

Session 8

REST API, Spring Security and Actuator

Actuator exposes health and metrics endpoints for operations. The session builds full CRUD REST APIs twice, once with annotations and once with XML configuration, then puts Spring Security's default login in front of them.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 33 to 36 of the manual: rest api, spring security and actuator
- 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
Q33	Add actuator dependency in Spring boot and Test Actuator (all available Endpoints)	Complete
Q34	Create REST API for CRUD operation using Spring Boot and Hibernate/JPA using annotation	Complete
Q35	Create a REST API for CRUD operation using Spring Boot and Hibernate/JPA using XML...	Complete
Q36	Add Spring Security dependency in the existing spring boot application created in...	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- Actuator endpoints: `/actuator/health`, `/actuator/info`, `/actuator/metrics`; expose them with `management.endpoints.web.exposure.include=*` for the lab.
- REST CRUD maps HTTP verbs: POST create, GET read, PUT update, DELETE delete. Test each with curl or Postman and keep the request and response in the record.
- Adding spring-boot-starter-security alone protects every URL with a generated password printed in the console.

This session continues the `admission-api` project from Session 7. Listings marked "replaces" overwrite the Session 7 file of the same name. Nothing was executed here; every listing and every expected response was checked by reading against Spring Boot 3.3.

Question 33

Problem Statement

■ Write in lab record

Add actuator dependency in Spring boot and Test Actuator (all available Endpoints).

Solution

■ Write in lab record

Steps

1. Confirm `spring-boot-starter-actuator` is in `pom.xml` (added in Question 29). If not, paste the block below inside `<dependencies>` and run Maven, Update Project.
2. Replace `application.properties` with the listing: it keeps the Session 7 settings and adds the `management.*` and `info.*` keys.
3. Restart. Open `http://localhost:8080/actuator` in a browser; the JSON lists every exposed endpoint with its URL.
4. Call each endpoint with curl and paste the responses into the record.

Configuration

The dependency block, for reference:

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-actuator</artifactId>
4 </dependency>
```

XML

application.properties (replaces)

```
1 # src/main/resources/application.properties (Session 8: Session 7 settings + Actuator, Q33)
2 spring.application.name=admission-api
3 server.port=8080
4
5 spring.datasource.url=jdbc:mysql://localhost:3306/admission_db?createDatabaseIfNotExist=true&serverT
6 spring.datasource.username=root
7 spring.datasource.password=root
8 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
9 spring.jpa.hibernate.ddl-auto=update
10 spring.jpa.show-sql=true
11 spring.jpa.defer-datasource-initialization=true
12 spring.sql.init.mode=always
13
14 # --- Actuator (Q33) ---
15 # By default only /actuator/health is exposed over HTTP. Expose every endpoint for the lab.
16 management.endpoints.web.exposure.include=*
17 # show the db and diskSpace components, not just {"status":"UP"}
18 management.endpoint.health.show-details=always
19 # allow shutdown via POST /actuator/shutdown (disabled by default)
20 management.endpoint.shutdown.enabled=true
21 # /actuator/info reads info.* properties only when env info is enabled
22 management.info.env.enabled=true
23 info.app.name=admission-api
24 info.app.course=MCSSL-222 Web Technologies Lab
25 info.app.version=0.0.1
```

Output

Checked by reading. The discovery page lists the endpoints this project exposes (Boot 3.3, no Prometheus, no Flyway, no logging.file):

```
1 curl -s http://localhost:8080/actuator
```

BASH

```
1 {"_links":{"self":{"href":"http://localhost:8080/actuator","templated":false},
2 "beans":{"href":"http://localhost:8080/actuator/beans","templated":false},
3 "health":{"href":"http://localhost:8080/actuator/health","templated":false},
4 "health-path":{"href":"http://localhost:8080/actuator/health/{*path}","templated":true},
5 "info":{"href":"http://localhost:8080/actuator/info","templated":false},
6 "conditions":{"href":"http://localhost:8080/actuator/conditions","templated":false},
7 "configprops":{"href":"http://localhost:8080/actuator/configprops","templated":false},
8 "env":{"href":"http://localhost:8080/actuator/env","templated":false},
9 "loggers":{"href":"http://localhost:8080/actuator/loggers","templated":false},
10 "heapdump":{"href":"http://localhost:8080/actuator/heapdump","templated":false},
11 "threaddump":{"href":"http://localhost:8080/actuator/threaddump","templated":false},
12 "metrics":{"href":"http://localhost:8080/actuator/metrics","templated":false},
13 "sbom":{"href":"http://localhost:8080/actuator/sbom","templated":false},
14 "scheduledtasks":{"href":"http://localhost:8080/actuator/scheduledtasks","templated":false},
15 "mappings":{"href":"http://localhost:8080/actuator/mappings","templated":false},
16 "shutdown":{"href":"http://localhost:8080/actuator/shutdown","templated":false}}}
```

Each endpoint, what it returns, and the command used:

Endpoint	Command	Expected response (shortened)
health	<code>curl -s localhost:8080/actuator/health</code>	<code>"status": "UP"</code> with <code>db</code> (MySQL, <code>isValid()</code>), <code>diskSpace</code> (total, free, threshold) and <code>ping</code> components
info	<code>curl -s localhost:8080/actuator/info</code>	<code>"app":</code> with name, course and version from the <code>info.*</code> properties
metrics	<code>curl -s localhost:8080/actuator/metrics</code>	<code>"names": [...]</code> about 60 metric names: <code>jvm.memory.used</code> , <code>http.server.requests</code> , <code>hikaricp.connections.active</code> , <code>system.cpu.usage</code> , ...
metrics by name	<code>curl -s localhost:8080/actuator/metrics/jvm.memory.used</code>	<code>"measurements": [</code> with <code>"statistic": "VALUE", "value": 1.2e8</code> and <code>availableTags</code> for <code>area</code> and <code>id</code>
env	<code>curl -s localhost:8080/actuator/env/server.port</code>	<code>"property":</code> with <code>"value": "8080"</code> and source <code>Config resource 'class path resource [application.properties]'</code>
beans	<code>curl -s localhost:8080/actuator/beans</code>	Every bean in the context with its scope, type and dependencies, including <code>studentRestController</code> and <code>studentRepository</code>
mappings	<code>curl -s localhost:8080/actuator/mappings</code>	<code>dispatcherServlets</code> with a handler entry per method, for example <code>GET /api/students/{id}</code>
loggers	<code>curl -s localhost:8080/actuator/loggers/in.ignou</code>	<code>"configuredLevel": null, "effectiveLevel": "INFO"</code>
configprops	<code>curl -s localhost:8080/actuator/configprops</code>	Bound <code>@ConfigurationProperties</code> such as <code>spring.datasource</code> (password masked as <code>*****</code>)
conditions	<code>curl -s localhost:8080/actuator/conditions</code>	<code>positiveMatches</code> and <code>negativeMatches</code> of every auto-configuration
threaddump	<code>curl -s localhost:8080/actuator/threaddump</code>	Array of threads with name, state and stack
heapdump	<code>curl -o heap.hprof localhost:8080/actuator/heapdump</code>	Binary <code>.hprof</code> file
scheduledtasks, sbom	<code>curl -s localhost:8080/actuator/scheduledtasks</code>	Empty lists (nothing scheduled; no SBOM file in the jar). <code>caches</code> is absent because no <code>CacheManager</code> bean exists
shutdown	<code>curl -s -X POST localhost:8080/actuator/shutdown</code>	<code>"message": "Shutting down, bye..."</code> and the process exits

Two full responses worth copying:

```
1 curl -s http://localhost:8080/actuator/health
```

BASH

```
1 {"status":"UP","components":{"
2   "db":{"status":"UP","details":{"database":"MySQL","validationQuery":"isValid()"}},
3   "diskSpace":{"status":"UP","details":{"total":499963174912,"free":201524125696,"threshold":10485760
4   "ping":{"status":"UP"}}}}
```

```
1 curl -s http://localhost:8080/actuator/info
```

BASH

```
1 {"app":{"name":"admission-api","course":"MCSL-222 Web Technologies Lab","version":"0.0.1"}}
```

Change a log level without restarting:

```
1 curl -s -X POST -H "Content-Type: application/json" -d '{"configuredLevel":"DEBUG"}' \
2   http://localhost:8080/actuator/loggers/in.ignou.admission
```

BASH

Response: HTTP 204 , then the console starts printing DEBUG lines from the project package.

Explanation

Actuator registers each endpoint as a bean; the `management.endpoints.web.exposure.include=*` line maps all of them under `/actuator`. Without that line only `health` is reachable over HTTP, which is the safe default because `env`, `beans` and `heapdump` leak configuration and memory. `show-details=always` makes `health` include the `db` component, which runs `Connection.isValid()` on a pooled connection; that is the check operations teams poll. `info` is empty in Boot 3 until `management.info.env.enabled=true` and some `info.*` keys exist. `shutdown` is the only endpoint disabled by default; it needs `management.endpoint.shutdown.enabled=true` and a POST. After Session 9 the whole `/actuator` tree sits behind login except `/actuator/health`.

Question 34

Problem Statement

■ Write in lab record

Create REST API for CRUD operation using Spring Boot and Hibernate/JPA using annotation.

Solution

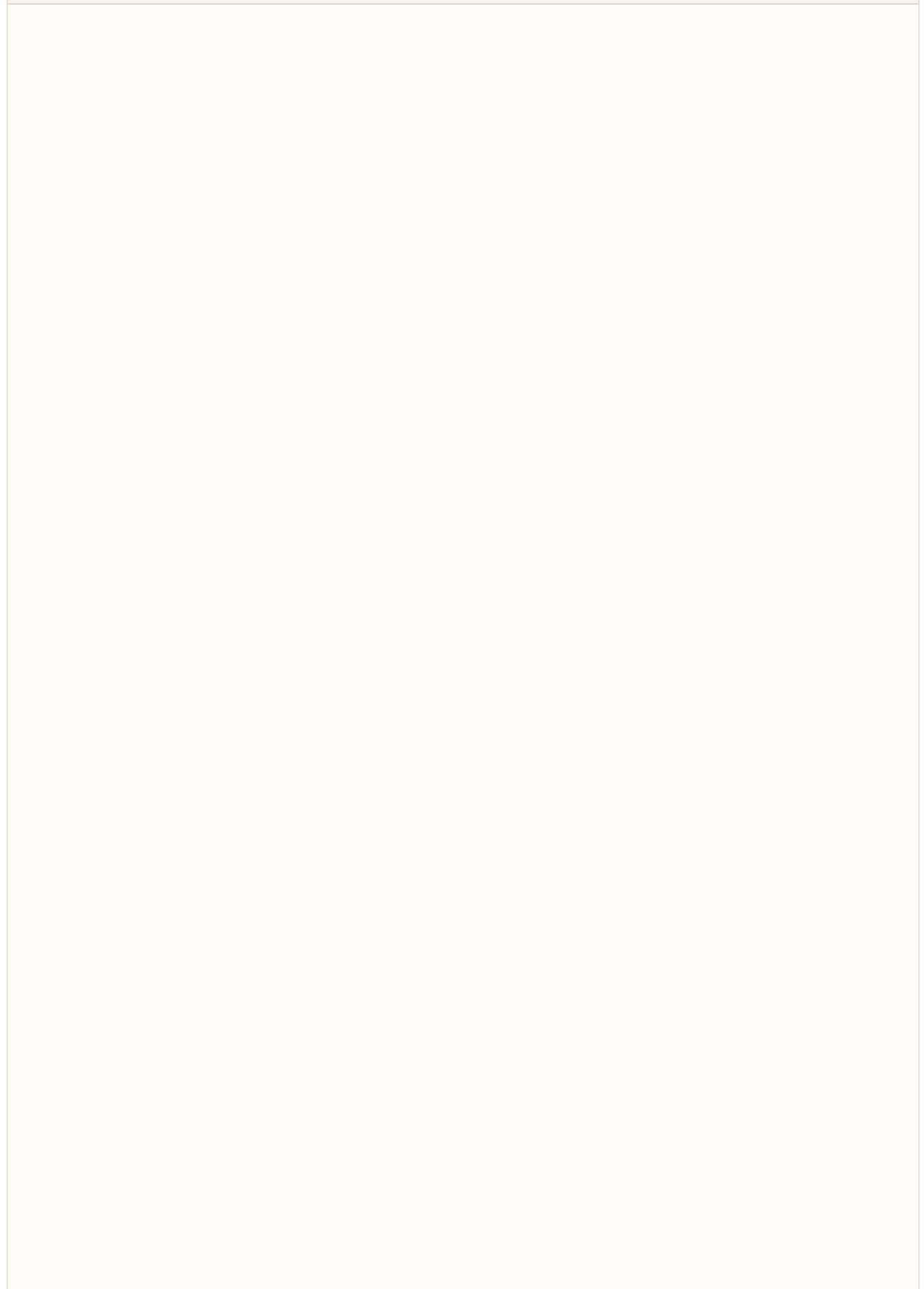
■ Write in lab record

Steps

1. Replace `StudentRestController.java` from Session 7 with the listing below; it keeps the two GET methods and adds POST, PUT and DELETE.

2. Restart with DevTools (saving the file is enough).
3. Run the five curl calls in order; the ids in the later calls come from the POST response.
4. Verify in MySQL after each step: `SELECT * FROM student;` .

Program

StudentRestController.java (replaces)

```
1 // src/main/java/in/ignou/admission/web/StudentRestController.java (Session 8, Q34: full CRUD)
2 package in.ignou.admission.web;
3
4 import java.util.List;
5
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.web.bind.annotation.DeleteMapping;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.PathVariable;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import org.springframework.web.bind.annotation.PutMapping;
13 import org.springframework.web.bind.annotation.RequestBody;
14 import org.springframework.web.bind.annotation.RequestMapping;
15 import org.springframework.web.bind.annotation.RequestParam;
16 import org.springframework.web.bind.annotation.ResponseStatus;
17 import org.springframework.web.bind.annotation.RestController;
18
19 import in.ignou.admission.entity.Student;
20 import in.ignou.admission.repository.StudentRepository;
21
22 /**
23  * Annotation-based REST CRUD.
24  * POST    /api/students      create → 201 + saved student
25  * GET     /api/students      read   → 200 + list
26  * GET     /api/students/{id} read   → 200 or 404
27  * PUT     /api/students/{id} update → 200 or 404
28  * DELETE  /api/students/{id} delete → 204 or 404
29  */
30 @RestController
31 @RequestMapping("/api/students")
32 public class StudentRestController {
33
34     private final StudentRepository students;
35
36     public StudentRestController(StudentRepository students) {
37         this.students = students;
38     }
39
40     /** @RequestBody: Jackson turns the JSON body into a Student; id is ignored and generated. */
41     @PostMapping
42     @ResponseStatus(HttpStatus.CREATED)
43     public Student create(@RequestBody Student student) {
44         student.setId(null);
45         return students.save(student);
46     }
47
48     @GetMapping
49     public List<Student> all(@RequestParam(required = false) String city) {
50         return city == null ? students.findAll() : students.findByCityIgnoreCase(city);
51     }
52
53     @GetMapping("/{id}")
54     public ResponseEntity<Student> one(@PathVariable Long id) {
55         return students.findById(id)
56             .map(ResponseEntity::ok)
```

```
57         .orElse(ResponseEntity.notFound().build());
58     }
59
60     /** Copy the editable fields onto the managed entity, then save (an UPDATE, because id is set).
61     @PutMapping("/{id}")
62     public ResponseEntity<Student> update(@PathVariable Long id, @RequestBody Student in) {
63         return students.findById(id).map(s -> {
64             s.setName(in.getName());
65             s.setEmail(in.getEmail());
66             s.setPhone(in.getPhone());
67             s.setCity(in.getCity());
68             s.setDateOfBirth(in.getDateOfBirth());
69             return ResponseEntity.ok(students.save(s));
70         }).orElse(ResponseEntity.notFound().build());
71     }
72
73     @DeleteMapping("/{id}")
74     public ResponseEntity<Void> delete(@PathVariable Long id) {
75         if (!students.existsById(id)) {
76             return ResponseEntity.notFound().build();
77         }
78         students.deleteById(id);
79         return ResponseEntity.noContent().build();
80     }
81 }
```

Output

Checked by reading. Create:

```
1 curl -i -X POST http://localhost:8080/api/students \
2 -H "Content-Type: application/json" \
3 -d '{"name":"Meera Nair","email":"meera@example.com","phone":"9000011111","city":"Kochi","dateOfBi
```

BASH

```
1 HTTP/1.1 201
2 Content-Type: application/json
3
4 {"id":3,"name":"Meera Nair","email":"meera@example.com","phone":"9000011111","city":"Kochi","dateOfB
```

Read all and read one:

```
1 curl -s http://localhost:8080/api/students
2 curl -s http://localhost:8080/api/students/3
```

BASH

```
1 [{"id":1,"name":"Asha Verma",...},{id":2,"name":"Ravi Kumar",...},{id":3,"name":"Meera Nair",...}]
2 {"id":3,"name":"Meera Nair","email":"meera@example.com","phone":"9000011111","city":"Kochi","dateOfB
```

Update (full replacement of the editable fields):

```
1 curl -s -X PUT http://localhost:8080/api/students/3 \  
2 -H "Content-Type: application/json" \  
3 -d '{"name":"Meera Nair","email":"meera.nair@example.com","phone":"9000011111","city":"Thrissur","
```

BASH

```
1 {"id":3,"name":"Meera Nair","email":"meera.nair@example.com","phone":"9000011111","city":"Thrissur",
```

Delete, then read again:

```
1 curl -i -X DELETE http://localhost:8080/api/students/3  
2 curl -i http://localhost:8080/api/students/3
```

BASH

```
1 HTTP/1.1 204  
2  
3 HTTP/1.1 404
```

Error cases: a duplicate email on POST violates the unique key and returns `HTTP 500` with `DataIntegrityViolationException` in the console; a body that is not JSON returns `HTTP 400` (`HttpMessageNotReadableException`).

Explanation

Each HTTP verb has its own annotation, all shortcuts for `@RequestMapping(method = ...)`. `@RequestBody` asks Jackson to build a `Student` from the request JSON; `student.setId(null)` guarantees an INSERT even if the client sent an id. `@ResponseStatus(CREATED)` sets 201 for the create path where the body is always present. PUT loads the managed entity, copies the fields, and calls `save`; because the entity has an id, Hibernate issues an UPDATE. DELETE checks `existsById` first so a missing id is 404 rather than a silent 204. Every method goes through the same `JpaRepository`, so the controller has no SQL and no session handling of its own; Boot's `OpenEntityManagerInViewInterceptor` and the repository's `@Transactional` methods do that.

Question 35

Problem Statement

■ Write in lab record

Create a REST API for CRUD operation using Spring Boot and Hibernate/JPA using XML configuration.

Solution

■ Write in lab record

Steps

1. Create a new package `in.ignou.xmlapi` beside (not inside) `in.ignou.admission`. Component scanning does not reach it, so nothing in it becomes a bean unless XML says so.
2. Add `CourseService` (a plain class) and `CourseRestController` to that package.
3. Create `src/main/resources/beans.xml` with the two `<bean>` definitions.

4. Replace `AdmissionApiApplication.java` with the version that carries `@ImportResource("classpath:beans.xml")`.
5. Restart and test `/xml/courses` with the curl commands.
6. Prove the wiring is XML: comment out the `courseRestController` bean in `beans.xml`, restart, and `GET /xml/courses` returns 404.

Program

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

`CourseService.java`

CourseService.java

```
1 // src/main/java/in/ignou/xmlapi/CourseService.java (Session 8, Q35)
2 // NOTE the package: in.ignou.xmlapi is OUTSIDE in.ignou.admission, so component scanning
3 // never sees these classes. Only beans.xml creates them.
4 package in.ignou.xmlapi;
5
6 import java.util.List;
7 import java.util.Optional;
8
9 import in.ignou.admission.entity.Course;
10 import in.ignou.admission.repository.CourseRepository;
11
12 /** Plain class: no @Service, no @Autowired. beans.xml supplies the repository through the construct
13 public class CourseService {
14
15     private final CourseRepository courses;
16
17     public CourseService(CourseRepository courses) {
18         this.courses = courses;
19     }
20
21     public Course create(Course c) {
22         c.setId(null);
23         return courses.save(c);
24     }
25
26     public List<Course> all() {
27         return courses.findAll();
28     }
29
30     public Optional<Course> find(Long id) {
31         return courses.findById(id);
32     }
33
34     public Optional<Course> update(Long id, Course in) {
35         return courses.findById(id).map(c -> {
36             c.setCode(in.getCode());
37             c.setTitle(in.getTitle());
38             c.setCredits(in.getCredits());
39             c.setProgramme(in.getProgramme());
40             return courses.save(c);
41         });
42     }
43
44     public boolean delete(Long id) {
45         if (!courses.existsById(id)) {
46             return false;
47         }
48         courses.deleteById(id);
49         return true;
50     }
51 }
```

Output

Checked by reading. A course carries its programme, so the JSON nests it:

```
1 curl -s http://localhost:8080/xml/courses
```

BASH

```
1 [{"id":1,"code":"MCS-221","title":"Data Warehousing and Data Mining","credits":4,"programme":{"id":1
2 {"id":2,"code":"MCSL-222","title":"Web Technologies Lab","credits":2,"programme":{"id":1,"code":"MC
3 {"id":3,"code":"BCS-011","title":"Computer Basics and PC Software","credits":3,"programme":{"id":2,
```

Create (only the programme id is needed; Hibernate resolves the reference):

```
1 curl -i -X POST http://localhost:8080/xml/courses -H "Content-Type: application/json" \
2 -d '{"code":"MCS-224","title":"Artificial Intelligence and Machine Learning","credits":4,"programm
```

BASH

```
1 HTTP/1.1 201
2
3 {"id":4,"code":"MCS-224","title":"Artificial Intelligence and Machine Learning","credits":4,"program
```

The nested programme shows nulls in the POST response because the request only carried the id; `GET /xml/courses/4` right after returns the full programme.

Read one, update, delete:

```
1 curl -s http://localhost:8080/xml/courses/4
2 curl -s -X PUT http://localhost:8080/xml/courses/4 -H "Content-Type: application/json" \
3   -d '{"code":"MCS-224","title":"AI and Machine Learning","credits":4,"programme":{"id":1}}'
4 curl -i -X DELETE http://localhost:8080/xml/courses/4
```

BASH

```
1 {"id":4,"code":"MCS-224","title":"Artificial Intelligence and Machine Learning","credits":4,"program
2 {"id":4,"code":"MCS-224","title":"AI and Machine Learning","credits":4,"programme":{"id":1,"code":nu
3 HTTP/1.1 204
```

Start-up evidence that the XML was read:

```
1 Loading XML bean definitions from class path resource [beans.xml]
```

Explanation

Annotation configuration and XML configuration produce the same thing, bean definitions in one `ApplicationContext`.

`@ImportResource` reads `beans.xml` into the Boot context, so an XML bean can reference `courseRepository`, which Spring Data registered from the interface. Constructor injection in XML is `<constructor-arg ref="..." />`; the classes have no `@Service`, `@Component` or `@Autowired`. The controller keeps `@RestController` and `@GetMapping` for one reason: since Spring 6, `RequestMappingHandlerMapping` accepts a bean as a handler only when its class is annotated with `@Controller`, and the URL-to-method table can only be declared with mapping annotations. What XML controls is the bean's existence and its dependencies, which is exactly why step 6 makes the endpoint disappear. Putting the classes in `in.ignou.xmlapi` avoids a `BeanDefinitionOverrideException`: if they sat under `in.ignou.admission`, scanning and XML would both try to register `courseRestController`.

Question 36

Problem Statement

■ Write in lab record

Add Spring Security dependency in the existing spring boot application created in Exercise 2 of Session 8 and configure it as the default login.

Solution

■ Write in lab record

Steps

1. Paste the dependency below into `pom.xml` and run Maven, Update Project.
2. Restart. Copy the generated password from the console line `Using generated security password: .`
3. Open `http://localhost:8080/api/students` in the browser. Spring redirects to `/login`, its built-in page with Username, Password and a Sign in button.
4. Log in as `user` with the generated password; the browser is redirected back to `/api/students` and the JSON appears.
5. Test from curl with HTTP Basic (`-u user:password`).
6. Optional: pin the credentials in `application.properties` so they survive restarts (`spring.security.user.name=admin` , `spring.security.user.password=admin123`).

Configuration

pom.xml fragment

```
1 <!-- Session 8, Q36: add inside <dependencies> of pom.xml, then Maven > Update Project -->
2 <dependency>
3   <groupId>org.springframework.boot</groupId>
4   <artifactId>spring-boot-starter-security</artifactId>
5 </dependency>
```

Output

Checked by reading. Console at start-up (the UUID changes every run):

```
1 Using generated security password: 6a1b7f5c-9d2e-4c3a-8b0f-1e2d3c4b5a69
2
3 This generated password is for development use only. Your security configuration must be updated before
```

Browser: any URL now shows Spring's default login page, a centred form titled "Please sign in" with Username, Password and a blue "Sign in" button. A wrong password shows "Bad credentials" above the form. After login, the original URL loads.

curl without credentials, then with:

```
1 curl -i http://localhost:8080/api/students
```

BASH

```
1 HTTP/1.1 401
2 WWW-Authenticate: Basic realm="Realm"
3 Set-Cookie: JSESSIONID=...; Path=/; HttpOnly
```

```
1 curl -s -u user:6a1b7f5c-9d2e-4c3a-8b0f-1e2d3c4b5a69 http://localhost:8080/api/students
```

BASH

```
1 [{"id":1,"name":"Asha Verma", ...},{ "id":2,"name":"Ravi Kumar", ...}]
```

Writes are blocked even with the right password, because the default chain also turns on CSRF:

```
1 curl -i -u user:6a1b7f5c-9d2e-4c3a-8b0f-1e2d3c4b5a69 -X DELETE http://localhost:8080/api/students/2
```

BASH

```
1 HTTP/1.1 403
```

Session 9 fixes that by excluding `/api/**` from CSRF in the custom `SecurityFilterChain`.

Explanation

Adding the starter triggers `SecurityAutoConfiguration`. With no `SecurityFilterChain` bean of our own, Boot installs its default: every request must be authenticated, form login at `/login` and HTTP Basic are both enabled, CSRF protection is on, and an `InMemoryUserDetailsManager` holds one user named `user` with a random UUID password printed once. Actuator endpoints are covered too, which is why `/actuator/health` also asks for a login now. The `WWW-Authenticate: Basic` header on the 401 is what lets curl's `-u` work; browsers get a 302 to `/login` instead because Spring detects the `Accept: text/html` header. The point of the exercise is to see the whole surface locked down with zero code; Session 9 replaces each default with a database-backed one.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** Which Actuator endpoint is exposed over HTTP by default and why only that one? **A:** `health`; the others (`env`, `beans`, `heapdump`) reveal configuration and memory, so they are opt-in.
- **Q:** What does `management.endpoint.health.show-details=always` change? **A:** The response includes the `db`, `diskSpace` and `ping` components instead of a bare status.
- **Q:** Which HTTP status should POST, DELETE and a missing id return? **A:** 201 Created, 204 No Content, 404 Not Found.
- **Q:** How does `save` decide between INSERT and UPDATE? **A:** If the entity id is null it persists (INSERT); otherwise it merges (UPDATE).
- **Q:** Why does the XML-configured controller still carry `@RestController`? **A:** Spring MVC 6 only treats `@Controller`-annotated beans as handlers; XML supplies the bean and its dependencies, not the URL mappings.
- **Q:** What would happen if `CourseService` were also annotated `@Service` inside the scanned package? **A:** Two definitions of the same bean name; Boot refuses to start with `BeanDefinitionOverrideException`.
- **Q:** Where does the generated security password come from? **A:** `UserDetailsServiceAutoConfiguration` creates an in-memory user `user` with a random UUID when no `UserDetailsService` bean exists.

- **Q:** Why does curl get 401 but the browser gets a redirect to `/login` ? **A:** The default chain enables both HTTP Basic and form login and picks the response by the request's `Accept` header.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Testing `/actuator/metrics` without the exposure property and reporting "Actuator is broken" when the 404 only means the endpoint is not exposed.
- Sending JSON without `Content-Type: application/json` ; Spring answers 415 Unsupported Media Type.
- Treating PUT as a partial update; a body with a missing field overwrites that column with null.
- Placing the XML-wired classes inside `in.ignou.admission` and getting duplicate-bean errors, or forgetting `@ImportResource` and getting 404 on `/xml/courses` .
- Copying the generated password from an old console; it changes on every restart.
- Expecting `DELETE` through curl to work right after adding the security starter; CSRF blocks it until the chain is customised.

Session Summary

■ Write in lab record

- `application.properties` with Actuator exposure, health details and `info.*` keys, plus the `/actuator` discovery JSON and the endpoint table with one curl per endpoint
- `StudentRestController` with POST, GET, PUT and DELETE and the five curl transcripts (201, 200, 200, 204, 404)
- `CourseService` , `CourseRestController` , `beans.xml` and the `@ImportResource` application class, with the `/xml/courses` transcripts
- The security dependency, the generated-password console line, the default login page description and the 401/403 curl results

🔗 Edit this page

< Previous
Session 7

Next >
Session 9