7SENG013W Software Development Project
2023/2024

Implementation of a machine learning web application that clusters stocks from the S&P500 and predict the next closing price trend

Student: Romain Charles Kuhne

Supervisor: Hamed Hamzeh

Date: 05/09/2024

MSc Software Engineering (Conversion)
School of Computer Science and Engineering
University of Westminster

Abstract

This project aims to build on fundamental aspects of web development, machine learning, probability and statistics, and finance to develop two robust models. The first model predicts the next day's closing price trend for stocks in the S&P 500, while the second model clusters the entire dataset based on annual average return and volatility, displaying the results on an interactive scatter plot. The goal is to assist users from diverse educational backgrounds in making more informed investment decisions.

Python was chosen for this project due to its extensive support for machine learning libraries. For the web framework, Python's FastAPI was selected as a lightweight and widely-used option for developing machine learning web applications. On the client side, JavaScript with Material-UI was used to implement the components rendered to the user. Client-server communication is handled via HTTP protocols and Security is ensured using JWT token authentication and role-based access to critical administrative functions.

The overall architecture follows the MVC (Model-View-Controller) design pattern. To facilitate deployment and manage system dependencies, Docker was used to containerize the application into microservices, each in its own environment. The application is divided into three containers: one for the backend, one for the frontend, and one for the database. The database container uses a PostgreSQL image pulled from Docker Hub to store the models' data.

The training dataset for the machine learning models is stored in a CSV file. To ensure the training data remains up to date, a scheduler was implemented using AWS Lambda to gather daily stock data from the S&P 500 and update the CSV file.

Declaration

This report is submitted in partial fulfilment of the requirements for the
MSc Software

Engineering (Conversion) Degree at the University of Westminster.

This report has been prepared based on my own work. Where other published and

unpublished source materials have been used, these have been acknowledged in the text

of the report and included in the list of references.

Word Count: 22476

Student Name: Romain Charles Kuhne

Date of Submission: 05/09/2024

Acknowledgements:

To Hamed Hamzeh for helping me throughout the project.

Table of Content:

List of Figures:

List of Figures:

Chapter 1: Introduction

1.1 Introduction:

Since the advent of computers, a drastic shift in the field of computing has been observed, moving from a centralised discipline towards a form of horizontal specialisation, where each field focuses on a distinct domain. In recent years, driven by its pattern recognition and predictive abilities, machine learning has sparked the interest of experts in various sectors of society, including medicine, government, pharmaceuticals, and finance. Guided by the ideology of profit maximisation envisioned by Milton Friedman, financial analysts and individuals in the field of quantitative finance began utilising this novel technology with the aim of making wiser investment decisions by reducing risks, while optimising return. This objective of utilising machine learning and data visualisation in finance to enhance the decision-making process regarding investments is the reason why this project was implemented as it can be quite challenging to decide on which stock to purchase with the current overload of information.

This web application is primarily intended to be used by individuals interested in the domain of finance who aim to invest in stocks from the S&P 500. This platform is not aimed at financial institutions as it only focuses on a subset of stocks, and doesn’t include other financial products such as commodities, derivatives, futures and bonds.

The user first authenticates or creates an account on the sign-in/sign-up landing page before being granted access to the web application's services. After successful authentication, the user is redirected to the home page, where they can visualise clusters of stocks based on their average annual return and volatility on an interactive scatter plot or predict the next day's closing price trend for a stock. A navigation menu offers the option to log out,

in which case the user will be redirected to the landing page. Additionally, the navigation menu includes a third link called "Accounts" for admins, where a real-time table displays all user accounts fetched from the server along with their information. The table allows admins to block or unblock accounts, update user details, and manage permissions. The server handles the necessary logic for managing user accounts, permissions, and authentication. It also gathers financial data using the Yahoo Finance library, stores it in a CSV file, and processes the training dataset. This data is then used to either cluster stocks via a k-means model for visualisation or predict the next day's closing price trend using a random forest classifier.

Finally, this project report does not assume the reader is familiar with applied machine learning or finance. Therefore, specific terminology and concepts will be carefully explained throughout the project report.

1.2 Project Aims and Objectives:

1.2.1 Project Aims:

(PA1)
Provide financial information in the form of an interactive scatter plot to enable users visualise distinct clusters of stocks based on their the average annual return and volatility.

(PA2)
Provide a fine-tuned machine learning model with a satisfactory accuracy to predict the next day’s closing price with a decent UI to indicate a bullish or a bearish trend.

(PA3)

Provide a secure platform with JWT authentication and role-based access to crucial features enabling the management of user's credentials.

1.2.2 Project Objectives:

Objectives for (PO1):

(PO1.1)

Research the most suitable machine learning models to accurately predict price trends and divide a training set of stocks into distinct subsets based on similarities.

(PO1.2)

Understand the mathematical intricacies of the models used in this project to leverage them effectively.

(PO1.3)

Process the training dataset to eliminate unnecessary features while computing the features required by the appropriate model.

(PO1.4)

Fine-tune both machine learning models using appropriate techniques to reduce and limit prediction errors.

(PO1.5)

Gather sufficient metrics to evaluate the predictive power of both models.

Objectives for (PO2):

(PO2.1)

Develop a storage solution to efficiently safeguard and retrieve past training data used by both machine learning models.

(PO2.2)

Implement logic to ensure the training dataset is updated daily for accurate predictions.

(PO2.3)
Implement logic to ensure that the retrieved daily financial data is stored correctly, sorted by ticker symbol and date.

Objectives for (PO3):

(PO3.1)
Implement a robust web application with python and FastAPI's library using the MCV design pattern to ensure low coupling and separation of concerns.

(PO3.2)
Design the required models, controllers, services and routes to ensure a functional platform allowing for a secure authentication and interaction between the user and both machine learning models. allow the web app to be Implement logic to ensure that the retrieved daily financial data is stored correctly, sorted by

(PO3.3)
Implement the server-side logic to guarantee a secure JWT authentication to protect the backend API and a role-based access to prevent regular users from interacting with endpoints intended to manage the user's account.

(PO3.4)
Implement data storage functionalities to ensure user's credentials and the JWT are securely stored in the database and to allow for updates, retrieval or deletion of data.

Objective of (PO4):

(PO4.1)
User Docker and Docker Compose to compartmentalize the web application in microservices each in their own container with their respective dependencies to facilitate the deployment process.

(PO4.2)
Configure the docker-compose file to ensure containers are connected appropriately to guarantee a smooth communication channel between the frontend container to the backend with the one used to hold the database and use Docker Volumes to ensure all data stored while all containers are running is stored permanently.

1.3 Project Ressources:

This project was undertaken using a 2023 14-inch MacBook Pro with 16GB of RAM, 1TB of storage, and an M2 Pro chip. The software used during the project includes Jupyter Notebook for the implementation, processing, fine-tuning, and aggregation of metrics for both machine learning models. For development, the only integrated development environment (IDE) used was Visual Studio Code. The server-side was developed using the FastAPI library, and the client-side was developed using JavaScript.

For version control, a GitHub repository was created for the project, with changes pushed approximately every five days as soon as a major part of the logic was completed. Docker was used for containerization, pulling images from Docker Hub to define the Python version used in this project and for the database. Docker Compose was used to containerize the application into multiple containers.

For testing, the pytest-asyncio library was used to perform unit and integration testing asynchronously. Additionally, the Jenkins CI/CD tool was used to automate testing and gather metrics by creating a pipeline that outlines the steps to run the tests. Finally, the project was deployed to the web using AWS Lambda with a scheduler to enable the continuous gathering of daily financial data for the training dataset stored in the CSV file.

1.4 Outline of the Report:

The project report is structured as follows: In the second chapter, a critical review of four systems is conducted with the aim of evaluating which one is most appropriate as a foundation for our project. Each system will be assessed based on specific review criteria. The third chapter focuses on the system requirements of the web application. In this section, a stakeholder diagram and a context diagram will be used to examine potential users of the platform and describe the external entities that interact with the developed system. Additionally, the necessary functional, non-functional, and UI requirements will be listed to establish the essential, desirable, and luxury features of the system. The fourth chapter will concentrate on the system architecture. In this chapter, a use-case diagram, a sequence diagram, as well as UML class and ERD diagrams, will be provided to illustrate the overall design of the application. Chapter five is focused on software architecture. This section will present a UML class diagram, along with a technological review of the web application, including its security aspects and an overview of issues encountered during development. In chapter six, testing strategies and metrics will be discussed to demonstrate which tests passed or failed. Finally, the last chapter provides a conclusion to this software development project, highlighting the key aspects of the development process and discussing its limitations.

Chapter 2 Requirement Analysis:

2.1 Introduction:

In this chapter, a critical review of similar systems will be performed by evaluating them according to specific criteria to for the purpose of deciding which features will be implemented into the software. Since the intended

project focuses on two different machine learning models, different past projects will be analysed.

2.2 Systems to be reviewed:

2.2.1 Price Trend Predictor:

Regarding the price trend predictor service, two past papers were chosen, each utilising a different machine learning model to generate predictions based on financial features. The first one implements a web application for generating stock price predictions by gathering financial data using the yahoo finance library and processing them via a linear regression. As for the second project, even though it isn't centred around a web application, it was selected because it details the mathematical aspect of building a stock price trend prediction using the random forest model on computed financial indicators.

2.2.2 Stock Clustering and Data Visualization:

As for the second service of the web application, two past projects were chosen, each utilising the k-means algorithm, an unsupervised machine learning model to divide a dataset of stocks into distinct clusters and visualise them using a scatter plot. Even though both projects use the same model, both of their scatter plots yield different results as contrasting financial features were used. The first one utilised the expected return and the value at risk to segregate stocks listed on the Indonesian Stock Exchange into groups (Ridwan et al., 2021). In contrast to the first one, the second study centers its model around the return on investment and on the asset turnover ratio (Bin, 2020).

2.3 Review Criterias:

In this section, review criterias required to analyse and evaluate past similar projects are listed. The principal objective of these criterias is to provide analytical metrics from which strengths and weaknesses of systems will be summarised with the end goal of selecting which architecture design, features or model appears to be the most appropriate for the development process of the software.

2.3.1 Review Criterias related to the price prediction model:

(RC1.1)
Periodicity of the training dataset used by the author - how many years of data did the author utilize to train the machine learning model.
(RC1.2)
Selection of financial features made by the author - Which feature set X did the author select to fit his training data set into the model.
(RC1.3)
Category of machine learning model used - Which machine learning algorithm did the author decide to employ.
(RC1.4)
Fine tuning methods used to enhance the model's prediction - Which method did the author put to use in order to improve the prediction capacity of the model.
(RC1.5):
Key performance Indicators of the model - Which evaluation metrics did the author utilize to evaluate his prediction capacity.

2.3.2 Review Criterias related to the clustering model:

(RC2.1)
Nature of the dataset used by the machine learning model - Which type of financial data did the author choose as training dataset.

(RC2.2)
Selection of financial features made by the author - Which feature set X did the author select to fit his training data set into the model.
(RC2.3)
Number of clusters from which the dataset will be divided into - In how many groups will the dataset be partitioned into.
(RC2.4)
Method used to remove potential outliers -Which method was chosen by the author to remove data points significantly different from others.
(RC2.5):
Visual interface used by the author to display the result of the clustering - Which type of plot did the author select to display the partitioned dataset.

2.4 Review of the Systems:

2.4.1 Review of the Price Prediction Model:

2.4.1.1 Price Prediction using Linear Regression:

The first system to be reviewed centers around predicting future prices of stocks from up to a year using a linear regression model to the user via a web based application (Antad et al., 2023). Regarding the first review criteria (RC1.1) related to the periodicity of the financial dataset used by the model, no information could be extracted as the author didn't disclose it in his report. As for (RC1.2), even though the author didn't explicitly mention which feature set X did he use to predict future prices, it can be assumed he used historical closing prices obtained by the yahoo finance library. As for the third review criteria related to the processing model (RC1.3), this project fits past financial data into a linear regression model to predict future prices as label y. This model functions as follows: the training dataset is plotted in a graph according to a feature x against its label y. Then a straight line used to generalise labels

is drawn by finding the slope that minimises the mean square error or the distance between each data point and the line (Deisenroth et al., 2024). As for the method the author selected to optimise the prediction of his algorithm, he mentioned using normalisation. Normalisation is a data processing technique commonly used in machine learning to ensure scale features in a uniform manner so that each feature equally contributes to the prediction (Seongjai, 2024). Lastly as for the last review criteria, related to the model's evaluation indicator (RC1.5), he only employed the accuracy score to characterise his software predictive capability claiming it ranged between 75% up to 85%.

Figure 2.4.1.1: Prediction results for the next 90 days for an Indian stock

2.4.1.2 Price Prediction using Random Forest:

The second system employs a different approach to address this problem by processing historical stock data with a random forest classifier to predict whether the next day's closing price will trend upward or downward (Basak et al., 2019). Regarding the first review criterion (RC1.1), the author mentioned that she utilised historical financial data from ten popular publicly traded stocks, covering the period from when they became publicly traded up to February 3, 2017. After calculating the period from which each stock became

public to the end date mentioned by the author, the average duration of the financial data is approximately 27 years. Concerning the second review criterion (RC1.2), which relates to the feature set used in the study, the author explicitly stated that she used the Relative Strength Index (RSI), William's Percentage (W%), the Stochastic Oscillator (SO), the Moving Average Convergence Divergence (MACD), the Price Rate of Change (PROC), and the On-Balance Volume (OBV) as the dataset. As for the third criteria (RC1.3) which relates to the choice of the machine learning model used, the author employed a tree classifier algorithm to predict future closing price trends. This algorithm functions as follows: the training data set recursively splits the dataset into subsets based on feature values and utilises entropy as a metric to evaluate the level of disorder of each subsets (Mueller et al., 2016). Regarding the fourth criteria related to the optimization method employed (RC1.4), a Random Forest algorithm was used. This algorithm basically attempts to aggregate the predictions made by all trees (Müller et al,. 2916). Finally, regarding the evaluation metrics (RC1.5) the author displayed multiple tables listing the results for each metrics such as the accuracy, recall, Precision, Specificity, F-score, Brier Score and AUC.

Company Name	Trading Window	Accuracy	Recall	Precision	Specificity	F-Score	Brier Score	AUC
AAPL	3	65.26	0.71	0.66	0.58	0.68	0.35	0.70
	5	72.55	0.78	0.73	0.67	0.75	0.28	0.79
	10	78.80	0.81	0.80	0.76	0.81	0.21	0.87
	15	82.01	0.85	0.82	0.78	0.84	0.17	0.90
	30	85.34	0.88	0.86	0.81	0.87	0.15	0.93
	60	90.44	0.93	0.90	0.86	0.92	0.10	0.96
	90	93.02	0.95	0.93	0.90	0.94	0.07	0.98

Figure 2.4.1.2 : Evaluation metrics of the Random Forest for AAPL

2.4.2 Review of the Stock Clustering Model:

In this subsection, two systems centred around the partition of a set of stocks into distinct clusters will be analysed using the respective review criterias listed previously to select which model appears to be the most suitable for the implementation of the data visualisation service in the software in development.

The first project clustered stocks listed on the Indonesian Stock Exchange according to their expected return versus volatility (Ridwan et al., 2021). Regarding the first review criterion related to the nature of the dataset (RC2.1), the author indicated that only Indonesian stocks large enough to be listed in the IDX30 were used for the project. For the second review criterion related to the features used by the model (RC2.2), the author explicitly mentioned using both the stock's expected return and its volatility as the feature set. Concerning the third review criterion related to the number of subsets the dataset is partitioned into (RC2.3), the author reported that the optimal number of clusters derived from the elbow curve is 2. As for the method chosen to remove potential outliers (RC2.4), the author stated that although some outliers were identified, they were not removed using any available method. Lastly, in the fifth review criterion related to data visualisation (RC2.5), distinct clusters are displayed in a two-dimensional scatter plot.

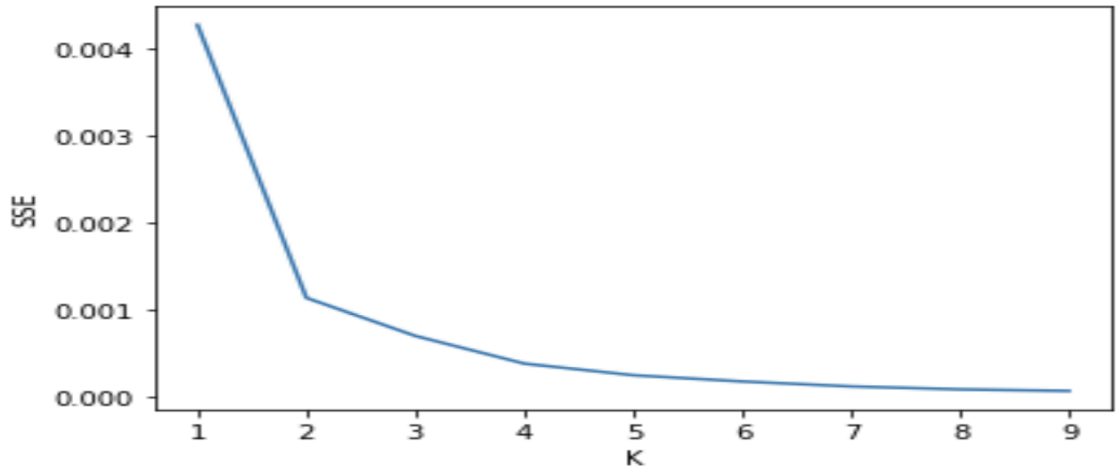


Figure 2.4.2.1: Plot of the Elbow curve to select the optimal number of clusters

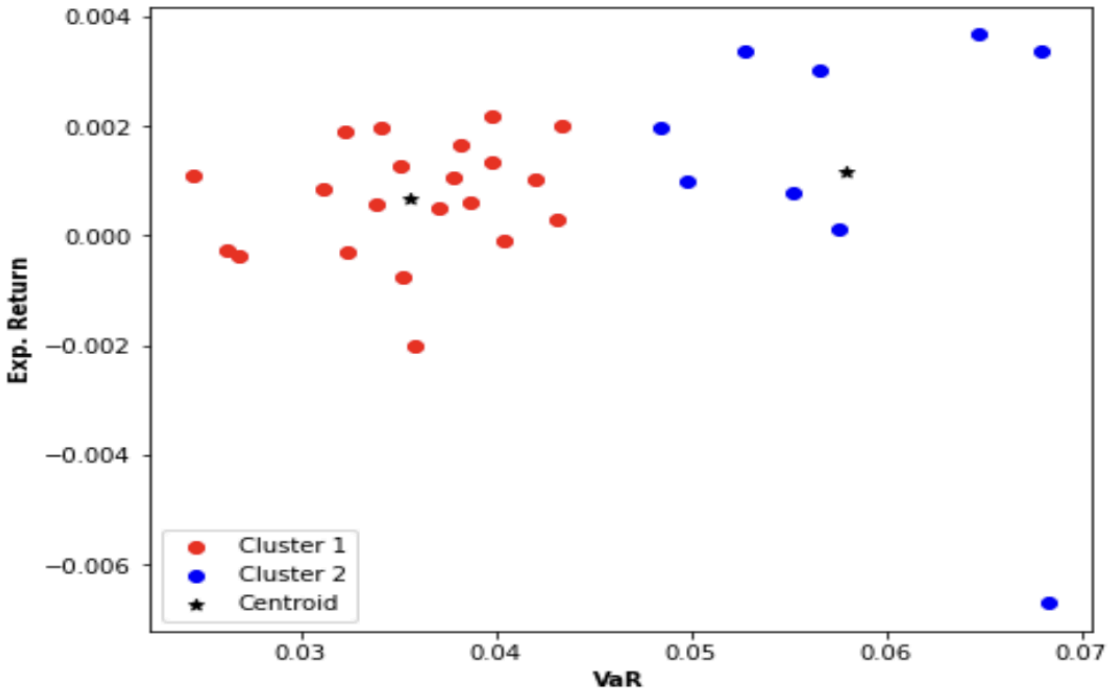


Figure 2.4.2.2: Scatter plot of stock clusters listed in the IDX30 index

Regarding the second system, stocks listed in the S&P 500 were clustered according to their respective computed return and asset turnover ratio using the k-means algorithm (Bin, 2020). For the first review criterion related to the nature of the dataset (RC2.1), the author utilised stocks from 232 publicly traded companies listed in the S&P 500 to fit the unsupervised model. Regarding the second review criterion related to the features used by the

model (RC2.2), the author explicitly mentioned using both stock return and asset turnover, derived from raw data gathered via the Yahoo Finance library. Concerning the number of subsets the dataset is partitioned into (RC2.3), the author employed a different approach by using the silhouette method to determine the optimal number of clusters. With this approach, the author identified different numbers of clusters: K = 4, K = 6, and K = 7. As for the method chosen to remove potential outliers (RC2.4), the author identified outliers using histograms but did not remove them from his scatter plots. Finally, in the fifth review criterion related to data visualisation (RC2.5), clusters are displayed in a three-dimensional scatter plot.

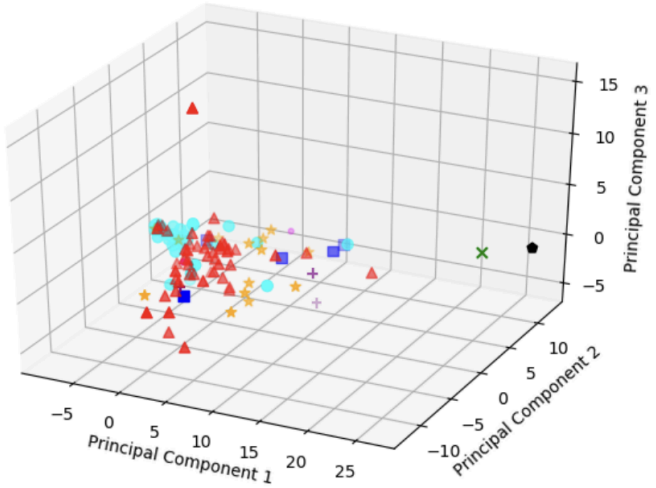


Figure 2.4.2.3: Scatter plot of the dataset using K = 7

2.5 Conclusion and Comparison of Systems:

In the previous subsections, a thorough analysis was performed on two distinct machine learning models using specific review criteria by comparing two systems for each model. After describing the respective features of each model according to their criteria, a critical evaluation will be conducted in this subsection to describe the strengths and weaknesses of each system, with the aim of selecting the one most suited to base the web application project on. Then, a comparison table will be created, listing the respective reviews

marked with a grade ranging from 1 to 5, to address potential issues inherent to each system.

2.5.1 Stock Price Prediction Model:

Table 2.5.1.1: Evaluation of system review criteria for Price Forecast Model

Review Criterias	RC1.1	RC1.2	RC1.3	RC1.4	RC1.5
System 1	1	2	3	3	2
System 2	5	4.5	4.5	4.5	5

Concerning the stock price forecasting model, both systems employ contrasting approaches to this problem. Regarding the periodicity of financial data gathered as the training dataset (RC1.1), the first system failed to disclose this crucial information, partially obscuring how the model was trained, while the second one revealed storing an average of 27 years of training data. With regard to the feature set utilised (RC1.2), since the first system relied on linear regression for its predictions, it was deduced that the closing price was the only feature set used. In contrast, the second system utilised a wider range of financial price indicators to reinforce the model's predictive ability. In respect to the machine learning model used (RC1.3), the first system employed a rather simple approach based on linear regression, while the second system used a more elegant and complex model, a tree classifier, to generalise. As for the method employed by both systems to enhance the predictive ability of their models, the first one used normalisation to ensure that features are scaled uniformly, preventing issues related to unequal participation in generalisation. In contrast, the second system resorted to a more complex method to optimise its model by utilising random forests, ensuring randomness in the choice of the dataset and the selected feature set to improve predictions. Finally, regarding the criteria for the evaluation metrics computed to characterise their models (RC1.5), the first system failed to provide sufficient key performance indicators, only stating that

its model yielded an accuracy rate ranging from 75% to 85%. In contrast, the second system utilised a wider range of indicators to evaluate the performance of its predictions, such as accuracy, recall, precision, specificity, F-score, Brier score, and AUC.

In conclusion, after a thorough analysis of both systems based on the review criteria listed previously, the second system appears to be the most suited to base the stock price forecasting service of the machine learning web application in development. It delivered a transparent methodology regarding the periodicity of the training dataset used, the manner in which the model was fine-tuned, and indicated promising generalisation performances according to the wide range of computed performance metrics. As for the first system, even though it utilised a simpler approach to solve the issue of stock price forecasting, it failed to disclose crucial information related to the model's training dataset and sufficient metrics to evaluate its predictive performance.

2.5.2 Kmeans Stock Clustering Model:

Table 2.5.2.1: Evaluation of system review criteria for clustering model

Review Criterias	RC2.1	RC2.2	RC2.3	RC2.4	RC2.5
System 1	3.5	4	4.5	1	4
System 2	5	4	3.5	3.5	4.5

Concerning the stock clustering model, although both systems used a similar approach to identify groups of related stocks, the methods employed varied, leading to distinct results. Regarding the review criterion related to the nature of the dataset (RC2.1), while the first system mentioned using stocks listed on the IDX30, it did not provide a table listing all the stocks included in the dataset. In contrast, the second system, which used a dataset of 232 companies listed in the S&P 500, provided a detailed table. As for the feature

set selected by each system (RC2.2), both systems used valid metrics to segregate their datasets. The first system utilised return on investment and volatility, while the second used asset turnover and the Sharpe ratio as indicators. The difference in approach reflects the different realities each system is attempting to depict. Regarding the method employed to select the optimal number of clusters K (RC2.3), the first system used the elbow curve method, selecting the K that led to the sharpest decrease in the objective function. The second system implemented the silhouette method, which measures how similar a data point is to its own cluster compared to other clusters, and determined that 4 was the optimal number of clusters. Both methods are technically valid and commonly used to define the number of clusters to divide the dataset. Regarding the methods employed to remove outliers in the model (RC2.4), while the first system failed to remove outliers, the second one utilised a histogram approach to analyse the distribution of the dataset and successfully identified outliers but did not remove them. Finally, regarding the visual medium used to plot each cluster (RC2.5), both systems utilised a scatter plot. The only difference is in the dimensionality of the scatter plot: as the first system used two features to fit the data through the k-means algorithm, each data point is segregated using two axes, while the second system required three.

In conclusion, after a thorough analysis of both clustering systems based on the review criteria listed previously, a combination of features from both will be used for the development of the clustering service in the web application. Regarding the nature of the dataset to be used, the second system appears to be more suitable, as it includes a larger number of stocks listed in the S&P 500. Concerning the feature set used by the k-means algorithm, the annual average return and volatility will be computed to partition the dataset, as these provide useful insights to investors about which stocks to select to reduce volatility while maximising returns. For the method employed to identify the optimum number of clusters, the elbow curve utilised by the first system will be

adopted, as it is a simple yet effective method. In terms of removing outliers, the approach from the second system, which focuses on removing data points that fall outside a central range, will be used. Finally, as the feature set will comprise only two features, the clusters of data will be displayed using a bi-dimensional interactive scatter plot.

Chapter 3 System Requirements:

3.1 Introduction:

In this chapter, a system requirement evaluation will be performed with the aim of identifying the main stakeholder the web application is intended for as well as to illustrate which main entities or stakeholder will be interacting with the software in question. Moreover, multiple features will be listed in an organised by priority to determine which of them is required by the software to deliver a minimal viable product to end-users.

3.2 Stakeholder's Diagram:

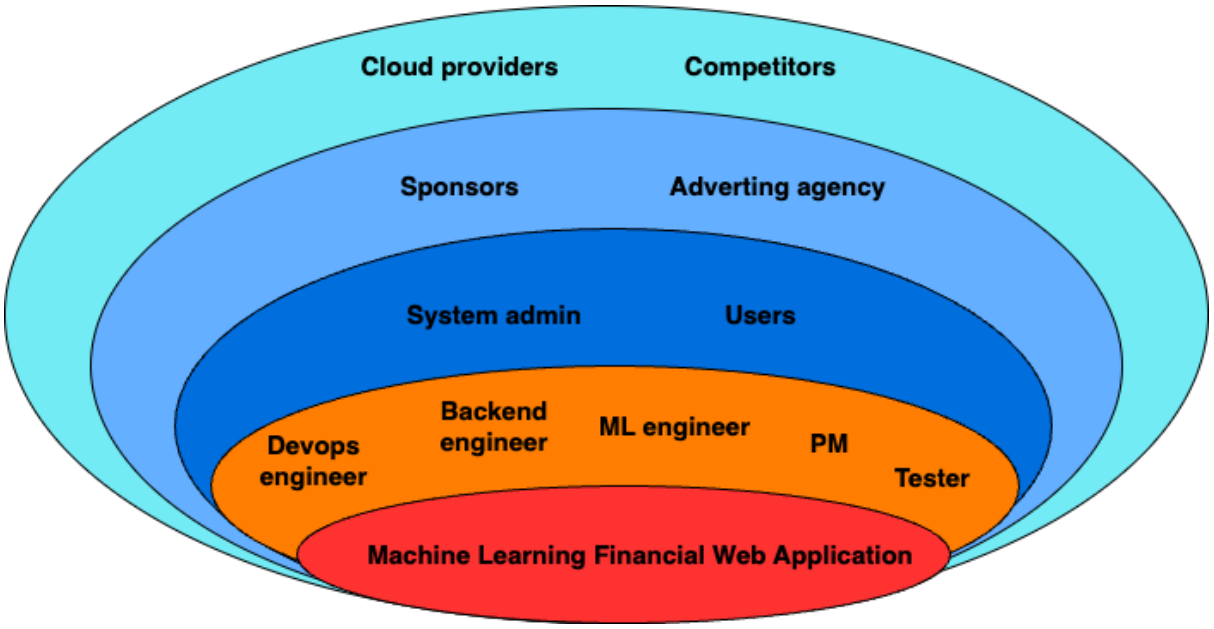


Figure 3.2.1: Stakeholder's diagram

Above is the stakeholder diagram responsible for identifying the main groups involved with the machine learning financial web application. To understand who is impacted by the product, it is crucial to comprehend how the diagram is structured. The first inner layer, or core layer, represents the product developed to be marketed to end users. The product in question is a financial web application that uses machine learning to offer two services. The first service allows users to visualise clusters of stocks according to their average annual return against volatility, providing insightful information regarding investment opportunities. The second service offered by the platform concerns closing price trend forecasting, enabling users to select a stock from a company listed in the S&P 500 and run a prediction for the next day to determine if the closing price will be higher or lower than the previous one.

The second inner layer represents primary stakeholders directly involved in the prototyping, design, testing, and management of the project to ensure a viable end product is delivered on time with the functional features to the public. This layer encompasses a machine learning developer responsible for

the processing of the data, computation of financial features, and the design and implementation of the appropriate algorithms and methods to fine-tune and optimise the model to improve its generalisation capability. The next stakeholder in this layer is the backend developer responsible for developing a secure and flexible server-side architecture to support both users and the machine learning model. A tester is also present, responsible for implementing a testing strategy (unit tests, integration tests) to ensure all features are correctly working together. Next is a DevOps engineer responsible for the continuous integration and deployment of the web application to ensure new features are added and that the web application is hosted in the appropriate environment. The product manager's role is to ensure schedules are respected, preventing delays, and to ensure all team members receive the right information.

The third inner layer represents secondary stakeholders who interact with the product but not at the same depth level as the primary stakeholders responsible for its success. Secondary stakeholders include end-users who utilise the product and system administrators whose tasks are to ensure the integrity of the deployed platform.

The fourth layer constitutes groups or individuals who do not directly interact with the product but whose interests are closely tied to its success. These are sponsors and advertising agencies whose roles are to either provide funds or advertise the software to the targeted audience using marketing tools.

Finally, the fifth layer represents stakeholders who are indirectly involved with or impacted by the product. This layer comprises cloud providers such as AWS, where the web application is deployed. It is included because as traffic grows across the platform, the cloud provider must ensure it can support the growing number of users. Lastly, competitors are mentioned as the core product can directly disrupt their business.

3.3 Context Diagram:

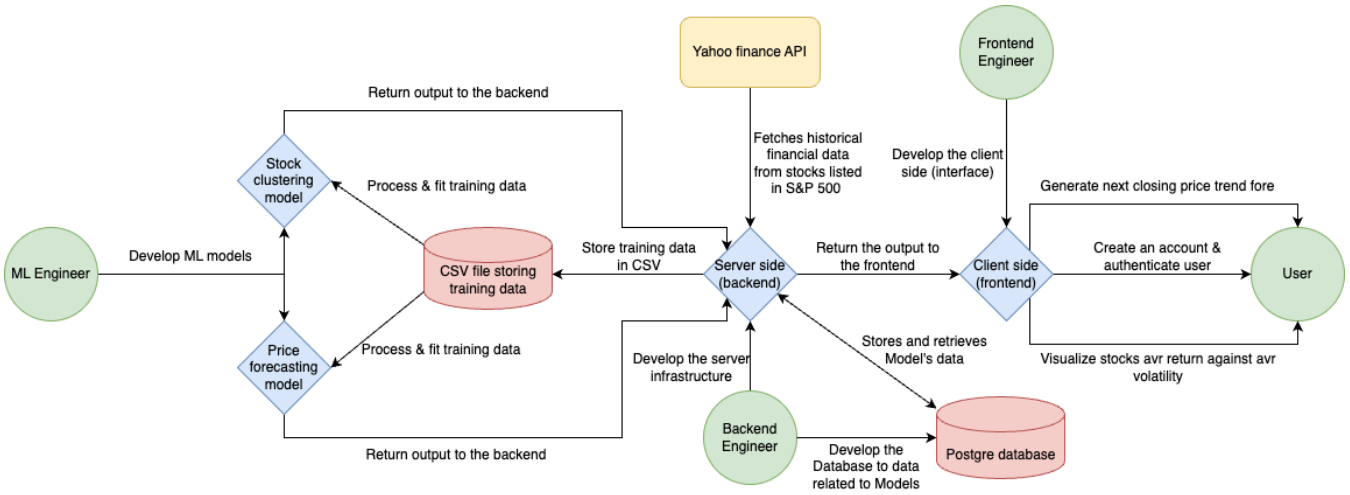


Figure 3.3.1: Context Diagram of the machine learning web application

The above diagram illustrates how the main entities interact with the machine learning web application. The main stakeholders involved in the application are depicted in green. The systems developed and interfaces necessary for the web application to function are illustrated in blue. The API used by the system is shown in yellow, and the storage medium used by the application is represented in red.

As illustrated above, the machine learning engineer is the stakeholder responsible for selecting the model used to forecast the next stock closing price trend, as well as the model used to segregate stocks into clusters according to their average annual return and volatility. In addition to developing the model, this stakeholder is also responsible for processing the data and selecting the method employed to fine-tune the model. They are also involved in computing the necessary metrics required to evaluate the model's performance. Regarding the training dataset, it is fetched using the Yahoo Finance API via the backend to retrieve historical data of all stocks listed in the S&P 500 for a period of five years, storing them in an ordered manner according to stock tickers and date in a CSV file.

The individual responsible for building the system's infrastructure to handle the machine learning model and users is the backend engineer. This stakeholder develops the logic and architecture to ensure a secure platform, stores data related to models in the appropriate database, and ensures the connection and functioning of both the machine learning model and the server side. They are also responsible for instantiating and managing the database, fetching historical training data into the CSV file, and incrementing it daily with recent stock prices to keep it up to date.

The frontend engineer is responsible for creating the user interface and the effects displayed to the user. In addition, they ensure the interaction between the user interface and the backend is maintained to enable smooth data transfer between both interfaces.

Finally, the end user is the one utilising the product. They will be able to create an account, authenticate with their credentials, generate a prediction on tomorrow's stock price trend for a specific stock they select, and visualise clusters of stocks partitioned into groups based on their average annual return and volatility.

3.4 System Requirements:

In order to understand which features will be implemented into the machine learning web application, it is essential to discuss the system requirements as they enable the stakeholders to know which functionalities are necessary to get a minimum viable product. In this section, the system requirements of the application will be detailed and listed in three subgroups respectively: functional requirements, non-functional requirements and user-interface requirements. In addition to these categories, each requirement in these groups will be ordered according to their level of priority. These levels of priorities are: essential, desirable and luxury. The first order of priority is for functionalities required for the product to be functional. The second order of priority englobes features that could improve the functionalities of the system

and user interfaces. The last order of priority encompasses nice to have features but not required for the system.

3.4.1 Functional Requirements:

3.4.1.1 Functional Essential Requirements:

(FR1)
Provide a stock price trend forecasting model with strong evaluation metrics using fine-tuning methods such as grid search, time series cross-validation, and random forest.

(FR2)
Provide a k-means algorithm that precisely segregates clusters of stocks from publicly traded companies listed in the S&P 500 according to their average annual return and volatility while removing potential outliers.

(FR3)
Provide a backend interface that fetches all financial data for all stocks in the S&P 500 to gather the required training dataset.

(FR4)
Provide a local means of storage to ensure training data fetched from Yahoo is stored in a structured manner according to groups of ticker symbols and their dates in increasing order.

3.4.1.2 Desirable Functional Requirements:

(FR5)
Implement logic in the backend to ensure the financial training dataset stored locally is updated daily, guaranteeing that the training data is continuously up-to-date.

(FR6)
Implement logic in the backend to ensure that newly fetched data during the incremental process is correctly stored in local storage according to its respective ticker symbol and date, ensuring the training dataset is in order.

(FR7)
Implement a text file that records the date when the training dataset was modified to keep track of all changes.

3.4.1.3 Luxury Functional requirements:

(FR8)
Provide a service that enables the user to purchase stocks with dummy currency to simulate investment strategies.

(FR9)
Provide users the option to follow daily stock price changes via candlestick graphs.

3.4.2 Non-Functional Requirements:

3.4.2.1 Essential Non-Functional Requirements:

(NFR1)
Implement User and Token Models and Controllers to handle the backend logic for user credential authentication and JWT access, protecting the backend API.

(NFR2)
Provide a suitable database that supports relational mapping between models to ensure that user credentials and JWT tokens are stored and CRUD operations can be performed on user details and JWT tokens.

(NFR3)
Implement a security service to correctly hash the JWT token payload with an appropriate algorithm.

3.4.2.2 Desiragle Non-Functional Requirements:

(NFR4)
Implement logic for role-based access to ensure that crucial functionalities related to user account management, such as updating user details, are appropriately managed.

(NFR5)
Implement a hashing service to encrypt user passwords in the database.

3.4.2.3 Luxury Non-Functional Requirements:

(NFR6)
Implement Kubernetes to orchestrate containers and ensure the system scales by adjusting the number of running pods based on real-time demand.

3.4.3 User Interface Requirements:

3.4.3.1 Essential User Interface Requirements:

(UIR1)
Ensure the interface of the machine learning web application is responsive and that rendered components are correctly displayed on different types of devices.

(UIR2)
Provide an interactive scatter plot where the user can click on a point with the

cursor to display the ticker symbol, average annual return, and volatility, with different colours used to identify each stock cluster.

(UIR3)
Provide a section where the user can select a stock and run the next day's closing price trend forecasting model. If the prediction indicates a bullish trend, a green arrow will be displayed; if bearish, a red arrow will be shown.

3.4.3.2 Desirable User Interface Requirements:

(UIR4)
Implement a private page where continuous checks are made to ensure that only users with a valid JWT token and the role of admin can access it. This page will display a table with all created accounts, allowing the admin to update user roles, block or unblock an account.

3.4.3.3 Luxury User Interface Requirements:

(UIR5)
Provide a section where users can select a stock price to be displayed in real-time on a candlestick graph.

(UIR6)
Provide a section where users can select stocks and add them to a portfolio, which will display a colour indicating whether their portfolio is profitable or not.

Chapter 4 System Design:

4.1 Introduction:

After presenting a detailed view of the main stakeholders involved with the machine learning web application and listing the functional, non-functional, and user interface features of the app in a structured manner, the next step is to perform a system design. In this chapter, a use case diagram will first be illustrated and explained to provide an understanding of the interactions between the system and external actors. Following that, a sequence diagram will be presented to demonstrate how objects interact within the system. An entity-relationship diagram will also be illustrated and explained to show the intricacies of how models and their data are stored in the database. Finally, screenshots of the client-side of the application will be provided.

4.2 Use Case Diagram:

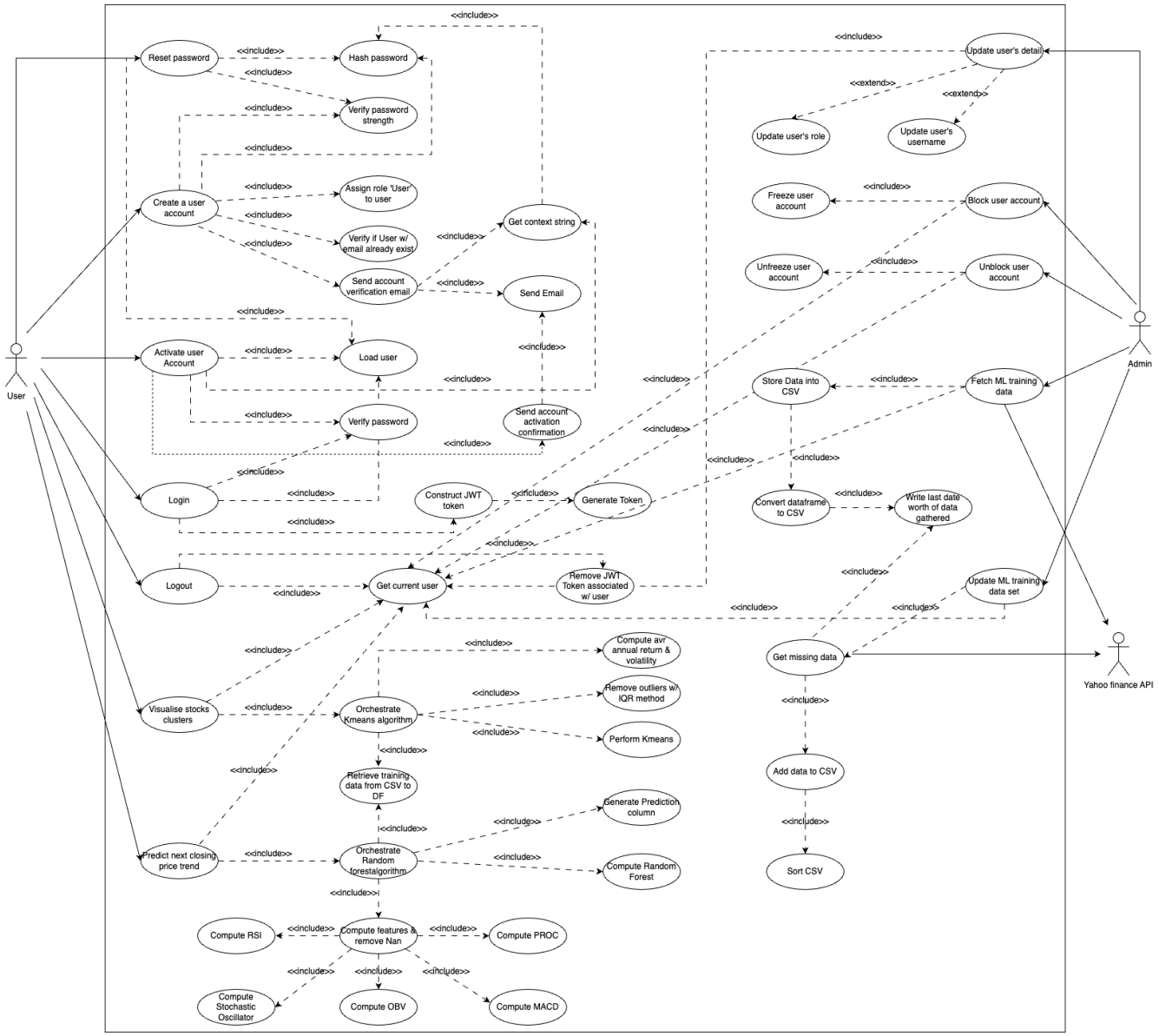


Figure 4.2.1: Use Case Diagram of the machine learning web application

4.2.1 Create a user account:

Principal actor: User

1) User enters his credentials (Email, username, & password) in the signup form.
2) A request is sent from the frontend to the respective backend API endpoint URL: POST://localhost:8080/users/signup with the following payload {email:EmailStr, username:str, password:str}
3) System verifies that a user doesn't exist with the same email and username.
4) System verifies the password strength.
5) If true, the system hashes the password to encrypt it in the database.
6) System assigns the role of 'user' to the user.
7) System commits these changes to the database.
8) System sends an account verification link to the user's email address with a unique key attached in the link.

4.2.2 Activate a user account:

Principal actor: User

Precondition: User has already created an account but not activated it
1) User logs into his email and clicks on the verification link.
2) Takes the user into this backend API route:
POST: POST://localhost:8080/users/verify-account" with the following payload: {email:EmailStr, token:str}
3) System loads the user from the database to use it as an object.
4) System generates and assigns a unique key to the user for account verification.
5) System verifies if the expected key and extracted key from the URL match.

6) If yes, the system updates the user's is_active and is_authenticated fields to true.
7) System commits these changes made to the user to the database.
8) System sends an account activation confirmation email to the user.

4.2.3 Reset user's password:

Principal actor: User
Precondition: User already has an active account but forgot his password
1) User enters his credentials (Email, username, and new password).
2) Request is sent from the frontend to the respective backend API endpoint URL:POST/localhost:8080/auth/reset-password" with the following payload: {email:EmailStr, username:str, new password:str}.
3) System verifies if the email and username belong to a user's account.
4) If true, the system hashes the new password and updates it in the database.
5) System commits the change in password to the database.

4.2.4 User login:

Principal actor: User or admin
Precondition: User or admin already has an active account
1) User enters his email and password into the login form.
2) Request is sent from the frontend to the respective backend API endpoint URL: POST://localhost:8080/auth/login" with the following payload:
{email:EmailStr, password:str}
3) System verifies the user's credentials by attempting to retrieve the user.
4) If true, and if the user's is_authenticated and is_active statuses are both true, the system generates and assigns a JWT token with an access key, refresh key, and expiration time. The corresponding user's role is also included in the payload.

4.2.5 User logout:

Principal actor: User or admin

Precondition: User or admin already is currently logged in

1) User clicks on the logout section in the navigation bar.

2) Request is sent from the frontend to the respective backend API endpoint URL: POST//localhost:8080/auth/logout

3) System retrieves the user from their JWT token.

4) System clears the JWT token from local storage, removes the token associated with the user from the database, and commits these changes to the Token table.

5) User is redirected to the default landing page (login/sign up).

4.2.6 Visualise stock clusters:

Principal actor: User

Precondition: User already is currently logged in

1) User clicks on the "Visualise stock clusters" button.

2) Request is sent from the frontend to the respective backend API endpoint URL: GET//localhost:8080/visualize/kmean-cluster

3) System fetches the user based on their token to determine if they are a valid user.

4) System converts the training data from the CSV file into a pandas DataFrame.

5) System computes the feature set for each stock X = {Avr Annual Return, Avr Annual Volatility} and creates a new DataFrame with the stock's ticker symbol as the index and both features as columns.

6) System removes potential outliers via the IQR method by computing the 25th and 75th percentiles, determining the lower and upper bounds, and removing data points outside of those ranges. The returned output is a DataFrame.

7) System performs the k-means algorithm, creating 4 clusters determined using the elbow method, and returns a JSON object to the frontend.
8) Frontend converts the JSON object and plots each data point in an interactive scatter plot, colour-coded according to their respective clusters.

4.2.7 Forecast next day's closing price trend:

Principal actor: User
Precondition: User already is currently logged in
1) User selects a stock listed in the S&P 500 and clicks on "Forecast Trend."
2) A request is sent from the frontend to the respective backend API endpoint URL:GET://localhost:8080/predict_next_closing_trend/predict_closing_price_trend with the ticker symbol as the payload
3) System verifies the user based on their token to determine if they are valid.
4) If valid, the system converts all training data from the CSV to a DataFrame with features (columns) = {High, Low, Close, Volume}.
5) System processes the DataFrame to compute the required features.
5.1) System computes the Relative Strength Index (RSI) and returns a DataFrame with the newly computed feature included in a column called RSI.
5.2) System utilises the returned DataFrame to compute stochastic oscillators (K%, D%, & R%) and returns a new DataFrame with features (columns) = {RSI, K%, D%, R%}.
5.3) System utilises the returned DataFrame to compute the Price Range Of Change (PROC) and returns a new DataFrame with features (columns) = {RSI, K%, D%, R%, PROC}.
5.4) System utilises the returned DataFrame to compute moving average convergence divergence ratios and returns a new DataFrame with features (columns) = {RSI, K%, D%, R%, PROC, MACD, MACD_Diff, MACD_Signal}.
5.5) System utilises the returned DataFrame to compute the On Balance Volume (OBV) and returns a new DataFrame with features (columns) = {RSI, K%, D%, R%, PROC, MACD, MACD_Diff, MACD_Signal, OBV}.

5.6) System removes NaN values present in the DataFrame.
6) System generates the prediction column and labels past data points according to the evolution of the next actual closing price: 0 if the next closing price was lower and 1 if it was higher. The system also removes unnecessary features from the feature set, specifically {High, Low, Close, Volume}.
7) System generates the prediction:
7.1) System performs a time series cross-validation with 10 splits to identify the best ratio for dividing the dataset into training and validation sets.
7.2) System identifies the best fold and ratio between training and validation sets according to the fold that minimises the mean squared error (MSE).
7.3) System performs a grid search to find the optimal parameters for the Random Forest algorithm to improve the model's prediction accuracy.
7.4) System performs the Random Forest algorithm using the optimal fold and parameters identified to generate a prediction in binary format (0 = Bearish price trend / 1 = Bullish price trend).
7.5) System returns the predicted outcome to the frontend to display either a green upward arrow if a higher closing price is expected or a red downward arrow if a lower closing price is expected.

4.2.8 Update Users details (Only Username):

Principal actor: Admin
Precondition: Admin is logged in and is in the Manage page
1) Admin clicks on the edit icon next to the user displayed in the table and enters the email, new username, and role = 'user'.
2) Request is sent from the frontend to the respective backend API endpoint URL:PUT:HTTP//localhost:8080/users/update-users-details with as payload: {user_id: int, new_username: str and new_user_role: str}
3) System attempts to fetch a user account from the database based on its ID.
4) If the user ID is valid and a user is fetched, the system then checks which fields differ from those of the user (in this case, only the username is different).

5) System updates the username of the user in the database with the new one provided and commits the changes.

4.2.9 Update Users details (Only Role):

Principal actor: Admin

Precondition: Admin is logged in and is in the Manage page

1) Admin clicks on the edit icon next to the user displayed in the table and enters the email, new username, and role = 'user'.

2) Request is sent from the frontend to the respective backend API endpoint URL:PUT:HTTP//localhost:8080/users/update-users-details with as payload: {user_id: int, new_username: str and new_user_role: str}

3) System attempts to fetch a user account from the database based on its ID.

4) If the user ID is valid and a user is fetched, the system then checks which fields differ from those of the user (in this case, only the role is different).

5) System updates the role of the user in the database with the new one provided and commits the changes.

4.2.10 Update Users details (new role and new username):

Principal actor: Admin

Precondition: Admin is logged in and is in the Manage page

1) Admin clicks on the edit icon next to the user displayed in the table and enters the email, new username, and role = 'user'.

2) Request is sent from the frontend to the respective backend API endpoint URL:PUT:HTTP//localhost:8080/users/update-users-details with as payload: {user_id: int, new_username: str and new_user_role: str}

3) System assesses if the individual performing this action is legitimate by fetching the user from the JWT token to determine if they are valid and if their role is set to admin.

4) If the user performing the editing action is legitimate, the system attempts to fetch a user account from the database based on their ID.
5) If the user ID is valid and a user is fetched, the system then checks which fields differ from those of the existing user (in this case, both username and role are different).
6) System updates the username and role of the user in the database with the new ones provided and commits the changes.

4.2.11 Block User's account:

Principal actor: Admin
Precondition: Admin is logged in and is in the Manage page
1) Admin clicks on the closed lock icon next to the user displayed in the table.
2) Request is sent from the frontend to the respective backend API endpoint URL:PUT://localhost:8080/users/block_user_account with as payload:
{user_id: int).
3) System assesses if the individual performing this action is legitimate by fetching the user from the JWT token to determine if they are valid and if their role is set to admin.
4) System using the given ID, retrieves the user from the database, updates their is_active field from True to False, and commits the change to the database.

4.2.11 Unblock User's account:

Principal actor: Admin
Precondition: Admin is logged in and is in the Manage page
1) Admin clicks on the open lock icon next to the user displayed in the table.
2) Request is sent from the frontend to the respective backend API endpoint URL:PUT://localhost:8080/users/unblock_user_account with as payload:

{user_id: int).
3) System assesses if the individual performing this action is legitimate by fetching the user from the JWT token to determine if they are valid and if their role is set to admin.
4) System using the given ID, retrieves the user from the database, updates their is_active field from False to True, and commits the change to the database.

4.2.12 Fetch training data:

Principal actor: Admin
Precondition: Admin is logged in and is in the Manage page
1) Admin clicks on the "Gather Data" button.
2) request is sent from the frontend to the respective backend API endpoint URL:GET://localhost:8080/data/fethch&store_financial_data
3) System assesses if the individual performing this action is legitimate by fetching the user from the JWT token to determine if they are valid and if their role is set to admin.
4) System fetches financial data for all stocks listed in the S&P 500 for the past 5 years from the Yahoo Finance API and stores it in a DataFrame.
5) System stores the training dataset into a CSV file.
6) System writes the latest date's worth of financial data gathered into a text file.

4.2.13 Fetch training data:

Principal actor: Admin
Precondition: Admin is logged in and is in the Manage page
1) Admin clicks on the "Update Database" button.
2) request is sent from the frontend to the respective backend API endpoint URL GET://localhost:8080/data/get_uptodate-data”

3) System assesses if the individual performing this action is legitimate by fetching the user from the JWT token to determine if they are valid and if their role is set to admin.

4) System gets the current date when the action is performed as the end date for data retrieval.

5) System opens and reads the record to get the date in the last line that indicates when financial data is stored in the CSV file.

6) System fetches missing data from the Yahoo Finance API with the following arguments: {start = end_date_retrieved_from_record, end = today_date} and stores it in a DataFrame.

7) System adds the newly gathered financial data to the CSV file.

8) System sorts the CSV training dataset to ensure all stocks are grouped together according to increasing date and ticker symbols.

9) System writes the date when the CSV file was updated into the record text file.

4.3 Sequence Diagram:

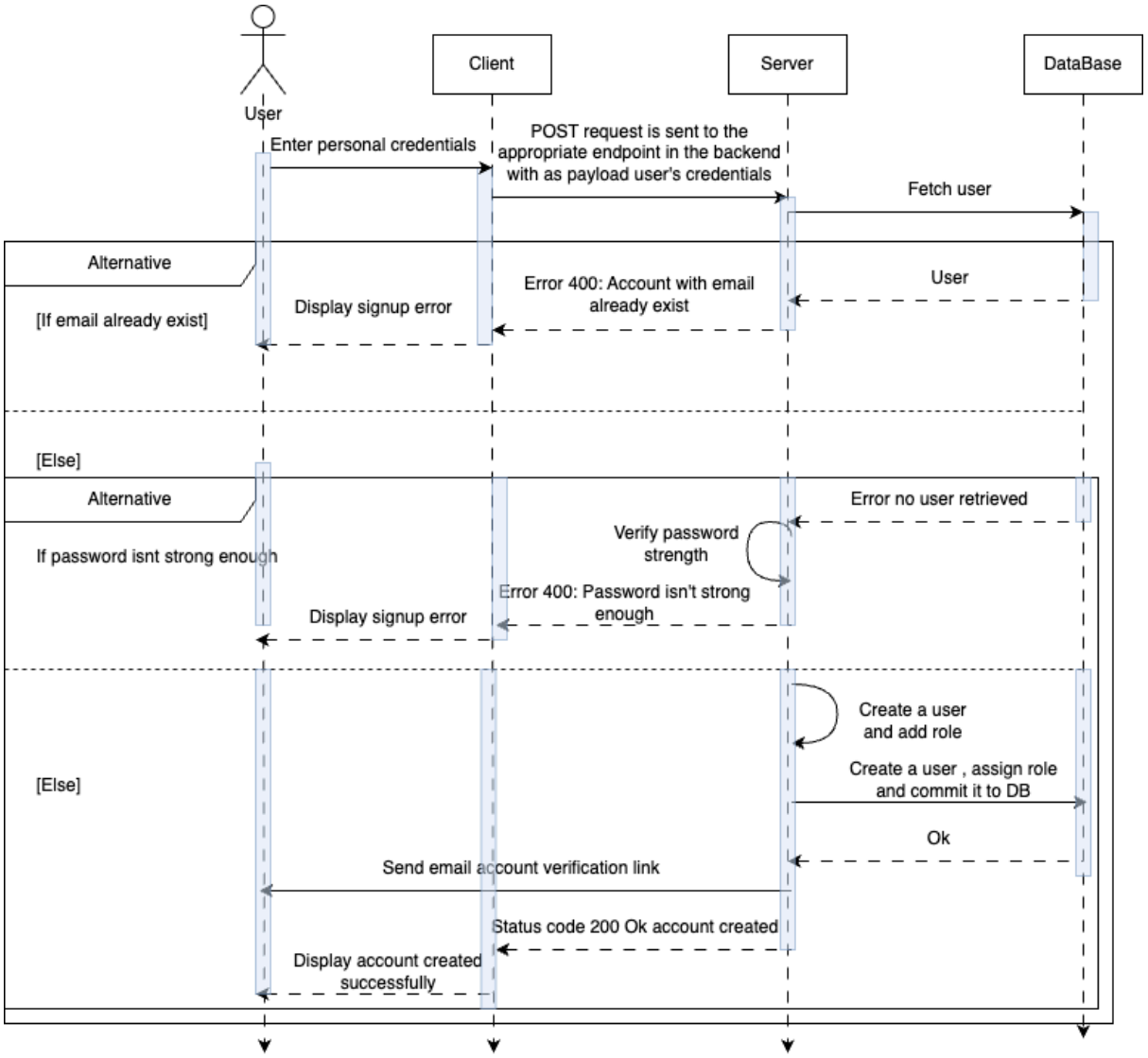


Figure 4.3.1: Sequence diagram - Create a user account

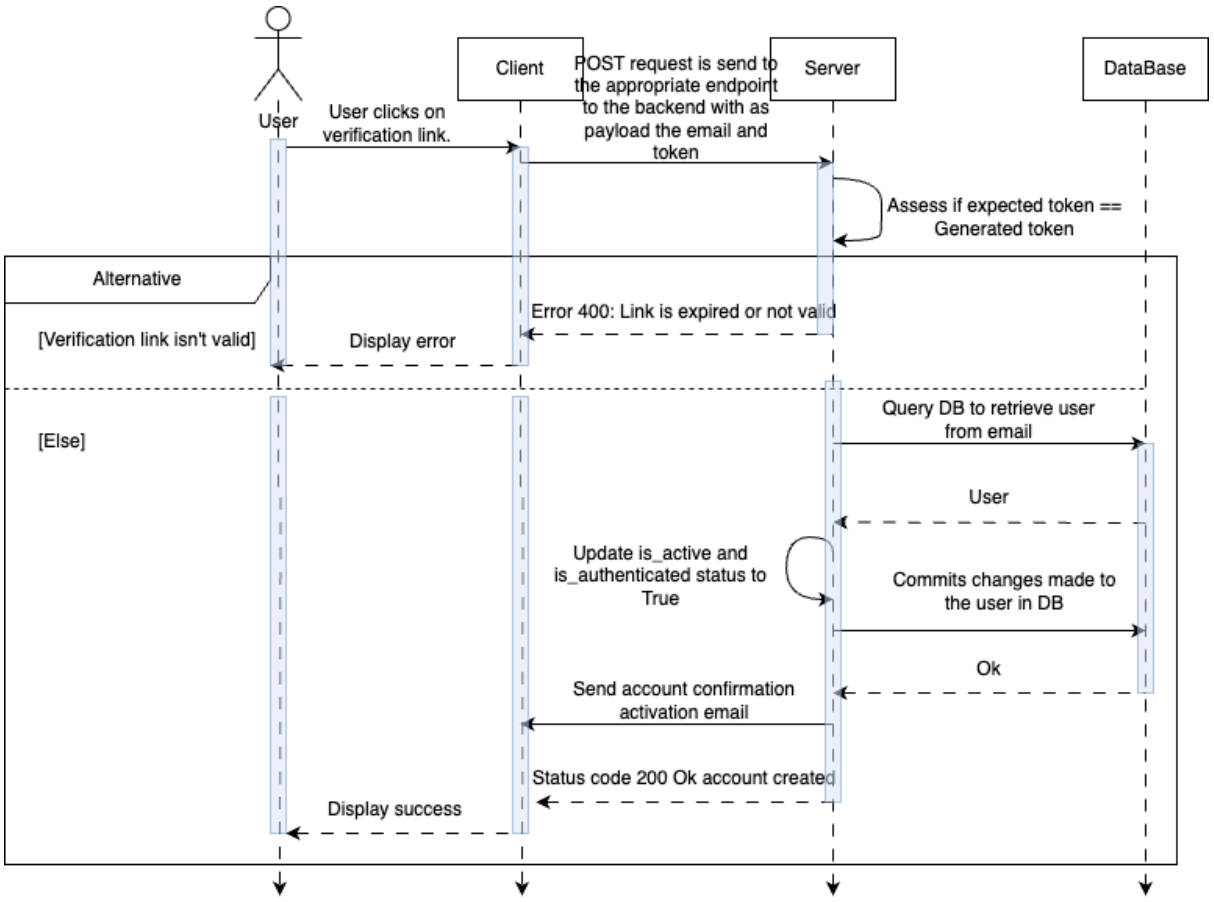


Figure 4.3.2: Sequence diagram - User activate his account

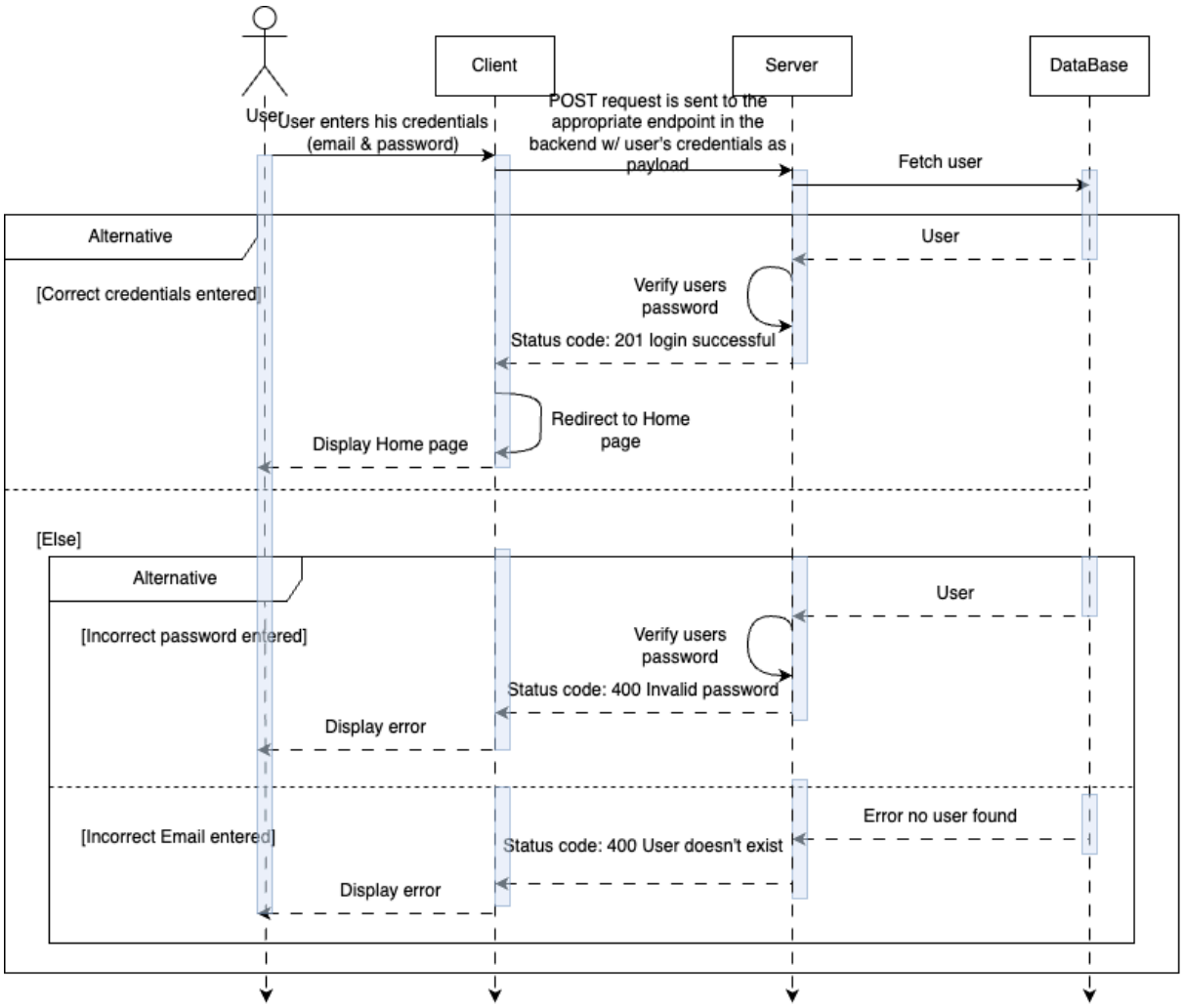


Figure 4.3.3: Sequence diagram - User logs in

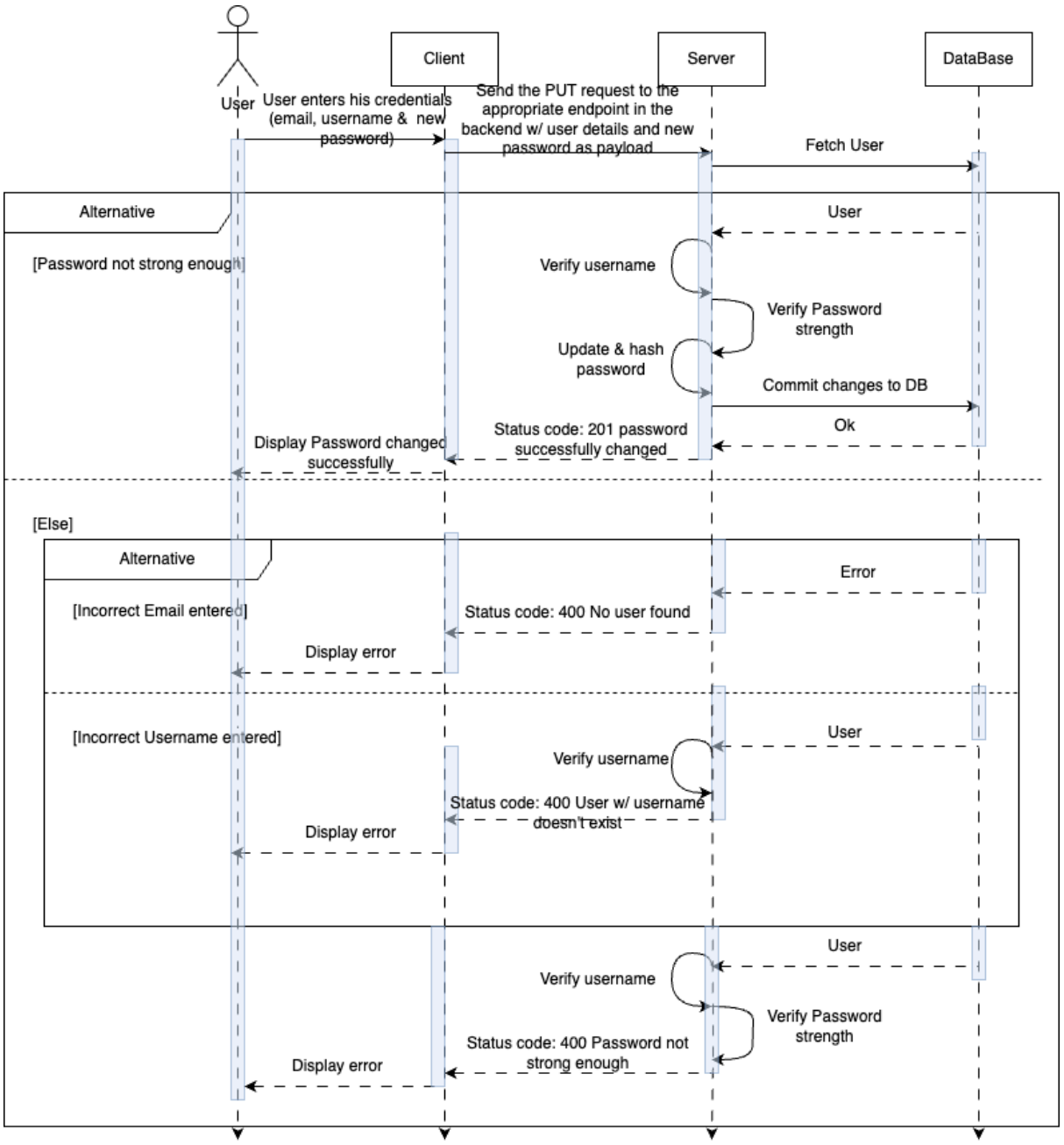


Figure 4.3.4: Sequence diagram - User resets password

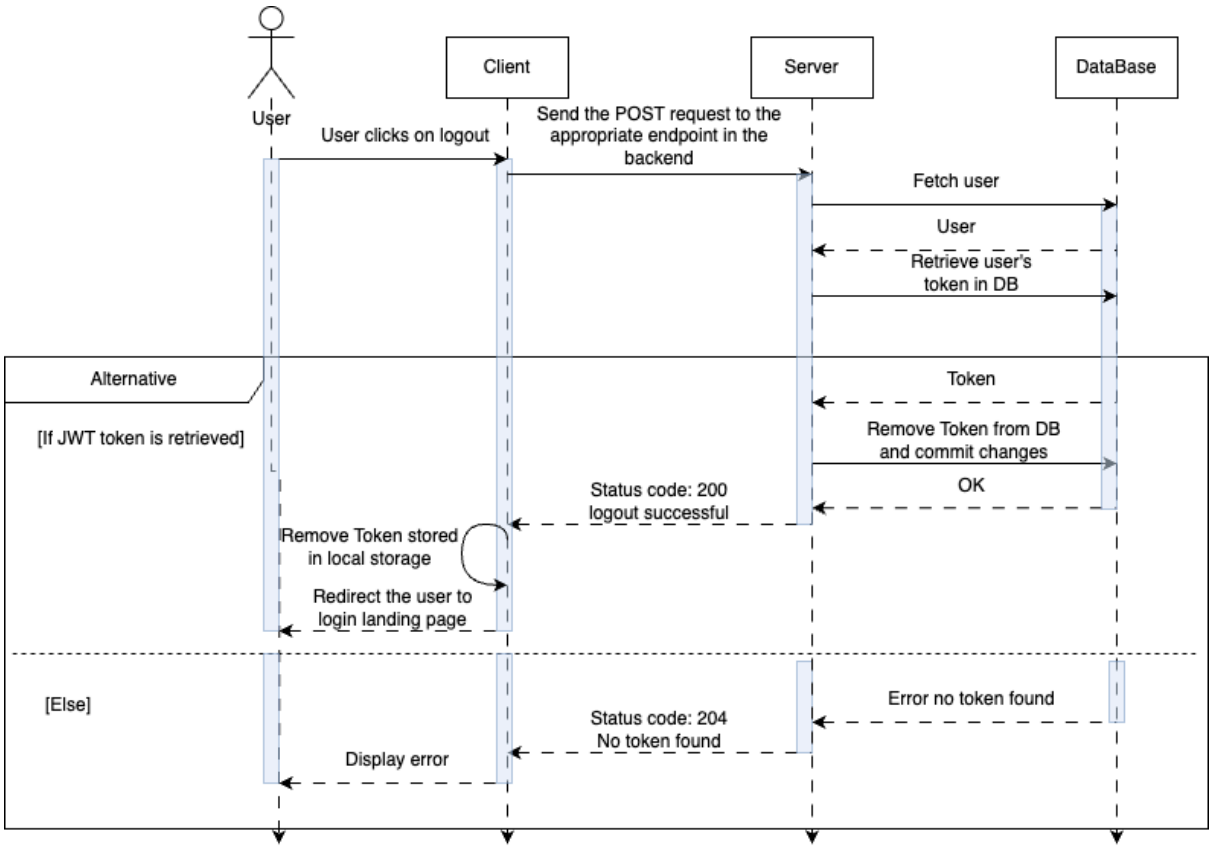


Figure 4.3.5: Sequence diagram - User logs out

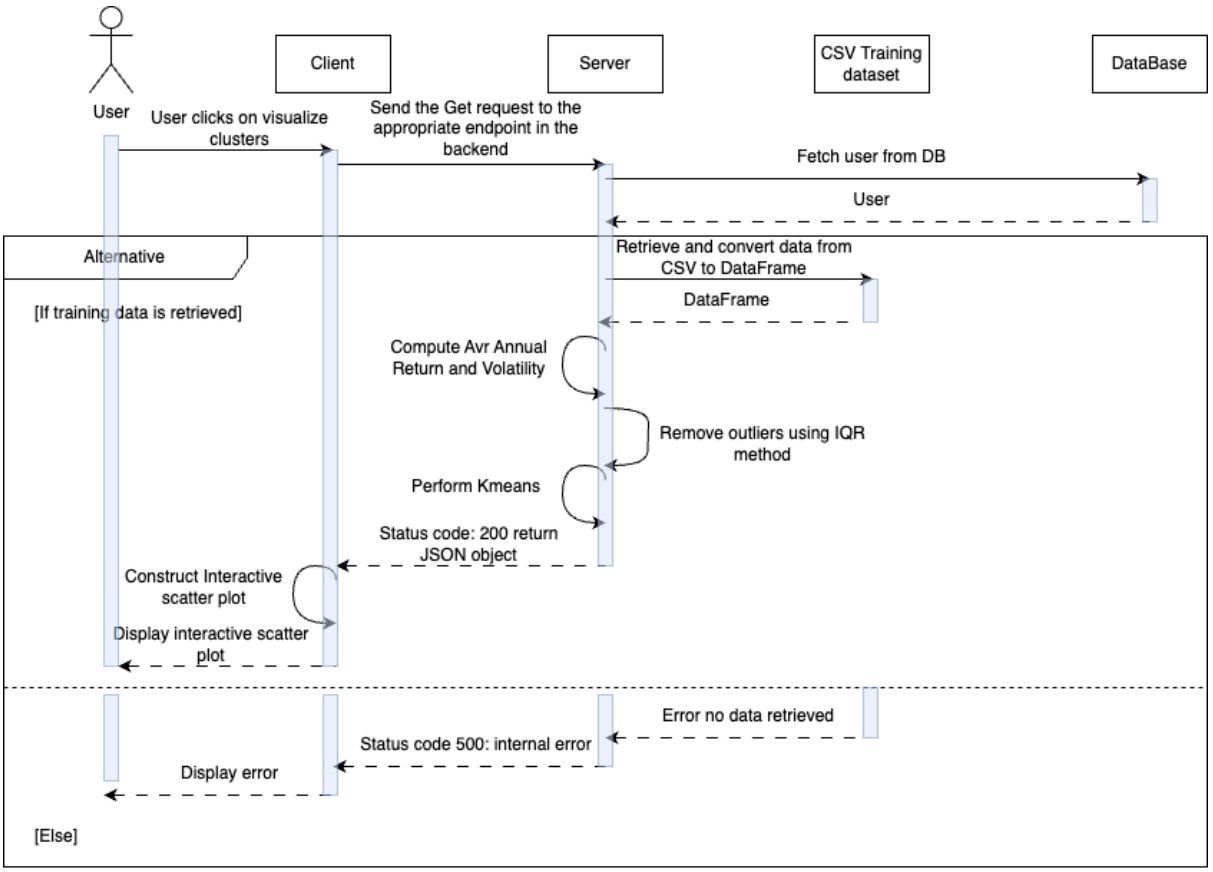


Figure 4.3.6: Visualise stocks clusters:

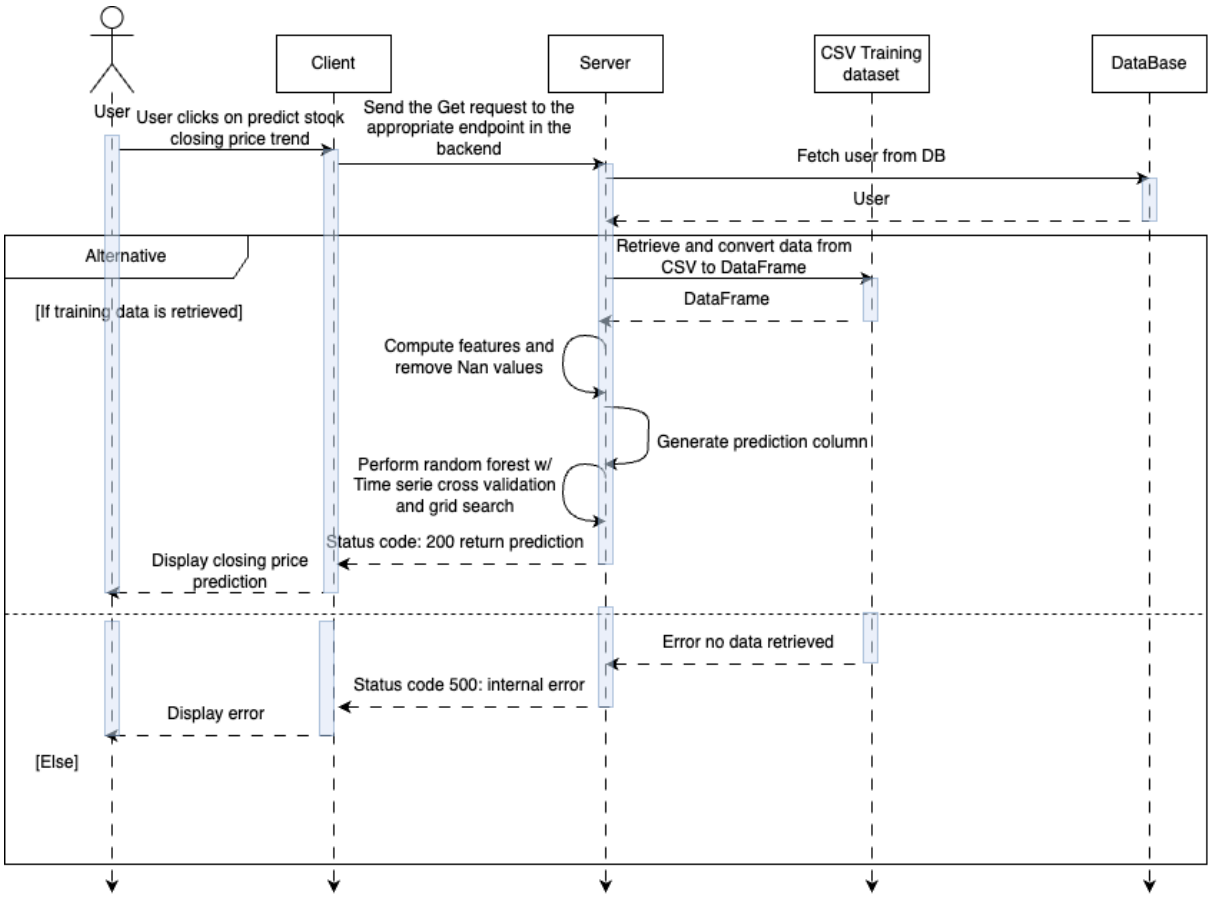


Figure 4.3.7: Forecast next closing price trend

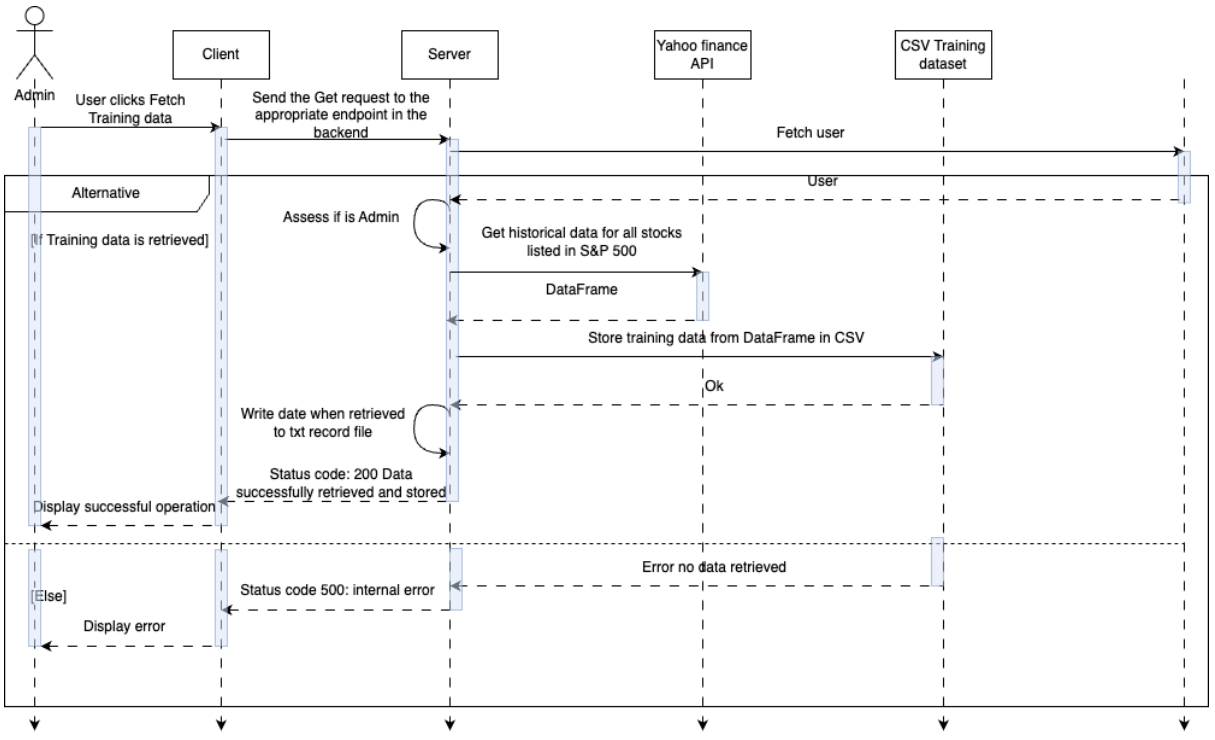


Figure 4.3.7: Fetch and store training data set

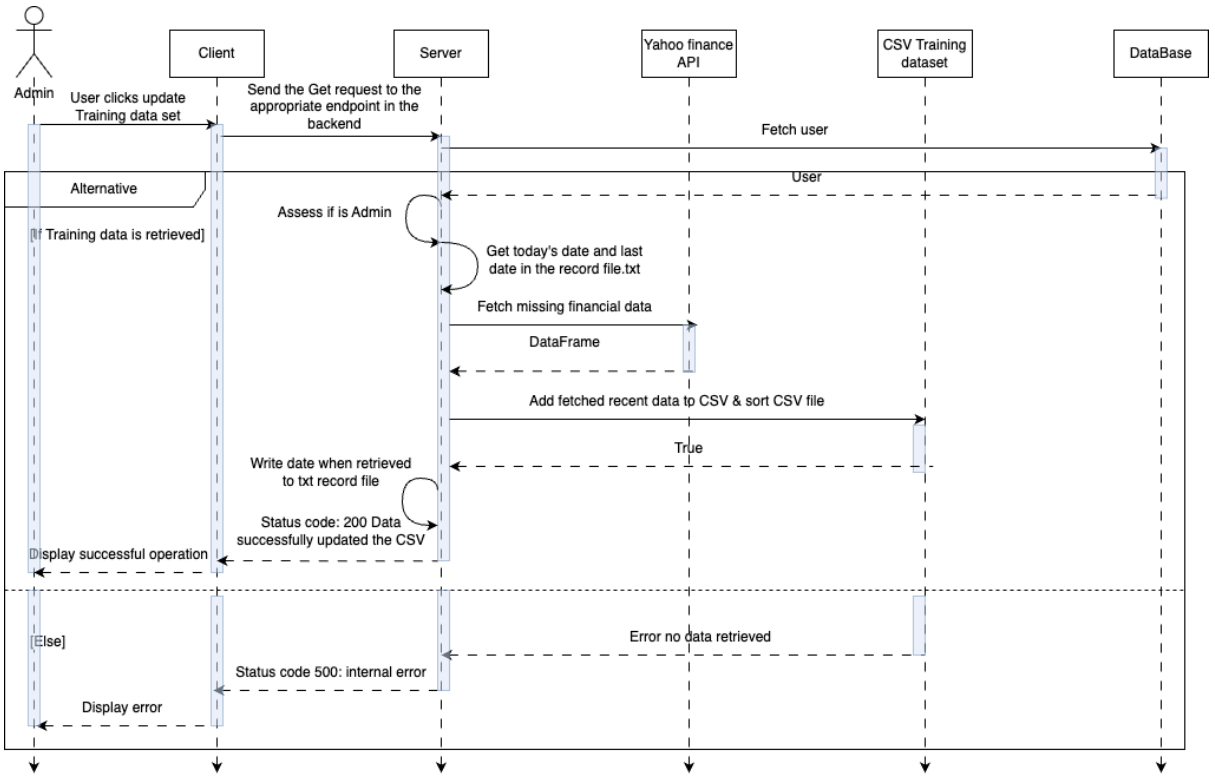


Figure 4.3.8: Update the training data set

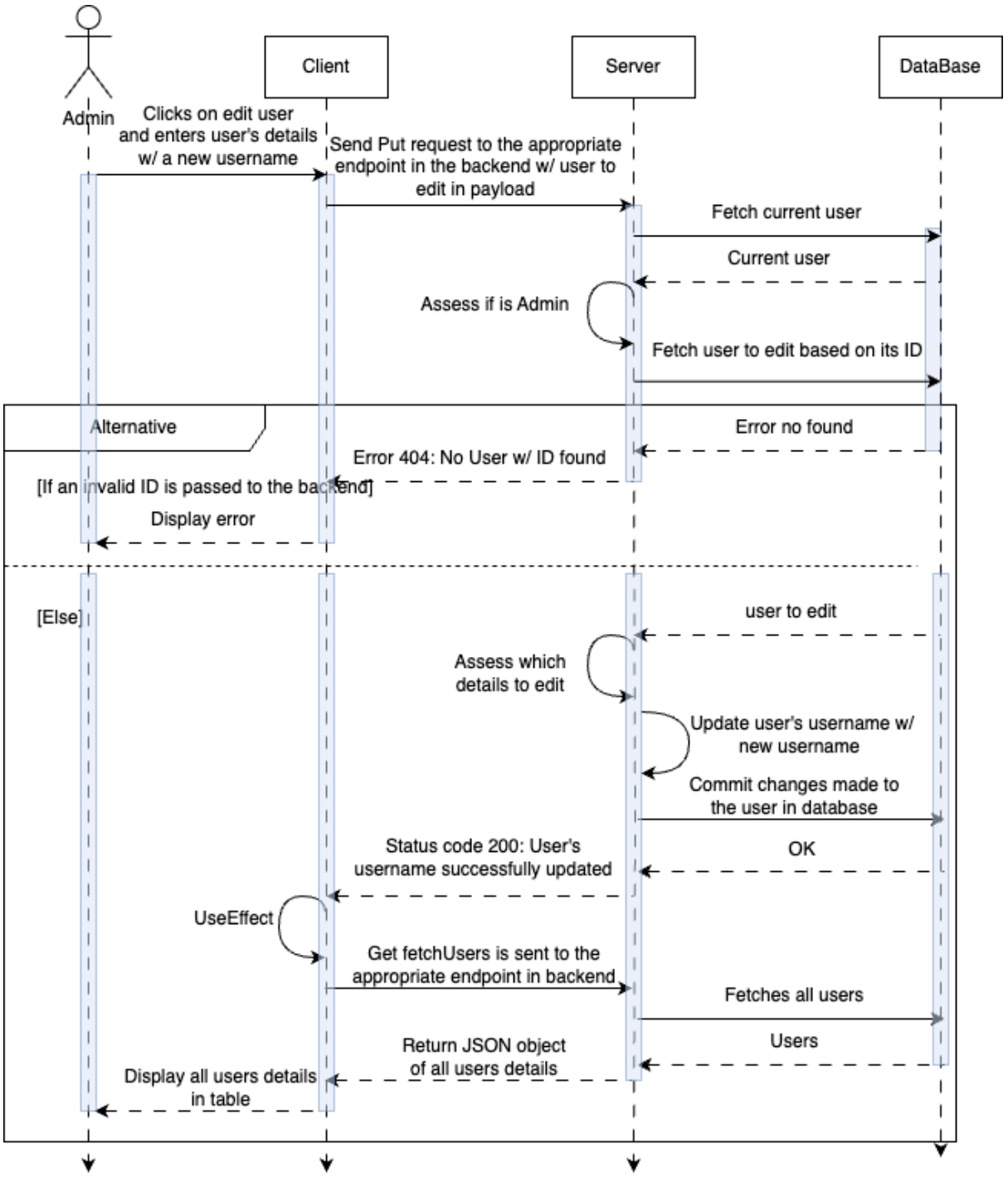


Figure 4.3.9: Update user's username:

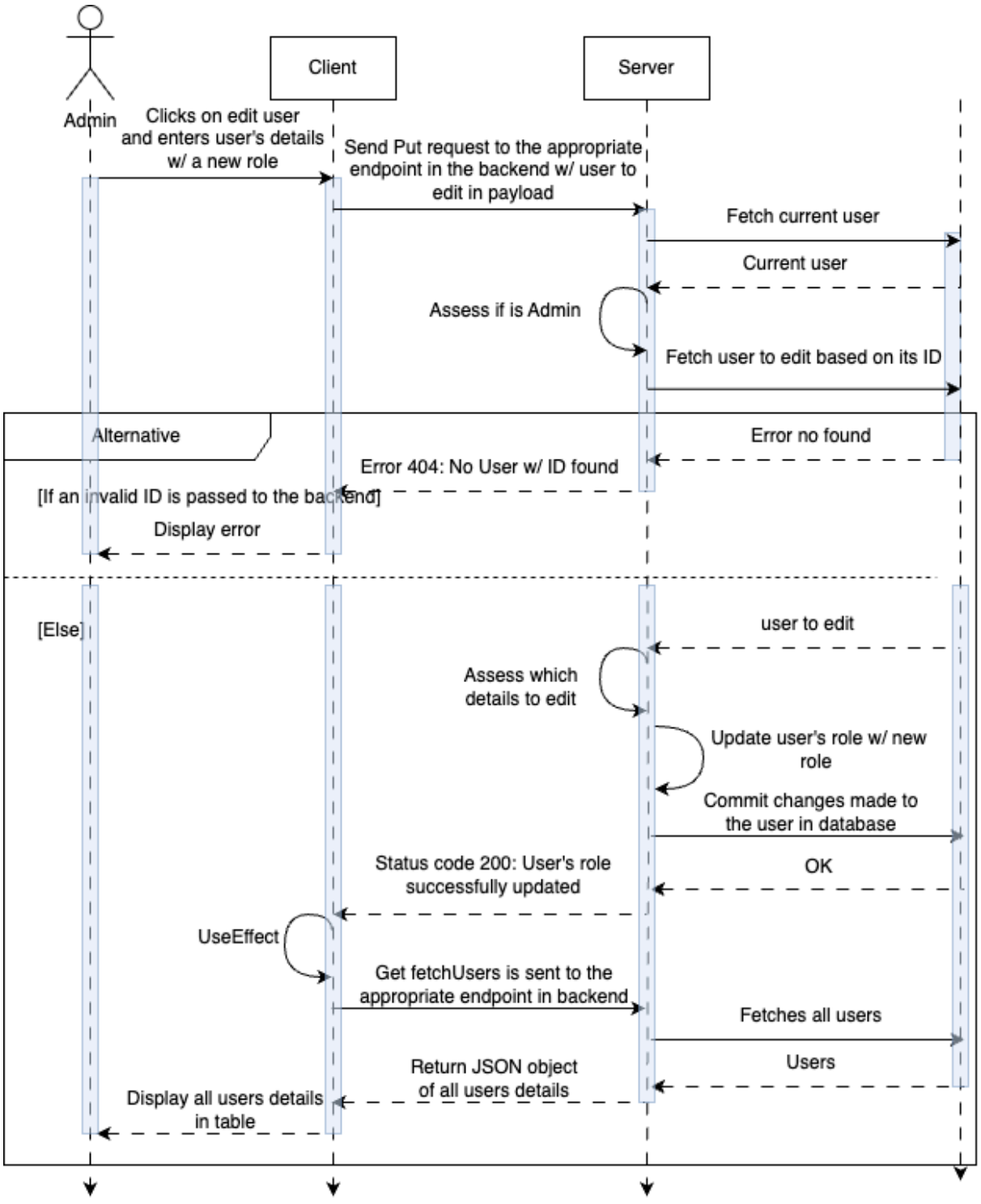


Figure 4.3.10: Update user's role:

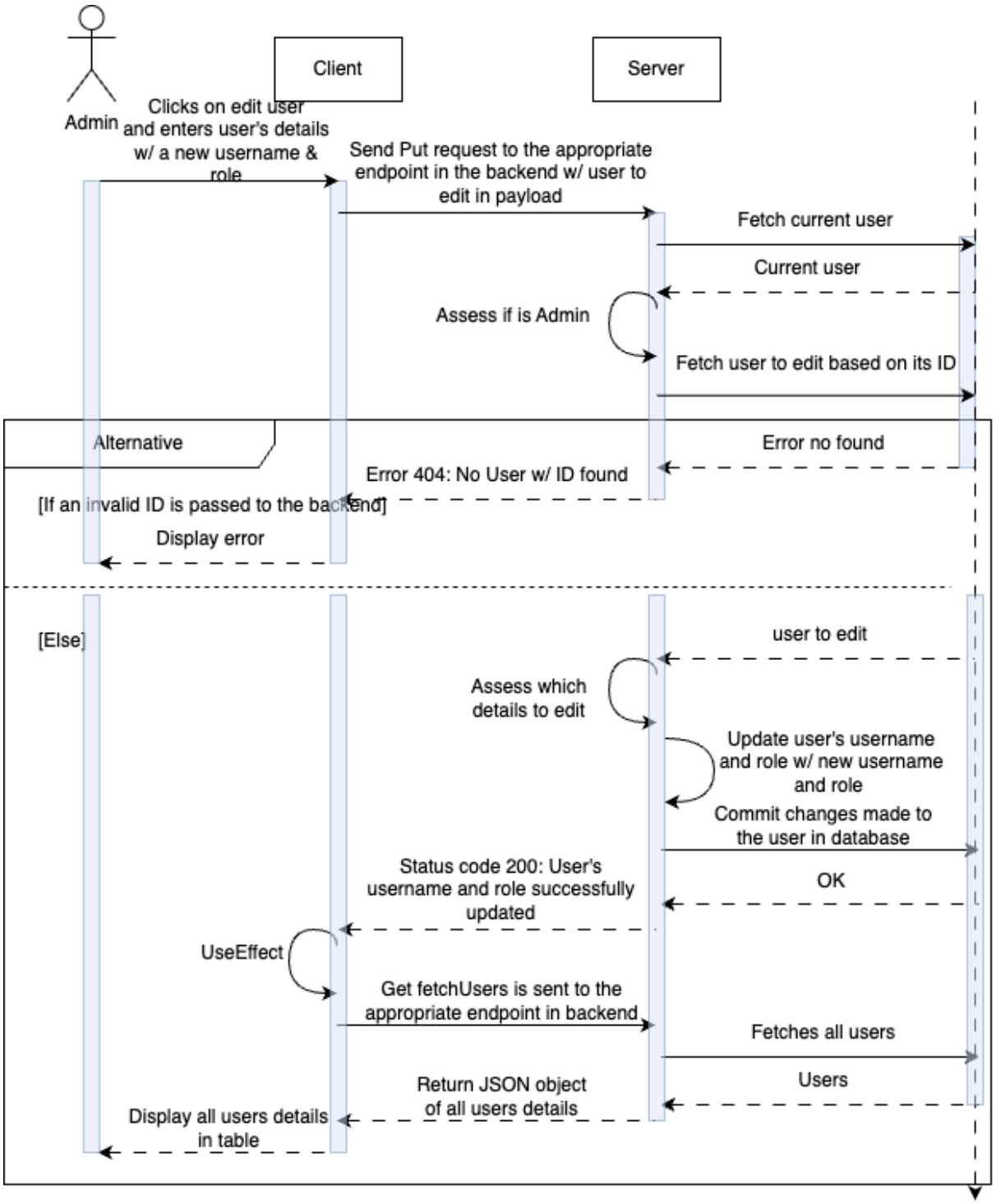


Figure 4.3.11: Update user's username and role:

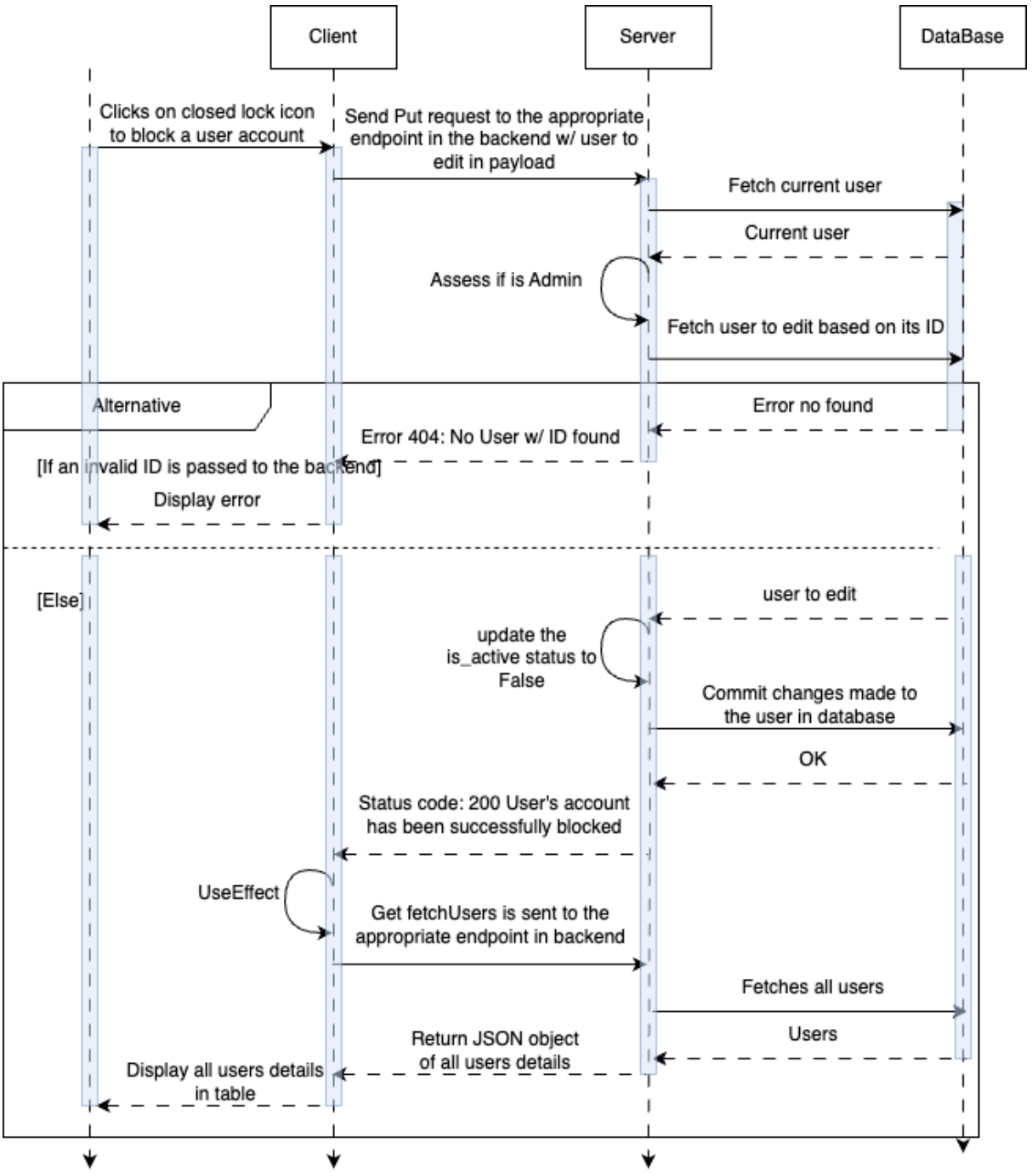


Figure 4.3.12: Block a user account:

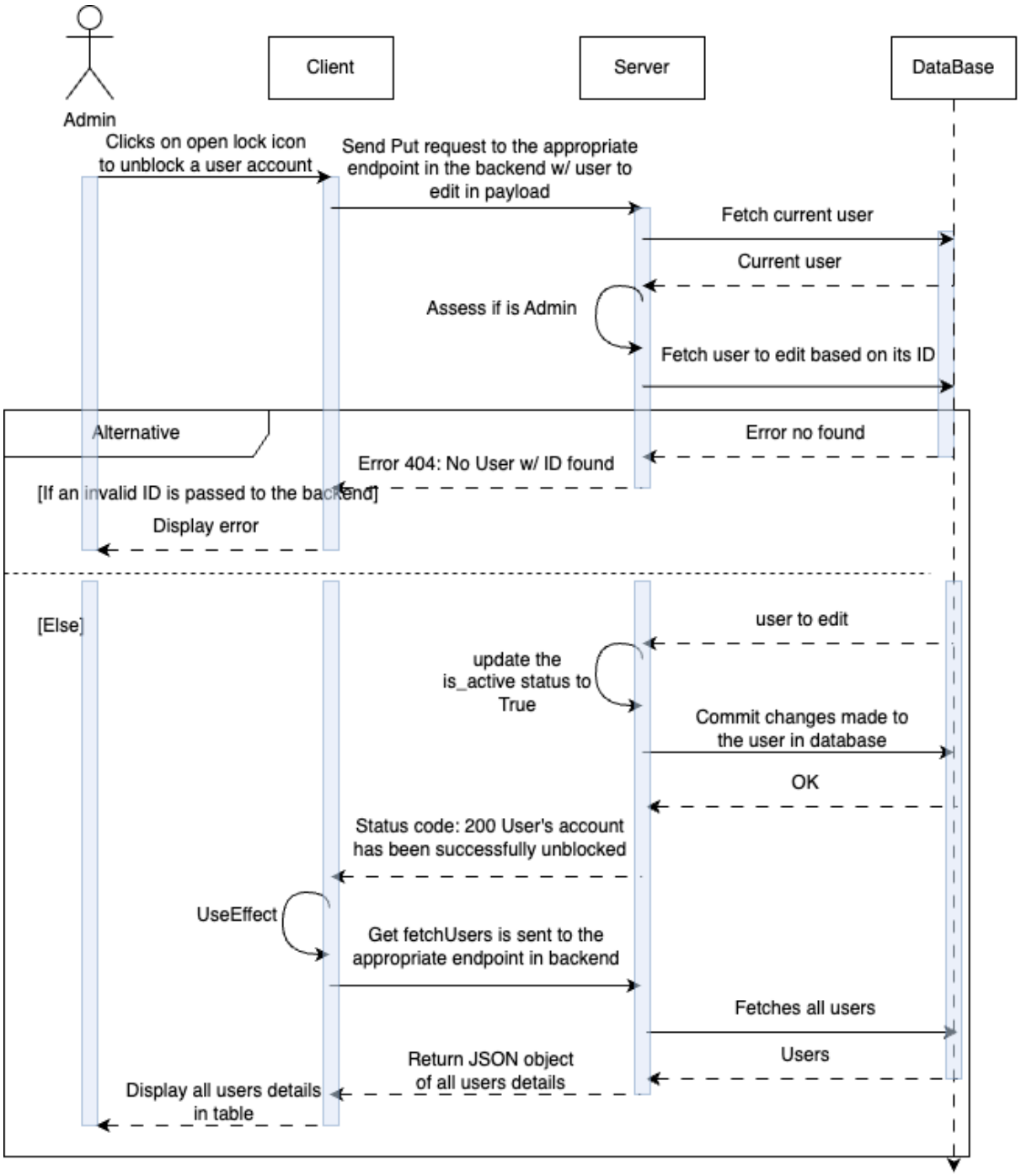


Figure 4.3.13: Unblock a user account:

4.4 Entity-Relational Diagram:

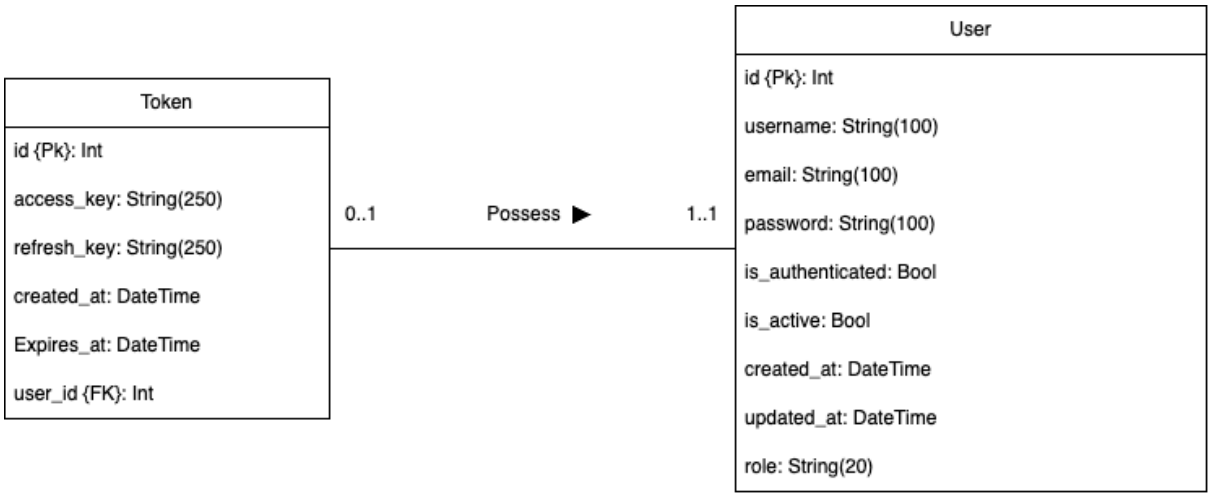


Figure 4.4.1: Entity-Relational Diagram

The entity-relationship diagram illustrates the type of relationship between models mapped as tables in the database and how their respective data is stored. For the web application, only two models were created: a user and a token. These two models are sufficient as they enable account creation, authentication, navigation, and logout from the web application. No stock models were created because they are not intended to be stored in the PostgreSQL database. Instead, they are stored in a CSV file for efficiency, avoiding additional time and steps in storing and retrieving this data from the database.

Regarding the user and token models, the nature of their relationship is 1:1, which implies that a token belongs to at most one user, and the user possesses at most one token. When a user creates an account, they submit their username, email, and password. The server then inspects the database to determine if a user with the same email already exists. If so, it returns an error, and the user cannot create an account. Otherwise, if the database doesn’t find any user with the same email, the server proceeds to assess the password strength. For a password to be considered strong, it must be longer than eight characters and include at least one uppercase letter, one lowercase

letter, a digit, and a special character. The function returns a boolean value of true if the password is valid.

After the password check, the server instantiates a new user and maps the entered credentials to the newly created object. The server then calls the hash password function to securely encrypt the password before storing it in the database. Simultaneously, the ‘is_active’ and ‘is_authenticated’ statuses of the user are set to false. A function is then called to add the default role of ‘user’ to prevent the newly created user from accessing critical functionalities in the web application related to account management and maintaining the training dataset. The server commits these changes to the database, and the user account is created but remains inactive.

After the commit, the server sends an email verification link to the user’s email address, encoding a token in the link. This ensures that the user can only activate their account by clicking on the link. When they do so, the user is retrieved from the database, and the server checks if the token generated during the verification process matches the one attached to the URL. If it does, the ‘is_active’ and ‘is_authenticated’ statuses are updated to true.

Once the account is activated, the user can log in to the web application. During this process, they must enter their email and password. When the user clicks submit, the server retrieves the user from the database based on the provided email, verifies the password by comparing it to the encrypted version stored in the database, and then checks both the ‘is_active’ and ‘is_authenticated’ statuses to ensure that only valid accounts can access the web application. If both fields are true, the user is authenticated and granted access. Upon successful login, the ‘generate_token’ function in the ‘TokenController.py’ generates the payload of the JWT and constructs a secured JWT token. A private route component is implemented in the frontend to assess if the user has a JWT token. When the user logs in, the JWT token is returned, and the private page component checks its validity before adding

it as a header in local storage to ensure the user is legitimate to access private pages of the web application.

The user possesses a ‘role’ field, which is crucial for enforcing role-based access control. In the web application’s navigation bar, there is a section called "Accounts" that only an admin can see and navigate to. During the login process, when the JWT token is validated and stored in local storage, a GET request is sent to the backend endpoint relevant to roles to retrieve the ‘role’ field stored in the User table in the database. The returned JSON object is then assessed to determine if it is equal to ‘Admin’. If it is, the boolean value ‘True’ is set to the ‘isAdmin’ global state stored in the React Redux store, ensuring that the user will be able to see and access the account page.

Finally, concerning the logout process, when the user clicks on the logout option in the navigation menu, a POST request is sent to the appropriate endpoint in the backend, where the server retrieves the user according to their access key stored in local storage. The server then queries the database to remove the token associated with the user. Afterward, local storage is cleared, and since all pages are private and require a valid access key in local storage, the user is redirected to the default sign-in landing page.

4.7 User Interface Design:

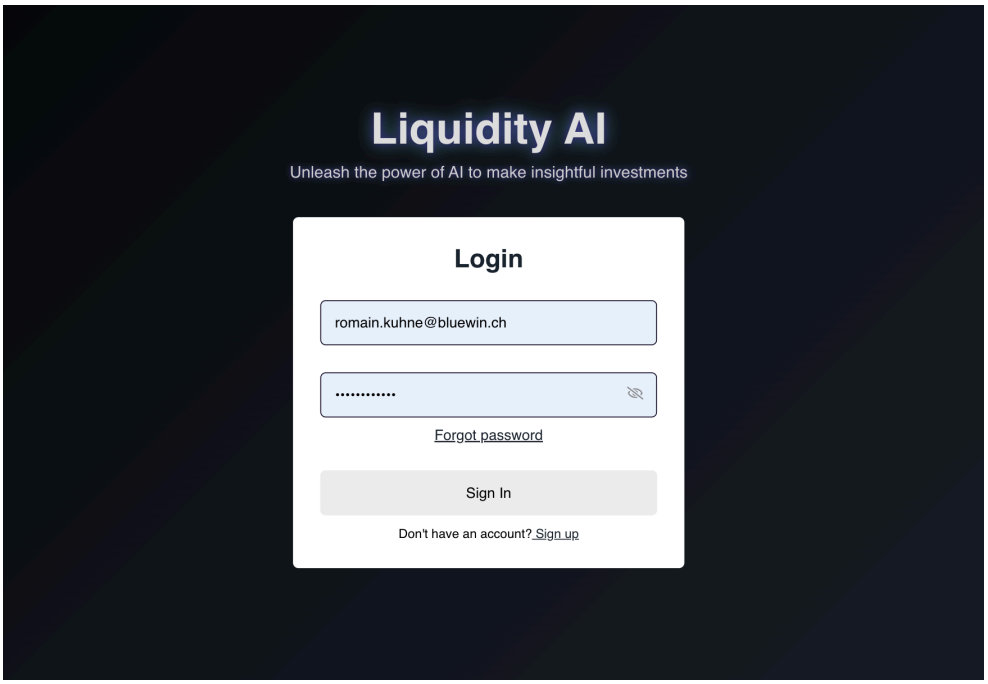


Figure 4.7.1: Landing page

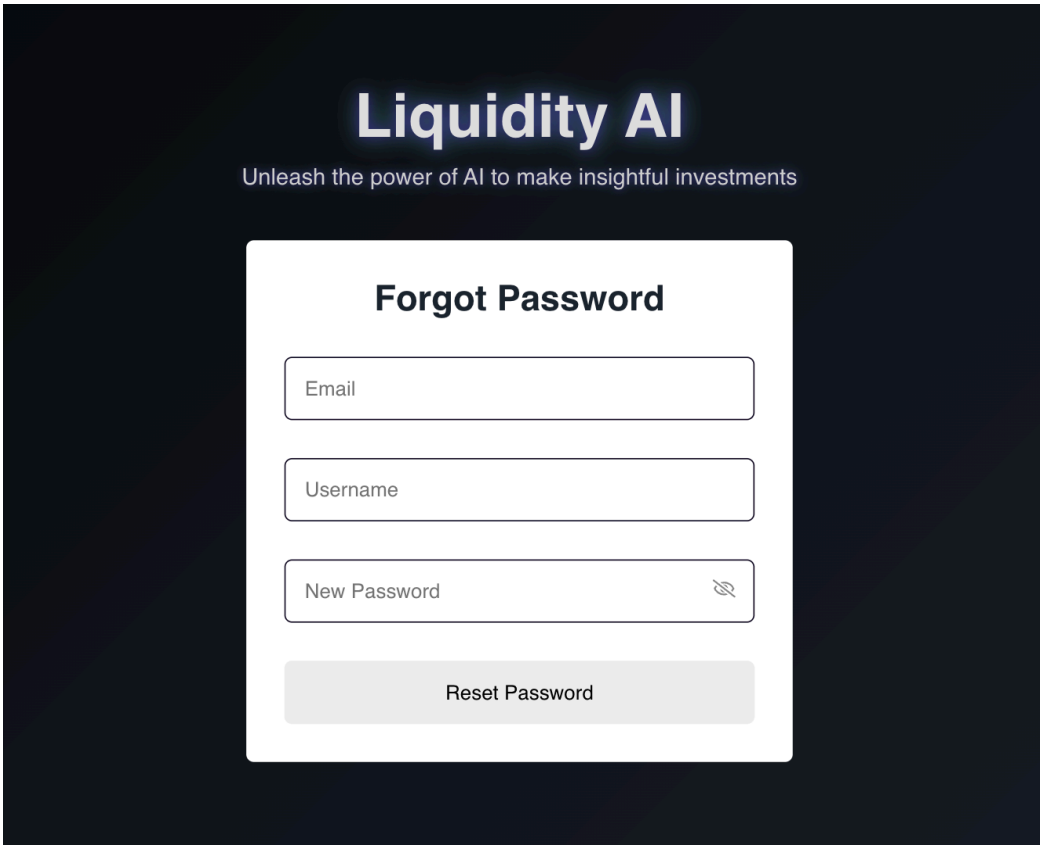


Figure 4.7.2: Reset password page

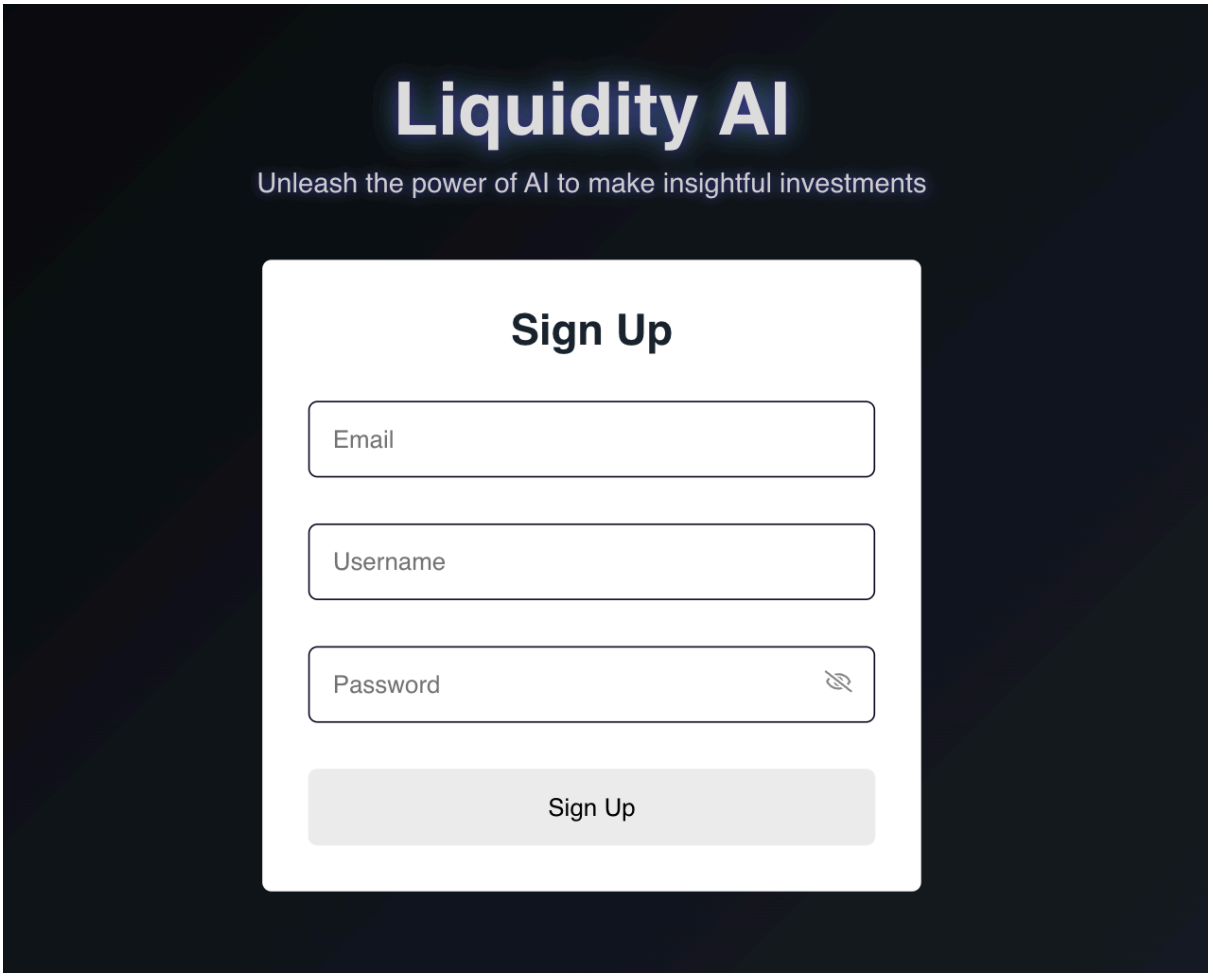


Figure 4.7.3: Signup page

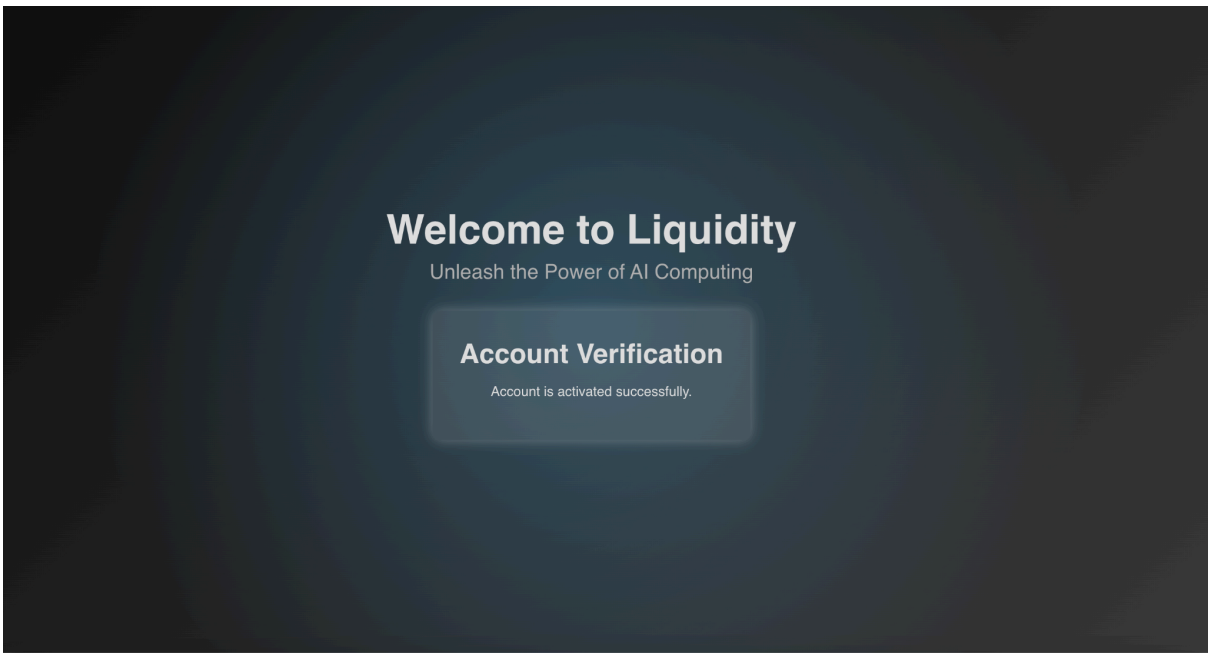


Figure 4.7.4: Account activation page

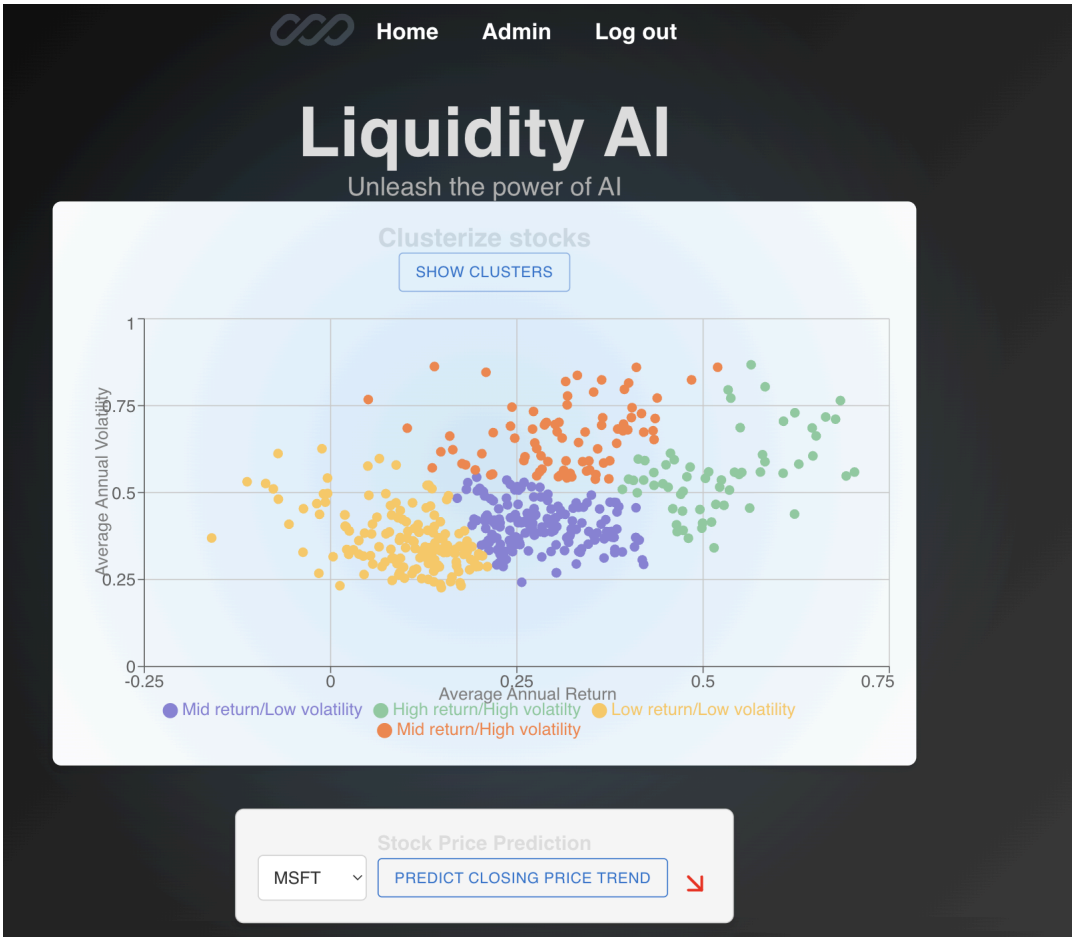


Figure 4.7.5: Home page

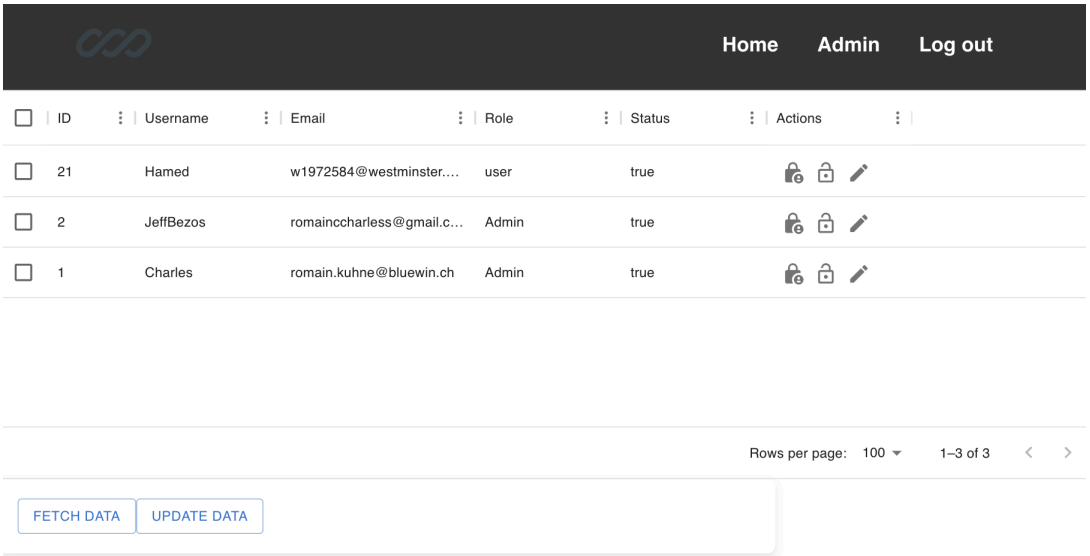


Figure 4.7.6: Admin page

Chapter 5 Implementation

5.1 Introduction:

In this section, the implementation of the web application will be discussed, starting from the mathematical intuition behind the machine learning models used, through their implementation, and finally to the features of each application component, from the server side to the client side. Before detailing the implementation, a system architecture diagram will be constructed to illustrate the key components and their interactions. Following this, a review of the technologies used in the software will be compiled to provide detailed insights into the tools that enable the application to function effectively. A subsection on the mathematical intuition behind both models will then be presented to enable the reader to grasp a fundamental understanding of how the models process data and generalise. After explaining the functioning of both models, their implementation within the application will be detailed. A class diagram will then be presented to illustrate the components used on the server side. Finally, the client-side components and their interactions will be explained, providing information on how the backend connects to the frontend.

5.2 System Architecture Diagram:

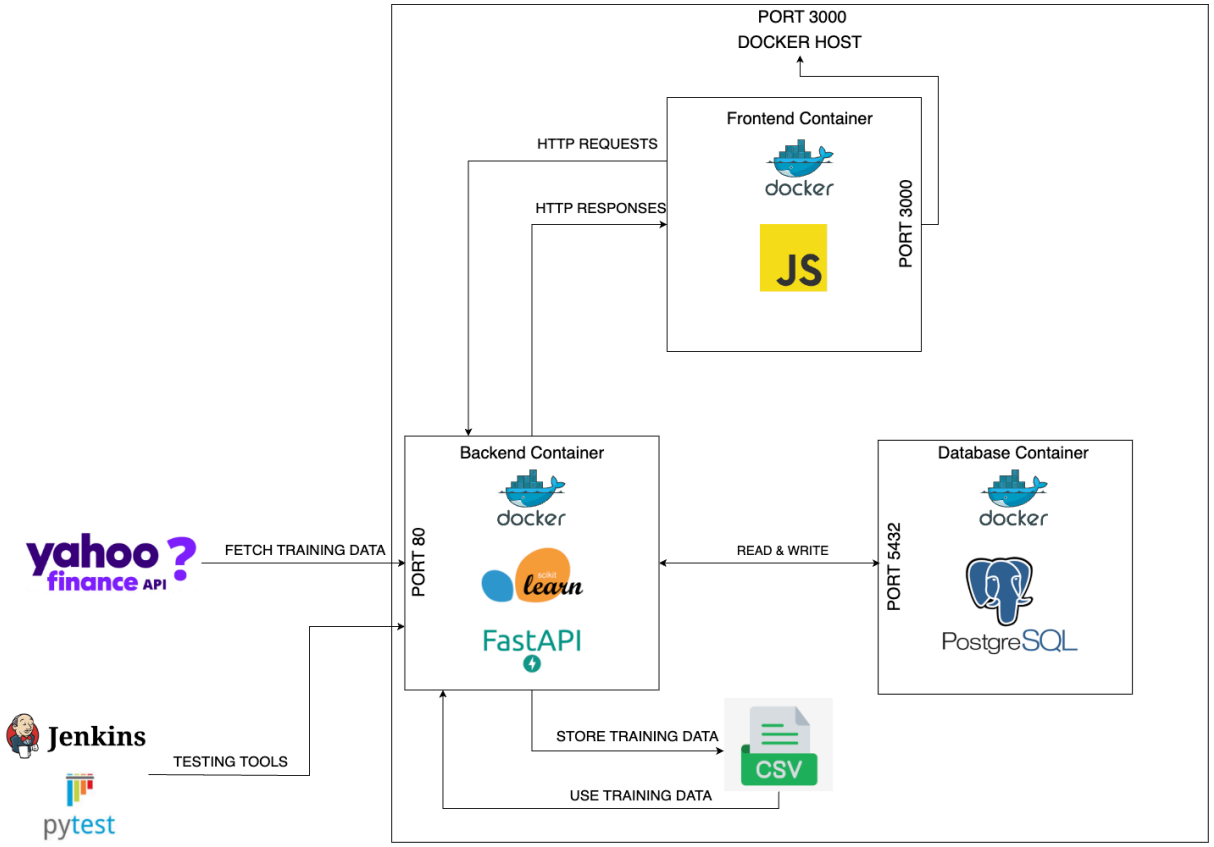


Figure 5.2.1: System Architecture Diagram of the web application

The system architecture of the developed machine learning web application is illustrated above. To ensure the software can run on any type of machine and to divide the application into microservices, Docker, a containerization tool, was used to separate each component, respectively the backend, frontend, and database into distinct containers, each with its own environment and settings.

Regarding the database, a PostgreSQL image is running within the container. To ensure that the data stored within the container persists, a Docker volume was created to safeguard user information and JWT tokens. The database is connected to the server container via port 5432, a crucial connection that

enables the server to perform CRUD operations on the tables within the database.

The backend container houses all the necessary dependencies to maintain the integrity of the web application, particularly concerning the signup process, JWT authentication, role-based access control, storage of training data in CSV files, as well as their retrieval and processing by the K-means and Random Forest machine learning models. The application's logic was implemented using Python's FastAPI library, coupled with an MVC design pattern to ensure separation of concerns and low coupling between components. On the server side, only two models were implemented, respectively 'UserModel' and 'TokenMode'l, as they play an essential role in authentication and access to the platform. Both models are stored in the PostgreSQL database using volumes. Initially, a 'StockMode'l was designed to store the financial data used by both machine learning models in the PostgreSQL database. However, since this entity isn't related to another one and storing this data adds overhead, an alternative strategy was employed. The financial training data for each stock in the S&P 500 are fetched and stored in a CSV file in an ordered manner according to their date and ticker symbol using the Yahoo Finance API. The retrieval and processing of this data are done using the Pandas library via DataFrames. The models were implemented and fine-tuned using the SciKit-Learn library.

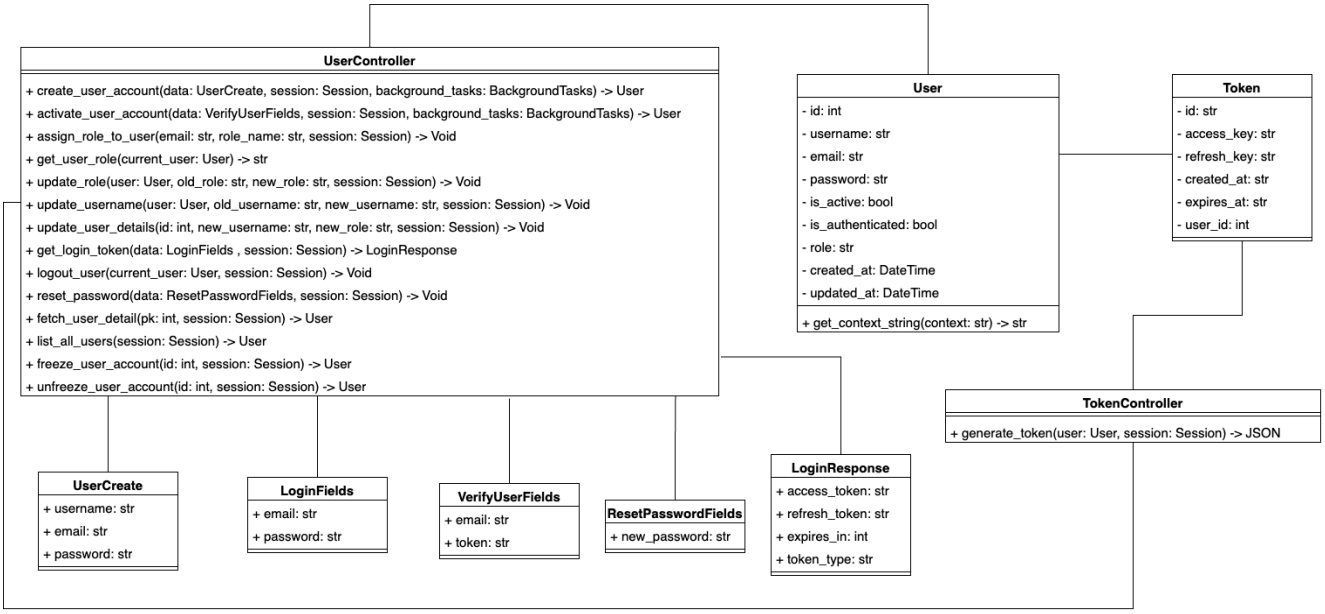
For the frontend container, a JavaScript image was used to develop the logic related to the application's responsiveness and to ensure connectivity between both containers. This guarantees that requests are sent to the server side and can be received by the backend.

Finally, regarding the testing tools used to evaluate the robustness of the software, the Pytest-asyncio library was employed to develop asynchronous test cases for both unit and integration tests. Jenkins was used to automate

unit and integration tests by specifying crucial steps to be undertaken through a pipeline shell script.

5.3 Implementation Class Diagram:

Figure 5.3.1: Implementation class diagram



The class diagram above illustrates the models implemented on the server side of the web application. As mentioned in the system architecture diagram, the platform was developed using the MVC design pattern. The models consist of a ‘User’ and a ‘Token’ to define the data associated with these objects, and the controllers include ‘UserController’ and ‘TokenController’, which are responsible for handling secure JWT authentication, role-based access, and managing user accounts.

Regarding token creation, a token is generated, assigned to the user, and stored in the respective table in the database when they successfully log in by providing their credentials to the ‘get_login_token’ function. During this process, the access token is attached to the header and stored in local storage. The token table in the database is not intended to permanently store each user’s token every time they log in. When the user logs out of their

account, the ‘logout_user’ function retrieves the token associated with the user and removes it from the table.

Concerning user details, they are meant to be permanently stored in the User table in the database. An account is created during the sign-up process by providing an email, username, and password as defined in the ‘UserCreate’ schema. During account creation, when a user is created, the ‘assign_role_to_user’ function ensures that each newly created user is assigned the role of ‘user’. This strategy is implemented to enforce role-based access, preventing users from accessing admin operations such as account management, fetching training data, and updating the database. This ensures that only the admin can perform operations related to accounts, such as blocking or unblocking a user, or updating user details and roles. After the user’s account is created, a verification link with a token and email attached is sent to the user’s email address to activate their account. When the user clicks on the link, they are redirected to an account activation page where they must press a button to send the POST request to the appropriate endpoint. The ‘activate_user_account’ function then verifies the validity of the token and email by comparing this key with a generated key. Only if the comparison matches will the user’s ‘is_active’ and ‘is_authenticated’ statuses be set to true, allowing them to log in to the platform.

Finally, regarding admin users, their task is also to continuously update the database to ensure that the training data is kept up to date, guaranteeing accuracy in predictions.

5.4 Decision Tree Classifier:

5.4.1 Intuition Behind The Random Forest:

Concerning the machine learning model used to implement the stock price trend forecasting service, a decision tree classifier was used in conjunction with the random forest algorithm. The decision tree is a supervised machine learning model that functions by recursively splitting the data into subsets according to the value of input features that characterise each data point until the leaf nodes contain only similar data points (Daume, 2023).

The process begins at the parent node, where the entire dataset is recursively split by internal nodes. Each internal node represents a decision, used to split the data based on a feature value, creating new branches. This recursive splitting continues until all data points in a node belong to the same category, at which point they reach a leaf node, representing the final prediction or label.

One of the key challenges in decision trees lies in defining the best splitting criterion that maximises information gain. To identify the most suitable splitting criterion for a feature, the concept of entropy $H(S)$ was used as a measure of the uncertainty or impurity of an outcome. Here, Sv represents the total number of data points present in a node resulting from the splitting criteria v and S represents the total number of data points in the dataset. In short, entropy measures the purity of the data in a node after applying a splitting criterion.

A higher entropy indicates greater uncertainty in the outcome, suggesting that a splitting criterion with value v does not effectively separate the classes of data, leading to a node with ambiguous or mixed classes. On the contrary, if the entropy $H(S)$ decreases, the uncertainty of the outcome is reduced. This implies that the splitting criterion has effectively separated the different classes

of data, resulting in a node with a high probability of belonging to one class and a low probability of another class.
The entropy equation is expressed as:

$$H(S) = -\sum \frac{|Sv|}{|S|} log_2\left(\frac{|Sv|}{|S|}\right)$$

The equation computes the weighted average of the logarithm of the probability of a data point belonging to a particular class after a split.

Finally, to enhance the predictive capability of the model, it was combined with a random forest algorithm. The random forest algorithm creates n decision trees, each using bootstrapping to randomly generate a new training dataset from the initial dataset and randomly selecting splitting criteria to generate predictions. The algorithm then aggregates the predictions made by the n decision trees to improve the generalisation of the final prediction.

5.4.2 Implementation of the Random Forest Algorithm:

```
### ============================================================================================================
###                                          Retrieve data From CSV to DF
async def retrieve_data_from_cvs_to_df(ticker: str) -> pd.DataFrame:
    try:
        # 1) Load the CSV file into a DataFrame
        df = pd.read_csv(CSV_FILE_PATH, parse_dates=['Date'])
        # 2) Filter the DataFrame by the specified ticker
        ticker_df = df[df['Ticker'] == ticker]
        # 3) Select the desired columns
        selected_columns = ['Date', 'High', 'Low', 'Close', 'Volume']
        ticker_df = ticker_df[selected_columns]
        # 4) Set the Date column as the index
        ticker_df.set_index('Date', inplace=True)
        return ticker_df

    except Exception as e:
        print(f"Failed to retrieve data for ticker {ticker}: {e}")
        return None
### ============================================================================================================
```

Figure 5.4.2.1: Retrieve_data_from_csv_to_df function

The first stage concerns the retrieval of 5 years worth of data of a specific ticker from the training data set. The ‘retrieve_data_from_csv_to_df’ function

gathers 5 years of data from a csv file and converts it into a pandas DataFrame object used to process the data.

After retrieving the financial data in the form of a DataFrame, the ‘compute_features_and_remove_nan’ function orchestrates the data processing required before fitting the training data into the random forest algorithm. Relative Strength Index (RSI) is computed by the ‘compute_rsi’ function. The RSI is a momentum oscillator that measures the speed and change of price movements, helping to identify overbought or oversold conditions in the market.Next, the Stochastic Oscillator (SO) which includes the indicators ‘K%’, ‘D%’, and ‘R%’, is calculated using the ‘compute_stochastic_oscillators function’. This oscillator compares a particular closing price to a range of its prices over a certain period, providing insights into the strength and momentum of price movements.The Price Rate of Change (PROC) is determined through the ‘compute_proc’ function. PROC measures the percentage change in price between the current closing price and the closing price from a set number of periods ago, highlighting the velocity of price changes.The Moving Average Convergence Divergence (MACD), along with its signal line and histogram (MACD_Signal and MACD_Diff), is generated by the ‘compute_macd’ function. MACD is a trend-following indicator that shows the relationship between two moving averages of a stock’s price.Finally, the On Balance Volume (OBV) which is computed by the ‘compute_obv’ function, reflects cumulative buying and selling pressure by adding volume on up days and subtracting it on down days.

```python
### Purpose: Compute Financial Features X and remove Nan values:
async def compute_features_and_remove_nan(df: pd.DataFrame) -> pd.DataFrame:
    # 1) Compute all financial indicators
    df = await compute_rsi(df)
    if df is None:
        logger.error("An error occured in the computation of RSI.")
        return None

    df = await compute_stochastic_oscillators(df)  # await is required here
    if df is None:
        logger.error("An error occured in the computation of K%, D%, or R%.")
        return None

    df = await compute_proc(df)  # await is required here
    if df is None:
        logger.error("An error occured in the computation of PROC.")
        return None

    df = await compute_macd(df)  # await is required here
    if df is None:
        logger.error("An error occured in the computation of MACD, MACD_Signal or MACD_Diff.")
        return None

    df = await compute_obv(df)  # await is required here
    if df is None:
        logger.error("An error occured in the computation of OBV.")
        return None

    # 2) Remove Nan values:
    df = df.dropna()
    return df
```

Figure 5.4.2.2: Compute_features_and_remove_nan function

Then the next step after computing all the features is to generate the prediction column, which represents the label and must be in binary format. This is crucial because the model used, a decision tree classifier, handles discrete values rather than continuous ones. The labels were generated by determining the market trend: if the closing price at the next time step is lower than the current closing price, the label is set to 0 (indicating a downward trend). Otherwise, the label is set to 1 (indicating an upward trend).

The ‘generate_prediction_column’ function facilitates this process. It starts by comparing the current closing price to the previous one to identify whether the price increased or decreased, thus generating a series of boolean values representing the market trend. These boolean values are then converted into integers, with 0 representing a downward trend and 1 representing an upward trend. A new column called ‘Prediction’ is added to the DataFrame to store these labels.

Additionally, the function removes unnecessary features such as 'High,' 'Low,' 'Close,' and 'Volume,' as they are no longer needed for the model. Finally, the

function checks that the label column contains only the binary values 0 and 1, ensuring the data is ready for the decision tree classifier.

This is how the training data set looks like after the processing phase:

Date	RSI	K%	D%	R%
2019-09-13 00:00:00-04:00	61.866092	66.796753	84.389455	-33.203247
2019-09-16 00:00:00-04:00	64.041811	71.774882	75.157232	-28.225118
2019-09-17 00:00:00-04:00	65.616518	75.238073	71.269903	-24.761927
2019-09-18 00:00:00-04:00	69.592262	83.558589	76.857181	-16.441411
2019-09-19 00:00:00-04:00	62.321859	75.405506	78.067389	-24.594494
...	...	...	...	...
2024-07-22 00:00:00-04:00	46.554507	40.036172	46.843388	-59.963828
2024-07-23 00:00:00-04:00	49.586727	32.857125	40.615427	-67.142875
2024-07-24 00:00:00-04:00	35.333707	7.014871	26.636056	-92.985129
2024-07-25 00:00:00-04:00	33.529066	12.693544	17.521847	-87.306456
2024-07-26 00:00:00-04:00	35.237460	14.772276	11.493563	-85.227724

Date	PROC	MACD	MACD_Signal	MACD_Diff
2019-09-13 00:00:00-04:00	7.950070	0.869902	0.526849	0.343053
2019-09-16 00:00:00-04:00	6.494250	0.899796	0.601438	0.298357
2019-09-17 00:00:00-04:00	8.101483	0.928419	0.666834	0.261584
2019-09-18 00:00:00-04:00	8.388086	0.980256	0.729519	0.250738
2019-09-19 00:00:00-04:00	5.717440	0.974730	0.778561	0.196169
...	...	...	...	...
2024-07-22 00:00:00-04:00	3.326416	6.186814	7.572643	-1.385829
2024-07-23 00:00:00-04:00	2.151900	5.660423	7.190199	-1.529776
2024-07-24 00:00:00-04:00	-1.358614	4.667377	6.685634	-2.018258
2024-07-25 00:00:00-04:00	-3.910043	3.752400	6.098988	-2.346588
2024-07-26 00:00:00-04:00	-4.327978	3.030268	5.485244	-2.454976

Date	OBV	Prediction
2019-09-13 00:00:00-04:00	98914000	0
2019-09-16 00:00:00-04:00	183546400	1
2019-09-17 00:00:00-04:00	256821200	1
2019-09-18 00:00:00-04:00	358181200	1
2019-09-19 00:00:00-04:00	269938800	0

Figure 5.4.2.3: Processed dataset for the price forecast model

The ‘compute_random_forest’ function is designed to predict the trend of tomorrow's closing price using a Random Forest model. The process starts by dividing the dataset into features (x_column) and labels (y_column). To ensure a robust model, time series cross-validation is first applied, which splits the data into multiple folds while maintaining the order of the time series data. This step helps in assessing the model's performance across different splits and selecting the best ratio between the training and validation sets based on the Mean Absolute Error (MAE).

After determining the best training-to-validation ratio, the function proceeds with a grid search to fine-tune the model's hyperparameters. This includes optimising parameters like the number of decision trees (n_estimators), the

maximum depth of each tree (max_depth), the criterion used for splitting (Gini's entropy), and the maximum number of features considered for splitting (max_features). The grid search helps in finding the optimal combination of these parameters to improve the model's accuracy.

Once the best model parameters are identified, the Random Forest model is fitted to the final training set. The function then evaluates the model on the test set, calculating the final Mean Absolute Error (MAE) and accuracy to assess its performance. Additionally, if available, the Out-of-Bag (OOB) score is printed, providing an internal validation measure of the Random Forest model.

Finally, the function uses the optimised model to predict the trend of the closing price for the next day, based on the most recent data available. The predicted trend is then returned as a binary output, indicating whether the closing price is expected to be higher or lower.

Lastly, regarding the evaluation metrics of the model implemented, The evaluation metrics provide insight into the performance of the machine learning model used for predicting stock trends. The model achieved an accuracy of approximately 82%, meaning it correctly predicted the trend of the closing price for about 82% of the test cases. Precision and recall are close to each other, with precision at 81.7% indicating how often the model's positive predictions were correct, and recall at 84.8% reflecting the model's ability to identify actual positive cases. The F1-score, which balances precision and recall, also suggests a well-performing model. The specificity, or true negative rate, is 79.7%, showing the model's effectiveness in correctly identifying down days.

	precision	recall	f1-score	support
Down Day	0.830986	0.797297	0.813793	74.000000
Up Day	0.817073	0.848101	0.832298	79.000000
accuracy	0.823529	0.823529	0.823529	0.823529
macro avg	0.824030	0.822699	0.823046	153.000000
weighted avg	0.823802	0.823529	0.823348	153.000000

Accuracy: 0.8235294117647058
Precision: 0.8170731707317073
Recall: 0.8481012658227848
Specificity: 0.7972972972972973

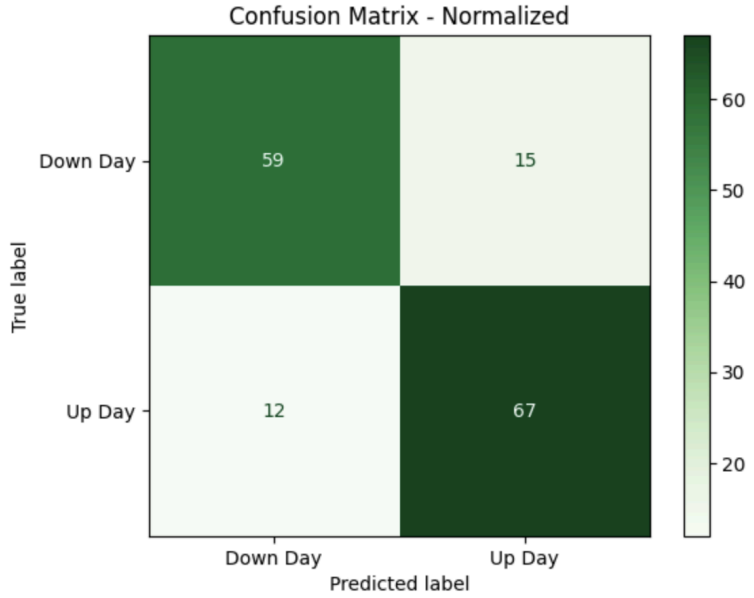


Figure 5.4.2.4: Evaluation metrics for the price forecast model

5.5 K Means:

5.5.1 Intuition Behind The Kmeans:

Concerning the machine learning model used for clustering stocks listed on the S&P 500 according to their average annual volatility and return, the K-means algorithm was employed. The objective of the model is to split an unlabeled data set in to groups of clusters of similar data point K.

As for the mathematical intuition behind this model, first a dataset $D = \{(X^n)\}$ is fitted in the model with chosen number of features n that defines the dimensionality of the feature set X. The algorithm then randomly initialises K cluster centres ($\mu_1..\mu_K$). The process continues until convergence is reached, meaning that the cluster centres stabilise and no longer change. During each

iteration, the algorithm assigns each data point to the nearest cluster centre by minimising the Euclidean distance between the data point and the cluster centres. After assigning the data points, the cluster centres are recalculated, and the process repeats until the cluster centres no longer change (Daume, 2023).

Here is the formula to compute the clusters centre:

$$\mu_K = \frac{1}{N_k} \Sigma x_n$$

Where, μ_K represents the new centre of the cluster, N is the number of data points in the cluster, and x_n denotes each data point in the cluster. This formula calculates the mean of all data points assigned to a cluster to determine the new centre.

The formula to assign each data point to a cluster K_i:

$y_n = argmin(K)||x_n - \mu_K||$

In this equation, y_n represents the label or the cluster assignment for data point x_n. The goal is to find the cluster centre K_i that minimises the Euclidean distance between the data point x_n and the cluster centre μ_K This ensures that each data point is assigned to the cluster whose centre is closest to it.

5.5.2 Implementation of the Stock Clustering Service:

In order to retrieve training data from the CSV file to utilise it, a similar function that the one used during the computation of the random forest was used. The only difference is that this time the whole data set is utilised. The function ‘retrieve_data_to_df’ fetches the training data from the CSV file and converts it to a pandas DataFrame object used for data processing for feature

computation. Here the only required columns needed are the date, ticker symbol and the closing price that will be used.

```python
### ============================================================================================================
###                                    Retrieve data From CSV to DF on all tickers:
async def retrieve_data_to_df() -> pd.DataFrame:
    try:
        # 1) Load the CSV file into a DataFrame
        df = pd.read_csv(CSV_FILE_PATH, parse_dates=['Date'])
        # 2) Select the desired columns
        selected_columns = ['Date', 'Ticker','Close']
        df = df[selected_columns]
        # 3) Set the Date column as the index
        df.set_index('Date', inplace=True)
        return df

    except Exception as e:
        print(f"Failed to retrieve data from {CSV_FILE_PATH}: {e}")
        return pd.DataFrame()
```

Figure 5.5.2.1: Retrieve_data_to_df function

Ater retrieving the data set in the form of a pandas dataframe, the ‘compute_avr_annual_return_and_volatility’ function is used to compute the average annual return and volatility of each stocks listed in the S&P 500 as feature set for the kmean model to segregate each stocks. The function returns a dataframe that will be

```python
### ============================================================================================================
###                     Compute Financial Features X = {Avr annual return, Avr annual volatility}:

### Purpose: Compute Avr annual return and volatility of all stocks and return it in a DF:
async def compute_avg_annual_return_and_volatility(df: pd.DataFrame) -> pd.DataFrame:
  # 1) Generate a DF with The Date as index and Close and Ticker as column:
  pivot_df = df.pivot(columns='Ticker', values='Close')
  # 2) Handle missing values explicitly using ffill and bfill:
  pivot_df.ffill(inplace=True)
  pivot_df.bfill(inplace=True)
  # 3) Compute daily returns:
  daily_returns = pivot_df.pct_change(fill_method=None)
  # 4) Calculate the average annual return:
  avg_annual_return = daily_returns.mean() * 252
  # 5) Compute annual volatility (std of daily returns scaled by sqrt root of 252 = nbr of traiding days/years):
  annual_volatility = daily_returns.std() * np.sqrt(252)
  # 6) Combine the result into a single Dataframe:
  result_df = pd.DataFrame({'Avr Annual Return': avg_annual_return, 'Avr Annual Volatility': annual_volatility})
  # 7) Ensure there is no Nan values or that one or more column is empty:
  if result_df[['Avr Annual Return', 'Avr Annual Volatility']].isnull().all().any():
    logger.error("Error, either the Avr Annual Return or the Avr Annual Volatility or both are either empty or contains Nan values.")
    return None
  return result_df
```

Figure 5.5.2.2: Compute_avr_annual_return_and_volatility function

Ticker	Avr Annual Return	Avr Annual Volatility
A	0.184138	0.295618
AAL	-0.027398	0.617271
AAPL	0.337491	0.319177
ABBV	0.259100	0.242153
ABNB	0.084483	0.426751
...	...	...
XYL	0.168676	0.311414
YUM	0.072915	0.261653
ZBH	0.017785	0.310548
ZBRA	0.175329	0.404741
ZTS	0.137332	0.290864

Figure 5.5.2.3: Processed dataset for the stock clustering model

Then to identify and remove potential outliers, the ‘remove_outliers_iqr’ function was implemented This function uses the Interquartile Range (IQR) method to detect and handle outliers in the dataset.

First, it computes the first quartile (Q1) and the third quartile (Q3) for the average annual return and volatility. The IQR is then calculated as the difference between Q3 and Q1. Using this range, the function defines lower and upper bounds for what constitutes an outlier. Data points that fall outside these bounds are considered outliers.

The function filters out these outliers from the dataset, ensuring that only data within the acceptable range is used for further processing. It also identifies and prints any detected outliers for review, if present. This step is crucial for maintaining the integrity of the clustering process and ensuring that the K-means algorithm performs accurately without being skewed by extreme values.

Concerning the number of clusters to use to segregate the dataset, the Elbow Method Method was used to identify the optimal number of clustersThis technique involves plotting a graph to show how the "distortion" which

represents a measure of how far away data points are from their cluster centres changes as the number of clusters increases.

To compute this curve, the processed data, which includes each stock's average annual return and volatility, was prepared and fed into the K-means algorithm for different numbers of clusters, ranging from 2 to 14. For each number of clusters, the algorithm calculates the distortion. A graph is then created to display these distortions. The curve typically drops sharply at first, then levels off as more clusters are added. The point where the curve starts to flatten out, reaching a steady state indicates the best number of clusters.

In this case, the curve flattened at 4 clusters, meaning that 4 is the optimal number. This choice effectively groups the stocks into four distinct categories: high volatility with low return, low volatility with high return, high volatility with high return, and low volatility with low return.

```
from pylab import plot,show
from matplotlib import pyplot as plt
import plotly.express as px
from numpy.random import rand
from scipy.cluster.vq import kmeans,vq
from math import sqrt
from sklearn.cluster import KMeans
from sklearn import preprocessing
# 2) Determine the optimum number of clusers K w/ the Elbow's curve.
# Since we want to classify stocks according to 2 features, most rational choice of K = 2^2
# Combination = {High-volatility/Low-return, Low-volatility/High-return, High-volatility/High-return, Low-volatility/Low-return

# Format the data as a numpy array to feed into the K-Means algorithm
dataset = np.asarray([np.asarray(yearly_data['Avr Annual Return']),
                      np.asarray(yearly_data['Avr Annual Volatility'])]).T

# Calculate the distortions for a range of cluster sizes
distortions = []
for k in range(2, 15):
    k_means = KMeans(n_clusters=k)
    k_means.fit(dataset)
    distortions.append(k_means.inertia_)

# Plot the Elbow Curve
fig = plt.figure(figsize=(10,3))
plt.plot(range(2, 15), distortions, marker='o')
plt.grid(True)
plt.title("Elbow's Curve")
plt.xlabel('Number of clusters (k)')
plt.ylabel('Distortion (Inertia)')
plt.show()
```

Figure 5.5.2.4: Function employed to plot the Elbow Curve

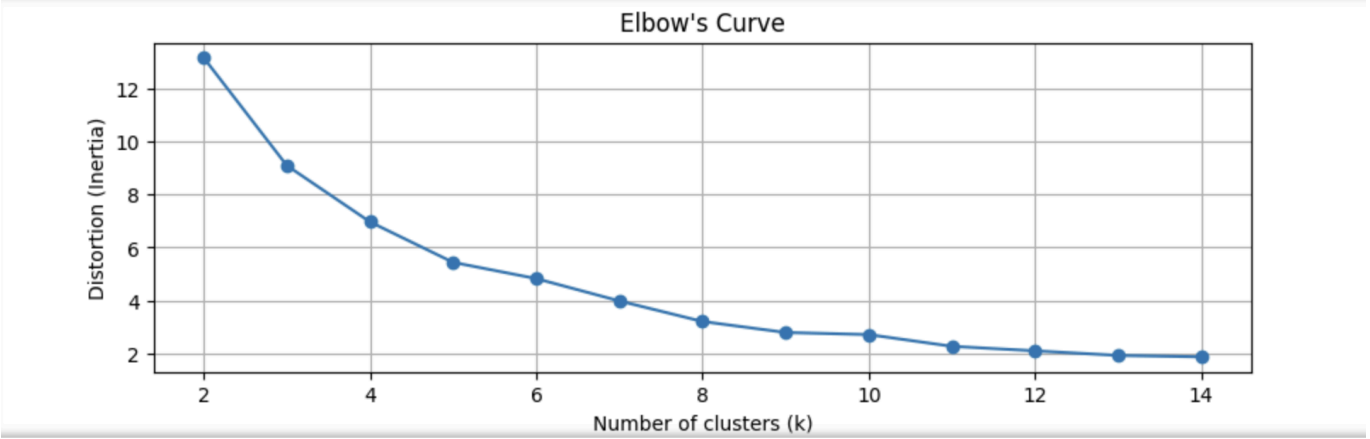


Figure 5.5.2.5: Elbow curve

5.6 Storing and daily incrementation of the training dataset:

To handle the storage of the training dataset, a function called ‘store_data_into_csv’ was implemented to fetch all stocks listed in the S&P 500 over a period of five years from the Yahoo Finance API. This function organises the data in the CSV file by grouping it according to each ticker symbol and date, ensuring efficient retrieval of the training data. Whenever new data is added to the CSV file, another function called ‘write_data_when_retrieved’ is triggered. This function appends the date to a record text file, maintaining a log of the last date of financial data recorded, which helps facilitate updates through daily increments.

```python
# Async function that gathers daily financial data to store it in a CSV file:
async def store_data_into_cvs(tickers: list) -> bool:
    # 1) Initialize an empty dataframe to store all the financial data:
    final_df = pd.DataFrame()
    # 2) Iterate through all ticker in tickers:
    for ticker in tickers:
        # 2.1) Gather the stock's financial data for past 5y:
        stock = yf.Ticker(ticker)
        stock_df = stock.history(period="5y")
        # 2.2) Assess if there df is empty => don't use this stock
        if not stock_df.empty:
            # 2.3) Add the stock's ticker symbol as a column:
            stock_df['Ticker'] = ticker
            # 2.4) append stock_df into the dataframe that will be converted into a CSV file:
            final_df = pd.concat([final_df, stock_df])
        else:
            logger.info(f"No financial data could be gathered for the following stock {ticker}")

    # 3) Assess whether the final dataframe is empty:
    if final_df.empty:
        logger.error(f"Error: The dataframe used to store all stock's financial result is empty")
        return False

    # 4) If final DF not empty, attempt to convert it in CVS
    result = await convert_df_to_csv(final_df)
    # 5) If operation was successful attempt to write the date where data ends_
    if result:
        return await write_date_when_retrieved()
    return False
```

Figure 5.6.1: Store_data_into_csv function

Regarding daily updates to the database, the initial plan was to deploy the application via AWS Lambda with a scheduler to perform endpoint requests for database updates. However, due to concerns about costs, a manual approach was adopted. In this approach, only the admin is responsible for updating the database by clicking the "Update Database" button, which sends a request to the appropriate endpoint. The ‘get_missing_recent_data’ function retrieves the last date of data from the database by reading the last entry in the ‘record.txt’ file. It then gathers the missing data for all stocks, starting from the retrieved date up to the current date when the function is executed, to collect the recent missing data. After adding this data to the CSV file, it is sorted so that each stock is once again ordered according to its ticker symbol and date. The ‘write_data_when_retrieved’ function is then called to log the date when the database was last updated.

```python
### Purpose: Gather missing financial data for all stocks from SP500:
async def get_missing_recent_data(tickers: list) -> bool:
    # 1) Initialize an empty DataFrame to store all the financial data:
    final_df = pd.DataFrame()
    # 2) Set the last closing day you want to get:
    yesterday_closing_date = datetime.now().strftime("%Y-%m-%d")
    logger.info(f"Yesterday's closing price is: {yesterday_closing_date}")
    print(f"Yesterday's closing price is: {yesterday_closing_date}")
    # 3) Attempt to retrieve the most recent date worth of financial data from txt file:
    # 3) This date represents the latest closing date of data we have in the csv file:
    last_closing_date_in_csv = await get_latest_date_of_financial_data()
    logger.info(f"The last closing date in CSV retrieved is: {last_closing_date_in_csv}")
    print((f"The last closing date in CSV retrieved is: {last_closing_date_in_csv}"))
    # 3.1) If operation was successful: Convert the date to strptime in the format YYYY-MM-DD:
    if last_closing_date_in_csv:
        last_closing_date_in_csv = (datetime.strptime(last_closing_date_in_csv, "%Y-%m-%d") + timedelta(days=1)).strftime("%Y-%m-%d")
        print(f"last closing date in CSV: {last_closing_date_in_csv}")
        # 4) Iterate through all tickers in the list:
        for ticker in tickers:
            # 4.1) Gather all financial data from last_closing_date to the most recent one = yesterday_closing_date:
            stock = yf.Ticker(ticker)
            stock_df = stock.history(start=last_closing_date_in_csv, end=yesterday_closing_date)
            # 4.2) Assess if the df of this stock is empty => don't use this stock:
            if not stock_df.empty:
                # 4.3) Add the stock's ticker symbol as a column:
                stock_df['Ticker'] = ticker
                # 4.4) Concat the stock_df with the final dataframe:
                final_df = pd.concat([final_df, stock_df])
            else:
                logger.info(f"No financial data could be gathered for the following stock {ticker}")
        # 5) Assess if the final dataframe where all stock's financial data is empty:
        if final_df.empty:
            logger.error("Error: The dataframe used to store all stock's financial results is empty")
            return False
        # 5.1) If dataframe not empty, try to append the latest data into the CSV file:
        else:
            success = await add_data_to_csv(final_df)
            # 5.2) If operation was successful, try to write the latest date worth of data retrieved into the txt file:
            if success:
                return await write_date_when_retrieved()
            else:
                return False
```

Figure 5.6.2: get_missing_recent_data function

Lastly, it's important to note that some stocks have been delisted from the S&P 500. The following stocks were excluded from the project due to the unavailability of data: ['GEV', 'KVUE', 'GEHC', 'VLTO', 'SOLV', 'BN.F', 'BRK.B', 'BF.B', 'WRK'].

5.7 Conclusion

In conclusion, in this chapter, the functioning of both models respectively, the Random Forest algorithm for price prediction and the K-Means clustering for stock segregation was thoroughly discussed. The chapter detailed the essential steps in processing the data, from computing features for price prediction to determining the optimal number of clusters for categorising stocks based on their average annual return and volatility. Additionally, the

chapter outlined the methods for storing and updating the training data, ensuring that the dataset remains current and well-organised for effective model training and prediction.

Chapter 6: Testing

6.1 Testing Strategy:

In this section, a thorough testing of the machine learning web application will be performed to evaluate the robustness of the application. To test both the client and server sides of the system, a black-box and white-box testing strategy will be used. A testing table recording all tests has been implemented to describe each test performed.

For black-box testing, unit and integration tests will be conducted using test cases written with ‘Pytest Asyncio’ library library to ensure conditions similar to those used in production. Jenkins will be used as the continuous integration tool to automate test cases by implementing a pipeline.

For white-box testing, the evaluation involves providing diverse cases of inputs and assessing the returned responses in the console, as well as the expected behaviour of the displayed components.

6.2 Black Box Testing:

6.2.1: Unit Test:

In order to replicate asynchronous test cases that mimic the production environment, a ‘configtest.py’ file was developed to configure the necessary fixtures for the ‘pytest-asyncio’ test cases to run. This file sets up a mock local SQLite database to replace the actual PostgreSQL database by overriding the

‘get_db’ session implemented for PostgreSQL with one that uses a SQLite test URL.

Fixtures used in the ‘configtest.py’ file to prepare the environment for testing:

- ‘App_test’ fixture: Sets up and tears down the FastAPI app for each test by creating and dropping the database tables.
- ‘Client’ fixture: Enables an instance of ‘TestClient’ to make requests.
- ‘Db_session’ fixture: Create a SQLite database session and enable it to overwrite the session using a PostgreSQL database.

Table 6.2.1: Unit testing table

Test No	Test Description	Test Data	Expected Result	Actual Result	Status
1	Test create user account function with strong password and valid email	mock_data = UserCreate(username=’testuser’, email=’testuser@example.com, password=’StrongPassword123$ ’)	- The user should be stored in the database with the following field values: username == ‘mock_data.username’, ‘email’ == mock_data.email, ‘role’ = ‘user’, ‘is_active’ and ‘is_authenticated’ statuses == False.	- The user was stored in the database with the following field values: username == ‘mock_data.username’, ‘email’ == mock_data.email, ‘role’ = ‘user’, ‘is_active’ and ‘is_authenticated’ statuses == False.	P
2	Test create user account with a weak password	weak_user_data= UserCreate(username=’testuser2’, email=’testuser@example2.com, password=’Strong’)	- Status code == 400 - Returned message: “Password isn’t strong enough.” - The user shouldn’t get created in the database.	- Status code == 400 - Returned message: “Password isn’t strong enough.” - The user wasn't created in the database.	P
3	Test create user function with an existing email	mock_data = UserCreate(username=’testuser’, email=’testuser@example.com, password=’StrongPassword123$ ’)	- Status code 400 - Returned message: “A user with this email already exists.” - The user shouldn’t get created in the database.	- Status code 400 - Returned message: “A user with this email already exists.” - The user shouldn’t get created in the database.	P
4	Test send email verification link function when a user account is	mock_data = UserCreate(username=’testuser’, email=’testuser@example.com, password=’StrongPassword123$ ’)	- A verification link with the corresponding valid arguments should be sent to the user’s email address.	- A verification link with the corresponding valid arguments was sent to the user’s email address.	P

	created				
5	Test activate account function when the user clicks on the activation link	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ')	- The user's 'is_active' and 'is_authenticated' statuses should be set to True.	- The user's 'is_active' and 'is_authenticated' statuses were set to True.	P
6	Test get_user_role function	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ')	- The 'role' of the user stored in the database should be returned.	- The 'role' of the user stored in the database was returned.	P
7	Test update_role function	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_role = 'Admin'	- The 'role' of the user stored in the database should be changed from the default 'user', when an account is created to 'Admin'.	- The 'role' of the user stored in the database is changed from the default 'user', when an account is created to 'Admin'.	P
8	Test update_role function when there is a DB error	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_role = 'Admin' Simulate a database error using MagicMock library by simulating a database error	- Status code 500 - Returned message: "Failed to update user role"	- Status code 500 - Returned message: "Failed to update user role"	P
9	Test update_username function	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_username = 'CoolNewUsername'	- The 'username' of the user stored in the database should be set to 'CoolNewUsername'.'	- The 'username' of the user stored in the database was set to 'CoolNewUsername'.''	P
10	Test update_username function when there is a DB error	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_username = 'CoolNewUsername' Simulate a database error using	- Status code 500 - Returned message: "Failed to update username"	- Status code 500 - Returned message: "Failed to update username"	P

		MagicMock library by simulating a database error			
11	Test update_user_detail function to update the username and role	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_username = "Jake" new_role = "Admin"	- The 'username' and 'role' of the user stored in the database should be respectively set to 'Jake' and 'Admin'.	- The 'username' and 'role' of the user stored in the database were respectively set to 'Jake' and 'Admin'.	P
12	Test update_user_detail function with only a new username provided	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_username = "Jake"	- The 'username' of the user stored in the database should be set to 'Jake'.	- The 'username' of the user stored in the database was set to 'Jake'.	P
13	Test update_user_detail function with only a new role provided	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_role = "Admin"	- The 'role' of the user stored in the database should be set to 'Admin'.	- The 'role' of the user stored in the database was set to 'Admin'.	P
14	Test the update_user_detail function when there is a DB error	mock_data = UserCreate(username='testuser', email='testuser@example.com, password='StrongPassword123$ ') new_username = "Jake" new_role = "Admin" A database error is simulated by using MagicMock library	- Status code 500 - Returned message: "Failed to update username and role"	- Status code 500 - Returned message: "Failed to update username and role"	P
15	Test list_all_users function	No data is required. In this situation two users are stored in the database.	- All users stored in the database along with their details should be returned.	- All users stored in the database along with their details were returned.	P
16	Test list_all_users function when no users are stored in the database	No data is required	- An empty list should be returned.	- An empty list is returned.	P
17	Test list_all_users function when an error	No data is required A database error is simulated by	- Status code 500 - Returned message: "An error occurred while	- Status code 500 - Returned message: "An error occurred while	P

	occurred in the DB	using MagicMock library.	listing users.”	listing users.”	
18	Test hash_password to assess if user’s password is hashed in the DB	mock_data = UserCreate(username=’testuser’, email=’testuser@example.com, password=’StrongPassword123$’)	- The password stored in the database should get encoded.	- The password stored in the database got encoded.	P
19	Test get_login_token to determine if login function works	Ensured that the user’s account is activated by simulating clicking on the verification link. required *login*data = { “email”:mock_data.email, “password”:mock_data.password }	- An access token, a refresh token and expiration time should be returned. The type of the token should be set to “Bearer”.	- An access token, a refresh token and expiration time are returned. The type of the token that is returned is set to “Bearer”.	P
20	Test get_login_token when a wrong password is provided	Required_login_data_with_incorrect_password = { “email”:mock_data.email, “password”:”IncorrectPassword65”}	- Status code 400 - Returned message: “Invalid password.”	- Status code 400 - Returned message: “Invalid password.”	P
21	Test get_login_token when a wrong email address is provided	Required_login_data_with_incorrect_password = { “email”:”wrongEmail@Yahoo.com”, “password”:mock_data.password }	- Status code 400 - Returned message: “user doesn’t exist”	- Status code 400 - Returned message: “user doesn’t exist”	P
22	Test get_login_token when is_active status == False	Ensured that user.is_active status == False required *login*data = { “email”:mock_data.email, “password”:mock_data.password }	- Status code 400 - Returned message: “your account isn’t active.”	- Status code 400 - Returned message: “your account isn’t active.”	P
23	Test get_login_token when is_authenticated status == False	Ensured that user.is_authenticated status == False required *login*data = { “email”:mock_data.email, “password”:mock_data.password }	- Status code 400 - Returned message: “your account isn’t authenticated.”	- Status code 400 - Returned message: “your account isn’t authenticated.”	P
24	Test freeze_user_account to update user.is_active status from	user = User(user=mock_data.email, password=hash_password(mock_data.password), is_active=True, is_authenticated=True,	- The ‘is_active’ status from the user stored in the database should be set to ‘False’.	- The ‘is_active’ status from the user stored in the database is set to ‘False’.	P

	True to False	role=”user”)			
25	Test unfreeze_user_account to update user.is_active status from False to True	user = User(user=mock_data.email, password=hash_password(mock_data.password), is_active=False, is_authenticated=True, role=”user”)	- The ‘is_active’ status from the user stored in the database should be set to ‘True’.	- The ‘is_active’ status from the user stored in the database is set to ‘True’.	P
26	Test freeze_user_account when the wrong user id is provided	user = User(user=mock_data.email, password=hash_password(mock_data.password), is_active=False, is_authenticated=True, role=”user”) Fake_user_id = 1919	- Status code 404 - Returned message: “User with ID=1919 not found.”	- Status code 404 - Returned message: “User with ID=1919 not found.”	P
27	Test unfreeze_user_account when the wrong user id is provided	user = User(user=mock_data.email, password=hash_password(mock_data.password), is_active=False, is_authenticated=True, role=”user”) Fake_user_id = 1919	- Status code 404 - returned message: “User with ID=1919 not found.”	- status code == 404 - returned message: “User with ID=1919 not found.”	P
28	Test reset_password function with a strong password provided	new_strong_psw = ResetPasswordFields(username=mock_data.username, email=mock_data.email, new_password=”ThisIsMyNewStrongPSW64$”)	- User’s password stored in the database should get updated.	- User’s password stored in the database got updated.	P
29	Test reset_password function with a weak password provided	new_strong_psw = ResetPasswordFields(username=mock_data.username, email=mock_data.email, new_password=”new”)	- Status code 400 - Returned message: “Password isn’t strong enough”	- Status code 400 - Returned message: “Password isn’t strong enough”	P
30	Test reset_password function when a wrong username has been provided	new_strong_psw = ResetPasswordFields(username=”Mark”, email=mock_data.email, new_password=”DDDrkjaiewj234”)	- Status code 400 - Returned message: “No user with the following username Mark matches the account linked to the email address.”	- Status code 400 - Returned message: “No user with the following username Mark matches the account linked to the email address.”	P
31	Test reset_passwo	new_strong_psw = ResetPasswordFields(	- Status code 400	- Status code 400	P

	rd function when a wrong email has been provided	username=mock_data.username , email=”ericGorge@yahoo.com”, new_password=”DDDrkjaiewj234 ”)	- Returned message: “No user with the following email ericGorge@yahoo.com has been found.”	- Returned message: “No user with the following email ericGorge@yahoo.com has been found.”	
32	Test fetch_user_de tail function	user.id from the DB was used.	- The ‘username’, ‘email’, ‘id’, ‘password’, ‘is_active’ and ‘is_authenticated’ status and ‘role’ should be returned.	- The ‘username’, ‘email’, ‘id’, ‘password’, ‘is_active’ and ‘is_authenticated’ status and ‘role’ are returned.	P
33	Test fetch_user_de tail when an incorrect user id has been provided	Incorrect_id = 2934	- Status code 400 - Returned message: “User does not exist.”	- Status code 400 - Returned message: “User does not exist.”	P
34	Test logout_user function	user = User(id=5 user=mock_data.email, password=hash_password(mock _data.password), is_active=True, is_authenticated=True, role=”user”)	- No tokens associated with the user should be stored in the Token table in the database. - An empty list should be returned when attempting to retrieve tokens associated with the user.	- No tokens associated with the user are stored in the Token table in the database. - An empty list is returned when attempting to retrieve tokens associated with the user.	P
35	Test _generate_to kens function in TokenControll er.py when a user logs in.	user = User(id=5 user=mock_data.email, password=hash_password(mock _data.password), is_active=True is_authenticated=True, role=”user”)	- The function should return an access token, a refresh token and an expiration time. - JWT token should be stored in the database and should be associated with the user. - token_expiration time > current time - When decoded, the user.id can be retrieved along with the access and refresh token payload.	- The function returns an access token, a refresh token and an expiration time. - JWT token is stored in the database and is associated with the user. - token_expiration time > current time When decoded, the user.id can be retrieved along with the access and refresh token payload.	P
36	Test get_current_u ser function with a valid jwt token.	user = User(id=5 user=mock_data.email, password=hash_password(mock _data.password), is_active=True	- The function should return the user. - user’s fields such as ‘emaié’, ‘username’,’ id’, ‘is_active’,	- The function returns the user. - user’s fields such as ‘emaié’, ‘username’,’ id’, ‘is_active’,	P

		is_authenticated=True, role="user") Provide the token returned by the 'get_login_token' function.	'is_authenticated' and 'role' should be identical as the ones defined in the user object created.	'is_authenticated' and 'role' are identical as the ones defined in the user object created.	
37	Test load_user function	user = User(id=5 user=mock_data.email, password=hash_password(mock_data.password), is_active=True is_authenticated=True, role="user") User's email is passed in the load_user function	- The function should return the user. - user's fields such as 'emaié', 'username',' id', 'is_active', 'is_authenticated' and 'role' should be identical as the ones defined in the user object created.	- The function returns the user. - user's fields such as 'emaié', 'username',' id', 'is_active', 'is_authenticated' and 'role' are identical as the ones defined in the user object created.	P
38	Test retrieved_data_to_df function that enables retrieving the training dataset stored in a CSV file.	No data required.	- The returned object should be a pandas DataFrame. - The returned object shouldn't be None or empty. - The DataFrame should contain the following columns: Ticker and Close. - The 'Date' column should be set as the index.	- The returned object is a pandas DataFrame. - The returned DataFrame isn't empty. - The DataFrame contains the following columns: Ticker and Close. - Date column is set as the index.	P
39	Test compute_annual_return_and_volatility function in KMeanService.py	No data required. Requirement: 1) The 'retrieve_data_to_df' function should first be called to retrieve from the CSV file the training data set to store it in a pandas DataFrame.	- The returned object should be a pandas DataFrame. - The returned object shouldn't be None or empty. - The DataFrame contain should the following columns: Avr Annual Return, Avr Annual Volatility. - The DataFrame shouldn't contain Nan values.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The DataFrame contains the following columns: Avr Annual Return, Avr Annual Volatility. - The DataFrame doesn't contain Nan values.	P

40	Test remove_outliers_iqr function used to remove outliers before performing the K Means algorithm in KMeanService.py	Dummy data created:
		
		Data = {
		‘Avr Annual Return’:[0.1, 0.2, 0.3, 0.4, 0.5, 5], (5 is the outlier)
		
		‘Avr Annual Volatility’:[0.1, 0.2, 0.3, 0.4, 0.5, 5], (5 is the outlier)
		}
			- Outliers should be removed from the DataFrame.	- Outliers are removed from the DataFrame.	P

41	Test perform_kmeans_and_create_dataframe function in KMeanService.py	Requires: 1) ‘retrieve_data_to_df’ function is called to get the training data in a pandas DataFrame format. 2)‘compute_avg_annual_return_and_volatility’ function is called to compute the features needed by the model. 3) The remove_outliers_iqr() function is called to remove potential outliers before fitting the training data.	- The returned object should be a JSON dictionary encoded in JSON and should contain the following keys: [Ticker, Avr Annual Return, Avr Annual Volatility, Cluster].	- The returned object is a JSON dictionary encoded in JSON and contains the following keys: [Ticker, Avr Annual Return, Avr Annual Volatility, Cluster].	P
42	Test retrieve_data_from_csv_to_df function that is used in RandomForestService.py	Ticker: ‘AAPL’	- The returned object is a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume] - The ‘Date’ column should be set as the index.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume] - Date column is set as the index.	P
43	Test compute_rsi function used in RandomForestService.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the ‘retrieve_data_from_csv_to_df’ function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI]. - The ‘RSI’ column should be empty.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI]. - The ‘RSI’ column isn’t empty. - Date column is set as the index.	P

			- The ‘Date’ column should be set as the index.		
44	Test compute_stochastic_oscillators function used in RandomForestService.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the ‘retrieve_data_from_csv_to_df’ function. 2) Compute the ‘RSI’ feature using the ‘compute_rsi’ function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R]. - The respective columns ‘%K’, ‘%D’ and ‘%D’ shouldn’t be empty. - The ‘Date’ column should be set as the index.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R]. - The respective columns ‘%K’, ‘%D’ and ‘%D’ aren’t empty. - The ‘Date’ column is set as the index.	P
45	Test compute_macd function used in RandomForestService.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the ‘retrieve_data_from_csv_to_df’ function. 2) Compute the ‘RSI’ feature using the ‘compute_rsi’ function. 3) Compute the stochastic oscillators using the ‘compute_stochastic_oscillators’ function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff]. - The respective columns ‘MACD’, ‘MACD_Signal’, and' MACD_Diff ‘MACD_Diff’ shouldn’t be empty. - The ‘Date’ column should be set as the index.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff]. - The respective columns ‘MACD’, ‘MACD_Signal’, and' MACD_Diff ‘MACD_Diff’ aren’t empty. - The ‘Date’ column is set as the index.	P

46	Test compute_obv function used in RandomForestService.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the ‘retrieve_data_from_csv_to_df’ function. 2) Compute the ‘RSI’ feature using the ‘compute_rsi’ function. 3) Compute the stochastic oscillators using the ‘compute_stochastic_oscillators’ function. 4) Compute ‘MACD’ indicators using the ‘compute_macd’ function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV]. - The ‘OBV’ column shouldn’t be empty. - The ‘Date’ column should be set as the index.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV]. - The ‘OBV’ column isn’t empty. - The ‘Date’ column is set as the index.	P
47	Test compute_proc function used in RandomForestService.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the ‘retrieve_data_from_csv_to_df’ function. 2) Compute the ‘RSI’ feature using the ‘compute_rsi’ function. 3) Compute the stochastic oscillators using the ‘compute_stochastic_oscillators’ function. 4) Compute ‘MACD’ indicators using the ‘compute_macd’ function. 5) Compute the ‘OBV’ indicator using the ‘compute_obv’ function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC]. - The ‘PROC’ column shouldn’t be empty. - The ‘Date’ column should be set as the index.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC]. - The ‘PROC’ column isn’t empty. - The ‘Date’ column is set as the index.	P

48	Test compute_features_and_remove_nan function used in RandomForest.py	Ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the retrieve_data_from_csv_to_df function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC]. - The ‘PROC’ column shouldn’t be empty. - The ‘Date’ column should be set as the index. - The Dataframe shouldn’t contain Nan values.	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC]. - The ‘PROC’ column isn't empty. - The ‘Date’ column is set as the index. - The Dataframe doesn’t contain Nan values.	P
49	Test generate_prediction_column function used in RandomForestService.py	Selected ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the retrieve_data_from_csv_to_df function. 2) Compute all features using the compute_features_and_remove_nan function.	- The returned object should be a pandas DataFrame. - The returned object shouldn’t be None or empty. - The returned DataFrame should possess the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC, Prediction]. - The ‘Date’ column should be set as the index. - The ‘Prediction’ column should contains binary values = {0,1}	- The returned object is a pandas DataFrame. - The returned object is not None or empty. - The returned DataFrame possesses the following columns = [High, Low, Close, Volume, RSI, %K, %D, %R, MACD, MACD_Signal, MACD_Diff, OBV, PROC, Prediction]. - The ‘Date’ column is set as the index. - The ‘Prediction’ column contains binary values = {0,1}	P

50	Test compute_random_forest function used in RandomForestService.py	Selected ticker: ‘AAPL’ Requires: 1) Fetch financial information regarding the ticker 'AAPL' using the retrieve_data_from_csv_to_df function. 2) Compute all features using the compute_features_and_remove_nan function. 3) Generate the label column using the generate_prediction_column function.	- The returned value must be either 0 (bearish prediction) or 1 (bullish prediction).	- The returned value are either 0 (bearish prediction) or 1 (bullish prediction)	P

6.2.2: Integration Test:

Regarding the integration tests, the initial approach was to use a local SQLite database to simulate the production environment. While this setup worked effectively for unit tests, it presented challenges for integration tests. The main issue was that the integration tests continued to attempt connections to the production database rather than using the intended mock database. After investigating discussions around similar integration test settings on StackOverflow, another approach was adopted. It became clear that a more suitable method was needed to replicate the production environment accurately.

To address this, a PostgreSQL container was set up using the TestContainers library. This approach allows for the creation of a temporary PostgreSQL instance that serves as a mock production database for the integration tests. By leveraging this PostgreSQL container, the tests interact with a realistic

database environment, ensuring that the test cases reflect actual production conditions more accurately.

The updated integration test configuration includes the following:

- A PostgreSQL container is initialised using the 'PostgresContainer' class from the 'TestContainer' library to create a temporary PostgreSQL instance that stimulates the production database.
- A connection URL is generated from the PostgreSQL container to establish a connection between the SQLAlchemy engine and the mock database.
- A mock SQLAlchemy session instance is generated using 'sessionmaker' to enable the test cases to interact with the database.
- A 'TestClient' instance is configured to enable the execution of API requests.

Table 6.2.2: Integration test table

Test No	Test Description	Test Data	Expected Result	Actual Result	Status
1	Create user account with a strong password	Mock_data = UserCreate(username='testuser', email='testuser@example.com , password='StrongPassword123$')	- Status code 201 - The user's account should be created and stored in the database with the role set to 'user', and both is_active and is_authenticated statuses set to True.	- Status code 201 - The user's account was created and stored in the database with the role set to 'user', and both is_active and is_authenticated statuses set to True.	P
2	Create user account with a weak password	Mock_data = UserCreate(username='testuser', email='testuser@example.com , password='weakpassword')	- Status code 400 - Returned message: "Password isn't strong enough." - The user's account shouldn't get created.	- Status code 400 - Returned message: "Password isn't strong enough." - The user's account didn't get created.	P
3	Create user account with an already used email	Mock_data = UserCreate(username='NewUser', email='testuser@example.com , password='StrongPassword123$')	- Status code 400 - Returned message: "A user with this email already exists." - The user's account shouldn't get created	- Status code 400 - Returned message: "A user with this email already exists." - User account not created - The user's account didn't get created	P

4	Test activate account route: Create a user and activate his account	Mock_data = UserCreate(username='NewUser', email='testuser3@example.com', password='StrongPassword123$')	- Status code 200 - Returned message: "Account is activated successfully." - user's is_active and is_Authenticated status should get set to True.	- Status code 200 - Returned message: "Account is activated successfully." - user's is_active and is_Authenticated status got set to True.	P
5	Test login route: Login using the previous user's account created and activated	Login_credentials = LoginFields(email='testuser3@example.com', password='StrongPassword123$')	- Status code 200 - A login response should be returned with an access key, a refresh key, and an expiry time. - Token type == Bearer	- Status code 200 - A login response is returned with an access key, a refresh key, and an expiry time. - Token type == Bearer	P
6	Test login route: Login using the wrong password	Login_credentials = LoginFields(email='testuser3@example.com', password='wrongPassword123$')	- Status code 400 - Returned message: "Invalid password."	- Status code 400 - Returned message: "Invalid password."	P
7	Test login route: Login using the wrong email	Login_credentials = LoginFields(email='false@example.com', password='StrongPassword123$')	- Status code 400 - Returned message: "User doesn't exist."	- Status code 400 - Returned message: "User doesn't exist."	P
8	Test login route: Login using an inactivated account	Login_credentials = LoginFields(email='testuser@example.com', password='StrongPassword123$')	- Status code 400 - Returned message: "Your account isn't active."	- Status code 400 - Returned message: "Your account isn't active."	P
9	Test Logout route	The access key attached in the header must be passed to the logout endpoint in the backend.	- Status code 200 - Returned message: "Logout successful" - Tokens associated with the user in the Token table should get removed.	- Status code 200 - Returned message: "Logout successful" - Tokens associated with the user in the Token table got removed.	P
10	Test reset password route with correct email and strong password	new_password_field= RessetPasswordFields(username='testuser3', email='testuser3@example.com', new_password='NewStrongPassword123$'	- Status code 200 - Returned message: "Your password has been updated." - New raw password should	- Status code 200 - Returned message: "Your password has been updated." - New raw password is similar	P

		)	be similar to the one encrypted in the database.	to the one encrypted in the database.	
11	Test reset password route with correct email and weak password	new_password_field= RessetPasswordFields(username='testuser3', email='testuser3@example.com', new_password='weak')	- Status code 400 - Returned message: "Password isn't strong enough."	- Status code 400 - Returned message: "Password isn't strong enough."	P
12	Test reset password route with incorrect email and strong password	new_password_field= RessetPasswordFields(username='testuser3', email='wrongemail@example.com', new_password='weak')	- Status code 400 - Returned message: "No user with the following email has been found."	- Status code 400 - Returned message: "No user with the following email has been found."	P
13	Test reset password route with incorrect username and strong password	new_password_field= RessetPasswordFields(username='wrongusername', email='testuser3@example.com', new_password='NewStrongPassword123$')	- Status code 400 - Returned message: "No user with the following username matches the account linked to the email address."	- Status code 400 - Returned message: "No user with the following username matches the account linked to the email address."	P
14	Test the list user's route with an admin account	Admin_user = User(username='adminuser', email='admin@example.com', password='StrongPassword123$' role = 'Admin' Is_active = 'True' Is_authenticated = 'True')	- Status code 200 - Response is a list - Length of response > 0 - Returned fields = {username, email, role, is_active, is_authenticated}	- Status code 200 - Response is a list - Length of response > 0 - Returned fields = {username, email, role, is_active, is_authenticated}	P
15	Test the list user's route with a user account	User account created and activated in case 4:	- Status code 401 - Returned message: "Not an Admin thus not authorised." -Length of the response == 0	- Status code 401 - Returned message: "Not an Admin thus not authorised." -Length of the response == 0	P
16	Test update user's detail route with an admin account (username only is modified)	User_mock_data = UserCreate(username='Jake', email='Jake@example.com', password='StrongPassword123$' Id = 15 New fields = UpdateUserDetailsField(user_id = 15, new_username = 'Alain', new_role = 'user')	- Status code 200 - The user's username stored in the database should get updated 'Alain'.'	- Status code 200 - The user's username stored in the database got updated to 'Alain'.''	P

17	Test update user's detail route with an admin account (role only is modified)	User_mock_data = UserCreate(username='Jake', email='Jake@example.com', password='StrongPassword123$' Id = 15 New fields = UpdateUserDetailsField(user_id = 15, new_username = 'Jake', new_role = 'Admin')	- Status code 200 - The role of the user stored in the database should get updated to 'Admin'.	- Status code 200 - The role of the user stored in the database got updated to 'Admin''.	P
18	Test update user's details route with an admin account (username and role modified	User_mock_data = UserCreate(username='Jake', email='Jake@example.com', password='StrongPassword123$' Id = 15 New fields = UpdateUserDetailsField(user_id = 15, new_username = 'Eric', new_role = 'Admin')	- Status code 200 - The user's username stored in the database should get updated 'Eric' - The role of the user stored in the database should get updated to 'Admin'.	- Status code 200 - The user's username stored in the database got get updated to 'Eric' - The role of the user stored in the database got updated to 'Admin''.	P
19	Test update user's details route with a user account (username and role modified	New fields = UpdateUserDetailsField(user_id = 1, new_username = 'Jeff', new_role = 'Admin')	- Status code 401 - Returned message: "Not an Admin thus not authorised."	- Status code 401 - Returned message: "Not an Admin thus not authorised."	P
20	Test block account route with an admin account	Account_to_freeze = FreezeAccountField(user_id=15))	- Status code 200 - user.is_active == False	- Status code 200 - user.is_active == False	P
21	Test unblock account route with an admin account	Account_to_freeze = FreezeAccountField(user_id=15)	- Status code 200 - user.is_active == True	- Status code 200 - user.is_active == True	P
22	Test block account route with a user account	Account_to_freeze = FreezeAccountField(user_id=15)	- Status code 401 - Returned message: "Not an Admin thus not authorised."	- Status code 401 - Returned message: "Not an Admin thus not authorised."	P
23	Test	Account_to_freeze =	- Status code 401	- Status code 401	P

	unblock account route with a user account	FreezeAccountField(user_id=15)	- Returned message: “Not an Admin thus not authorised.”	- Returned message: “Not an Admin thus not authorised.”	
24	Test get clusters Kmeans route	No data required	- Status code 200 - Response: JSON dictionary encoded in the JSON string contains the following keys: {Ticker, Avr Annual Return, Avr Annual Volatility, Cluster}	- Status code 200 - Response: JSON dictionary encoded in the JSON string contains the following keys: {Ticker, Avr Annual Return, Avr Annual Volatility, Cluster}	P
25	Test generate prediction random forest route	No data required	- Status code 200 - JSON response should be either 0 or 1.	- Status code 200 - JSON response should be either 0 or 1.	P
26	Test fetch training data from yahoo finance route	No data required	- Status code 200 - Returned message: “Data has been successfully stored into a CSV file.” - Fields in the CSV file: {Date, Open, High, Low, Close,Volume, Dividends, Stock Splits, Ticker} - The date when the CSV file was updated should get appended at the end of a text file.	- Status code 200 - Returned message: “Data has been successfully stored into a CSV file.” - Fields in the CSV file: {Date, Open, High, Low, Close,Volume, Dividends, Stock Splits, Ticker} - The date when the CSV file was updated is appended at the end of a text file.	P
27	Test update training data set route	No data required	- Status code 200 - Returned message: “Data has been successfully updated and stored in the CSV file.” - The CSV file is sorted according to the ticker symbol and date. - The date when the CSV file was updated should get appended at the end of a text file.	- Status code 200 - Returned message: “Data has been successfully updated and stored in the CSV file.” - The CSV file is sorted according to the ticker symbol and date. - The date when the CSV file was updated is appended at the end of a text file.	P

6.2.3: Automated Test using Jenkins:

To automate both unit and integration testing, Jenkins was utilized. A pipeline was first developed using a Bash shell script to set up the environment, run the necessary Docker Compose file for integration testing, execute the tests, and finally tear down the environment and containers.

From the Jenkins console output, the shell script successfully configured the virtual environment, started the Docker Compose file to set up all required containers, and executed the unit tests. The unit test section reported no errors and indicated that all test cases passed. However, despite the correct installation of the Docker file with all dependencies, the integration testing task failed because Jenkins could not recognize the TestContainers library. Consequently, integration testing was conducted within Visual Studio Code.

Figure 6.2.3.1: Automated Unit test output using Jenkins

6.2 White Box Testing:

Table 6.2.1: White box testing table:

Test No	Test Description	Test Data	Expected Result	Actual Result	Status

1	Create user account with a strong password	- username: BillGates
		- email: w1972584@westminster.ac.uk
		- password: StrongPassword64$
			- Status code: 201
			
			- Returned JSON obj with all of the user's fields stored in the database.
			
			- A notification should pops up on the screen indicating that the user's account has been created and that a verification link has been sent to the email address.
			
			- Account verification link should get sent to the user's email account with an authentication token attached to the link.
				- Status code: 200
				- Returned JSON obj:
				{
				- id: 6,
				- username: BillGate,
				- email: w1972584@westminster.ac.uk
				- role: user,
				- is_active: false,
				- is_authenticated: false,
				- created_at: 2024-0831T19:01:13.307897
				}
				
				- A notification pops up on the screen indicating the user's account creation was successful.
				
				- Account verification link was sent to the email address.
					P

2	Create a user account with incorrect credentials (weak password)	- username: BillGates - email: w1972584@westminster.ac.uk - password:weakPassword	- Status code: 400 - A notification should pop up on the screen, indicating that the account creation operation failed. - No verification link should be sent to the user's email address.	- Status code: 400 - A notification pops up on the screen, indicating that the account creation operation failed. - No verification link is sent to the user's email address	P
3	Reset user's password	- username: BillGates - email: w1972584@westminster.ac.uk - password: NewPassword64$	- Status code: 200 - A notification should pop up on the screen, indicating that the password reset operation was successful.	- Status code: 200 - A notification pops up on the screen, indicating that the password reset operation was successful.	P

4	Reset user's password with an incorrect username and/ or email	- username: JeffBezos - email: w1972584@westminster.ac.uk - password: NewPassword64$ - username: BillGates - email: fake@westminster.ac.uk - password: NewPassword64$	- Status code: 400 - A notification should pop up on the screen, indicating that the password reset operation was unsuccessful because either the email and/or username is incorrect.	- Status code: 400 - A notification pops up on the screen, indicating that the password reset operation was unsuccessful because either the email and/or username is incorrect.	P
5	Login with correct credentials	- email: w1972584@westminster.ac.uk - password: NewPassword64$	- Status code: 200 - Returned JSON object: the generated JWT token associated with the user. - The access key should get attached to the header and is stored in the local storage. - A JavaScript function should fetch the user's role to determine their permissions and stores it in a global state in the Redux store. - The user should be granted access to private pages and redirected to the home page.	- Status code: 200 - The JWT token is returned, comprising an access key, refresh key and expiration time. - Access key gets attached to the header and is stored in the local storage. - The role is fetched from the user and its value is set in a global state in the redux store. - The user is granted access to private pages and is redirected to the home page.	P
6	Login with incorrect credentials	- email: w1972584@westminster.ac.uk - password: weakPassword64$	- Status code: 400 - A notification should pop up on the screen, indicating that either the email or password provided is incorrect. - The user shouldn't get redirected to the private home page.	- Status code: 400 - A notification pops up on the screen, indicating that either the email or password provided is incorrect. - The user doesn't get redirected to the private home page.	P
7	Update the role of a user	- new_username: "Alain", - new_role: "Admin"	- Status code: 200 - A notification should pop up on the screen, indicating the user's details were successfully updated. - The FetchUsers function in the frontend should retrieve and display all current users stored in the database	- Status code: 200 - A notification pops up on the screen, indicating the user's details were successfully updated. - The FetchUsers function in the frontend retrieves and displays all current users stored in the database with their details. The user's updated details are	P

			with their details. The user's updated details should be displayed on the table.	displayed on the table.	
8	Block a user's account	No data required. The admin only requires clicking on the closed lock symbol for the request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - The FetchUsers function in the frontend should retrieve and display all current users stored in the database with their details. The user's is_active status should display as False.	- Status code: 200 - The FetchUsers function in the frontend retrieves and displays all current users stored in the database with their details. The user's is_active status displays False.	P
9	Unblock a user's account	No data required. The admin only requires clicking on the open lock symbol for the request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - The FetchUsers function in the frontend should retrieve and display all current users stored in the database with their details. The user's is_active status should display as True.	- Status code: 200 - The FetchUsers function in the frontend retrieves and displays all current users stored in the database with their details. The user's is_active status displays True.	P
10	Fetch & store financial training data	- No data is required. The only requirement is for the admin to click on the 'FETCH DATA' button in the admin page for the GET request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - A notification should pop up on the screen indicating the training data was successfully fetched and stored.	- Status code: 200 - A notification pops up on the screen indicating the training data was successfully fetched and stored.	P
11	Update the training dataset	- No data is required. The only requirement is for the admin to click on the 'UPDATE DATA' button in the admin page for the GET request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - A notification should pop up on the screen indicating the training data set was successfully updated.	- Status code: 200 - A notification pops up on the screen indicating the training data set was successfully updated.	P

12	Visualise stock clusters	- No data is required. The only requirement is for the user to click the ‘Clusterize Stocks’ button which sends a GET request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - An interactive scatter plot should be displayed, showing all the stocks from the S&P 500 segregated into clusters. - Users should be able to select a stock on the scatter plot to see its ticker symbol, average annual return, and volatility.	- Status code: 200 - An interactive scatter plot is displayed, showing all the stocks from the S&P 500 segregated into clusters. - Users can select a stock on the scatter plot to see its ticker symbol, average annual return, and volatility.	P
12	Generate a forecast for tomorrow’s closing price trend	- No data is required. The only requirement is for the user to click the ‘Forecast Closing Price’ button which sends a GET request to be sent to the appropriate endpoint in the backend.	- Status code: 200 - Based on the returned value, a green upward arrow will be displayed if the closing price is expected to be higher (1), or a red downward arrow if it is expected to be lower (0).	- Status code: 200 - A red downward arrow indicates that tomorrow’s closing price trend is expected to be lower than the previous day.	P

Chapter 7 Conclusion:

7.1 Review of Project Aims and Objectives:

In conclusion, all the project aims and objectives outlined at the beginning were successfully accomplished. The interactive scatter plot was implemented, enabling users to visualise distinct clusters of stocks based on their average annual return and volatility, fulfilling (PA1). The machine learning models were fine-tuned and integrated into a user-friendly interface, providing accurate predictions of the next day’s closing price and effectively indicating bullish or bearish trends, thus achieving (PA2). Additionally, a secure platform was developed with robust JWT authentication and role-based access control,

ensuring the protection of user credentials and the secure management of crucial features, thereby meeting (PA3).

The project also met its objectives by thoroughly researching and implementing suitable machine learning models, processing and updating the training dataset, and ensuring the accuracy and security of the system. The implementation of Docker and Docker Compose further ensured the efficient deployment of the application, fulfilling the technical and security requirements. Overall, the project successfully delivered on its goals, providing a comprehensive solution that met the intended financial analysis and prediction needs.

7.2 Review of Requirements:

7.2.1: Functional Requirements

Regarding the functional requirements defined for this project, all of them outlined were successfully met, with the exception of the last two luxury requirements due to complexity issues. The stock price trend forecasting model (FR1) was developed with strong evaluation metrics, utilising fine-tuning methods such as grid search, time series cross-validation, and random forest. The k-means algorithm (FR2) effectively segregated clusters of stocks from the S&P 500, accurately categorising them based on their average annual return and volatility while also handling potential outliers.

The backend interface (FR3) was successfully implemented to fetch financial data for all S&P 500 stocks, ensuring that the required training dataset was accurately gathered. Additionally, a structured local storage solution (FR4) was provided, organising the data by ticker symbols and dates in ascending order.

For the desirable requirements, the backend logic was developed to ensure that the financial training dataset is updated daily (FR5) and that newly

fetched data is correctly stored (FR6). A text file was also implemented (FR7) to record the date of the last modification to the training dataset, allowing for easy tracking of updates.

However, the luxury functional requirements to provide a stock purchasing simulation service (FR8) and the option for users to follow daily stock price changes via candlestick graphs (FR9) were not implemented due to the added complexity they would introduce to the application. Despite this, the core functionalities of the project were successfully delivered, achieving the main goals of the project.

7.2.2: Non-Functional Requirements:

Regarding all the non-functional requirements listed for the project, they were all successfully implemented, with the exception of the final luxury requirement concerning container orchestration. The essential requirements were fully achieved, including the implementation of User and Token Models and Controllers (NFR1), which handle the backend logic for user credential authentication and JWT access, ensuring the backend API is secure. A suitable relational database was also provided (NFR2), supporting the storage and management of user credentials and JWT tokens, along with the ability to perform CRUD operations.

Additionally, a security service (NFR3) was developed to hash JWT token payloads with an appropriate algorithm, ensuring the security of the tokens. For the desirable requirements, role-based access logic (NFR4) was successfully implemented to manage crucial functionalities related to user account management, and a hashing service (NFR5) was deployed to encrypt user passwords in the database, safeguarding user data.

However, the luxury requirement to implement Kubernetes for container orchestration (NFR6) was not realised due to the added complexity it would introduce.

7.2.3: User Interface Requirements:

All the user interface requirements were successfully met, except for the final luxury requirement. The essential requirements were thoroughly addressed. The web application's interface is fully responsive, ensuring that all rendered components are correctly displayed across different types of devices (UIR1). An interactive scatter plot was implemented, allowing users to click on individual points to display the ticker symbol, average annual return, and volatility, with distinct colours to represent different stock clusters (UIR2). Additionally, a section was created for users to select a stock and run the next day’s closing price trend forecasting model, with a green arrow indicating a bullish trend and a red arrow indicating a bearish trend (UIR3).

Furthermore, a desirable feature was added in the form of a private admin page (UIR4), which ensures that only users with a valid JWT token and admin role can access it. This page allows the admin to view a table of all created accounts, with the ability to update user roles, block, or unblock accounts.

However, the luxury requirement to provide a section where users can view real-time stock prices on a candlestick graph (UIR5) was not implemented.

7.3 Conclusion:

This project was a valuable learning experience that greatly improved my understanding of mathematics, statistics, and machine learning. I gained practical knowledge of using Docker, successfully splitting the application into separate backend, frontend, and database containers. Using Docker volumes was important, as it allowed me to permanently store user information and tokens, which strengthened my skills in handling data storage.

Beyond the machine learning aspects, I also learned a lot about web development. Securing the API was a major challenge, but it helped me become more confident in protecting web applications. Setting up a test environment and creating unit and integration tests improved my skills in

software testing. I also learned about automated testing by building pipelines in Jenkins, which gave me hands-on experience with continuous integration and deployment.

The frontend was the most challenging part of the project for me. Since I'm not very confident with HTML, CSS, and React, I faced some difficulties, but it also pushed me to get better in those areas. Despite these challenges, the project helped me develop a wide range of skills across different areas of software development.

Bibliography:

Bin, S. (2020). *K-means stock clustering analysis based on historical price movements and financial ratios* (Senior thesis, Claremont McKenna College). Scholarship@Claremont. https://scholarship.claremont.edu/cmc_theses/2435

Daume, H. (2023). *A course in machine learning*. Alanna Maldonado.

Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.

Kim, S. (2024). *Mathematical foundations of machine learning* [YouTube channel]. Mississippi State University, Department of Mathematics and Statistics. https://www.youtube.com/@mathtalent

Mueller, A. C., & Guido, S. (2016). *Introduction to machine learning with Python: A guide for data scientists*. O'Reilly Media.

Ridwan, A. F., Subiyanto, & Supian, S. (2021). IDX30 stocks clustering with K-means algorithm based on expected return and value at risk. *International Journal of Quantitative Research and Modeling, 2*(4), 201-208. https://journal.rescollacomm.com/index.php/ijqrm/index

Basak, S., Kar, S., Saha, S., Khaidem, L., & Roy Dey, S. (2019). Predicting the direction of stock market prices using tree-based classifiers. *The North*

American Journal of Economics and Finance, 47, 552-567. https://doi.org/10.1016/j.najef.2018.06.013