**1.Introduction**

**Project Title:**

### STOCK SENSE: YOUR INTUITIVE STOCK TRADING PLATFORM

**Team Members:**

**K. SAI LAKSHMI (218x1a0440)TEAM LEADER**

**G. ANJALI KUMARI (218x1a0452) MEMBER 1**

**K. SOWMYA (218x1a0462) MEMBER 2**

**D. PRAVEEN KUMAR (218x1a0411)MEMBER 3**

**G. RAVINDRA (218x1a0466) MEMBER 4**

**2. PROJECT OVERVIEW**

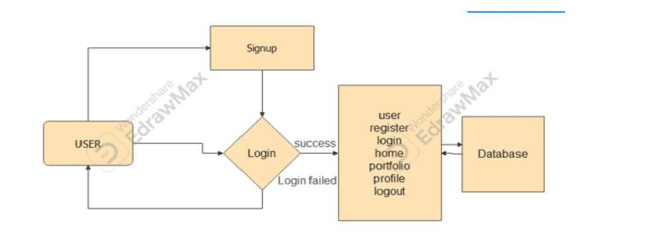
**Purpose:**

This project shows how to use the entire development process to create a personal expenses tracker application using Full-stack development with MongoDB as a database. The app aims to simplify personal financial management by providing users with a seamless platform where they can effectively track their expenses. Through a user-friendly interface user an access and allocate expenses, view analytics, set budget goals and receive timely reports. The application leverages MongoDB’s focused data model to provide powerful data management and scalability. Integration of Front-End Technologies such as HTML, CSS and JavaScript with Back-End Technologies such as Node.js, Express.js and MongoDB increase effort and efficiency. The project not only demonstrates knowledge of development as a whole, but also meet requirements of good financial management in today’s world. The personal expenses tracker addresses the contemporary need for streamlined personal finance management to the Dynamics lifestyle of modern users.

**Key Features:**

* Secure User Authentication
* Easy Expense Management
* Detailed Expense Analysis
* Budget Planning with Notifications
* Comprehensive Functionality
* Enhanced Security Features
* User Friendly Interface
* Seamless Navigation

**3.ARCHITECTURE**

****

Designing a system architecture for a stock sensing application involves creating a structure that supports real-time data processing, user interactions, and scalability. Here’s a high-level overview of a typical architecture, including key components and their interactions:

###### **Client Side (Front End):**

* + **Web Application**: Built using frameworks like React, Angular, or Vue.js, providing the user interface for displaying stock data, charts, and analytics.
  + **Mobile Application**: A cross-platform app developed using React Native or Flutter to access stock data on mobile devices.

###### **API Layer**

**Restful API / Graph QL:**

* + This layer handles requests from the client-side applications, processing them and returning the necessary data.
  + Built using frameworks like Express.js (Node.js), Flask, or Django.

###### **Business Logic Layer (Backend)**

**Application Server:**

* + This component processes business logic, including user authentication, data processing, and business rules.
  + Technologies: Node.js, Python (Django/Flask), or Java (Spring Boot).

###### **Data Layer**

**Database:**

* + **Relational Database** (e.g., Postgre SQL or My SQL) for structured data such as user profiles, transaction history, and stock metadata.
  + **No SQL Database** (e.g., Mongo DB or Redis) for unstructured data, caching, and real-time analytics.

###### **Real-Time Data Processing**

* + **Web Socket / Socket.IO**: For real-time updates on stock prices and market data, allowing the client to receive instant notifications.
  + **Message Queue** (e.g., Rabbit MQ or Apache Kafka): Used for handling asynchronous data processing and ensuring reliable message delivery.

1. **Setup Instructions**

To develop a full-stack web application for a stock trading web app Application.System using React.js, Node.js, and MongoDB, several essential prerequisites must be installed. These include setting up Node.js and npm for backend development, installing MongoDB for database management, and configuring React.js for the frontend. Additionally, necessary dependencies such as Express.js for API handling, Mongoose for database interaction, and React Router for navigation should be installed. Proper setup of authentication, state management, and UI libraries will ensure a smooth and efficient development process.Node.js and npm.

Node.js is essential for running JavaScript on the server side, and npm (Node Package Manager) is used for managing project dependencies.

* 1. Download Node.js: Download Node.js
  2. Installation Instructions: Install Node.js via Package Manager

1. **MongoDB**

MongoDB is the NoSQL database used to store data such as users, appointments, Payments and

doctors. You can either install MongoDB locally or use a cloud-based MongoDB service like

MongoDB Atlas.

• **Download MongoDB:** Download MongoDB Community Edition

• **Installation Instructions:** MongoDB Installation Guide

1. **Express.js**

Express.js is a web framework for Node.js that simplifies server-side development by providing tools

for routing, middleware, and API development.

• **Install Express.js:** Open your terminal or command prompt and run the following command: npm install express

1. **React.js**

React.js is the JavaScript library used to build the frontend user interface. React enables the

development of dynamic, component-based applications that allow for fast and responsive user

experiences.

**Steps to Set Up React:**

1. **Create a New React Project:**

• Install the Create React App tool, which sets up a new project with all required

configurations: npx create-react-app client

**2. Navigate to the Project Directory:** cd client

**3. Start the React Development Server:**

• Launch the development server by running:

npm run dev

• Open your browser and go to http://localhost:3000 to view your running React app

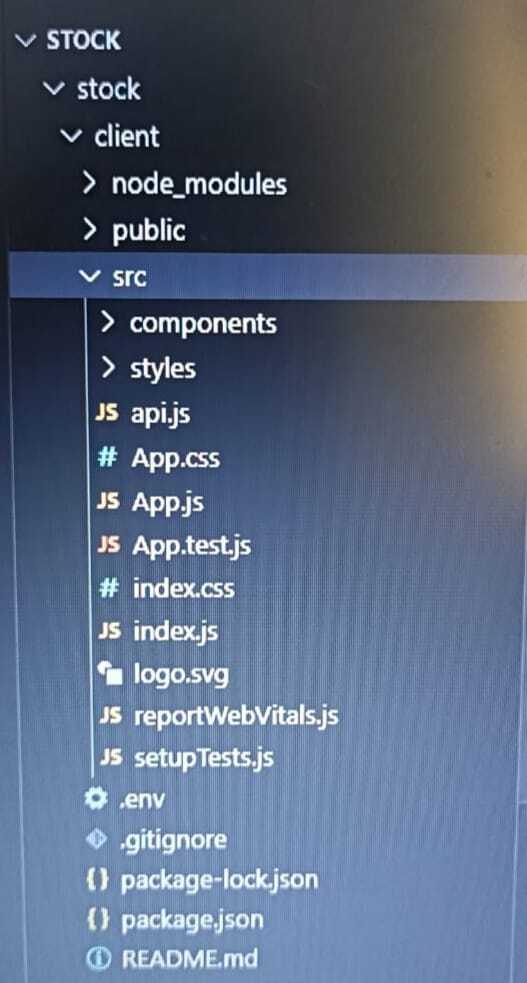
* **Navigate to the project directory**: cd server
* **Start the running of server with**: node server.js
* Open your browser then register and run the trading platform.

**5.Folder Structure**

The core structure of the React frontend project typically looks like this:

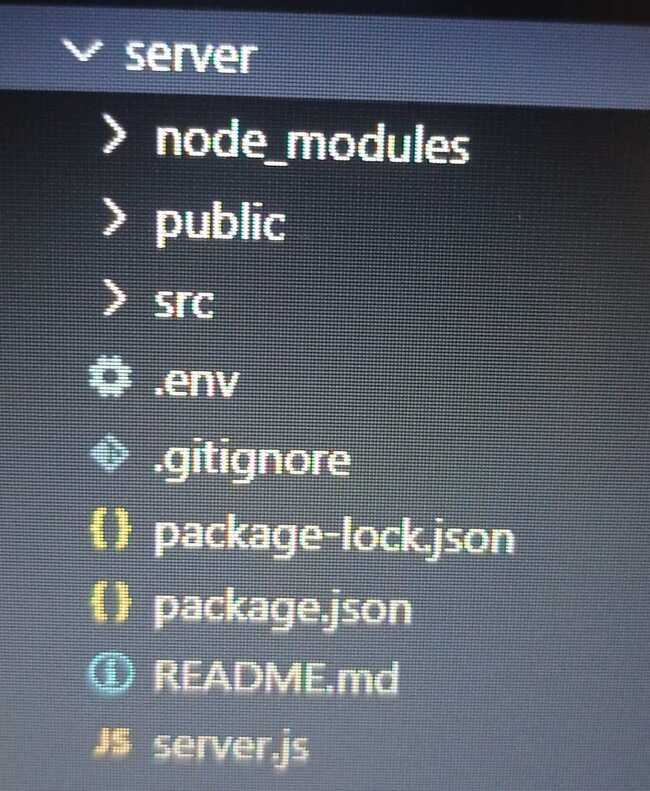
**Client directory:**

The below directory structure represents the directories and files in the client folder (front end) where, react.js is used along with API’S .

****

The core structure of the React Backend project typically looks like this:

**Server directory:**

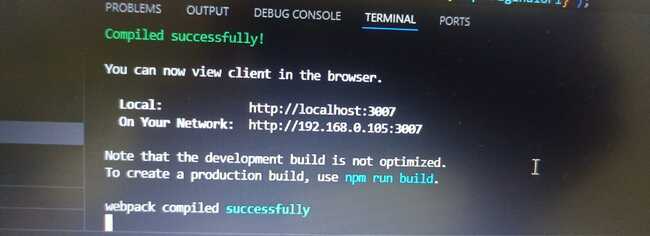
****

1. **Running The Application**

Running the Full-Stack Application:

1. **Once both the frontend and backend are set up**:
2. Start frontend:

• Navigate to the frontend directory and run: npm start

****

Run npm run dev in the React project directory (client) and open

http://localhost:3000 in your browser.

1. Start backend:

* Navigate to the backend directory and run: node server.js



3.Database Operations:

• Ensure MongoDB is running locally or via MongoDB Atlas for remote access.

1. **Development Environment:**

Choose a code editor or Integrated Development Environment (IDE) that suits your

preferences, such as Visual Studio Code.

• Visual Studio Code: Download fromhttps://code.visualstudio.com/download

**API DOCUMENTATION:**

**Backend Development:**

Here’s a high-level API documentation built on the MERN stack, detailing six key API categories without delving into code examples. Each section includes the request methods, parameters, and expected responses.

**1. User Authentication:**

Endpoints:

* Register: POST /auth/register
* Description: Registers a new user on the platform.

Parameters:

* name (required): Full name of the user.
* email (required): Email address.
* password (required): Password for the account.
* Response: Confirmation message with user details.
* Login: POST /auth/login.
* Description: Authenticates the user and provides a JWT token for session management.

Parameters:

* email (required): User’s email address.
* password (required): User’s password.
* Response: JWT token and basic user info.
* Get Profile: GET /auth/me.
* Description: Retrieves the current authenticated user’s profile information.
* Headers: Authorization: Bearer <JWT token>
* Response: User profile data (name, email, balance, etc.).

**2. Stock Market Data:**

Endpoints:

* Get Stock Data: GET /stocks/:symbol.
* Description: Retrieves real-time market data for a specific stock.

Parameters:

* symbol (required): Stock ticker symbol (e.g., AAPL, TSLA).
* Response: Real-time stock data, including current price, volume, open, high, low, and change percentage.
* Top Gainers and Losers: GET /stocks/top-gainers-losers.
* Description: Fetches the top market performers and decliners for the day.
* Response: A list of the top 5 gainers and losers in the stock market.

**3. Portfolio Management:**

Endpoints:

* Get Portfolio: GET /portfolio.
* Description: Retrieves the user’s stock portfolio, including holdings and balance.
* Headers: Authorization: Bearer <JWT token>
* Response: Detailed user portfolio with stock symbols, shares, average price, and total balance.
* Add Stock to Portfolio: POST /portfolio/add.
* Description: Adds a new stock to the user’s portfolio.

Parameters:

* symbol (required): Stock ticker symbol.
* shares (required): Number of shares purchased.
* price (required): Purchase price per share.
* Headers: Authorization: Bearer <JWT token>
* Response: Confirmation of the stock being added to the portfolio, including updated portfolio data.

**4. Trade Execution:**

Endpoints:

* Buy Stock: POST /trade/buy.
* Description: Executes a buy order for a stock.

Parameters:

* symbol (required): Stock ticker symbol.
* shares (required): Number of shares to buy.
* price (required): Price at which to buy the shares.
* Headers: Authorization: Bearer <JWT token>
* Response: Confirmation of the executed buy order, updated portfolio, and remaining balance.
* Sell Stock: POST /trade/sell.
* Description: Executes a sell order for a stock.

Parameters:

* symbol (required): Stock ticker symbol.
* shares (required): Number of shares to sell.
* price (required): Selling price per share.
* Headers: Authorization: Bearer <JWT token>
* Response: Confirmation of the executed sell order, updated portfolio, and remaining balance.

**5. Watch list Management:**

Endpoints:

* Get Watch list: GET /watch list.
* Description: Retrieves the user’s watch list of stocks.
* Headers: Authorization: Bearer <JWT token>
* Response: List of stock symbols currently in the user’s watch list.
* Add Stock to Watch list: POST /watch list/add.
* Description: Adds a new stock to the user’s watch list.

Parameters:

* symbol (required): Stock ticker symbol.
* Headers: Authorization: Bearer <JWT token>
* Response: Confirmation of the stock being added to the watch list.

**6. Notifications:**

Endpoints:

* Get Notifications: GET /notifications.
* Description: Fetches user-specific notifications such as trade confirmations.
* stock alerts, and system messages.
* Headers: Authorization: Bearer <JWT token>
* Response: A list of notifications relevant to the user.

**AUTHENTICATION:**

Most of these API endpoints require authentication using JWT tokens, which are passed in the Authorization header.

**Authentication Flow:**

**1. User Registration:**

* User provides email, username, and password.
* Password is hashed and stored in the database.
* Verification email sent to user's email address.

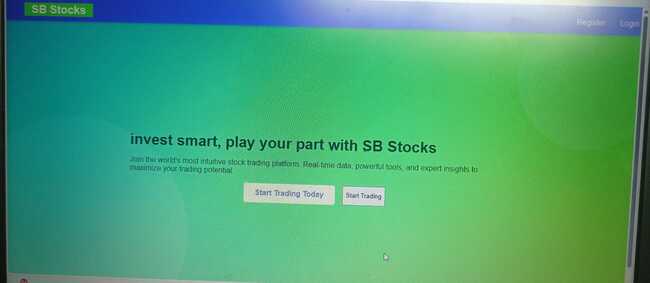
**2. User Login:**

* User enters email and password.
* Password is verified against stored hash.
* If valid, JSON Web Token (JWT) is generated.

**USER INTERFACE:**

**Homepage:**

Default landing page for user where he will get the information about the project



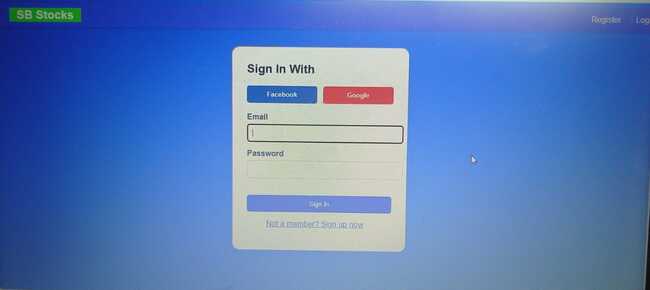
**Register page:**

Save user data to the database and send a confirmation email



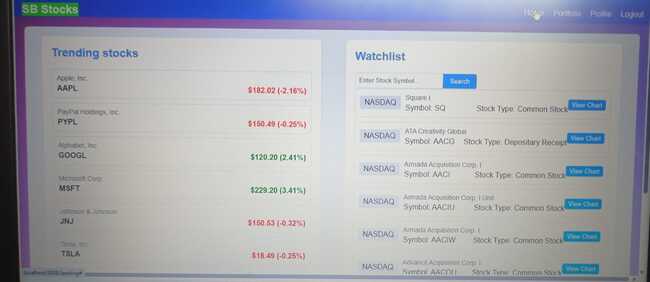
**Login page:**

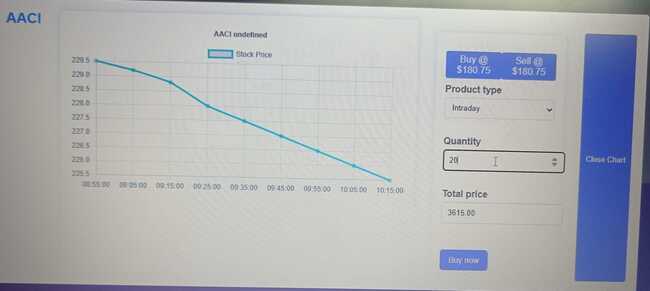
Authenticate user and start session



**Home:**

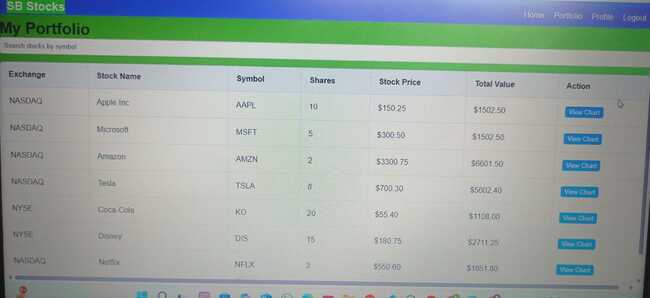
It contains trending stocks and watch list of a user.





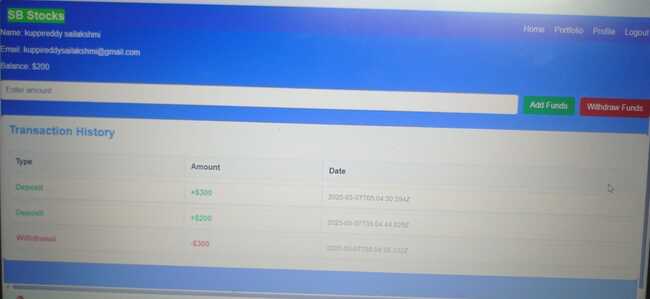
**Portfolio:**

They can view the user transaction related to stock.



**Profile:**

The user can see the transaction history and the balance amount.



**TESTING:**

**System Testing:**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product .Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Types of Tests:**

**Unit testing:**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing:**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing:**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Functional testing:**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**White Box Testing:**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing:**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Test objectives:**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested:**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page**.**

**TEST CASES:**

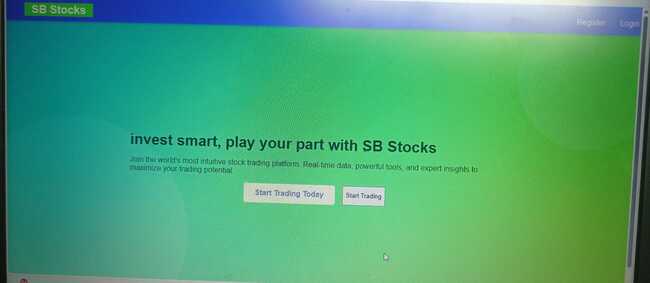
**User Module:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Precondition** | **Test Steps** | **Expected Result** |
| TC01 | Register a new user | User is on the registration page | 1. Enter valid user details (e.g., username, email, password).  2. Click on "Register" button. | User is successfully registered, and a confirmation message appears. |
| TC02 | Register with existing email | User is on the registration page | 1. Enter details with an existing email.  2. Click on "Register" button. | Registration fails with an error message stating that the email is already in use. |
| TC03 | Login with valid credentials | User is on the login page | 1. Enter correct email and password.  2. Click on "Login" button. | User is successfully logged in and redirected to the dashboard. |
| TC04 | Login with invalid credentials | User is on the login page | 1. Enter incorrect email or password.  2. Click on "Login" button. | Login fails with an error message indicating incorrect email or password. |
| TC08 | View user profile | User is logged in | 1. Navigate to the "Profile" section. | User profile details (e.g., name, email, etc.) are displayed. |
| TC09 | Update profile information | User is logged in and on Profile page | 1. Click on "Edit Profile".  2. Modify profile details.  3. Click on "Save" button. | Profile information is updated, and a success message is shown. |
| TC10 | Logout from the application | User is logged in | 1. Click on the "Logout" button. | User is logged out, and the session is terminated, redirecting to the login page. |

**SCREENSHOTS OR DEMO:**

**Homepage:**

Default landing page for user where he will get the information about the project



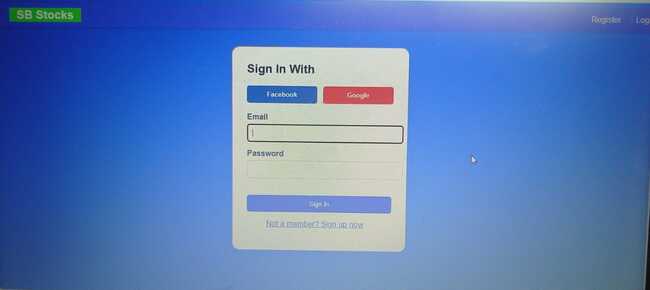
**Register page:**

Save user data to the database and send a confirmation email



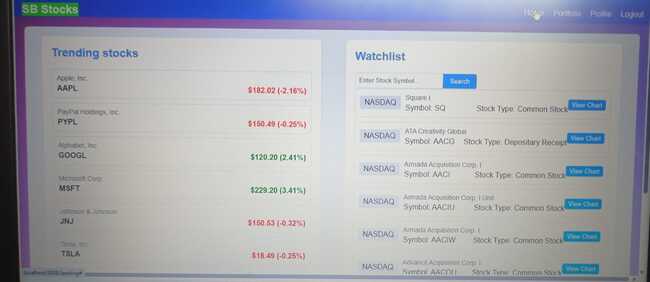
**Login page:**

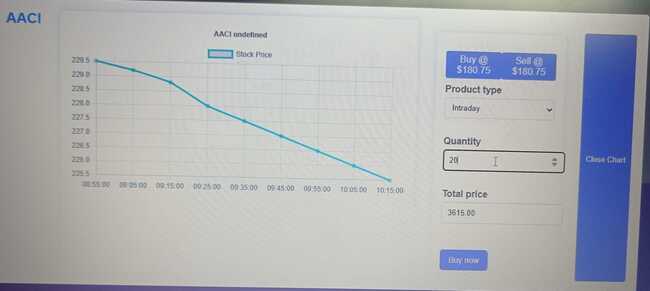
Authenticate user and start session



**Home:**

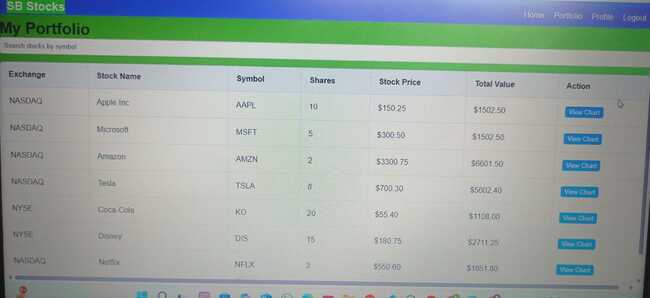
It contains trending stocks and watch list of a user.





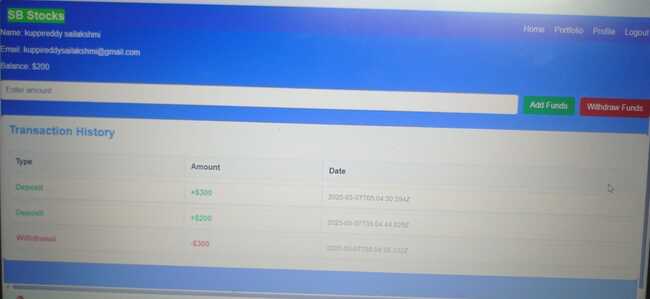
**Portfolio:**

They can view the user transaction related to stock.



**Profile:**

The user can see the transaction history and the balance amount.



1. **known issues:**

### 1. ****Latency and Slow Response Times****

* **Issue**: Delays in receiving stock quotes or executing trades can affect trading outcomes, especially in fast-paced markets.
* **Impact**: It could lead to missed opportunities or unexpected trade executions at undesirable prices.
* **Solution**: Optimize the backend and reduce the number of external API calls. Use a content delivery network (CDN) for faster access.

### 2. ****Real-Time Data Inaccuracy****

* **Issue**: Inaccurate or delayed data for stock prices, order books, or market depth can affect users’ decisions.
* **Impact**: Incorrect data can result in poor investment decisions, causing financial losses.
* **Solution**: Integrate with reliable, real-time data providers, and ensure the app is consistently updated with correct information.

### 3. ****Margin and Leverage Risks****

* **Issue**: Some trading apps allow users to trade on margin or use leverage. If not properly managed, this can result in significant losses.
* **Impact**: Users may lose more than their initial investment, leading to debt or bankruptcy.
* **Solution**: Ensure clear risk warnings are displayed, and include features for automatic margin calls or liquidation to minimize excessive risk exposure.

### 4. ****Account Management and Withdrawals****

* **Issue**: Delays or complications in withdrawing funds or accessing account data can frustrate users.
* **Impact**: Users may feel their funds are not safe or accessible when needed.
* **Solution**: Ensure withdrawal processes are clear and secure, with timely customer support for any issues.

### 5. ****Lack of Support for Advanced Orders****

* **Issue**: Basic apps may not support complex order types like stop-loss orders, limit orders, or trailing stops.
* **Impact**: Traders may be limited in their ability to manage risk or execute strategies effectively.
* **Solution**: Ensure the platform supports a variety of order types to suit different trading strategies.

### 6. ****Mobile App Limitations****

* **Issue**: Some web apps may not have well-optimized mobile versions, leading to a subpar experience on smartphones.
* **Impact**: Mobile traders may experience delays, lack of features, or difficulty navigating the app.
* **Solution**: Ensure the app is responsive and mobile-friendly, providing a smooth experience across all devices.

**FUTURE ENHANCEMENT:**

As the world of stock trading continues to evolve, several enhancements can be made to further refine and elevate the SB Stocks platform experience. First, integrating real-time market data would provide users with accurate and up-to-the-minute information, making the paper trading experience more reflective of actual market conditions. Additionally, incorporating social trading and leaderboards would enable users to follow and learn from top-performing traders, encouraging community engagement and peer-driven learning.To further enhance the platform's capabilities, AI-powered trading insights could be introduced to offer personalized stock suggestions based on market trends and user preferences. Expanding the app's accessibility by developing a mobile version for IOS and Android would allow users to manage their portfolios and execute trades on the go.