### Riphah International University Lahore, Pakistan



**Riphah School of Computing & Innovation**

**Final Year Project**

### PROJECT REPORT (Part-1)

**[STOCK VISION AI]**

Project ID: **[write ID here Issued by FYP Manager]**

##### Project Team

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student Name** | **Student ID** | **Program** | **Contact Number** | **Email Address** |
| ABDUL QUDDOUS | 36514 | BSCS | 03457228219 | imquddous@gmail.com |
| ABDULLAH BIN ATA | 38602 | BSCS | 03160484410 | 38602@students.riphah.edu.pk |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**[Mr. NAEEM ABBAS]**

([Designation])

**Project Report**

**[ STOCKVISION AI ]**

**Change Record**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Author(s)** | **Version** | **Date** | **Notes** | **Supervisor’s Signature** |
|  | 1.0 |  | <Original Draft> |  |
|  |  |  | <Changes Based on Feedback from Supervisor> |  |
|  |  |  | <Changes Based on Feedback From Faculty> |  |
|  |  |  | <Added Project Plan> |  |
|  |  |  | <Changes Based on Feedback from Supervisor> |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |



**APPROVAL**



|  |  |
| --- | --- |
| **PROJECT SUPERVISOR** |  |
| Comments: | |
|  | |
| Name: |  |
| Date: | Signature: |

|  |  |
| --- | --- |
| **PROJECT MANAGER** |  |
| Comments: | |
|  | |
| Date: | Signature: |

|  |  |
| --- | --- |
| **HEAD OF THE DEPARTMENT** |  |
| Comments: | |
|  | |
| Date: | Signature: |

*. . .*

#### Table of Contents

[Dedication iv](#_gjdgxs)

[Acknowledgements v](#_30j0zll)

[Executive Summary vi](#_1fob9te)

[Table of Contents vii](#_3znysh7)

[List of Figures ix](#_2et92p0)

[List of Tables x](#_tyjcwt)

[Chapter 1 1](#_3dy6vkm)

[Introduction 1](#_1t3h5sf)

* 1. [Background 2](#_4d34og8)
  2. [Motivations and Challenges 2](#_2s8eyo1)
  3. [Goals and Objectives 2](#_17dp8vu)
  4. [Literature Review/Existing Solutions 2](#_3rdcrjn)
  5. [Gap Analysis 2](#_26in1rg)
  6. [Proposed Solution 2](#_lnxbz9)
  7. [Project Plan 3](#_35nkun2)
     1. [Work Breakdown Structure 3](#_1ksv4uv)
     2. [Roles & Responsibility Matrix 3](#_44sinio)
     3. [Gantt Chart 3](#_2jxsxqh)
  8. [Report Outline 3](#_z337ya)

[Chapter 2 4](#_3j2qqm3)

[Software Requirement Specifications 4](#_1y810tw)

* 1. [Introduction 5](#_4i7ojhp)
     1. [Purpose 5](#_2xcytpi)
     2. [Document Conventions 5](#_1ci93xb)
     3. [Intended Audience and Reading Suggestions 5](#_3whwml4)
     4. [Product Scope 6](#_2bn6wsx)
     5. [References 6](#_qsh70q)
  2. [Overall Description 6](#_3as4poj)
     1. [Product Perspective 6](#_1pxezwc)
     2. [Product Functions 7](#_49x2ik5)
     3. [User Classes and Characteristics 7](#_2p2csry)
     4. [Operating Environment 7](#_147n2zr)
     5. [Design and Implementation Constraints 7](#_3o7alnk)
     6. [User Documentation 8](#_23ckvvd)
     7. [Assumptions and Dependencies 8](#_ihv636)
  3. [External Interface Requirements 8](#_32hioqz)
     1. [User Interfaces 8](#_1hmsyys)
     2. [Hardware Interfaces 9](#_41mghml)
     3. [Software Interfaces 9](#_2grqrue)
     4. [Communications Interfaces 9](#_vx1227)
  4. [System Features 9](#_3fwokq0)
     1. [System Feature 1 9](#_1v1yuxt)
        1. [Description and Priority 9](#_4f1mdlm)
        2. [Stimulus/Response Sequences 10](#_2u6wntf)
        3. [Functional Requirements 10](#_19c6y18)
     2. System Feature 2 **Error! Bookmark not defined.**
        1. [Description and Priority 10](#_3tbugp1)
        2. [Stimulus/Response Sequences 11](#_28h4qwu)
        3. [Functional Requirements 11](#_nmf14n)
     3. System Feature 3 (and so on) **Error! Bookmark not defined.**
  5. [Other Nonfunctional Requirements 19](#_37m2jsg)
     1. [Performance Requirements 19](#_1mrcu09)
     2. [Safety Requirements 20](#_46r0co2)
     3. [Security Requirements 20](#_2lwamvv)
     4. [Software Quality Attributes 20](#_111kx3o)
     5. [Business Rules 21](#_3l18frh)
  6. [Other Requirements 21](#_206ipza)

[Chapter 3 22](#_4k668n3)

[Use Case Analysis 22](#_2zbgiuw)

* 1. [Use Case Model 23](#_1egqt2p)
  2. [Fully Dressed Use Cases 31](#_3ygebqi)

[Chapter 4 35](#_2dlolyb)

[System Design 35](#_sqyw64)

* 1. [Architecture Diagram 36](#_3cqmetx)
  2. [Domain Model 39](#_1rvwp1q)
  3. [Entity Relationship Diagram with data dictionary 40](#_4bvk7pj)
  4. [Class Diagram 41](#_2r0uhxc)
  5. [Sequence / Collaboration Diagram 42](#_1664s55)
  6. [Operation contracts 48](#_3vac5uf)
  7. [Activity Diagram 48](#_25b2l0r)
  8. [State Transition Diagram 56](#_kgcv8k)
  9. [Component Diagram 57](#_2afmg28)
  10. [Deployment Diagram 57](#_pkwqa1)
  11. [Data Flow diagram [only if structured approach is used - Level 0 and 1] 57](#_kgcv8k)

[Chapter 5 59](#_kgcv8k)

[Implementation 59](#_34g0dwd)

* 1. [Important Flow Control/Pseudo codes 60](#_1jlao46)
  2. [Components, Libraries, Web Services and stubs 60](#_43ky6rz)
  3. [Deployment Environment 60](#_2iq8gzs)
  4. [Tools and Techniques 61](#_xvir7l)
  5. [Best Practices / Coding Standards 61](#_3hv69ve)
  6. [Version Control 61](#_1x0gk37)

[Appendices 62](#_4h042r0)

[Appendix A: Information / Promotional Material 63](#_2w5ecyt)

[Reference and Bibliography 66](#_1baon6m)

Index **Error! Bookmark not defined.**

#### List of Figures

* 1. Caption of first figure of first chapter 6
  2. Caption of second figure of first chapter 7
  3. Caption of first figure of second chapter 14
  4. Caption of second figure of second chapter 22
  5. Caption of third figure of second chapter 26
  6. Caption of first figure of fifth chapter 49
  7. Caption of second figure of fifth chapter 49

#### List of Tables

* 1. label of first table of first chapter 6
  2. label of second table of first chapter 7
  3. label of first table of second chapter 14
  4. label of second table of second chapter 22
  5. label of third table of second chapter 26
  6. label of first table of fifth chapter 49
  7. label of second table of fifth chapter 4

# Chapter 1

## Introduction

**Chapter 1:** Introduction

The identification of intelligent and automated trading techniques that provide successful results is an ongoing challenge in the ever-changing financial market environment. Inaccurate market predictions in the stock market are the result of poor market analysis and trends, which damage both businesses and individuals. As a result, the volume of the stock market is also reduced.

In this project, we propose **StockVision AI**, which merges several deep learning models and financial expertise. The **StockVision AI** initiative, which primarily focuses on NASDAQ-listed stocks, marks a significant change in how we deal with the constantly changing world of stocks. Our project seeks to transform the way we approach investing in the stock market by automating the trading process using enormous deep learning techniques and methodologies.

We combine multiple deep learning models, each with distinct advantages and forecasting abilities, to create accurate predictions of the closing price of different stocks and analysis of the market. We improve the precision of our trading decision-making by this fusion technique in which every model identifies the stock market patterns individually, and when we combine the predictions made on different dimensional analyses of historical market data, the overall results produce more accurate predictions.

The foundation of our project is the valuable storage facility of financial data that carefully acquires, analyses, and utilizes the versatile **yfinance** (Yahoo Finance) library. The essential knowledge needed to successfully navigate the complicated world of stock trading is included in this data, which acts as the lifeblood of the stock trading sector.

We also provide a simple and user-friendly interface. This interface helps users understand what the complicated deep learning model predicts about stocks. With this system, traders can easily see predictions and make better decisions. They can even set rules for the computer to trade automatically based on market conditions and current capital.

We also include the prices of crude oil, the dollar, interest rates, and gold, as these heavily influence the overall market. Historical data analysis shows that both have an inverse relationship with the stock market. Whenever crude oil and gold prices increase, the stock market experiences negative impacts. This is very helpful for the models in the overall analysis of the markets.

The main objective of **StockVision AI** is to automate and improve the trading of NASDAQ stocks, providing investors with a tool that not only analyzes the market but also makes use of deep learning-based prediction models to execute trades precisely. We will provide a user-friendly frontend where traders can easily set up their trades, analyze the market, and make predictions from the models.

#### Background

The stock market is a dynamic and unpredictable domain that significantly impacts global economies. Traditional methods of stock prediction rely heavily on historical data and often fail to capture real-time market sentiment and patterns. With advancements in machine learning, it has become possible to analyze large datasets and predict market trends with greater accuracy. Our project leverages these technologies to fill the gap between traditional analysis and modern data-driven predictions, offering a robust solution for investors seeking to make well-informed decisions in a volatile market environment.

#### Motivations and Challenges

The motivation behind this project is to improve stock price prediction accuracy using machine learning techniques. Traditional methods often struggle to account for the complexities of market behavior and external factors like news sentiment. Challenges include collecting diverse datasets, integrating real-time data, and dealing with market volatility. The aim is to develop a system that offers accurate and reliable predictions to assist investors in making informed decisions.

#### Goals and Objectives

The goal of this project is to develop a machine learning model for accurate stock price prediction. The objectives include collecting relevant financial data, applying appropriate algorithms, and integrating sentiment analysis for improved prediction accuracy. The project also aims to create a user-friendly interface for real-time stock predictions, enabling investors to make better-informed decisions.

#### Literature Review/Existing Solutions

Stock price prediction has evolved from traditional methods like **ARIMA** and time-series models, which rely on historical data, to more advanced machine learning techniques such as **LSTM** and **GRU**. Recent solutions like **Alpaca** and **Trade Ideas** focus on algorithmic trading but often lack sentiment analysis integration. Some platforms, like **Sentient Trader**, include sentiment analysis but still don't fully incorporate real-time data. Our approach combines financial data with real-time sentiment from news and social media to provide more accurate and actionable predictions for investors.

#### Gap Analysis

While existing stock prediction solutions, such as **ARIMA** models and platforms like **Alpaca** and **Trade Ideas**, provide valuable insights based on historical data, they often fail to account for real-time market sentiment. Many models also struggle to incorporate data from multiple sources, such as news and social media, which can significantly impact stock prices. Additionally, these solutions are limited in their ability to adapt to sudden market changes or volatile conditions. Our project addresses these gaps by integrating real-time sentiment analysis and using advanced machine learning models like **BI-LSTM** and **BI-GRU** to improve prediction accuracy and responsiveness to market fluctuations.

#### Proposed Solution

The proposed solution is to develop a stock price prediction system that integrates machine learning models with real-time sentiment analysis. By combining **historical stock data**, **news**, and **social media sentiment**, the system will provide more accurate predictions of stock trends. We will use advanced models like **BI-LSTM** and **BI-GRU** to process time-series data and capture long-term dependencies. Additionally, the system will offer an intuitive user interface for real-time predictions and automated trading execution through APIs like **Alpaca**. This approach aims to help investors make more informed decisions and adapt to market changes effectively.

### ****Phase 1: Requirement Analysis & Planning (October 2024)****

**Tasks:**

* Define system requirements and scope.
* Research UI trends for financial applications.
* Draft initial project documentation.

**Deliverables:**

* Requirements document, system architecture draft.

### ****Phase 2: UI Design & Wireframing (November 2024)****

**Tasks:**

* Create wireframes for the user interface (UI).
* Develop mockups for the web-based dashboard.
* Finalize UX flow for prediction visualization and user interaction.

**Deliverables:**

* UI wireframes, mockups, finalized UX flow.

### ****Phase 3: Frontend Development (December 2024 - January 2025)****

**Tasks:**

* Implement UI components using React.js, and Tailwind CSS.
* Design real-time chart visualizations for stock trends and sentiment analysis.
* Ensure mobile responsiveness and accessibility.

**Deliverables:**

* Interactive stock prediction dashboard.

### ****Phase 4: Documentation Preparation (February 2025)****

**Tasks:**

* Draft technical documentation for frontend development.
* Document API usage (e.g., stock price retrieval, sentiment data integration).

**Deliverables:**

* Technical documentation, API reference guide.

### ****Phase 5: Deployment & Final Documentation (JAN 2025 - FEB 2025)****

**Tasks:**

* Finalize and publish system documentation.
* Deploy the frontend on AWS or a cloud platform.
* Submit the final FYP documentation and user manual.

**Deliverables:**

* Final report, UI/UX documentation, deployment guide.

##### Work Breakdown Structure

The **Work Breakdown Structure (WBS)** breaks the project into smaller, manageable tasks and deliverables. For the **Stock Price Prediction System**, the WBS is divided into several phases, each consisting of specific tasks.

#### 1. Project Management

* **1.1. Requirement Gathering**
  + Define project scope and objectives.
  + Identify data sources and model selection.
* **1.2. Team Formation**
  + Assign roles and responsibilities.
  + Set communication channels and meetings.
* **1.3. Project Planning**
  + Create project timeline and milestones.
  + Set deliverables for each phase.

#### 2. Reports/Documentation

* **2.1. Literature Review**
  + Research existing prediction models and sentiment analysis techniques.
* **2.2. Requirements Document**
  + Detail project requirements, functionalities, and expected outcomes.
* **2.3. Final Report**
  + Summarize results, challenges, and outcomes of the project.

#### 3. System Development

* **3.1. Data Collection**
  + 3.1.1. Collect historical stock price data.
  + 3.1.2. Collect real-time sentiment data (social media, news).
* **3.2. Data Preprocessing**
  + 3.2.1. Clean and format stock price data.
  + 3.2.2. Tokenize and clean sentiment data.
  + 3.2.3. Normalize data for model training.
* **3.3. Model Development**
  + 3.3.1. Develop LSTM model.
  + 3.3.2. Develop GRU model.
  + 3.3.3. Train and evaluate models.
* **3.4. Frontend Development**
  + 3.4.1. Design user interface (React.js).
  + 3.4.2. Display predictions and sentiment trends.
* **3.5. Backend Development**
  + 3.5.1. Develop prediction engine integration.
  + 3.5.2. Set up API connections for data handling.

#### 4. Testing & Deployment

* **4.1. System Testing**
  + 4.1.1. Unit testing for individual modules.
  + 4.1.2. Integration testing for system components.
* **4.2. Deployment**
  + 4.2.1. Deploy system on AWS cloud for scalability.
  + 4.2.2. Ensure system handles real-time data and predictions.
* **4.3. User Testing**
  + 4.3.1. User interface testing.
  + 4.3.2. Evaluate user feedback and adjust.

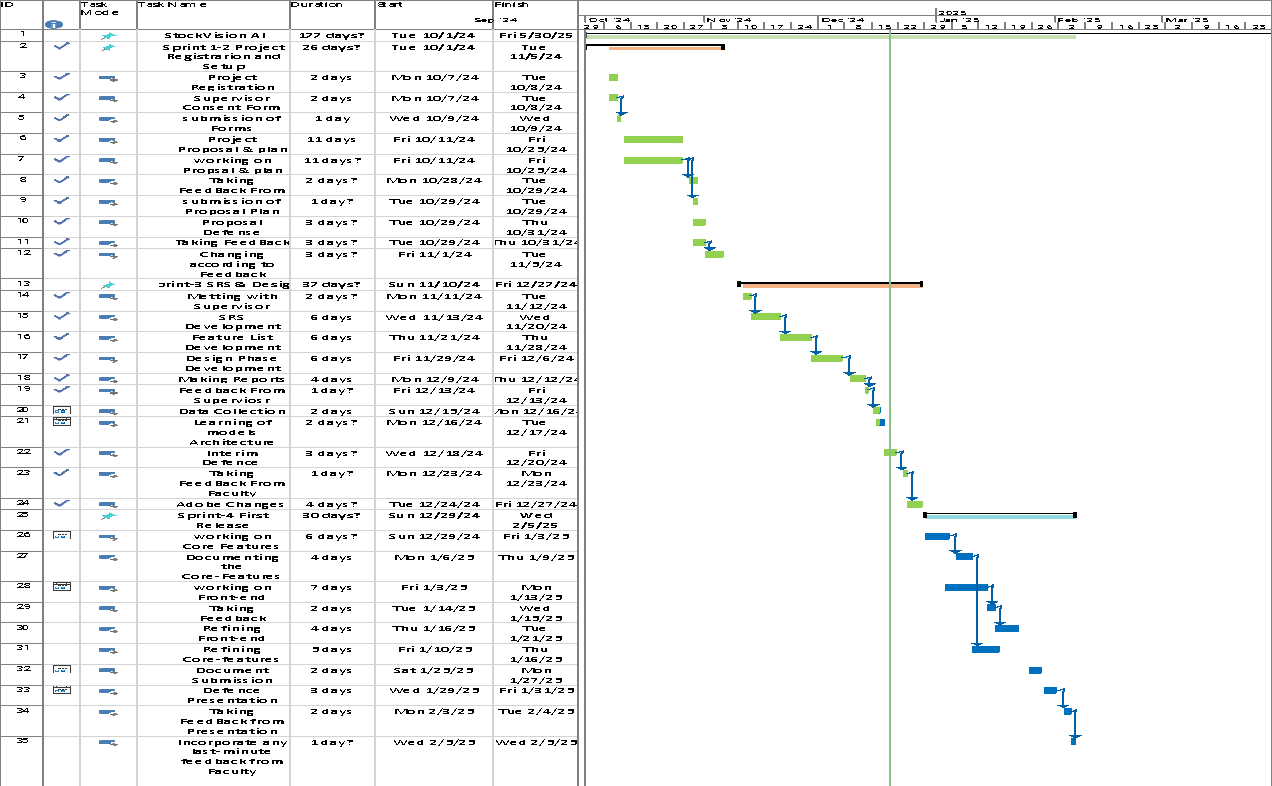
#### 5. Final Review and Documentation

* **5.1. Final Review**
  + Review system performance and results.
  + Conduct final adjustments and improvements.
* **5.2. Documentation**
  + 5.2.1. Document system architecture and setup.
  + 5.2.2. Prepare user guides and reports.

##### Roles & Responsibility Matrix

| **Task** | **Abdull Quddous** | **Abdullah** |
| --- | --- | --- |
| **1. Project Management** |  |  |
| Requirement Gathering | Lead - Define system requirements | Assist with identifying data sources |
| Project Planning | Lead - Create project timeline | Assist with resource planning |
| Team Formation | Assist in role allocation | Lead team communication setup |
| **2. Reports/Documentation** |  |  |
| Literature Review | Research on prediction models | Assist with research on sentiment analysis |
| Requirements Document | Draft system requirements | Review and provide feedback |
| Final Report | Write final report on project outcomes | Assist with final report writing |
| **3. System Development** |  |  |
| Data Collection | Collect stock price data | Collect sentiment data from news/social media |
| Data Preprocessing | Preprocess stock data | Preprocess sentiment data |
| Model Development | Develop LSTM model | Assist with GRU model development |
| Model Training & Evaluation | Train LSTM model and evaluate | Train and evaluate GRU model |
| Frontend Development | Assist in design and integration | Lead frontend development (React.js) |
| Backend Development | Backend development for prediction engine | Assist with backend API integration |
| **4. Testing & Deployment** |  |  |
| System Testing | Lead unit testing and integration testing | Assist with testing and bug fixing |
| Deployment | Deploy system on AWS | Assist with deployment process |
| User Testing | Lead user testing and feedback | Assist with user interface testing |
| **5. Final Review and Documentation** |  |  |
| Final Review | Lead final system review | Assist with final system evaluation |
| Documentation | Lead project documentation | Assist with documentation and reporting |

##### Gantt Chart



# C

# CChapter 2

## Software Requirement

**Chapter 2:** Software Requirement Specifications

#### Introduction

##### Purpose

The purpose of this document is to specify the software requirements for the named "Stock Vision AI", version 1.0. This SRS covers the entire system, including the trading bot’s prediction functionalities, user interface, integration with Alpaca API, and the backend components responsible for real-time sentiment analysis and deep learning-based stock price predictions. The primary scope is stock prediction for NASDAQ markets, focusing on short and medium-term closing prices.

##### Document Conventions

1. **Font Style**: Requirements are listed in **bold** for clarity.
2. **Priority Notation**: Critical requirements are labeled as [High Priority], and optional features are marked as [Optional].
3. **Typographical Standards**: GUI terms are enclosed in quotes (e.g., "Trade Now"), and technical terms are italicized (e.g., *Bi-LSTM*).

##### Intended Audience and Reading Suggestions

1. **Developers**: Focus on Sections 2.2.4 (Operating Environment), 2.3.3 (Software Interfaces), and 2.3.4 (Communications Interfaces).
2. **Project Managers**: Refer to Sections 2.2.1 (Product Perspective) and 2.3.1 (User Interfaces).
3. **Traders and Non-Traders**: Focus on Sections 2.3.1 (User Interfaces) and 2.2.3 (User Classes and Characteristics).
4. **Stock Analysts**: Refer to Section 2.2.2 (Product Functions).
5. **Support Team**: Refer to Section 2.2.6 (User Documentation).

##### Product Scope

**Stock Vision AI** leverages advanced deep learning models (Bi-LSTM and Bi-GRU) combined with sentiment analysis from news and social media to predict stock prices. The system focuses on NASDAQ stocks and provides features like automated trading through Alpaca API, offering accurate, actionable insights for traders. Its user-friendly interface caters to various stakeholder groups, ensuring enhanced decision-making and trading efficiency.

##### References

1. Touzani et al. (2020). *Deep learning models for Moroccan stock market predictions*.
2. Rouf et al. (2019). *Machine learning techniques for stock market evolution analysis*.
3. Corizzo et al. (2022). *Innovative stacking ensemble approach for stock prediction*.

#### Overall Description

##### Product Perspective

StockVision AI is a standalone product that combines traditional stock prediction models with real-time sentiment analysis. Unlike existing systems, it integrates financial data from Alpaca API and social sentiment data, leveraging deep learning for higher accuracy. Its key innovation lies in blending Bi-LSTM and Bi-GRU models to improve both short-term and medium-term predictions.

##### Product Functions

Key features include:

1. **Sentiment Analysis**: Real-time analysis from news, social media, and financial reports.
2. **Stock Predictions**: Short and medium-term NASDAQ stock price forecasting.
3. **Automated Trading**: Execution of predefined trading rules via Alpaca API.
4. **User Interface**: Intuitive interface for manual and automated trading.

##### User Classes and Characteristics

1. **Traders**: High-frequency users engaging with predictions and trades.
2. **Non-Traders**: Occasional users seeking market insights.
3. **Stock Analysts**: Expert users validating model predictions and offering feedback.
4. **Support Team**: Assists users and provides system feedback.
5. **Exchange (Alpaca)**: Facilitates API integration for trading and data access.

##### Operating Environment

1. **Hardware**: Cloud-based infrastructure on AWS.
2. **Software**:
3. Backend: Python (TensorFlow, Keras, Pandas).
4. Frontend: React.js.
5. APIs: Alpaca, yfinance.

##### Design and Implementation Constraints

1. Compliance with Alpaca API guidelines.
2. Use of Bi-LSTM and Bi-GRU models for prediction.
3. Security protocols for trading and user data protection.
4. Hosting limitations based on AWS resources.

##### User Documentation

1. User Manual (web interface and trading operations).
2. Quick Start Guide for setting up automated trading.
3. API Reference for advanced users.

##### Assumptions and Dependencies

1. Reliance on Alpaca API for stock data and trading.
2. Consistent availability of financial news and social media feeds for sentiment analysis.
3. Use of cloud-based infrastructure for scalability.

#### External Interface Requirements

##### User Interfaces

The system provides:

1. A dashboard with real-time stock predictions and sentiment indicators.
2. Manual trading controls (e.g., "Trade Now" button).
3. Automated trading configuration via an intuitive setup page.

##### Hardware Interfaces

1. Compatibility with standard personal computers or mobile devices.
2. Integration with servers hosted on AWS.

##### Software Interfaces

1. **Alpaca API**: Fetches stock data and executes trades.
2. **yfinance**: Provides historical price data.
3. **News APIs**: Supplies sentiment analysis data.

#### System Features

Here is the updated **System Features** section, combining the previously defined and newly provided functional requirements into a comprehensive structure:

##### Stock Prediction

* + - 1. **Description and Priority**

The system predicts stock prices using advanced deep learning models and sentiment analysis.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| **User selects a stock for prediction** | *System retrieves historical and real-time data, processes it,*  *and visualizes the prediction* |
| **User selects a prediction timeframe (e.g., next 5**  **days).** | *System displays predictions for the selected duration* |

* + - 1. **Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF1-1** | *Stock Price prediction must be done in the Trading Bot* |
| **REQ-SF1-2** | *Prediction visualization must be given* |
| **REQ-SF1-3** | *Multiple time frames predictions must be available* |
| **REQ-SF1-4** | *Predictions of multiple stock should be available for the user* |

##### Automated Trading

* + - 1. **Description and Priority**

The system automates trade execution based on user-defined conditions and thresholds.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User specifies trading rules***  ***(e.g., profit/loss thresholds).*** | *System executes trades when conditions are met.* |
| ***User selects stocks and***  ***specifies quantities*** | *System schedules and executes trades accordingly* |

* + - 1. **Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF2-1** | *Enable automated trading based on predefined user rules* |
| **REQ-SF2-2** | *User can select the stock from the list of available stocks* |
| **REQ-SF2-3** | *User can specify the number of stocks* |
| **REQ-SF2-4** | *System shall setup the take profit and stop loss of the trade* |

##### Order Execution

* + - 1. **Description and Priority**

Allows execution of various order types, including market, limit, and stop-loss orders.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User places a market order*** | *System executes the order immediately at the current market*  *price.* |
| ***User sets a limit order.*** | *System executes the order when conditions are met* |

**5.3.3.3 Functional Requirements**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF3-1** | *: The system must be capable of placing market orders, limit orders, and stop-*  *loss orders based on its predefined strategy.* |
| **REQ-SF3-2** | *The system must apply the stop-loss and take profit on limit orders* |

##### Transaction

* + - 1. **Description and Priority**

Manages safe transactions and connects with Alpaca API for seamless trade execution.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User initiates a transaction*** | *System securely processes the transaction* |
| ***User requests transaction***  ***history*** | *System displays transaction logs* |

**5.3.4.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF4-1** | *: The system must be capable of placing market orders, limit orders, and stop-*  *loss orders based on its predefined strategy.* |
| **REQ-SF4-2** | *The system must apply the stop-loss and take profit on limit orders* |

##### User Login

* + - 1. **Description and Priority**

Handles user account login functionality.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User logs in*** | *System verifies and maintains user credentials* |

**5.3.5.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF5-1** | *The web page must allow user to login to the system.* |
| **REQ-SF5-2** | *The website shall check the user credentials from the database.* |
| **REQ-SF5-3** | *The website shall give the module of forget password.* |

##### User Register

* + - 1. **Description and Priority**

Handles user account Register functionality.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User Register in*** | *System verifies and maintains user credentials* |

**5.3.6.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF6-1** | *The web page must allow user to create account.* |
| **REQ-SF6-2** | *The website shall hold and maintain the data of the user* |
| **REQ-SF6-3** | *: User shall connect to the Alpaca account.* |

##### User Logout

* + - 1. **Description and Priority**

Handles user account logout functionality.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User logout in*** | *System verifies and maintains user credentials* |

**5.3.7.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF7-1** | *System should allow the user to successfully log out of their accounts* |
| **REQ-SF7-2** | *System should give the warning of closing the trades before logging out* |

##### Order Cancellation

* + - 1. **Description and Priority**

The system allows users to cancel orders before they are executed. Additionally, the system provides a warning message before performing the cancellation to ensure the user is aware of the action.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User selects an active order and clicks the "Cancel"***  ***button.*** | *System checks if the order is still pending and eligible for cancellation.* |
| ***System determines that the order is eligible for***  ***cancellation.*** | *System displays a warning prompt: "Are you sure you want to cancel this order?"* |
| ***User confirms the***  ***cancellation.*** | *System cancels the order and updates the user's order history.* |
| ***User dismisses the warning***  ***prompt.*** | *System retains the order without making changes.* |

**5.3.8.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF8-1** | *User shall be able to cancel the orders before their execution* |
| **REQ-SF8-2** | *System shall give the warning before performing the cancellation* |

* + 1. **Closing Trades**
       1. **Description and Priority**

The system enables users to close active trades during their execution. Before closing a trade, the system provides a detailed view of the current profit or loss, ensuring that users are well-informed before finalizing the action.

.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User selects an active trade and clicks the "Close Trade"***  ***button*** | *System fetches the real-time profit or loss for the selected trade* |
| ***System displays the profit***  ***or loss summary to the user.*** | *User reviews the summary* |
| ***User confirms the trade***  ***closure*** | *System closes the trade and updates the user's trade history* |
| ***User cancels the closure***  ***action*** | *System retains the trade in its active state* |

**5.3.9.3 Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF9-1** | *User shall be able to close the trades during their execution* |
| **REQ-SF9-2** | *System shall give the warning before performing the cancellation* |

* + 1. **Manual Trades**
       1. **Description and Priority**

Provides functionality for users to execute trades manually.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User sets up a manual***  ***trade.*** | *System executes the trade based on user inputs* |
| ***User sets stop-loss and***  ***take-profit thresholds.*** | *System applies thresholds to the trade* |

* + - 1. **Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF10-1** | *User must be able to setup the manual trade* |
| **REQ-SF10-2** | *User must be able to setup the stop loss and take profit of the trade order.* |
| **REQ-SF10-3** | *User can select the specific stock and the amount of stock* |
| **REQ-SF10-4** | *User must be able to select the buy and sell direction of the trade order*  *manually* |

#### Alerts and Notification

* + - 1. **Description and Priority**

Sends alerts and notifications when stock price thresholds or conditions are met.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User sets stock price***  ***threshold alerts.*** | *System sends notifications when conditions are met* |

* + - 1. **Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF11-1** | *It shall allow users to set alerts and receive notifications when certain stock price*  *thresholds or conditions are met* |

#### Trade Tracking

* + - 1. **Description and Priority**

Enables users to track their trades and view profit/loss summaries.

* + - 1. **Stimulus/Response Sequences**

|  |  |
| --- | --- |
| ***Stimulus*** | ***Response*** |
| ***User requests trade***  ***tracking*** | *System displays executed trades and their outcomes* |

* + - 1. **Functional Requirement**

|  |  |
| --- | --- |
| **ID** | **Functional Requirement** |
| **REQ-SF12-1** | *User shall be able to track the executed trades* |
| **REQ-SF12-1** | *Display profit and loss details for each trade* |

#### Other Nonfunctional Requirements

* + 1. **Performance Requirements**

|  |  |
| --- | --- |
| **ID** | **Non-Functional Requirement** |
| **REQ-N*F*1-*1*** | *The system is required to efficiently process predictions and execute*  *corresponding orders within a strict time frame, ensuring timely and accurate decision-making to meet real-time operational demands.* |
| **REQ-N*F*1-2** | *Handle at least 100 simultaneous users without performance degradation* |

* + 1. **Safety Requirements**

|  |  |
| --- | --- |
| **ID** | **Non-Functional Requirement** |
| **REQ-N*F*2-*1*** | *Ensure all transactions and data exchanges are encrypted* |
| **REQ-N*F*2-2** | *Warn users about potential financial risks associated with automated trading* |

* + 1. **Security Requirements**

|  |  |
| --- | --- |
| **ID** | **Non-Functional Requirement** |
| **REQ-N*F*3-*1*** | *Implement a simple username and password system to verify users.* |
| **REQ-N*F*3-2** | *Use a secure encryption method, such as hashing sensitive data, to protect user*  *information.* |

* + 1. **Software Quality Attributes**

|  |  |
| --- | --- |
| **ID** | **Non-Functional Requirement** |
| **REQ-N*F*4-*1*** | *The application shall have a straightforward, basic style that makes it very*  *simple to explore all the choices and put them to use.* |
| **REQ-N*F*4-2** | *The system must deliver exceptional reliability, guaranteeing continuous*  *operation with minimal interruptions, achieving near-perfect availability.* |
| **REQ-N*F*4-3** | *Any Web Browser will be supported by our modules and interface shall be*  *responsive giving a portable structure.* |

* + 1. **Business Rules**

|  |  |
| --- | --- |
| **ID** | **Non-Functional Requirement** |
| **REQ-N*F*5-*1*** | *Only authenticated users can execute trades.* |
| **REQ-N*F*5-2** | *All trades must comply with the Alpaca API terms and conditions* |
| **REQ-N*F*5-3** | *User information must be correct before trading* |

#### Other Requirements

##### Database Requirements

* The system must use a reliable database, such as PostgreSQL or MongoDB, to store and manage data.
* All data should support indexing for faster queries.
* Regular backups must be maintained to prevent data loss.

##### Internationalization Requirements

* The platform should support multiple languages, starting with English and, ability to add more languages easily.
* All date, time, and currency formats must adapt to the user’s locale settings.

##### Reuse Objectives

* Develop modular components to facilitate reuse in future projects.
* Design the architecture to allow integration with other systems or APIs.

##### Performance Requirements

* The system should handle a minimum simultaneous user without degradation in performance.

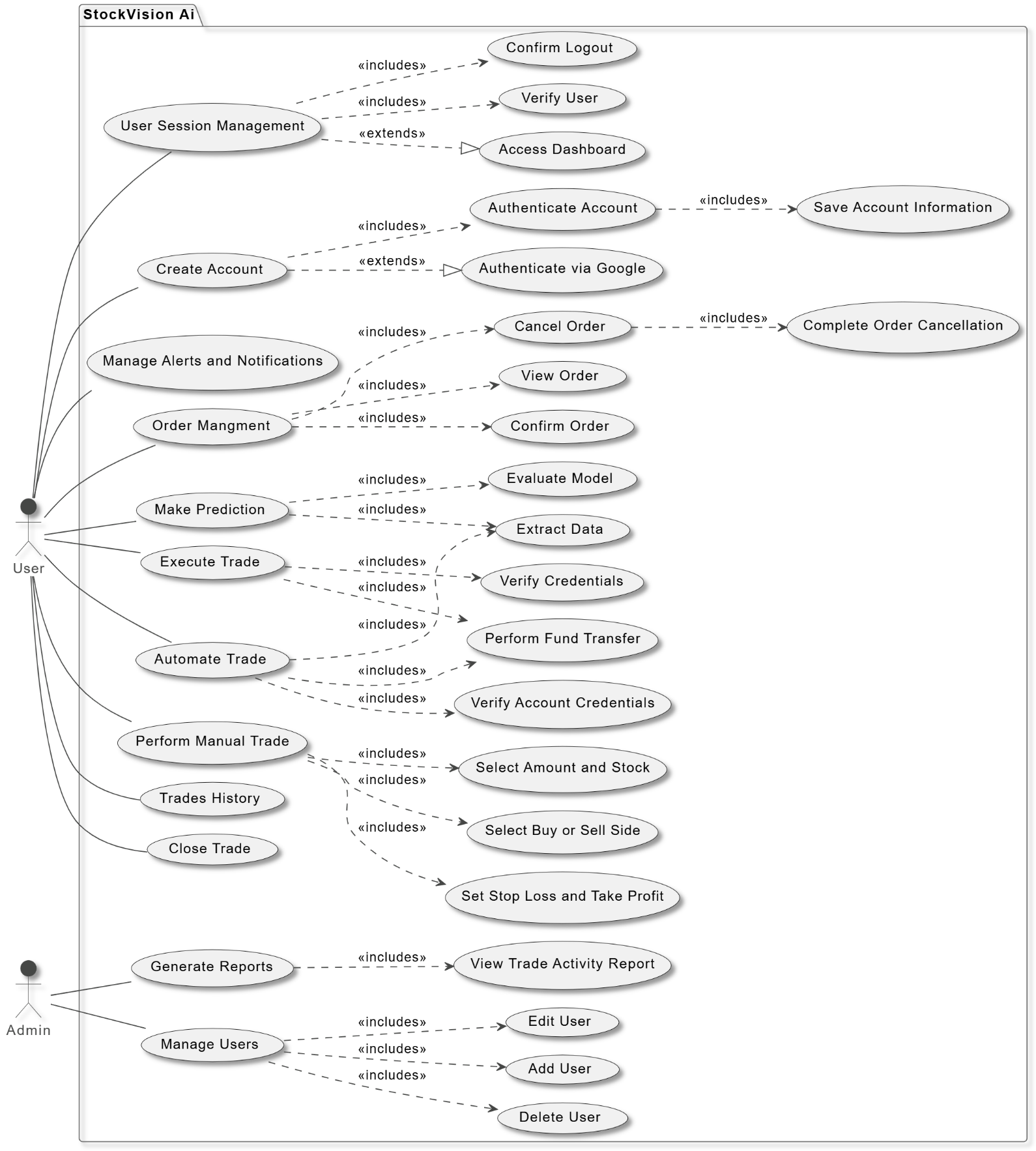
# CChapter 3

## Use Case Analysis

### Chapter 3: System Analysis

This chapter delves into the analysis of the **StockVision AI** system, highlighting its structure, interactions, and functional requirements. It provides a Use Case Model to outline the interaction between users and system components, along with Fully Dressed Use Cases to describe the step-by-step process of each feature. The chapter ensures a comprehensive understanding of system behavior, covering stock prediction, sentiment analysis, and automated trading functionalities.

#### Use Case Model



#### 

#### Fully Dressed Use Cases

##### Introduction:

"This section provides detailed descriptions of each use case identified in the Use Case Model. Each use case is described in terms of actors, description, preconditions and successful completion."

Fully Dressed Use Case: Sign Up

**Use Case Name:** Sign Up

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. User is not logged in. 2. User has access to the registration interface. |
| **Main Success Scenario** | 1. The user accesses the "Sign Up" page. 2. The user enters their details (e.g., name, email, password). 3. The system validates the input fields. 4. The system includes the **Authentication** use case to verify credentials. 5. If validation is successful, the system saves the information. 6. The system displays a confirmation message and redirects the user to the Dashboard. |
| **Postconditions** | User is registered and redirected to the Dashboard. |
| **Basic Flow** | 1. The user accesses the "Sign Up" page. 2. The user enters required details (e.g., name, email, password). 3. The system validates the input fields. 4. The system checks if the email is unique. 5. The system saves the user's information in the database. 6. The user receives a confirmation and is redirected to the dashboard. |
| **Alternative Flow** | * At Step 3: If any input is invalid, the system highlights the error and prompts for correction. * At Step 4: If the email is already in use, the system notifies the user and suggests using a different email. |

Fully Dressed Use Case: Login

**Use Case Name:** Login

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user is registered. |
| **Main Success Scenario** | 1. The user enters their login credentials. 2. The system validates the credentials (includes **Authentication** use case). 3. If validation succeeds, the user is redirected to the Dashboard. |
| **Postconditions** | User gains access to the Dashboard |
| **Basic Flow** | 1. The user enters their credentials on the login page. 2. The system validates the credentials using the **Authentication** use case. 3. Upon successful validation, the user is redirected to the dashboard. |
| **Alternative Flow** | * At Step 2: If the credentials are incorrect, the system displays an error message and allows the user to re-enter credentials. |

Fully Dressed Use Case: Authentication

**Use Case Name:** Authentication

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Level** | Sub-function of "Sign Up" and "Login" |
| **Pre-condition** | 1. The user has initiated a Sign-Up or Login request. 2. The system has the user's input credentials. |
| **Main Success Scenario** | 1. The system receives user credentials. 2. The system verifies the credentials against stored data. 3. If successful, the system proceeds to the next step in the Sign-Up or Login process. |
| **Postconditions** | The system authenticates the user and provides access to the requested functionality |
| **Basic Flow** | 1. The system receives user credentials. 2. The system verifies the credentials against the database. 3. If valid, the user is authenticated, and access is granted. |
| **Alternative Flow** | * At Step 2: If the credentials do not match, the system rejects the authentication and prompts the user to retry. |

Fully Dressed Use Case: Dashboard

**Use Case Name:** Dashboard

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | User must be logged in |
| **Main Success Scenario** | 1. The user logs in successfully. 2. The system loads personalized account information. 3. The user can navigate the platform. |
| **Postconditions** | The user can access their account details and perform actions |
| **Basic Flow** | 1. The user logs in successfully. 2. The system loads user-specific data (e.g., account details, portfolio). 3. The user can navigate through dashboard features like stock analysis and predictions. |
| **Alternative Flow** | * At Step 2: If data fails to load, the system displays a retry option or fallback content |

Fully Dressed Use Case: Google Authentication

**Use Case Name:** Google Authentication

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user has selected the "Sign Up with Google" option. |
| **Level** | Sub-function of "Sign Up" |
| **Main Success Scenario** | 1. The user clicks "Sign Up with Google." 2. The system redirects the user to Google's authentication page. 3. The user provides Google credentials. 4. The system includes the **Authentication** use case to verify Google credentials. 5. Upon success, the system saves the user’s Google account information. |
| **Postconditions** | The user's Google account is linked, and they are registered on the platform. |
| **Basic Flow** | 1. The user selects "Sign Up with Google." 2. The system redirects to Google’s authentication page. 3. The user logs in with Google credentials. 4. The system retrieves and verifies the user’s Google account information. 5. The user’s account is created, and they are redirected to the dashboard. |
| **Alternative Flow** | 1. At Step 3: If Google login fails, the system notifies the user and prompts them to retry. 2. At Step 4: If the system cannot retrieve Google account information, it displays an error message |

Fully Dressed Use Case: Stock Price Prediction

**Use Case Name:** Make Prediction

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user must have access to the Prediction Page. |
| **Main Success Scenario** | 1. The user selects the "Stock Price Prediction" option. 2. The system includes the **Data Extraction** and **Model Evaluation** use cases. 3. The system displays the predicted stock price. |
| **Postconditions** | The predicted stock price is displayed |
| **Basic Flow** | 1. The user selects a stock for prediction. 2. The system triggers the **Data Extraction** and **Model Evaluation** use cases. 3. The system processes the data through the model and generates a stock price prediction. 4. The predicted stock price is displayed to the user. |
| 1. **Alternative Flow** | 1. At Step 2: If data extraction fails, the system notifies the user and suggests retrying. 2. At Step 3: If the model fails to evaluate, the system logs the issue and notifies the user. |

Fully Dressed Use Case: Fund Transfer

**Use Case Name**: Perform Fund Transfer

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user must have sufficient balance |
| **Level** | Sub-function of "Sign Up" |
| **Main Success Scenario** | 1. The user initiates a fund transfer. 2. The system validates the user’s balance. 3. The system processes the transfer. 4. The system displays a success message. |
| **Postconditions** | Funds are successfully transferred. |
| **Basic Flow** | 1. The user initiates a fund transfer request. 2. The system checks the user's account balance. 3. The system processes the transfer and updates the account details. 4. The user receives a success notification. |
| **Alternative Flow** | 1. At Step 2: If the balance is insufficient, the system notifies the user and cancels the transfer. 2. At Step 3: If the transfer fails due to a network or system error, the system logs the failure and prompts the user to retry. |

Fully Dressed Use Case: Data Extraction

**Use Case Name**: Extract Data

|  |  |
| --- | --- |
| **Actor** | **System in case of automated extraction** |
| **Pre-condition** | 1. The system has access to the stock data source (e.g., API or database). 2. The user has initiated a stock price prediction request. |
| **Main Success Scenario** | 1. The user selects the stock for which they want a price prediction. 2. The system connects to the stock data source (e.g., external API or database). 3. The system retrieves the relevant stock data (e.g., historical prices, trading volume, etc.). 4. The system validates and cleans the extracted data. 5. The system prepares the data for input into the machine learning model. |
| **Postconditions** | Required stock data is successfully gathered and ready for model evaluation. |
| **Basic Flow** | 1. The system connects to the stock data source. 2. The system retrieves relevant stock data (e.g., historical prices). 3. The system validates and cleans the extracted data. 4. The system prepares the data for model evaluation. |
| **Alternative Flow** | 1. At Step 1: If the system cannot connect to the data source, it retries or notifies the user. 2. At Step 3: If the data is incomplete or invalid, the system logs the issue and attempts to clean or request alternative data. |

Fully Dressed Use Case: Model Evaluation

**Use Case Name**: Evaluate Model

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Level** | Sub-function of Prediction |
| **Pre-condition** | 1. Required stock data has been successfully extracted and preprocessed. 2. The system has access to a trained machine learning model. |
| **Main Success Scenario** | 1. The system receives the preprocessed stock data from the "Data Extraction" use case. 2. The system loads the trained machine learning model. 3. The system inputs the stock data into the model. 4. The system evaluates the model to generate the predicted stock price. 5. The system returns the predicted stock price to the user. |
| **Postconditions** | The system outputs the predicted stock price based on the evaluated model |
| **Basic Flow** | 1. The system receives preprocessed stock data. 2. The system loads the trained machine learning model. 3. The stock data is input into the model. 4. The model generates a stock price prediction. 5. The system displays the prediction to the user. |
| **Alternative Flow** | 1. At Step 2: If the model is unavailable, the system notifies the user and logs the issue. 2. At Step 4: If the evaluation fails, the system displays an error message and suggests retrying or using different data. |

Fully Dressed Use Case: Credential verification

**Use Case Name**: Verify Credential

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user must attempt to log in or access a sensitive feature requiring authentication |
| **Main Success Scenario** | 1. User provides valid credentials (e.g., username and password). 2. System successfully validates the credentials. 3. User is authenticated and granted access to the requested operation or feature. |
| **Postconditions** | The user is authenticated and allowed to proceed with the requested operation |
| **Basic Flow** | 1. User provides valid credentials. 2. System validates the credentials successfully. 3. Access to requested feature or operation is granted. |
| **Alternative Flow** | 1. System detects invalid credentials. 2. User receives an error message and is prompted to retry or recover the account. |

Fully Dressed Use Case: Execute Trade

**Use Case Name**: Execute Trade

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must be authenticated. 2. Required funds or stocks must be available for the trade. |
| **Main Success Scenario** | 1. User selects "Execute Trade". 2. User enters valid trade details (e.g., stock name, quantity, buy/sell). 3. System verifies fund/stock availability and validates inputs. 4. Trade is successfully executed, and the user's portfolio is updated. |
| **Postconditions** | The trade is executed, and the system records the transaction in the user's trade history. |
| **Basic Flow** | 1. User selects "Execute Trade". 2. User enters trade details (e.g., stock, quantity, type). 3. System validates inputs and fund availability. 4. Trade is executed, and the portfolio is updated**.** |
| **Alternative Flow** | 1. System notifies the user of insufficient resources. 2. Trade execution is canceled. |

Fully Dressed Use Case: Set Stop/loss

**Use Case Name**: Set Stop/loss And Take Profit

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must have an active trade. 2. The user must be logged in. |
| **Main Success Scenario** | 1. User selects the "Set Stoploss" option for an active trade. 2. User enters a valid stop-loss value. 3. System confirms the setup and begins monitoring market conditions. 4. If the stop-loss condition is met, the system automatically closes the trade. |
| **Postconditions** | The stop-loss is successfully set, and the system updates the trade with the configured parameter. |
| **Basic Flow** | 1. User selects "Set StopLoss". 2. User enters the stop-loss value. 3. System confirms the stop-loss setup and starts monitoring. |
| **Alternative Flow** | System identifies an invalid stop-loss value (e.g., higher than the current market value).  User is prompted to re-enter a valid value. |

Fully Dressed Use Case: Buy/sell

**Use Case Name**: Buy/sell

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must be logged in. 2. The user must have sufficient funds or stocks to perform the trade. |
| **Main Success Scenario** | 1. User selects "Buy/Sell" from the dashboard. 2. User specifies the stock, action type (buy/sell), and details (quantity, price). 3. System successfully validates inputs and processes the transaction. 4. Transaction is executed, and the portfolio and trade history are updated. |
| **Postconditions** | The transaction is completed, and the system updates the user's portfolio and trade history. |
| **Basic Flow** | 1. User selects "Buy/Sell". 2. User specifies stock details and action type. 3. System validates inputs and executes the transaction. |
| **Alternative Flow** | 1. System notifies the user of insufficient resources. 2. Transaction is terminated. |

Fully Dressed Use Case: Stock Selection

**Use Case Name**: Stock Selection

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | The user must be logged in. |
| **Main Success Scenario** | 1. User navigates to the stock selection feature. 2. User applies search filters (e.g., by name, price, category). 3. System retrieves matching stock options and displays them. 4. User selects a stock for further action (e.g., trading). |
| **Postconditions** | The selected stock details are displayed for further action. |
| **Basic Flow** | 1. User navigates to stock selection. 2. User applies search filters (e.g., by name, price, category). 3. System retrieves and displays matching stock options. |
| **Alternative Flow** | 1. System identifies no stocks matching the filters. 2. System notifies the user and allows filter modification. |

Fully Dressed Use Case: Perform Manual Trading

**Use Case Name**: Perform Manual Trading

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must have selected a stock and defined trade details. |
| **Main Success Scenario** | 1. User selects the manual trade option. 2. User enters valid trade details (e.g., stock type, quantity, price). 3. System validates inputs and executes the trade. 4. User's portfolio and trade history are successfully updated. |
| **Postconditions** | The manual trade is successfully executed and recorded. |
| **Basic Flow** | 1. User selects manual trade and enters trade details. 2. System validates inputs and executes the trade. 3. Portfolio and trade history are updated. |
| **Alternative Flow** | 1. System detects errors in trade inputs (e.g., price mismatch, quantity too high). 2. User is prompted to correct the errors before retrying. |

Fully Dressed Use Case: Perform Automated Trading

**Use Case Name**: Performed Automating Trading

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must have predefined rules for automated trading. 2. The system must have access to real-time market data. |
| **Main Success Scenario** | 1. System monitors market conditions based on predefined trading rules. 2. Market conditions match the user's defined criteria. 3. System automatically executes the trade. 4. Trade is recorded in the user's history, and the portfolio is updated. |
| **Postconditions** | 1. The automated trade is executed, and the user receives a notification. |
| **Basic Flow** | 1. System monitors market conditions. 2. Predefined rules match market conditions. 3. Trade is executed automatically and logged in the system**.** |
| **Alternative Flow** | 1. System encounters issues retrieving market data. 2. Trade execution is delayed until data access is restored. |

Fully Dressed Use Case: Trade History

**Use Case Name**: Trade History

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must be logged in. |
| **Main Success Scenario** | 1. User selects "Trade History" from the dashboard. 2. System retrieves the user's trade records. 3. Trade history is displayed, and the user applies filters or sorting options as needed. |
| **Postconditions** | 1. The trade history is displayed, providing insights into past trading activity |
| **Basic Flow** | 1. User selects "Trade History". 2. System retrieves and displays the user's trade records. 3. User applies filters or sorting options to refine the display. |
| **Alternative Flow** | 1. System identifies no available trade history for the user. 2. User is notified and offered guidance for placing trades. |

Fully Dressed Use Case: Order Mangment / Cancel order / Confirm order

**Use Case Name**: Order Mangment

|  |  |
| --- | --- |
| **Actor** | **User** |
| **Pre-condition** | 1. The user must have pending or active orders. |
| **Main Success Scenario** | 1. User navigates to the "Order Management" feature. 2. User views order and chooses to cancel or confirm an order. 3. System successfully updates the status of the selected order. |
| **Postconditions** | Order statuses are updated, reflecting the user's actions. |
| **Basic Flow** | 1. User navigates to "Order Management". 2. User views, cancels, or confirms orders. 3. System updates order status accordingly. |
| **Alternative Flow** | 1. System detects an issue (e.g., order already processed). 2. User is notified, and no changes are made to the order. |

# CChapter 4

## System Design

**Chapter 4:**

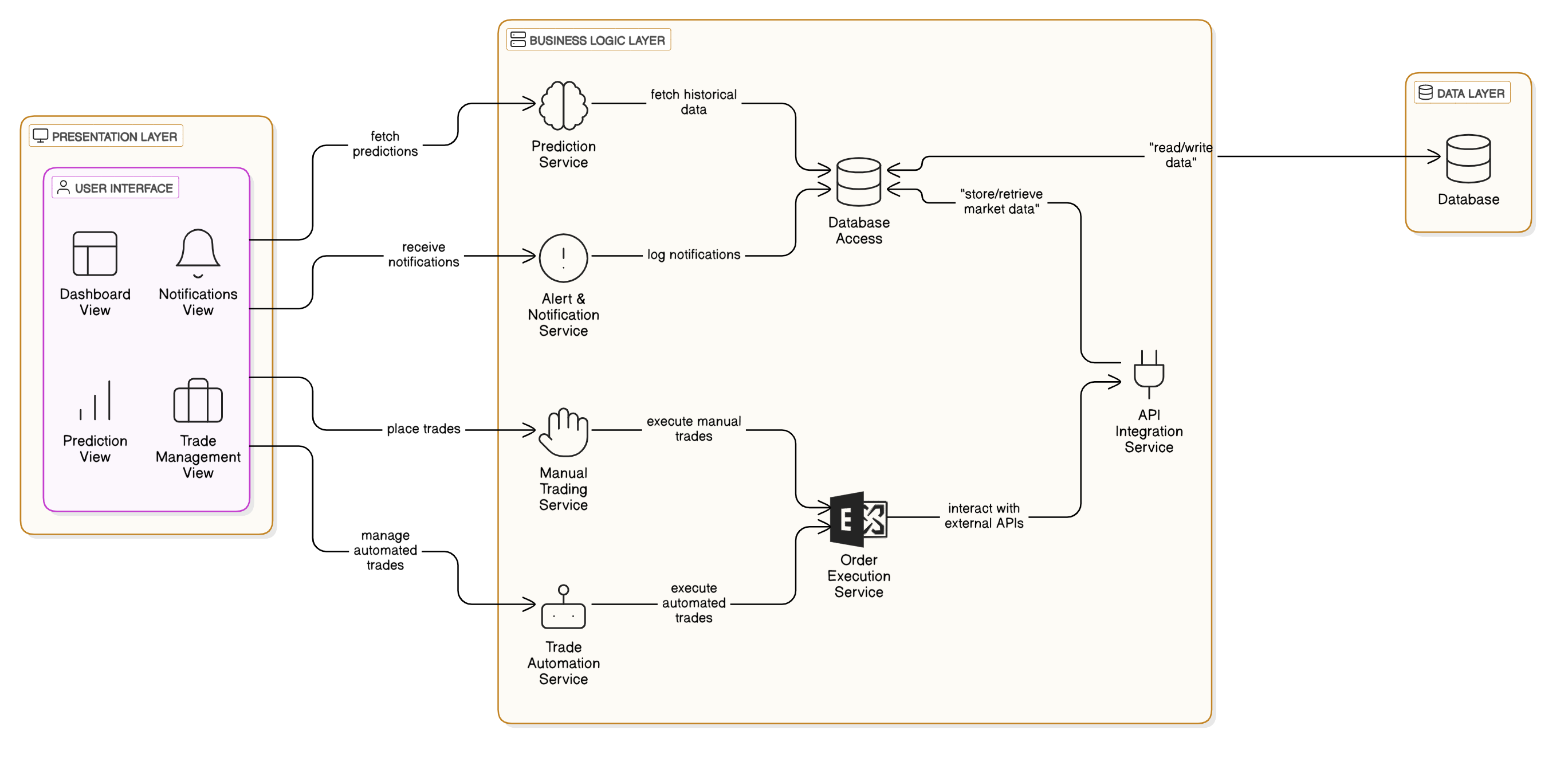
System Design

System Design focuses on outlining the architecture, components, and data flow of the **StockVision AI** system. It describes how the system's components interact to achieve stock prediction, sentiment analysis, and automated trading. The chapter includes architectural diagrams, data flow representations, and design decisions, ensuring a scalable and efficient implementation.

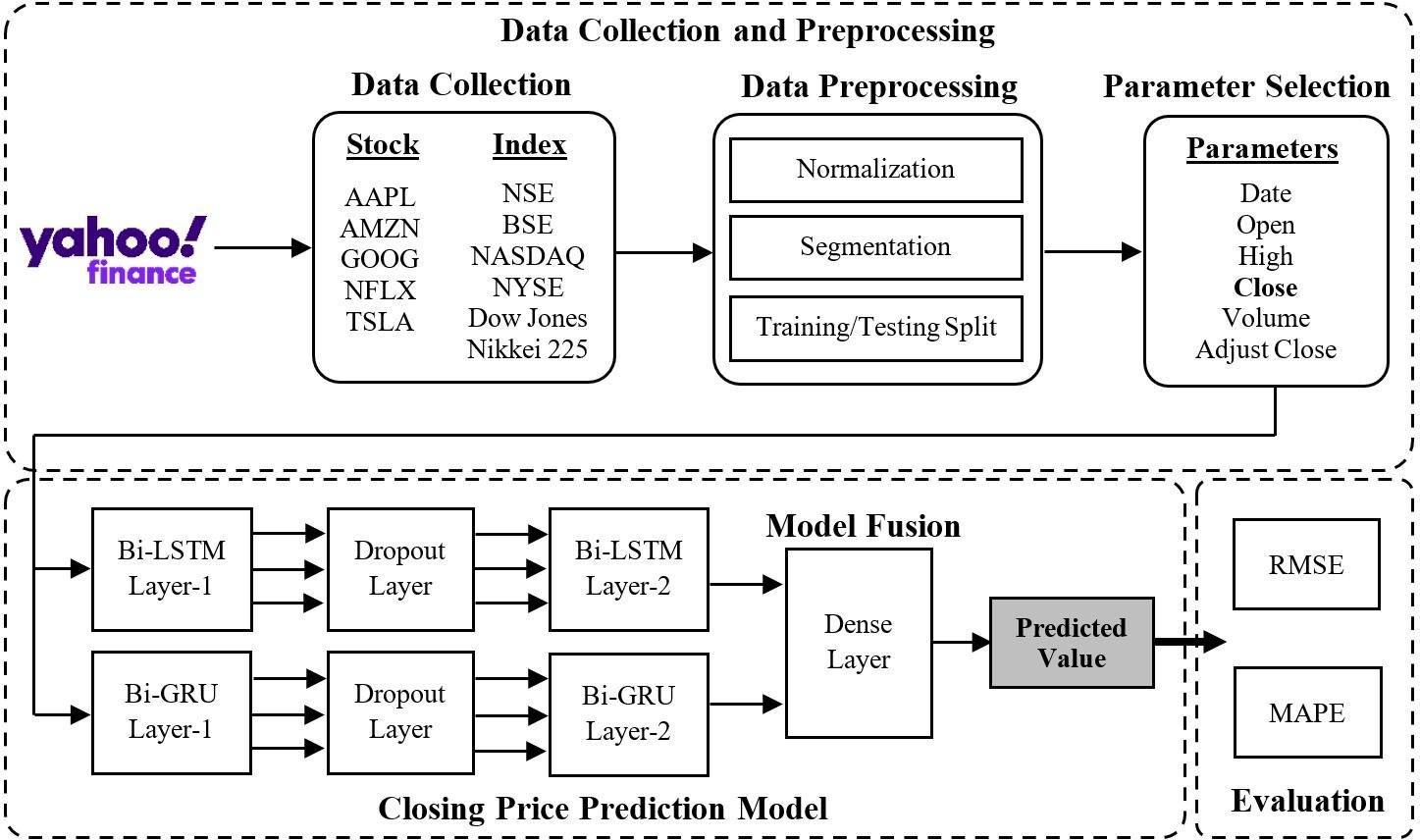
=

#### Architecture Diagram

X`



**Data-Driven Machine Learning Pipeline**



##### Data Collection and Data Preprocessing

First, we collected data from multiple reliable sources to make sure a comprehensive coverage of the factors influencing stock market movements. By utilizing the yfinance and finnhub and Alpaca, we were able to acquire both historical and real-time data on companies listed on the NASDAQ, which allowed us to accurately identify trends and patterns in the market.

After data collection, the data is processed and is ready for model input. To address any flaws or missing numbers, we checked the data thoroughly and if required doing data cleaning that guaranteed the accuracy and reliability of our dataset. We do feature scaling and normalization to normalize the data across many variables making model training and interpretation easier.

##### Models Implementation

The proposed model combines two powerful techniques—**Bi-LSTM** and **Bi-GRU**—to improve prediction accuracy.

* + - 1. **Input Data**: The data is fed into both **Bi-LSTM** and **Bi-GRU** layers at the same time.
         * **Bi-LSTM** (Bidirectional Long Short-Term Memory): Helps remember long-term patterns in the data.
         * **Bi-GRU** (Bidirectional Gated Recurrent Unit): Focuses on short-term patterns and is faster to compute.
      2. **Bidirectional Learning**: Both layers process the data in two directions:
         * From **past to future** and
         * From **future to past**, allowing the model to learn better patterns.
      3. **Dropout Layer**: To prevent the model from overfitting (memorizing the data instead of learning patterns), a **dropout layer** is added, which randomly ignores some connections during training. The dropout rate is set to **20%**.
      4. **Fusion of Outputs**: The outputs from the **Bi-LSTM** and **Bi-GRU** layers are combined (concatenated) to capture diverse patterns from both layers.
      5. **Final Dense Layer**: The combined output is passed to a **dense layer**, which processes the information and predicts the **next day’s closing price**.

##### Sandbox Environment

This system helps users test their **stock trading strategies** and **predict next-day closing prices** for popular stocks like **Tesla (TSLA)**, **Amazon (AMZN)**, **Google (GOOG)**, **Apple (AAPL)**, and **Microsoft (MSFT)** all without risking real money.

**How It Works:**

* + - 1. **Sandbox Environment**:
         * It’s a **virtual platform** where users can simulate (pretend) trading.
         * Users connect their accounts to this sandbox, which uses **Alpaca’s API** to access fake trading features.
      2. **What Users Can Do**:
         * **Manual Trading**: Trade manually based on the predictions provided by the system.
         * **Automated Trading**: Use the **trading bot** to make automatic trades for them.
      3. **Why It’s Useful**:
         * Users can **try out different strategies** without losing real money.
         * They can check how accurate the system’s predictions are.
         * They can also test how well the **trading bot performs**.
      4. **Build Confidence**:

Before switching to real trading, users can refine their strategies, improve their decision- making, and gain confidence in their approach.

##### Frontend Development and Deployment

For the frontend development, we used **React.js** and **Tailwind CSS**. React.js was chosen because of its flexibility and efficient component-based architecture, making it ideal for building dynamic user interfaces. **Tailwind CSS** was used to quickly and easily design modern, responsive interfaces. On the backend, we used **Django** to ensure scalability, stability, and robust

functionality for managing the application’s data and logic

The project was first deployed on **Vercel** due to its ease of use and smooth interface with React.js; however, it was eventually moved to **AWS** to allow for more control and flexibility in resource management and scaling. By combining

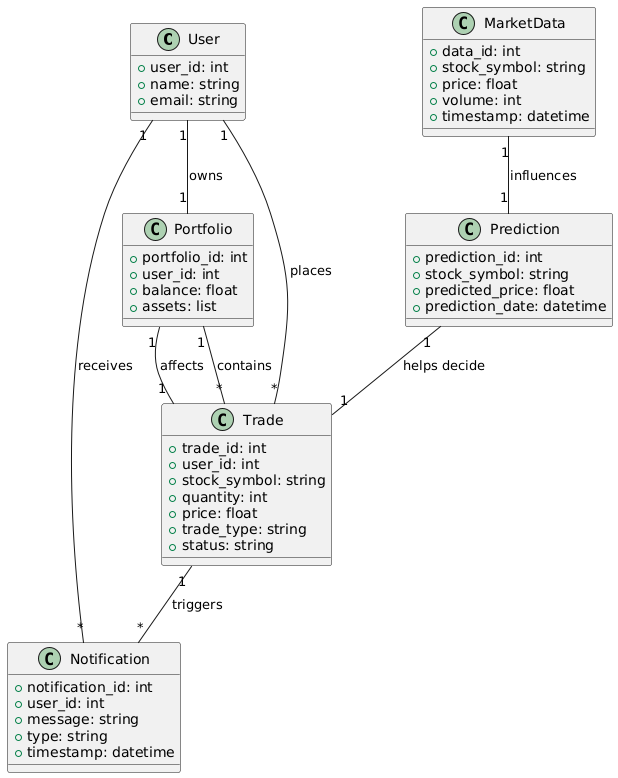
these technologies, it was possible to guarantee quick deployment and scalability on cloud infrastructure and to create an interface that was both responsive and easy to use.

#### Domain Model

##### Key Entities and Relationships

1. **Trader/User**:
   * Attributes: Name, Email, Preferences, Account Balance.
   * Relationships:
     + Can view **Stock Predictions**.
     + Can execute **Trades**.
     + Interacts with the **User Interface**.
2. **Stock**:
   * Attributes: Ticker Symbol, Name, Current Price, Historical Prices, Volatility.
   * Relationships:
     + Has associated **Sentiment Data**.
     + Is processed by the **Prediction Engine**.
3. **Sentiment Data**:
   * Attributes: Sentiment Score, News Source, Social Media Mentions.
   * Relationships:
     + Collected from **News and Social Media APIs**.
     + Merged with **Stock** data for prediction.
4. **Prediction Engine**:
   * Attributes: Model Type (LSTM, GRU), Training Data, Accuracy.
   * Relationships:
     + Consumes **Stock** and **Sentiment Data**.
     + Provides results to the **User Interface** and **Trading Logic**.
5. **Trade**:
   * Attributes: Trade ID, Stock Symbol, Buy/Sell Action, Quantity, Price, Timestamp.
   * Relationships:
     + Executed via **Trading API**.
     + Triggered by the **Prediction Engine** or **User**.
6. **Trading API (Alpaca)**:
   * Attributes: API Key, Account Info, Trade Execution Status.
   * Relationships:
     + Handles trade execution for **Trader/User**.
     + Interacts with the **Prediction Engine** for automated trading.
7. **User Interface (UI)**:
   * Attributes: Display Predictions, Trading Dashboard, Sentiment Analysis Visuals.
   * Relationships:
     + Provides access to **Trader/User**.
     + Displays outputs from the **Prediction Engine** and **Trading API**.

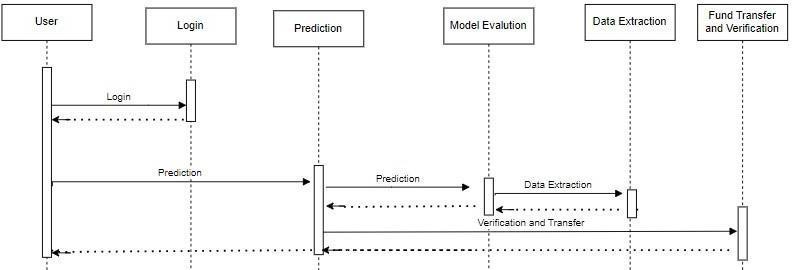
**DOMAIN MODEL DIAGRAM :**



#### Sequence Diagram

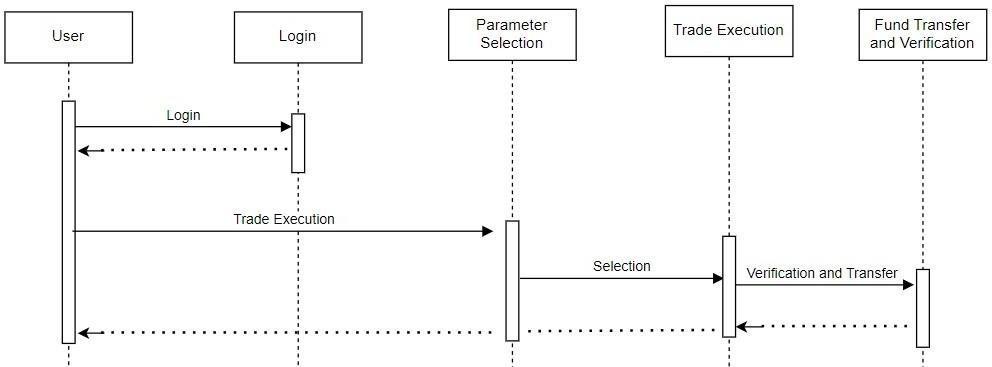
##### Stock Price Prediction

Figure represents how user logs in, enters stock symbol and prediction period, and clicks predict button. System extracts historical data, trains model, predicts stock price, and displays prediction to user.

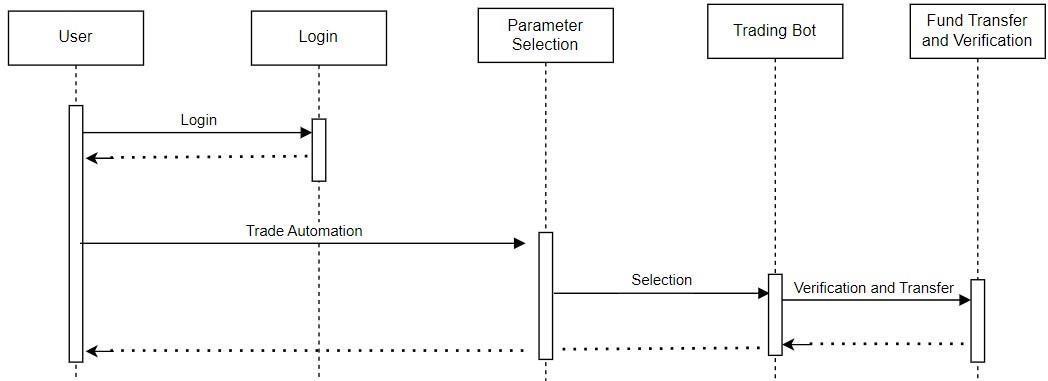


##### Trade Execution

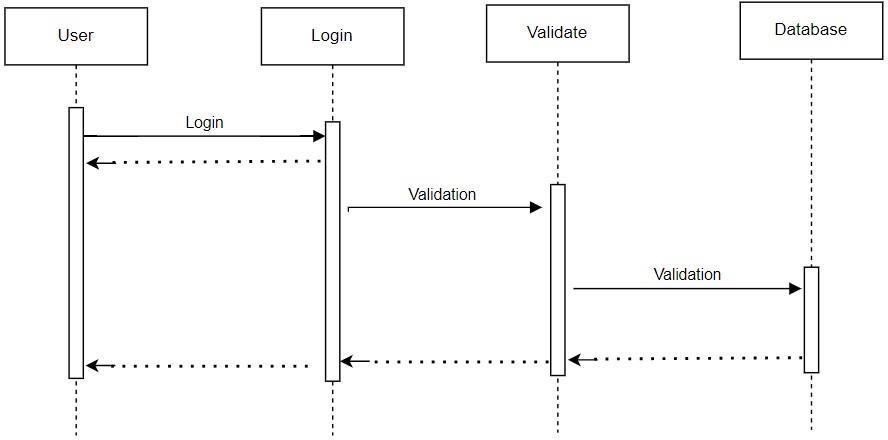
Figure shows how user submits trade order and waits for confirmation. System verifies credentials, validates order, executes trade, and sends confirmation to user.



##### Trade Automation via Stock Vison

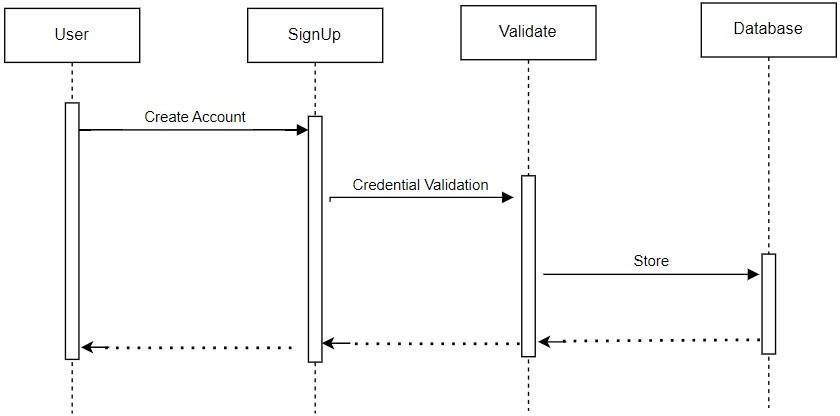
Figure shows how user enables trade automation and sets up parameters. System monitors market data, triggers trades based on parameters, funds verification and transfer are being done and sends notifications to the user about trade automation

##### Login

The Figure sequence diagram shows the steps involved in logging in to a system. The user first enters their username and password and clicks the login button. The system then verifies the user's credentials. If the credentials are correct, the system redirects the user to the dashboard. Otherwise, the system displays an error message.  
  
  


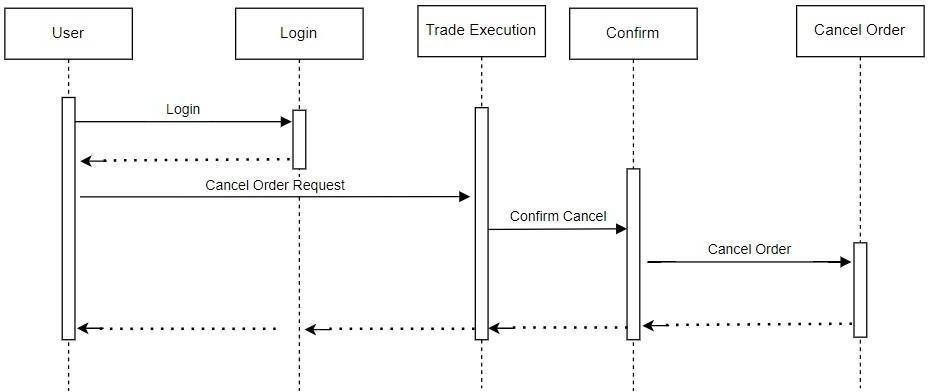
##### Sign Up

The Figure sequence diagram shows how a user can sign up. The credentials entered are first verified and them stored in database.



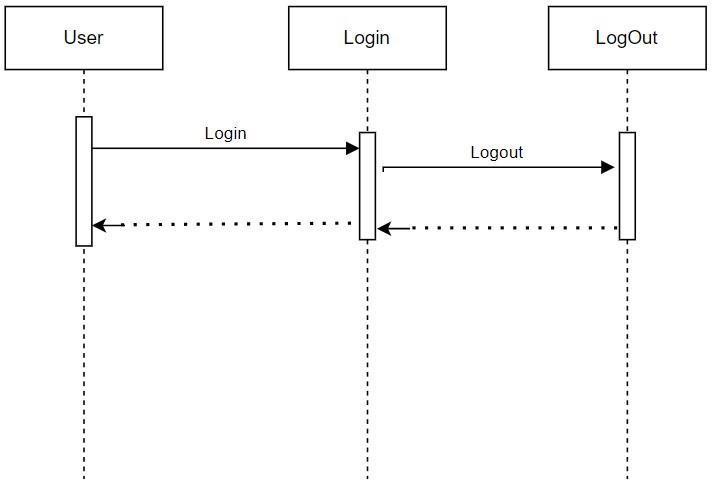
##### Trade Close

This sequence diagram figure shows how the user can close a executing trade.

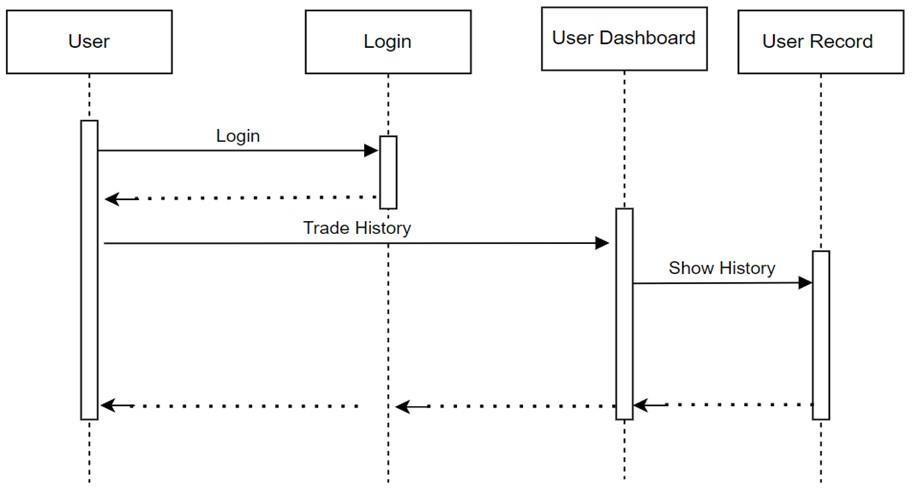


##### Logout

This sequence diagram figure shows that how a user can logout from the web interface.

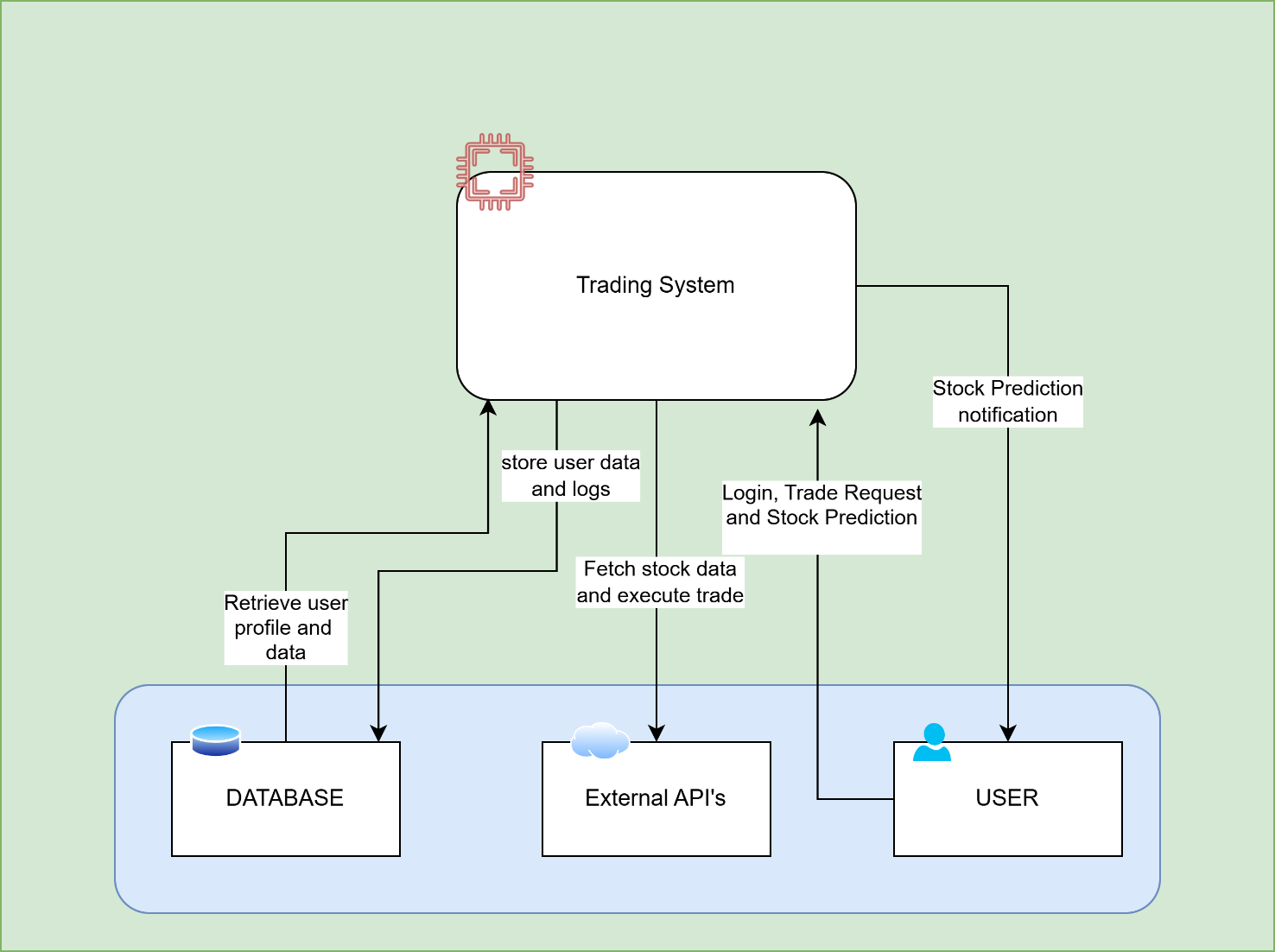


##### Trade Tracking

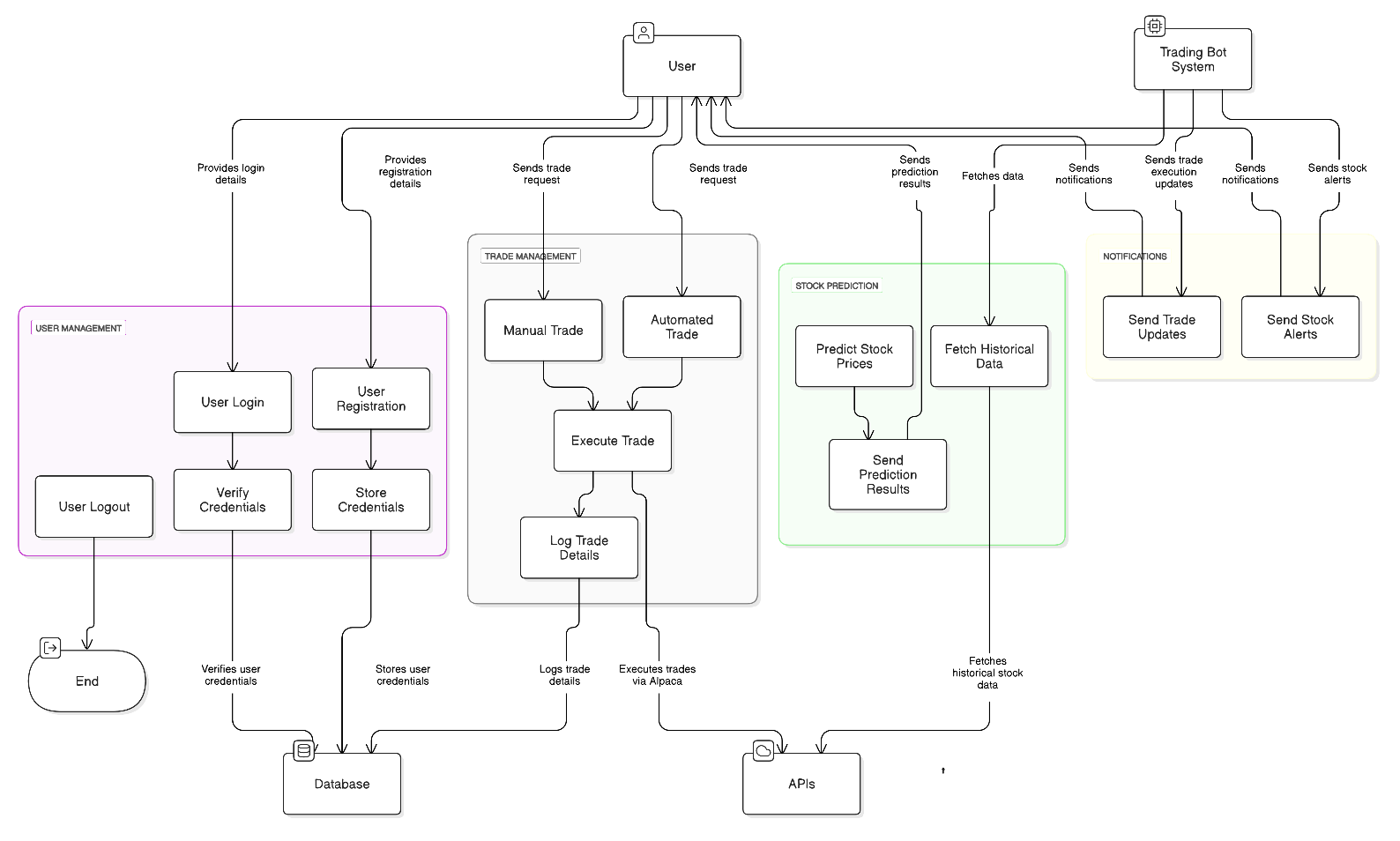
User can see its trade activity which is saved in user profile.

* 1. **Data Flow diagram**

**Data flow Diagram Level 0:**



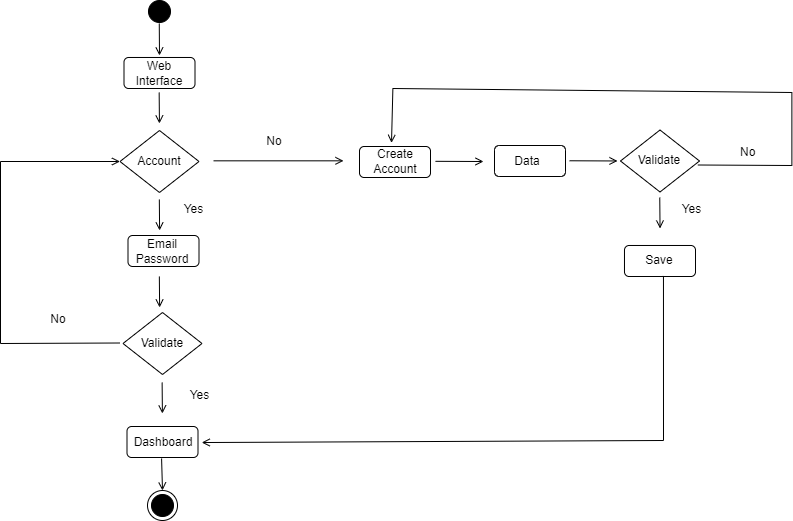
**Data flow Diagram Level 1:**



#### 

##### Login/Signup Activity

In this activity diagram figure, user is redirected to dashboard by validating account information. If user has no account, he/she is moved to signup page and after account creation, user is redirected to dashboard. Otherwise, user just logs in to existing account and is moved to dashboard.



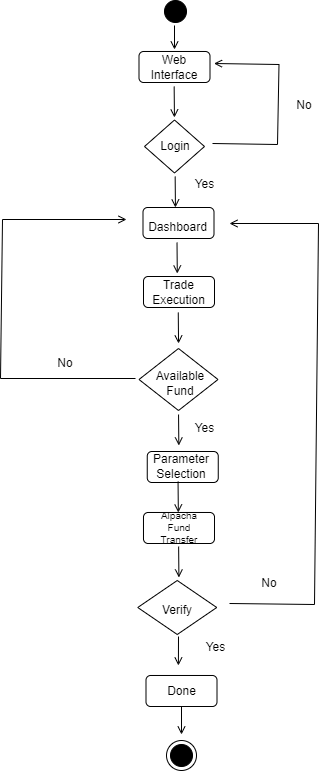
##### Prediction Activity Diagram

In this activity diagram figure, after successful login, user demands for prediction of his/her selected stock by selecting stock symbol and prediction period. System makes prediction and displays prediction to user if he/she is a premium user.



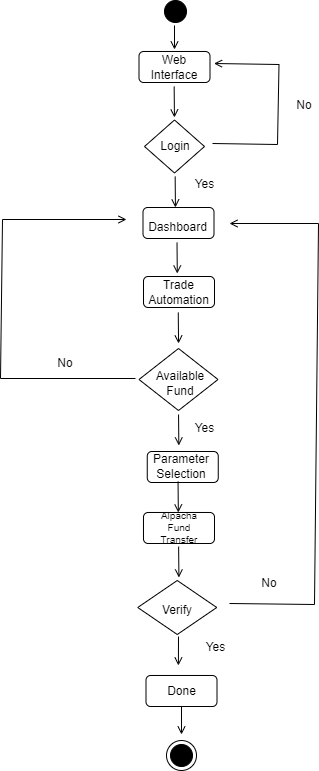
##### Trade Execution Activity

In this activity diagram figure, after successful login, user wants to execute self- trade. Available funds and parameters selections are confirmed and trade is places on Alpaca after fund transfer.



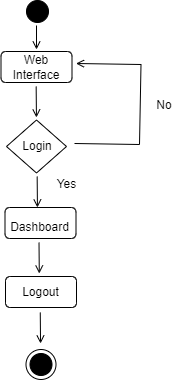
##### Trade Automation Activity

In this activity diagram figure, after successful login, user wants to execute auto-trade. Similar to self-trade, Available funds and paramters selections are confirmed and trade is places on Alpaca after fund transfer.



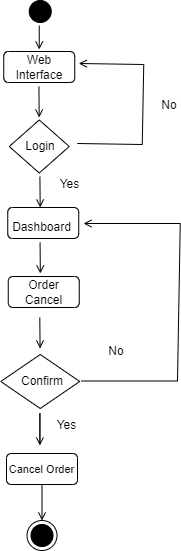
##### Logout Activity

After user successful login, user can easily select the option of logout from the dashboard as shown in activity diagram.



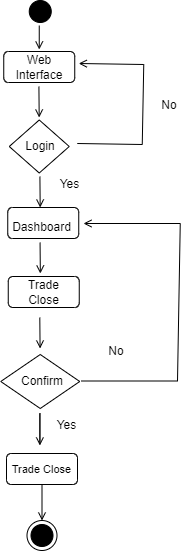
##### Cancel Order Activity

If a user wants to cancel an unexecuted order, he can easily cancel it. The system first asks for the confirmation and the cancel the order, but it has a precondition that user must be logged in.



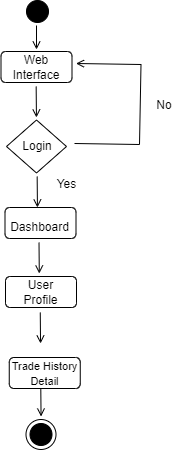
##### Close Trade Activity

If a trade is executing, we have given the liberty to user that he can cancel the trade. But the user must be logged in as shown in.



##### Tracking Trade Activity

This activity diagram figure shows that after logging in, user can check its history, total number of profit and loss, total orders in his user profile.



# CChapter 5

## Implementation

**Chapter 5:** Implementation

This chapter provides an in-depth look at the implementation details of **StockVision AI**, covering **important flow control, pseudo codes, key components, libraries, web services, deployment environments, tools, techniques, coding standards, and version control practices**. It outlines how **AI-powered stock predictions, real-time market analysis, and automated trading functionalities** are integrated into the system.

#### Important Flow Control/Pseudo codes

* **Stock Price Prediction Flow (Pseudo Code)**

1. Fetch livestock data from API (e.g., Yahoo Finance, Finnhub).

2. Preprocess the data (Normalize, Handle missing values).

3. Use LSTM/GRU model for stock price prediction.

4. Analyze model output and calculate confidence score.

5. Display predicted price on the Prediction Page.

6. If trade automation is enabled, execute buy/sell based on predefined strategy.

* **Trade Execution Flow (Pseudo Code)**

1. User selects stock and defines trade parameters (buy/sell, stop-loss, take-profit).

2. Verify user's account balance and trading permissions.

3. Send order to trading API (e.g., Alpaca).

4. Wait for trade confirmation.

5. Update trading history and notify the user.

#### Components, Libraries, Web Services and stubs

**Core Components:**

* **Frontend (React.js)**: UI for stock visualization, trade execution, and user management.
* **Backend (Django/Node.js)**: Handles API integration, trade logic, and user authentication.
* **AI Model (Python, TensorFlow/Keras)**: Processes historical stock data for predictions.
* **Database (MongoDB)**: Stores user profiles, trade history, and preferences.

**Key Libraries & Frameworks:**

* **Frontend:** React.js, TailwindCSS, Chart.js, ApexCharts
* **Backend:** Django REST Framework, Express.js
* **Machine Learning:** TensorFlow, Scikit-learn, Pandas, NumPy
* **APIs & Web Services:**
  + **Yahoo Finance API / Polygon API / Finnhub API** (Livestock Data)
  + **Alpaca API** (Trade Execution)
  + **News Sentiment API** (Market Sentiment Analysis)

**Stubs:**

For **testing without live services**, API responses are stubbed using **Mock APIs** (e.g., Postman).

#### Deployment Environment

StockVision AI is deployed in a **cloud-based environment** to ensure **scalability and real-time performance**.

* **Frontend Hosting:** Vercel / Netlify
* **Backend Hosting:** Aws
* **Database:** MongoDB Atlas
* **AI Model Deployment:** Google Collab

#### Tools and Techniques

To enhance **development efficiency**, the following tools are used:

* **Version Control:** Git, GitHub
* **Testing:** Jest (React), Postman (API Testing),
* **Development Tools:** VS Code, Jupyter Notebook

#### Best Practices / Coding Standards

To maintain code quality and efficiency, the following best practices are followed

* **Backend Development:**
  + Use **RESTful API design** for structured data flow.
  + Implement **error handling & logging**.
* **Frontend Development:**
  + Use **React functional components & hooks**.
  + Maintain **modular & reusable UI components**.
* **Machine Learning Model:**
  + Use **MinMaxScaler** for feature scaling.
  + Train model using **cross-validation** to avoid overfitting.

#### Version Control

[StockVision AI uses Git & GitHub for version control, ensuring collaborative development & code tracking.

* **Branching Strategy:**
  + main (Stable Production Code)
  + dev (Feature Development)
  + feature-branch (Individual Features)
* **Commit Standards:**
  + Follow **conventional commit messages**:
    - feat: Added trade execution API
    - fix: Resolved prediction model accuracy issue
    - refactor: Improved database indexing for faster queries
* **Pull Request (PR) Workflow:**
  + All changes require **code review** before merging into main