Software Requirements Specification

For

Fox of Hood

Version 1.0 approved

Prepared by:
Sai Bharadwaj Mukkera
Nithin Reddy Aenugu
Purna Sai Kumar Cheedirala
Shreeya Krishna Shiva
Tejas Manu Srinivasan

Hood College

1st October 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification defines the functional and non-functional requirements of the FOX OF HOOD Stock Portfolio Simulator. This document will refer to the initial release of the system, which will provide features like multiuser access, stock trading functionalities, real-time stock price updates using a free API, and financial data visualization. The main purpose of this web-application is to give customers a user friendly and interactive platform to manage stock portfolios, with authentication for security.

1.2 Document Conventions

This Software Requirements Specification (SRS) document adheres to the following conventions and formatting standards:

- **Font Style:** The document uses standard fonts such as Times New Roman (12pt) for the body text and Arial (bold, 14pt) for headings.
- **Section Numbering:** Each section is numbered hierarchically for clarity (e.g., 1, 1.1, 1.2, etc.).
- **Requirement Identification:** All functional requirements are identified with a unique tag (e.g., REQ-1, REQ-2) to ensure traceability throughout the document.
- **Emphasis:** Important terms are either bolded or italicized for emphasis.
- **Priority:** Functional requirements are assigned priority levels—High, Medium, or Low—based on their importance to the system functionality.
- **Tables and Diagrams:** All diagrams, such as flowcharts or entity-relationship diagrams, are labeled and placed under the appropriate sections with relevant descriptions.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers, project managers, testers, and stakeholders involved in the development of the FOX OF HOOD web application. Developers should concentrate on the sections detailing functional and non-functional requirements, while project managers may want to give priority to the scope and project duration. Testers should review the sections detailing user activities and error handling. It is recommended to start with an overview of the sections before proceeding to detailed requirements.

1.4 Project Scope

The FOX OF HOOD Stock Portfolio Simulator is designed to allow users to trade stocks, monitor real-time stock prices, and manage their portfolios through a user-friendly interface. Key features include multi-user access, authentication, detailed logging of user activities, persistent data storage, and graphical financial data representations. The purpose of this application is to provide users with a modern and efficient tool for managing their financial portfolios.

1.5 References

[1] IEE Computer Society (https://www.computer.org/resources/software-requirements-specifications_)

2. Overall Description

2.1 Product Perspective

This stock trading system is designed as an integrated web-based platform. It integrates various real-time data processing, trading, and analysis features to enable users to perform stock trading activities with ease. The system fits within the broader landscape of online financial trading platforms, offering users a seamless and secure environment to buy and sell stocks.

1. **System Integration**
   The stock trading platform uses huge system integrations, especially with external services that include real-time stock data APIs. These APIs provide current market data, which is fetched by the application for display to users in order to make decisions on trading. The system also integrates with authentication tools, including CAPTCHA and user credentials for secure login measures, protecting it from unauthorized access. This will support a wide range of integrations in terms of smoother communications between the components for better functionality.

2. **User Interface and Experience**

   The platform is an intuitive user interface featuring a number of dashboards that serve multiple purposes. Portfolio Dashboard-the place for the user to monitor their investments-while the Web-App User Dashboard represents the main area where trading, analysis, and data search functions are performed. It is designed to be user-friendly for any category of trader, from a novice to professional, hence its interface is friendly yet powerful.

3. **Core Functionality**

   One of the core functionalities is the Trading Engine that buys and sells transactions while interacting with the database that stores information about all users, transaction logs, and stock data. The database will also provide the facility of search and periodic backup for assurance of data integrity. It supports a modular and efficient architecture for various activities, such as trade management and analysis, information retrieval, and so on. This will let the system efficiently handle high volumes of transactions because of the centralization of data and processing in the database.

4. **Security and Data Integrity**

   This is a crucial aspect in the system. The introduction of the CAPTCHA authentication by the user makes sure that it identifies the users of the system. The system also allows the regular backup activities and logs all the events for the integrity of data and traceability of the actions taken by the system. These prevent any breach in the users' information or failure of the system from affecting the users; hence, they are assured of consistency in the performance of the systems. This logging will also enable auditing and problem determination for better reliability.

5. **Scalability**

   This architecture is ready for scaling with the growth of users and trades. Because it is modular, integrating other data sources or scaling up functionality is easier without large renovations of the core. Whether through increased user traffic or through the integration of new external APIs, the platform will seamlessly scale to accommodate growth in whatever form. This scalability ensures that the system will evolve according to needs-both of the financial market and its users.

2.2 Product Features

The Stock Portfolio Simulator application provides users with a comprehensive platform for managing virtual stock portfolios, real-time stock market interactions, and detailed portfolio analysis. The major features are summarized below:

1. **User Authentication:** This feature allows users to create accounts, log in securely, and access their portfolio. It includes password hashing, account lockout mechanisms after multiple failed login attempts, and session management with automatic logout after a period of inactivity.
2. **Stock Market Data Fetching:** Users can search for and view real-time market data for selected stocks. The system fetches up-to-date stock prices every 10 seconds from a free stock market API, with visual indicators showing stock price changes.
3. **Portfolio Management:** Users can manage their virtual portfolios by buying and selling stocks with dummy funds. The system tracks transactions, updates portfolio balances in real-time, and maintains a history of all stock trades.
4. **Portfolio Analysis and Visualization:** This feature provides users with graphical insights into their portfolio's performance. The system generates charts, such as pie charts for stock distribution and line charts for historical performance, allowing users to analyze their investments over different time periods.

2.3 User Classes and Characteristics

Regular Users

1. These users will interact with the platform by managing virtual portfolios, executing trades, and viewing financial data.
2. No prior technical knowledge of stock trading is required.
3. **Security Level:** Standard user-level privileges with access only to their personal portfolio and trading history.

Administrators

1. These users will have elevated privileges allowing them to manage user accounts, monitor system performance, and view system logs.

2. **Technical expertise:** Basic understanding of account management and system administration.
3. **Security Level:** Elevated privileges for user management and oversight.

Developers

1. Developers working on this project require access to all system components for troubleshooting and feature enhancements.
2. **Expertise:** Proficiency in React, Node.js, database systems (e.g., MongoDB), and API integrations.

2.4 Operating Environment

The stock trading system is designed to operate within a Windows Desktop Processing Power environment, ensuring compatibility with widely used desktop platforms. The system leverages modern processing capabilities to handle real-time stock data, intensive calculations, and high-frequency trading operations efficiently. Windows, with its broad hardware support and stable desktop environment, provides the optimal foundation for this platform's operations.

The system integrates a real-time stock price API, connecting directly with external financial data providers to retrieve and update live stock prices. This integration ensures that the platform always has the most accurate and current market data for users. The database used for managing user data, transactions, and stock price logs is PostgreSQL, chosen for its robustness, scalability, and ability to handle large volumes of transactional data efficiently. The integration between the backend, the web development framework, and the database ensures seamless communication and high performance in delivering real-time trading services to users.

2.5 Design and Implementation Constraints

1. **Data Privacy:** The system must ensure the privacy of user data, including credentials, transaction histories, and portfolio information. All user data must be securely stored and transmitted using encryption techniques to prevent unauthorized access or data breaches.

2. **Security Standards:** The platform will implement secure communication protocols such as HTTPS with SSL/TLS to ensure encrypted transmission of data between users and the platform. CAPTCHA functionality is required for authentication, and the system must support face recognition as an optional security measure.
3. **Performance Metrics:** The system is required to handle real-time stock data updates and transactions efficiently, ensuring quick response times and the ability to handle multiple concurrent users without performance degradation. Regular performance testing will ensure that the platform meets the expected performance under various conditions.

2.6 User Documentation.

2.6.1 User Manual

The User Manual is a comprehensive guide designed to assist end-users in navigating and using the FOX OF HOOD Stock Portfolio Simulator. The manual will provide step-by-step instructions and detailed explanations on how to access, operate, and manage the various functionalities of the simulator.

2.7 Assumptions and Dependencies

1. **Stock Price Accuracy:** A key assumption is that the platform will consistently receive accurate and up-to-date stock price information from external data providers. The performance and reliability of the application are highly dependent on the accuracy of the stock prices, as users base their trading decisions on this data. Any delays, inaccuracies, or disruptions in the stock price feed could impact the user experience, potentially leading to incorrect trades or financial losses. The platform assumes that its integration with real-time stock data APIs will maintain a high level of precision and consistency, ensuring users receive live updates without interruption.
2. **Technical Dependencies:** The platform has several critical technical dependencies. One of the primary dependencies is on the network infrastructure. Given the real-time nature of the application, a stable and high-speed internet connection is assumed for both the platform's backend services and end-users. Any network latency or disruptions could affect the timely processing of trades and the display of stock price data, leading to a degraded user experience. Additionally, the system depends on cloud or local server infrastructure to

manage data storage, real-time processing, and high-volume transactions. The application assumes that the hosting environment will have sufficient processing power, storage, and uptime reliability to meet the platform’s demands.

3. **Backend and API Reliability:** The platform is also dependent on the stability and reliability of the backend systems and external APIs. The external APIs providing stock prices, news, and financial data must be consistently available and performant. Any downtime or slow response times from these APIs could affect the real-time trading functionality of the platform. The backend infrastructure, including the database and web application, must remain operational and capable of handling the load imposed by a growing user base. This includes ensuring that the backend can scale horizontally as user demand increases, allowing the platform to continue operating efficiently without service interruptions.

3. System Features

3.1 User Authentication

3.1.1 Description and Priority

The **User Authentication** feature allows users to create an account, log in, and securely access their portfolio. This feature is of **High Priority** as it ensures that user data is protected and only accessible to authorized users.

3.1.2 Stimulus/Response Sequences

1. **Stimulus:** The user navigates to the login page and enters their credentials (username and password).
   Response: The system verifies the credentials and grants access to the user's dashboard upon successful authentication.
2. **Stimulus:** The user enters invalid credentials.
   Response: The system displays an error message prompting the user to re-enter valid credentials.

3.1.3 Functional Requirements

REQ-1: The system must allow users to create an account by providing a username, password, and valid email.

REQ-2: The password must be hashed using a secure hashing algorithm before being stored in the database.

REQ-3: The system must validate user credentials during login and grant access only if the credentials match.

REQ-4: The system must provide an option for users to reset their passwords via email if they forget their login credentials.

REQ-5: If the user enters incorrect credentials, the system should display an error message.

REQ-6: After five failed login attempts, the system should lock the user's account and send an email notification.

REQ-7: The system must automatically log out users after 15 minutes of inactivity.

3.2 Stock Market Data Fetching

3.2.1 Description and Priority

The **Stock Market Data Fetching** feature allows users to view real-time market rates for selected stocks. This feature is of **High Priority** because users need to interact with up-to-date market information.

3.2.2 Stimulus/Response Sequences

1. **Stimulus:** The user selects a stock symbol or navigates to their portfolio page
   Response: The system fetches the latest stock data from the external API and displays it on the dashboard.
2. **Stimulus:** The stock data is updated every 10 seconds.
   Response: The system refreshes the displayed data without requiring user input.

3.2.3 Functional Requirements

REQ-1: The system must allow users to search for stock symbols and display their current market rates.

REQ-2: The system must fetch stock data from a free stock market API, updating the information every 10 seconds.

REQ-3: The system should display stock price changes (up or down) using visual indicators (e.g., green for increase, red for decrease).

REQ-4: If the API request fails, the system must display an error message and prompt the user to retry.

REQ-5: The system should handle edge cases such as invalid stock symbols by displaying appropriate error messages.

3.3 Portfolio Management

3.3.1 Description and Priority

The Portfolio Management feature allows users to manage their virtual stock portfolios, track stock performance, and view financial summaries. This feature is of Medium Priority as it is essential for users but does not impact the core functionality of viewing stock data.

3.3.2 Stimulus/Response Sequences

1. **Stimulus:** The user adds a stock to their portfolio.
   Response: The system adds the selected stock, records the purchase, and updates the portfolio value.
2. **Stimulus:** The user sells a stock from their portfolio.
   Response: The system updates the portfolio balance, removes the stock, and records the transaction.

3.3.3 Functional Requirements

REQ-1: The system must allow users to add stocks to their portfolio using dummy funds.

REQ-2: The system must update the user's portfolio in real time when stocks are bought or sold.

REQ-3: The system must display the user's current portfolio balance and performance metrics.

REQ-4: The system should provide a history of all transactions, including stock purchases and sales.

REQ-5: If the user attempts to buy a stock with insufficient funds, the system should display an error message.

3.4 Portfolio Analysis and Visualization

3.4.1 Description and Priority

The Portfolio Analysis and Visualization feature provides users with graphical representations of their portfolio's performance, helping them make informed decisions. This feature is of Medium Priority as it enhances the user experience by providing insights.

3.4.2 Stimulus/Response Sequences

1. **Stimulus:** The user views their portfolio analysis dashboard.
   Response: The system generates graphs, such as pie charts and line charts, to visualize stock distribution and portfolio growth.
2. **Stimulus:** The user selects a specific stock for detailed analysis.
   Response: The system displays detailed stock performance graphs for the selected stock.

3.4.3 Functional Requirements

REQ-1: The system must generate pie charts showing the percentage distribution of stocks in the user's portfolio.

REQ-2: The system must generate line charts showing the historical performance of the user's portfolio over time.

REQ-3: The system should allow users to filter portfolio performance data by date range.

REQ-4: The system should display detailed stock analysis, including historical price trends, for each stock in the user's portfolio.

REQ-5: If data for analysis cannot be retrieved, the system must display an appropriate error message and offer a retry option.

4. External Interface Requirements

4.1 User Interfaces

User interfaces (UIs) are the medium through which users interact with your system. The UI should be designed for ease of use, accessibility, and efficiency. A clear, well-designed UI reduces user errors and enhances the overall user experience.

a. **Login Screen:** The user enters their username and password. The system should validate the input and provide an error message if the credentials are incorrect.
b. **Navigation Menus:** Use clear and concise labels for buttons, such as “Submit,” “Cancel,” and “Back.” Avoid overwhelming users with too many options on one screen.
c. **Forms, Buttons, and Navigation:** Detail the types of buttons (e.g., Submit, Cancel), forms, and navigation elements (e.g., menus, breadcrumbs) that the users will interact with.
d. **Input Methods:** Explain how the users will provide input—via keyboard, mouse, touchscreen, etc. Define any required keyboard shortcuts or gestures.
e. **Accessibility:** Mention compliance with accessibility standards (e.g., WCAG) to support users with disabilities, such as screen reader support or color contrast standards.

Requirements:

1. The login page must provide real-time feedback, such as showing error messages if the user leaves a required field blank (e.g., "Username is required").
2. A "Help" button must be available on every screen for users to access tips or tutorials.

3. Error messages must be clearly displayed in red text and appear next to the field that triggered the error.

Common Errors:

A. Form submission errors (e.g., user forgets to fill a required field or enters invalid data).
B. Navigation errors (e.g., users can't find key functionality).

How to Overcome:

I. **Error Handling:** Provide clear, actionable error messages, such as "Invalid email format" or "Password must be at least 8 characters long."
II. **User Guidance:** Include tooltips or field validation hints to help users avoid errors before they occur.

4.2 Hardware Interfaces

NA

4.3 Software Interfaces

Software interfaces define how your product interacts with other software systems, such as databases, third-party libraries, or web services. These interactions typically require proper API integration or database queries.

a. **Database Interface:** The system retrieves customer data from a MySQL database and displays it on a dashboard.
b. **Third-Party API:** The software integrates with Google Maps API to provide real-time geolocation services.
c. **Operating Systems:** List any operating systems with which the software must be compatible (e.g., Windows, Linux, macOS), and detail how the system interacts with OS-level functions (file management, memory usage, etc.).
d. **Third-party Libraries:** Mention any third-party libraries or frameworks that will be integrated into the software (e.g., authentication libraries, analytics tools).
e. **Version Control:** For integrations with software components, specify version compatibility (e.g., supports API v2.0 or higher).

Requirements:

1. The system must query the MySQL database using secure prepared statements to prevent SQL injection attacks.
2. The system must use the REST API provided by the third-party payment gateway to process transactions, and handle API responses (e.g., success, failure, timeout).

Common Errors:

A. **Database Connection Failure:** The software may fail to connect to the database due to network issues or incorrect credentials.

B. **API Versioning Conflicts:** If the third-party API is updated, the software may no longer be compatible with the new version.

How to Overcome:

I. **Retry Mechanism:** For database or API failures, implement retry logic that attempts to reconnect after a short delay.

II. **Version Control:** Document the supported API version and implement testing during API updates to ensure backward compatibility.

4.4 Communications Interfaces

Communication interfaces involve how your software communicates over networks (LAN, Internet), including data transmission, email, or web service calls. Networked systems need to manage latency, bandwidth, and security.

a. **Web Services:** The software communicates with an external web service via HTTP to exchange JSON data for updating inventory.

b. **Latency and Bandwidth Requirements:** Mention performance requirements related to network communication, such as acceptable latency for real-time communication or minimum required bandwidth.

c. **Network Communication Protocols:** Describe network protocols that the system will use, such as HTTP/HTTPS, FTP, TCP/IP, or Web Sockets, and their role in the system's operations.

d. **Data Transfer Standards:** Specify the standards for data exchange across the network, such as using XML, JSON, or proprietary formats for data transmission.
e. **Security:** Describe security requirements for communication, such as the need for encryption (e.g., SSL/TLS) and any certificates that must be installed on the client or server.
f. **Synchronization Mechanisms:** If the software must synchronize data with another system, explain how synchronization will occur (e.g., periodic data pulls, real-time updates, or batch processing).

Requirements:

1. The system must send email notifications through a secure SMTP server using TLS encryption.
2. The system must support HTTP/2 for faster communication with external web services

Common Errors:

A. **Timeouts:** If the software tries to communicate with a web service but the connection is too slow, a timeout error occurs.
B. **Data Corruption:** If the data being transferred is not properly formatted, it may become corrupt during transmission.

How to Overcome:

I. **Error Handling and Retries:** For timeouts, retry the operation up to three times before showing an error message to the user (e.g., "Unable to connect to the server, please try again later").
II. **Validation:** Validate incoming and outgoing data formats (e.g., ensure JSON or XML is correctly structured) before and after transmission to avoid data corruption.

Summary of Common Errors and Solutions

User Input Errors:

Error: Invalid input or incomplete form fields.

Solution: Use real-time validation to guide users (e.g., highlight errors immediately as they occur).

Hardware Device Failures:

Error: Lost connection to hardware devices like printers or scanners.

Solution: Implement a retry mechanism or prompt users to reconnect the device.

Database Connection Issues:

Error: The software fails to connect to a database.

Solution: Use retries and provide clear error messages like "Database connection failed. Please check your network connection."

API Incompatibility:

Error: The software fails to work with an updated API version.

Solution: Regularly test the software with newer API versions and update the integration accordingly.

Communication Delays or Failures:

Error: Network communication is too slow or fails altogether.

Solution: Use timeouts and retries, and log errors for further analysis

Interface-Specific Considerations

I. **User Interfaces,** mockups or screen designs could be helpful to visualize key elements.

II. **Hardware Interfaces,** testing on actual hardware prototypes should be mentioned if applicable.

III. **Software Interfaces,** make sure you list any dependencies on third-party software or services and specify integration tests to ensure everything works as expected.

IV. **Communications Interfaces,** error handling for network failures (e.g., retries, notifications) and communication logs could be essential to track data flows and ensure system reliability.

Resources for Additional Information:

I. IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications

II. ISO/IEC/IEEE 29148:2018: Systems and Software Engineering – Life Cycle Processes – Requirements Engineering

III. W3C Web Accessibility Initiative (WAI) for accessibility guidelines.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Real-Time Requirements

Stock Price Updates: Real-time updates in the application should be provided at least once per minute, reflecting the changes and updates of the stock market.

Latency: The delay between a stock price change and its display on the user's interface should be minimal, ideally less than a minute.

Response time: Orders made in the application should be executed within a few seconds of submission to minimize the impact of market changes.

Confirmation: Confirmation of order execution should be provided to the user without any delay.

5.1.2 Non-Real-Time Requirements

Calculation time: Portfolio valuations should be calculated within a less timeframe, ideally less than 10 seconds.

Accuracy: Calculations should be accurate and based on the most recent stock prices.

5.1.3 Chart and Graph Providing:

Providing time: Charts and graphs should be provided within a few seconds of user interaction or data updates.

Interactivity: Users should be able to interact with charts and graphs smoothly, without lag.

Search Functionality: Search results should be displayed within a second or two of the user entering a query.

Data Loading: The initial loading of the application and user data should be completed within a reasonable timeframe, ideally less than 10 seconds.

5.2 Safety Requirements

5.2.1 Data Loss

Safeguards: Implement regular backups of user data and transaction history to mitigate the risk of data loss due to hardware failures, software bugs, or security breaches.

Actions: Conduct routine backups and store backups in a secure location, preferably off-site

5.2.2 Financial Loss

Safeguards: Educate users about the risks associated with investing in stocks and provide tools to help them make informed decisions.

Actions: Include disclaimers and risk disclosures in the application, and consider offering educational resources on financial literacy.

5.2.3 Security

Safeguards: Employ strong security measures, such as encryption, secure authentication, and regular security audits.

Actions: Use industry-standard encryption algorithms to protect sensitive data, enforce password policies, and conduct vulnerability assessments.

5.2.4 Unauthorized Access

Safeguards: Implement robust access controls and authentication mechanisms to prevent unauthorized access to user accounts and data.

Actions: Require strong passwords, enable two-factor authentication (2FA), and regularly monitor for suspicious activity.

5.2.5 System Failures

Safeguards: Design the system to be fault-tolerant and implement redundancy to minimize the impact of system failures.

Actions: Use redundant hardware components, implement failover mechanisms, and regularly test the system's resilience to failures.

5.3 Security Requirements

User Identity Authentication

Strong passwords: Require users to create strong passwords with a combination of uppercase and lowercase letters, numbers, and symbols.

Two-step verification: Implement 2SV as an optional security measure, allowing users to verify their identity using a second factor, such as a code sent to their mobile device.

Face Recognition: Consider Face Recognition as an additional layer of security.

Data encryption: Encrypt sensitive data, including user credentials, financial information, and transaction history.

5.4 Software Quality Attribute

5.4.1 For Customers

- **Usability:** The application should have a clear and intuitive interface, with minimal onboarding required for new users.
    - **Quantitative:** 90% of new users should be able to complete a basic transaction within 5 minutes.
- **Reliability:** with minimal downtime and accurate data.
    - **Quantitative:** 99.9% uptime and data accuracy rate of 99.9%.
- **Security:** User data and financial information should be protected with robust security measures.
- **Performance:** The application should performs efficiently, even during peak usage times.
    - **Quantitative:** Average response time of under 5 seconds for most actions.

- **Adaptability:** The application should be able to adapt to changing market conditions and user preferences.
    - **Qualitative:** Ability to integrate with new data sources and customizable features.

5.4.2 For Developers

- **Maintainability:** The codebase should be well-organized, modular, and easy to understand.
- **Testability:** The application should be easy to test, with a high test coverage.
    - **Quantitative:** 80% code coverage with unit and integration tests.
- **Flexibility:** The application should be flexible and adaptable to future changes.
    - **Qualitative:** Use of modular design patterns and decoupled components.
- **Portability:** The application should be deployable on various platforms and environments.
- **Interoperability:** The application should be able to integrate with other systems and services.
    - **Qualitative:** Use of standard APIs and protocols for data exchange.
- **Ease of use:** The application should be intuitive and easy to navigate, with clear and concise instructions.
- **Learnability:** Users should be able to learn the basic functionalities within 15 minutes of initial use.
- **Modularity:** The codebase should be well-structured and modular, making it easy to maintain and update.
- **Documentation:** Comprehensive documentation should be provided, including user manuals, developer guides, and API specifications**.**
- **Compatibility:** The application should be compatible with various operating systems and devices. Supported by different browsers.
- **Error handling:** The application should be able to handle unexpected inputs and errors gracefully.
- **Security:** The application should be resistant to security threats and vulnerabilities.

6. Other Requirements

Data Types: define data types for each field to ensure data accuracy (e.g., use DECIMAL for financial values, DATETIME for timestamps).

Indexing: Create appropriate indexes on queried fields to better performance.

Backup and Recovery: Implement regular backups and a disaster recovery plan to protect data.

Security: Implement security measures to protect sensitive data, such as encryption and access controls.

Language Support: Support multiple languages for user content.

Date and Time Formatting: Display dates and times in the user's preferred format.

Terms of Service: Create clear terms of service for users, outlining the application's usage rights and limitations.

Privacy Policy: Develop a comprehensive privacy policy that explains how user data is collected, used, and protected.

Appendix A: Glossary

SRS: Software Requirements Specification

GUI: Graphical User Interface

CAPTCHA: Completely Automated Public Turing test to tell Computers and Humans Apart

API: Application Programming Interface

Project-Specific Terms

FOX OF HOOD: Development codename for the Stock Portfolio Simulator project.

Portfolio: A collection of investments, such as stocks, bonds, or mutual funds.

Stock: A share of ownership in a company.

Trade: The buying or selling of securities.

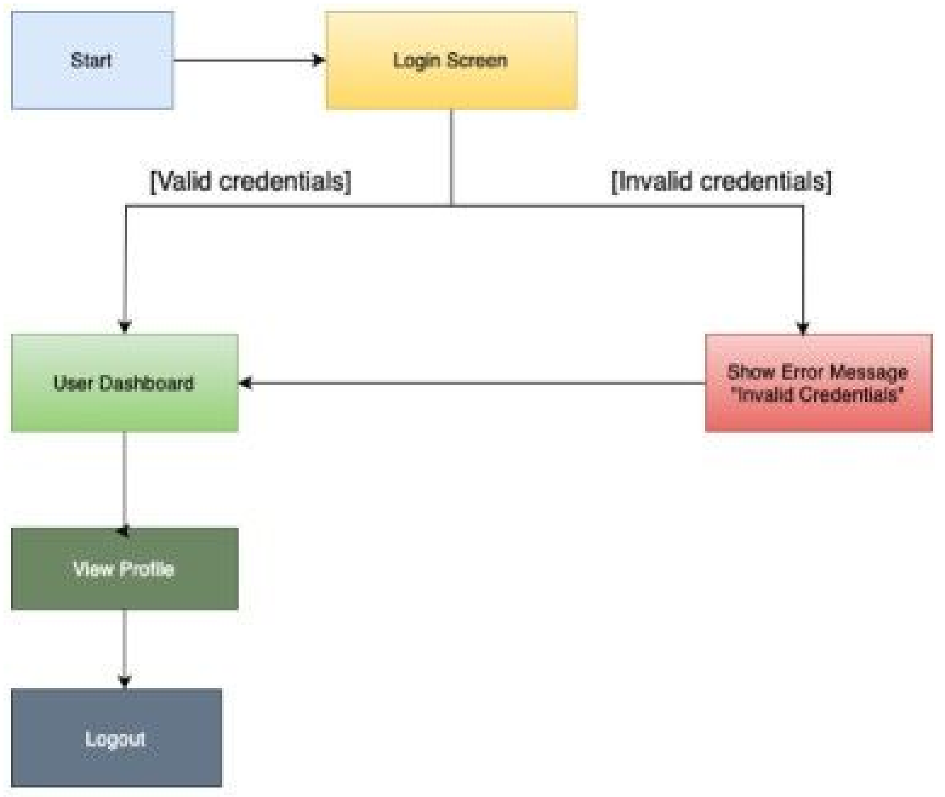
Position: The quantity of a particular security held by an investor.

Up-to-date pricing: Current market prices for stocks.

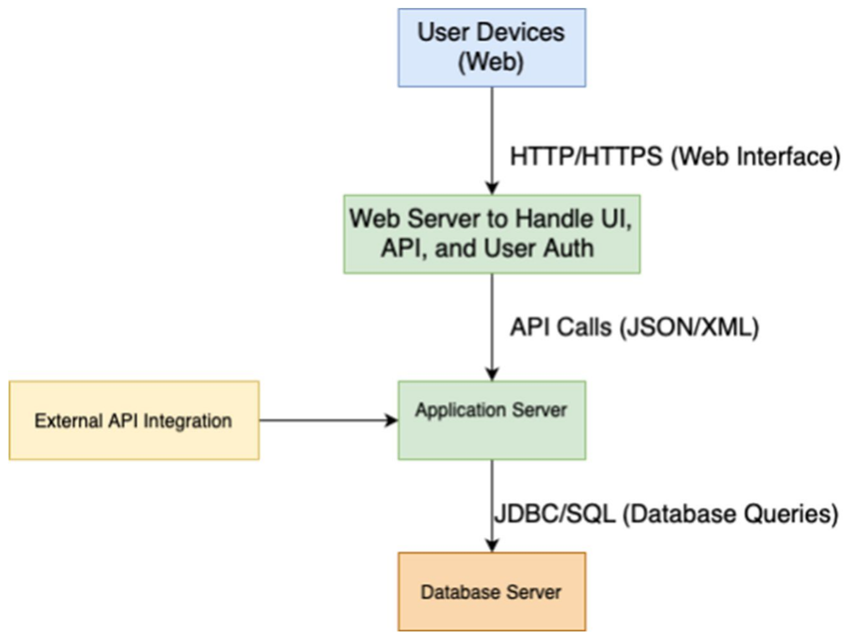
Persistent storage: A storage system that retains data even when the application is closed.

Appendix B: Analysis Models

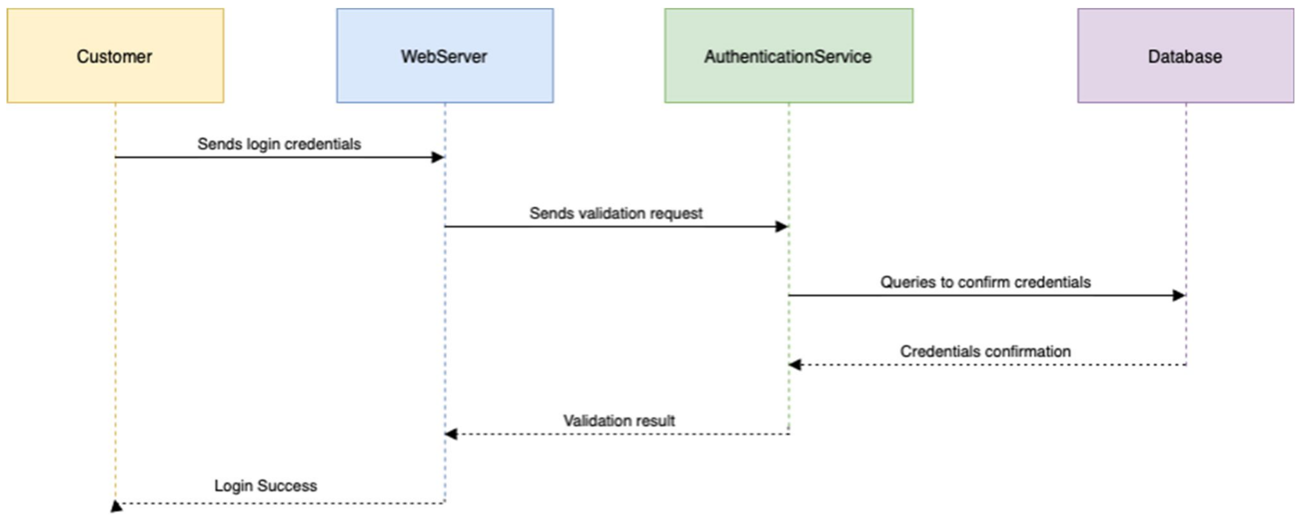
1. Flowchart for User Interface (UI) Navigation:



2. System Architecture Diagram:


User Devices (Web)
HTTP/HTTPS (Web Interface)
Web Server to Handle UI, API, and User Auth
API Calls (JSON/XML)
External API Integration
Application Server
JDBC/SQL (Database Queries)
Database Server

3. Sequence Diagram:



Appendix C: Issues List

1. Issue #1: - Decision pending on whether to use MongoDB, PostgreSQL or Firebase as the persistent data storage solution.

2. Issue #2: - Integration of facial recognition for authentication is currently in the research phase.

3. Issue #3: - Determining if real-time stock data updates every 10 seconds meet user expectations for accuracy and performance.