**Full Stack Development with MERN**

**Project Documentation format**

**1. Introduction**

**• Project Title:** Stock Trading Web App

**• Team Members:**

Adithya: Project Manager

Mukundan: Lead Frontend Developer

Kirthick: Backend Developer

Anish: Database Administrator

**2. Project Overview**

**• Purpose:**

The purpose of the Stock Trading Web App is to provide users with a robust platform to manage and trade stocks. The app aims to offer real-time data, efficient trading options, and comprehensive portfolio management tools.

**• Features:**

User registration and authentication

Real-time stock price updates

Buy and sell stocks

Portfolio management

Transaction history

Interactive charts and graphs

**3. Architecture**

• Frontend:

The frontend is built using React, a popular JavaScript library for building user interfaces. It uses Redux for state management, React Router for navigation, and Axios for making HTTP requests.

• Backend:

The backend is developed using Node.js and Express.js. It provides RESTful APIs for the frontend to interact with. The backend handles user authentication, stock transactions, and data retrieval.

• Database:

MongoDB is used as the database for this application. The database schema includes collections for users, stocks, transactions, and portfolios. Mongoose is used for object data modeling (ODM).

**4. Setup Instructions**

• Prerequisites:

Node.js (v14 or above)

MongoDB (v4 or above)

Git

• Installation:

Clone the repository: git clone https://github.com/yourusername/stock-trading-web-app.git

Navigate to the project directory: cd stock-trading-web-app

Install frontend dependencies: cd client && npm install

Install backend dependencies: cd ../server && npm install

Set up environment variables: Create a .env file in the server directory and add the required variables.

**5. Folder Structure**

• Client:

The React frontend is organized as follows:

client/

├── public/

├── src/

│ ├── components/

│ ├── pages/

│ ├── redux/

│ ├── App.js

│ ├── index.js

• Server:

The Node.js backend is organized as follows:

server/

├── controllers/

├── models/

├── routes/

├── utils/

├── app.js

├── server.js

**6. Running the Application**

• To start the frontend server:

Bash:

cd client

npm start

• To start the backend server:

Bash:

cd server

npm start

**7. API Documentation**

**User Management**

* **GET /api/user/-** Retrieves user information by ID. Using the ID, we fetch the rest of user data like email, username and password.
* **PUT /api/user/**- Updates user information by ID.
* Parameters:json

{ "username": "john\_doe", "password": "securepassword" }

* Response:json

{ "message": "User registered successfully" }

**Transactions**

* **POST /api/deposit** – Finds the user and updates Transaction History with the current balance.
* **POST /api/withdraw** – Finds the user and updates Transaction History and balance.

**Buying/Selling Stocks**

* **POST /api/buyStocks** – Buys new Stocks.
* **POST /api/sellStocks** – Sells Stocks.
* Parameters: None
* Response:

[ { "symbol": "AAPL", "price": 150 }, { "symbol": "GOOGL", "price": 2800 }]

**Transactions and Users**

* **GET /api/transactions** - Retrieves all transactions from the MongoDB Collection.
* **GET /api/users** - Retrieves all users from the MongoDB Collection.

**8. Authentication**

• Authentication and authorization are handled using JWT (JSON Web Tokens).

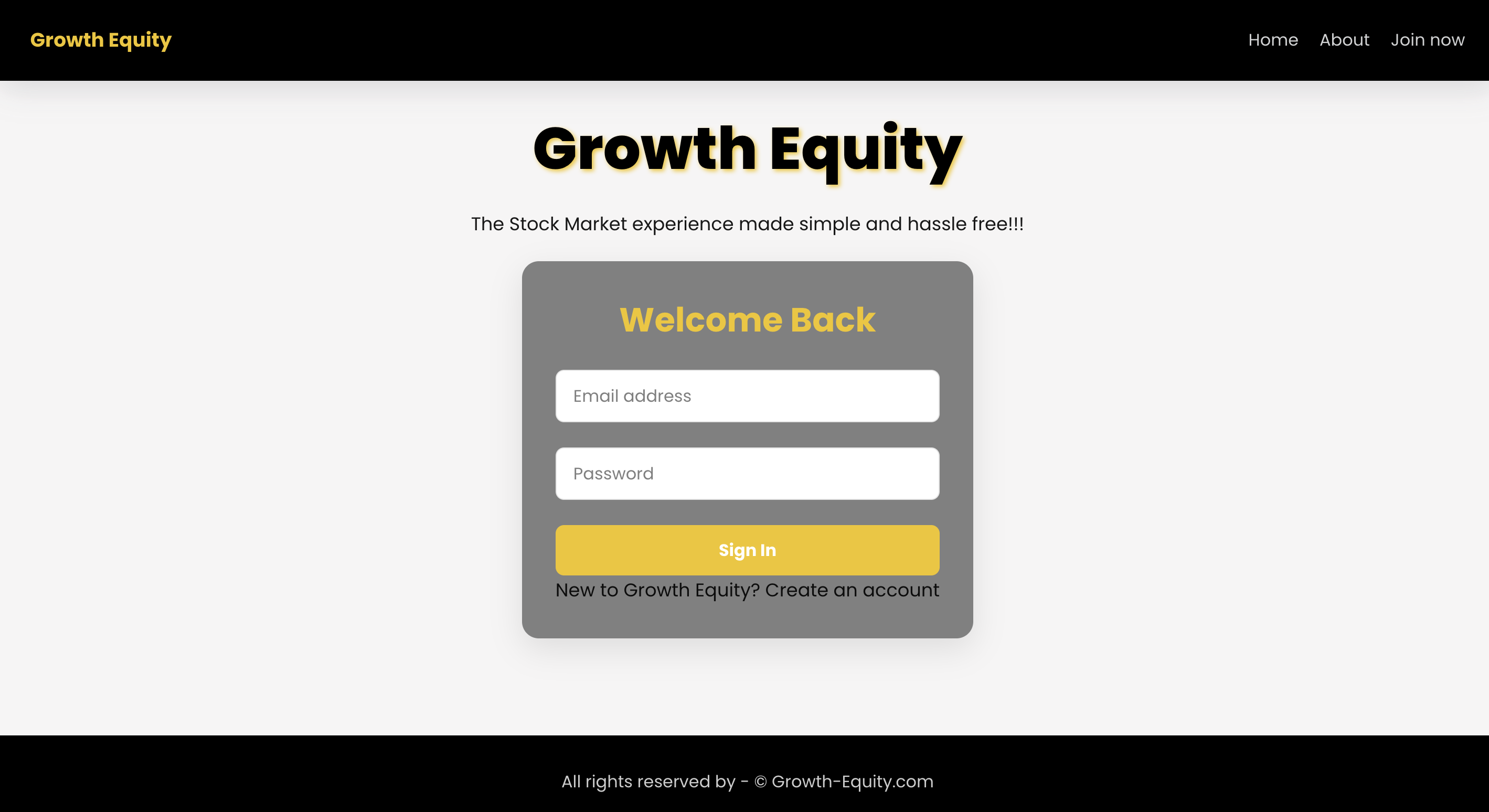
• Upon successful login, a token is generated and sent to the client. This token must be included in the Authorization header of subsequent requests.

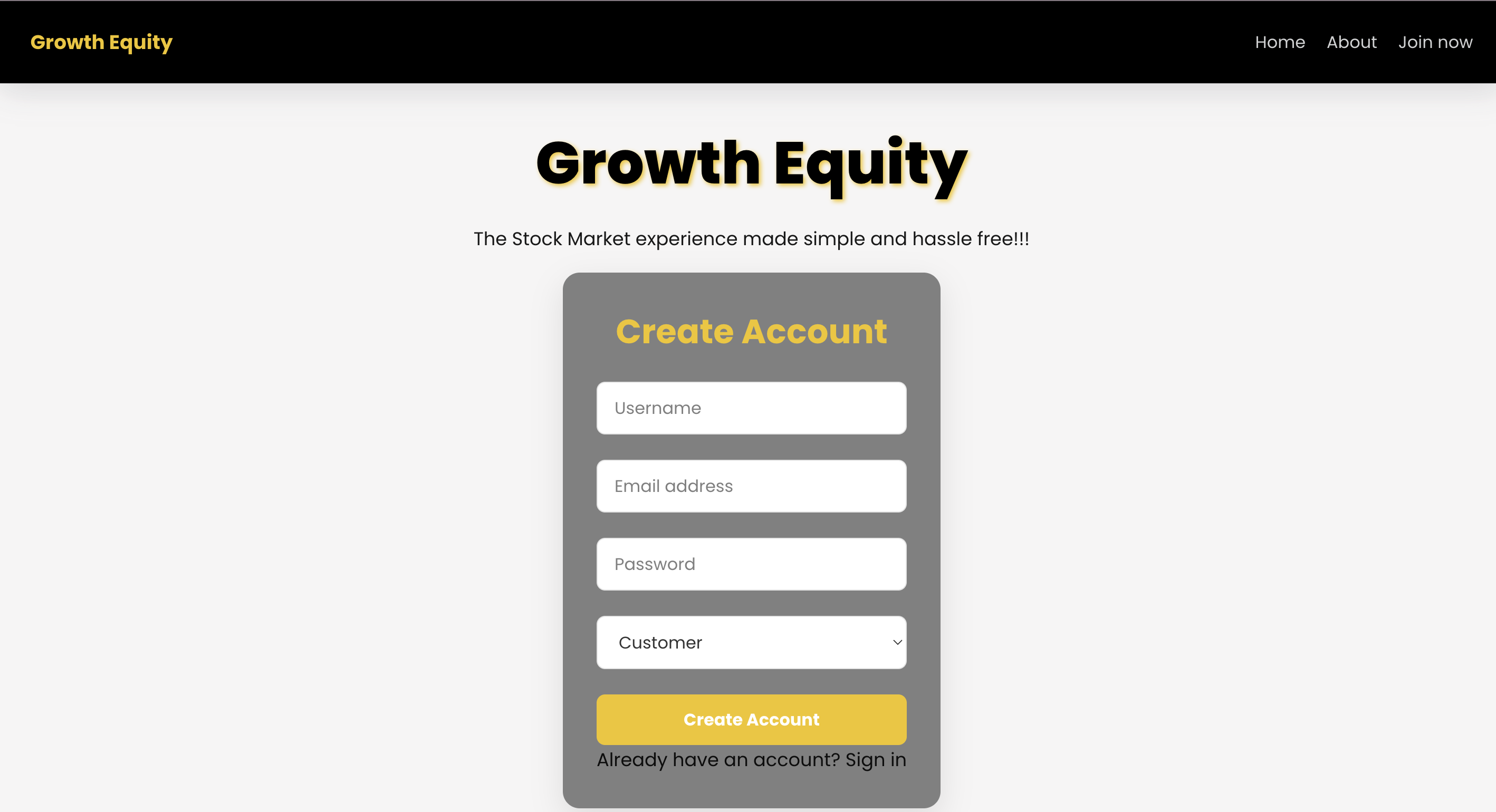
• Sessions are not used; all authentication is stateless.

**9. User Interface**

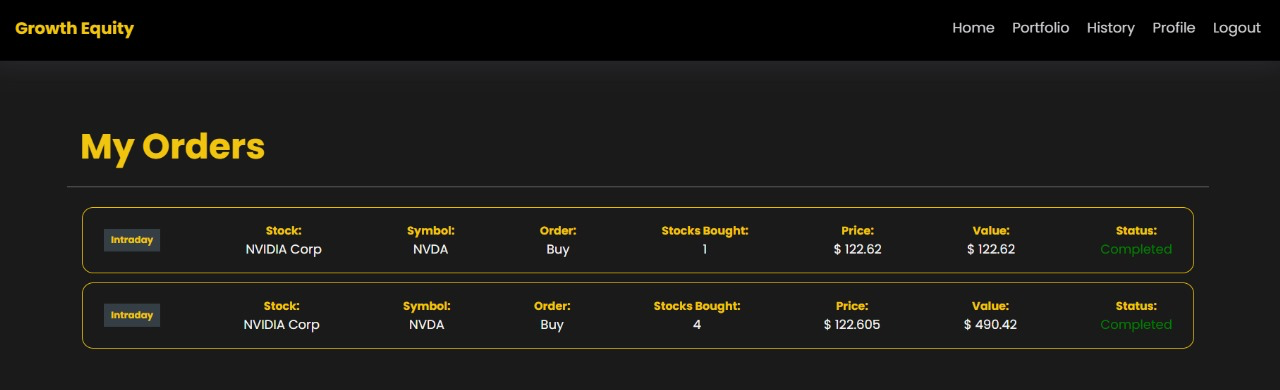
• The UI is designed to be user-friendly and intuitive. Below are some screenshots showcasing different features:

Login Page:

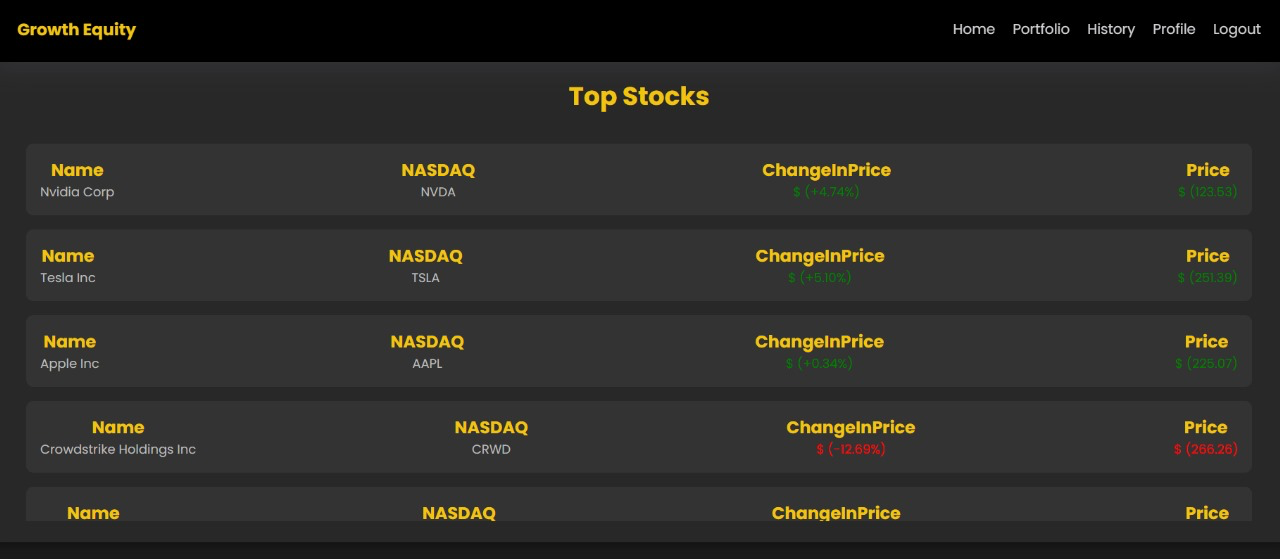


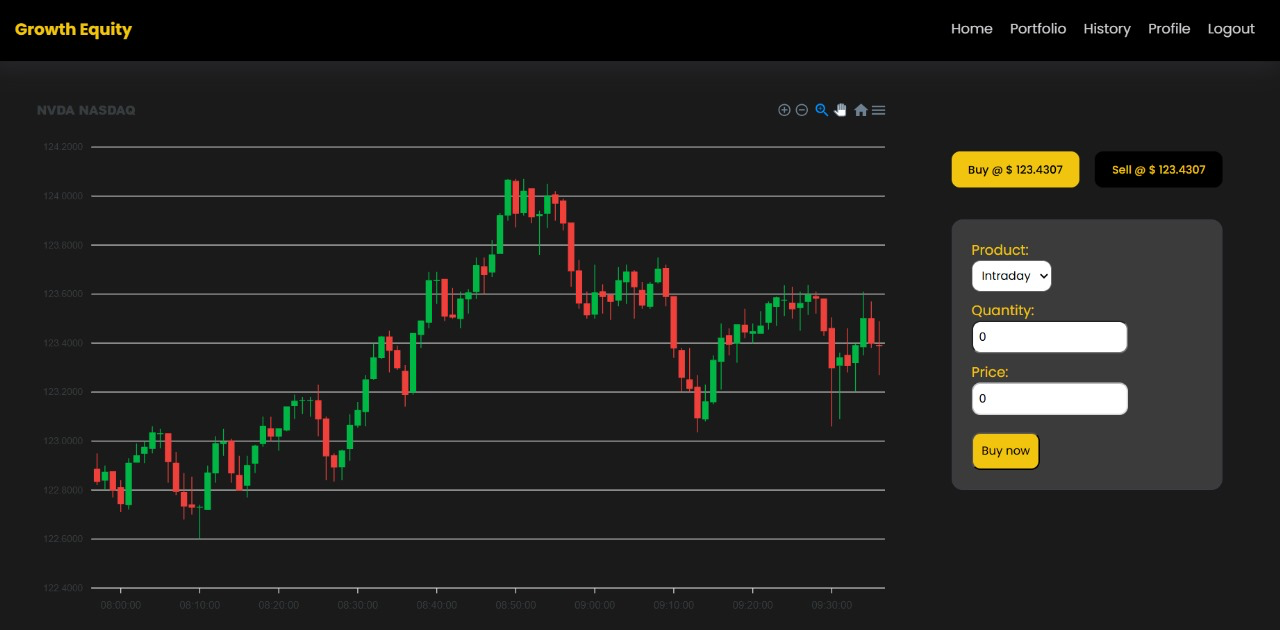


Dashboard:



Trading Page:





**10. Testing**

• The testing strategy includes both unit and integration tests.

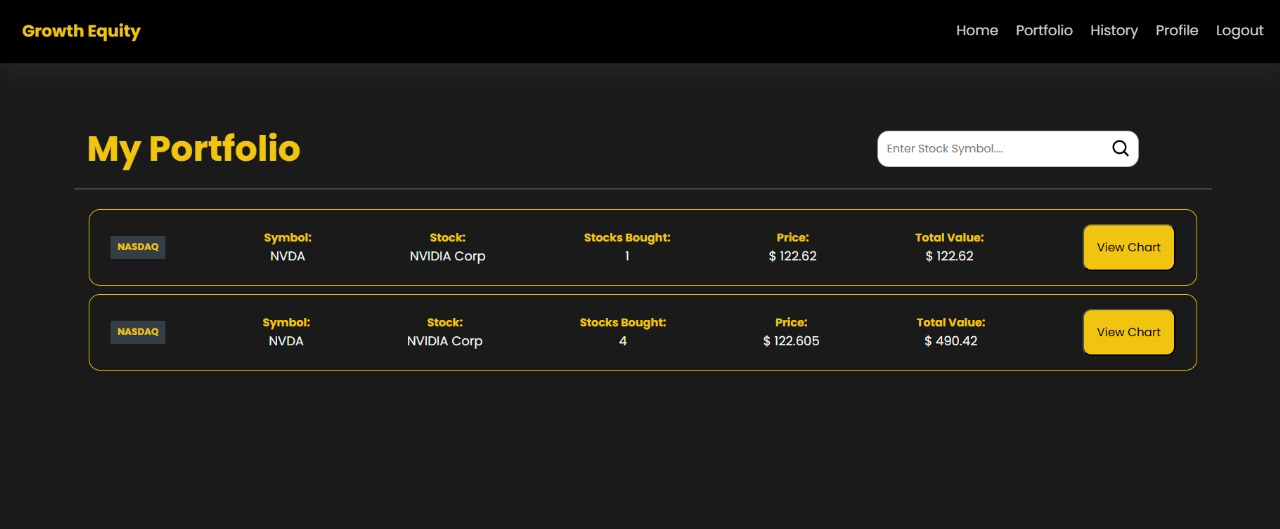
• To ensure the reliability of a stock trading web app, implement a multi-layered testing strategy that includes unit testing for individual components, integration testing for module interactions, end-to-end testing for user workflows, performance testing for responsiveness under load, and security testing to identify vulnerabilities.

This approach ensures the app performs well, is secure, and offers a seamless user experience across different devices and browsers.

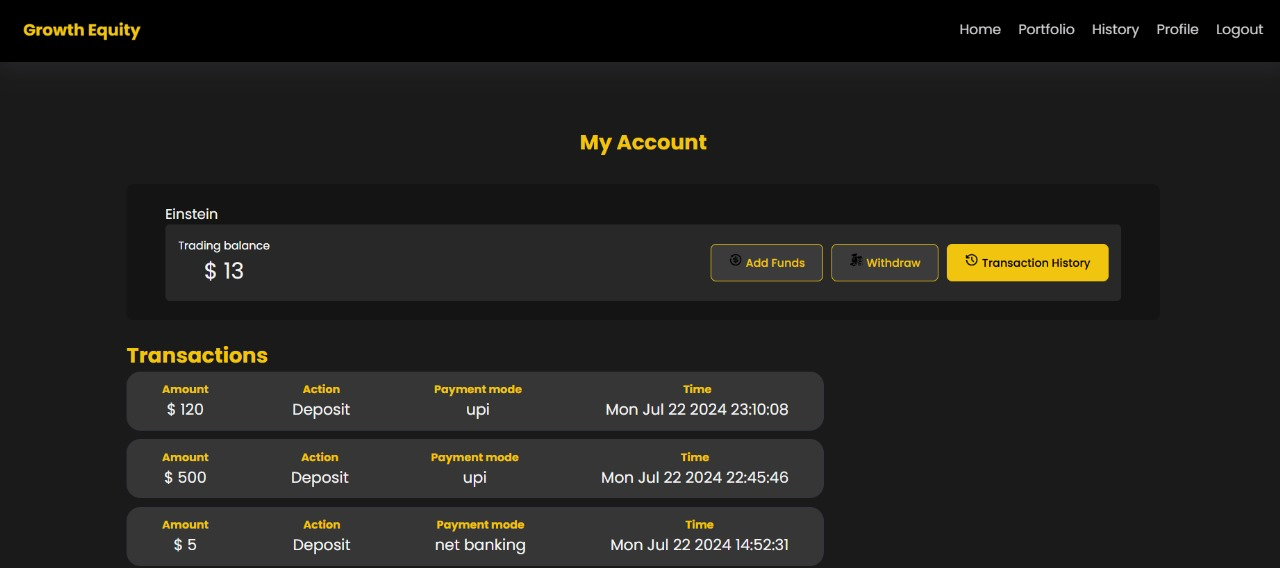
**11. Screenshots or Demo**

**• Additional screenshots:**

Portfolio Page:

****

Transaction History:

****

**12. Known Issues**

• There is a known issue with the real-time stock price updates occasionally lagging. We are working on optimizing the WebSocket connections to fix this.

• The application may experience slow performance under heavy load. Further optimization is planned.

**13. Future Enhancements**

• Integration with additional stock exchanges.

• Mobile app development for iOS and Android.

• Enhanced data analytics and reporting features.

• Social trading features allowing users to follow and copy trades from successful traders.