(IntelliJ IDEA, Eclipse, VS Code).

STEP 2: Configure Web Security (Authentication & Authorization)

This is where we define security rules, user details, and password encoding.

- Create a new package com.example.security.config in src/main/java/.
- Create a Java class named SecurityConfig.java inside this package:

```
// src/main/java/com/example/security/config/SecurityConfig.java
package com.example.security.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity; // For method security
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; // For password encoding
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager; // For in-memory users
import org.springframework.security.web.SecurityFilterChain;

/**
 * Spring Security configuration class.
 * @EnableWebSecurity: Enables Spring Security's web security features.
```

```
 * @EnableMethodSecurity: Enables method-level security annotations like
@PreAuthorize.
 */
@Configuration
@EnableWebSecurity
@EnableMethodSecurity // Enables @PreAuthorize, @PostAuthorize, @Secured,
@RolesAllowed
public class SecurityConfig {

    /**
     * Configures the SecurityFilterChain, defining authorization rules for HTTP
requests.
     * This is the core of Web Security Configuration.
     *
     * @param http HttpSecurity object to configure security.
     * @return A SecurityFilterChain instance.
     * @throws Exception if configuration fails.
     */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
        http
            .authorizeHttpRequests(authorize -> authorize
                // Publicly accessible paths
                .requestMatchers("/", "/home", "/public").permitAll()
                // Paths requiring specific roles (Authorization)
                .requestMatchers("/user/**").hasAnyRole("USER", "ADMIN") // /user
paths require USER or ADMIN role
                .requestMatchers("/admin/**").hasRole("ADMIN") // /admin paths require
ADMIN role
                // All other requests require authentication
```

```
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login") // Custom login page URL
                .permitAll() // Allow everyone to access the login page
            )
            .logout(logout -> logout
                .permitAll() // Allow everyone to logout
            );
            // CSRF Protection is enabled by default in Spring Security.
            // For simple forms, Spring Security automatically adds a CSRF token.
            // For REST APIs, you might disable it or handle it differently.
            // .csrf(csrf -> csrf.disable()); // Uncomment to disable CSRF (e.g., for
stateless REST APIs)

        return http.build();
    }

    /**
     * Configures an in-memory UserDetailsService for Authentication.
     * In a real application, this would typically retrieve users from a database.
     *
     * @return A UserDetailsService with predefined users.
     */
    @Bean
    public UserDetailsService userDetailsService(PasswordEncoder
passwordEncoder) {
        // Define a 'user' with password 'password' and role 'USER'
        UserDetails user = User.builder()
            .username("user")
            .password(passwordEncoder.encode("password")) // Encode password
```

```
            .roles("USER")
            .build();

        // Define an 'admin' with password 'admin' and role 'ADMIN'
        UserDetails admin = User.builder()
            .username("admin")
            .password(passwordEncoder.encode("admin")) // Encode password
            .roles("ADMIN", "USER") // Admin also has USER role
            .build();

        // Return an InMemoryUserDetailsManager with these users
        return new InMemoryUserDetailsManager(user, admin);
    }

    /**
     * Configures a PasswordEncoder.
     * BCryptPasswordEncoder is recommended for strong password hashing (Password Encoding).
     *
     * @return A BCryptPasswordEncoder instance.
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

STEP 3: Create Controllers and Views

We'll create simple controllers and Thymeleaf HTML files to demonstrate different access levels.

- Create a new package com.example.security.controller in src/main/java/.

- Create a Java class named WebController.java inside this package:

```
// src/main/java/com/example/security/controller/WebController.java
package com.example.security.controller;

import org.springframework.security.access.prepost.PreAuthorize; // For Method Security
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 * Web Controller for serving HTML pages and demonstrating access control.
 */
@Controller
public class WebController {

    @GetMapping("/home")
    public String home() {
        return "home"; // Resolves to src/main/resources/templates/home.html
    }

      @GetMapping("/")
       public String defaultPage() {
           return "home"; // Resolves to src/main/resources/templates/home.html
       }
@GetMapping("/public")
public String publicPage() {
   return "public-page"; // Resolves to src/main/resources/templates/public-page.html
}
```

```
    @GetMapping("/user/dashboard")
    public String userDashboard(Model model) {
        model.addAttribute("message", "Welcome, User! This is your dashboard.");
        return "user-dashboard"; // Resolves to src/main/resources/templates/user-
dashboard.html
    }

    @GetMapping("/admin/panel")
    public String adminPanel(Model model) {
        model.addAttribute("message", "Welcome, Admin! This is your control panel.");
        return "admin-panel"; // Resolves to src/main/resources/templates/admin-
panel.html
    }

    @GetMapping("/login")
    public String login() {
        return "login"; // Resolves to src/main/resources/templates/login.html
    }

    /**
     * Demonstrates Method Security using @PreAuthorize.
     * Only users with the 'ADMIN' role can access this method.
     */
    @PreAuthorize("hasRole('ADMIN')") // Method Security: Requires ADMIN role
    @GetMapping("/admin/secure-action")
    public String secureAdminAction(Model model) {
        model.addAttribute("message", "You accessed a highly secure admin action!");
        return "admin-panel"; // Redirect back to admin panel with a message
    }

    /**
```

```
     * Demonstrates Method Security using @PreAuthorize.
     * Only users with the 'USER' role (or ADMIN, since ADMIN has USER role) can access this method.
     */
    @PreAuthorize("hasRole('USER')") // Method Security: Requires USER role
    @GetMapping("/user/secure-action")
    public String secureUserAction(Model model) {
        model.addAttribute("message", "You accessed a secure user action!");
        return "user-dashboard"; // Redirect back to user dashboard with a message
    }
}
```

- Create a new package com.example.security.rest in src/main/java/.
- Create a Java class named SecureRestController.java inside this package:

```
// src/main/java/com/example/security/rest/SecureRestController.java
package com.example.security.rest;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * REST Controller to demonstrate Security in REST APIs.
 * Spring Security applies to REST endpoints just like regular web endpoints.
 */
@RestController
@RequestMapping("/api")
public class SecureRestController {

    @GetMapping("/public-data")
```

```
    public String getPublicData() {
        return "This data is accessible to anyone.";
    }

    @GetMapping("/user-data")
    @PreAuthorize("hasRole('USER')") // Requires USER role via Method Security
    public String getUserData() {
        return "This data is accessible to authenticated users with USER role.";
    }

    @GetMapping("/admin-data")
    @PreAuthorize("hasRole('ADMIN')") // Requires ADMIN role via Method Security
    public String getAdminData() {
        return "This data is accessible only to users with ADMIN role.";
    }
}
```

- Create Thymeleaf HTML files in src/main/resources/templates/:
  1. **home.html**:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-color:
#f4f4f4; color: #333; }
        .container { background-color: #fff; padding: 30px; border-radius: 8px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: auto; text-align:
center; }
        h1 { color: #0056b3; }
        a { color: #007bff; text-decoration: none; margin: 0 10px; }
        a:hover { text-decoration: underline; }
        .nav-links { margin-top: 20px; }
    </style>
```

```
</head>
<body>
    <div class="container">
        <h1>Welcome to the Spring Security Demo!</h1>
        <p>Explore different pages based on your roles.</p>
        <div class="nav-links">
            <a th:href="@{/public}">Public Page</a>
            <a th:href="@{/user/dashboard}">User Dashboard</a>
            <a th:href="@{/admin/panel}">Admin Panel</a>
            <a th:href="@{/login}">Login</a>
            <form th:action="@{/logout}" method="post" style="display:inline;">
                <button type="submit" style="background: none; border: none; color:
#dc3545; cursor: pointer; font-size: 1em; text-decoration:
underline;">Logout</button>
            </form>
        </div>
    </div>
</body>
</html>
```

2. **public-page.html**:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Public Page</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-color:
#f4f4f4; color: #333; }
        .container { background-color: #fff; padding: 30px; border-radius: 8px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: auto; text-align:
center; }
        h1 { color: #28a745; }
        a { color: #007bff; text-decoration: none; margin: 0 10px; }
        a:hover { text-decoration: underline; }
    </style>
</head>
<body>
    <div class="container">
        <h1>This is a Public Page</h1>
        <p>Anyone can access this page without logging in.</p>
```

```
        <a th:href="@{/home}">Go to Home</a>
    </div>
</body>
</html>
```

3. **user-dashboard.html**:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>User Dashboard</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-color:
#f4f4f4; color: #333; }
        .container { background-color: #fff; padding: 30px; border-radius: 8px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: auto; text-align:
center; }
        h1 { color: #ffc107; }
        p { font-size: 1.1em; color: #555; }
        a { color: #007bff; text-decoration: none; margin: 0 10px; }
        a:hover { text-decoration: underline; }
        .message { margin-top: 20px; padding: 10px; background-color: #e2f0d9;
border-left: 5px solid #28a745; text-align: left; }
    </style>
</head>
<body>
    <div class="container">
        <h1>User Dashboard</h1>
        <p th:text="${message}"></p>
        <p>You have successfully logged in as a USER (or ADMIN).</p>
        <div th:if="${#authorization.expression('hasRole("ADMIN")')}">
            <p>You also have ADMIN privileges.</p>
        </div>
        <a th:href="@{/user/secure-action}">Access Secure User Action (Method
Security)</a>
        <a th:href="@{/home}">Go to Home</a>
        <form th:action="@{/logout}" method="post" style="display:inline;">
            <button type="submit" style="background: none; border: none; color:
#dc3545; cursor: pointer; font-size: 1em; text-decoration:
underline;">Logout</button>
        </form>
```

```
    </div>
</body>
</html>
```

4. **admin-panel.html**:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Admin Panel</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-color:
#f4f4f4; color: #333; }
        .container { background-color: #fff; padding: 30px; border-radius: 8px; box-
shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; margin: auto; text-align:
center; }
        h1 { color: #dc3545; }
        p { font-size: 1.1em; color: #555; }
        a { color: #007bff; text-decoration: none; margin: 0 10px; }
        a:hover { text-decoration: underline; }
        .message { margin-top: 20px; padding: 10px; background-color: #f8d7da;
border-left: 5px solid #dc3545; text-align: left; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Admin Panel</h1>
        <p th:text="${message}"></p>
        <p>You have successfully logged in as an ADMIN.</p>
        <a th:href="@{/admin/secure-action}">Access Highly Secure Admin Action
(Method Security)</a>
        <a th:href="@{/home}">Go to Home</a>
        <form th:action="@{/logout}" method="post" style="display:inline;">
            <button type="submit" style="background: none; border: none; color:
#dc3545; cursor: pointer; font-size: 1em; text-decoration:
underline;">Logout</button>
        </form>
    </div>
</body>
</html>
```

5. **login.html**:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
    <style>
        body { font-family: Arial, sans-serif; background-color: #f4f4f4; display: flex;
justify-content: center; align-items: center; height: 100vh; margin: 0; }
        .login-container { background-color: #fff; padding: 40px; border-radius:
8px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); width: 350px; text-align: center; }
        h1 { color: #0056b3; margin-bottom: 25px; }
        .form-group { margin-bottom: 20px; text-align: left; }
        .form-group label { display: block; margin-bottom: 8px; font-weight: bold;
color: #555; }
        .form-group input[type="text"],
        .form-group input[type="password"] {
            width: calc(100% - 22px);
            padding: 12px;
            border: 1px solid #ddd;
            border-radius: 5px;
            font-size: 1em;
            box-sizing: border-box;
        }
        .error-message { color: #dc3545; margin-bottom: 15px; font-weight: bold; }
        .login-button {
            background-color: #007bff;
            color: white;
            padding: 12px 25px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            font-size: 1.1em;
            transition: background-color 0.3s ease;
            width: 100%;
        }
        .login-button:hover { background-color: #0056b3; }
        .info-text { margin-top: 20px; font-size: 0.9em; color: #777; }
        .info-text strong { color: #333; }
    </style>
</head>
```

```
<body>
    <div class="login-container">
        <h1>Login</h1>
        <div th:if="${param.error}" class="error-message">
            Invalid username or password.
        </div>
        <div th:if="${param.logout}" class="error-message" style="color: green;">
            You have been logged out.
        </div>
        <form th:action="@{/login}" method="post">
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" required>
            </div>
            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" required>
            </div>
            <button type="submit" class="login-button">Log In</button>
        </form>
        <div class="info-text">
            <p>Try with:</p>
            <p><strong>Username:</strong> user, <strong>Password:</strong>
password (Role: USER)</p>
            <p><strong>Username:</strong> admin, <strong>Password:</strong>
admin (Role: ADMIN)</p>
        </div>
    </div>
</body>
</html>
```

STEP 4: Main Spring Boot Application Class

This is the standard Spring Boot application entry point.

- Open your main application class (e.g., SecurityDemoApplication.java in com.example.security).
  // src/main/java/com/example/security/app/SecurityDemoApplication.java
  package com.example.security;

  import org.springframework.boot.SpringApplication;
  import org.springframework.boot.autoconfigure.SpringBootApplication;

```
@SpringBootApplication
public class SecurityDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SecurityDemoApplication.class, args);
    }
}
```

STEP 5: Run the Application and Test Security

1. **Run the Spring Boot Application:**
    - Open your main application class (SecurityDemoApplication.java).
    - Run it as a Java Application from your IDE, or use mvn spring-boot:run from the terminal in your project root.
2. **Test Access Control:**
    - **Public Page:**
        - Open your browser and go to http://localhost:8080/public. You should be able to access this page without logging in.
        - Go to http://localhost:8080/home. This is also publicly accessible.
    - **Login Page:**
        - Try to go to http://localhost:8080/user/dashboard. Spring Security will redirect you to the /login page because you are not authenticated.
        - Use the provided credentials:
            - **Username:** user, **Password:** password
            - **Username:** admin, **Password:** admin
    - **User Dashboard (Authorization):**
        - Log in as user.
        - Go to http://localhost:8080/user/dashboard. You should see the user dashboard.
        - Try to go to http://localhost:8080/admin/panel. You should be redirected to an "Access Denied" page (or login page if not authenticated) because the user role does not have access to /admin/** paths.
    - **Admin Panel (Authorization):**

        - Log out (click the logout button on the home/dashboard page) and log in as admin.
        - Go to http://localhost:8080/admin/panel. You should see the admin panel.
        - You can also access http://localhost:8080/user/dashboard as admin because the admin user has both ADMIN and USER roles.
    - **Method Security (@PreAuthorize):**
        - Log in as user.
        - Go to http://localhost:8080/user/secure-action. You should see the message "You accessed a secure user action!"
        - Try to go to http://localhost:8080/admin/secure-action. You should get an "Access Denied" error because the user role cannot access this method.
        - Log out and log in as admin.
        - Go to http://localhost:8080/admin/secure-action. You should now successfully access this method.
    - **Security in REST APIs:**
        - Open a new browser tab or use a tool like Postman/curl.
        - http://localhost:8080/api/public-data: Accessible without authentication.
        - http://localhost:8080/api/user-data: Requires authentication with USER or ADMIN role. If not logged in, it will redirect to login (for browser) or return 401 Unauthorized (for API clients).
        - http://localhost:8080/api/admin-data: Requires authentication with ADMIN role.

CSRF Protection

- Spring Security enables CSRF (Cross-Site Request Forgery) protection by default.
- For form submissions (like our login form), Spring Security automatically injects a hidden input field named _csrf with a token. You can see this in login.html form.
- When you submit the form, this token is sent with the request, and Spring Security validates it. If the token is missing or invalid, the request is rejected.
- For stateless REST APIs, CSRF protection is often disabled (.csrf(csrf -> csrf.disable()) in SecurityConfig) because tokens are typically handled differently (e.g., JWTs) or not applicable. However, for a stateful web application, keeping

CSRF enabled is a good practice.

Password Encoding

- We used BCryptPasswordEncoder in SecurityConfig. This is a strong, one-way hashing algorithm for passwords.
- When a user registers or logs in, Spring Security uses this PasswordEncoder to hash the provided password and compare it with the stored (hashed) password. It never stores or compares plain-text passwords.

You have successfully implemented a basic Spring Security setup! You've learned about:

- The fundamental concepts of authentication and authorization.
- Configuring web security rules using HttpSecurity.
- Defining in-memory users with UserDetailsService.
- Implementing method-level security with @PreAuthorize.
- The importance of PasswordEncoder for secure password storage.
- Basic security considerations for REST APIs and CSRF protection.

This activity provides a strong foundation for securing your Spring applications.

Activity 8.1: Spring Remoting “HTTP-based Communication with RestTemplate”

This activity will guide you through implementing HTTP-based communication using Spring's RestTemplate. You will create two separate Spring Boot applications: a **Server** that exposes a REST API, and a **Client** that consumes it using RestTemplate.

In this activity, we will focus on using RestTemplate to demonstrate client-server communication over HTTP.

STEP 1: Project Setup (Server Application - REST API)

First, let's set up the Spring Boot application that will expose our REST endpoints.

1. **Go to Spring Initializr:** Open your web browser and navigate to https://start.spring.io/.
2. **Configure Your Project:**
   - **Project:** Maven Project
   - **Language:** Java
   - **Spring Boot:** Choose the latest stable version (e.g., 3.x.x).
   - **Group:** com.example.restcomm
   - **Artifact:** rest-server
   - **Name:** rest-server
   - **Description:** Spring REST Server Demo
   - **Package Name:** com.example.restcomm.server
   - **Packaging:** Jar
   - **Java:** Choose Java 17 or higher.
3. **Add Dependencies:** In the "Dependencies" section, search for and add the following:
   - **Spring Web:** Essential for building REST controllers.
   - **Lombok:** (Optional but recommended) Reduces boilerplate code.
4. **Generate and Download:** Click the "Generate" button. Download the .zip file.
5. **Import into IDE:** Unzip the downloaded file and import the project into your IDE (IntelliJ IDEA, Eclipse, VS Code).