Tp 08/09 : security

Objective:

The goal of this PW is to implement a simple **login application** using **Spring Boot** and **Spring Security**. With a **MySQL** database. we will learn how to secure an application by implementing authentication mechanisms, understand the structure of a Spring Boot application, and apply basic security techniques like form login and user role management.

You are required to create a Spring Boot application that includes:

- A login page.
- User authentication using Spring Security.
- User roles (e.g., Admin and User).
- Secure access to certain pages based on roles, storing user data in a MySQL database.
- In general , we have :
    - `home.html`: A simple welcome page.
    - `login.html`: The login form.
    - `user.html`: Accessible by users.
    - `admin.html`: Accessible only by admins.

Choose the following dependencies:

- Spring Web
- Spring Security
- Thymeleaf
- Spring Data JPA
- MySQL Driver

1/ Create a new MySQL database named **securityDb**. Then configure the MySQL connection.

2/ Create an entity class **User** that will be used for authentication. This class will map to a table ‘’users””(@Table(name = "users") in MySQL to store user credentials and roles.

```
public class User {
```

```
    private String username;
    private String password;
    private String role;

    // Getters and setters
}
```

3/ Create a repository interface to interact with the MySQL database using Spring Data JPA

```
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {     Optional<User> findByUsername(String username); }
```

Insert Default Users into the Database

Now, let's insert the default users (admin and user) into the database when the application starts. We'll use a CommandLineRunner or DataInitializer to accomplish this.

Using **CommandLineRunner**:

Create a class '' DataInitializer'' to automatically run at startup and insert default users into the MySQL database.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Override
    public void run(String... args) throws Exception {
        // Check if the users already exist in the database
        if (userRepository.findByUsername("admin").isEmpty()) {
            // Create and save the admin user
            User admin = new User();
            admin.setUsername("admin");
            admin.setPassword(passwordEncoder.encode("admin123"));
            admin.setRole("ROLE_ADMIN");
            userRepository.save(admin);
        }
```

```
        if (userRepository.findByUsername("user").isEmpty()) {
            // Create and save the user
            User user = new User();
            user.setUsername("user");
            user.setPassword(passwordEncoder.encode("user123"));
            user.setRole("ROLE_USER");
            userRepository.save(user);
        }
    }
}
```

You can use to post the data, by @PostConstruct, which will ensure that the data gets initialized once the Spring Boot application starts up.

```
package com.example.demo.config;

import com.example.demo.model.Role;
import com.example.demo.model.User;
import com.example.demo.repository.RoleRepository;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

@Component
public class DataInitializer {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RoleRepository roleRepository;

    @PostConstruct
    public void init() {
        // Create roles
        Role adminRole = new Role("ROLE_ADMIN");
        Role userRole = new Role("ROLE_USER");

        // Save roles to DB if they do not exist
        if (roleRepository.count() == 0) {
            roleRepository.save(adminRole);
            roleRepository.save(userRole);
        }

        // Create users with roles
        if (userRepository.count() == 0) {
            User adminUser = new User("admin", "admin123", adminRole);
            User normalUser = new User("user", "user123", userRole);

            userRepository.save(adminUser);
            userRepository.save(normalUser);  } }}
```

or you can also, create the ''user'' & ''admin'' directly in the database.

4/ Create a **`WebSecurityConfig`** class to configure Spring Security for authentication.

- We will configure Spring Security to authenticate users based on their username and password stored in the MySQL database.
- Also, we will set up roles such as `ROLE_USER` and `ROLE_ADMIN` to restrict access to certain pages.
- Example configuration:

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService; // to load user details
from the database

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .antMatchers("/user/**").hasAnyRole("USER", "ADMIN")
                .antMatchers("/", "/login").permitAll()
                .anyRequest().authenticated()
            .and()
                .formLogin()
                .loginPage("/login")
                .permitAll()
            .and()
                .logout()
                .permitAll();
    } // configure URLs to allow access for roles

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();}
```

// The `passwordEncoder()` method defines the `BCryptPasswordEncoder` for encoding and matching passwords.

5/ Create a service that will load user details from the MySQL database.

Example `Service`:

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not
found: " + username));

        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                AuthorityUtils.createAuthorityList(user.getRole())
        );
    }
}
```

6/ Create a LoginController to handle login requests and role-based redirections for user & admin.

Example `LoginController`:

```
@Controller
public class LoginController {

    @GetMapping("/")
    public String home() {
        return "home";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }

    @GetMapping("/user")
    public String userPage() {
        return "user";
    }

    @GetMapping("/admin")
    public String adminPage() {
        return "admin"; }}
```

7/ Create basic HTML templates for login, home, user, and admin pages using Thymeleaf:

Example for `login.html`:

```
<html>
<body>
    <h2>Login</h2>
    <form action="/login" method="post">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
        </div>
        <button type="submit">Login</button>
```

```
    </form>
</body>
</html>
```

Testing:

- Run your application and test **the login functionality.**
- Try logging in **as `admin`** and accessing the `/admin` page.
- Try logging in as `user` and accessing the `/user` page.
- Ensure that only users with the correct roles can access the pages.