

Session 10

Bootstrap, role based authentication and CSRF

The final session adds roles so that an admin sees pages a student cannot, shows who is logged in, and turns on CSRF protection so that forms cannot be submitted from another site.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 41 to 46 of the manual: bootstrap, role based authentication and csrf
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✕ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q41	Add Bootstrap styling in the Login and registration page in the above exercises in...	Complete
Q42	Create a Role Table in the database and configure it with Spring Boot, Security and...	Complete
Q43	Write code for role-based authentication using Spring Security	Complete
Q44	Display current logged in User details on the dashboard along with client IP, data...	Complete
Q45	Add CSRF functionality in the authentication	Complete
Q46	Restrict role-based access to views in Spring Boot Security	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- Role table with a many-to-many link to User; load roles as `GrantedAuthority` in `UserDetailsService`.
- Restrict URLs with `.requestMatchers("/admin/**").hasRole("ADMIN")` and views with `sec:authorize` (Thymeleaf) or `<sec:authorize>` (JSP).
- CSRF is on by default in Spring Security 6; forms must carry the token (`th:action` adds it automatically, plain HTML forms need a hidden `_csrf` input).

This is the last set of changes to `admission-api`. Files marked "replaces" overwrite the Session 9 version. Two accounts exist after the seeder runs: `admin` / `admin123` with `ROLE_ADMIN` and `ROLE_STUDENT`, and `asha` / `asha123` with `ROLE_STUDENT` only. Nothing was executed here; every listing and expected page was checked by reading against Spring Security 6.3 and Bootstrap 5.3.

Question 41

Problem Statement

■ Write in lab record

Add Bootstrap styling in the Login and registration page in the above exercises in session 9.

Solution

■ Write in lab record

Steps

1. Replace `templates/login.html` and `templates/register.html` with the listings. Both load Bootstrap 5.3 from the jsDelivr CDN with one `<link>` tag; no JavaScript bundle is needed for forms and cards.
2. Offline alternative (the manual's WebJars route): add `org.webjars:bootstrap:5.3.3` and `org.webjars:webjars-locator-core` to `pom.xml`, then link `/webjars/bootstrap/css/bootstrap.min.css`. Add `/webjars/**` to the `permitAll()` list.
3. Restart and open `/login` and `/register`. Resize the window; the card stays centred and shrinks to full width below 768 px.

4. Submit an invalid registration; the invalid fields get a red border and the message appears below the field.

Program

■ Lab record: every tab is one file of the answer. Write all of them.

login.html

login.html (replaces)

```
1 <!DOCTYPE html>
2 <!-- src/main/resources/templates/login.html (Session 10: Q41 Bootstrap, Q
3 <html xmlns:th="http://www.thymeleaf.org">
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1">
7   <title>Admission Portal - Login</title>
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstra
9 </head>
10 <body class="bg-light">
11 <div class="container">
12   <div class="row justify-content-center mt-5">
13     <div class="col-md-4">
14       <div class="card shadow-sm">
15         <div class="card-body">
16           <h4 class="card-title text-center mb-3">Student Admission Portal</
17
18           <div th:if="${param.error}" class="alert alert-danger py-2">Invali
19           <div th:if="${param.logout}" class="alert alert-success py-2">You
20
21           <!-- Plain action (not th:action) so the CSRF token must be added
22           <form action="/login" method="post">
23             <input type="hidden" th:name="${_csrf.parameterName}" th:value="
24             <div class="mb-3">
25               <label for="username" class="form-label">Username</label>
26               <input type="text" class="form-control" id="username" name="us
27             </div>
28             <div class="mb-3">
29               <label for="password" class="form-label">Password</label>
30               <input type="password" class="form-control" id="password" name
31             </div>
32             <button type="submit" class="btn btn-primary w-100">Sign in</but
33           </form>
34
35           <p class="text-center mt-3 mb-0">New student? <a th:href="@{/regis
36         </div>
37       </div>
38     </div>
39   </div>
40 </div>
41 </body>
42 </html>
```

Output

Checked by reading. `/login`: light grey page, a white card about a third of the width, centred, with the title "Student Admission Portal", two labelled inputs with rounded borders, a full-width blue "Sign in" button and a centred "New student? Register" line. After a wrong password a red alert "Invalid username or password." appears above the form; after logout a green alert "You have been logged out."

`/register`: same layout, slightly wider card, green "Register and sign in" button. On a failed submit each offending input turns red-bordered (`is-invalid`) and its message shows in red under it, for example "Username must be 4 to 50 characters".

Explanation

Bootstrap is only CSS classes, so the Thymeleaf attributes from Session 9 are untouched. The layout is the standard grid: `container`, `row justify-content-center`, `col-md-4` (four of twelve columns on medium screens and up, full width below). `form-control` styles the inputs and `form-label` the labels. Validation display uses two Bootstrap conventions together: `th:classappend` adds `is-invalid` to the input when `#fields.hasErrors` is true, and the sibling `div.invalid-feedback` is shown by Bootstrap only when the previous input has `is-invalid`, so no extra JavaScript is required. `novalidate` on the register form turns off the browser's own popups so the server-side messages are the ones the examiner sees.

Question 42

Problem Statement

■ Write in lab record

Create a Role Table in the database and configure it with Spring Boot, Security and Hibernate.

Solution

■ Write in lab record

Steps

1. Add `Role` entity and `RoleRepository`.

2. Replace `User.java` with the version that has the `@ManyToMany` `roles` set.
3. Replace `DataSeeder.java` so it creates the two roles and two users, and `AppUserDetailsService.java` so it turns the roles into authorities.
4. Drop the old `users` rows once (`DELETE FROM users;`) so the seeder recreates `admin` with roles, or add the link rows by hand.
5. Restart; check with the `SELECT` at the end of `roles.sql` .

Program

- Lab record: every tab is one file of the answer. Write all of them.

Role.java

Role.java

```
1 // src/main/java/in/ignou/admission/entity/Role.java (Session 10, Q42)
2 package in.ignou.admission.entity;
3
4 import jakarta.persistence.Column;
5 import jakarta.persistence.Entity;
6 import jakarta.persistence.GeneratedValue;
7 import jakarta.persistence.GenerationType;
8 import jakarta.persistence.Id;
9 import jakarta.persistence.Table;
10
11 /** Name carries the ROLE_ prefix so it maps 1:1 to a GrantedAuthority string
12 @Entity
13 @Table(name = "roles")
14 public class Role {
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19
20     @Column(nullable = false, unique = true, length = 30)
21     private String name;
22
23     public Role() { }
24
25     public Role(String name) { this.name = name; }
26
27     public Long getId() { return id; }
28     public void setId(Long id) { this.id = id; }
29     public String getName() { return name; }
30     public void setName(String name) { this.name = name; }
31 }
```

Output

Checked by reading. Console on the first start after the change:

```

1 Hibernate: create table roles (id bigint not null auto_increment, name varchar
2 Hibernate: create table user_roles (role_id bigint not null, user_id bigint
3 Hibernate: alter table user_roles add constraint FK... foreign key (role_id)
4 Hibernate: alter table user_roles add constraint FK... foreign key (user_id)
5 Seeded admin with 2 role(s)
6 Seeded asha with 1 role(s)

```

The join query from `roles.sql` :

```

1 +-----+-----+
2 | username | name      |
3 +-----+-----+
4 | admin    | ROLE_ADMIN |
5 | admin    | ROLE_STUDENT |
6 | asha     | ROLE_STUDENT |
7 +-----+-----+

```

Explanation

A user can hold several roles and a role belongs to many users, so the relation is many-to-many and needs a join table. `@JoinTable(name = "user_roles", joinColumns = user_id, inverseJoinColumns = role_id)` names it; Hibernate creates it with a composite primary key and two foreign keys. `FetchType.EAGER` is deliberate: the roles are needed on every login, the set is tiny, and a lazy set would fail with `LazyInitializationException` once the session closes after `loadUserByUsername`. The role name carries the `ROLE_` prefix in the table so that `new SimpleGrantedAuthority(r.getName())` is the exact string Spring compares: `hasRole("ADMIN")` checks for the authority `ROLE_ADMIN`. The seeder uses `findByName(...).orElseGet(save)` so it is safe to run on every start.

Question 43

Problem Statement

■ Write in lab record

Write code for role-based authentication using Spring Security.

Solution

■ Write in lab record

Steps

1. Replace `SecurityConfig.java` with the version whose `authorizeHttpRequests` block has the role rules.
2. Replace `PageController.java` (it gains `/admin/users`) and add `templates/admin.html`.
3. Restart. Log in as `asha`, open `/admin/users`: a white "Whitelabel Error Page" with status 403 Forbidden. Log in as `admin`: the user table appears.
4. Repeat from curl with HTTP Basic to capture the status codes.

Program

■ Lab record: every tab is one file of the answer. Write all of them.

SecurityConfig.java

SecurityConfig.java (replaces)

```
1 // src/main/java/in/ignou/admission/security/SecurityConfig.java (Session
2 package in.ignou.admission.security;
3
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.security.authentication.AuthenticationManager;
7 import org.springframework.security.config.Customizer;
8 import org.springframework.security.config.annotation.authentication.configu
9 import org.springframework.security.config.annotation.web.builders.HttpSecur
10 import org.springframework.security.config.annotation.web.configuration.Enab
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12 import org.springframework.security.crypto.password.PasswordEncoder;
13 import org.springframework.security.web.SecurityFilterChain;
14
15 @Configuration
16 @EnableWebSecurity
17 public class SecurityConfig {
18
19     @Bean
20     public PasswordEncoder passwordEncoder() {
21         return new BCryptPasswordEncoder();
22     }
23
24     @Bean
25     public AuthenticationManager authenticationManager(AuthenticationConfigu
26         return config.getAuthenticationManager();
27     }
28
29     @Bean
30     public SecurityFilterChain filterChain(HttpSecurity http) throws Excepti
31         http
32             // Q43: rules are checked top to bottom, first match wins; put t
33             .authorizeHttpRequests(auth → auth
34                 .requestMatchers("/login", "/register", "/css/**", "/actuato
35                 .requestMatchers("/admin/**").hasRole("ADMIN")
36                 .requestMatchers("/actuator/**").hasRole("ADMIN")
37                 .requestMatchers("/api/**", "/xml/**").hasAnyRole("ADMIN", "
38                 .anyRequest().authenticated())
39             .formLogin(form → form
40                 .loginPage("/login")
41                 .defaultSuccessUrl("/dashboard", true)
42                 .permitAll())
43             .logout(logout → logout
```

```
44         .logoutUrl("/logout")
45         .logoutSuccessUrl("/login?logout")
46         .invalidateHttpSession(true)
47         .deleteCookies("JSESSIONID")
48         .permitAll())
49     .httpBasic(Customizer.withDefaults())
50     // Q45: CSRF stays ON for every browser form (login, register, l
51     // Only the token-less JSON endpoints used from curl are exclude
52     .csrf(csrf → csrf.ignoringRequestMatchers("/api/**", "/xml/**",
53     return http.build();
54 }
55 }
```

Output

Checked by reading. The access matrix, each row one curl call:

```
1 curl -s -o /dev/null -w "%{http_code}\n" -u asha:asha123 http://localhost:8080/login?logout BASH:
2 curl -s -o /dev/null -w "%{http_code}\n" -u admin:admin123 http://localhost:8080/login?logout
3 curl -s -o /dev/null -w "%{http_code}\n" -u asha:asha123 http://localhost:8080/login?logout
4 curl -s -o /dev/null -w "%{http_code}\n" -u asha:asha123 http://localhost:8080/login?logout
5 curl -s -o /dev/null -w "%{http_code}\n" http://localhost:8080/login?logout
6 curl -s -o /dev/null -w "%{http_code}\n" http://localhost:8080/login?logout
```

```
1 403
2 200
3 200
4 403
5 200
6 302
```

URL	Anonymous	asha (STUDENT)	admin (ADMIN, STUDENT)
/login , /register , /actuator/health	200	200	200
/dashboard	302 to /login	200	200
/api/students , /xml/courses	401 (curl) or 302 (browser)	200	200
/admin/users	401 or 302	403	200
/actuator/metrics , /actuator/env	401 or 302	403	200

admin.html as admin, with the seeded rows:

```

1 Registered users (seen by admin)
2 Id Username Enabled Roles
3 1 admin true ROLE_ADMIN ROLE_STUDENT
4 2 asha true ROLE_STUDENT

```

Explanation

Authentication answers “who are you”, authorisation answers “may you”. The role rules live in `authorizeHttpRequests` and are evaluated top to bottom, first match wins, so the specific patterns (`/admin/**` , `/actuator/**` , `/api/**`) come before `anyRequest() . hasRole("ADMIN")` is shorthand for `hasAuthority("ROLE_ADMIN")` ; `hasAnyRole` accepts any of the list. An authenticated user who fails a rule gets 403 from `ExceptionTranslationFilter` , while an anonymous one gets the login redirect (or 401 for Basic clients), which is why the same URL shows two different failures in the matrix. `/actuator/health` stays open so a monitoring tool can poll it without a password, but the rest of Actuator is admin-only because `env` and `heapdump` leak secrets. `admin` holds both roles so the same person can test every row of the matrix.

Question 44

Problem Statement

■ Write in lab record

Display current logged in User details on the dashboard along with client IP, data time and user's current role.

Solution

■ Write in lab record

Steps

1. `PageController.dashboard` (replaced in Question 43) takes `Authentication` and `HttpServletRequest` as parameters and puts `username`, `roles`, `clientId`, `serverTime` and `sessionId` in the model.
2. Replace `templates/dashboard.html` with the listing.
3. Log in as `admin`, then as `asha`, and compare the Role(s) row and the buttons (buttons belong to Question 46).
4. Open the dashboard from another machine on the LAN (`http://YOUR-IP:8080/dashboard`) to see a client IP other than `127.0.0.1`.

Program

dashboard.html (replaces)

```

1 <!DOCTYPE html>
2 <!-- src/main/resources/templates/dashboard.html (Session 10: Q44 user det
3 <html xmlns:th="http://www.thymeleaf.org"
4     xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
5 <head>
6     <meta charset="UTF-8">
7     <!-- Q45: token for JavaScript callers (fetch/XMLHttpRequest) -->
8     <meta name="_csrf" th:content="${_csrf.token}">
9     <meta name="_csrf_header" th:content="${_csrf.headerName}">
10    <title>Admission Portal - Dashboard</title>
11    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstra
12 </head>
13 <body class="bg-light">
14 <nav class="navbar navbar-dark bg-dark">
15     <div class="container">
16         <span class="navbar-brand">Student Admission Portal</span>
17         <form th:action="@{/logout}" method="post" class="d-flex">
18             <button type="submit" class="btn btn-outline-light btn-sm">Logout</but
19         </form>
20     </div>
21 </nav>
22
23 <div class="container mt-4">
24     <div class="card shadow-sm">
25         <div class="card-header">Current session</div>
26         <table class="table mb-0">
27             <tr><th>User</th><td th:text="${username}">admin</td></tr>
28             <tr><th>Role(s)</th><td th:text="${roles}">ROLE_ADMIN</td></tr>
29             <tr><th>Client IP</th><td th:text="${clientIp}">127.0.0.1</td></tr>
30             <tr><th>Date and time</th><td th:text="${serverTime}">01-09-2026 10:15
31             <tr><th>Session id</th><td th:text="${sessionId}"></td></tr>
32         </table>
33     </div>
34
35     <div class="mt-4">
36         <!-- Q46: everyone who is logged in sees this -->
37         <a sec:authorize="isAuthenticated()" th:href="@{/api/students}" class="b
38
39         <!-- Q46: rendered only when the authority ROLE_ADMIN is present -->
40         <a sec:authorize="hasRole('ADMIN')" th:href="@{/admin/users}" class="btn
41         <a sec:authorize="hasRole('ADMIN')" th:href="@{/actuator/health}" class=
42
43         <p sec:authorize="!hasRole('ADMIN')" class="text-muted mt-2">Admin funct

```

```
44     </div>
45 </div>
46 </body>
47 </html>
```

The handler that fills the model, from `PageController` above:

```
1 @GetMapping({"/", "/dashboard"})
2 public String dashboard(Authentication auth, HttpServletRequest request, Model model) {
3     model.addAttribute("username", auth.getName());
4     model.addAttribute("roles", auth.getAuthorities().stream()
5         .map(GrantedAuthority::getAuthority)
6         .collect(Collectors.joining(", ")));
7     model.addAttribute("clientIp", request.getRemoteAddr());
8     model.addAttribute("serverTime", LocalDateTime.now().format(FMT));
9     model.addAttribute("sessionId", request.getSession().getId());
10    return "dashboard";
11 }
```

Output

Checked by reading. Logged in as `admin` from the same machine:

```
1 Current session
2 User          admin
3 Role(s)       ROLE_ADMIN, ROLE_STUDENT
4 Client IP     127.0.0.1
5 Date and time 26-09-2026 10:15:42
6 Session id    3F1A9C2E7B5D4A6F8E0C1B2D3A4F5E6C
7
8 [Students (JSON)] [Admin: user list] [Health]
```

Logged in as `asha` from a laptop on the same Wi-Fi:

```
1 User          asha
2 Role(s)       ROLE_STUDENT
3 Client IP     192.168.1.23
4 Date and time 26-09-2026 10:17:05
5 Session id    9B7E2D4C1F0A3E5B6D8C7A9F2E1B4C3D
6
7 [Students (JSON)]
8 Admin functions are hidden for your role.
```

If the browser uses IPv6 for localhost the IP row shows `0:0:0:0:0:0:0:1`; that is correct, not a bug.

Explanation

Spring MVC resolves an `Authentication` parameter from the `SecurityContextHolder`, so the controller never touches the holder directly. `auth.getName()` is the username; `getAuthorities()` is the collection built in `AppUserDetailsService`, joined into one string for display. The IP comes from the servlet request; `getRemoteAddr()` is the TCP peer, which is the client on a LAN and the proxy if one is in front (then read `X-Forwarded-For`). The time is the server clock, formatted once with a shared `DateTimeFormatter` because the formatter is thread-safe and the pattern never changes. The session id is included because it is the value the logout in Session 9 destroys; comparing it before and after logout is a quick viva demonstration.

Question 45

Problem Statement

■ Write in lab record

Add CSRF functionality in the authentication.

Solution

■ Write in lab record

Steps

1. Nothing to add: `CsrfFilter` is part of the chain unless `.csrf(csrf → csrf.disable())` is written. The work is to carry the token on every state-changing form and to prove the filter rejects requests without it.
2. `login.html` (Question 41) uses a plain `action="/login"` with an explicit hidden input built from `_csrf.parameterName` and `_csrf.token`; `register.html` and the logout form use `th:action`, which inserts the same field automatically. View source on both pages and find the hidden input.
3. `dashboard.html` carries two `<meta>` tags with the token and header name for JavaScript callers.
4. Run the curl tests below: a POST without the token gets 403, the same POST with the token gets 302.
5. The API is excluded with `ignoringRequestMatchers("/api/**", "/xml/**", "/actuator/**")` in `SecurityConfig`; keep that line and be ready to explain it.

Program

The pieces, all in files already listed:

HTML

```

1 <!-- login.html: token by hand -->
2 <form action="/login" method="post">
3   <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.to
4
5 <!-- register.html and the logout form: th:action adds the same hidden input
6 <form th:action="@{/register}" th:object="${form}" method="post">
7
8 <!-- dashboard.html: for fetch() calls -->
9 <meta name="_csrf" th:content="${_csrf.token}">
10 <meta name="_csrf_header" th:content="${_csrf.headerName}">

```

JAVA

```

1 // SecurityConfig: CSRF on for browser forms, off for the token-less JSON AP
2 .csrf(csrf → csrf.ignoringRequestMatchers("/api/**", "/xml/**", "/actuator/

```

Output

Checked by reading. The rendered login form contains:

```

1 <input type="hidden" name="_csrf" value="mK4Zr9tQ- ... -64 url-safe characters

```

Without a token the login POST is rejected before the credentials are even read:

```
1 curl -i -X POST http://localhost:8080/login -d "username=admin" -d "password=admin123" -d "csrf_token="
```

```
1 HTTP/1.1 403
```

With the token from a fresh GET, using the same cookie jar (the token is bound to the session):

```
1 TOKEN=$(curl -s -c jar.txt http://localhost:8080/login | grep -o 'name=_csrf=' | sed 's/.*=//')
2 curl -i -b jar.txt -c jar.txt -X POST http://localhost:8080/login \
3   -d "username=admin" -d "password=admin123" -d "_csrf=$TOKEN"
```

```
1 HTTP/1.1 302
2 Location: http://localhost:8080/dashboard
```

The excluded API still accepts writes with Basic auth and no token:

```
1 curl -s -o /dev/null -w "%{http_code}\n" -u admin:admin123 -X DELETE http://localhost:8080/api/1
```

```
1 204
```

Using a token from one session with the cookie of another also gives 403, which is the actual attack the filter stops.

Explanation

A cross-site request forgery is a form on an attacker's page that posts to our site; the browser attaches our session cookie automatically, so the request looks logged in. The defence is a second secret the attacker cannot read: a random token stored in the session and echoed in every form. `CsrfFilter` compares the `_csrf` parameter (or the `X-CSRF-TOKEN` header) with the session's token on every POST, PUT, PATCH and DELETE, and answers 403 on mismatch; GET is never checked because it must not change state. Spring Security 6 also wraps the token per request (`XorCsrfTokenRequestAttributeHandler`), so the value in the page differs each time while

validating to the same secret. The API is excluded because its clients authenticate with HTTP Basic on each call and hold no session cookie an attacker could reuse.

Question 46

Problem Statement

■ Write in lab record

Restrict role-based access to views in Spring Boot Security.

Solution

■ Write in lab record

Steps

1. Add `thymeleaf-extras-springsecurity6` to `pom.xml` and run Maven, Update Project. Boot's parent manages the version.
2. In the templates, declare `xmlns:sec="http://www.thymeleaf.org/extras/spring-security"` on the `html` tag (already in `dashboard.html` and `admin.html`).
3. Use `sec:authorize="hasRole('ADMIN')"` on the elements only admins may see, `sec:authorize="isAuthenticated()"` for any logged-in user, and `sec:authentication="name"` to print the username.
4. Log in as `asha`, then `admin`, and compare the dashboard buttons. Also type `/admin/users` as `asha`: the link is hidden and the URL rule still returns 403.

Program

pom.xml fragment

```
1 <!-- Session 10, Q46: sec:authorize / sec:authentication in Thymeleaf. Add i
2 <dependency>
3   <groupId>org.thymeleaf.extras</groupId>
4   <artifactId>thymeleaf-extras-springsecurity6</artifactId>
5 </dependency>
```

The view rules, from `dashboard.html` and `admin.html` above:

```

1 <a sec:authorize="isAuthenticated()" th:href="@{/api/students}" class="btn btn-primary">Students (JSON)</a>
2 <a sec:authorize="hasRole('ADMIN')" th:href="@{/admin/users}" class="btn btn-danger">Admin: user list</a>
3 <a sec:authorize="hasRole('ADMIN')" th:href="@{/actuator/health}" class="btn btn-secondary">Health</a>
4 <p sec:authorize="!hasRole('ADMIN')" class="text-muted mt-2">Admin functions</p>
5
6 <span sec:authentication="name">admin</span>

```

Output

Checked by reading. Rendered HTML of the button block for `admin`:

```

1 <a href="/api/students" class="btn btn-primary">Students (JSON)</a>
2 <a href="/admin/users" class="btn btn-danger">Admin: user list</a>
3 <a href="/actuator/health" class="btn btn-secondary">Health</a>

```

For `asha`, the two admin anchors are absent from the page source, not just hidden by CSS:

```

1 <a href="/api/students" class="btn btn-primary">Students (JSON)</a>
2 <p class="text-muted mt-2">Admin functions are hidden for your role.</p>

```

And `asha` typing `/admin/users` by hand still gets the 403 from Question 43.

Explanation

`sec:authorize` evaluates the same SpEL expressions as `authorizeHttpRequests` (`hasRole`, `hasAnyRole`, `isAuthenticated`, `!`), against the `Authentication` in the current request. When the expression is false, Thymeleaf removes the element from the output entirely, so nothing leaks into view source. This is a convenience layer, not the security boundary: the URL rule in `SecurityConfig` is what actually protects `/admin/users`. Both are needed. Without the view rule, students see links that fail; without the URL rule, anyone who guesses the address gets in.

`sec:authentication="name"` reads a property of the `Authentication` object, the same value the controller put in the model as `username`.

Viva Questions

× Do not copy. Read for understanding and the viva

- **Q:** Why is User to Role many-to-many and what table does JPA create for it? **A:** A user has several roles and a role has many users; `user_roles(user_id, role_id)` with a composite key.
- **Q:** What is the difference between `hasRole("ADMIN")` and `hasAuthority("ROLE_ADMIN")`? **A:** None in effect; `hasRole` adds the `ROLE_` prefix before comparing.
- **Q:** Why are the roles fetched eagerly? **A:** They are needed at every login and the session is closed once `loadUserByUsername` returns; a lazy set would throw `LazyInitializationException`.
- **Q:** Why does a logged-in student get 403 on `/admin/users` while an anonymous visitor gets a redirect? **A:** `ExceptionTranslationFilter` sends unauthenticated users to log in; authenticated users who fail authorisation get Access Denied.
- **Q:** How does the controller obtain the current user without touching `SecurityContextHolder`? **A:** Spring MVC injects `Authentication` (or `@AuthenticationPrincipal`) as a handler argument.
- **Q:** Which HTTP methods does `CsrfFilter` check? **A:** POST, PUT, PATCH and DELETE; GET, HEAD, OPTIONS and TRACE are exempt.
- **Q:** Why exclude `/api/**` from CSRF? **A:** Its clients use HTTP Basic per request and carry no session cookie, so the forgery scenario does not apply.
- **Q:** Is `sec:authorize` enough to protect a page? **A:** No. It only hides markup; the URL rule in the `SecurityFilterChain` is the real check.

Common Mistakes

× Do not copy. Read for understanding and the viva

- Storing `ADMIN` in the role table and calling `hasRole("ADMIN")`; the authority must be `ROLE_ADMIN` or `hasAuthority` must be used.
- Putting `anyRequest().authenticated()` before the `/admin/**` rule; the first match wins and the admin rule is never reached (Spring 6 refuses to start when a pattern follows `anyRequest()`).
- Disabling CSRF globally to make one curl call work, instead of excluding just the JSON paths.
- Reading the client IP behind a proxy with `getRemoteAddr()` and reporting the proxy's address.
- Forgetting `thymeleaf-extras-springsecurity6`; `sec:` attributes are silently ignored and every user sees every link.
- Using `sec:authorize` alone and skipping the URL rule, so the admin page is reachable by typing its address.

Session Summary

■ Write in lab record

- Bootstrap `login.html` and `register.html` with the CDN link and the `is-invalid` / `invalid-feedback` validation display
- `Role` , `RoleRepository` , the many-to-many `User` , the seeder with two roles and two users, the `roles` and `user_roles` DDL and the join-query result
- `SecurityConfig` with the role rules, `admin.html` , and the access matrix with one curl per cell
- `dashboard.html` showing username, roles, client IP, date and time and session id, with sample renderings for admin and student
- The CSRF hidden field, meta tags, the 403-then-302 curl transcript and the API exclusion
- `sec:authorize` view rules and the rendered button block for each role

[✎ Edit this page](#)

[< Previous](#)
Session 9

Section 1 · Computer Networks Lab (NS-3) [Next >](#)