TradeIT: Trading Robot Web Application *

Stefan Court

Abstract

In the realm of financial markets, being able to validate and refine trading strategies is critical for success. A platform has yet to be created that allows paper money to be used to predict historical data in an intraday trading manner with an engaging user interface. To enable this I have created a sophisticated web application that allows users to place trades using virtual funds either manually or through artificial intelligence gaining valuable insights into the effectiveness of strategies without risking any capital. The application uses a JavaScript front-end, Django back-end, the built-in SQLite3 database and a robust algorithmic trading robot to achieve this. This literature report details the design and implementation of the application with reasons behind certain actions, the project specification and an evaluation of how effective the methods used were in delivering the project objectives and aims.

*Stefan Court: *I certify that all material in this dissertation which is not my own work has been identified.*

Contents

1 Introduction

With so many people using the internet, being "5.4 billion" which is around 67 per cent of the global population [1]. Online brokerage platforms such as Trading212 [2] and Etoro [3] have taken over the industry making physical stockbrokers redundant; this is visible through the markets themselves as when the markets were digitalised, volatility increased as more people were able to trade on them as shown in figure 2 of [4]. With this new age of being able to trade on the stock market so easily through a press of a button, it is imperative to provide information on how to trade using certain strategies with good risk management. This is where TradeIt was created, the all-encompassing trading software that allows users to manage their balance, place trades and view relevant news headlines all within one application.

1.1 Motivations

Having previously engaged in trading stocks and shares and experiencing losses in capital, I recognised the necessity for a piece of software where individuals could learn about trading without any financial risk or transaction fees. This realisation underscored the significance of software that simulated trading scenarios without monetary repercussions. Creating an AI component to the website would allow me to gain insight into how technical indicators may influence outcomes. Aspiring to pursue a career in quantitative analysis, I identified the development of a trading robot as a logical extension of my skills in website development.

1.2 Aims and Objectives

The development of this trading robot web application is guided by a set of distinct aims and objectives aimed at modernising ways that users place trades. One of the primary aims of this project is to automate various parts of the trading process including market analysis, placing and exiting trades and balance management, thereby reducing manual effort. Concurrently, the objective is to provide a user-friendly web application that encompasses a comprehensive suite of functionalities for fund and trade management, incorporating the news, balance management and the placing of trades. Moreover, the application aims to reduce the influence of human bias and emotion in the placing of trades effectively managing risk, thus creating more lucrative and informed investment decisions. Ultimately, the goal of the project is to create an application that is profitable over multiple time frames and different types of stock that keeps users engaged and builds up the courage to place trades with real funds on a brokerage account. Pursuing these aims and objectives, the application seeks to redefine the landscape of trading and provide success in navigating historical markets.

2 Summary of Literature Review and Specification

2.1 Requirements

2.1.1 Functional

R1	User can register and login to account
R2	Software can securely authenticate users
R3	Allow users to manage and view balance
R4	Enable users to place buy/sell orders
R5	Provide stop-loss and take-profit options
R6	Allow users to delegate funds to living expenses
R7	Fetch historical data of stock market prices
R8	Fetch real-time news of certain assets
R9	Integrate various trading algorithms into one complete algorithm
R10	Allow users access to multiple types of stock

2.1.2 Non-Functional

NR1	Ensure system can handle concurrent users
NR2	Executing of trades should be seamless
NR3	Encrypt sensitive information to the user
NR4	Ensure compatibility with multiple web browsers
NR5	Design an intuitive user interface that is easy to navigate
NR6	The software must run reliably and efficiently

2.2 Evaluation Criteria

2.2.1 Web Application

To evaluate the success of the software of the project, the project will be compared with the mentioned functional and non-functional requirements. A fluid operation encompassing all specified features will indicate the project's efficacy. Feedback on the usability and front-end style of the project will be gathered, containing user ratings of the website and notification of any changes that should be made or new features that should be implemented. Additionally, testing protocols, including unit and boundary testing, will be applied to assess the reliability of back-end functionalities. These tests will scrutinise the interaction with Django models and certain functions, ensuring robustness and integrity within the system's core components.

2.2.2 AI Robot

To assess the effectiveness of the machine learning component within the project, different testing procedures will be implemented. The trading robot will undergo evaluation across multiple stocks, utilising varied start dates to simulate diverse market conditions. Performance metrics will be found by comparing the robot's trading outcomes against the actual performance of the selected stocks. Success will be measured by the degree of profitability achieved, with a positive correlation indicating effectiveness, while conversely, losses will denote areas of weakness in the algorithm.

3 Design

3.1 AI Design

There have been many iterations of the design of the AI initially starting with some strategies such as the moving average and continuing with them and some such as arbitrage not being suitable for the model. Some new algorithms that are used as indicators are the ADX and RSI which are subsets of momentum indicators. The reason why VWAP is not used in the application is due to VWAP mainly being used for intraday trades, as the volume that is used in the formula is reset after the day is over, this makes the use of the algorithm redundant in TradeIt as the app looks at each day as a ticker. Arbitrage was not used as it utilises price discrepancies in different markets. As Yahoo Finance [5] is the only market API data covered and has only daily data, there is no use for this trading strategy.

3.1.1 Moving Average

One of the algorithms used to predict the market is the moving average which can be used in many ways to predict the market. In one paper it is shown that the Moving Average works on predicting the S&P500 [6] which gives a good incentive to the possibility that this could work for some big corporations. In this instance, the moving average crossover strategy will be used placing buy and sell orders when the short and long moving averages cross. Initially, a simple moving average was implemented to predict the market, but on further analysis, an exponential moving average seemed more viable. This is where the more recent trades have a higher weighting in their respective windows, meaning that the moving average can change quicker than SMA. This enables the program to produce trades faster and predict the market shifting in a certain direction faster than the simple alternative. The mathematical formula is as follows:

For the Simple Moving Average (SMA):

$$\text{SMA}(t) = \left(\text{Close}(t) + \frac{\text{Close}(t-1) + \text{Close}(t-2) + \ldots + \text{Close}(t-N+1)}{N}\right)$$

Where:

$\text{SMA}(t)$ is the simple moving average at time t
$\text{Close}(t)$ represents the closing price at time t
N is the size of the rolling window

For the Exponentially Weighted Moving Average (EWMA) [7]:

$$\text{EMA}(t) = (1 - a) \times \text{EMA}(t - 1) + a \times \text{Close}(t)$$

Where:

$\text{EMA}(t)$ is the exponentially weighted moving average at time t
$\text{Close}(t)$ is the closing price at time t
a is the smoothing factor calculated as $a = \dfrac{2}{\text{span} + 1}$

When 'adjust=False':

$$\text{EMA}(t) = (1 - a)^t \times \text{Close}(0) + (1 - (1 - a)^t) \times \text{Close}(1) + (1 - (1 - a)^{t-1}) \times \text{Close}(2)$$

3.1.2 ADX

Another algorithm used to predict the market is the ADX indicator. This is used to determine when the price is trending strongly in a certain direction. It accomplishes this with a certain threshold value and window of dates. If above a certain threshold, and the two directional movement indicators crossover, this notifies the algorithm there is a strong trend. For example, a buy order should be placed if there is a strong trend upwards. The mathematical formula is as follows [8]:

$$\pm\text{DI} = \left(\frac{\text{Smoothed} \pm \text{DI}}{\text{ATR}} \right) \times 100$$

$$\text{DX} = \frac{|+\text{DI} - \text{-DI}|}{|+\text{DI} + \text{-DI}|} \times 100$$

$$\text{ADX} = \frac{(\text{Prior ADX} \times (\text{ADX Window} - 1)) + \text{Current ADX}}{\text{ADX Window}}$$

Where:

ADX(t) is the ADX value in the DataFrame at time t

ADX Window(x) is the amount of previous trades checked for x

+DM (Directional Movement) = Current High − Previous High

-DM (Directional Movement) = Previous Low − Current Low

$$\text{Smoothed } \pm\text{DM} = \sum_{t=1}^{\text{ADX Window}} \text{DM} - \left(\frac{1}{\text{ADX Window}} \sum_{t=1}^{\text{ADX Window}} \text{DM}\right) + \text{CDM}$$

CDM = Current DM

ATR = Average True Range

3.1.3 RSI

The final indicator used to predict the market is the RSI indicator. This is where the market is scanned for overbought and oversold conditions. It does this by calculating the average gains and losses over a specified period and generates a value between 1 and 100. The higher the value the more bought the stock is, with a neutral value being 50. The mathematical formula is as follows [9]:

Step One:

$$RSI_1 = 100 - \left[\frac{100}{1 + \frac{\text{Average Loss}}{\text{Average Gain}}}\right]$$

Step Two:

$$RSI_2 = 100 - \left[\frac{100}{1 + \frac{((\text{Previous Average Loss} \times (\text{RSI window} - 1)) + \text{Current Loss})}{((\text{Previous Average Gain} \times (\text{RSI window} - 1)) + \text{Current Gain})}}\right]$$

Where:

RSI is the RSI value

RSI window (x) is the amount of previous trades checked x

Average Loss is the sum of losses over the RSI_window Average Loss $= \frac{\text{price decline}}{x}$

3.2 Web Application Design

3.2.1 Database

The database is a repository of user information essential for user creation and login procedures. This contains a unique *user_id* crucial for managing the user's balance and trades. Furthermore, the database holds information regarding the type of stock involved in each user's trades, within this same table, the total profit and loss for each user is stored.

3.2.2 General Pages

All pages within the program extend the 'Base.html' file, which serves as the foundation for a consistent layout throughout all pages. This template contains essential elements such as the navigation bar and sidebar components throughout all the pages. This navigation bar features the page title, a convenient logout button, displays the funds in balance and a menu button. Upon activation of the menu button, the sidebar expands presenting users with links to all available pages, a back button to exit the sidebar and a toggle switch for activating dark mode. If the back button is pressed the sidebar will disappear and the page will return to its original state. Clicking on any of the links within the sidebar effortlessly redirects the user to the page wanted. For all pages, if the dark mode is toggled on, then the dark mode will be activated for the page where styling for each element will be different, the dark mode is then retained when navigating away from the page by saving the toggle to the local storage of the device using JavaScript.

When dark mode is disabled, all pages feature a uniform light blue colour scheme, complemented by green and red accents indicating positive and negative actions respectively. However, upon toggling dark mode, the pages change to a dark blue colour scheme using light blue accents within the page to reduce eye strain for the user.

Each page has been coded to be accessible for phones, tablets and computers using CSS to determine the width of the page. if this width is above a certain threshold, the page will use different styles for certain elements.

3.2.3 Graphs

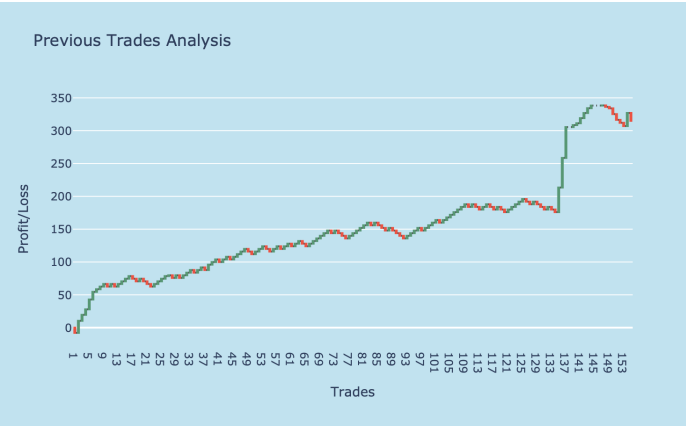
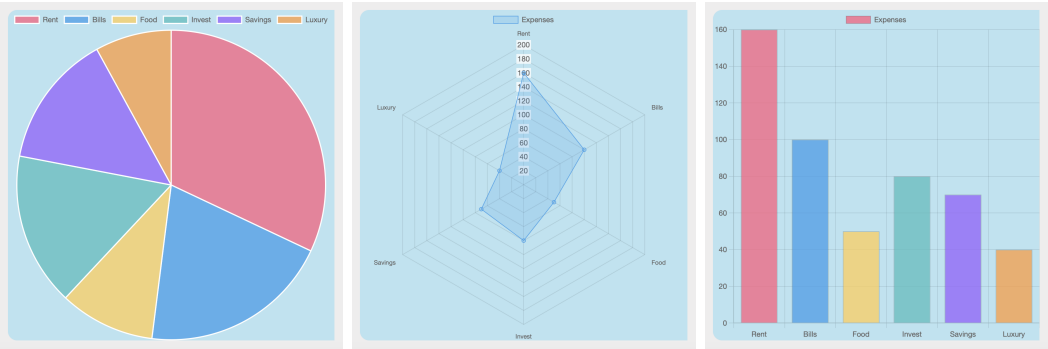


Figure 1: User portfolio of Apple Stock

The statistics page utilises a cascade chart to illustrate how trades influence the account balance of the user, with positive trades corresponding to a green bar which increases on the y-axis and negative trades corresponding to a red bar which decreases on the y-axis. This is updated every time the user either navigates to the statistics page or submits a form containing information on the stock type they would like to view in their portfolio. In both the `'trade'` and `'ai-trade'` pages, Chart.js [10] is employed to present a dynamic graph refreshed every second with historical dataset information, including the date and close price. The graph showcases light blue dots and lines,

complementing the page's colour of both light and dark modes. Notably, no animations are present in adding points to the graph to prevent all lines from starting at zero on the scale and increasing to the data's closing value for the day. JavaScript is used on the pause and play buttons of these graphs, allowing users to stop and continue the graph's autonomous actions at any point. This proves to be very useful on the trade page as users can pause the graph before placing a buy/sell order adding precision to each trade.

Within the 'finance' page, Chart.js [10] is deployed to create a graph based on user input, featuring an array of vibrant colours. This design ensures users can easily discern different sections of the chart representing various assets. Furthermore, when hovering over each section of the chart, the amount inputted, and section details are displayed, preventing any potential confusion. There are many graph types, ranging from bar charts, to polar area charts, each adaptable to the discrete data inputted. This thereby enables user's flexibility in managing their finances according to their preference.



(a) Pie Chart (b) Radar Chart (c) Bar Chart

Figure 2: Charts

4 Development and Feature Implementation

4.1 AI Implementation

To implement this AI into the ai-trade app a `'signal.csv'` file is written to the signals folder which is named after the specific robot's id containing the signals for the algorithm specified.

4.1.1 Moving Average

The moving average signal is generated if there is an upcoming forecast of a change in trend. This is observed when the short window moving average crosses the long moving average which is called a moving average crossover. The buy and sell signals are not generated

frequently as the moving averages do not cross frequently especially if the windows for both moving averages are long. The signal is determined:

$$\text{MA\_Signal} = \begin{cases} -1.0 & \text{if short\_ma} < \text{long\_ma \& previous short\_ma} > \text{previous long\_ma} \\ 1.0 & \text{if short\_ma} > \text{long\_ma \& previous short\_ma} < \text{previous long\_ma} \\ 0.0 & \text{otherwise} \end{cases}$$

Where:

short_ma is the short window for the moving average
long_ma is the long window for the moving average

4.1.2 Average Direction Index

The ADX signal is produced only when very niche conditions occur being, when the directional indicators cross and if the ADX value is above that of the threshold signifying that there is a strong enough trend to suggest that a trade should be placed. This like the moving average indicator mentioned before, produces signals quite rarely as both conditions have to be met where the trend is changing and the strength of this trend is strong. The signal is determined:

$$\text{ADX\_Signal} = \begin{cases} -1.0 & \text{if -DI} > \text{+DI \& previous +DI} > \text{previous -DI} \\ & \quad \text{\& ADX} > \text{ADX\_threshold} \\ 1.0 & \text{if -DI} < \text{+DI \& previous +DI} < \text{previous -DI} \\ & \quad \text{\& ADX} > \text{ADX\_threshold} \\ 0.0 & \text{otherwise} \end{cases}$$

Where:

DI is the Directional Indicator
ADX Is the Average Directional Index
ADX_threshold Is the Average Directional Index threshold

4.1.3 Relative Strength Index

The RSI signal is generated if the RSI is above the overbought threshold or under the oversold threshold. This is due to the RSI being a momentum indicator that states that if a commodity is overbought it should sell eventually. This indicator produces a lot of signals as there is no specific condition for when the buy/sell signals should be generated.

This is very useful when combining indicators in the combination algorithm. The signal is calculated:

$$\text{RSI\_Signal} = \begin{cases} -1.0 & \text{if RSI} > \text{overbought\_threshold} \\ 1.0 & \text{if RSI} < \text{oversold\_threshold} \\ 0.0 & \text{otherwise} \end{cases}$$

Where:

RSI is the RSI value

overbought_threshold is the threshold for identifying overbought conditions.

oversold_threshold be the threshold for identifying oversold conditions.

4.1.4 Combination

For combining the algorithms, as there are three algorithms and all were useful in different situations, a signal was generated for each algorithm and the more positive/negative the sum of the signals was the more likely the trade would be profitable In most scenarios this is where the ADX or Moving average produce a signal that also correlates to the RSI signal generated. Where in mathematical terms:

$$\text{Combination\_Signal} = \begin{cases} -1.0 & \text{if } \sum signals <= \text{-2.0} \\ 1.0 & \text{if } \sum signals >= 2.0 \\ 0.0 & \text{otherwise} \end{cases}$$

4.2 Web Application implementation

4.2.1 Database

When changing models and intending to retain those modifications, it's necessary to run two crucial scripts *'python manage.py makemigrations'* and *'python manage.py migrate'* [11] These commands serve the purpose of preserving the alterations made to fields within the models and integrating any new models that were previously absent.

To download each of the stocks to a CSV file Django's command feature must be used [12], *'python manage.py api'* must be executed which downloads all data of selected stocks to a CSV file. Subsequently, to transfer this CSV to a Django model for downloading the script *'python manage.py load_stock'* this loads all used information including details such as the date and close price of the stock to Django.

The database where all objects are stored is *SQLite3* [13]. When users are created via registration a Django form named $'RegisterUserForm'$ is employed to authenticate users generating a hashed password that serves as their access token or the website. Upon successful submission and validation of this form, a $'UserProfile'$ object is instantaneously

created. This object establishes a linkage between the user's account and their available funds denoted as *'money_in_account'* via a *'user_id'* identifier.

Upon exiting a trade an entry is appended to the model that includes the type of stock, profit/loss of the trade, new balance and the *user_id* which is required to filter through the stats page and see all trades related to the user that is sent from the view to the webpage.

4.2.2 Folder structure

The server-side aspect of the application, which is Django-based, necessitates the presence of apps to run the project. The root directory of the project is the folder, *tradingsite*. Within this folder, you must execute *'python manage.py runserver'* to initiate server operation. For greater organisation and adaptability, most pages have a separate app delegated to each [14]. The database associated with Django *SQLite3* is located in the *tradingsite* folder where all the apps can seamlessly interact with the database. Within each app's directory, you can find the corresponding HTML and CSS within their respective static and template folders, along with a JavaScript file if the page requires it.

4.2.3 Back-end (Django)

Each app is accompanied by its respective view, responsible for rendering the page for user interaction. Within these views, a validation is performed to confirm the user's authentication status. If authenticated, the requested page is displayed; otherwise, users are redirected to the login page, with an option to navigate to the sign-up page. Across all views, both forms and models are employed. Models are used to view user-specific data, while forms are used when the program requires input from the user to execute a certain task. in the trading pages, when a POST request is made to the server, a JsonResponse is transmitted, containing information including the stop loss and take profit of the trade where it is handled by the consumers file.

```
def view(request)
    if request.user.is_authenticated:
        if request.method == "POST":
            context={#POST info}
            return JsonResponse({#Info to be sent to JS})
        else:
            context={#Info for rendering the page}
        return render(request, "page.html", context=context)
    else:
        return redirect('/')
```

Both the `'trade'` and `'ai-trade'` pages feature a consumers file leveraging Django channels [15] to establish connections with the WebSocket. Within these files, the trading strategies are shown and a GraphConsumer is instantiated to transmit data to the

JavaScript console. Upon the initial pass of the consumer, start date, stock type, and user_id are relayed. in the case of `ai-trade'`, additional information including the bot's unique identifier (uuid) and ai_type are provided. Subsequently, the robot generates files for all signals for each date of the algorithm, being buy, sell or hold. The date and close price of the stock is then sent to the WebSocket where it is handled by the JavaScript file.

On subsequent calls to the consumer, if either the stop loss or take profit is triggered, the user's profit, total profit/loss and balance is sent to the database to be utilised in the views. Regardless of whether either condition is met, the next date for the desired stock and its closing price are relayed to the WebSocket. One challenge encountered was handling cases where the next date was absent within the database. In such scenarios, the consumer must progress to the next available date in the database without transmitting any other data or closing price to the WebSocket. This process is managed through a while loop, where the start date received from the JavaScript file is compared to the subsequent row of the stocks model. If a mismatch occurs, the start value is incremented by one day.

Both the `'news'` and `'trade'` incorporate files *'api.py'* and *'load_stock.py'* respectively within their commands folder. These files should be executed when wanting to update the news or historical data executed by *'python3 manage.py command'*. 'api.py' leverages requests to retrieve data from AlphaVantage [16], a stocks and shares API known for providing news on a wide range of stocks along with their relevance scores. To access this data, an API key is required, which is given upon registration to the website. Data is fetched by retrieving a GET request to the URL with the specified ticker and desired URL parameters. The *'load_stock.py'* file is responsible for downloading data from the Yahoo Finance API [5] for all specified stocks and saving them to their respective CSV files. Each of these stocks' data is then imported into the database by creating an entry for each row in the CSV file.

```
# load_stock.py
class Command(BaseCommand):
    help = 'Load data from stock_data file'
    def handle(self, *args, **kwargs):
        datafile = Path(settings.BASE_DIR) / 'stock_data'
        multi_data = yf.download(["AAPL","...", period="10y")

        for stock_name in multi_data["Open"]:
            yf.download([stock_name], period="10y").to_csv(datafile /
            ↪  (stock_name + '_hist.csv'))

        datafile = settings.BASE_DIR / 'stock_data' / 'AAPL_hist.csv'

        with open(datafile, 'r') as csvfile:
            reader = csv.DictReader(csvfile)

```

```
            for row in reader:
            AAPLStock.objects.get_or_create(date=row['Date'],
            ↪  open=row['Open'], high=row['High'], low=row['Low'],
            ↪  close=row['Close'])
```

To facilitate asynchronous behaviour within the application, a Redis server is employed [17]. This functionality is integral as it enables multiple users to make concurrent requests across various pages without encountering conflicts. Redis achieves this through its single-threading, event-driven architecture and support for atomic operations. The program harnesses this capability by utilising asynchronous operators such as sync_to_async and await within the consumer file's receive function. An issue arose on the ai-trading page, where multiple consumers were accessing the same bot. To resolve this, the implementation of a unique identifier for each WebSocket connection was a necessity.

4.2.4 Front-end

Every page in the application extends 'base.html' which contains the essential sidebar and navigation bar elements along with their styling and contents. Additionally, each page is furnished with its own styling, utilising classes to style HTML elements. Many of these elements feature reusable classes that are employed multiple times throughout the page, such as those altering the font and colour of the text. Moreover, each element is equipped with styling adjustments to seamlessly transition into dark mode, which users can activate by toggling the corresponding option in the sidebar.

Chart.js [10] plays a significant role in the website, being integrated across multiple pages such as `'finance'`, `'trade'` and `'ai-trade'`. It enables the creation of responsive and dynamic graphs that can be updated with ease, either through a simple click of a button or at predetermined intervals. An illustration of its functionality can be seen on the finance page, where triggering a button initiates a form submission to the current URL '/finance/'. This form submission includes information detailing the allocation of the user's balance to various necessities, along with specifications for the type of graph to be displayed. Subsequently, an Ajax request is sent to the server, updating the 'graph' HTML element and presenting the graph on the page. Should the form be submitted again with different parameters, the existing graph is replaced with a new one reflecting the updated information.

```
//Empty chart to add close price and stop loss/take profit
const ctx = document.getElementById('myChart');
var graphData = {
    type: 'line', data: {
      labels: [], datasets: [{
        label: 'Price of stock',
        data: [],
```

```
        borderWidth: 1 }]
    }, options: {}
}
var myChart = new Chart(ctx, graphData);
```

On the `'trade'` page, Chart.js [10] is utilised to create an axis upon the page's opening. Upon pressing the play button and specifying a particular stock and date, the graph is dynamically updated with the next day's information. As the graph accumulates data, it maintains a maximum of 36 entries by deleting the oldest entry and appending the newest one. This process continues indefinitely until the pause button is pressed or no further data is available for charting. Additionally, upon inputting stop loss and take profit values into the form, a corresponding line appears on the graph, which is removed once the defined thresholds are reached.

To facilitate communication between the Django back-end and the JavaScript client, a WebSocket is initiated and used. This mechanism is used in scenarios where the WebSocket receives incoming information from the relative 'consumers.py' file. This data comprises the stock market data for the next day in the list. The WebSocket checks for this data using the 'socket.onmessage' function to assess if any conditions have been met, such as the stop loss being triggered. If such stop loss has been triggered, the WebSocket then sends this information to the corresponding 'consumers.py' file where the event is handled.

```
function createWebSocket() {
    if (socket === null || socket.readyState === WebSocket.CLOSED) {
        socket = new WebSocket('ws://localhost:8000/ws/ai-trade/?id=' +
        ↪  uuid);
        socket.onopen = function (e) {
            //Information to send to consumer on socket open
        };
        socket.onmessage = function (e) {
            //Checks if certain conditions are met
            //Information to send to consumer on socket message
        }
    }
}
```

JavaScript [18] is also employed in generating modal windows for error messages within the application. These modals are triggered when specific criteria are met, necessitating the display of an error message.

5 Testing and User Feedback

5.1 AI Functionality

To test the AI of the program a separate Jupyter [19] folder was created to encompass all testing of the model. This proved useful due to Jupyter's data visualisation capabilities with graphs being able to be drawn in separate cells. The final code used was then copied to the program from the notebook without the testing and graphs to ensure that the code used in the application was the same as the one tested.

To validate the AI model many tests were run with many different argument sizes for each algorithm on multiple stocks. This was to see if there were general parameters that worked for all stocks or only some, being somewhat random. This testing was also done over many sectors, including tech, healthcare and finance.

I initially tested a wide variety of window lengths and thresholds for each algorithm with £1000 used in each trade for the 6 stocks selected i did this by printing the number of trades, win percentage and profit/loss of each of these combinations. By doing this I was able to determine that a high overbought and oversold threshold with a large RSI window was a good combination. For the moving average and ADX algorithms, some combinations were not usable as the number of trades was too low. This is a problem due to not being able to test the efficacy of the algorithm precisely as there were not enough trades taking place to create a strong hypothesis. For the other combinations that were going to be used for the final algorithm a graph needed to be plotted which would show the amount of trades the graph created was a line plot over a bar graph that contains the profit/loss of the combinations.

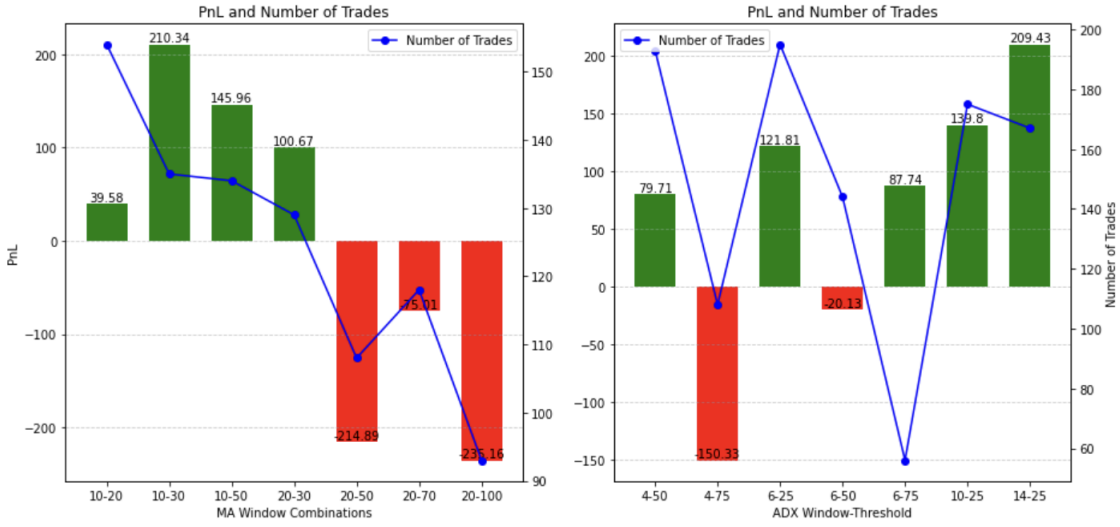


Figure 3: Moving Average and ADX Graphs

From (Figure 3), it is possible to deduce that the longer the windows for the moving average the worse the outcome of the trade as the number of trades decreases and the outcome is a loss for all 6 stocks combined. The ADX algorithm having a high ADX threshold of 75 is not applicable as the amount of trades completed was very low and there is varied data with a low profit and high loss. Baring this in mind the conclusion for the final combinations that have been chosen are as follows:

- Moving Average: short window - 10, long window - 30
- ADX: window - 14, threshold - 25

These were chosen due to the high profit accompanied by a high amount of trades completed. Although the number of trades was 20 higher for the 10-20 window in the moving average algorithm, the profit was around 170 less than the 10-30 window which justified the use of 10-30.

For choosing a stop loss and take profit for each algorithm profit/loss bar charts were implemented to help visualise the different amounts to different take profits. The stop loss was not moved as a value of 0.05 times lower than the close price allowed the algorithm to function having a small window of error but cutting the trade short if the algorithm predicts the trade wrongly.

After viewing these different take profits in (Figure 4), we see initially that the algorithms that have been chosen are indeed profitable and a take profit of 4 times the stop loss, being 0.2 times higher than the stock price increases the profits the most in almost all scenarios. Although for some algorithms such as RSI and Moving Average, the profits increased, having a stop loss 5-8 times higher incurs much more risk when the number of trades is lower and increases the probability of the profits turning into losses. An example of this is visible in the Moving Average graph of Figure 5 where after a take profit of 0.3 times higher than the stock price there is a sharp decline in the amount profited by the trading bot.

This information was used in creating the completed trading robot that incorporated all algorithms into one. the ADX threshold was kept at 25 for all algorithms as this was always the most profitable not changing the number of trades in any case, however, the ADX window was increased the slower the algorithms. The moving average for the quick algorithm was adjusted to have a short window of 5 and a long window of 10, whereas the other two combinations had the original 10-30 split. The RSI threshold was changed in all of the combinations with 'quick', 'medium' and 'slow' having a 70-60, 80-60, 80-50 (overbought-oversold) split respectively

From this graph the quick algorithm was chosen as although the other algorithms were effective with no losses, the amount of trades being greater from the quick algorithm allowed more profit to be generated. This is also beneficial for the users as user retention should be increased due to more trades being carried out .

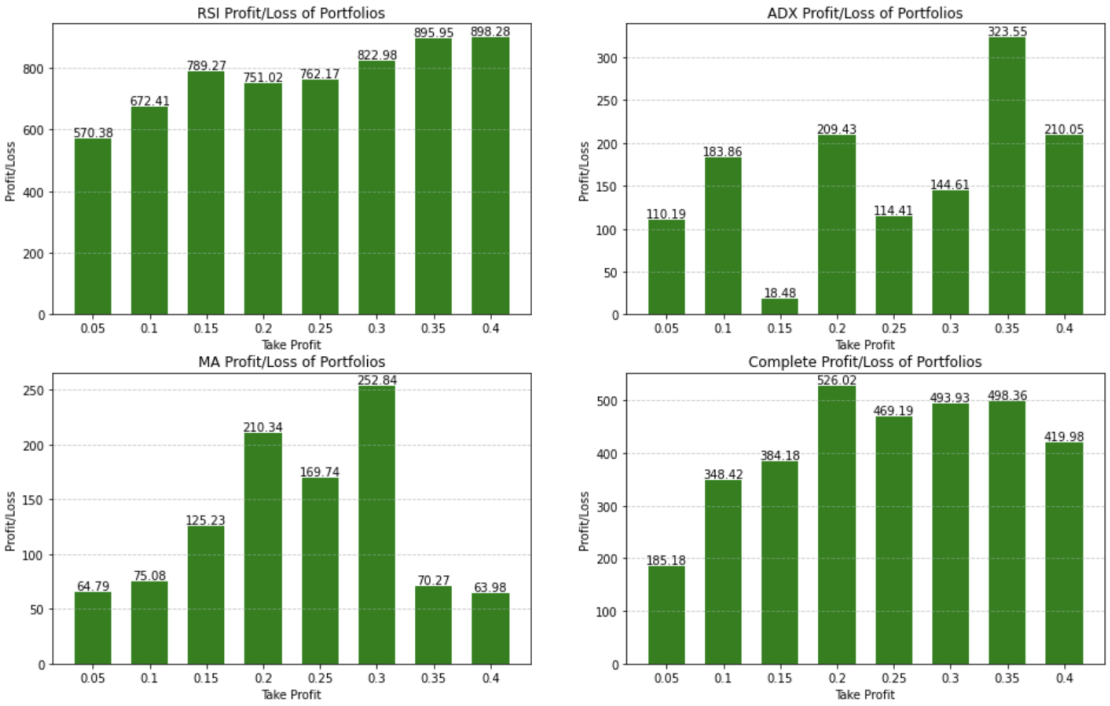


Figure 4: Profit/Loss at different Take Profits

5.2 Unit Testing

In a Django project, unit testing holds paramount importance as it serves to increase the stability and reliability of the application by validating the behaviour of functions and methods as seen throughout [20]. These tests are vital for detecting any bugs or errors present in the code base. They play a significant role in the software development process, ensuring the smooth operation of models and forms within each app. Django simplifies this by generating a 'test.py' file on initiation of the app [21], facilitating comprehensive testing of models, forms and views within each app. It is also of important when testing Django to fully understand how testing the software works so that it can be reverse-engineered if something breaks. "Breaking things on purpose" [22] showcases ways in which you can go about this.

The testing of forms involved inputting data to verify the form's validity and ensure the proper functioning of its widgets and classes. The basics of this were grasped through the webpage [23]. Testing of models included the creation of objects and confirming that the expected value matches the output. Regarding views, validation encompassed checking the authentication status of the viewer, if not authenticated, redirection to the login page should occur. Conversely, authenticated viewers underwent evaluations to ensure correct

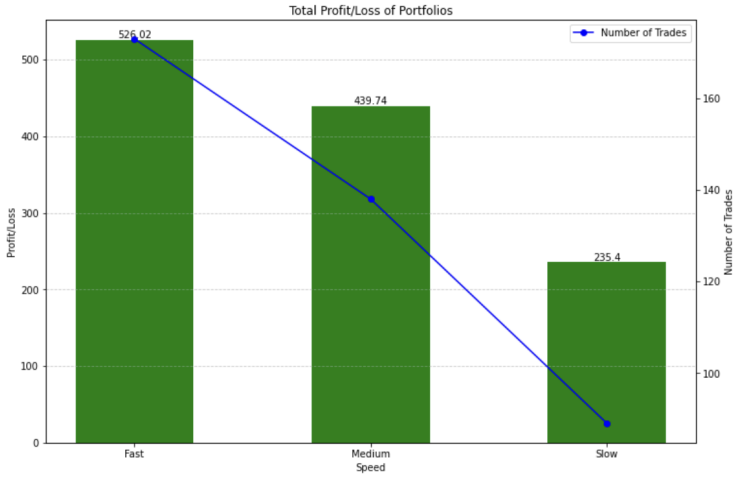


Figure 5: Combination Algorithm Profit/Number-of-Trades

page loading for both get and post requests, with post requests accurately transmitting JSON data. Proper user authentication was tested through the register view, along with logging out the user correctly with the logout view.

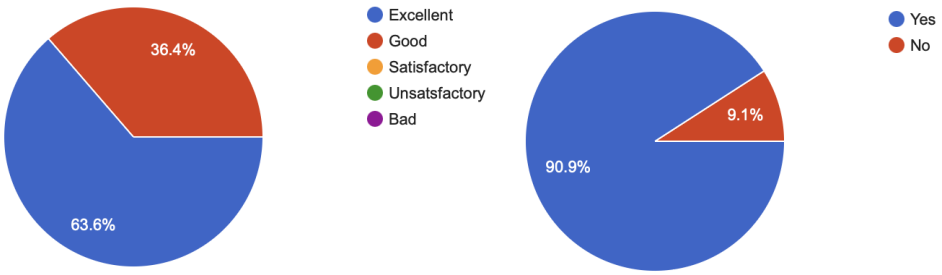
5.3 Boundary Testing

Given that a substantial component of the software relies on the transmission and reception of data via a websocket, boundary testing emerged as a crucial aspect of software development. This emphasis stemmed from the recognition that numerous varied inputs could potentially trigger errors when attempting to process disallowed values. Following extensive software usage, boundary testing became integral to a few key pages, including `'trade'`, `'ai-trade'` and `'finance'` pages. Notably in both `'trade'` and `'ai-trade'` pages, boundary testing focused on date inputs for selecting stock start dates. Should an invalid start date be provided, an error modal should be thrown prompting the user to input a valid start date. For `'finance'` and `'ai-trade'` pages, if the amount inputted exceeds the user's balance, an error should alert users to input a lower amount. Furthermore on the 'trade' page posting a trade requires many variables to be checked, including ensuring that the stop loss does not surpass the take profit on a buy order and questioning if both the stop loss and take profit are higher than the by price at that time.

5.4 User Testing

User testing is an imperative part of the development of the software as ultimately the users' satisfaction determines if the the application was a success. It is used to gauge if users understand how to use the software effectively and efficiently and if this is not the case

the project could be deemed as a failure. The 5 key components used when asking users about the program were, "Learnability, Efficiency, Memorability, Errors and Satisfaction." [24] To participate in the user testing I asked people within the university to participate in using the software and then filling out a Google form [25] that was sent out to them. All participants were under the age of 25 however, ages ranged from 18-25, where 18-30 is the age range that the software has been designed for. In the future bringing more people from various backgrounds into the user testing would be beneficial. (10/11) said they would use the software again as seen in (Figure 6b) with high scores for ratings on all of the pages within the app. People on average thought the User Interface was of high quality with all responses being either excellent or good as seen in (Figure 6a). There were two responses regarding future additions/improvements to the app one stated that there could be more options added when initiating a trade such as being able to draw over the graph and another hinted that more user customisation options would be a good addition by adding a settings page.



(a) User Interface Pie Chart (b) User Satisfaction Pie Chart

Figure 6: Pie Charts

6 Evaluation

6.1 Requirements Evaluation

6.1.1 Functional Requirements

To satisfy R1 and R2 users can log in, log out and register using Django's User forms where they are authenticated. This authentication is checked on every page's view through the *'request.user.is_authenticated'* function. The stats page satisfies R3 with users being able to view their overall balance and check their profit/loss for different stocks. R4 is a huge component and the main part of the application with users being able to place buy/sell orders on both `trade` and `ai-trade` pages. R5 is accomplished within the coding of these pages allowing users to place their own stop loss and take profit, along with having the robot place this for them. R6 directly correlates to the `finance` page where users can manage

their expenses in their balance visualising this with different types of graphs. both R7 and R8 are accomplished through Django commands where *'load_stock.py'* and *'api.py'* satisfy the requirements respectively. The complete algorithm shown in Section 5.1 allows users to place trades using one complete algorithm (R9). The coding throughout the application allows users to apply all these features to all different types of stock available, satisfying R10.

6.1.2 Non-Functional Requirements

NR1 was satisfied when implementing the Redis server to allow multiple users to use the app at the same time. NR2, NR5 and NR6 were achieved through Section 5.4 where most of the tested users' opinions were positive on an overall scale of the project. Unfortunately, NR3 and NR4 were not achieved as there was no information encrypted on the users' information and although the software ran on Safari, other browsers were not tested for compatibility.

6.2 Comparisons to similar apps

6.2.1 Alpaca

In comparison to Alpaca [26], a platform that enables users to engage in a simulated trading environment using algorithms to forecast the market. My platform distinguishes itself by offering additional functionalities such as a comprehensive financial management tool and real-time access to stock-related news. However, Alpaca boasts a unique feature, which is the ability for users to trade using custom-user-built AI algorithms. This feature allows users to experiment with algorithmic strategies without risking actual capital, as virtual funds are utilised. Integrating a similar feature into my platform would be a good future development feature.

6.2.2 Brokerage Apps

In comparing the application with established brokerage platforms such as Trading212[2] and Etoro [3], several similarities and differences emerge. One commonality is the allowing of placing either buy or sell orders with a stop loss in place. However, while these brokerage platforms allow paper money to be used, it can be added and subtracted to any account whenever wanted which leads to misuse and a lack of trade accountability. Despite shared functionalities, significant differences do exist between my application and online brokerage platforms. Primarily brokerage platforms facilitate trading with real money, a capability that is absent in my application. Incorporating such functionality would require the opening of a business where certain laws must be abode and ethical concerns may become an issue. Additionally, while trading bots are often absent in brokerage platforms the reason as to why this could be is due to the software being proprietary. If the user loses any funds with

this robot the company could face backlash from its customers. If the company decided to make the software open source, there could be altercations between users as to what should be considered a 'winning' strategy.

6.3 Future Improvements

As the application is of quite broad structure many other pages or functionalities could be added in the future. One functionality that will be added is the publication of the app on the cloud using a service such as Google Cloud [27] or AWS [28]. This is a big task as many settings of the Django project will need to be changed such as IP addresses and databases. These databases will need to be migrated to a cloud server rather than running on a local machine.

Another of these functionalities is a friend system where users can add friends and where information can be sent from one another through a messaging service. Although this would not add to the main functionality and goal of the project it would be beneficial for user satisfaction. This is why it was not put at the forefront of development.

To complete achievement of all functional requirements proper encryption will be used to encrypt all types of data that may be sensitive to the user. Although there is a layer of encryption from Django built-in on further inspection this may not be enough to stop penetration testers from obtaining sensitive information.

7 Conclusion

In conclusion, the overall goal of this project was to create a trading app that would allow users to place trades either by themselves or through the use of a trading robot. I can confidently say that this goal has been obtained with all Functional requirements being achieved and most Non-Functional requirements being achieved. By allowing users to manage their balance in multiple ways by managing expenses or placing trades, with a responsive design and a high user rating score, all aspects of the project's aims have been covered.

The Artificial Intelligence aspect of this project was a huge part of the project being at the forefront of the development process. With all algorithms leading to profitable outcomes, although a small number of trades have taken place in some of these algorithms it shows that each algorithm is functioning as intended with high returns if they were to be used over the 6 stocks tested in the past market.

8 References

References

[1] A. Petrosyan, "Number of internet users worldwide from 2005 to 2023," 2024. viewed: 26 March 2024.

[2] B. N. Ivan Ashminov, "Trading 212," 2024. viewed: 8 March 2024.

[3] D. R. Yoni Assia, Ronen Assia, "Etoro," 2024. viewed: 8 March 2024.

[4] J. Danielsson, M. Valenzuela, and I. Zer, "Learning from history: Volatility and financial crises," *The Review of Financial Studies*, vol. 31, no. 7, pp. 2774–2805, 2018.

[5] Y. Finance, "Yahoo finance api," 2024. viewed: 7 March 2024.

[6] I. Gurrib, "The moving average crossover strategy: Does it work for the s&p500 market index?," *Gurrib, I.(2016), Optimization of the Double Crossover Strategy for the S&P500 Market Index, Global Review of Accounting and Finance*, vol. 7, no. 1, pp. 92–107, 2016.

[7] Pandas, "Pandas exponentially weighted moving average," 2024. viewed: 16 March 2024.

[8] C. mitchell, "Average directional index (adx): Definition and formula," 2024. viewed: 16 March 2024.

[9] J. Fernando, "Relative strength index (rsi) indicator explained with formula," 2024. viewed: 16 March 2024.

[10] Chart.js, "Chart.js," 2024. viewed: 3 April 2024.

[11] Django, "Django migrations," 2024. viewed: 13 March 2024.

[12] Django, "Django commands," 2024. viewed: 17 March 2024.

[13] SQLite, "Sqlite," 2024. viewed: 13 March 2024.

[14] J. C. S. Perez, "Django project structure: A comprehensive guide," 2023. viewed: 8 March 2024.

[15] Django, "Django channels," 2024. viewed: 13 March 2024.

[16] AlphaVantage, "Stock market api, reimagined," 2024. viewed: 10 April 2024.

[17] Redis, "Redis server," 2024. viewed: 16 March 2024.

[18] JavaScript, "Javascript," 2024. viewed: 16 March 2024.

[19] Jupyter, "Jupyter notebook," 2024. viewed: 21 April 2024.

[20] H. Percival, *Test-driven development with Python: obey the testing goat: using Django, Selenium, and JavaScript.* " O'Reilly Media, Inc.", 2014.

[21] Django, "Testing in django," 2024. viewed: 6 April 2024.

[22] K. M. Tracey, *Django 1.1 Testing and Debugging.* Packt Publishing Ltd, 2010.

[23] A. Johnson, "How to unit test a django form," 2020. viewed: 4 April 2024.

[24] M. Niranjanamurthy, A. Nagaraj, H. Gattu, and P. K. Shetty, "Research study on importance of usability testing/user experience (ux) testing," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 10, pp. 78–85, 2014.

[25] Google, "How to unit test a django form," 2024. viewed: 20 March 2024.

[26] Alpaca, "Alpaca markets," 2024. viewed: 8 March 2024.

[27] Google, "Google cloud," 2024. viewed: 16 March 2024.

[28] Amazon, "Amazon web services," 2024. viewed: 16 March 2024.

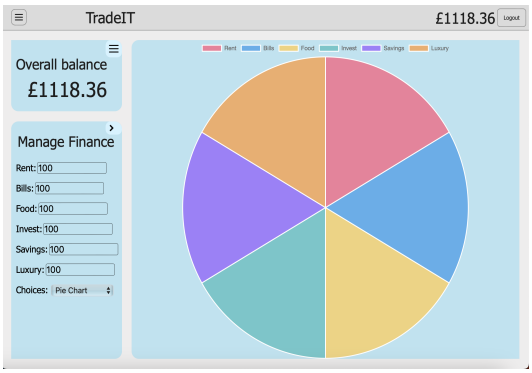
9 Slides & Code Video Link

https://universityofexeteruk-my.sharepoint.com/:v:/g/personal/sc1136_exeter_ac_uk/EYp3fSViDD1CmRiN6pmWOSUBFk5rjMVOqTg01kEC99TX1Q?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGluaONvcHkifX0&e=R4AtSe

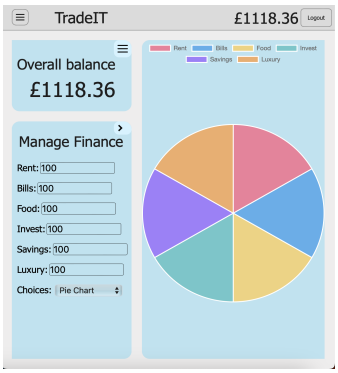
10 Appendix

10.1 A. Trade in Action

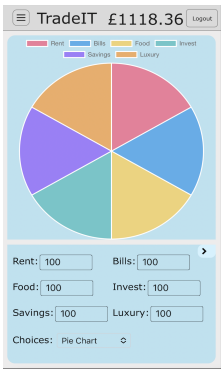
10.2 B. Different Viewing Modes



(a) Computer



(b) Tablet



(c) Phone

10.3 C. Implementation of Moving Average

```
def moving_average(data, span):
    return data['Close'].ewm(span=span, adjust=False).mean()
    # return data['Close'].rolling(window=span).mean() <- SMA
```

10.4 D. Implementation of ADX

```
def calculate_adx(data, window):

    data['tr0'] = abs(data['High'] - data['Low'])
    data['tr1'] = abs(data['High'] - data['Close'].shift())
    data['tr2'] = abs(data['Low'] - data['Close'].shift())
    data['TR'] = data[['tr0', 'tr1', 'tr2']].max(axis=1)

    data['DM+'] = np.where((data['High'] - data['High'].shift()) >
    ↪  (data['Low'].shift() - data['Low']), data['High'] -
    ↪  data['High'].shift(), 0)
    data['DM-'] = np.where((data['Low'].shift() - data['Low']) >
    ↪  (data['High'] - data['High'].shift()), data['Low'].shift() -
    ↪  data['Low'], 0)

    data['ATR'] = data['TR'].rolling(window=window).mean()

    data['ADM+'] = data['DM+'].rolling(window=window).mean()
    data['ADM-'] = data['DM-'].rolling(window=window).mean()

    data['DI+'] = (data['ADM+'] / data['ATR']) * 100
    data['DI-'] = (data['ADM-'] / data['ATR']) * 100

    data['DX'] = abs(data['DI+'] - data['DI-']) / (data['DI+'] + data['DI-'])
    ↪  * 100

    adx_values = data['DX'].rolling(window=window).mean()

    return (adx_values, data['DI+'], data['DI-'])
```

10.5 E. Implementation of RSI

```
def calculate_rsi(data, window):
    delta = data['Close'].diff()
    gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
    loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
    rs = gain / loss
```

```
    rsi = 100 - (100 / (1 + rs))
    return rsi
```

10.6 F. Implementation of Complete Algorithm

```
if complete:
    signals_sum = signals['signal_ma'] + signals['signal_rsi'] +
    ↪  signals['signal_adx']
    signals['sum'] = signals_sum
    signals['signal'] = 0.0
    signals.loc[signals['sum'] >= 2.0, 'signal'] = 1.0
    signals.loc[signals['sum'] <= -2.0, 'signal'] = -1.0
```

10.7 G. Loop through Stock Objects

```
# Loop is needed as some days are not available in the dataset
n = 1
while n < len(stock):
    if stock[0].date.isoformat() == start:
        row = dataframe.loc[dataframe['Date'] == start]
        if data.get('take_profit') or data.get("stop_loss"):
            # Sends the user's balance to change in the page dynamically
            await self.send(json.dumps({"date": stock[0].date.isoformat(),
            ↪  "close": stock[0].close, "signal": row["signal"].values[0],
            ↪  "money_in_account": user_profile.money_in_account}))
        else:
            await self.send(json.dumps({"date": stock[0].date.isoformat(),
            ↪  "close": stock[0].close, "signal": row["signal"].values[0]}))
        await sleep(1)
        break
    else:
        # Changes the start date to one day ahead
        n += 1
        start_date = datetime.datetime.strptime(start, "%Y-%m-%d")
        start_date += datetime.timedelta(days=1)
        start = start_date.strftime("%Y-%m-%d")
```

10.8 H. Add Take Profit Line to Graph

```
socket.onmessage = function (e) {
    if (!isPaused) { // Check if updates are paused
        // Add the Take Profit line data to the datasets
        myChart.data.datasets.push({
```

```
            label: 'Take Profit',
            data: takeProfitLineData,
            borderColor: 'green',
            borderWidth: 1,
            fill: false,
            pointRadius: 0
        });
        takeProfitLineAdded = true;

        myChart.update('none');

        if (takeProfitLineAdded) {
            // Adds point to the take profit line on each run
            var newGraphTakeLine = graphData.data.datasets[1].data;
            newGraphTakeLine.push(takeProfitValue);
            graphData.data.datasets[1].data = newGraphTakeLine;
        }
        // Points are animated from 0 axis which looks clunky so no
        ↪  animations
        myChart.update('none');
    }
}
```

10.9 I. Google Form

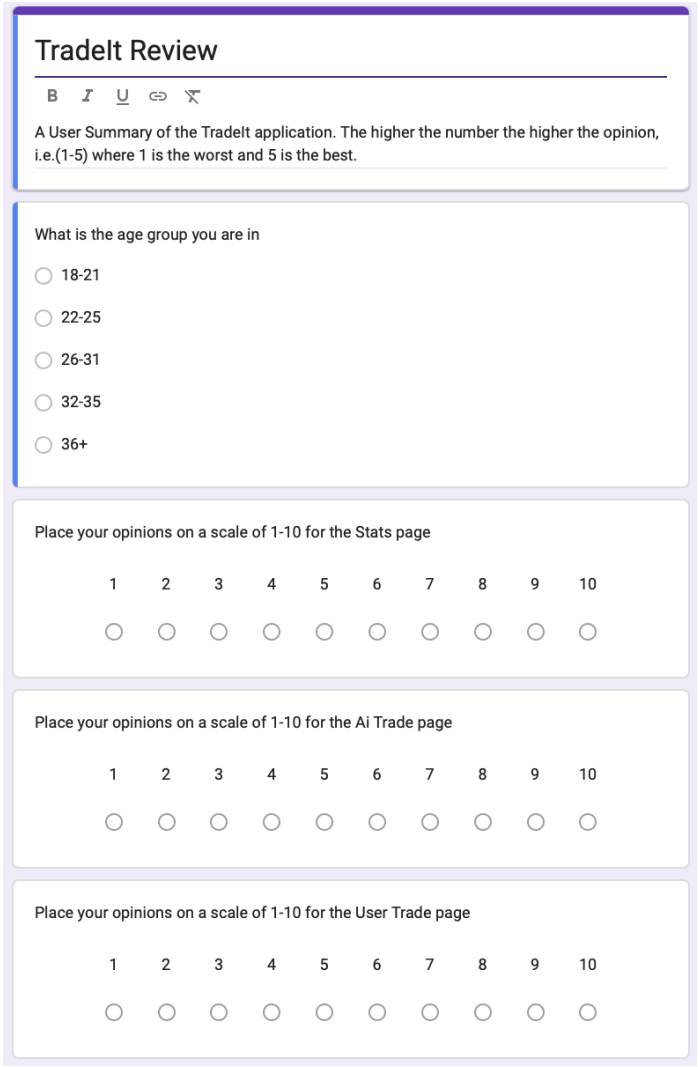


Figure 8: Page 1

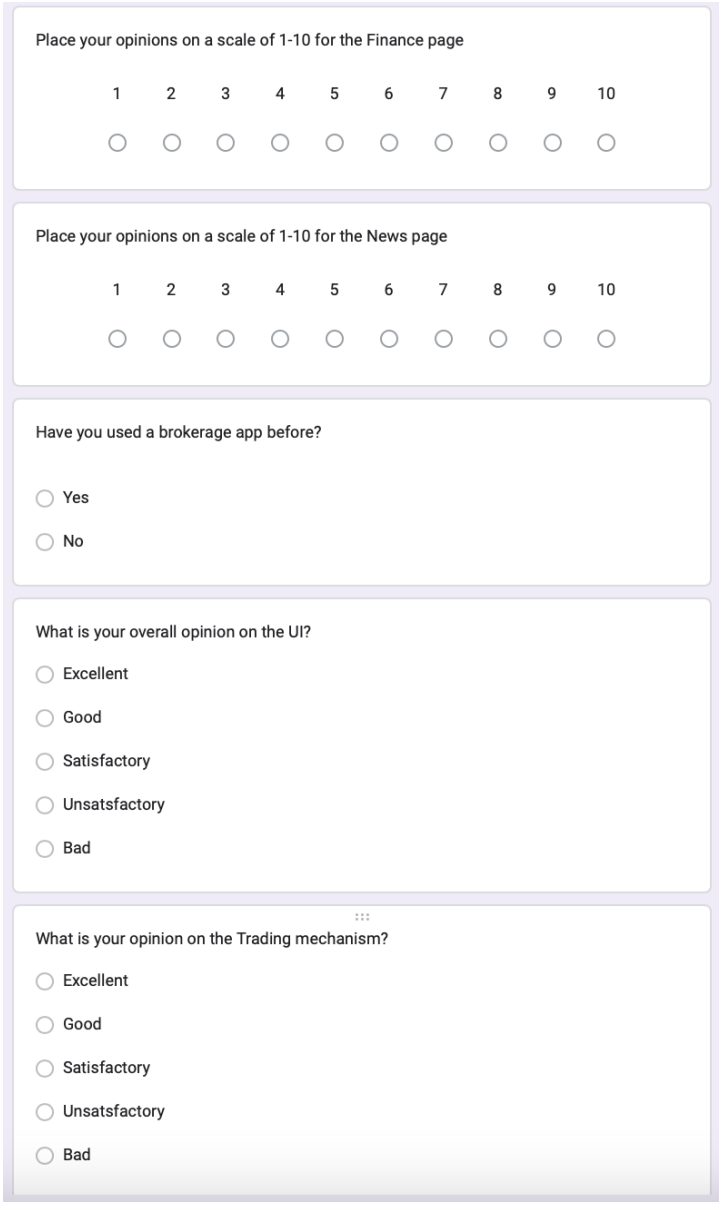


Figure 9: Page 2

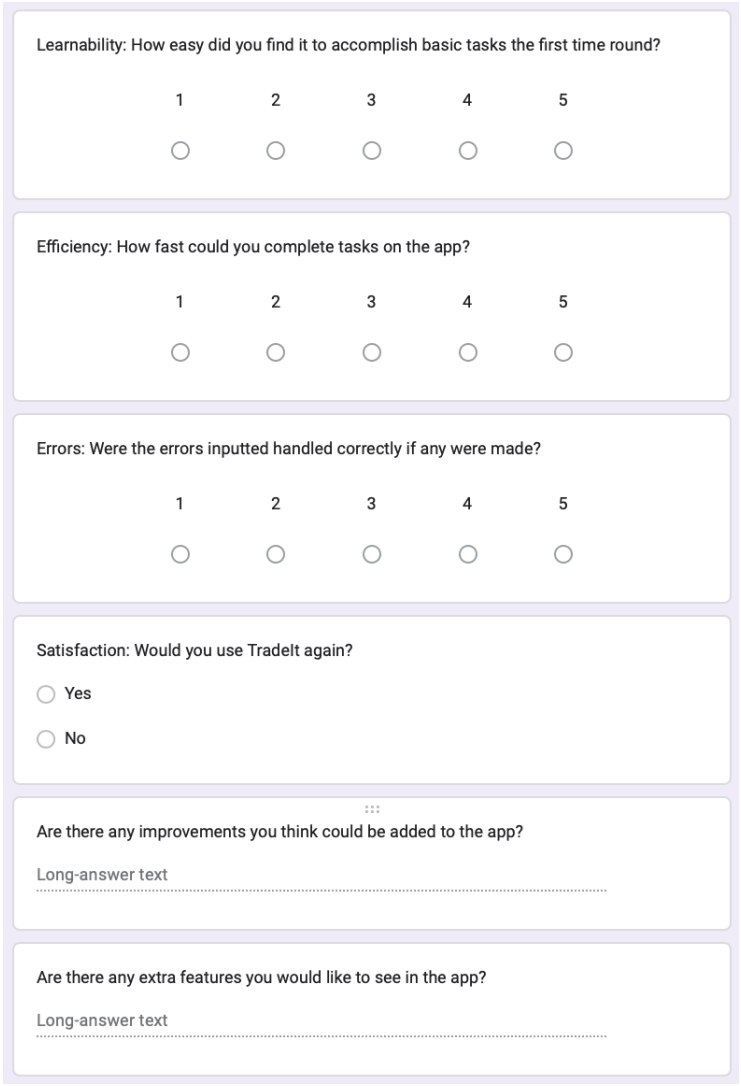


Figure 10: Page 3