Tutorial 8 – JPA-Based Authentication with Spring Security

❖ Contents:

- We're gonna create a blog web application in Spring Boot. And because of the complexity of the application, this project will span a few tutorials, starting at this tutorial.
- In this tutorial, we will create the project and inplement the authentication part of it. To be specific, create a simple web application that uses JPA-based authentication (login and registration) based on the method learned in Lecture 8.
- We will also apply Hibernate Validation that we learned in Lecture 8 into the registration page.
- We will add other features such as: posts management, home page, monthly post archives, commenting… in the up coming tutorials.

❖ Instructions:

1. Create a new Spring Boot project with the following dependencies:
   - Spring Web
   - Thymeleaf
   - Spring Data JPA
   - Validation
   - MySQL Driver (or MariaDB Driver)
   - Spring Security
2. Check the actual type of your database (MySQL or MariaDB) so that you:

   a) Specify the correct connection string in application.properties.
   For MySQL, the connection string looks like:

   ```
   spring.datasource.url=jdbc:mysql://localhost:3306/se2tut8
   ```

   But for MariaDB, it looks like:

   ```
   spring.datasource.url=jdbc:mariadb://localhost:3306/se2tut8
   ```

   Also, Spring Boot recommends that you don't specify the database dialect, so you should remove or comment out this line in `application.properties`:

   ```
   spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
   ```

b) In your `pom.xml` file, use the correct JDBC Driver for your type of database server. For MySQL:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

And for MariaDB:

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
</dependency>
```

3. Create a `User` entity class under `model` package for storing your users' information such as `username`, `password`, `roles`, `address`, `avatar`, etc.

4. Needless to say, we should create a `UserRepository` class under `repository` package. Please note that, we need a method to find a single user given his username because this method will be used in the `JpaUserDetailsService` later.

5. Create a `MyUserDetails` class under `model` package to be used with the `JpaUserDetailsService` later. This class implements the `UserDetails` interface that Spring Security will use to authenticate your users. As suggested in Lecture 8, this `MyUserDetails` object should hold a `User` object as its attribute.

6. Create a `UserDto` class under `model` package for validating different user attributes. Please add suitable validation rules for your `username`, `password`, etc. in this `UserDto` class. To make it convenient to create a `User` from a `UserDto` object, add a constructor to the `User` class which receives a `UserDto` object and has access to a `PasswordEncoder` so that the validated password from `UserDto` can be encoded.

   (*) Please note that eventhough `MyUserDetails` and `UserDto` are under the model package, they are NOT entity classes.

7. Create a `JpaUserDetailsService` class in the `sevice` package that:
   - Implements the `UserDetailsService` interface.
   - Overrides the only essential method: `loadUserByUsername` (your job is to implement this method).

8. Add the `SecurityCfg` class to your project with the following details:

- Specifies a `JpaUserDetailsService` object as its user details service using the `userDetailsService()` method in the security filter chain. Please refer to the source code examples from Lecture 8 if you're unsure how to do this.
- Uses the **form login** method with default settings.
- Only allow authenticated users to access the `/member/` directory and everything inside it. Allows everyone to access all other URLs of the website.
- Create a `PasswordEncoder` bean which returns a `BCryptPasswordEncoder` object so that other classes in your application can use this `PasswordEncoder` through dependency injection.

9. Create a `HomeController` class under controller package that has a controller method for the root URL of the website (which should be `http://localhost:8080/`). Also create a Thymeleaf template named `home` as the view returned by this controller method.

10. Create an `AuthController` class under `controller` package that contains two controller methods that handle the following URLs:
    - `/register` (method GET): renders a Thymeleaf template named `register` which contains the registration form (`method="post"`). You should pass a new a `UserDto` object to be used in the `th:object` attribute of the `<form>` tag in the Thymeleaf template.
    - `/register` (method POST): this method handles the registration form. This controller method will:
        i. Validate the received `UserDto` object.
        ii. Use `UserRepository` to check if the username already exists in database. In this case, the `UserDto` object should be considered *invalid* and an *error message* string (saying the username already exists) should be passed into the `register` Thymeleaf template and showed to web user.
        iii. Renders the registration form again with error messages if there's any validation error.
        iv. If the `UserDto` object is valid, create a `User` object from the `UserDto` object.
        v. Save the `User` object into your database.
        vi. Redirects to the `/login` page (or better, shows a "registration successful" page that contains a link to the login page).

(*) Refer the source code examples in Lecture 8 to know how to handle validation error messages in Thymeleaf.

11. Create a `MemberController` class under `controller` package which contains a controller method which handles the GET request to the `/member/home` URL and shows a simple web page. To access this page, one must login. Put a link in Home page if you don't want to access this page by manually typing its URL.

❖ Run, debug & submit your project

Once finished, compress your project into a `.zip` file and submit the file to the tutorial's submission box.