FOX OF HOOD

Programmer's Guide for Portfolio Management System

Team-D

Sai Bharadwaj Mukkera

Nithin Reddy Aenugu

Purna Sai Kumar Cheedirala

Tejas Manu Srinivasan

Shreeya Krishna Shiva

Contents

1. Introduction

This guide is designed to provide a comprehensive understanding of the Portfolio Management System for developers. Designed for multi-user environments, offering a robust and user-friendly web-based interface. The system enables users to manage their stock portfolios effectively by viewing real-time data, analyzing investment performance, and performing transactions such as buying and selling stocks. Additionally, the system incorporates an administrative interface for user management and security measures for data integrity and protection.

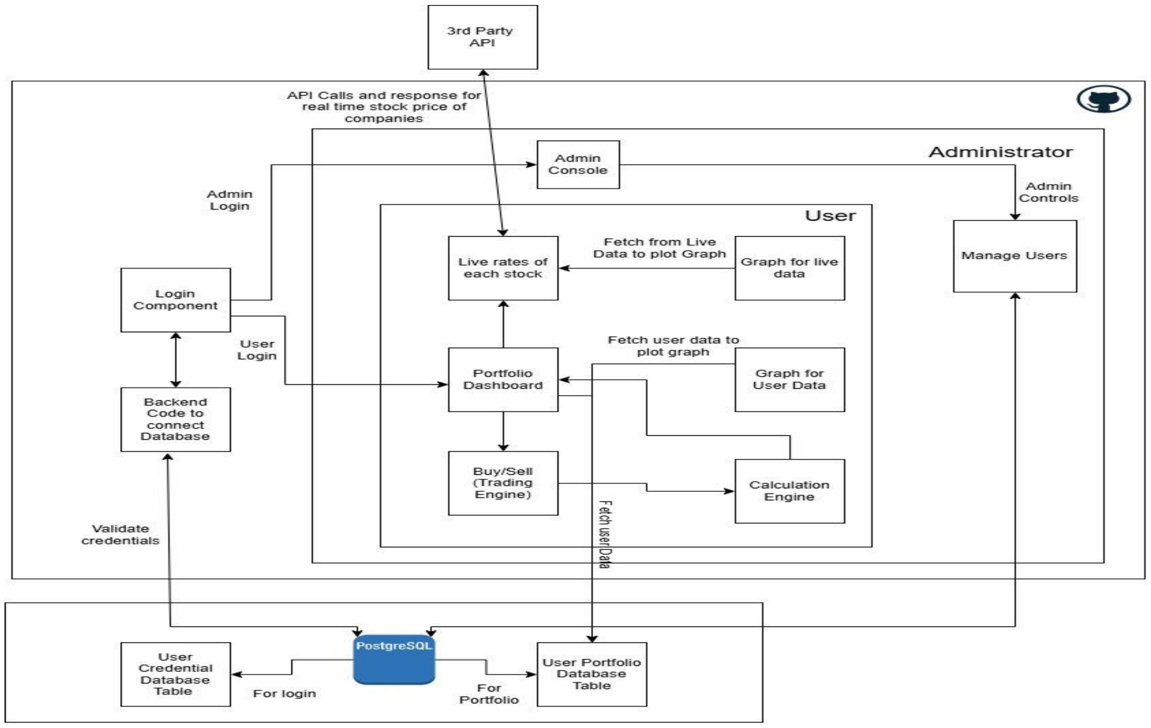
This document aims to facilitate Key features like User Authentication, Portfolio Management, and Admin Controls, the maintenance, extension, and scalability of the system while ensuring adherence to best practices.

---

2. System Architecture

The system comprises various components connected to a centralized PostgreSQL database for data storage. A calculation engine ensures real-time updates and synchronization of stock data on the dashboard. The Portfolio Management System is built using a modern multi-tier architecture to ensure scalability, maintainability, and modularity.

- **Frontend:** A React.js-based single-page application (SPA) offering an interactive and user-friendly interface.
- **Backend:** A Node.js/Express.js RESTful API handling the business logic, data processing, and communication with external APIs.
- **Database:** PostgreSQL is used for secure and efficient data storage.



ARCHITECTURE DIAGRAM 1

Third-Party Integrations:

Alpha Vantage API: Provides real-time stock data and historical price information. The system components communicate through RESTful API endpoints, enabling independent development and seamless integration.

---

3. Frontend Overview

The frontend is built with React.js and adheres to the principles of modular design, allowing for reusable components and efficient development.

Framework:

ReactJS was chosen for its component-based architecture, which facilitates code reuse, scalability, and efficient rendering. The dynamic nature of React allows the application to handle frequent updates to stock data and user interactions without reloading the entire page.

3.1 Key Components

3.1.1 Login Component

The Login Component is responsible for handling the user authentication process, ensuring secure and seamless access to the application.

Features:

- **User Registration:** Allows new users to register by providing their email, username, and password. The data is securely stored in a PostgreSQL database.
- **Login Functionality:** Authenticates existing users by verifying their credentials against the database using Node.js backend logic.
- **Captcha:** A basic captcha functionality which is generated in the backend of the application and validating it with the user input to block bots and automated logins.
- **Error Handling:** Provides feedback messages for incorrect credentials, CAPTCHA errors, or account lockouts.

3.1.2 Admin Panel

The **Admin Panel** is a restricted area accessible only to administrators, offering comprehensive user account management.

Features:

- **Add New Users:** Administrators can create new user accounts manually.
- **Delete Users:** Remove inactive or unauthorized accounts from the system.
- **Access Control:** Admin functionality is protected by role-based access control (RBAC), ensuring only authorized users can access sensitive operations.

3.1.3 Company List Component

The Company List Component displays a comprehensive list of 25 available stocks, providing users with real-time market data and facilitating trade operations.

Features:

- **Stock Information:** Displays stock name, ticker symbol, current price, Fetches live data from the Alpha Vantage API and updates dynamically.
- **Buy/Sell Operations:** Allows users to purchase new stocks or sell existing holdings directly from the list. Provides input fields for specifying the number of shares and

calculates the total transaction cost in real-time. Includes validation to prevent invalid transactions, such as insufficient funds or exceeding available stock quantity.

3.1.4 Transaction History Component

The **Transaction History Component** provides users with a detailed log of all their past trading activities, enabling them to monitor their portfolio transactions and evaluate performance trends.

Features:

- **Comprehensive Transaction Log**: Displays a chronological list of all buy and sell transactions, including:
  - Stock name
  - Ticker symbol
  - Transaction type (BUY/SELL)
  - Quantity of shares traded
  - Price per share at the time of the transaction
  - Total transaction amount
  - Transaction date and time
  - Fetches data dynamically from the backend to ensure up-to-date records.

3.1.5 Profile Component

The **Profile Component** provides users with a centralized interface to manage and view their personal details and account settings.

Features

- **User Information Display**: Shows key profile details, including:
  - Full Name (First and Last Name)
  - Username
  - Email Address
  - Phone Number
  - City and State
  - Current Account Balance
  - Invested Amount
- **Profile Update**: Allows users to edit and update their profile details such as:
  - First Name

    - Last Name
    - Phone Number
    - City
    - State
    - Validates input fields for data integrity (e.g., valid phone numbers, required fields).
- **Password Management**: Securely enables users to change their password by:
    - Verifying the old password.
    - Setting a new password with validation (e.g., minimum length, special characters).

3.2 Styling and Design:

The interface is designed using custom CSS to ensure:

- **Responsiveness:** Optimized for different devices and screen sizes.
- **Professional Appearance:** Focused on delivering a clean, modern, and user-centric design.
- **CSS3 and SCSS:** For custom styling and design consistency across components.
- **Material-UI:** A popular React UI framework, used to implement pre-built UI components that ensure responsiveness and a modern aesthetic.
- **Chart.js / D3.js:** For interactive and visually appealing charts and graphs that display users' financial data and stock performance trends.

---

4. Backend Overview

4.1 API Routing

This API provides functionality for user authentication, portfolio management, stock data retrieval, and administrative operations. It is built with Node.js, Express, and PostgreSQL, integrating Alpha Vantage for real-time stock data.

Base URL

http://localhost:5000

4.1.1 Authentication Routes

Manages user authentication, including login, registration, and session management.

1. **POST /auth/register**

   Description: Registers a new user and stores user details securely in the database.

   Input:

   - Username: String
   - Password: String (hashed with bcrypt)

   Output:

   - Success message with the user object.
   - Error message if registration fails.

2. **POST /auth/login**

   Description: Authenticates a user by validating credentials and issues a JWT.

   Input:

   - username: String
   - password: String
   - captchaKey: String
   - userCaptcha: String

   Output:

   - Success message with user details and JWT.
   - Error message for invalid credentials or CAPTCHA.

3. **GET /auth/logout**

   Description: Logs out the user by invalidating their JWT.

   Output:

   - Success message.

4.1.2 Portfolio Management Routes

Handles stock buying, selling, and portfolio retrieval for users.

1. **POST /portfolio/buy**

   Description: Allows a user to purchase stocks, updates their portfolio, and recalculates their finances.

Input:

- userId: Integer
- symbol: String (stock ticker)
- quantity: Integer
- price: Float

Output:

- Updated portfolio details.
- Error message if insufficient funds or invalid data.

2. **POST /portfolio/sell**

   Description: Processes the sale of stocks and updates the user's portfolio.

   Input:

   - userId: Integer
   - symbol: String
   - quantity: Integer
   - price: Float

   Output:

   - Updated portfolio details with gain/loss data.
   - Error message if the user does not own enough shares.

3. **GET /portfolio**

   Description: Retrieves the current portfolio of a user, including holdings, stock prices, and performance.

   Headers:

   - userid: Integer

   Output:

   - Portfolio details with stock performance.
   - Error message if the user is not found.

4.1.3 Admin Operations Routes

Provides functionalities to manage user accounts and view system-wide data.

1. **GET /admin/users**

   Description: Retrieves a list of all registered users, including their roles and account status.

Headers:

- id: Admin ID (Integer)

Output:

- List of all users.

2. **POST /admin/add-user**

   Description: Allows an admin to add a new user to the system.

   Headers:

   - id: Admin ID (Integer)
   - **Input**:
   - username: String
   - password: String

   Output:

   - Success message with user details.
   - Error message if operation fails.

3. **DELETE /admin/delete-user/:id**

   Description: Deletes a user account from the system.

   Headers:

   - id: Admin ID (Integer)

   Path Parameter:

   - id: Integer (User ID)

   Output:

   - Success or error message.

4.1.4 Stock Data Routes

Integrates with Alpha Vantage API for real-time and historical stock data.

1. **GET /stocks**

   Description: Retrieves stock data for a list of symbols, either from the cache or Alpha Vantage API.

   Output:

- Array of stock data with latest and previous prices.
- Error message if the data retrieval fails.

2. **GET /stock/:symbol**

   Description: Fetches the latest stock price for a specific symbol.

   Path Parameter:

   - symbol: String (Stock ticker)

   Output:

   - Latest stock price and metadata.
   - Error message if the stock is not found.

4.2 Middleware

The backend uses a set of middleware components to handle cross-cutting concerns such as logging, authentication, and error management.

4.2.1 Logging

Winston is used as the logging framework to capture detailed logs for:

- Incoming API requests and responses
- Database interactions
- Errors and exceptions
- Logs are stored in files and can be integrated with external logging services for advanced monitoring and debugging.

4.2.2 Authentication Middleware

Validates user sessions by decoding and verifying the JWT provided with each request.

Implements **role-based access control (RBAC)** to restrict access to certain endpoints based on the user's role (e.g., admin vs. regular user).

Ensures secure access by checking for session expiration and invalidating compromised tokens.

---

5. Database Design

The PostgreSQL database is optimized for secure and efficient data management.

5.1 Key Tables

1. **Users Table:** Stores user credentials, roles (admin/user), and basic financial information. Ensures unique usernames and secure password storage using bcrypt hashing.
2. **Portfolio Table:** Maintains user-specific stock holdings, including quantities and purchase prices. Enforces data integrity using a composite primary key (`user_id` and `symbol`).
3. **Stock Cache Table:** Caches the latest stock prices and historical data for quick retrieval. Includes timestamps for tracking data currency.
4. **Transaction History Table:** Stores the transaction history of every user by the user ID which is the unique identifier for this table.

---

6. Features

6.1 Holdings Summary:

Displays all stocks owned by the user, also displays the user wallet balance and the invested amount of the user. Calculates the Net Profit/Loss of the user based on the current price and the purchased price of the shares owned.

Key Features:

- Displays the stocks owned along with the following details:
    - **Stock Symbol:** The ticker symbol (e.g., AAPL, TSLA).
    - **Quantity:** Number of shares held.
    - **Current Market Price:** Real-time stock price fetched from the Alpha Vantage API.
    - **Purchase Price:** Original price at which the stock was bought.
    - **Current Value:** Calculated as Quantity × Current Market Price.
    - **Gain/Loss:** Shows the percentage and absolute gain or loss for each stock and the overall portfolio.

6.2 Buy/Sell Stocks:

The Buy/Sell Stocks feature allows users to execute secure and real-time stock transactions directly from the platform.

Key Features:

- **Stock Selection:** Displays real-time stock information such as current price, daily high/low, and market trends.
- **Transaction Processing:**
- **Buy Stocks:** Users specify the stock symbol, quantity, and desired price. Validates available funds before completing the transaction. Updates the user's portfolio and recalculates total value and gains/losses.
- **Sell Stocks:** Users select the stock and quantity to sell. Validates sufficient holdings before completing the sale. Updates the portfolio and logs the transaction.

6.3 Graphical Insights

The **Graphical Insights** feature enhances the user experience by visualizing investment performance through interactive charts and graphs.

Key Features:

- **Historical Performance Graph:** Displays a line graph showing portfolio value over time, helping users track long-term performance.
- **Daily Gain/Loss Chart:** Bar chart representing daily gains and losses for quick analysis of short-term performance.
- **Asset Allocation Pie Chart:** Visual representation of how the user’s assets are distributed across various stocks or sectors, aiding in diversification analysis.
- **Interactive Elements:** Users can hover over data points to view detailed information, zoom in on specific time periods, and filter data based on custom criteria.

6.4 Administrative Features

The **administrative features** are designed to provide system administrators with full control over user accounts and system operations, ensuring smooth management and enhanced security.

6.4.1 User Management

The **User Management** module allows administrators to add, modify, and remove user accounts.

Key Features:

- **Add New Users:** Admins can create new user accounts. Input fields include username, and temporary password.
- **Delete Users:** Permanently remove inactive or unauthorized user accounts.

6.4.2 Audit Logging

The **Audit Logging** feature tracks all significant actions performed within the system, providing a comprehensive log for accountability and security.

Key Features:

- **User Activity Logs:** Tracks user logins, logouts, and failed login attempts. Records buy/sell transactions and portfolio modifications.
- **Administrative Actions:** Logs actions such as user creation, deletion, and role changes. Tracks configuration changes and database updates performed by administrators.
- **Log Storage and Access:** Logs are stored securely in the database and can be exported for external auditing. Admins can search and filter logs based on date, user, or action type.

6.3 Security Features

6.3.1 Password Security

Passwords are securely stored using modern hashing algorithms to protect against unauthorized access.

Key Features:

- **Password Hashing:** Passwords are hashed using bcrypt, a strong hashing algorithm that includes salting to protect against brute-force and rainbow table attacks.
- **Password Reset:** Users can request password resets via email, ensuring only the account owner can reset the password. Temporary reset tokens are time-limited and securely stored in the database.

6.3.2 Role-Based Access Control (RBAC)

Role-Based Access Control ensures that users only have access to the features and data appropriate to their role.

Key Features:

- **User Roles:**
    - **Admin:** Full access to all system features, including user management and audit logs.
    - **User:** Limited access to personal portfolio management and stock trading functionalities.
- **Endpoint Protection:**
    - Each API endpoint checks the user's role before allowing access, ensuring that sensitive operations are restricted to authorized users.

6.3.3 CAPTCHA Validation

The CAPTCHA system is implemented to prevent automated abuse, such as bot-driven account creation and brute-force login attempts. This basic CAPTCHA system is built directly within the backend code.

Key Features:

- **Custom CAPTCHA Implementation**: A simple CAPTCHA is generated and validated entirely on the backend.
- **Dynamic CAPTCHA Activation**: CAPTCHA validation is included in sensitive endpoints, such as login, to mitigate brute-force attacks.

7. Installation and Configuration

7.1 Prerequisites

- Node.js: v14.x or higher
- PostgreSQL: v13.x or higher
- npm: v6.x or higher

7.2 Backend Setup

- **Clone the repository:**

```
git clone https://github.com/tejas-manu/fox-of-hood
```

- **Install dependencies:**

```
npm install
```

7.3 Configure PostgreSQL:

- Create a database named myapp_db.
- Run the SQL scripts in src/backend/db/schema.sql to set up tables.
- Update the database connection in server.js:

```
const pool = new pg.Pool({
user: 'postgres',
host: 'localhost',
database: 'myapp_db',
password: 'your_password',
port: 5432,
});
```

7.4 Start the backend server:

- Navigate to the backend directory using the below code
  - `cd fox-of-hood/src/backend`
- Start the backend server using the following command
  - `node server.js`

7.5 Frontend Setup

- Open a new terminal and navigate to the root directory if not already on it:
  - ***cd fox-of-hood***
- Install dependencies:
  - ***npm install***
- Start the frontend server:
  - ***npm start***

Access the application at http://localhost:3000

---

8. Testing and Debugging

8.1 Testing

8.1.1 Frontend Testing

Frontend testing ensures that the user interface (UI) is responsive, functional, and provides a seamless user experience across different devices and browsers.

Test Scenarios:

- **User Authentication:**

  Verify user registration, login, and logout workflows.

  Test invalid login attempts with incorrect credentials and check error messages.

  Test CAPTCHA functionality during login and registration.

- **Portfolio Management:**

  Verify the display of user holdings, portfolio distribution, and real-time stock data.

  Test buy and sell transactions and validate that the changes are reflected in the UI.

  Check portfolio charts for accuracy and interactivity.

- **Testing Tools:**

  Manual Testing: Open the application in a web browser (e.g., Chrome, Firefox) and manually test each workflow.

8.1.2 Backend Testing

Backend testing focuses on ensuring that API endpoints function correctly, handle edge cases, and return the expected responses.

Test Scenarios:

- **Authentication APIs:**

  Test user registration, login, and logout endpoints.

  Verify that JWTs are generated, validated, and expired correctly.

- **Portfolio Management APIs:**

  Test endpoints for buying, selling, and retrieving portfolio data.

  Verify input validation for stock symbols, quantities, and prices.

  Ensure error handling for insufficient funds, invalid stock symbols, and database connection issues.

- **Admin APIs:**

  Test endpoints for adding, editing, and deleting user accounts.

  Verify that only administrators can access admin-specific endpoints.

- **Testing Tools:**

  Postman: Use Postman to manually test API endpoints by sending HTTP requests and verifying the responses.

  Test different HTTP methods (GET, POST, PUT, DELETE).

  Check for correct status codes (200, 400, 401, 403, 500) and response payloads.

8.1.3 Database Testing

Database testing ensures data integrity, consistency, and performance during various operations.

Test Scenarios:

- **Data Integrity:**

  Verify that user registration inserts correct data into the Users table.

  Check that buy/sell transactions update the Portfolio table accurately.

  Ensure that stock data is correctly cached in the Stock Cache table.

- **Data Retrieval:**

  Test SQL queries used for retrieving portfolio data and stock information.

  Verify that queries return correct and complete results for different users.

- **Testing Tools:**

  Manual SQL Queries: Use a GUI tool like **pgAdmin** to run SQL queries and verify the data manually.

8.2 Debugging Tools

Effective debugging tools are essential for identifying and resolving issues in the application.

8.2.1 Frontend Debugging

Frontend debugging focuses on identifying issues related to JavaScript, DOM manipulation, and UI rendering.

Debugging Tools:

- **Chrome DevTools:**
  - **Elements Tab:** Inspect and modify the DOM and CSS styles in real-time.
  - **Console Tab:** View error messages, warnings, and logs generated by the frontend code.
  - **Network Tab:** Monitor network requests and responses to ensure that API calls are successful and return the expected data.
  - **Performance Tab:** Analyze the performance of the frontend, identify slow-rendering components, and optimize page load times.

8.2.2 Backend Debugging

Backend debugging focuses on identifying issues related to API logic, database interactions, and server performance.

Debugging Tools:

- **Winston Logging:**
  - Use **Winston** to log important events, such as incoming requests, database queries, and errors.
  - Configure different log levels (info, warn, error) for better log management.
- **Node.js Debugger:**
  - Use the built-in Node.js debugger to set breakpoints and step through code execution.
  - Use a debugging tool like **Visual Studio Code** to attach to the running process and debug in an integrated development environment (IDE).

8.2.3 Database Debugging

Database debugging focuses on identifying issues with SQL queries, schema design, and data performance.

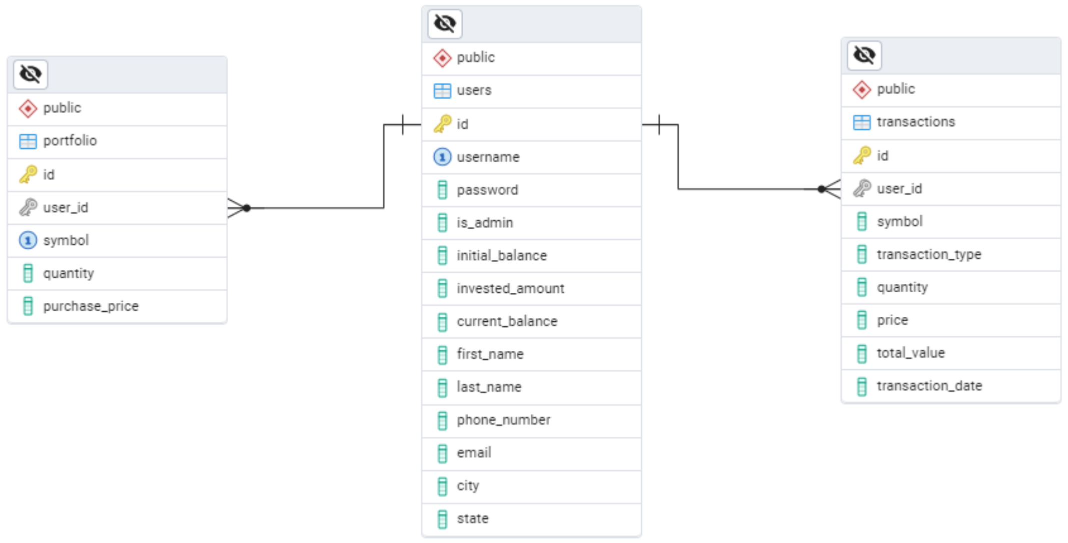
Debugging Tools:

- **SQL Query Logs:** Enable query logging in PostgreSQL to capture and analyze all queries executed by the application.

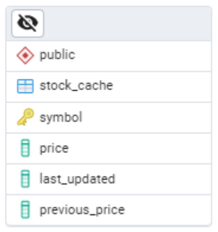
---

9. Other artifacts used

9.1 ER Diagram for Portfolio and user table



9.2 ER Diagram for Stock market Cache


public
stock_cache
symbol
price
last_updated
previous_price