Git - https://github.com/Balajitrn/lms

User Microservice Summary

1. Overview:
The user microservice is designed to manage user-related operations within the application. It includes features for user registration, login, and profile management, along with incorporating Spring Security for authentication and authorization.

2. Core Functionalities:
- **User Registration:** Allows new users to create accounts.
- **User Login:** Authenticates users and provides JWT tokens for subsequent requests.
- **Profile Management:** Allows authenticated users to view their profiles.

3. Components Involved:

- **Controller**: Exposes RESTful endpoints for user interactions, handling requests for registration, login, and profile viewing.

- **Service Layer**: Contains the business logic for user operations. It interacts with the repository to perform CRUD operations on user data.

- **Repository (Data Access Layer):** Interfaces with the database, allowing the service layer to query and manipulate user data.

- **JwtTokenProvider**: A component responsible for generating, validating, and parsing JWT tokens. It assists in managing user authentication and authorization.

- **SecurityConfig**: The main configuration class for Spring Security. It defines the security rules, such as URL patterns that are open or secure, password encoding, and the integration of JWT authentication.

- **JwtConfigurer and JwtTokenFilter**: Classes that configure how JWT tokens are handled in HTTP requests. JwtTokenFilter inspects incoming requests for valid JWT tokens.

- **CustomUserDetailsService**: An implementation of Spring Security's UserDetailsService, responsible for loading user details from the database. It plays a crucial role in authenticating users based on the tokens provided in requests.

- **BCryptPasswordEncoder**: Used to hash user passwords, enhancing the security of stored credentials.

- **Database**: The underlying data store where user information is persisted.

4. **Security Features**:
  - **Stateless Authentication**: Using JWT tokens for stateless authentication, ensuring that each request contains all the information needed to be processed.
  - **Password Encryption**: Passwords are securely hashed using BCrypt before storage.
  - **Role-Based Authorization (Optional):** Can be extended to support different levels of access based on user roles.

5. Technologies and Frameworks:
  - **Spring Boot**: Provides the foundational framework.
  - **Spring Security**: Manages security aspects like authentication and authorization.
  - **JSON Web Tokens (JWT)**: Used for token-based authentication.
  - **Database (H2)**: For persisting user data.

This user microservice represents a modular and scalable approach to user management within a modern microservices-based architecture. By following best practices in security and design, it lays a solid foundation for building a comprehensive and robust application.

Below is a detailed technical documentation explaining each component and their roles, methods involved, different annotations used, and their benefits.

1. Controller

Role: Exposes ***RESTful*** endpoints for user interactions.

Methods:
- **registerUser()**: Handles user registration.
- **login()**: Authenticates users and provides JWT tokens.
- **getProfile()**: Allows authenticated users to view profiles.

Annotations:
- **@RestController**: Indicates that the class is a REST controller, meaning that it returns data directly to the client.
- **@RequestMapping**: Maps specific request paths to corresponding methods.
- **@PostMapping**: Specifies that a method handles POST requests.
- `@GetMapping`: Specifies that a method handles GET requests.

2. Service Layer

Role: Contains business logic for user operations.

Methods:
- **registerUser()**: Validates and saves new user information.
- **findByUsername()**: Retrieves user details by username.

Annotations:
- **@Service**: Indicates that the class is a service, containing business logic.

3. Repository (Data Access Layer)

Role: Interfaces with the database.

Methods:
- **save()**: Inserts or updates user data.
- **findByUsername()**: Retrieves user by username.

Annotations:
- **@Repository**: Indicates that the class is a repository, encapsulating database interactions.

4. JwtTokenProvider

Role:Manages JWT tokens.

Methods:
- **getAuthentication()**: Constructs authentication object from token.
- **generateToken()**: Generates JWT tokens.
- **validateToken()**: Validates tokens.

5. SecurityConfig

Role: Main security configuration.

Methods:
- **configure()**: Configures security rules.

Annotations:
- **@EnableWebSecurity**: Enables Spring Security's web security support.
- **@Bean**: Indicates that a method produces a bean to be managed by the Spring container.

6. JwtConfigurer and JwtTokenFilter

Role: Handle JWT tokens in HTTP requests.

Methods:
- **configure()**: Configures JWT handling.
- **doFilterInternal()**: Inspects incoming requests for valid JWT tokens.

7. CustomUserDetailsService

Role: Loads user details from the database.

Methods:
- **loadUserByUsername()**: Loads user details by username.

Annotations:
- **@Autowired**: Enables automatic dependency injection.

8. BCryptPasswordEncoder

Role: Hashes user passwords.

Annotations:
- **@Bean**: Produces a bean managed by the Spring container.

Benefits and Usage of Annotations:
- **Encapsulation**: Annotations like **@Service**, **@Repository**, and **@RestController** encapsulate specific roles within the application, promoting clear separation of concerns.
- **Dependency Injection**: Annotations like **@Autowired** facilitate dependency injection, promoting loose coupling and ease of testing.
- **Configuration**: Annotations like **@EnableWebSecurity** and **@Bean** simplify configuration, allowing declarative setup of various components.
- **Request Mapping**: Annotations like **@RequestMapping**, **@PostMapping**, and **@GetMapping** provide intuitive routing and request handling, making it easier to define RESTful endpoints.

This detailed Tech spec documentation provides an understanding of each component's role and functionality, the methods involved, and the usage of different annotations. It serves as a comprehensive guide to the user microservice's design and can be used for reference purposes to learn the principles of modern web application development, security practices, and Spring-based development.