Stock market trading simulator

Joshua Tomsett

Contents

[Analysis 3](#_Toc95467672)

[Problem Identification 3](#_Toc95467673)

[Introduction 3](#_Toc95467674)

[Stakeholders 3](#_Toc95467675)

[Why a Computational Solution? 3](#_Toc95467676)

[Research 5](#_Toc95467677)

[Existing solutions 5](#_Toc95467678)

[Research of features 8](#_Toc95467679)

[Features 9](#_Toc95467680)

[Essential Features 9](#_Toc95467681)

[Limitations 10](#_Toc95467682)

[Requirements 11](#_Toc95467683)

[Hardware 11](#_Toc95467684)

[Software 11](#_Toc95467685)

[Success Criteria 12](#_Toc95467686)

[Design 14](#_Toc95467687)

[Version Breakdown 14](#_Toc95467688)

[User Interface 15](#_Toc95467689)

[Version 1 15](#_Toc95467690)

[Version 2 15](#_Toc95467691)

[Version 3 16](#_Toc95467692)

[User Input 17](#_Toc95467693)

[Decomposition 18](#_Toc95467694)

[Version 1 18](#_Toc95467695)

[Version 2 23](#_Toc95467696)

[Version 3 28](#_Toc95467697)

[Classes, Functions, and Variables 32](#_Toc95467698)

[Classes 32](#_Toc95467699)

[Functions 33](#_Toc95467700)

[Variables 34](#_Toc95467701)

[Validation 35](#_Toc95467702)

[Test Plan 36](#_Toc95467703)

[Version 1 36](#_Toc95467704)

[Version 2 37](#_Toc95467705)

[Version 3 38](#_Toc95467706)

[Post Development 39](#_Toc95467707)

[Development 40](#_Toc95467708)

[Version 1 40](#_Toc95467709)

[Development 40](#_Toc95467710)

[Testing 50](#_Toc95467711)

[Review 53](#_Toc95467712)

[Version 2 54](#_Toc95467713)

[Development 54](#_Toc95467714)

[Testing 62](#_Toc95467715)

[Review 66](#_Toc95467716)

[Version 3 67](#_Toc95467717)

[Development 67](#_Toc95467718)

[Testing 78](#_Toc95467719)

[Review 82](#_Toc95467720)

[Full Code Listing 83](#_Toc95467721)

[Python 83](#_Toc95467722)

[Kivy 89](#_Toc95467723)

[Evaluation 97](#_Toc95467724)

[Testing 97](#_Toc95467725)

[Regression testing 97](#_Toc95467726)

[Testing against success criteria 99](#_Toc95467727)

[Testing for robustness 105](#_Toc95467728)

[Testing for usability 106](#_Toc95467729)

[Usability Features 107](#_Toc95467730)

[Maintenance 109](#_Toc95467731)

[Limitations 110](#_Toc95467732)

[Future Development 111](#_Toc95467733)

[Conclusion 112](#_Toc95467734)

# **Analysis**

## Problem Identification

### Introduction

I plan to produce a program capable of allowing users to trade virtual stocks and shares in a risk-free environment. The stocks and shares being traded are real companies and use real prices. The prices and historical data, used to create the graphs, will be retrieved from the Yahoo finance API and I will use matplotlib to visualize the data. All user and stock data will be stored in a database which I will access using sqlite3.

### Stakeholders

The stakeholders interested in my project would be people wanting to learn how to trade on the stock market, or to improve at it. They could be of any age group, but most likely 16+. My project allows user to train in a risk-free environment, as they would be trading with fake currency, so their personal funds are not at risk.

Another stakeholder could be a GCSE or A level business teacher wanting to teach their students about the stock market. This would be ideal for them as it is free and simple to use, while still being accurate.

The criteria of my stakeholders would a program that allows them to buy and sell stocks and shares using real time prices. Because of this, I will use the Yahoo finance API to get the data and a user interface to allow the users to interact this the program.

### Why a Computational Solution?

My project suits a computational approach as attempting to practise trading on the stock market without a computer would be very tedious. A computational approach allows data to be easily accessed and viewed through graphs. It also allows for easy tracking of a user’s current positions and fault proof algorithms to buy and sell stocks, as no mathematical errors will take place by a computer whereas if done by people calculation errors are inevitable. Using a computational approach also allows for the use of a database. This means that all data can be stored easily and allows for multiple users to use the program.

There are other benefits to this solution regarding specific methods of programming. The method of **decomposition** is useful in this instance as I can plan components of my code I want to solve individually. In my case, my main steps are:

1. Retrieving data from the API and creating graphs with matplotlib
2. The algorithms for buying and selling shares
3. The database that stores the user’s information
4. Kivy GUI to displaying the user interface for users to interreact with

This will allow for me to focus on each step at a time when making my program and allows for easy testing of each component. The first version of my code will contain the first and second steps of this decomposition. The second version will integrate multiple users through a database as described in step 3. The third and final version will incorporate a Kivy GUI shown in step 4.

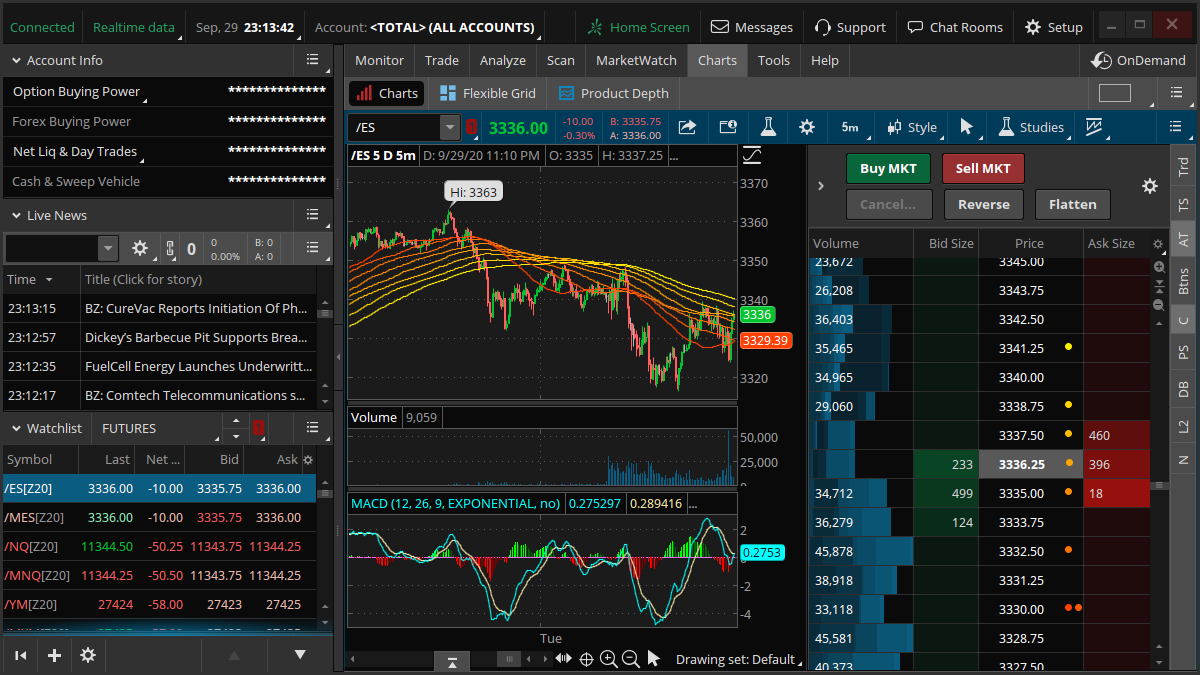
Another method I can make use of is **abstraction** as it allows me to reduce the workload for the project and focus on my main steps to produce a functional virtual stock trader. Some examples of my potential abstraction are as follows:

* Graphs are not interactable as matplotlib does not allow for it
* Graphs can only display data for a week, month, or year. This is because to generate a graph you need to retrieve the data and then plot it. So, adding more date options would take much longer
* Limited information about user’s current stock positions. Adding more information means the database will get larger much faster therefore taking up more space
* Not able to find a ticker without first adding it to the database. The database is stored on the user’s local device so having every ticker in it would take up a large amount of space, also the program displays all the tickers as buttons so having them all in the database would mean slower performance.

## Research

### Existing solutions

**Thinkorswim**

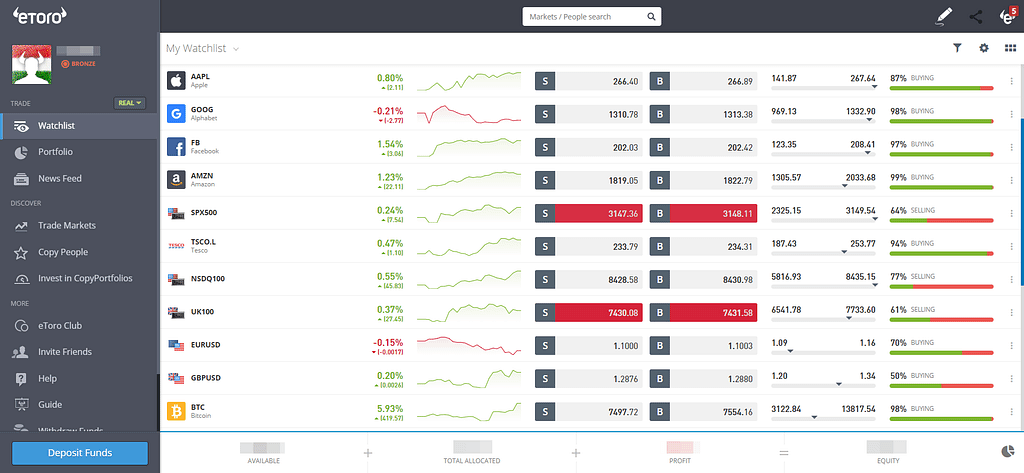
Thinkorswim is a professional-level trading platform made by TD Ameritrade for trading. It gives users an enormous number of tools as well as coming in web, desktop, and mobile form.

Source : <https://www.tdameritrade.com/tools-and-platforms.html>

|  |  |
| --- | --- |
| **Features** | **Essential Feature Y/N** |
| Graph to display data | **Yes**, as the user needs some way of seeing how the stock is doing. |
| Ability to have multiple graphs | **No**, as it will take time to create and display the one graph so adding multiple would cause the program to be very slow. |
| Buy and Sell buttons | **Yes**, as it is essential for the users to use the program. |
| Live news | **No**, as I have no way of getting the news into the program. |
| Analysis tools | **No**, as I am using matplotlib and they do not offer those tools. |
| Portfolio screen | **Yes**, as users must be able to see and manage the stocks they currently hold. |
| Multiple platforms | **No**, I will only be making a desktop app version as it will be made in python so converting to a web app would be very difficult and a mobile version would involve having to redesign the layout and extra optimization. |

**eToro**

EToro is a very popular trading platform with a modern design and a lot of useful tools. Such as the CopyTrader™ system which allows users to copy popular investors.



Source : <https://www.etoro.com/trading/platforms/>

|  |  |
| --- | --- |
| **Features** | **Essential Feature Y/N** |
| Graphs to display data | **Yes**, as the user needs some way of seeing how the stock is doing. |
| Buy and Sell buttons | **Yes**, as it is essential for the users to use the program. |
| News Feed | **No**, as I have no way of getting the news into the program. |
| Portfolio | **Yes**, as users must be able to see and manage the stocks they currently hold. |
| Social system | **No**, as the program will only run client side there is no network so no way to connect users to each other. |
| CopyTrader™ | **No**, as there would be no one else using the app so no one to copy. |
| Demo account | **Yes**, the purpose of my program is to act as a demo account by using virtual funds. |

**Trading 212**

Trading 212 is the UK’s largest trading platform with over 15 million users. They offer a modern trading app and allows users to have either a invest, ISA, or CFD account.

Source : <https://www.trading212.com/>

|  |  |
| --- | --- |
| **Features** | **Essential Feature Y/N** |
| Graphs to display data | **Yes**, as the user needs some way of seeing how the stock is doing. |
| Buy and Sell buttons | **Yes**, as it is essential for the users to use the program. |
| Portfolio | **Yes**, as users must be able to see and manage the stocks they currently hold. |
| Demo account | **Yes**, the purpose of my program is to act as a demo account by using virtual funds. |
| Different account types | **No**, as the user is using virtual funds different account types are not needed. |
| Auto invest | **No**, as the purpose of the program is for users to decide which stocks to buy so an auto invest system would defeat the purpose of my project. |
| Multiple platforms | **No**, I will only be making a desktop app version as it will be made in python so converting to a web app would be very difficult and a mobile version would involve having to redesign the layout and extra optimization. |

### Research of features

**Kivy / KivyMD**

Kivy is an open-source python library for applications that make use of user interfaces. It is not very popular, so I had to learn most of it from the [Kivy documentation](https://kivy.org/doc/stable/). Upon researching Kivy I stumbled across KivyMD, an extension to Kivy where the widgets are designed after [Googles Material Design](https://material.io/). I decided to primary use KivyMD as in general it just looks better. KivyMD is also not very popular, so I learnt almost all of it from the [KivyMD documentation](https://kivymd.readthedocs.io/en/latest/)



The graphs used in the program are generated using matplotlib, in their original form they cannot be displayed in kivy. While researching I found a kivy add on called [kivy garden](https://kivy.org/doc/stable/api-kivy.garden.html). This add on has a widget called FigureCanvasKivyAgg, which basically allows the graph to be drawn onto a canvas on the program.

**Hashing**

For the user’s security their passwords must be encrypted before being stored in the database. To do this I needed to find a way to hash the password, python has a built-in library for hashing called [hashlib](https://docs.python.org/3/library/hashlib.html) so I decided to use that as it allows for hashing in a single line of code.

**Yahoo finance API**

From the beginning of my project I knew I would need some method to get data into the program, after looking at all the options I decided to use the [Yahoo Finance](https://uk.finance.yahoo.com/) API. It allows me to get current and historical data so users will know the current price as well as have the historical data plotted on a graph within the program. The Yahoo Finance API is also free to use. To access the API, I need to use [Pandas-datareader](https://pandas-datareader.readthedocs.io/en/latest/) and yfinance.

## Features

### Essential Features

|  |  |  |
| --- | --- | --- |
|  | Feature | Explanation |
| 1. | Graph to track the price of the share against time | The main purpose of the program is for users to analyse data and then decide on whether to buy the stock or not, because of this it is essential for users to be able to see the data in a more visual way. A graph works well as it allows users to quickly see highs and lows. |
| 2. | Buy and sell buttons | For users to buy and sell they obviously need some way of telling the computer they want to do so. The best way to do this is through buttons as it only needs a single mouse click. |
| 3. | Text input for the amount a user’s wants to buy or sell | Without this user’s would only be able to buy or sell a single share at a time, which in almost all cases would be very inconvenient. |
| 4. | Text with information about the currently selected stock. | When a user selects a stock to look at they should be presented with some information about the stock, such as the current market price. This will be displayed next to the buy and sell buttons for users to see easily. |
| 5. | Scrollable list of buttons for each stock  There may need to be a limit at 3000 tickers | For the users to select a stock I decide to use buttons in a scrollable list, is must be scrollable as the number of stocks will grow over time. |
| 6. | Button to suggest a stock to be added to the list of stocks | It is essential for users to be able to add new stocks, otherwise you could not buy anything you have not bought before. |
| 7. | Search bar for the list of stocks | Once the list gets bigger it will become harder to find the stock you are looking for so a search bar will allow users to find any stock quickly. |
| 8. | Tabs at the top of the screen | Essential for users to switch between the different screens (main screen, portfolio screen, and settings screen). |
| 9. | Text to display the users current balance | The users will need to know what their current balance is so they can decide how much they want to spend in the future. |
| 10. | The portfolio | For the users to see what stocks they own there must be a portfolio section. I did this by having a scrollable list with custom widgets that display the stock ticker, current market price, the total units the user has, and the total value of all the units they own. |
| 11. | The settings | In the settings screen it is essential for users to have a way to sign out of their account. This could be because they want to login to a different account or just want to log out for security. |
| 12. | Login and sign-up screen | For users to be able to login to their account they will of course need a way to input their username and password, there must also be a button to sign up as otherwise no one would be able to create an account |

### Limitations

|  |  |  |
| --- | --- | --- |
|  | Feature | Explanation |
| 1. | Interactable graphs and analysis tools | I will using matplotlib to create the graphs and then a canvas widget to draw the graph as an image, because of this I am unable to have the graph be interactable. I am also not able to integrate any analysis tools as they would take too long and be too difficult to make. |
| 2. | Multiple graphs | It would be possible to have multiple graphs, but due to the time is takes to retrieve the data and then plot said data onto a graph means having more than one will just take too long. It is also not needed for a simple simulation. |
| 3. | News feed | I have not found any method to retrieve this kind of data so I of course cannot include this feature. It is also not needed for functionality, so maybe not even wanted by stakeholders. |
| 4. | Can only see graphs for week, month, or year | As I said previously making a graph takes time and the program works by generating graphs for the week, month, and the year at the same time so adding more time options would mean the program taking longer to load after a ticker button is pressed. |
| 5. | Users must manually add stocks to the program | I could possibly be able to get every ticker into the database, but if a ticker expires or when a new ticker is created the database won’t be updated so over time it will become obsolete. Also, the database will take up much more space with every ticker in it, the program also displaying each ticker in the database as a button so the user can select one, so having every ticker would mean the program takes far too long to start up. |

## Requirements

### Hardware

**A computer**: A computer is required to run the program, almost any computer will suffice but it would be recommended to have plenty of spare storage and memory.

**peripherals**: Just a mouse, keyboard, and a monitor are required to interact with and see the program.

### Software

**Windows 10 operating system**: This is one of the standard operating systems. I have decided to use this over the others as I am most familiar with it, and it is the most available operating system to me. Python supports it.

**Python interpreter**: The program is written in Python 3.9, therefore a Python interpreter is required to read and run the code.

**Python Modules**: The program will have a lot of dependencies, all of which are listed below:

* Pandas-datareader – used for reading data from the yahoo finance API

<https://pandas.pydata.org/>

* Matplotlib – used to create graphs

<https://matplotlib.org/>

* Kivy – A user interface library

<https://kivy.org/#home>

* kivyMD – A collection of Material Design compliant widgets for use with Kivy

<https://kivymd.readthedocs.io/en/latest/>

* sqlite3 – Library for accessing databases

<https://docs.python.org/3/library/sqlite3.html>

* hashlib – Library consisting of many hashing algorithms

<https://docs.python.org/3/library/hashlib.html>

* yfinance – Library for interacting with the yahoo finance API

<https://pypi.org/project/yfinance/>

## Success Criteria

|  |  |  |
| --- | --- | --- |
|  | Criteria | Explanation |
| 1.1 | Login – username and password | Is the user asked to enter a username and password upon launching the program? |
| 1.2 | Login – database verification | Once the user has entered a username and password, are these compared to the database to see if they are valid inputs? |
| 1.3 | Login – completed login | Once a login has been successfully verified, is the user sent to the main menu?  Is the user’s data loaded into the program from the database? |
| 2.1 | Signup – username, password, and re-password | Is the user able to input a username, password, and Re-Password? |
| 2.2 | Signup – database verification | Once the user has entered a username, password, and re-password are these successfully validated? |
| 2.3 | Signup – completed login | If valid, are the entered username and password stored in the database?  Is the user sent back to the login screen? |
| 3.1 | Ticker buttons – displaying the buttons | Are the tickers stored in the database taken and displayed as buttons within the program? |
| 3.2 | Ticker buttons – button functionality | When a button is pressed is a graph, and stock information displayed. |
| 3.3 | Ticker buttons – search field | Is there a search field so that users can easily find the ticker they want? |
| 3.4 | Ticker buttons – suggest button | Is there a button that allows users to add tickers to the database?  Once successfully adding a ticker, is it displayed in the program? |
| 4.1 | Graph – the graph | When a ticker button is pressed, is a graph displayed plotting the stock’s price against time?  Is the default time range a month? |
| 4.2 | Graph – time buttons | Are there buttons for the user to change the time range of the graph? |
| 5.1 | Buying and selling - buttons | Is there a buy button and a sell button?  Is this buy button green and the sell button red? |
| 5.2 | Buying and selling – amount input | Is there a text input field for the amount? |
| 5.3 | Buying and selling – verification | Does the program know what the current ticker is?  For selling, does the user own the ticker and do they have enough to sell?  For buying, does the user have enough to afford the amount they have entered?  Is the user ask to approve the transaction? |
| 5.4 | Buying and selling - database | Does the user entry table get updated for the user’s new portfolio?  Does the users table get updated for the users’ new balance? |
| 6.1 | Portfolio - balance | Is the users balance retrieved from the database and displayed to the user? |
| 6.2 | Portfolio – stock positions | For each stock the user owns is the ticker, units owned, current price, and total value displayed? |

# **Design**

## Version Breakdown

Before starting my design, I need to break down this project into individual steps for each version, together these steps will form a final version.

The features of version 1 are the essential criteria for a functioning simulation. Users must be able to see their portfolio and must be able to buy or sell a stock, graphs are not essential to functionality but are an important feature to include in all versions.

Version 2 adds the database, allowing multiple different users or one user with multiple different accounts. With the introduction of a database there must be a method of logging in and signing up. It also offers more and higher quality data such as information about the current stock and different time options for the graphs.

Version 3 adds the user interface. The three main sections are the main screen, the portfolio, and the settings. The main screen will have buttons to select a stock, a graph with different time options, text information about the stock, and buttons to buy and sell as well as a text input for the amount. The portfolio will have the users balance at the top and then the user’s current positions, displaying the stock name, units owned, current market price, and total value of all your units. Finally, the settings screen has options to sign out, in order to change account, and to change the colour theme. There is also screens to login and sign up with text inputs for the username and password.

## User Interface

### Version 1

The User interface for version 1 will be a simple and text based.

Text

Description automatically generated

This is the intended look for the version 1 user interface. As you can see there will be six options and numbers next to each for the user to knew what to input. Following this output, the user will be presented a text input allowing them to select an option. Option one simply outputs the users balance, option two outputs the current market price of the selected ticker as well as a graph for this week’s prices. Option three outputs the users current stock positions, option 4 and 5 allow users to buy or sell a stock, and finally option 6 just exits the program.

### Version 2

The main interface in version 2 will be identical to that of version 1, as the primary introduction of version 2 is the database. With the introduction of a database comes the possibility for multiple users, this means the user will need some way of telling the program which account to use. To achieve this, I will implement a second menu that will appear before the main menu and ask the user to login or signup.

Text

Description automatically generated

This is the intended look for the account menu, it is the same style as the main menu just a simple numbered text list.

### Version 3

Version 3 will introduce the Kivy GUI, this will be a much better user experience as the text-based user interface can get very messy. Below is a mock design for my planned GUI.

Text

Description automatically generated

This will be the login screen, a very simple design consisting of two inputs and a button. The sign-up screen will be almost identical to this but will have one extra input for the re-password.

Graphical user interface, application

Description automatically generated

This will be the main screen the user will spend most of their time on, the buttons on the side allow users to select a stock. Upon a stock being selected a graph will appear in the space in the top right, information about the stock will be displayed in the space below and to the right. Finally, there are buy and sell buttons as well as an input for the amount. At the top there will be tabs to switch between the main screen, portfolio, and settings screen.

Table

Description automatically generated

This will be the portfolio, which displays the users balance at the top and then each of the user’s current stock positions. For each stock position the stock ticker, units owned, market price, and total value will be displayed.

## User Input

Only the basic keyboard and mouse are required, for versions 1 and 2 the user interface will be text based, so only the keyboard is required. For version 3 the mouse will be essential due to the introduction of the GUI.

## Decomposition

### Version 1

The objective of version 1 is to create a basic functional program allowing a single user to practise trading.

Diagram

Description automatically generated

**Main Menu**

Upon running the code, the user should be prompted with a main menu. After this an input shall be collected for the corresponding choice, this input should be in the range of 1 – 6. When the input is received the program should execute the code for the choice chosen.

**Pseudocode for displaying the main menu:**

Print("""

1 – View balance

2 – View Stock Data

3 – View Portfolio

4 – Buy

5 – Sell

6 - Exit

""")

**1**

Option 1 should access the csv file and collect the user’s balance, then simply output it to the user.

**2**

Option 2 will ask the user to enter a stock ticker, one verified the stocks data is to be displayed. There will be two different display methods, first the current price will be outputted to the console, and secondly a matplotlib graph of the stock from the past week will be displayed.

**3**

Option 3 will access the csv file and retrieve all the user’s stock positions, then each one will be outputted to the console.

**4 & 5**

Options 4 and 5 will ask the user to enter a stock ticker, then update the user’s data. **See how these options will work in trading section**

**6**

Option 6 will just run the exit function, closing the console.

**Trading**

**Buying**

In version 1 the process for buying is very simple. The user will be asked for three different inputs, the ticker, the amount, and to confirm the purchase. Each input will come at a different stage in the transaction. First the user entered ticker will be validated making sure is exists, then the amount will be asked and the program will confirm the user has enough to make the purchase. Finally, the user is asked to confirm the purchase.

The user data in version 1 will be stored in a single csv file, once a transaction has finalized that csv file will need to be updated. This process is quite simple, when a csv file to be read or written in python the contents are stored in a list. When the user makes a purchase, this list can be updated and then written out to the file.

Below is the algorithm for buying a stock in the form of a flow chart.

Diagram

Description automatically generated

**Selling**

In version 1 selling works similarly to buying, but instead of adding to the portfolio you subtract from it. The user is first asked for the stock they want to sell, then the amount they want to sell, and finally to confirm that they are selling. Of course, each of these inputs are validated before moving to the next stage.

Updating the csv file is done by removing the stock from the list if the user sells all of it and updating the amount they own if the user only sells some of it. The process of updating the csv file is the exact same as buying.

Below is the algorithm for selling a stock in the form of a flow chart.

**Diagram

Description automatically generated**

**Csv file**

In version 1 user data is stored in a csv file. I decided to use a csv file over a database as a goal for version 1 is a simple functional program, this means a complex database is not needed until later versions.

The csv file’s structure is quite simple with the first element in the first row is the balance and then the rows after that are the user’s portfolio, as shown below.

**Table

Description automatically generated**When the file is read in the balance should be set in a variable and then the first element of the list will be deleted, leaving just the portfolio.

### Version 2

The main feature of version 2 is the database. This will add the possibility for multiple users as well as removing the need for a csv file.

**Database**

To interact with the database, I am planning to use a built-in python module called sqlite3 I have decided to use this library as it implements a fast and reliable database that does not take up much room SQLite is the most used database engine in the world and therefore is built into many different devices.

In its first form the database consisted of two tables: Users, and Stocks. These formed a many to many relationship, which is not recommended as it can lead to duplication of data.

A picture containing table, worktable

Description automatically generated

To solve the problem of a many to many relationship you must add another table, forming two many to one relationships. In this case the User Entry table to added. Now that the many to many relationship has been removed the database is in third normal form.

A picture containing text, table

Description automatically generated

As shown above the final form of the database consists of three tables Users, UserEntry, and Stocks. Users will store the username, password, and balance of each member. The username acts as a primary key, and as such must be unique for each member. The UserEntry table stores all data relating to a user’s portfolio, to identify the records the username and stockID come together and form a composite key this allows for each record to be unique and also allows the database to be queried for just a single users portfolio. Finally, the stocks table stores the stock ticker for each stock currently in use by any user, in version 2 this table could be removed but it is essential for version 3 so I have decided to implement it at this stage.

|  |  |  |
| --- | --- | --- |
| Users | UserEntry | Stocks |
| Username (PK) | Username (CK) | StockID (PK) |
| Password | StockID (CK) |  |
| Balance | Amount |  |

**Login**

The process for logging in will be as follows:

* Get the user to input a username and password
* Hash the password
* Get the password from the database that is connected to the entered username
* Compare the passwords
* Either log the user in or reject the inputs

This system works simply by using the username to get the password, which is stored in the database hashed, and then comparing that retrieved password to the entered one that has been hashed. If the two passwords are the same the user will be sent to the main menu and the users’ settings will be set, if the passwords do not match then the inputs will be rejected and the user will be asked to enter different ones.

pseudocode for login

username = input("Enter username: ")

password = input("Enter password: ")

password\_hash = hash\_func(password)

c = conn.execute(f"SELECT Password from USERS WHERE Username = '{username}'")

database\_password = c[0]

if password\_hash == database\_password:

menu()

else:

print("## Invalid password ##")

The hashing library I am using is hashlib, a built-in module that gives the user access to many different hashing algorithms, I have decided to use this library as it does not require external installation. The hashing algorithm I am using is MD5 as it is simple to implement and returns a result quickly.

**Signup**

The process for signing up is as follows:

* Get the user to enter a username
* Check the username is not in use
* Ask the user for a password
* Hash the password
* Create the account and store it in the database

Signing up is quite simple, the inputs are the same as logging in but what is done with those inputs is a bit different. As we saw in the database section the USERS table consists of a username, password, and balance. The username is the primary key and as such must be unique this is step two of the process above. Once the username is validated, the password is collected, checked, and then hashed. Finally, the username and password are stored in the database along side the default initial balance.

pseudocode for login

username = input("Enter username: ")

c = conn.execute(f"SELECT Username from USERS WHERE Username = '{username}'")

if c.length() == 0:

password = input("Enter password: ")

password\_hash = hash\_func(password)

conn.execute("INSERT INTO USERS VALUES(?,?,?)", (username,password\_hash,"10000"))

conn.commit()

else:

print(“## Username taken ##”)

**Portfolio**

In this database all portfolio data is stored in the userENTRY table. As we saw in the database section this table consists of three columns, the username, stockID, and amount. Together the username and stockID form the composite key to uniquely identify the record. When a user either buys or sells a stock the table is updated either adding a record, removing a record, or updating a record. The username will always be the userID of the user involved in the transaction, and the stockID is the stock ticker that the user is trading. The amount simply holds the number of shares the user owns.

Within the program when a user wants to view their portfolio an SQL query must be executed on the database. To get all stock positions for a single user the SQL statement must look something like this:

SELECT StockID, Amount from userENTRY WHERE Username = 'UserID'

This will return the stock and amount of each record with a common username. If the user wishes to see only a single record, the where command will be amended to include a condition for the stock ticker.

**Improving Data Presentation**

One of the small changes I want to make in version 2 is allowing the user to have more control over the data they want. To achieve this a will implement an option to enter the number of days the user wants to see when they are using option 2 of the main menu. Currently the user is only presented with the current market price and a graph of the previous week, in version 2 the user will have to input a number for the number of days that will appear on the graph. Below is pseudocode for how this will work.

try:

no\_of\_days = input("Number of days: ")

if no\_of\_days > 0:

print("Current price is:", get\_price(ticker))

create\_graph(ticker , no\_of\_days)

except:

print("\nInvalid number of days")

Integrating this feature is relatively simple. First the user is asked to enter the number of days, then the entered number is validated. The requirements for a valid input are a real integer greater than zero. To makes sure the input falls within the requirements the program will run a try except command so that any errors, in this case type errors, will be dealt with. This takes care of the real integer component, next the input is checked to be greater than zero. If it returns True we know that the input satisfies the requirements. Finally, I want to note why there is no limit to the number of days, this is due to a feature in the yahoo finance API library that removes any days with no data, so even if the user enters a number of days that is more than the number of days where data is available, the API will just return NAN values for those days.

### Version 3

The purpose of Version 3 is to introduce a GUI, this will improve user experience by allowing a nicer and faster method to interact with the program. The planned layout for this GUI is stated in the user interface section, it will consist of a login and signup screen and a main screen which has the trading tab, portfolio tab, and settings tab. Version 3 will also make some changes to the database, the first is making use of the stock table, and second is a change to the hashing algorithm.

**Database Changes**

**Stock Table**

The database was first introduced in version 2 and include three tables, to have version 2 completely functional only two were needed and the one that was not was the stock table. I decided to include it in version 2 as I knew it would be needed for version 3. In version 3 the stocks table is used to store the ticker name of all stocks the user currently has positions on and also all stocks the user has either finished with or is looking to invest in at a later date. The purpose of storing these ticker names is so that they can be displayed in the main screen, as shown in the user interface section no the left-hand side of the main screen there is a scrollable list of buttons, one for each ticker. These buttons allow the user to change the current stock loaded in the program. There is also a suggest button which allows the user to add tickers to the database and the list, they may do this if they either want to trade that stock or if they just want to look at the graph of it to analysis the data.

**Hashing Algorithm**

In version 2 I was using the MD5 hashing algorithm, after doing more research I discovered that it has many well documented weaknesses. For this reason, I have decided to change to hashing algorithm I am going to use in order to improve security. I was still limited by the algorithms available on the hashlib library, but I have now decided that the best hashing algorithm for my project is SHA-256 or secure hashing algorithm 256 where 256 is the digest size. The SHA-256 algorithms output size is twice as big as MD5 and also has a lower probability of collisions along side also not taking much longer to return a value there should be no negative effects of this new hashing algorithm. To implement SHA-256 all that is needed is to change a single line of code in the hash function I have made.

In version 2 the hashing function looks like this:

A picture containing logo

Description automatically generated

In order to change over to the SHA-256 algorithm all that is needed is to change the md5 to a sha256.

**GUI**

**The KV File**

The GUI library I am using is Kivy, an open-source python library for application development. Kivy is like any other GUI library except for the fact that it uses a separate file and a different language for the creation and styling of widgets. This other file is simply a Kivy file, using the “.kv” file name. The Kivy language is very simple to learn and read, similar to python it works by indentation and colons. Below is an example of a Kivy language program that displays a label with the words hello world on it.



As you can see the languages syntax is very simple and English-like, this code alone is not enough to display a GUI, but it is imported into the python code and then can be used to display to the user.

**KivyMD**

KivyMD is a collection of Material Design compliment widgets for use with Kivy, the purpose of this addon is to approximate Google’s Material Design spec and allow developers to build programs that mimic Googles iconic style. I have decided to incorporate KivyMD as the final product looks much better as well as having more user-friendly interface. Not only does it alter the existing widgets but it also adds many new ones, such as the tabs widget which I plan to use.

**Login Screen**

As stated in the user interface section below is the proposed login screen.

Text

Description automatically generated

Text inputs

Button

As you can see the design is very simple and minimalistic, it consists of two input fields and a button. The input fields act as any other the user may click on them and they enter text with the keyboard, once they have entered their details, they may click the login button. Once the login button has been clicked the username and password data is taken and verified the same way as in version 2. If the verification is successful the user is sent to the main screen, if it is not then the inputs are cleared and the user is asked to re-enter values. There will be no limitation as to the amount of times a user can try to login because it would take time and space to implement.

**Signup Screen**

The signup screen looks almost identical to the login screen, the few changes are that the user must re-enter their password before clicking the button, this is to avoid mistakes with the user accidently entering the wrong password when creating an account as if they don’t have the correct password they will be permanently locked out as they is no password changing feature. When the user has entered their information and clicked the button the data is taken as verified using the same code as in version 2.

**Main Screen**

The main screen is where the user will spend most of their time. It contains three sections trading, portfolio, and settings. The trading section is where the user will view data and make transactions, the portfolio is where the user can see their balance and current stock positions, and the settings will allow the user to sign out.

**1**

Graphical user interface, application

Description automatically generated

**5**

**4**

**3**

**2**

Above is the planned layout for the trading section, and I have number each of its features.

1. Each of the sections can be accessed by the tabs here.
2. This is the where the stored tickers will be displayed, each will be a button and upon being pressed will change the current ticker and graph on the right.
3. The buy and sell buttons as well as the amount input field, allows the user to trade stocks.
4. The graph that will display data for the current ticker, the default time range will be a month and the buttons on the right-hand side of the graph will change the time range of the graph.
5. This is where information about the current stock will be displayed such as the ticker name and current market price.

## Classes, Functions, and Variables

### Classes

The decision to use object-oriented programming came directly from the user interface library that I am using, because of this classes will only be used in version 3.

Graphical user interface

Description automatically generated

|  |  |
| --- | --- |
| Class MyApp Methods | What it does |
| Build() | Loads the kv ui file and creates the screen manager |
| On\_start() | Equivalent of \_\_init\_\_, just sets the initial attributes |
| Set\_user() | Sets the userID and changes to the main menu screen |
| Add\_item\_list() / portfolio() | Adds items to dictionaries for them to be displayed in the recycleview. |
| Amount\_error() | Creates a popup that notifies the user of and error involving the amount they have entered |
| Update\_graph() | Removes the old graph and replaces it with the new one |
| Ticker\_error() | Popup for an invalid ticker |
| Create\_graph() | Generates a new graph |

|  |  |
| --- | --- |
| Class LoginScreen Methods | What it does |
| Login() | Takes the username and password and validates them using the database, if they are valid the user is put into the main menu |
| signUp() | Sends the user to the sign-up screen |

|  |  |
| --- | --- |
| Class SignUpScreen Methods | What it does |
| signUp() | If the username and password are valid an account is created in the database |
| Back\_to\_login() | Once an account is created the user is sent back to the login screen |

### Functions

This section contains the functions used for each version, version 1 has all of its function, while versions 2 and 3 will only include new, and or modified functions.

Version 1

|  |  |
| --- | --- |
| Global Functions | What it does |
| get\_price(ticker) | Returns the current market value of a given ticker |
| Test\_ticker(ticker) | Returns a Boolean value for if the ticker is valid or not |
| Get\_balance(UserID) | Returns the users balance |
| Create\_graph(ticker) | Generates a new graph |
| Clean\_data(stock\_data) | Cleans the data that has just been retrieved by removing weekends and null values |
| Read\_csv(directory) | Reads data in from a csv file |
| Write\_csv(List\_ , directory) | Writes data out to a csv file |

Version 2

|  |  |
| --- | --- |
| Global Functions | What it does |
| Hash\_func(item) | Takes in the user password as a parameter and applies the MD5 hashing algorithm, it then returns the value obtained. |
| Get\_balance() | Query’s the database for the balance of the user from the users table. |
| Login() | First it displays a list of options to the user and collects an input, these options consist of login, signup, and exit. For the login the user enters a username and password, which are checked validated with the database. The signup option asks the user for a username, password, and re-password that are verified and then added to the database. |

Version 3

Version 3 does not add any new functions as all new features are implemented used class methods as shown above in the classes section, the addition of classes also means most of the global function used in previous versions can be integrated into classes or become class methods themselves.

### Variables

This section contains the notable or useful variables used for each version, version 1 has all of its notable variables, while versions 2 and 3 will only include new, and or modified variables.

Version 1

|  |  |  |
| --- | --- | --- |
| Variable | Type | What it is |
| Balance | Integer | Holds the user’s balance |
| User\_stocks | List | Holds the user’s portfolio |
| option | Integer | A value between 1 – 6 that the user enters to choose an option for the main menu |
| Amount | Integer | The amount of a stock the user wants to buy or sell |
| Ticker | String | The ticker the user has entered |

Version 2

|  |  |  |
| --- | --- | --- |
| Variable | Type | What it is |
| Conn |  | Establishes a connection to the database, this variable is used to do all SQL statements. |
| username | String | The users entered username |
| password | String | The users entered password |
| Password\_hash | String | The value returned when the password is hashed |
| UserID | String | The number used as primary key in the users database, this variable is used for SQL statements |

Version 3

|  |  |  |
| --- | --- | --- |
| Variable | Type | What it is |
| Current\_time | String | States the current graph to use (year, month, week) |
| Current\_ticker | String | Stores the name of the current ticker in use |
| Rv\_data | ListProperty | Stores the data that is used to display the stock buttons |
| Profile\_data | ListProperty | Stores the data that is used to display the items in the portfolio |
| Sm | ScreenManager | Is used to change between different screens using sm.current = “screen name” |

### Validation

Version 1

In version 1 there will be three different validation checks, testing if a ticker is valid, validation of buying, and for selling. For testing a ticker, I have a function test\_ticker() which when passed a ticker request data from the past week. If the data is retrieved successfully then the ticker is valid otherwise it is not. For buying and selling there is validation when the ticker is entered, and for the amount the user wants to buy/sell. If the user does not have enough in their balance to buy the amount, they have entered the transaction will terminate, or if the user is trying to sell shares, they don’t own the transaction will again be terminated.

Version 2

In version 2 the same validation from version 1 is carried over but a new component of the project must be validated, the database. When creating or logging into an account the username and password must be validated. For logging in the database is queried for the password using the username entered if the password entered is the same as the one from the database a login is successful. For creating an account, the username must be checked to be unique and the password must match the re-password.

Version 3

In version 3 the GUI is introduced; the validation is almost the same as version 2 but popups are used for either validating an input or notifying the user of an invalid input.

## Test Plan

### Version 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions and results | Criteria |
| 1.1 | Is the users balance correctly outputted | Enter option 1. | 6.1 |
| 1.2 | Does the current market price get outputted | Choose option 2 and enter a valid ticker. | 3.2 |
| 1.3 | Is a graph for the week displayed | Choose option 2 and enter a valid ticker. | 4.1 |
| 1.4 | Are all of the user’s stock positions displayed | Upon choosing option 3 the user’s portfolio should be outputted with each position on a new line. | 6.2 |
| 1.5 | Is everything verified when buying or selling a stock | When a user is buying or selling each of their inputs must be validated, and they must be asked to confirm the transaction before the balance or portfolio are altered. | 5.3 |
| 1.6 | Is the balance changed when a stock is bought or sold | The program should calculate the new balance and update the variable as well as the csv file. | 5.4 |
| 1.7 | Is the csv file updated upon buying or selling a stock | If a stock is bought for the first time it should be added to the csv file, if the user already owns the stock the value of the units owned should be updated. | 5.4 |
| 1.8 | Is the program halted | When the user chooses option 6 the program should stop running. | x |

### Version 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions and results | Criteria |
| 2.1 | Does the graph display data for the correct number of days | Choose option 2 and enter a number of days. | 4.1 |
| 2.2 | Does the portfolio only display the logged in users’ positions | Choose option 3 | 6.2 |
| 2.3 | Is the database correctly updated when buying or selling a stock | Buy or sell a stock. If buying a new stock, a new record should be added to the database, if buying or selling an already owned stock just updated the amount, if selling all positions of a currently owned stock remove the record from the database. | 5.2  5.3  5.4 |
| 2.4 | Does the user’s login get verified correctly | Select the login option and enter your username and password, the hashed password connected to the entered username should get collected and compared with the hashed version of the entered password, if they are the same the login is successful. | 1.1  1.2  1.3 |
| 2.5 | When signing up is the username checked to be unique | Select the signup option and enter a username, the database should be queried and return the usernames that match the entered one. If a value is returned the username is not unique. | 2.1  2.3 |
| 2.6 | When signing up is the password validated | Enter a password, the only password criteria for version 2 is that it is not a blank string. | 2.2 |

### Version 3

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions and results | Criteria |
| 3.1 | Does the user’s login get verified correctly | Enter your login details into the text inputs on the login screen, if successful the user should be sent to the main screen. | 1.1  1.2  1.3 |
| 3.2 | When signing up is the username and password validated | Select the signup option and enter a username, the database should be queried and return the usernames that match the entered one. If a value is returned the username is not unique. | 2.1  2.2  2.3 |
| 3.3 | Are the ticker buttons displayed and do they work | When the user is sent to the main screen, on the left-hand side there should be a list of buttons for each ticker in the database, when clicked they should change the graph and the currently stored ticker. | 3.1  3.2 |
| 3.4 | Is the search field displayed and is it functional, and does the suggest button work | When the user is sent to the main screen, on the left-hand side there should be a search field to filter the ticker buttons, and a suggest button to add tickers to the database and button list. | 3.3  3.4 |
| 3.5 | Is the graph correctly displayed and are there time buttons | On initially entering the main screen the graph should be blank, when a ticker is selected the graph should display the last month of data for that ticker, there should also be three time buttons next to the graph to change between week, month, and year. | 4.1  4.2 |
| 3.6 | Do the buying, selling, and amount elements work | Under the graph there should be buying and selling buttons, and an amount text input. When either the buying or selling buttons are clicked the amount is grabbed and validated in the respective algorithm. | 5.1  5.2 |
| 3.7 | Does the buying and selling validation work | Just try to buy or sell a stock and check the database before and after to see if everything is correct. | 5.3  5.4 |
| 3.8 | In the portfolio is the users balance displayed and are each of the user’s stock positions shown | At the top of the portfolio screen the user’s balance should be displayed and below it each of the user’s stock positions should be there with the ticker name, units owned, current price, and total value. | 6.1  6.2 |

## Post Development

During development I will make certain that all features work throughout all versions, when testing a new version, I will include tests of all previous versions. This will make sure nothing has disrupted the program during development and allows me to fix mistakes before going onto the next version.

# Development

## Version 1

### Development

**Python Modules**

The modules used in version 1 can be split into two groups. First the csv module which I am using to read and write from the excel file that stores user data, and secondly the modules that allow me to get and display stock market data.

Text

Description automatically generated

Pandas datareader and yfinance are the modules that deal with the retrieval of data. Yfinance directly accesses the Yahoo finance API and gets the data, then that data is sent through the imported data function to be stored as in panda’s data structure. Matplotlib is used for the plotting of data, and the structure the data is stored in allows for easy plotting due to synchronicity between pandas and matplotlib. Finally, datetime simply allows me to create a date range for when I want to get historical data from the API.

**Global Functions**

To stop the same code being written multiple times my program makes use of functions.

*Csv functions*

Each time any data is changed the csv file must be updated, to simplify this process I have written two functions, one to read and one to write.

The read function takes a directory as a parameter and then simply returns the data from that location in as list structure. The write function takes both a list of the data and a directory as parameters, it then just writes the list to the location.

Text, letter

Description automatically generated

*Price function*

The market value of a stock is needed often throughout the program for buying, selling, and displaying in the portfolio.

Text, letter

Description automatically generated

This function first takes a ticker as a parameter, then using the datetime module a date range is created from the time of executing to a week before. Data from this range is retrieved from the API and stored in the week data variable. That variable is then sent through the clean data function, which will be covered later, so that only the adjusted close data is included. Finally, the last item is returned, the last item will just be the most recent price.

*Ticker validation function*

Every time the user enters a ticker it must be verified.

Text

Description automatically generated

This function first takes a ticker as a parameter, then using the datetime module a date range is created from the time of executing to a week before. The program then tries to retrieve some data, if the ticker is valid the data will be stored as a panda’s data structure in the week data variable, if the ticker is not valid the week data variable becomes a string consisting of “Download Failed”. The program than returns false if the ticker is the string otherwise if returns true.

*Graph plotting functions*

Plotting a graph consists of three different functions, create graph, create plot, and clean data.

First the create graph function is called, taking a ticker as a parameter, it creates a date range and retrieves the data for the given ticker. It then passes the data through the clean data function which removes all but the date and adjusted close columns. It also removes any weekends as the stock market is closed on those days. The cleaned data is then returned and stored in the week price variable. Finally, the data is sent to the create plot function, the data is then plotted using matplotlib and the axis labels and title is set.

Text

Description automatically generated

**22**

**1**

1. Here a date range is created, the columns are removed, and the clean data is returned. When the columns are being removed, I only select the column that corresponds with the index of stock data, for this situation it is the adjusted close column as that is essentially the price. I then reindex over the date range, which just means all empty or null values will be removed, this is for removing the weekends to avoid the graph dipping to zero every Saturday and Sunday. Finally, I run the fillna method on the data, which just fills any NaN values.
2. Here I called the show method on the current plot, this very simply shows the user the graph in the matplotlib viewer, which I will cover in the menu section.

**Initializing the program**

Upon execution the program must do a couple of things, it must read in the data from the csv file and store the users balance and portfolio in variables.

Text, letter

Description automatically generated

**1**

**2**

**3**

**4**

1. Declaring the global variables.
2. This allows the program to use yfinance for requesting data but makes it so the data is returned in the panda’s format.
3. Read the data from the csv file and set the users balance and portfolio. The balance is just the first element in the list, while the portfolio starts from the second element, so the first is deleted.
4. Calls the menu function.

**Menu**

The main component of version 1 is the menu. As stated in the design section, upon running the code the user is presented with six options and asked to choose one. The menu is outputted using in a single statement and is displayed to the user in text as shown below.

Text, letter

Description automatically generatedText

Description automatically generated

*Input validation*

Text

Description automatically generated with medium confidence

When the user is asked to enter an option, the inputted number is checked to be greater than six or less than one, if this is the case the user is asked to enter another number. Once the user enters a valid number, the loop is broken. By default, the value of option is zero as it enters the loop.

*Option 1*

A picture containing chart

Description automatically generated

The balance is outputted and then the menu function is called, displaying the menu.

*Option 2*

The stock data option allows the user to enter a ticker and see the stock’s current price and a graph of the last week. The current price is displayed as text, while the graph is displayed in the matplotlib viewer as shown below.

Chart, line chart

Description automatically generated

Text

Description automatically generated

**1**

**2,3,4**

1. The user enters a ticker and it is checked using the test ticker function. If the ticker is valid the loop is broken and the input is set, if not the loop continues.
2. The tickers current price is outputted.
3. The graph is created and displayed using the create graph function.
4. The menu function is called

*Option 3*

Text

Description automatically generated with medium confidence

This option is used for displaying the user’s portfolio. First the title row is outputted, then the program goes through the user stocks variable outputting the ticker and amount for each item in the list. Finally, the menu function is called at the end.

*Option 4*

This option is for buying. There are multiple stages of inputs and validation for the ticker, amount, and to confirm.

1. Text

   Description automatically generatedThe user is asked to enter a ticker until a valid one is inputted.

**4**

**5**

**3**

**2**

**1**

1. This section deals with the amount. The user enters an amount if a non-integer value is entered the amount is set to zero. Next if the amount is less than or equal to zero, the program outputs “Invalid Amount”. If the amount multiplied by the stock price is greater than the users balance the program outputs “You do not have enough”.
2. Once the ticker and amount have been validated the user is asked to confirm.
3. If the user confirms this section is run. First, the program loops though the user’s portfolio and checks if the user already owns the stock they want to buy. If they do the amount the user owns is increased by the amount selected and their balance is decreased by the stock price multiplied by the amount. If the user does not already own the stock the ticker and amount is added to the portfolio and then the users balance is updated.
4. The end of this option deals with updating the csv file and then calling the menu function.

Text

Description automatically generated*Option 5*

**7**

**6**

**5**

**4**

**3**

**2**

**1**

1. Get the user to input a ticker, then loop through the user’s portfolio to check if they own any of the ticker stocks.
2. If the user does not have any of the ticker stocks, output “Invalid ticker”.
3. If the user does own stocks, get an input for the amount they would like to sell. If the input is not valid set the amount to zero.
4. Validating the amount. If the amount is more than the amount the user has or if the amount is less than or equal to zero, output “Invalid amount”.
5. Once the ticker and the amount have been verified, ask the user to confirm the transaction.
6. If the user confirms the transaction the program first calculates the new amount the user will own, this is just the current amount subtract the entered amount. Next check if the new amount is zero if so, delete the stock from the user’s portfolio and update their balance. If the new amount is not zero, update the amount owned in the user’s portfolio and the user’s balance. Finally, if the user does not confirm the transaction nothing will happen.
7. Update the csv file and then call the menu function.

*Option 6*

This option very simply calls the exit function, which halts the program.



### Testing

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions | What happened? |
| 1.1 | Is the users balance correctly outputted | Enter option 1. | The user’s balance was correctly outputted to the user. |
| 1.2 | Does the current market price get outputted | Choose option 2 and enter a valid ticker. | The current price of the selected ticker was outputted to the user. |
| 1.3 | Is a graph for the week displayed | Choose option 2 and enter a valid ticker. | A graph pops up in the matplotlib viewer. |
| 1.4 | Are all the user’s stock positions displayed | Upon choosing option 3 the user’s portfolio should be outputted with each position on a new line. | Each of the stocks in the user’s portfolio are displayed with the ticker and the units owned on each line. |
| 1.5 | Is everything verified when buying or selling a stock | When a user is buying or selling each of their inputs must be validated, and they must be asked to confirm the transaction before the balance or portfolio are altered. | All validation was successful and the user is asked to confirm before the transaction is completed. |
| 1.6 | Is the balance changed when a stock is bought or sold | The program should calculate the new balance and update the variable as well as the csv file. | The user balance was updated with the new amount and the csv file was also updated. |
| 1.7 | Is the csv file updated upon buying or selling a stock | If a stock is bought for the first time it should be added to the csv file, if the user already owns the stock the value of the units owned should be updated. | The user’s balance and portfolio were both updated in the csv file. |
| 1.8 | Is the program halted | When the user chooses option 6 the program should stop running. | The program was halted. |

**1.1 - PASS**

**A picture containing graphical user interface

Description automatically generated**

**1.2 - PASS**

****

**1.3 - PASS**

Chart, line chart

Description automatically generated

**1.4 - PASS**

A picture containing chart

Description automatically generatedTable

Description automatically generated

**1.5 - PASS**



**1.6 - PASS**

 Initial balance

 balance after purchasing AMZN for £3572

**1.7 - PASS**

Table

Description automatically generated

Portfolio before the transaction

Table

Description automatically generated

Portfolio after purchasing one share of Amazon for £3572

**Failed tests**

While none of the proposed tests failed, in doing the testing I noticed that sometimes the price of a stock was zero. This issue was being caused by the casting of prices to integers, I did this to make the program cleaner as having lots of decimals looks messy, but as a result any stock valued below £0.5 was set to zero.





This is the final line in the get price function, as you can see it is casting the value to an integer before returning it. To fix this issue I will need to instead cast it to a float, and cap it to two decimal places.

### Review

The objective for version 1 was to create a simple functional program that allows a single user to buy and sell stocks using virtual funds. I have successfully done this and in doing so have satisfied criteria 4.1, 5.2, 5.3, and 6 as stated in the analysis success criteria. I have not been able to satisfy criteria 1 or 2 as they require the presence of a database and have only partially completed criteria 4 and 5 as I am not using a GUI in version 1.

There were no failed tests in version 1 however I did discover a small bug with the pricing. In version 2 I will fix this issue by instead casting the price to a float, this may cause other problems as the program is designed to use integers. To deal with these potential new problems I will have to introduce extra tests for all elements of the code that deal with the price of a stock.

In version 2 alongside fixing the pricing bug I will be adding the database and making some improvements to option 2. The purpose of the database is to allow multiple different users to use the program. The introduction of a database will mean the csv file will not be needed anymore, as all data can be stored in the database instead. At the moment option 2 is very basic, you can only see a graph from the past week and the user has no option to change this time frame. In the next version I want to allow the user to enter the number of days they want to see on the graph. I could possibly add this feature to version 1, but I did not think of it until after finishing development and testing, so decided to add it to version 2 instead.

For version 2 I do not aim to change much of the logic, so the tests will be much of the same with a couple of additions such as the new features and most importantly the database.

## Version 2

### Development

This section will cover the development of version 2, as the main change in version 2 was the introduction of the database a lot of the program has changed and I will be focusing primarily on the changes made. There are also some new additions that come with version 2 such as the user system which will be covered in the login and signup section. I will go over all the code in version 2, but any sections that have little or no changes from version 1 will be covered very briefly.

**Python Modules**

The imports for version 2 are mostly the same. The csv module is no longer needed, so it has been removed, and two new modules are being introduced. The sqlite3 module will be used to handle all database creation and manipulation and the hashlib module is where I am getting my hash algorithm from.

Text

Description automatically generated

The last four modules are the same as in version 1.

**Database Creation**

To create the database, I used some one-off code in an external python script as shown below. Firstly, the tables are created and the various keys are defined. In the userENTRY table the composite key is set by declaring both the username and stockID as primary keys, and foreign keys. Once the tables are done, the connection is committed and all changes to the database are saved.

Text, letter

Description automatically generated

**Global Functions**

*Unchanged functions*

The ticker validation and graph plotting functions from version 1 are included in version 2 but the code behind them is unchanged. The csv functions are not in version 2 as the csv file system has been removed due to the introduction of the database.

*Price functions*

At the end of version 2 I found a bug that resulted in low value stocks having a value of zero. To combat to bug I alter the program to use two decimal point floats instead of integers, below is the altered code that implements this fix.

A screenshot of a computer

Description automatically generated with medium confidence

*Hash functions*

The hash function is relatively simple and is used for making the passwords more secure while they are stored. The hashing algorithm I am using is the MD5 hash, to access this algorithm I am using the hashlib python library as stated previously. The hash is also turned into hexadecimal values using the hexdigest method.

A picture containing text

Description automatically generated

*Balance function*

A picture containing logo

Description automatically generated

**2**

**1**

1. This line runs a simple SQL statement on the database, it selects the balance from the users table from the user with the username of userID.
2. This is where the results of the SQL query are returned to the user, the results from a database query are returned in a list data structure so the list must be iterated through to access the value.

**Initializing the program**

The set up in version 2 is much easier, a connection is made to the database, the API is set up, and finally the login menu is presented.

**22**

Graphical user interface

Description automatically generated with medium confidence

**3**

**1**

1. Creates the connection to the database.
2. This allows the program to use yfinance for requesting data but makes it so the data is returned in the panda’s format.
3. Runs the login menu.

**Login and Signup**

*Menu*

Text

Description automatically generated with medium confidence

The login menu is very similar to the main menu in design, the user is presented with options and then asked to choose one, for this reason the code is the same but with the options changed and the validation is between 1 and 3, due to the fewer options.

*Login*

A picture containing text

Description automatically generated

**3**

**2**

**1**

1. Get the user to input a username and password, then run the password through the hash function to get the password hash.
2. This section validates the username, the database is queried and if the entered username is found in the database it is valid.
3. Finally, if the username is valid then the password is retrieved from the database, this is then compared to the password hash. If they match the login is successful, the userID is set and the main menu is presented. If either is username or password is not valid, the user is told so and the login process is halted.

*Signup*

Text

Description automatically generated

**3**

**2**

**1**

1. The user is asked to enter a username.
2. This section is used to validate the entered username and make sure it is unique. The database is queried for the username of any user with the entered username, if the query returns any values the username is not unique and is rejected.
3. If the username is valid then the user is asked to enter a password, the password is validated to make sure it is not null and is then hashed and an account is created in the users table. The account consists of the username, hashed password, and initial stating balance of 10,000. The database transaction is then committed and the user is sent back to the login menu.

**Menu**

The main menu in version 2 is visually identical to that in version 1. There are some changes to the options which I will go through now.

*Option 2*

In version 2 option 2, I introduced the feature of allowing the user to enter the number of days they want to see on the graph. To do this the user is asked to input a value and if the value is an integer and greater than zero then the graph is created.

Text, letter

Description automatically generated

*Option 3*

The changes for option 3 consist of retrieving the portfolio from the database instead of from the csv file. The query very simply selects the stockID and amount from the userENTRTY table where the username is the current userID.

A picture containing logo

Description automatically generated

*Option 4*

The changes to the buying algorithm are only changes to do with the database, all the logic and validation is identical to version 1, but when the transaction has been confirmed the data is stored in the database instead of a csv file. You can see all changes as each changed line stars with conn and is followed by either an SQL statement or a transaction completion.

Text

Description automatically generated with low confidence

*Option 5*

The changes to the selling algorithm are the same as the changes to the buying algorithm. The only changes are to the storing of data in the database. The data is stored in the database using the insert and update SQL statements, inserting adds a new record, while update changes a certain value or values. The algorithm sometimes also uses the delete statement in the case that the user sells all their stock, the record is deleted and the balance in the user’s tables is increased.

Text

Description automatically generated

### Testing

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions | What happened? |
| 2.1 | Does the graph display data for the correct number of days | Choose option 2 and enter your desired number of days. | The graph is displayed with the correct number of days of data on it |
| 2.2 | Does the portfolio only display the logged in users’ positions | Choose option 3 | The portfolio was correctly displayed |
| 2.3 | Is the database correctly updated when buying or selling a stock | Buy or sell a stock. If buying a new stock, a new record should be added to the database, if buying or selling an already owned stock just updated the amount, if selling all positions of a currently owned stock remove the record from the database. | The database is correctly updated |
| 2.4 | Does the user’s login get verified correctly | Select the login option and enter your username and password, the hashed password connected to the entered username should get collected and compared with the hashed version of the entered password, if they are the same the login is successful. | The user’s login is successfully validated |
| 2.5 | When signing up is the username checked to be unique | Select the signup option and enter a username, the database should be queried and return the usernames that match the entered one. If a value is returned the username is not unique. | The program does notice if you enter a username that is already in user and it informs you of the error before halting the process |
| 2.6 | When signing up is the password validated | Enter a password, the only password criteria for version 2 is that it is not a blank string. | The password is not accepted if you enter a null value so the validation is successful |

Chart, line chart

Description automatically generated**2.1 - PASS**

Text

Description automatically generated

Above you can see I ask to see the past 100 days of GOOG, and on the right you can see a graph of GOOG starting from 100 days ago and ending at the day of this test.

**2.2 - PASS**

Table

Description automatically generated

As you can see only the records with the user of user123 are displayed on the portfolio.

Chart, scatter chart

Description automatically generated with medium confidence

**2.3 - PASS**

Table

Description automatically generated

The first image shows the database before any purchase, and the second image shows the database after buying 20 shares of TRI.L, as you can see the database is updated to reflect the users new stock position.

Table

Description automatically generated

**2.4 - PASS**

Text

Description automatically generated

To complete this test, I altered the code a bit to output the two passwords, as we can see the program retrieves the correct password from the database and the login is successfully completed.

**2.5 - PASS**

The username is already taken, the program notices this and informs the user that the entered username is taken.

Text, letter

Description automatically generated

**2.6 - PASS**

On the left we can see the password was accepted and, on the right, we see that when I enter a null value for the password it gets rejected.

Text

Description automatically generatedText

Description automatically generated

**Failed tests**

In version 2 there were no major issues, however in test 2.6 I did discover that when a user enters a blank string the hashing algorithm returns a null value. The issue comes from the fact that the database does not accept null values for the password column so an error occurs and the account creation is unsuccessful. I have already fixed this issue by simply validating the input to check for a null string, and if it detects one the user is asked to re-enter a new password.

### Review

The objective for version 2 was to create a fully functional program that allows multiple users to buy and sell stocks using virtual funds. I have successfully done this and in doing so have satisfied criteria 1, 2, 4.1, 5.2, 5.3, 5.4, and 6 as stated in the analysis success criteria. I have not been able to satisfy criteria 3, 4.2, and 5.1 as version 2 does not use a GUI. In version 3 the GUI is introduced and therefore all the final criteria will be satisfied.

There were no failed tests in version 2, however I did discover a small issue with the password validation which I have fixed, to make sure this issue does not appear in version 3 I will make sure to include the same semi-failed test (2.6) and while developing the code I will make sure the validation is only changed to accommodate new features and inputs.

In version 3 all the logic and backend development will remain the same as version 2 includes all the features I want to include except for the GUI. The main change in version 3 is the GUI and the changes it will make to the frontend of the program. Introducing a GUI will make the user experience much better and introduce new ways to interact with the program. The planned layout is shown in the design section under the user interface header, the login and signup screens will be very simple just text inputs and confirmation buttons, and the main program will consist of the data screen, portfolio screen, and settings screen.

## Version 3

### Development

**Python Modules and initialization**

**1**

A picture containing text

Description automatically generated

**6**

**5**

**4**

**3**

**2**

1. Importing the sqlite3 and hashlib libraries, these Are the same as in version 2.
2. Importing all the data retrieval and graph plotting libraries, these are also the same as version 2.
3. These are the Kivy imports, most are just widgets but a few serve a different purpose such as builder, and screen manager. I will cover these later in this development section.
4. These are the KivyMD imports, again most are just widgets except one. The MDApp is a class that I inherit from, this allows the user interface to run.
5. The first part for this is setting the window is maximized mode, and the second is removing the feature of the program closing when the escape key is pressed as it caused issues with accidental shutdowns.
6. This setup is the same as in version 2, just connect to the database and override the Yahoo finance API.

**Global Functions**

*Unchanged functions*

The following functions are unchanged in version 3

* Get\_price()
* Test\_ticker()
* Get\_balance()
* Clean\_data()

*Hash function*

As I said in the design section for version 3 I have changed the hashing algorithm from MD5 to SHA-256, to implement this all that is required is changing the hashlib method, which I have done as shown below.

A picture containing logo

Description automatically generated

**Custom widgets**

In some cases, I needed a widget but none of the pre-built ones were suitable, for this reason I had to learn to create my own custom widgets. To do this you have to define the widget class, inherit from the classes need, and define the parameters of the widget. Below are the two custom widgets I am using in my program, the buttons for the tickers and the portfolio items.

**1**

**Graphical user interface, text, application

Description automatically generated**

**2**

1. Define the class and inherit from the recycle view behaviour and box layout classes, to make the widget displayable.
2. Define the parameters of the widget, in this case the name of the ticker on the button.

To use these custom widgets, they must be styled in the KV file as shown below.

Text

Description automatically generated

**Login Screen Class**

The login class contains the methods and template for the login screen.

A picture containing timeline

Description automatically generated

**3**

**2**

**1**

1. Getting the username and password inputs and hash the password.
2. Login validation like version 2 but with some user interface changes such as when either the username or password is wrong the input fields are cleared and an error message is displayed to inform the user of the issue.
3. The method that is called when the sign-up button is pressed, it changes the screen to the sign-up screen.

The user interface design and styling all take place in the KV file, this is very lengthy so will not be shown here but all the KV file code will be shown at the end the development section.

**Signup Screen Class**

The sign-up class contains the methods and template for the sign-up screen.

Text

Description automatically generated

**3**

**2**

**1**

1. Get the user inputs for the username, password, and re-password. The password is then hashed.
2. Sign-up validation like version 2 but with user interface changes such as clearing input fields and displaying error messages.
3. The method that sends the user back to the login screen when an account has been successfully created.

As with the Login screen all the user interface design and styling is done in the KV file that will be shown at the end the development section.

**Main Screen Class**

As with the login and signup classes, this class also inherits from the Kivy screen class and acts as a template for the user interface on the KV file. This class contains no methods or attributes as all logic is done in the MyApp class instead.



The main screen is where the majority of the Kivy code is applied, it is where the trading, portfolio, and settings sections are defined and styled.

**MyApp Class**

*Setup*

The setup in version 3 happens in two stages, the user interface, and the account. The user interface is setup in the build method, which runs automatically upon running the program, the screen manager is created and each screen is added to it. The KV file is also imported using the builder load file function that comes with kivy. Next the on-start method is called, this initializes all the class attributes and sets up the ticker buttons with the add item list method which is explain later.

To set up the user’s account the set user method is called when a login is successful. The global variable userID is set and the user balance on the portfolio screen is updated. The code for both user interface and user account setup is shown below.

Graphical user interface, text, application

Description automatically generated

*Adding to recycle views*

The two methods below are used to display the ticker buttons and portfolio in their respective recycle views.

Text

Description automatically generated

**3**

**2**

**2**

**1**

**1**

1. Reset the current data to remove any pre-existing items.
2. Adding each item to the recycle view, this is done by appending them to their respective list property variable as dictionaries.
3. If the method is called from the search box the search parameter is true and the text parameter contains the text in the search box. If search is true then before an item is added to the recycle view it checks to make sure that the value of the item contains the text passed in.

*Popups*

Throughout the program multiple popups are used, for trading confirmation, error notifications, and user input. Here I will be describing the suggest popup as it is the most complex of them and therefore if you can understand it, you should be able to understand any of the others.

The first step is to define the popup, this is done as a MDDialog widget, the title, type, content, and buttons are set. The title is displayed at the top of the popup and the type just allows the content to be custom build as shown later. The content\_cls attribute allows the popup to inherit from the KV file using the UI defined under the suggest content class. Finally, the buttons are defined and displayed in the bottom right-hand corner.

Text, letter

Description automatically generated

The frontend of the popup is shown below, the design and styling are done in the KV file (top image) and final product is shown in bottom image. As stated before, the content of the popup is defined in the KV file and imported via the content\_cls attribute of the MDDialog widget.

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The following code is for closing and validating the popup. To close it the dismiss method is called, and the validation for this case is testing the entered ticker to make sure it is valid, if it is then the ticker is added to the database and the ticker button recycle view is updated.

Text

Description automatically generated

*Graphing*

The graphing in version 3 works quite differently, I am still using matplotlib so the creation of the graphs is the same, but the methods used to display the graph are completely different. The graphing process contains two sections, the creation and plotting of graphs, and updating the user interface.

In version 3 graphs are created when a new ticker is selected, I have decided to make it so that graphs for the week, month, and year are all create at once. My reasoning for this decision is to make it so that users won’t have to wait for graphs to be generated every time they switch between a time range, however this will mean that the time taken to switch between tickers is a bit longer. The code is not very different but instead of creating a graph for one time range is creates three.

Text

Description automatically generated

The code for plotting a graph is almost identical but with the small change that instead of showing the graph, the subplot is returned and stored as three graphs need to be generated and stored in their respective variables.

A screenshot of a computer

Description automatically generated with medium confidence

When the time range buttons are pressed the method bellow is run, first the current graph is removed and then the new graph is displayed depending on the time passed in.

Text

Description automatically generated

### Testing

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Instructions | What happened? |
| 3.1 | Is every ticker in the database displayed as a button | Look to see the tickers in the database and then check the program to see if all the same tickers are displayed there. | All the tickers were displayed |
| 3.2 | Does the search bar on the ticker buttons work | Click in the search bar and enter any text, the ticker buttons left should only be the ones that contain that text somewhere within them. | The search bar worked |
| 3.3 | Does the suggest button work | Click on the suggest button to enter a ticker. Try and then a ticker you know is valid and one you know is invalid, they should both function as expected. | Both inputs worked as expected |
| 3.4 | Does all input validation work | The validation for all inputs should work, the inputs are the amount, suggest popup, login, and signup. | All worked but the suggest popup which when a null value was entered, caused a crash |
| 3.5 | Is the user’s portfolio displayed correctly | Look to see the stock positions in the database and then makes sure all the correct ones are displayed in the program. | The portfolio was correctly displayed |
| 3.6 | When the time buttons are pressed, is the graph updated to the new time. | Click one of the time buttons and look to see if the time range has changed. | The graph was successfully updated |

**3.1 - PASS**

Table

Description automatically generatedGraphical user interface, application

Description automatically generated

All the tickers from the database are displayed as buttons in the program.

**3.2 - PASS**

Graphical user interface, application

Description automatically generated

As you can see above after entering “a” into the search bar only tickers with the letter A in them are displayed.

**3.3 - PASS**

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated with medium confidence

The suggested ticker was added to the database and displayed as a button.

**3.4 - FAILED**

For this test all but one input was successful, the input that did not function properly was the suggest popup, when a value of “” was entered, the test ticker function didn’t recognise the value and it caused a crash.

**3.5 - PASS**

To the left is all the records in the userENTRY table and bellow is the portfolio displayed within the program, as you can see all the user’s stock positions are displayed.

Table

Description automatically generated

Graphical user interface, application

Description automatically generated

**3.6 - PASS**

Chart, histogram

Description automatically generatedChart, line chart

Description automatically generated

Above there are two graphs, the first is AMZN from the past month and then after clicking the year button it changed to AMZN from the past year.

**Failed tests**

As stated in test 3.4 the suggest popup caused a crash when the value “” was entered, to fix this issue is quite simple and only requires a small addition to the validation this being an if statement that checks for the mentioned value, this statement is shown below.

Text

Description automatically generated

Another issue I found was overlapping on the x-axis of the graphs, an example is shown below.

A picture containing table

Description automatically generated

To solve this issue, the dates must be rotated so that they do not overlap. Luckily matplotlib has a built-in method to do this as shown below.

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text

Description automatically generated

### Review

In version 3 I have implemented the GUI, after completing the proposed tests and considering the tests from version 1 and 2 I can conclude that all the success criteria have been completed. There were some failed tests but as I have already shown they have been rectified. Version 3 is the final version of my project and is in a state I would call complete; I would expect my stakeholders to be satisfied with its quality.

## Full Code Listing

### Python

Text

Description automatically generated

Text

Description automatically generatedA picture containing text

Description automatically generated

Text

Description automatically generated with medium confidence

Text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A picture containing table

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

### Kivy

<CompanyButtons>

    MDRectangleFlatButton:

        size\_hint: 1 , 1

        font\_size: 20

        text: root.value

        on\_release: app.create\_graph(self.text)

<PortfolioWidget>

    BoxLayout:

        orientation: "horizontal"

        MDLabel:

            text: root.ticker\_name

            font\_size: 50

        MDLabel:

            text: root.units

        MDLabel:

            text: root.value

        MDLabel:

            text: root.total\_value

<SuggestContent>:

    orientation: "vertical"

    spacing: "12dp"

    size\_hint\_y: None

    MDLabel:

        markup: True

        text: '[ref=some]Find tickers at Yahoo Finance[/ref]'

        theme\_text\_color: "Custom"

        text\_color: 0, 0, 1, 1

        on\_ref\_press:

            import webbrowser

            webbrowser.open('https://uk.finance.yahoo.com/')

    MDTextField:

        id: suggestTicker

        hint\_text: "Ticker"

<MainScreen>:

    MDBoxLayout:

        orientation: "vertical"

        MDTabs:

            tab\_hint\_x: True

            allow\_stretch: True

            lock\_swiping: True

            tab\_bar\_height: 80

            # Main screen

            Tab:

                icon: "chart-line"

                GridLayout:         # page layout

                    cols: 2

                    BoxLayout:          # list of buttons layout

                        size\_hint: 0.6 , 1

                        orientation: "vertical"

                        spacing: "10dp"

                        padding: "20dp"

                        MDBoxLayout:

                            adaptive\_height: True

                            MDIconButton:

                                icon: "magnify"

                            MDTextField:

                                id: search\_field

                                hint\_text: "Search Company"

                                on\_text: app.add\_item\_list(self.text,True)

                        RecycleView:

                            id: rv

                            viewclass: "CompanyButtons"

                            data: app.rv\_data

                            RecycleBoxLayout:

                                orientation: "vertical"

                                default\_size: [0, dp(50)]

                                default\_size\_hint: 1, None

                                size\_hint\_x: 1

                                size\_hint\_y: None

                                height: self.minimum\_height

                        MDRaisedButton:

                            size\_hint: 1 , 0.09

                            text: "Suggest"

                            on\_release: app.suggest\_popup()

                    BoxLayout:          # layout for graph and buy/sell

                        orientation: "vertical"

                        GridLayout:

                            cols: 2

                            BoxLayout: # graph

                                orientation: "vertical"

                                id: graphLayout

                                Label:

                                    text: ""

                            BoxLayout: # time buttons

                                orientation: "vertical"

                                size\_hint: 0.1 , 1

                                MDFlatButton:

                                    size\_hint: 1 , 1

                                    font\_size: 20

                                    text: "Week"

                                    on\_release: app.update\_graph("week")

                                MDFlatButton:

                                    size\_hint: 1 , 1

                                    font\_size: 20

                                    text: "Month"

                                    on\_release: app.update\_graph("month")

                                MDFlatButton:

                                    size\_hint: 1 , 1

                                    font\_size: 20

                                    text: "Year"

                                    on\_release: app.update\_graph("year")

                        GridLayout: # buy /sell and price infomation

                            cols: 2

                            size\_hint: 1 , 0.3

                            BoxLayout: # buy /sell

                                padding: "20dp"

                                adaptive\_size: True

                                spacing: "20dp"

                                orientation: "vertical"

                                pos\_hint: {"center\_x": .5}

                                MDFillRoundFlatButton:

                                    text: "Buy"

                                    on\_release: app.buy\_stock()

                                    font\_size: 20

                                    pos\_hint: {"center\_x": .5}

                                    md\_bg\_color: 0,1,0,1

                                    size\_hint: 0.45 , 1

                                MDTextField:

                                    id: amount\_text

                                    hint\_text: "Amount"

                                    pos\_hint: {"center\_x": .5}

                                    size\_hint: 0.4 , None

                                MDFillRoundFlatButton:

                                    text: "Sell"

                                    on\_release: app.sell\_stock()

                                    font\_size: 20

                                    pos\_hint: {"center\_x": .5}

                                    md\_bg\_color: 1,0,0,1

                                    size\_hint: 0.45 , 1

                            MDLabel: # stock infomation

                                id: stock\_info

                                text: "Ticker: \n\nCurrent Price: "

            # Profile screen

            Tab:

                icon: "account-box-outline"

                BoxLayout:

                    orientation: "vertical"

                    MDToolbar:

                        left\_action\_items: [['refresh' , lambda x: app.add\_item\_portfolio()]]

                        id: profileLabel

                        title: "profile"

                        anchor\_title: "center"

                    BoxLayout:

                        orientation: "horizontal"

                        size\_hint\_y: 0.07

                        OneLineListItem:

                            text: "Ticker"

                        OneLineListItem:

                            text: "Units"

                        OneLineListItem:

                            text: "Value"

                        OneLineListItem:

                            text: "Total Value"

                    RecycleView:

                        id: rv\_profile

                        viewclass: "PortfolioWidget"

                        data: app.profile\_data

                        RecycleBoxLayout:

                            orientation: "vertical"

                            default\_size: [0, dp(50)]

                            default\_size\_hint: 1, None

                            size\_hint\_x: 1

                            size\_hint\_y: None

                            height: self.minimum\_height

            # Settings screen

            Tab:

                icon: "cog"

                GridLayout:

                    cols: 1

                    MDList:

                        OneLineListItem:

                            text: "Sign out"

                            on\_release: app.sign\_out()

<LoginScreen>:

    BoxLayout:

        orientation: "vertical"

        MDToolbar:

        BoxLayout:

            orientation: "vertical"

            padding: 20

            spacing: 30

            MDLabel:

                text: 'LOGIN'

                font\_style: 'Button'

                font\_size: 45

                halign: "center"

            MDTextField:

                id: username

                hint\_text: "username"

                icon\_right: "account"

                helper\_text: "Invalid username"

                helper\_text\_mode: "on\_error"

                size\_hint\_x: None

                width: 220

                font\_size: 20

                pos\_hint: {"center\_x":.5}

                color\_active: [1,1,1,1]

            MDTextField:

                id: password

                hint\_text: "password"

                icon\_right: "eye-off"

                helper\_text: "Incorrect password"

                helper\_text\_mode: "on\_error"

                size\_hint\_x: None

                width: 220

                font\_size: 20

                pos\_hint: {"center\_x":.5}

                color\_active: [1,1,1,1]

                password: True

            MDFillRoundFlatButton:

                text: 'LOGIN'

                pos\_hint: {"center\_x":.5}

                font\_size: 15

                on\_release:

                    root.login()

            MDFlatButton:

                text: 'SIGN-UP'

                pos\_hint: {"center\_x":.5}

                font\_size: 15

                on\_release:

                    root.signUp()

            Widget:

<SignUpScreen>:

    BoxLayout:

        orientation: "vertical"

        MDToolbar:

            left\_action\_items: [['arrow-left' , lambda x: root.back\_to\_login()]]

        BoxLayout:

            orientation: "vertical"

            padding: 20

            spacing: 30

            MDLabel:

                text: 'Create \nAccount'

                font\_style: 'Button'

                font\_size: 45

                halign: "center"

            MDTextField:

                id: username

                hint\_text: "username"

                icon\_right: "account"

                helper\_text: "username is taken"

                size\_hint\_x: None

                width: 220

                font\_size: 20

                pos\_hint: {"center\_x":.5}

                color\_active: [1,1,1,1]

            MDTextField:

                id: password

                hint\_text: "password"

                icon\_right: "eye-off"

                size\_hint\_x: None

                width: 220

                font\_size: 20

                pos\_hint: {"center\_x":.5}

                color\_active: [1,1,1,1]

                password: True

            MDTextField:

                id: REpassword

                hint\_text: "Re-enter password"

                icon\_right: "eye-off"

                helper\_text: "password does not match"

                size\_hint\_x: None

                width: 220

                font\_size: 20

                pos\_hint: {"center\_x":.5}

                color\_active: [1,1,1,1]

                password: True

            MDFillRoundFlatButton:

                text: 'SIGN-UP'

                pos\_hint: {"center\_x":.5}

                font\_size: 15

                on\_release:

                    root.signUp()

            Widget:

# Evaluation

## Testing

### Regression testing

To ensure the program has not been negatively affected by any changes over my different versions I will retest the program using the same tests I proposed in version 1 and 2 for the development section. Some tests from version 1 have been left out as they do not relate to the final version of the code.

Version 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Original result | Final result |
| 1.1 | Is the users balance correctly outputted | The user selects the view balance option and the balance is displayed to the user. | When the user navigates to the portfolio screen the balance can be see at the top of the screen above the user stock positions. |
| 1.2 | Does the current market price get outputted | The user selects the data option and enters a ticker, the market price is then displayed. | When a ticker button is pressed the current market price is displayed in the bottom right next to the buy and sell buttons. |
| 1.4 | Are all the user’s stock positions displayed | The user selects the portfolio option and all their stock positions are displayed. | When the user navigates to the portfolio screen and clicks the refresh button their portfolio is displayed. |
| 1.5 | Is everything verified when buying or selling a stock | All validation was successful and the user is asked to confirm before the transaction is completed. | All validation was successful and the user is asked to confirm before the transaction is completed. |
| 1.6 | Is the balance changed when a stock is bought or sold | The user balance was updated with the new amount and the csv file was also updated. | The user balance is calculated and then updated in the database. |

Version 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test | Original result | Final result |
| 2.1 | Does the graph display data for the correct number of days | The graph is displayed with the correct number of days of data on it. | When a graph is created the user has the option to switch between data for the week, month, or year. |
| 2.2 | Does the portfolio only display the logged in users’ positions | The portfolio was correctly displayed. | The portfolio was correctly displayed. |
| 2.3 | Is the database correctly updated when buying or selling a stock | The database is correctly updated. | The database is correctly updated. |
| 2.4 | Does the user’s login get verified correctly | The user’s login is successfully validated. | The user’s login is successfully validated. |
| 2.5 | When signing up is the username checked to be unique | The program does notice if you enter a username that is already in user and it informs you of the error before halting the process. | The program does notice if you enter a username that is already in user and it informs you of the error before halting the process. |
| 2.6 | When signing up is the password validated | The password is not accepted if you enter a null value so the validation is successful. | The password is not accepted if you enter a null value so the validation is successful. |

Summary

Most tests had the same result in the final version, with only some of them being slightly different because of the GUI. As nothing has been broken or damaged throughout development, I can conclude that the final product is successful.

### Testing against success criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Criteria | Explanation | Justification |
| 1 | 1.1 | Login – username and password | Is the user asked to enter a username and password upon launching the program? | **Criteria met** |
| 2 | 1.2 | Login – database verification | Once the user has entered a username and password, are these compared to the database to see if they are valid inputs? | **Criteria met** |
| 3 | 1.3 | Login – completed login | Once a login has been successfully verified, is the user sent to the main menu?  Is the user’s data loaded into the program from the database? | **Criteria met** |
| 4 | 2.1 | Signup – username, password, and re-password | Is the user able to input a username, password, and Re-Password? | **Criteria met** |
| 5 | 2.2 | Signup – database verification | Once the user has entered a username, password, and re-password are these successfully validated? | **Criteria met** |
| 6 | 2.3 | Signup – completed login | If valid, are the entered username and password stored in the database?  Is the user sent back to the login screen? | **Criteria met** |
| 7 | 3.1 | Ticker buttons – displaying the buttons | Are the tickers stored in the database taken and displayed as buttons within the program? | **Criteria met** |
| 8 | 3.2 | Ticker buttons – button functionality | When a button is pressed is a graph, and stock information displayed. | **Criteria met** |
| 9 | 3.3 | Ticker buttons – search field | Is there a search field so that users can easily find the ticker they want? | **Criteria met** |
| 10 | 3.4 | Ticker buttons – suggest button | Is there a button that allows users to add tickers to the database?  Once successfully adding a ticker, is it displayed in the program? | **Criteria met** |
| 11 | 4.1 | Graph – the graph | When a ticker button is pressed, is a graph displayed plotting the stock’s price against time?  Is the default time range a month? | **Criteria met** |
| 12 | 4.2 | Graph – time buttons | Are there buttons for the user to change the time range of the graph? | **Criteria met** |
| 13 | 5.1 | Buying and selling - buttons | Is there a buy button and a sell button?  Is this buy button green and the sell button red? | **Criteria met** |
| 14 | 5.2 | Buying and selling – amount input | Is there a text input field for the amount? | **Criteria met** |
| 15 | 5.3 | Buying and selling – verification | Does the program know what the current ticker is?  For selling, does the user own the ticker and do they have enough to sell?  For buying, does the user have enough to afford the amount they have entered?  Is the user ask to approve the transaction? | **Criteria met** |
| 16 | 5.4 | Buying and selling - database | Does the user entry table get updated for the user’s new portfolio?  Does the users table get updated for the users’ new balance? | **Criteria met** |
| 17 | 6.1 | Portfolio - balance | Is the users balance retrieved from the database and displayed to the user? | **Criteria met** |
| 18 | 6.2 | Portfolio – stock positions | For each stock the user owns is the ticker, units owned, current price, and total value displayed? | **Criteria met** |

**Test 1**

Graphical user interface, text, application, chat or text message

Description automatically generated

When the program first launches the user is presented with the login screen shown to the left. Here they can enter a username and password then click the login button.

**Test 2**

The program uses the username as the criteria in an SQL statement to retrieve the password. Then the entered password is hashed and compared to the password retrieved, if they are the same the validation is successful. In my testing the program always worked correctly.

**Test 3**

After a successful login the current screen is set to the trading screen and the users balance and portfolio are stored in their respective locations.

**Test 4**

Upon clicking the signup button on the login screen the signup screen is displayed as shown to the left. There are text fields for a username, password, and re-password as well as a signup button.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Test 5**

The validation for creating an account is simple as all it requires is a unique username and valid inputs for all fields. In my testing the username is checked to be unique and any non-string inputs are rejected.

**Test 6**

After creating an account, I can see in the database that the new account is added with the same username and hashed password entered by the user.

**Test 7**

Table

Description automatically generated with medium confidence

To the left is an image of the ticker buttons displayed in order from the database.

**Test 8**

Graphical user interface, text, application

Description automatically generated with medium confidenceChart, line chart

Description automatically generated

After clicking the GOOG button the graph and current price are displayed as seen here.

**Test 9**

Above the ticker buttons there is a search bar that in my testing has worked every time.

A picture containing application

Description automatically generated

**Test 10**

A picture containing treemap chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The suggest button is displayed bellow the ticker buttons and upon being invoked displayed the popup as seen above.

**Test 11**

After clicking the GOOG ticker button the graph to left is displayed, and as you can see the default time range is a month.

Chart, line chart

Description automatically generated

**Test 12**

To the right of the graph three time buttons are displayed, the buttons are for changing the time frame of the graph between a week, month, and year.

**Test 13 / 14**

Graphical user interface, application

Description automatically generated

To the left is the buy and sell buttons as well as the amount text field. These are displayed upon navigation to the trading screen.

**Test 15**

When the user selects a ticker and clicks either buy or sell the confirmation popup shown below is displayed.

Graphical user interface, text

Description automatically generated

If the user is happy with the proposed trade, then the buy or sell button is pressed and the transaction is processed.

**Test 16**

Upon a trade being confirmed the user entry table is successfully updated with the new information. I can verify this simply by looking in the database after a transaction has taken place.

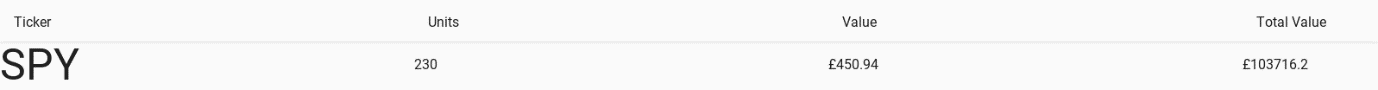
**Test 17**

Text

Description automatically generated

When a user goes to the portfolio screen their balance is displayed at the top.

**Test 18**



When entering the portfolio screen all the user’s current stock positions are displayed in the form shown above. For each record the ticker name, units held, individual value, and total value is shown.

### Testing for robustness

This will involve testing the program to see if it is possible to break it or cause a crash. All validation has already been tested so will not be testing again here.

|  |  |  |
| --- | --- | --- |
| Test | Explanation | Result |
| 1. Open lots and lots of graphs | While using my program I discovered an error message that said there were too many matplotlib figures. In this test I will see what happens if I keep making more and more graphs. | As more graphs are opened the program uses more and more memory. This is a problem but can be easily fixed simply by closing the old graphs when new ones are created. |
| 2. Clicking ticker buttons while one is loading | When a ticker button is pressed the program must do many things. The idea of this test is to see what happens if I click a different ticker before the first has finished loading. | When a ticker button is pressed the program is frozen while the instructions are done, this bug actual means that no other ticker buttons can be pressed. |

Test 1 – evidence

The error message that alerted me to this issue is as follows:

RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory.

After some testing I have found that each new graph takes up around 10MB of memory. Below is an image from windows task manager showing the programs hardware usage with memory being in the second column.



As you can see upon first launch the program uses 130MB of memory, but after clicking on 10 tickers and creating 30 graphs the usage jumps to 450MB as shown below.



The fix

To fix this issue is very simple as matplotlib has a method for closing figures. In the create graph method of my program I will just need to add the following line of code to close the current graphs before creating new ones.



### Testing for usability

To make sure that my program is in an acceptable state for my stakeholders I will test for its usability. I will do this by giving my program to users to test and then make suggestions.

User Feedback

|  |  |
| --- | --- |
| What did they like? | * The user interface looked good. * The program was simple and easy to use. * While simple, the program still has a lot of usability features. |
| What did they dislike? | * The program freezes when processing the instructions from a ticker button being pressed. * There is no validation on the password when creating an account. * Outside of the trading screen there is a limited number of features. |

Overall, the user feedback suggests that the program was simple and easy to use but lacked features beyond the essentials. The things that they did not like were all a result of the time limit of this project. The password validation and user experience features could have been added if more time was allowed for development. The program freezing when a ticker button is pressed is an issue I was aware of prior to user testing but was unable to think of a solution to until now.

## Usability Features

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generated

The login and signup screens are very standard and easy to use, and user feedback has confirmed this. As they only contain text fields and labelled buttons the user is easily able to work out what they need to do to operate the program.

Graphical user interface, chart

Description automatically generated

The trading screen is the most complex component of the program, from user the user feedback I have found that the buying and selling section is easy to use, but the rest is not obvious at first glance.

Graphical user interface, application, table

Description automatically generated

The portfolio screen is very simple and straight forward, the user balance is displayed at the top and then each of the user’s stock positions are displayed. User feedback has confirmed that the portfolio screen is easy to user, but they also said that it was a bit limited on the features available.

Overall, the program is quite simple to use with only the trading screen requiring a little bit of explanation to use. In future versions I would include a brieve explanation of how to operate the program upon the user first opening the program.

## Maintenance

My program functions through a main python file, kivy gui file, and a database. It also makes use of quite a few external modules. The main issues with maintenance would be the database and API.

As the database contains all the required data for the program over time the size of the database will get bigger and bigger. As all data is stored locally it is unlikely to get absurdly big but to prevent any possible space issues, I could implement a feature of limiting the maximum number of tickers or account.

The API is probably the most likely thing to cause issues in the future as it is entirely dependent on yahoo finance keeping their API operational. As with other APIs sometimes the company who operate them close them down. If this happened my program would no longer be functional and I would have to migrate to a different API, of which there are many.

The external modules I use work currently but if the modules are updated it could cause issues for future users, to avoid this I would ship the final product with the external modules already installed and contained within the files, this would ensure the program always works.

Overall, I would say that the program is maintainable, the only exception being the API which if taken down would require a fair amount of work to change over to a new API.

## Limitations

There are a few components of my program that either have limited amounts of features or features with problems I wasn’t able to fix, I have listed some examples bellow.

|  |  |
| --- | --- |
| Freezing when a ticker button is pressed | At the moment when a new ticker is selected using the buttons on the left side of the screen the program freezes. This is caused by the program waiting for the API request to complete as well as the graphs to be created. A possible solution for this problem would be to make use of multiple threads of the CPU. This can be done using pythons threading module. Using multiple threads allows the program to complete the API and graphing instructions simultaneously with the kivy GUI instructions. |
| Account creation validation | In my final version I still did not include any validation for the password when creating an account. I did this as I wanted to focus on the core functions of the program, but now that those parts are completed, I would like to implement some password validation when creating an account. This could be something simple such as the password must contain a capital or a more security focused approach. |
| Lack of portfolio features | From the user feedback I have received it became obvious that the portfolio is lacking in features. At the moment all you can do is see your balance and stock positions; in futures versions I would like to add the feature to see transaction logs as well as more informational data about the user’s stock positions. These features will be discussed more in the future development section. |
| Lack of settings features | As with the portfolio screen user feedback suggests that the settings are very limited, this is quite clear considering there is currently only a single option in it, the sign out button. |

## Future Development

The final version of my program meets all the success criteria I proposed in the analysis section, but there are some features that I have thought of since then and some that stakeholders have suggested while testing my program. Most of these are listed in the limitations above but will go over them in more detail now.

Portfolio additions

To improve the portfolio screen in my program I had some ideas to give the user more information about their stock positions. The features I want to add are transaction logs and percentage change indicators. The transaction logs will help users to know how much they purchased a stock for and when, and the percentage indicators will inform the user if they make money by selling. These features give the user far greater knowledge of their portfolio and will help them to make better trading decisions.

Settings additions

Currently the settings screen consists of a single option, the sign out option. There are many possible things to include in the settings such as account alteration, and colour themes. Being able to change the colour theme of the program will improve the usability as colour blind or anyone with sight programs will be able to adjust the colours to their needs. From my user feedback I have also been told that some settings for deleting or changing account details would be beneficial. This will give the user more control if they decide to change and of their account details, it will also improve maintainability of the program as any old accounts can be deleted if out of use, meaning that the database will increase in size slower.

Account creation password validation

Another feature I would like to add is some validation and criteria for a user’s password. At the moment when creating an account, you can use any string of characters as your password, with this change the password will need to meet some criteria for the account to be successfully created, the exact criteria I am not certain of but examples would be things such as capital letters and numbers must be contained within the password.

## Conclusion

My goal for this project was to build a stock market trading simulator than can be used by my stakeholders to improve their trading ability before trying on the real market. I believe that the final version of my project meets this goal. Through the use of regression testing and testing against the success criteria stated in the analysis section I have shown that my program satisfies all the criteria I originally wanted and throughout development existing features have not been damaged or broken by the implementation of new features.

Testing my program for its robustness has shown that besides one fixable issue the program is very resilient and is unlikely to crash or break when being operated by a user. The problem that I did find as I have said is easily fixable and I have covered it in the robustness testing section. Testing for usability has also shown that my users find the program easy to use but lacking in features for some sections such as the portfolio and settings screens.

While considering the maintainability of my program I have found two areas that may cause problems in the future. First the database as over time it will only get bigger and at the moment there is no restrictions to its size, and secondly is the API as at any time Yahoo could decide to stop providing at API and the program would break. These problems are fixable as I could migrate to a new API and in future versions could include a limit the size of the database through the use of things such as an account number maximum or a maximum number of tickers at any one time.

In the future I have plans for more features to include, as stated in the future development section. These are mostly user experience changes such as more features for the portfolio and settings screens, but also some are robustness changes such as implementing some password strength validation.

Overall, I believe that my project has been a success as I have achieved all the success criteria set out in the beginning and from the feedback of my users, I can conclude that my stakeholders would be happy with the final version.