Project Report: Angular Task Manager

24CSE441 Full Stack Development with Java

Group - 6

Sidharth P - AM.EN.U4EAC21063
Nikhil Rajiv- AM.EN.U4EAC21055
Sreedutt Ram J- AM.EN.U4EAC21067
Harin BS - AM.EN.U4EAC21030

Project Overview

This document outlines two projects that demonstrate expertise in full-stack web development using various technologies. The projects are as follows:

1. **Task Manager Application**
2. **Login Application**

Task Manager Application

Prerequisites Before starting, make sure you have the following installed:

1. Node.js and npm (latest stable version)
2. Angular CLI (version 18.2 or higher)
3. IDE: Visual Studio Code

Overview

This project is a full-stack application designed to manage a todo list. It consists of a frontend built with Angular and a backend built with Node.js and Express. The application allows users to register, log in, and manage their Todo items. The backend handles authentication using JWT and stores user and todo data in JSON files.

Frontend (Angular)

App Module

- ***app.component.ts:*** The root component of the Angular application. It serves as the entry point for the app and contains the main template.
- ***app.config.ts:*** Configuration for the Angular application, including providers and interceptors. It sets up the necessary modules and services for the app.
- ***app.routes.ts***: Defines the routes for the application, including guards for authentication. It ensures that users are directed to the correct components based on their authentication status.

Core Module

- **Constants**
  - ***constants.ts:*** Contains application-wide constants, including API endpoints. This file centralizes the configuration values used throughout the app.
- **Guards**

    - ***auth.guard.ts:*** Guard to protect routes that require authentication. It checks if the user is authenticated before allowing access to certain routes.
    - ***guest.guard.ts:*** Guard to protect routes that should only be accessible to unauthenticated users. It prevents logged-in users from accessing the login and registration pages.
- **Interceptors**
    - ***http.interceptor.ts:*** Interceptor to add the JWT token to HTTP requests. It ensures that authenticated requests include the necessary token for backend validation.
- **Models**
    - ***auth.model.ts:*** Interfaces for authentication-related data. It defines the structure of the data used in authentication operations.
    - ***todo.model.ts:*** Interfaces for todo-related data. It defines the structure of the data used in todo operations.
- **Services**
    - ***auth.service.ts:*** Service for handling authentication-related operations. It includes methods for login, registration, logout, and retrieving the username.
    - ***todo.service.ts:*** Service for handling todo-related operations. It includes methods for fetching, adding, updating, and deleting todos.

Pages

- **Login**
    - ***login.component.ts:*** Component for the login page. It handles user input and communicates with the AuthService to authenticate users.
- **Register**
    - ***register.component.ts:*** Component for the registration page. It handles user input and communicates with the AuthService to register new users.
- **Todo**
    - ***todo.component.ts:*** Component for the todo list page. It displays the list of todos and provides functionality for adding, updating, and deleting todos.

Shared Components

- **Todo Card**
    - ***todo-card.component.ts:*** Component for displaying individual todo items. It is used within the TodoComponent to render each todo.
- **Layouts**
    - ***default.component.ts:*** Layout component for unauthenticated routes. It wraps the login and registration pages.
    - ***master.component.ts:*** Layout component for authenticated routes. It wraps the main application content, including the todo list.
- **UI Components**
    - ***slide-panel.component.ts:*** Component for displaying a slide panel. It is used in the TodoComponent to show the form for adding and updating todos.

Backend (Node.js/Express)

Controllers

- ***authController.js:*** Handles user registration and login, including JWT token generation. It validates user credentials, generates tokens, and responds with appropriate messages.

- ***todoController.js***: Handles CRUD operations for todos. It manages the creation, retrieval, updating, and deletion of todo items, ensuring that only authorized users can modify their todos.

Middleware

- ***authMiddleware.js:*** Middleware to authenticate requests using JWT tokens. It verifies the token provided in the request headers and attaches the user information to the request object.

Models

- ***User.js:*** Model for user data, including methods for finding, saving, and comparing users. It handles user-related operations such as hashing passwords and validating credentials.
- ***Todo.js:*** Model for todo data, including methods for finding, saving, updating, and deleting todos. It manages the storage and retrieval of todo items from the JSON file.

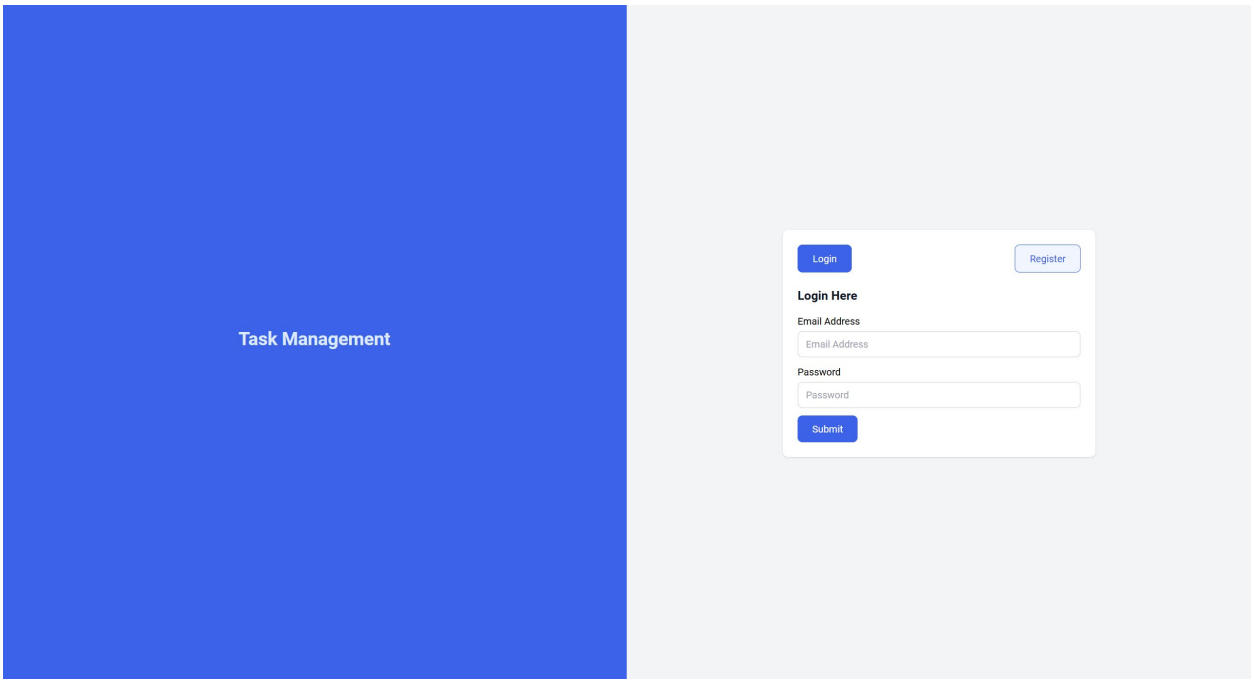
Routes

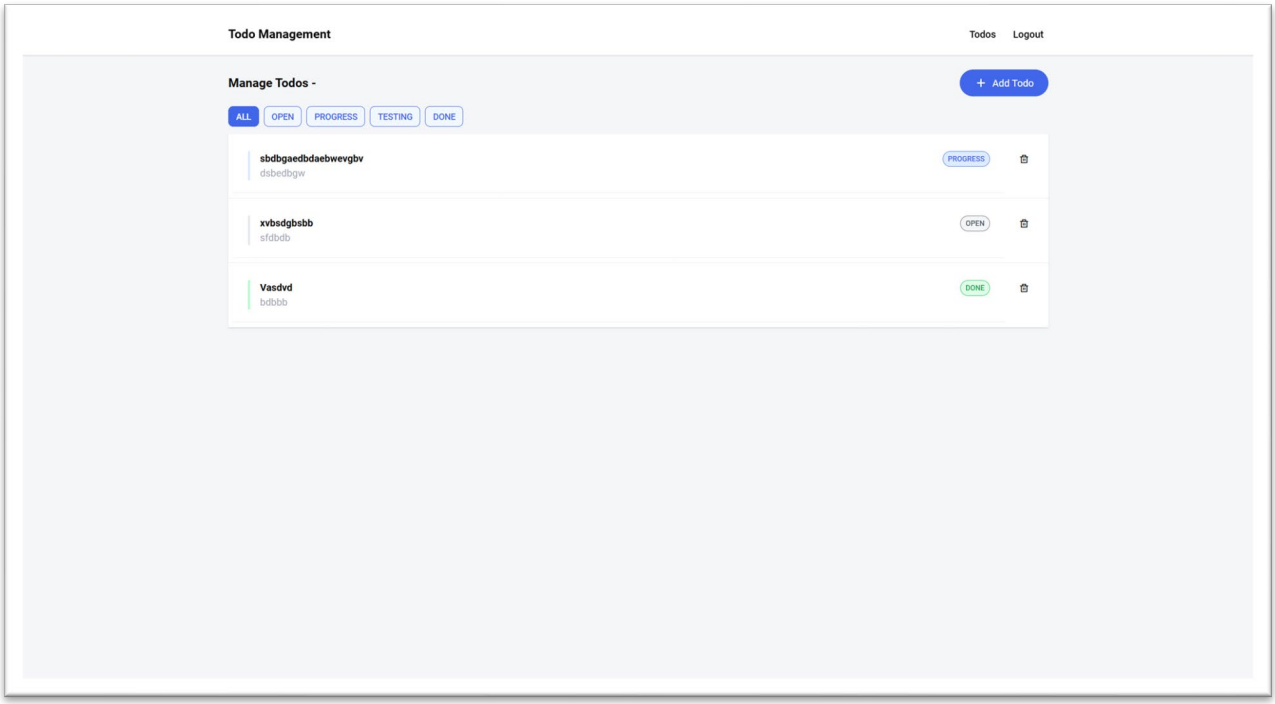
- ***authRoutes.js:*** Routes for authentication-related endpoints. It defines the endpoints for user registration and login.
- ***todoRoutes.js:*** Routes for todo-related endpoints, protected by the authentication middleware. It defines the endpoints for CRUD operations on todos.

Server

- ***server.js:*** Entry point for the backend application. It sets up middleware and routes, and starts the server. It also initializes the default user if not already present.

Webpage:





Stored Data:

```
[
  {
    "email": "test@example.com",
    "password": "$2a$10$p3zF2rBitw.
    bJpArqkNJWO6RdBggMbpmSv8XDFg751MxZodl7JNu6"
  },
  {
    "name": "temp",
    "email": "temp@gmail.com",
    "password": "$2a$10$fU2czBjO2xJ4/mPp9t8AIelaFMFLYfIBJAV.DBh1BD73lQ7f8TSh.
    "
  },
  {
    "name": "abc",
    "email": "abc@gmail.com",
    "password": "$2a$10$7bNOV8BXzDNRAsnSlx/NpeXeSEPkQsWeYrp/
    602vrE0yGdTt71NJO"
  }
]
```

User Data

```
[
  {
    "id": "ff4a30d5-2c6d-4111-bacc-86f89a047606",
    "title": "test",
    "description": "vbhvu",
    "status": "TESTING",
    "user": "temp"
  },
  {
    "id": "b3d677ab-9b80-4476-8d7f-85591785922d",
    "title": "sbdbgaedbdaebwevgbv",
    "description": "dsbedbgw",
    "status": "PROGRESS"
  },
  {
    "id": "25319482-f4da-4bbc-a5ff-1e740c97ba45",
    "title": "xvbsdgbsbb",
    "description": "sfdbdb",
    "status": "OPEN"
  },
  {
    "id": "6124e0c3-c32a-4bc8-a9e7-71a4084db53a",
    "title": "Vasdvd",
    "description": "bdbbb",
    "status": "DONE"
  },
  {
    "id": "4114053c-8ae0-44ec-bca7-afeb54da9b0f",
    "title": "vdge",
    "description": "rbrntn",
    "status": "OPEN",
    "user": "temp"
  }
]
```

Todo Data

Working of the Application

User Registration and Login

- Users can register by providing a username, email, and password. The authController handles the registration, ensuring that the email and username are unique and saving the hashed password.
- Users can log in by providing their email and password. The authController validates the credentials, generates a JWT token, and responds with the token.

Authentication Middleware

- The authMiddleware verifies the JWT token included in the request headers. If the token is valid, it attaches the user information to the request object and allows the request to proceed.

Todo Management

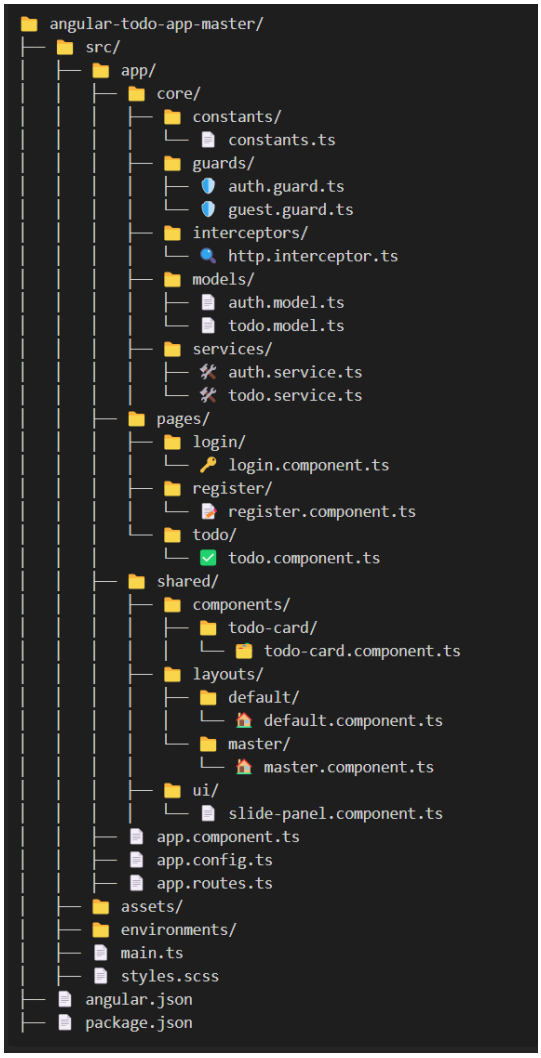
- Authenticated users can perform CRUD operations on their todos. The todoController handles these operations, ensuring that only the owner of the todo can modify or delete it.
- The TodoService in the frontend communicates with the backend to fetch, add, update, and delete todos. The TodoComponent displays the todos and provides the UI for managing them.

Code (GitHub link due to lengthy code):

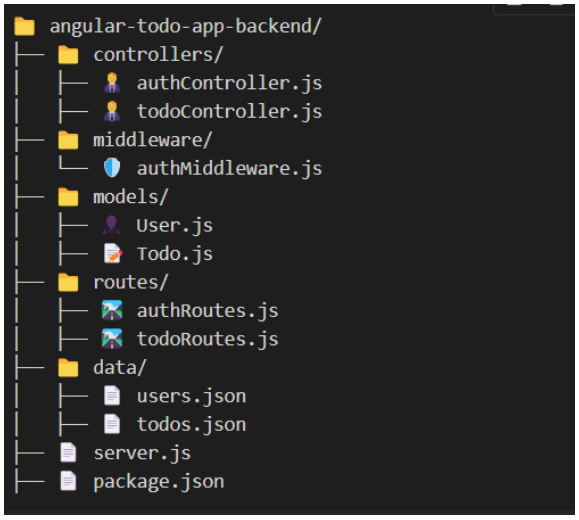
https://github.com/Sid-WC121/Task-Manager

File Structures:

Frontend (Angular)



Backend (Node.js/Express)



Conclusion

This project is a comprehensive full-stack application that demonstrates the integration of an Angular frontend with a Node.js/Express backend. It includes user authentication, protected routes, and CRUD operations for managing todos. The code is well-structured, with clear separation of concerns between different modules and components. The use of JWT for authentication ensures secure communication between the frontend and backend.

Login Application

Login Frontend Application

Overview

The Login Frontend Application is an Angular-based project designed to provide a user interface for user authentication. It interacts with a backend API to validate user credentials and display appropriate messages based on the authentication result. The frontend application is built using Angular, a powerful framework for building dynamic web applications.

Key Features

- **User Login Form**: A user-friendly login form that captures username and password.
- **Form Validation**: Ensures that the user provides the necessary credentials before submitting the form.
- **Error Handling**: Displays error messages for failed login attempts.
- **Responsive Design**: Ensures that the login form is accessible and usable on various devices.

Technologies Used

- **Angular**: A platform for building mobile and desktop web applications.
- **TypeScript**: A statically typed superset of JavaScript used for writing Angular applications.
- **HTML & CSS**: Markup and styling for the login form.
- **Angular CLI**: A command-line interface tool for initializing, developing, and maintaining Angular applications.

Project Structure

- **Components**: The application consists of a LoginComponent that handles the login form and its logic.
- **Services**: An AuthService is used to communicate with the backend API for authentication.
- **Modules**: The application is modularized using Angular modules for better organization and maintainability.

Dependencies

The project relies on several key dependencies, including:

- @angular/core: The core Angular framework.
- @angular/forms: Provides support for template-driven and reactive forms.
- @angular/common/http: Facilitates HTTP communication with the backend API.
- @angular/router: Enables navigation and routing within the application.

Conclusion

The Login Frontend Application provides a clean and responsive user interface for user authentication. By leveraging Angular's powerful features, it ensures a seamless and interactive experience for users. The application communicates with a backend API to validate user credentials and provides appropriate feedback based on the authentication result.

Login Backend Application

Overview

The Login Backend Application is a Spring Boot-based project designed to handle user authentication and authorization. It provides a RESTful API for user login, leveraging Spring Boot's powerful features for web development, data persistence, and security. The backend communicates with a MySQL database to store and retrieve user credentials securely.

Key Features

- **User Authentication**: Validates user credentials and provides appropriate responses for successful or failed login attempts.
- **RESTful API**: Exposes endpoints for user login and other related operations.
- **Data Persistence**: Utilizes Spring Data JPA for seamless integration with the MySQL database.
- **Spring Boot Integration**: Leverages Spring Boot's starter dependencies for rapid development and configuration.

Technologies Used

- **Spring Boot**: A framework for building production-ready applications quickly.
- **Spring Data JPA**: Simplifies database interactions and provides a repository-based approach.
- **Spring Web**: Facilitates the creation of RESTful web services.
- **MySQL**: A relational database management system used for storing user data.
- **Java 11**: The programming language used for developing the application.

Project Structure

- **Controller Layer**: Handles incoming HTTP requests and delegates them to the service layer.
- **Service Layer**: Contains business logic for user authentication.
- **Repository Layer**: Interacts with the database using Spring Data JPA.
- **Model Layer**: Defines the User entity and other related data models.

Dependencies

The project relies on several key dependencies, including:

- spring-boot-starter-web: Provides the necessary components for building web applications.
- spring-boot-starter-data-jpa: Simplifies database interactions using JPA.
- mysql-connector-j: A JDBC driver for MySQL.
- spring-boot-starter-test: Provides testing support for the application.

Response:

- **Success: Authenticated**
- **Failure: Authentication Failed**

Code (GitHub link due to lengthy code):

https://github.com/Sid-WC121/Task-Manager

Conclusion

The Login Backend Application provides a robust foundation for user authentication in a web application. By leveraging Spring Boot and related technologies, it ensures a scalable and maintainable solution for handling user login operations.

Database:

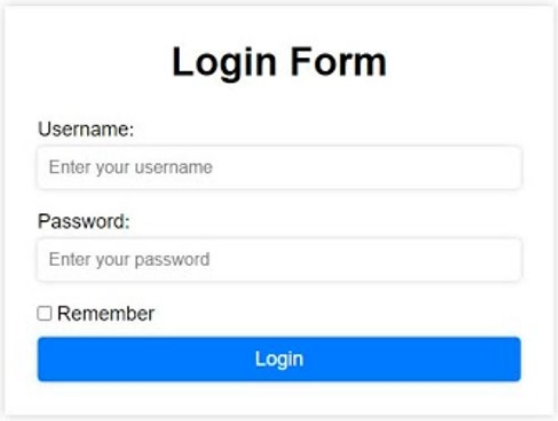
login-backend > src > main > resources > schema.sql

```sql
CREATE DATABASE loginapp;
USE loginapp;

CREATE TABLE users (
    username VARCHAR(50) PRIMARY KEY,
    password VARCHAR(255) NOT NULL
);

INSERT INTO users (username, password) VALUES
('admin', 'admin123'),
('user', 'user123');
```

Result Grid | Filter Rows: | Edi

username	password
admin	admin123
user	user123
NULL	NULL

Login UI




```java
package com.yourcompany.loginapp.model;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(
    name = "users"
)
public class User {
    @Id
    private String username;
    private String password;

    public User() {
    }

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return this.password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Authenticated Access

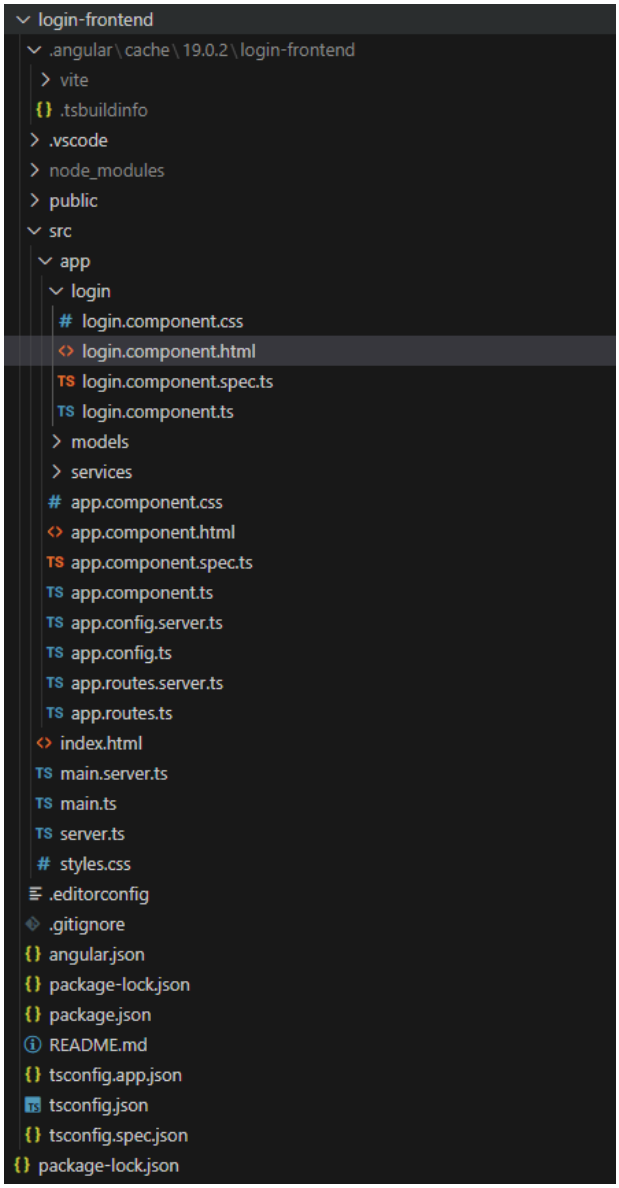
```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.loginapp:login-backend >------------------------
[INFO] Building login-backend 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ login-backend ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ login-backend ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Nikhil\Downloads\login\login-backend\target\cl
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ login-bac
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Nikhil\Downloads\login\login-backend\
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ login-backend
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Nikhil\Downloads\login\login-backend\target\te
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ login-backend ---
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ login-backend ---
[INFO] Building jar: C:\Users\Nikhil\Downloads\login\login-backend\target\login-backend-0
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] Authentication result: Authenticated
```

Authentication Failed

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.loginapp:login-backend >------------------------
[INFO] Building login-backend 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ login-backend ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ login-backend ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Nikhil\Downloads\login\login-backend\target\cla
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ login-back
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Nikhil\Downloads\login\login-backend\s
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ login-backend -
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Nikhil\Downloads\login\login-backend\target\tes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ login-backend ---
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ login-backend ---
[INFO] Building jar: C:\Users\Nikhil\Downloads\login\login-backend\target\login-backend-0.
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.5:run (default-cli) @ login-backend ---
[INFO] Attaching agents: []
[INFO]
[INFO] Authentication result: Authentication Failed
```

File Structure:

Frontend (Angular)



Backend (Spring Boot)