Basic Spring Security - Assignment

Assignment

Objective

In this project, you will implement a basic **Spring Boot security configuration** to control access to REST API endpoints based on user roles. The objective is to help you understand the core concepts of **Spring Security**, including in-memory authentication, role-based authorization, and configuration of protected and public endpoints.

You will be working with a pre-configured Spring Boot application structure. Your task is to complete and understand the setup that enables restricted access to API endpoints for users with roles `ADMIN` and `USER`, using both **form-based login** and **basic authentication**.

Project Setup

Your project structure is:

- **Main Application Class (`BasicSecurityApplication`)**:
The entry point of the Spring Boot application. It launches the application context and scans the defined base package to detect components.

- **Controller Class (`MyController`)**:
A REST controller that defines secure endpoints `/api/user` and `/api/admin`. These endpoints are restricted to users with `USER` and `ADMIN` roles respectively.

- **Security Configuration Class (`SecurityConfig`)**:
Configures Spring Security for the application. It sets up:

- Role-based access rules using `AntPathRequestMatcher`
- In-memory authentication with users and roles
- Password encoding using `BCryptPasswordEncoder`
- Support for both form login and basic HTTP authentication
- Access permissions for development tools like the H2 console

- **Entity Class (`User`)**:
A simple JPA entity representing a user with fields for `id`, `username`, `password`, and `role`. This can be extended in the future to support persistence-based authentication.

Key Components to Implement

1. Controller Class

In the `MyController` class, you need to implement the following functionality:

- Class must have proper annotation to make it a rest controller.

- It must be mapped with the `/api` base request path.

- **Endpoint 1: User Endpoint (GET)**

  - Return Type: `ResponseEntity<String>`
  - Parameters: None
  - HTTP Method and Endpoint: Handles HTTP GET requests to the `/api/user` endpoint.
  - Functionality: Returns a greeting message `"Hello, User!"` for users with the `USER` role.

- **Endpoint 2: Admin Endpoint (GET)**

  - Return Type: `ResponseEntity<String>`
  - Parameters: None
  - HTTP Method and Endpoint: Handles HTTP GET requests to the `/api/admin` endpoint.
  - Functionality: Returns a greeting message `"Hello, Admin!"` for users with the `ADMIN` role.

2. Security Configuration Class

In the `SecurityConfig` class, you need to implement the following functionality:

1. **Class Annotation**:
   The class must be annotated as a configuration class to indicate that it provides Spring Security configuration.

2. **Security Filter Chain:**
   - **Definition**: Define a `SecurityFilterChain` bean to customize HTTP security for the application.
   - **URL Access Control**:
     - ➔ `/admin/**` URLs should be accessible only by users with the `ADMIN` role.

➔ `/user/**` URLs should be accessible only by users with the `USER` role.

➔ `/h2-console/**` should be publicly accessible for development purposes.

➔ All other requests must be authenticated.

- **Authentication Types:**
  - ➔ Enable form-based login for UI-based login flows.
  - ➔ Enable HTTP Basic authentication for testing and API access.
- **Security Settings:**
  - ➔ CSRF protection is disabled for simplicity and development/testing compatibility.

3. **Password Encoder Bean:**
   - **Purpose**: Define a `PasswordEncoder` bean using `BCryptPasswordEncoder` to securely encode passwords.
4. **Authentication Manager Bean:**
   - **Purpose:** Configure an `AuthenticationManager` with in-memory authentication.
   - **User Setup:**
     - ➔ User: `admin`, Password: `admin`, Role: `ADMIN`
     - ➔ User: `user`, Password: `password`, Role: `USER`
   - **Functionality:** Provides a custom authentication manager to validate user credentials and roles.

3. Entity Class – User

In the `User` entity class, you need to implement the following:
1. **Entity Annotation**: Add the appropriate annotation to the class to mark it as a JPA entity, which will map to a database table named `User`.

2. **Primary Key**: Add an annotation to the `id` field to mark it as the primary key, and configure it to auto-generate its value using the Identity strategy.

Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three**

horizontal lines at left top) -> Terminal -> New Terminal.

3. cd into your backend project folder.
4. To build your project use command:

   mvn clean package -Dmaven.test.skip

5. To launch your application, move into the target folder (cd target). Run the following command to run the application:

   java -jar <your application jar file name>

6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN. Please use 127.0.0.1 instead of localhost to test rest endpoints.
7. Mandatory: Before final submission run the following command:

   mvn test

8. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.