**PayMaytrix --- Authentication & Authorization Implementation**

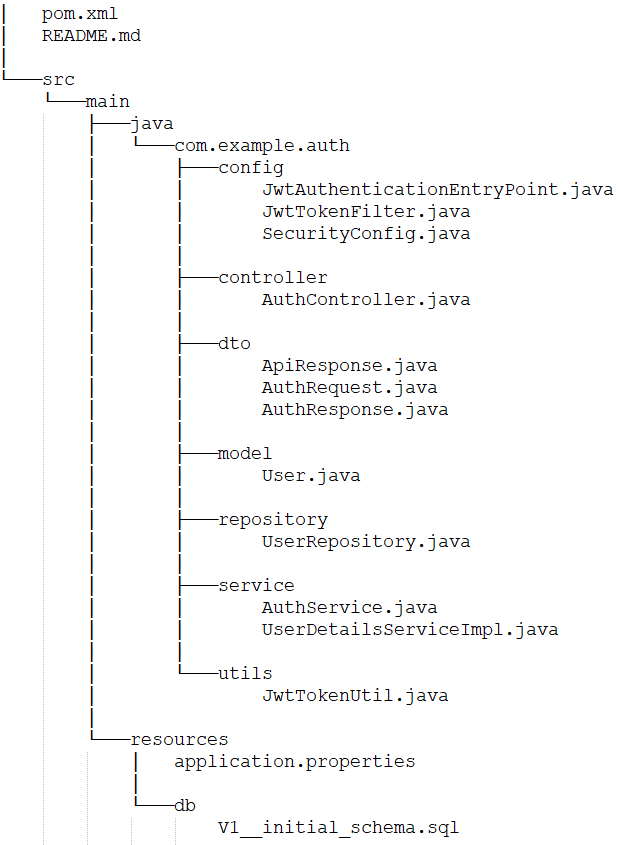
Technology Stack:

* Spring Boot
* Spring Security
* JSON Web Tokens (JWT)
* PostgreSQL (Database)
* Maven (Build tool)



Project Structure:

🡪 com.paymaytrix.authentication-authorization-service





Implementation Details:

1. **User Entity:**
   * Create a **User** entity to represent user information.
   * The entity will have fields like **id**, **username**, **password**, **email**, etc.
   * The **User** entity will be mapped to the database table using JPA.



1. **User Repository (UserRepository):**
   * Create a **UserRepository** interface that extends **JpaRepository<User, Long>**.
   * This repository will handle database operations for the **User** entity.



1. **Authentication Controller (AuthController):**
   * Implement an **AuthController** to handle user authentication and registration.
   * Expose endpoints for user registration and login.



1. **Authentication Service (AuthService):**
   * Create an **AuthService** interface that will define methods for user registration, login, and token generation.
   * Implement the **AuthService** interface in a class (**AuthServiceImpl**).



1. **User Details Service (UserDetailsServiceImpl):**
   * Implement the **UserDetailsService** interface from Spring Security to load user details from the database.
   * This service will be used by Spring Security for authentication.



1. **Security Configuration (SecurityConfig):**
   * Configure Spring Security to enable JWT-based authentication.
   * Set up authentication and authorization rules for different endpoints.
   * Use the **JwtTokenFilter** to validate JWT tokens for secured endpoints.



1. **JWT Token Util (JwtTokenUtil):**
   * Implement a utility class to generate, validate, and parse JWT tokens.



1. **JwtTokenFilter:**
   * Implement a filter (**JwtTokenFilter**) to intercept incoming requests and validate JWT tokens.
   * The filter will check the validity of the token and set the authenticated user in the security context.



1. **Database Schema:**
   * Define the database schema using the SQL script (**V1\_\_initial\_schema.sql**) to create the **user** table.



1. **Application Properties (application.properties):**

* Configure the PostgreSQL database connection properties.
* Set up JWT-related properties like token expiration time, secret key, etc.



1. **Exception Handling:**

* Implement custom exception handling to provide meaningful error responses.



Notes:

* For simplicity, we assume that user registration involves only basic user information like username, password, and email. You can add more fields as needed.
* The passwords will be securely hashed before saving to the database using BCrypt or a similar hashing algorithm.
* Upon successful registration and login, the service will generate JWT tokens that users can use to authenticate themselves for subsequent requests.

