**FULL STACK DEVELOPMENT WITH MERN**

Submitted in accordance with the requirement for the degree of

**BACHELOR OF TECHNOLOGY IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Under the faculty Guideship**

**MRS.B.NAGA VARDHINI**

**DEPARTMENTOFCOMPUTERSCIENCE ANDENGINEERING**

**KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY**

(Autonomous)

**Approved by AICTE- New Delhi, Accredited by NAAC with ‘A’ Grade**

**Permanently Affiliated to JNTUK, Kakinada**

**NH-5, Chowdavaram, Guntur – 522019**

**2021-2025**

**FULL STACK DEVELOPMENT WITH MERN**

**PROJECT DOCUMENTATION**

**PROJECT TITLE:**

**STOCK SENSE:YOUR INTUITIVE STOCK TRADING PLATFORM**

****

**Submitted in complete fulfillment of the requirement by**

**TEAM MEMBERS**

|  |  |
| --- | --- |
| **218X1A0536** | **K.KEERTHI PRIYA** |
| 218X1A0513 | D.VAMSI KUMAR |
| 218x1a0555 | G. SAI AYYAPA REDDY |
| 228X5A0506 | J.SOMASEKHARA SRIRAM PRASAD |

**LIST OF CONTENTS :**

|  |  |  |
| --- | --- | --- |
| **1.** | **PROJECT OVERVIEW** | * **PURPOSE** * **FEATURES** |
| **2.** | **ARCHITECTURE** | * **FRONTEND** * **BACKEND** * **DATABASE** |
| **3.** | **SETUP INSTRUCTIONS** | * **PREREQUISITES** * **INSTALLATION** |
| **4.** | **FOLDER STRUCTURE** | * **CLIENT** * **SERVER** |
| **5.** | **RUNNING THE APPLICATION** | * **FRONTEND** * **BACKEND** |
| **6.** | **API DOCUMENTATION** |  |
| **7.** | **AUTHENTICATION** |  |
| **8.** | **USER INTERFACE** |  |
| **9.** | **TESTING** |  |
| **10.** | **SCREENSHOTS OR DEMO** |  |
| **11.** | **KNOWN ISSUES** |  |
| **12.** | **FUTURE ENHANCEMENTS** |  |

**PROJECT OVERVIEW:**

**PURPOSE:**

1. Ease of Use: Deliver a clean, easy-to-navigate interface for users to trade, track, and analyze stocks effortlessly.

2. Real-Time Market Data: Provide up-to-date stock market information, charts, and trends to enable informed decision making.

3.Mobile-First Experience: Ensure the platform is optimized for mobile devices, allowing users to trade on the go.

**FEATURES:**

**CORE FEATURES:**

1. User-Friendly Interface: Simple, intuitive design for effortless trading.

2. Real-Time Market Data: Live market data, news, and analysis.

**ANALYTICS AND INSIGHTS:**

1. Fundamental Analysis: Company financials, ratios, and estimates.

2. Technical Analysis: Identify trends and patterns with technical indicators.

**RISK MANAGEMENT:**

1. Risk Assessment Tools: Evaluate potential risks and rewards.

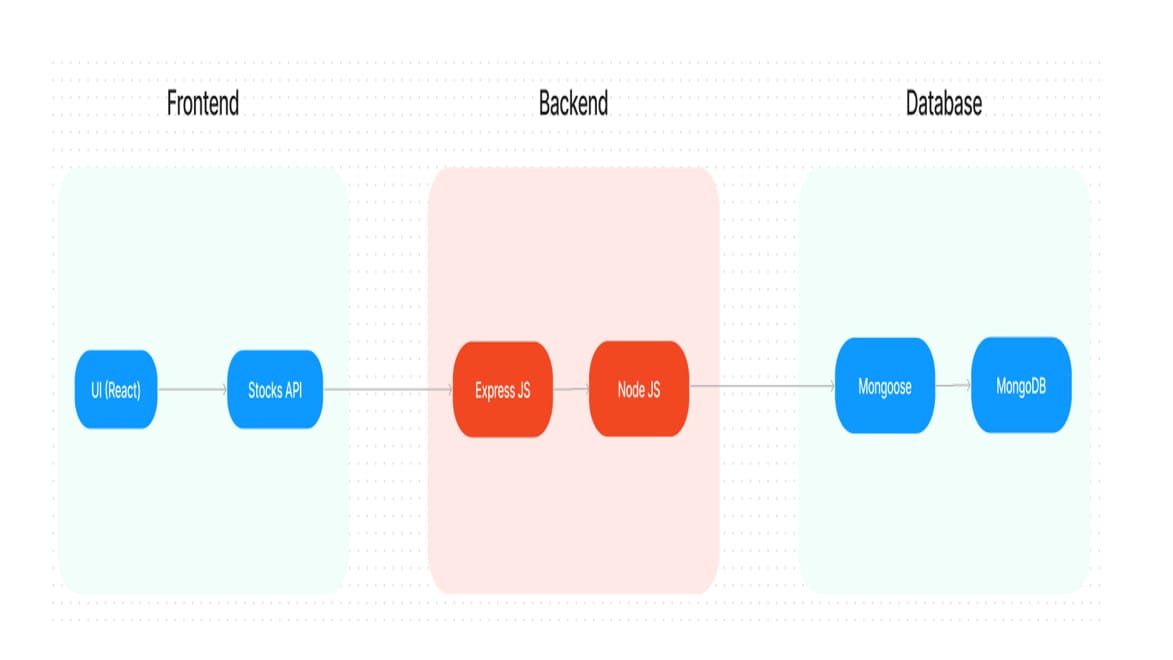
2. Stop-Loss Orders: Automate risk management.

**MOBILE AND ACCESSEBILITY:**

1. Mobile App: Access Stock Sense on-the-go

2. Web Platform: Accessible on desktop, tablet, and mobile devices.

**ARCHITECTURE:**



1. Frontend (React): The frontend of StockSense can be built using modern JavaScript frameworks like React, Angular, or Vue.js for a responsive, intuitive, and dynamic user experience. It communicates with the backend via RESTful APIs or GraphQL to fetch real-time stock data, execute trades, and display market insights.

2. Backend (Node.js/Express.js): The backend, built using Node.js and Express, is designed using a microservices architecture. Each service (e.g., user management, stock data, trading, notifications) operates independently and communicates via APIs.

3. Database (MongoDB,Mangoose): MongoDB is used as the primary database to store and manage user profiles, transaction histories, stock data, and market trends.

**SETUP INSTURCTIONS:**

**PREREQUISITES:**

**1. Proficiency in JavaScript and MERN Stack:**

- Strong understanding of \*JavaScript\* as it is the core language for both frontend and backend.

- Familiarity with the \*MERN stack\* (MongoDB, Express.js, React.js, Node.js) for full-stack development:

- \*Node.js\* for server-side logic.

- \*Express.js\* to build APIs.

- \*React.js\* for the frontend user interface.

- \*MongoDB\* for database management.

**2. API Integration for Stock Data:**

- Understanding how to integrate \*third-party stock market APIs\* (e.g., Alpha Vantage, IEX Cloud) to fetch real-time stock prices and other financial data.

- Ability to work with \*REST APIs\* or \*WebSockets\* for real-time data updates.

**3. User Authentication & Authorization:**

- Implement secure user authentication using \*JWT (JSON Web Tokens)\* or \*OAuth\* for login, registration, and session management.

- Understanding of role-based access controls (RBAC) to secure sensitive operations, such as placing trades.

**4.Database Design & Management:**

- Knowledge of \*MongoDB\* for designing and managing database schemas, such as user profiles, transaction history, and stock data.

- Implement basic CRUD operations and ensure \*data consistency\* for financial transactions.

**5. Frontend Design & Data Visualization:**

- Familiarity with \*React.js\* for building interactive and responsive UIs.

- Experience in using \*charting libraries\* like D3.js or Chart.js to visualize stock data, portfolio performance, and market trends in real time.

These key areas will ensure that you can effectively build and manage a stock trading platform using the MERN stack.

**INSTALLATION:**

To set up \*StockSense\* using the \*MERN stack\* (MongoDB, Express.js, React, Node.js), follow this main step-by-step guide. I’ll break it down into key stages for the backend, frontend, database, and deployment without code snippets.

**1. Initial Setup for MERN Stack:**

a. Backend (Node.js & Express) Setup:

- Install \*Node.js\*: Ensure Node.js is installed on your machine.

- Set up your \*project folder\*:

1. Navigate to your directory and initialize a new Node.js project:

npm init

2. Install Express and other dependencies:

npm install express mongoose cors dotenv

- Set up \*server.js\* to configure your Express server.

**b. Database (MongoDB) Setup:**

- Create an account on \*MongoDB Atlas\* for a cloud-based database solution.

- Create a new \*Cluster\* and a \*database\* named something like "StockSense".

- Add your IP to the allowed IP list in MongoDB Atlas, and get your \*connection URI\*.

- In your project, use mongoose to connect to MongoDB by setting up a .env file to store sensitive data like your database URI.

**c. Frontend (React) Setup:**

- Use \*create-react-app\* to scaffold a new React application:

npx create-react-app stocksense-client

- Set up the basic structure for your frontend including:

- A landing page

- Stock search and display functionality

- Buy/Sell functionality (with components)

**d. Linking Backend & Frontend:**

- Use \*Axios\* or \*Fetch API\* to make HTTP requests from the React frontend to your Express backend for things like:

- Stock price retrieval

- Stock transactions (buy/sell)

- User portfolio management

- Implement \*CORS\* middleware in Express to allow your React frontend to communicate with your backend.

**2. Key Components for StockSense:**

**a. User Authentication:**

- Set up \*JWT (JSON Web Tokens)\* or any other preferred authentication method to manage user accounts and sessions.

- Implement signup, login, and user profile management.

**b. Stock Market Data API:**

- Integrate with a \*stock market data API\* (like Alpha Vantage, Yahoo Finance, or IEX Cloud) to fetch real-time stock prices and data.

- Set up routes in your Express backend to handle stock search and buy/sell transactions.

**c. Database Models:**

- Create MongoDB models for:

- \*User\* (name, email, password, portfolio)

- \*Transactions\* (buy/sell, stock ticker, number of shares, price)

- \*Portfolio\* (current holdings, value)

**d. Trading Logic:**

- On the backend, implement logic for buying and selling stocks.

- When a user buys a stock, update their portfolio and create a new transaction entry.

- When a user sells, update their portfolio accordingly and remove stocks from their holdings.

**3. Deployment:**

**a. Frontend Deployment:**

- You can deploy the React frontend using services like \*Netlify\* or \*Vercel\*. Simply build the React project and connect to your hosting service via GitHub or similar.

**b. Backend Deployment:**

- Deploy your Node.js backend using \*Heroku, \*\*Render\*, or any other preferred cloud service. Make sure to set up the MongoDB connection in your deployment settings.

**c. Environment Variables:**

- Ensure your \*API keys, MongoDB URI, and any sensitive information are stored securely using environment variables in services like Netlify or Heroku.

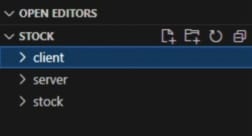
**d. Integrate Frontend with Backend:**

- Update your React app to make API calls to the deployed backend URL (from Heroku/Render) instead of the local server.

By following this high-level setup, you’ll have a working stock trading platform with the MERN stack.

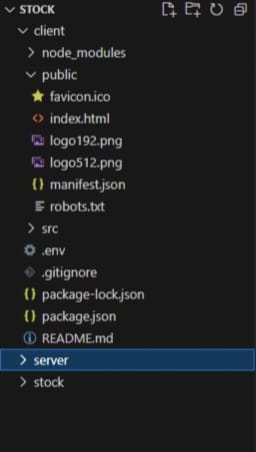
**FOLDER STRUCTURE:**

Inside the Stock Sense trading platform ,we have the following folders:



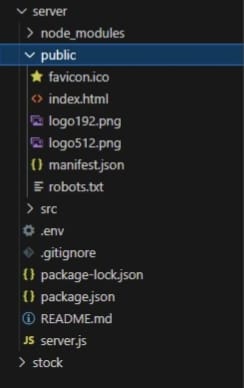
**CLIENT DIRECTORY:**

The below directory structure represents the directories and files in the client folder (front end) where, react js is used along with Api’s such as socket.io and agora.



**SERVER DIRECTORY:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io Api.



**RUNNING THE APPLICATION:**

It provides commands to start the frontend and backend servers locally.

**FRONTEND:**

npm start in the client directory.

**BACKEND:**

npm start in the server directory.

**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. Watchlist Management:**

Endpoints:

- Get Watchlist: GET /watchlist

- Description: Retrieves the user’s watchlist of stocks.

- Headers: Authorization: Bearer <JWT token>

- Response: List of stock symbols currently in the user’s watchlist.

- Add Stock to Watchlist: POST /watchlist/add

- Description: Adds a new stock to the user’s watchlist.

-Parameters:

- symbol (required): Stock ticker symbol.

- Headers: Authorization: Bearer <JWT token>

- Response: Confirmation of the stock being added to the watchlist.

**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.

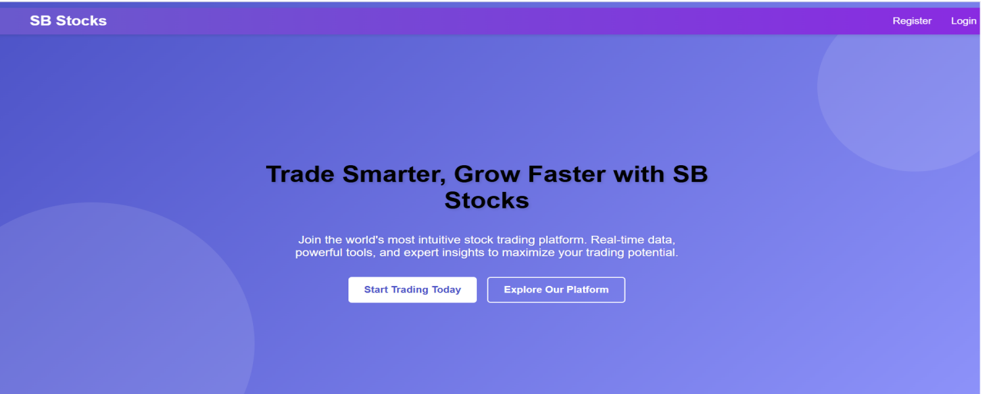
- Password is verified against stored hash.

- If valid, JSON Web Token (JWT) is generated.

**USER INTERFACE:**

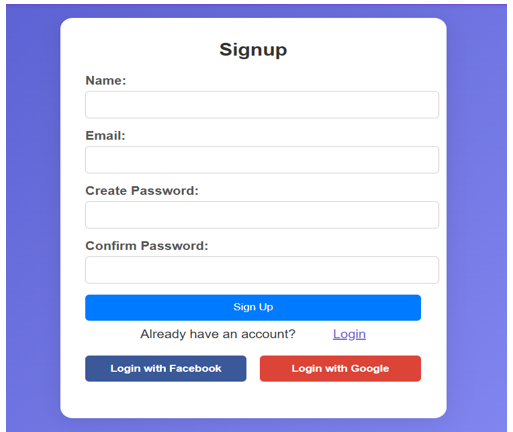
**Homepage:**

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



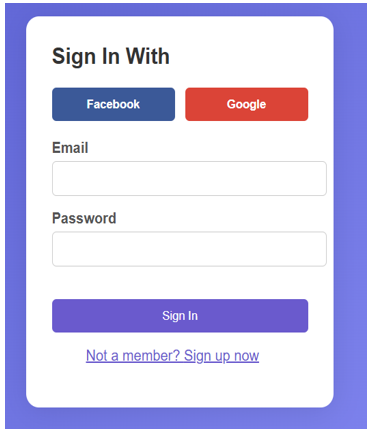
**Register page:**

Save user data to the database and send a confirmation email

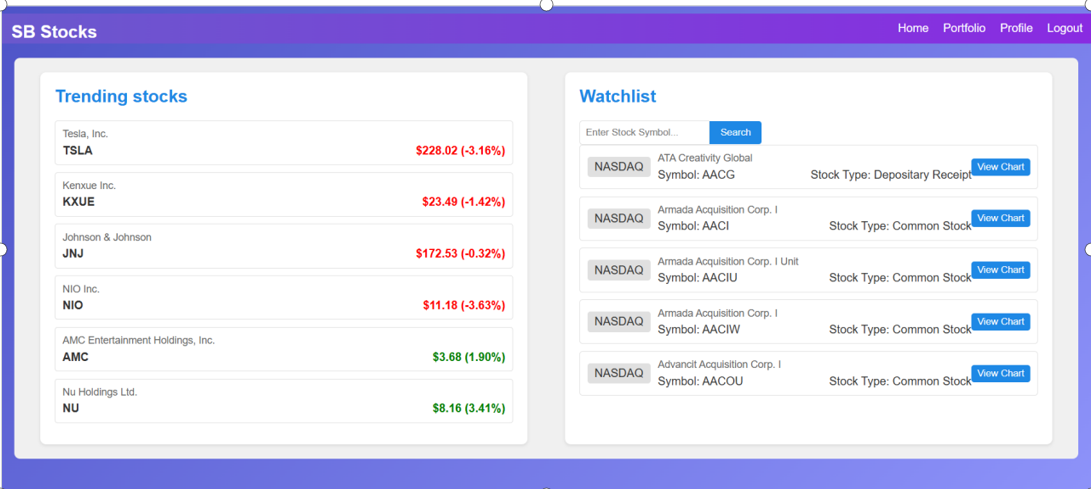


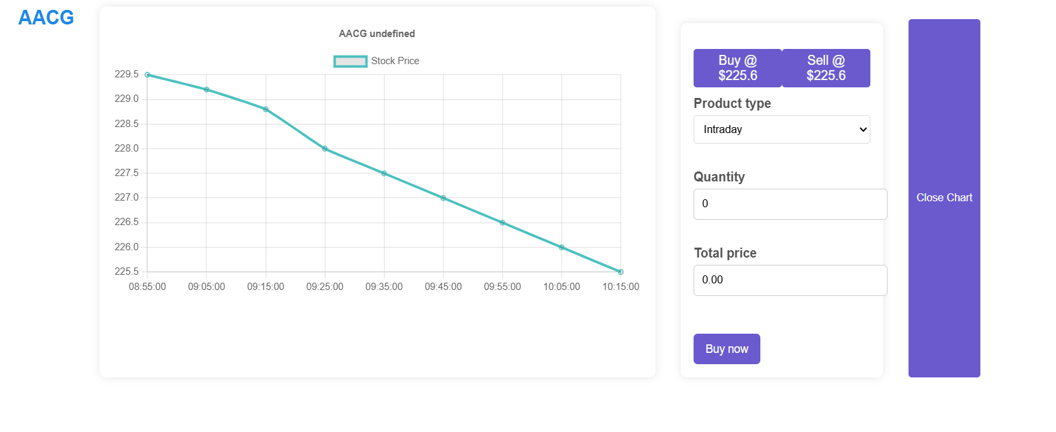
**Login page:**

Authenticate user and start session

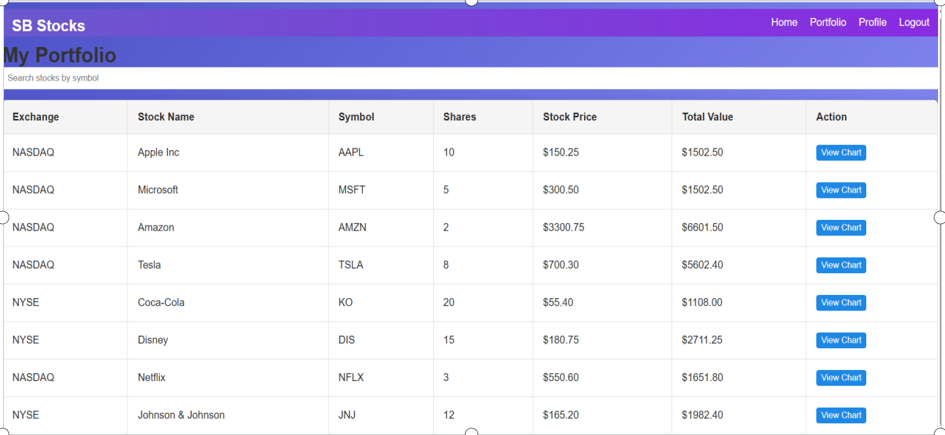


**Home:** It contains trending stocks and watchlist of a user

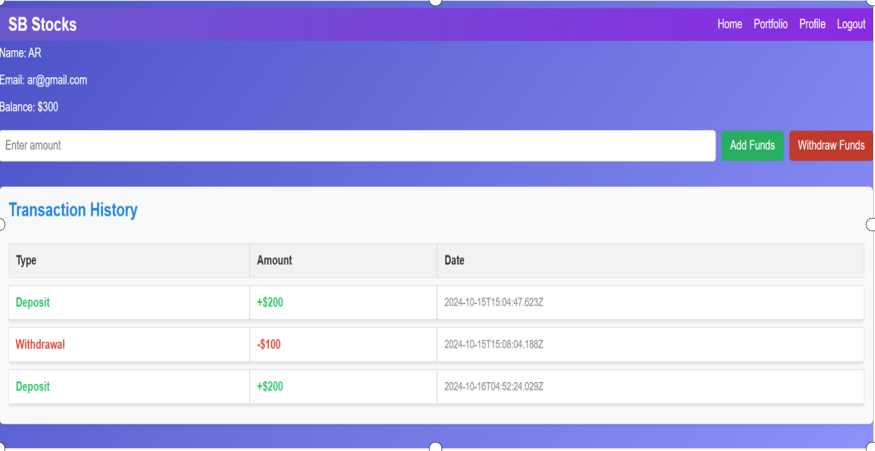




**Portfolio:** They can view the user transaction related to stock.



**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 It is the process of exercising software with the intent of ensuring that the

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 rsesponses 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**.**

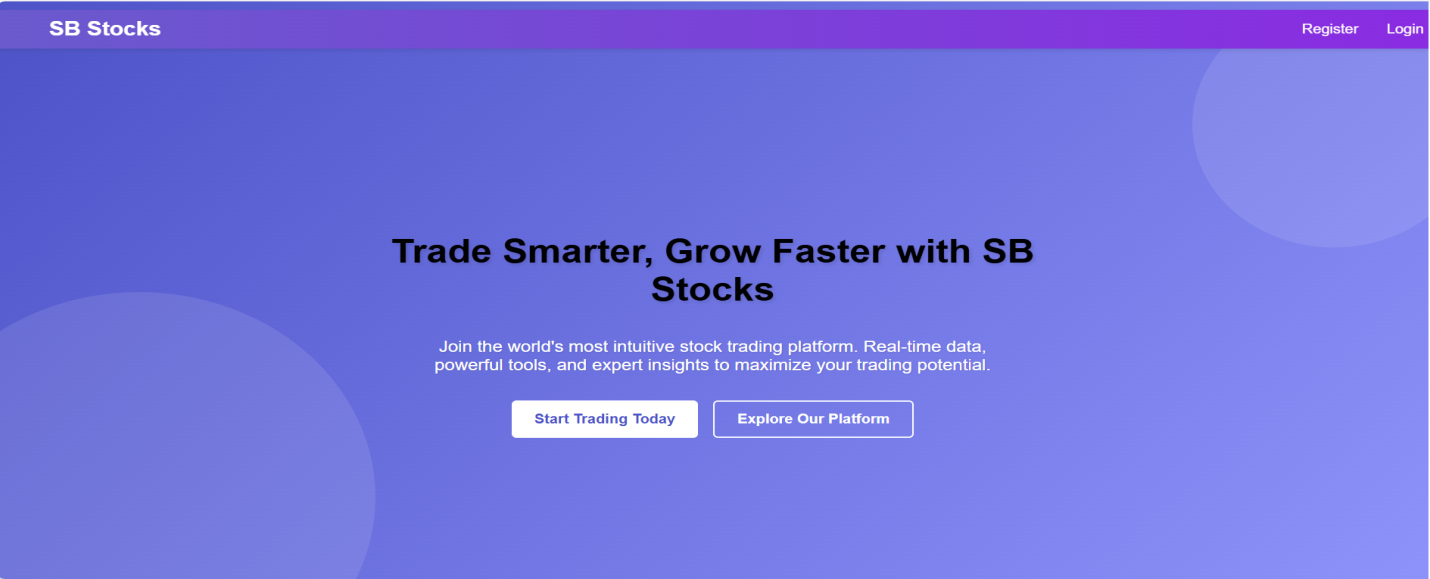
**TEST CASES:**

**User Module:**

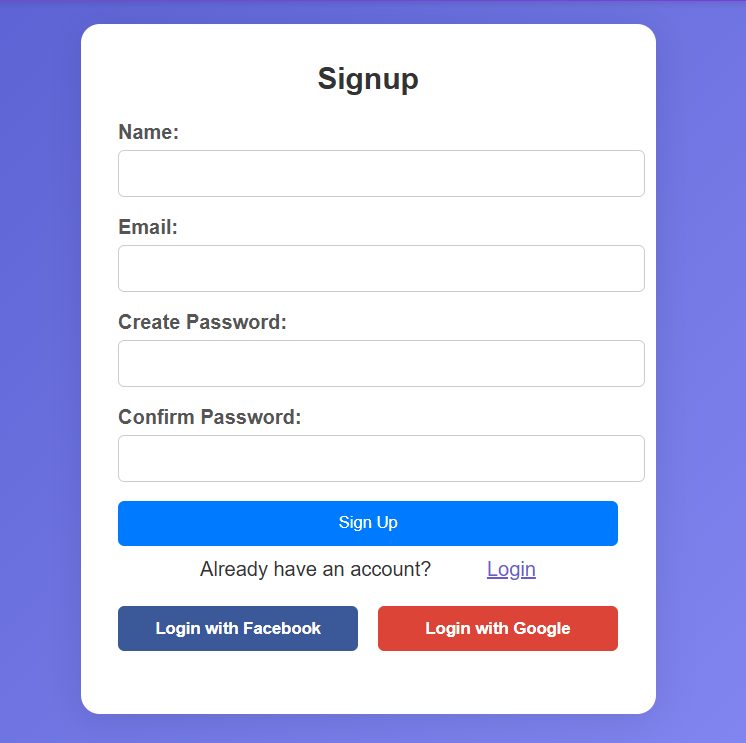
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Precondition** | **Test Steps** | **Expected Result** |
| TC01 | Register a new user | User is on the registration page | 1. Enter valid user details (e.g., username, email, password).  2. Click on "Register" 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:**

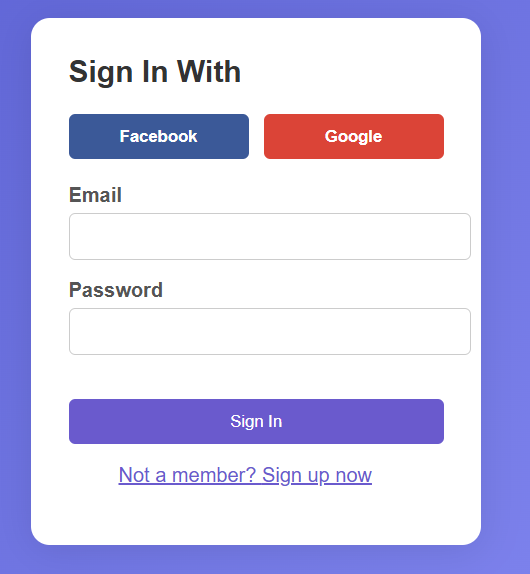
**Landing page:** default page for all users



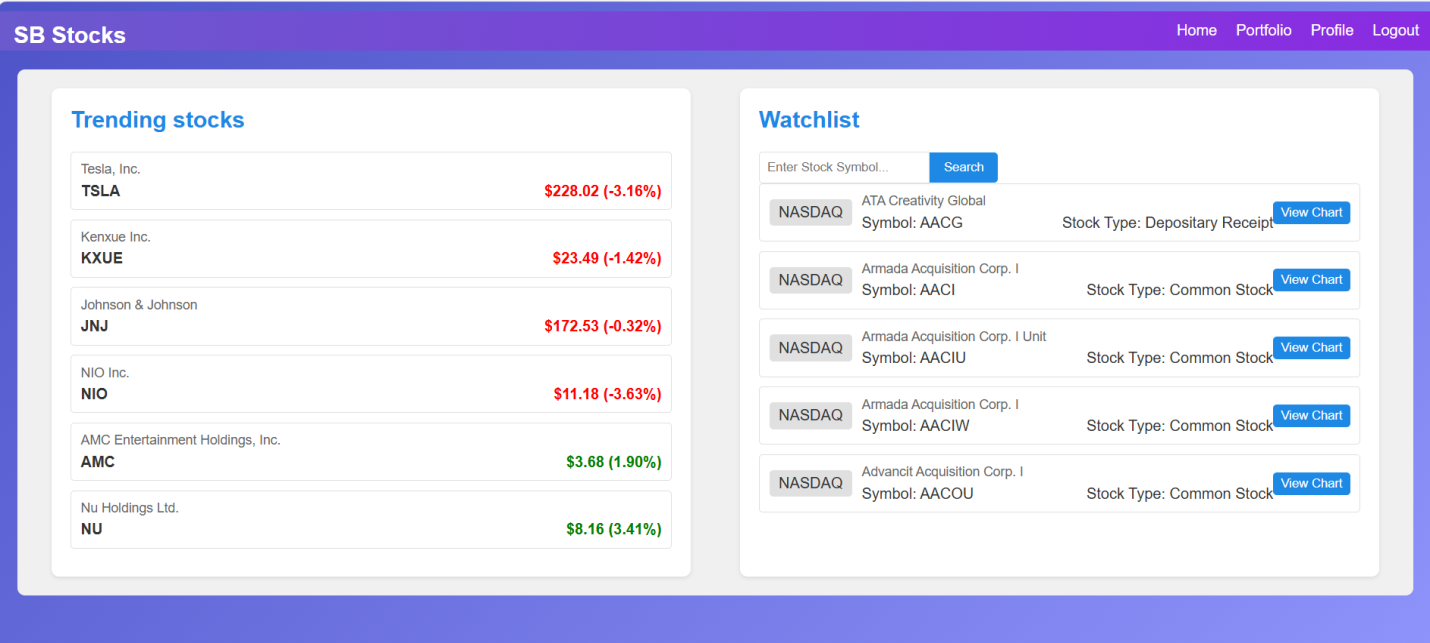
**Register:** save the user details to the database



**Login:** Authenticate user and start the session.



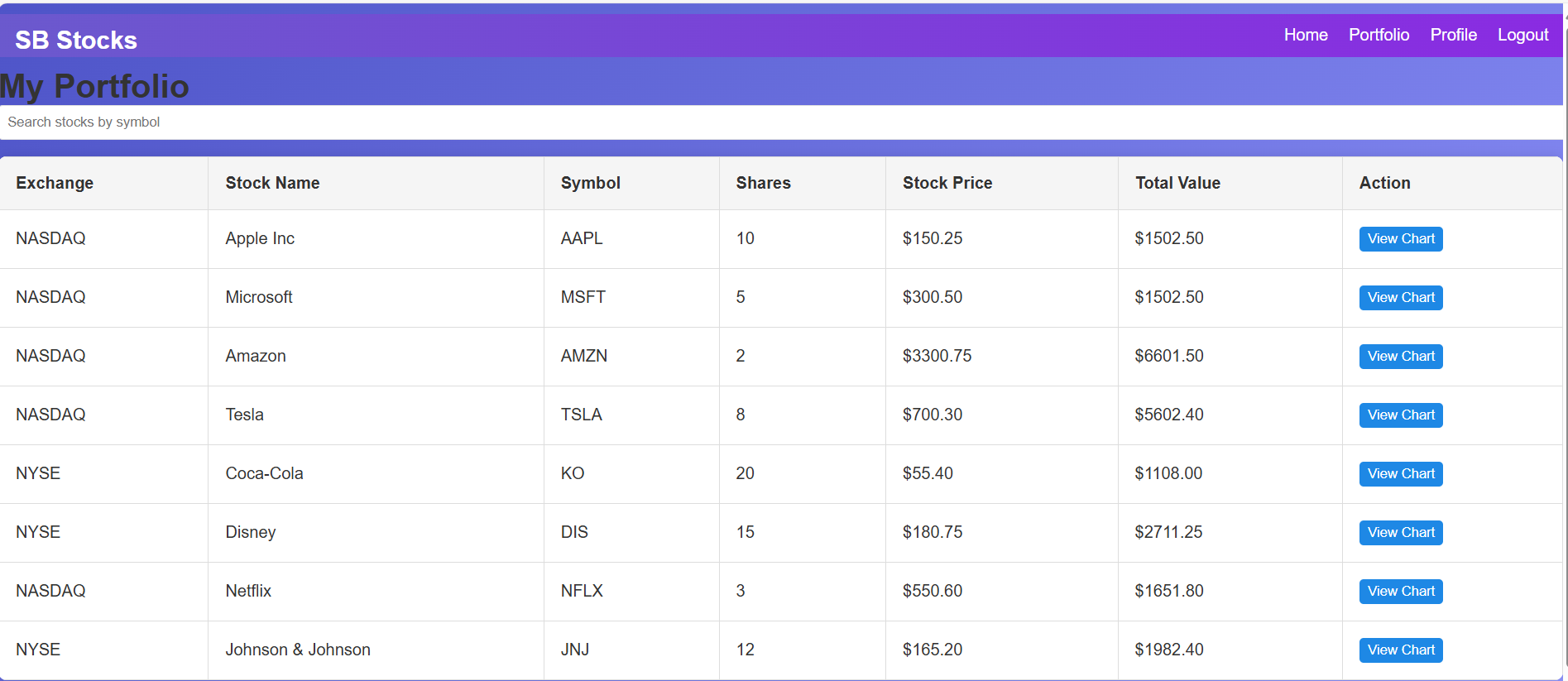
**Home:** It contains trending stocks and watchlist of a user



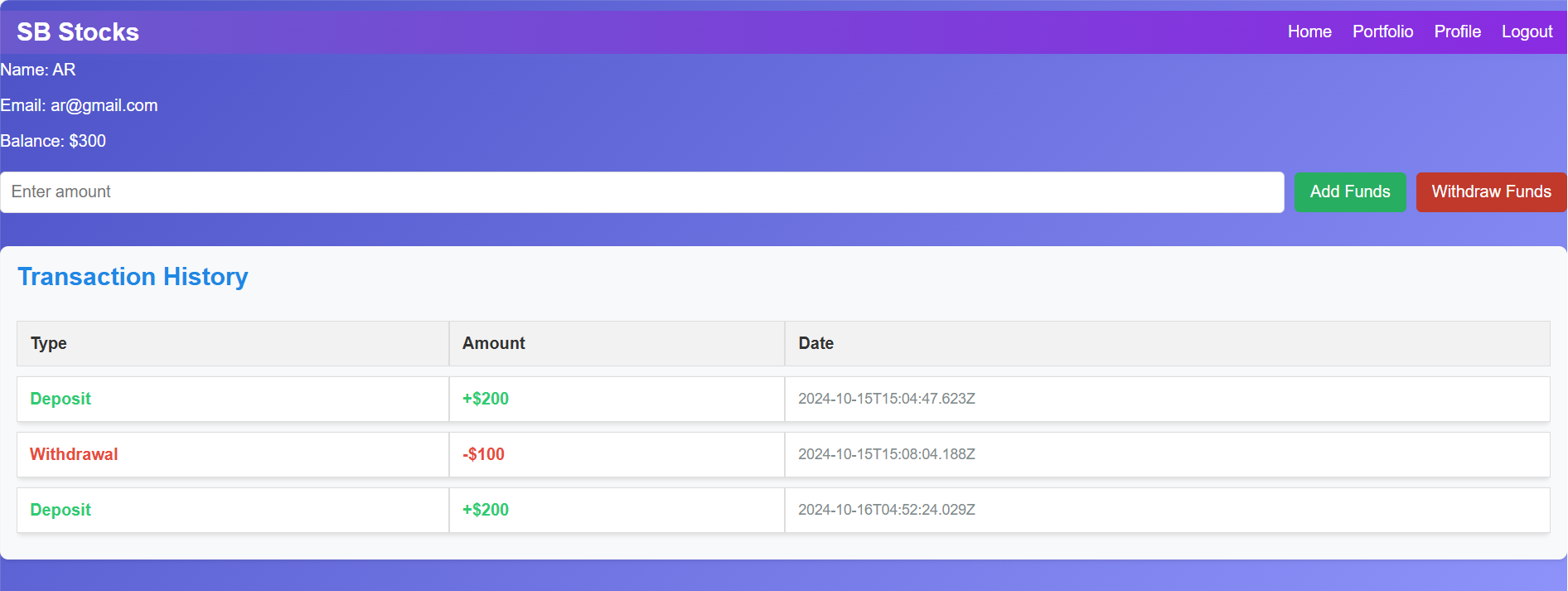
By clicking on view chart we can buy or sell the stocks



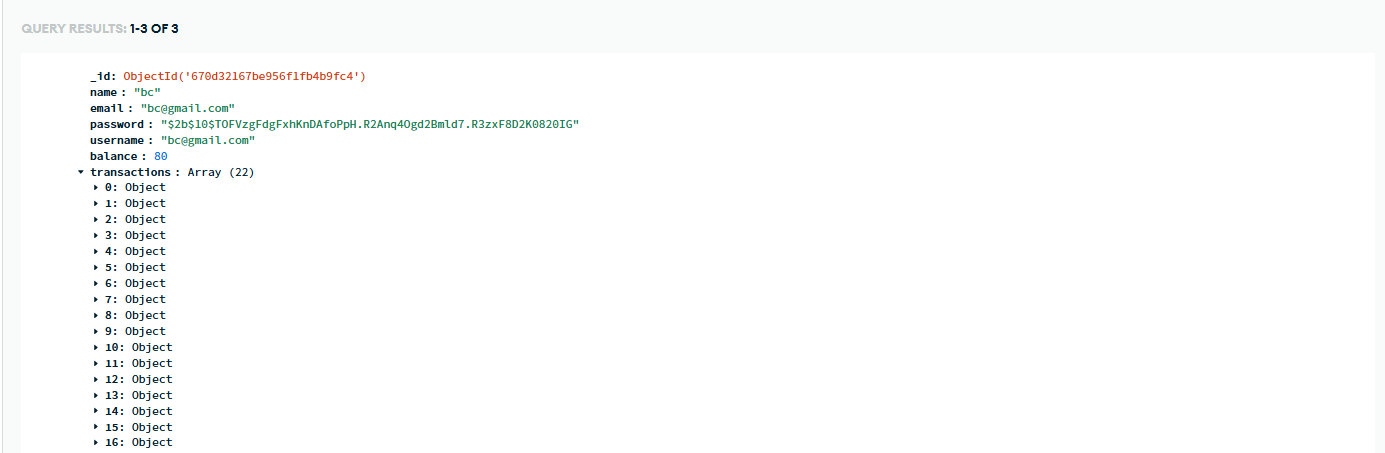
**Portfolio:** They can view the user transaction related to stock.



**Profile:** The user can see the transaction history and the balance amount



**DATABASE VISUALIZING IN MONGODB:**

****

**CONCLUSION:**

SB Stocks is designed to transform your stock trading experience, offering a seamless and user-friendly platform for aspiring and experienced traders alike. Our web app provides easy access to a wide array of stocks from the US Stock market, allowing users to practice their trading skills through paper trading without the financial risks.
With real-time stock prices, historical data, and detailed performance analytics, SB Stocks ensures that you can make well-informed trading decisions. Whether you're building virtual portfolios or testing new strategies, our platform delivers a realistic trading environment for refining your skills.

**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.