Lab 11-12. Authentication and authorization (Spring Security foundations). Secure data layer and access control basics.

Objective: Implement authentication using Spring Security: session-based (MVC) or JWT-based (REST). Add user login, role-based access, session or token-based auth. Build a secure database-backed feature with full CRUD, validation, and access control preventing access to other users' data.

Tasks:

Task 1. Integrate Spring Security, implement login and registration, store sessions or generate JWT tokens, and restrict access to protected routes using roles.

Part A - Session-based Auth (MVC track)

1. Implement custom UserDetailsService.
2. Configure SecurityFilterChain to require authentication for all pages except /login, /register.
3. Implement login/logout with Spring Security session handling.
4. Add CSRF protection (mandatory).
5. Add authorization rules, e.g.:

   /admin/** --> ROLE_ADMIN

   /user/** --> ROLE_USER

Part B - Token-based Auth (REST track)

1. Implement /auth/login returning JWT.
2. Store JWT in **HTTP-only cookie** or **Authorization: Bearer**.
3. Build JwtAuthFilter.
4. Protect endpoints with @PreAuthorize.
5. Use **SecurityContextHolder** to identify the authenticated user.

Task 2. Create a new entity (e.g., Note, Task, Project, Contact). Build everything for a new enrity: Flyway migration --> Entity --> Repository --> Service --> Controller.

Use security rules:

- Users may only access their own data.
- Data access must use the authenticated user's ID.
- Validate all fields.
- Use prepared statements for at least one query.

Database Layer

- Add Flyway migration for entity.
- Implement repository using JDBC Template **OR** Spring Data JPA (your choice).

- Require at least one raw SQL query with **prepared statements**.

Service Layer with Security Controls

- CRUD operations *must* enforce access rules: Users can only access/update/delete their own data.
- Use user_id foreign key in DB.

Strong Password Security

- Use bcrypt with strength parameter.
- Create password policy: length, charset, no common passwords.

Model Validation

- Add validation in DTO, in entity, or service layer.
- Prevent common attacks: SQL injection, mass assignment, invalid types.

Error Handling (optional)

- Add a **GlobalExceptionHandler (@ControllerAdvice)** returning safe JSON/HTML.

MVC vs REST

- **MVC:** return Thymeleaf pages, handle form validation errors.
- **REST:** return JSON error details.

<u>Checklist:</u>

Security Configuration

- Created custom UserDetailsService.
- Implemented SecurityFilterChain.
- Configured which routes are public/private.
- Added password hashing (bcrypt).

Authentication

MVC (session-based)

- Implemented login form + handler.
- Implemented logout.
- CSRF enabled and working.
- Added session timeout configuration.

REST (JWT)

- /auth/login returns a JWT token.
- Implemented custom JWT filter.
- Tokens validated on every request.
- Token stored in: HTTP-only cookie or Authorization: Bearer header

Authorization

- Defined roles (e.g., USER, ADMIN).
- Protected endpoints using: @PreAuthorize or route-based config in HttpSecurity.
- Denied access returns correct status (401/403).

Error Handling

- Invalid login returns safe error message. No stack traces in responses.

Database layer

- Created Flyway migration V<number>__create_<entity>.sql.
- Entity table includes user_id foreign key.
- Implemented JPA repository or JDBC Template.
- At least one prepared-statement raw SQL (if JDBC template).

Entity and validation

- Created entity class with fields + constraints.
- DTO for input with validation annotations.
- Prevented mass assignment (no direct mapping from DTO → entity without whitelisting).

Service layer

- CRUD operations implemented.
- Access control implemented: only owners can read/update/delete; proper "404" returned if accessing someone else's resource.

Security and error handling

- No SQL injection is possible.
- Global exception handler returns safe, non-sensitive messages (optional).
- Handled non-existent resources.

Controller

- CRUD routes implemented.
- Proper HTTP status codes used.
- MVC: Thymeleaf templates for CRUD interaction.
- REST: JSON controllers for CRUD.