User Service Application

A. Problem Statement:

Designed and developed a user service application with roles and authorisation for an e-commerce web application. The application should allow users to register, authenticate, and update their profiles. Users should have different roles assigned to them, such as "**customer**" and "**admin**", each with specific permissions. The application should implement authorisation checks to ensure that users can only access resources and perform actions allowed by their assigned roles.

B. Features of the User Application:

The application should allow to:

- **Implement User Registration:** Allow users to create an account with the required information, such as username, email, and password.
- **User Authentication and Login:** Develop a secure authentication mechanism to verify user credentials and allow them to log in to the application.
- **User Roles and Permissions:** Design a role-based access control system to assign users different roles (e.g., admin, customer) and manage their permissions.
- **Authorisation and Access Control:** Ensure users can only access features and resources based on their assigned roles and permissions.
- **Profile Management:** Enable users to view and update their profile information, such as name, contact details, and address.
- **Error Handling and Exception Management:** Develop robust error handling mechanisms and implement exception management to handle unexpected scenarios gracefully.

C. Steps to Create the Project:

Step 1: Setup a Spring Boot Project:

1. Create a new Spring Boot project using your preferred IDE or start.spring.io.

2. Include the necessary dependencies, such as Spring Web, Spring Data JPA, and Spring Security.
    a. Spring web
    b. Spring version - 2.7.5
    c. Spring Data JPA
    d. MySQL Driver
    e. Java version - 17
    f. Spring Security
    g. Spring Boot Devtools
    h. Eureka Server
    i. Spring Validation

Step 2: Design User and Role Entities:

1. Create a User entity class with attributes like id, name, username, email, password, etc.
2. Implement a Role entity class with ID and name attributes.
3. Establish a many-to-many relationship between User and Role entities using appropriate annotations.

Step 3: Implement User and Role Repositories:

1. Create a UserRepository interface that extends JpaRepository<User, Long>.
2. Create a RoleRepository interface that extends JpaRepository<Role, Long>.

Step 4: Implement User Registration:

1. Create a UserController class to handle user-related endpoints.
2. Implement a registration endpoint that receives user registration data (username, email, password).
3. Validate the input data by checking for unique usernames or emails, password complexity, etc.
4. Save the user to the database using the UserRepository.

Step 5: Implement User Login and Authentication:

1. Implement a login endpoint that accepts user credentials (username/email and password).
2. Validate the credentials and authenticate the user.
3. Generate and return an authentication token (e.g., JWT) to the client for subsequent authenticated requests.

Step 6: Implement Role-based Authorization:

1. Define roles (e.g., admin, user) and their corresponding permissions.
2. Configure role-based authorisation using Spring Security or custom authorisation mechanisms.
3. Annotate API endpoints with appropriate security annotations like @PreAuthorize to enforce role-based access control.

Step 7: Secure API Endpoints:

1. Use Spring Security configurations to secure API endpoints based on user roles.
2. Configure security rules to allow or deny access to specific endpoints based on user roles and permissions.
3. Handle unauthorised access by returning appropriate error responses.

Step 8: Error Handling and Exception Management:

1. Implement global exception handling to handle and return meaningful error responses for various exceptions during user operations.
2. Customise error responses to provide informative error messages and appropriate HTTP status codes.

D. End Points to be created:

For the backend application to work with the Frontend, you must expose these APIs.

User Endpoints:

- POST /api/auth/signin: Authenticate user credentials and generate an access token.
- POST /api/auth/signup (Body: User Object): Create a new user.
- GET /api/auth/test: allows to check the health of the application

Additional endpoints:

- GET /users/all: Retrieve a list of all users.
- GET /users/{userId}: Retrieve a specific user by ID.
- GET /users/search/{keywords}: To search a user
- PUT /users/{userId} (Body: UserDto Object): Update the user with that particular user-id
- DELETE /users/{userId}: Delete the user with that particular user-id

Note: Students have to identify which endpoint will be allocated to admin and user

E. Testing on Postman:

After successfully creating the application, you have to test your application.

1. Your application should test the following:
    - The user's registration should be successfully stored in the database.
2. Your application should throw an error if:
    - The user is trying to register a new user with a username already in the database.
3. Test all the exposed APIs of Backend on Postman.