit/loss for that particular purchase (i.e amount paid – Total current worth). By default only purchases with remaining unsold units are displayed, however clicking the 'Include purchases with no remaining units' button will display all sold and unsold purchases. You can also filter purchases to only those of a particular stock using the 'Stock Filter' dropdown.



My Purchases

Stock Filter

Stock Code	Date Purchased	Purchase Price	Original Units Purchased	Units Sold	Remaing Units Owned	Amount Paid for Units Owned	Current Price	Current Worth for Units Owned	Profit/Loss
AAPL	Nov. 14, 2020, 1:21 p.m.	118.50	1	0	1	118.50	119.49	119.49	0.99
FB	Nov. 14, 2020, 10:06 p.m.	277.27	2	0	2	554.54	277.27	554.54	0.0
AAPL	Nov. 14, 2020, 10:13 p.m.	119.49	2	0	2	238.98	119.49	238.98	0.0

Figure 14

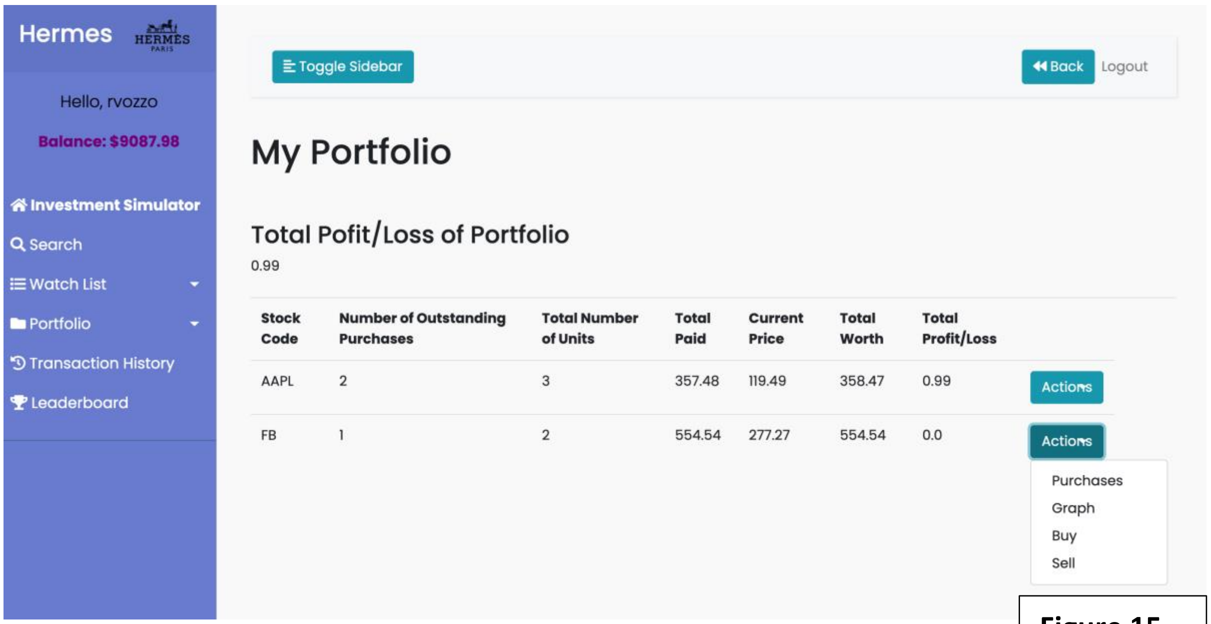
5.2.8 My Portfolio

Clicking on the 'Portfolio -> My Portfolio' tab will take you to a page tabling **aggregate** information about stock you have purchased, including the number of individual purchases, the total units purchased, the total amount paid, the current price, the total current worth, and the total profit/loss. What makes the 'My Portfolio' page different from the 'Purchase List' page is that all individual purchases of the same type of stock are aggregated together, meaning users can see the total profit/loss they have made for ALL purchases of a certain stock. Alternatively, the 'Purchase List' page is useful because users may purchase stock AAA for $10 one day, and $20 another day, meaning the profit they would make on these individual purchases will differ. An example of this can be seen in Figure 15, where the profit for the units of APPL purchased at 1:21pm is higher than those purchased at 10:13pm.

Each row in the portfolio also has an 'Actions' drop down, with an number of available actions:

- Purchases: Takes you to the 'Purchase List' page with the filter for that stock already applied.
- Graph: Displays an interactive graph of historical prices (see Section 5.2.4)
- Buy: Lets you buy that stock (see Section 5.2.6)
- Sell: Lets you sell that stock (see Section 5.2.9)

The total profit/loss of the entire portfolio (i.e. The total profit or loss you would make if all of currently stock units were sold at their current market price per unit), can also be viewed on the 'My Portfolio' page.



My Portfolio

Total Pofit/Loss of Portfolio

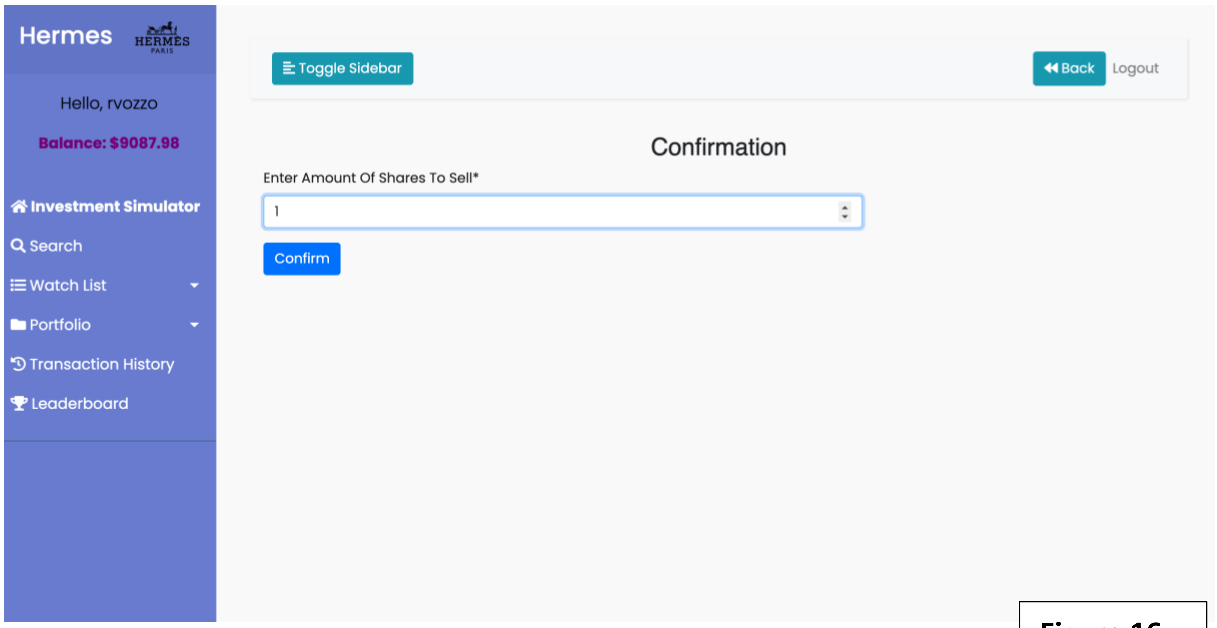
0.99

Stock Code	Number of Outstanding Purchases	Total Number of Units	Total Paid	Current Price	Total Worth	Total Profit/Loss
AAPL	2	3	357.48	119.49	358.47	0.99
FB	1	2	554.54	277.27	554.54	0.0

Figure 15

5.2.9 Sell Stock

Stocks can be sold from a number of pages in the system, including the 'Search' and 'My Portfolio' page. Clicking on the 'Sell' button on these pages will take you to a form where you can enter the number of units you would like to sell and a 'Confirm' button (Figure 16). Provided that you have enough units of the stock in your portfolio, the stock units will be sold and your account balance will be updated accordingly.



Confirmation

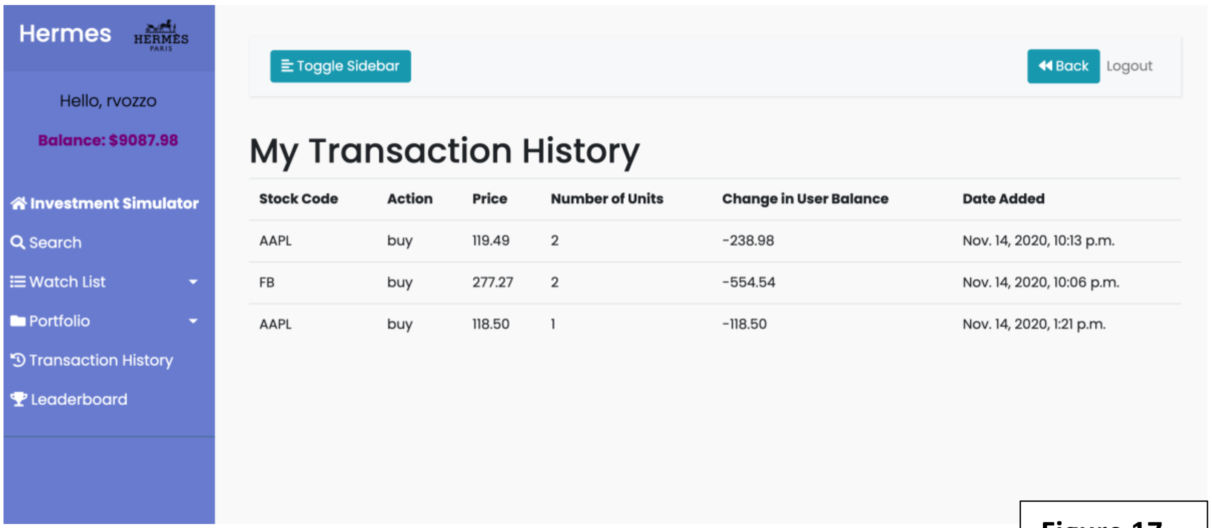
Enter Amount Of Shares To Sell*

1

Figure 16

5.2.10 Transaction History

Click on the 'Transaction History' tab to see a complete history of all the successful buy and sell orders you have made since the creation of your account (Figure 17).



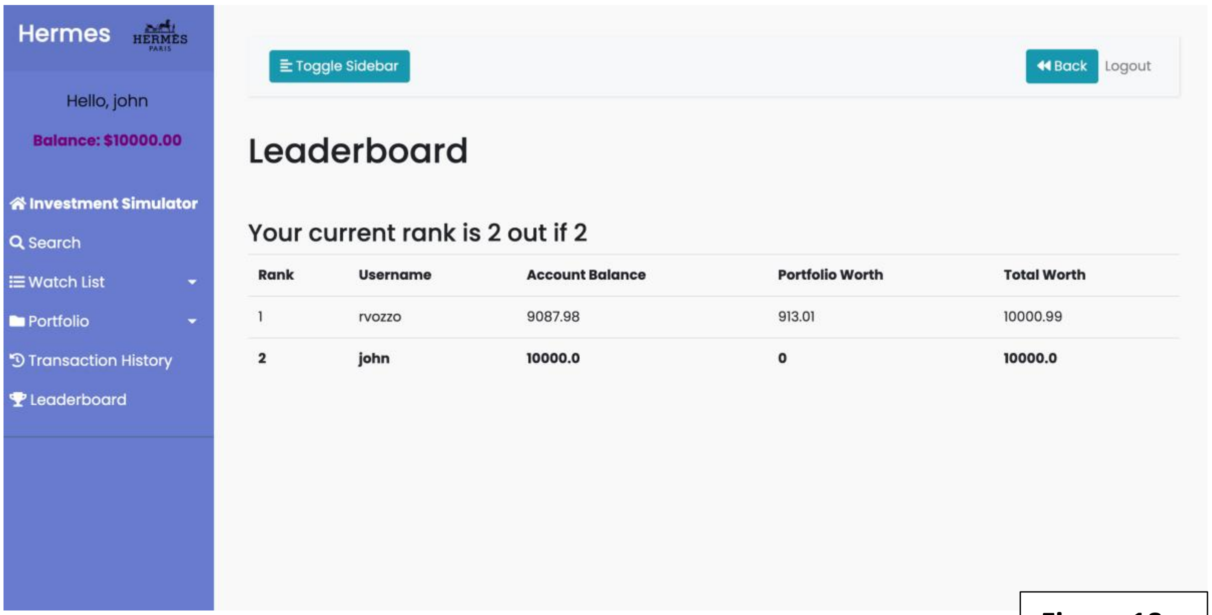
My Transaction History

Stock Code	Action	Price	Number of Units	Change in User Balance	Date Added
AAPL	buy	119.49	2	-238.98	Nov. 14, 2020, 10:13 p.m.
FB	buy	277.27	2	-554.54	Nov. 14, 2020, 10:06 p.m.
AAPL	buy	118.50	1	-118.50	Nov. 14, 2020, 1:21 p.m.

Figure 17

5.2.11 Leader board

Click on the 'Leaderboard' tab to view the account balance, portfolio worth, and total worth of all users who have an account with Hermes, ranked from highest total worth to lowest. The page will also display your current rank amongst the other users (Figure 18).



Leaderboard

Your current rank is 2 out if 2

Rank	Username	Account Balance	Portfolio Worth	Total Worth
1	rvozzo	9087.98	913.01	10000.99
2	**john**	**10000.0**	**0**	**10000.0**

Figure 18

6. Reference List

1. Candlestick Charts. (n.d.). Retrieved from plotly.com website: https://plotly.com/python/candlestick-charts/
2. Chandra, R. V., & Varanasi, B. S. (2015). *Python requests essentials*. Packt Publishing Ltd. website: https://docs.python.org/3/library/datetime.html
3. Finnhub.io. (n.d.-a). Finnhub - Free realtime APIs for stock, forex and cryptocurrency. Retrieved from finnhub.io website: https://finnhub.io/
4. Finnhub.io. (n.d.-b). Finnhub - Free realtime APIs for stock, forex and cryptocurrency. Retrieved from finnhub.io website: https://finnhub.io/docs/api
5. Heroku. (n.d.). Retrieved from dashboard.heroku.com website: https://dashboard.heroku.com/
6. threading — Thread-based parallelism — Python 3.9.0 documentation. (n.d.). Retrieved from docs.python.org website: https://docs.python.org/3/library/threading.html
7. Scikit-learn.org. 2020. 1.1. Linear Models — Scikit-Learn 0.23.2 Documentation. Available at: https://scikit-learn.org/stable/modules/linear_model.html
8. Chartjs.org. n.d. *Introduction · Chart.Js Documentation*. [online] Available at: <https://www.chartjs.org/docs/latest/> [Accessed 14 November 2020].
9. pandas - Python Data Analysis Library https://pandas.pydata.org/
10. MDN Web Docs. 2020. Django introduction - Learn web development | MDN. [ONLINE] Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. [Accessed 14 November 2020].
11. Timothy Ko. 2020. A Quick Glance of Django. An overview of Django for Beginners | by Timothy Ko | Medium. [ONLINE] Available at: https://medium.com/@timmykko/a-quick-glance-of-django-for-beginners-688bc6630fab. [Accessed 14 November 2020].
12. Wikipedia. 2020. Shared-nothing architecture - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Shared-nothing_architecture. [Accessed 14 November 2020].