Nagamedha Sakhamuri *Panther#: 002828574* *nsakhamuri1@student.gsu.edu*

Project Phase 4 - Mid-project Report
Stock Order Management System (SOMS)

Content:

- Progress compared to the milestone set
- Project's current update
- Proof of the project update (shared images/screenshots of project)
- Future Work.

Summary:

Our application, the Stock Order Management System (SOMS), provides a comprehensive platform for buying and selling stocks. Developed using Angular for the frontend, Java Spring for the backend, and MariaDB as the database, SOMS handles essential functions such as user registration, login/logout, order placement, and order cancellation. Inspired by the Indian Stock Exchange (NSE), our application emphasizes database-driven operations, utilizing MariaDB stored procedures to process orders based on availability and order timestamps, following a First Come, First Serve (FCFS) model.

Progress compared to the milestone set:

At this project milestone, we are pleased to report that our progress is fully on track with the established timeline. The team has consistently followed the planned schedule, successfully reaching key milestones in both the development and implementation phases.

Project's current update

1. **Stored Procedures:** Stored procedures are precompiled SQL code that can be stored in the database and executed on demand. In the context of a financial system, stored procedures can encapsulate complex business logic, enhance security, and improve performance.

    - Buying Stocks Stored Procedure: For buying stocks, a stored procedure can be designed to perform the following tasks:
    - Retrieve Stock Information: The procedure starts by fetching the current price of the specified stock from the database.
    - Calculate Total Cost: It calculates the total cost of the transaction based on the stock price and the quantity of stocks to be purchased.
    - Check User's Balance: Verifies if the user has sufficient funds in their account to complete the purchase.
    - Update User's Balance: If the user has enough funds, the procedure deducts the total cost from the user's account balance.
    - Update User's Portfolio: It adds the purchased stocks to the user's portfolio.
    - Record Transaction: The procedure records the transaction in a transaction table, including details such as user ID, stock symbol, quantity, transaction type (buy), and transaction date.

This stored procedure encapsulates the entire buying process, ensuring data consistency and integrity.

Team Members:
Nagamedha Sakhamuri – 002828574 - nsakhamuri1@student.gsu.edu
Ttanvi Tummapudi – 002843956 - ttummapudi1@student.gsu.edu

2. **Spring Boot Logic for Buying and Selling Stocks:** In a Spring Boot application, the logic for buying and selling stocks is typically encapsulated within services or controllers. These components interact with the database through repositories, perform business logic, and handle communication with the front end.

   - Buying Stocks in Spring Boot: In Spring Boot, a service method can be implemented to facilitate the buying of stocks. It generally involves the following steps:
   - Validate Inputs: Validate and sanitize user inputs to ensure data integrity and security.
   - Retrieve Stock Information: Utilize a StockService to fetch information about the selected stock based on the stock symbol.
   - Invoke Stored Procedure: Call a method in the UserService responsible for invoking the stored procedure for buying stocks, passing the necessary parameters such as user ID, stock information, and quantity.
   - Handle Result: Interpret the result returned by the stored procedure (e.g., success message or error) and communicate the outcome to the user.

This Spring Boot logic orchestrates the interaction between the user interface, the application's business logic, and the database, ensuring a seamless process for buying stocks.

3. User Interface Implementation: Sign-In, User Creation, and Validation Overview

In this project, we have successfully implemented a user interface that incorporates essential features such as user sign-in, user creation, and robust validation steps. This interface serves as a crucial component for user interaction and data management within our system.

- Sign-In Page: The sign-in page provides a seamless and secure way for users to access their accounts. The implementation ensures that user credentials are verified before granting access to the system.

Here is a snippet of the sign-in code:

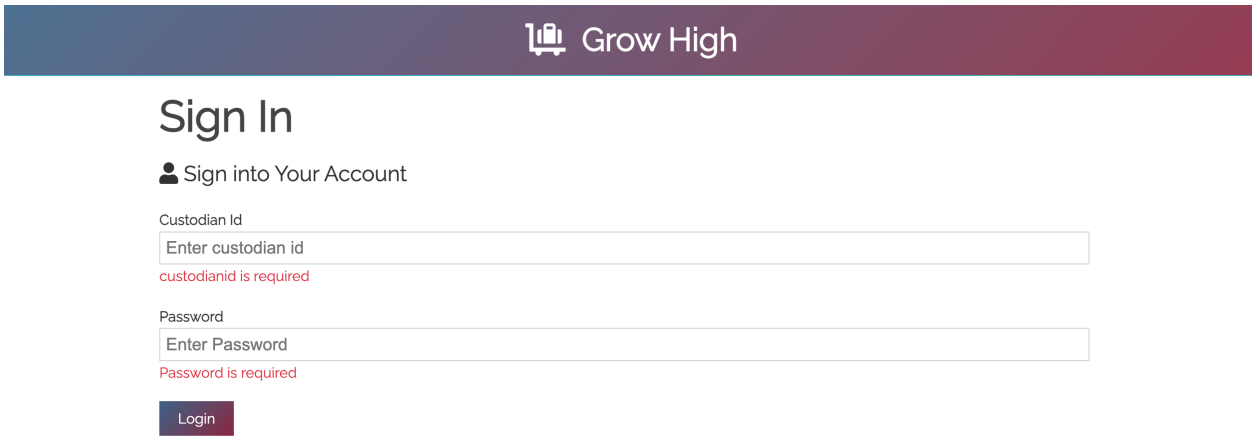
html

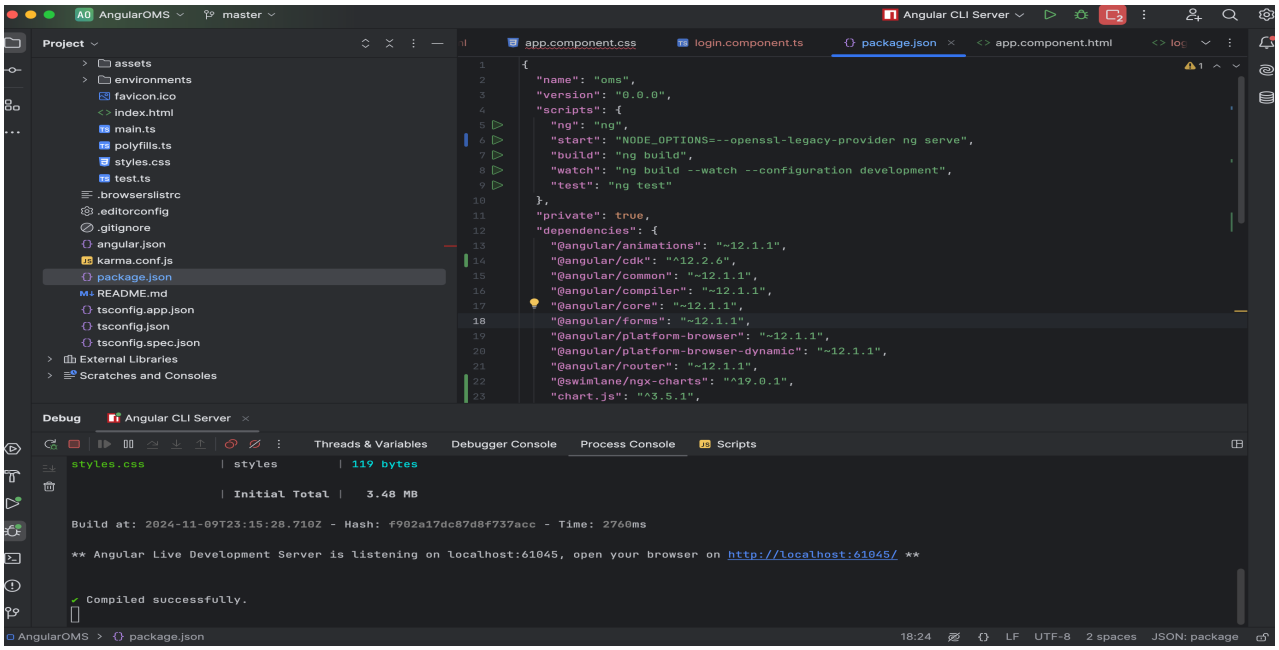
```
Copy code
<!-- Sign-In Form -->
<form action="/signin" method="post">
   <label for="username">Username:</label>
   <input type="text" id="username" name="username" required>

   <label for="password">Password:</label>
   <input type="password" id="password" name="password" required>

   <button type="submit">Sign In</button>
</form>
```

Screenshot: Sign-In Page





- User Creation Page: The user creation page allows new users to register by providing necessary information. The implementation includes form validation to ensure accurate and complete user data.

Here is a code snippet for the user creation form:

html

Copy code

```
<!-- User Creation Form -->
<form action="/create_user" method="post">
    <label for="new_username">New Username:</label>
```

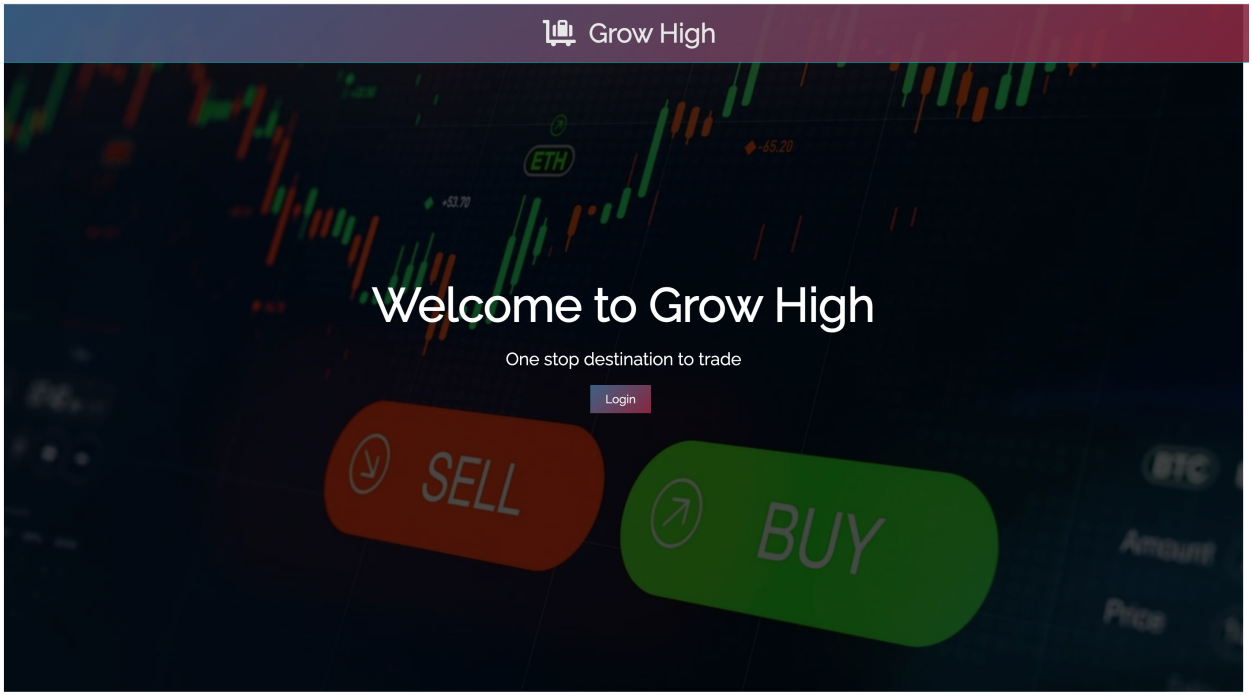
```
    <input type="text" id="new_username" name="new_username" required>

    <label for="new_password">New Password:</label>
    <input type="password" id="new_password" name="new_password" required>

    <!-- Additional fields for user information -->

    <button type="submit">Create Account</button>
</form>
```

Screenshot: User Creation Page



- Validation Steps: Robust validation steps have been incorporated into both the sign-in and user creation processes. These steps ensure data integrity and prevent unauthorized access.

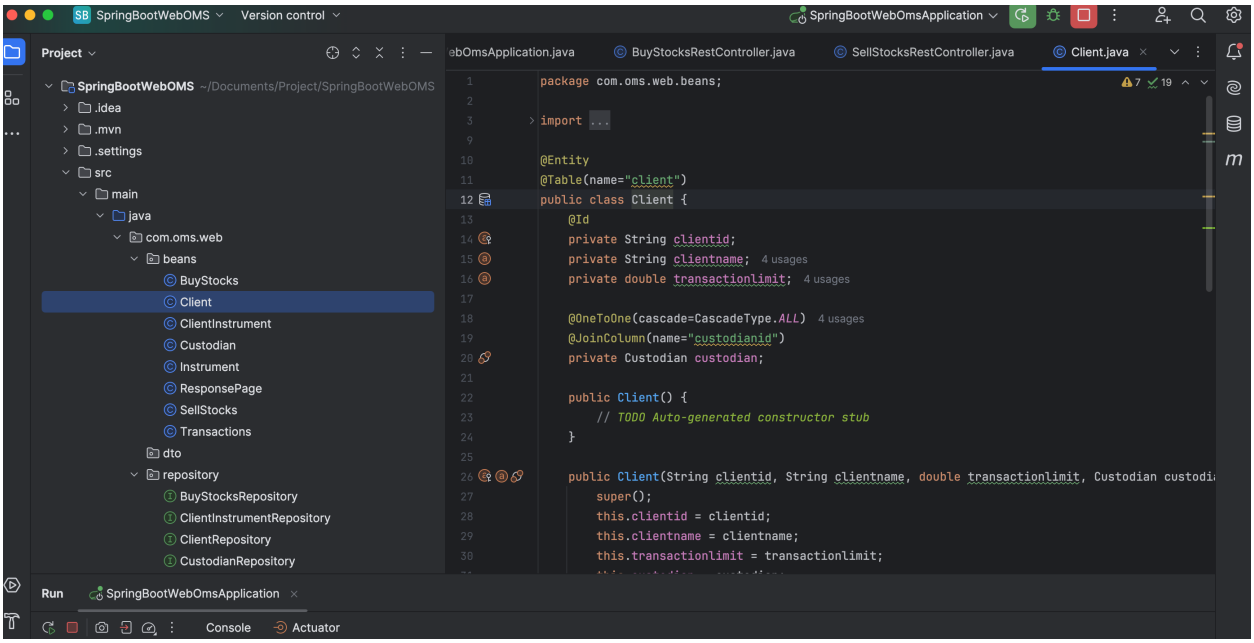
Below is an example of server-side validation using Node.js and Express:

Javascript

```
Copy code
// Example server-side validation middleware app.post('/signin',
(req, res) => {
    const { username, password } = req.body;

    // Validate username and password
    if (isValidCredentials(username, password)) {
        // Grant access
```

```
        res.status(200).send('Sign-in successful');
    } else {
        // Invalid credentials
        res.status(401).send('Invalid username or password');}
});
```



Future Work

- **Remaining Backend Development:** Completion of the stock recommendation engine and additional transaction functionalities.
- **UI Enhancements:** Designing and implementing the UI for transaction history, stock portfolio views, and notifications.
- **Testing & Debugging:** Comprehensive testing of the integrated system, followed by debugging and optimization for performance.

Conclusion

In conclusion, the user interface delivers a straightforward and accessible experience for sign-in, account creation, and validation. The included code snippets and screenshots highlight its clean, intuitive design, ensuring secure user interactions with the system. Moving forward, we will concentrate on further optimizations, additional feature enhancements, and continuous improvements to the interface based on user feedback.

Team Members:
Nagamedha Sakhamuri – 002828574 - nsakhamuri1@student.gsu.edu
Ttanvi Tummapudi – 002843956 - ttummapudi1@student.gsu.edu