Session 7

Spring Boot and REST controllers

Spring Boot removes most configuration: an embedded server, auto-configured data source and one property file. REST controllers return JSON instead of views, which is how modern front ends consume the same student data.

Objectives

× Do not copy. Read for understanding and the viva

- Complete questions 29 to 32 of the manual: spring boot and rest controllers
- 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
Q29	Create a Spring Boot application using Spring Initializer. Add the following...	Complete
Q30	Configure Database settings through the property file in Spring Boot	Complete
Q31	Create JPA Repositories for all entities used in the Student Admission lifecycle	Complete
Q32	Create Rest Controller to fetch Student Information using JPA Repository; the response...	Complete

Preparation

× Do not copy. Read for understanding and the viva

- Use Spring Initializr with the listed dependencies; the exact starter names are spring-boot-starter-web, spring-boot-starter-data-jpa, spring-boot-starter-thymeleaf, spring-boot-devtools, spring-boot-starter-actuator and the database driver.
- Database settings go in `application.properties`: url, username, password, `spring.jpa.hibernate.ddl-auto`.
- A `JpaRepository<Student, Long>` interface needs no implementation; Spring generates it.

Sessions 7 to 10 build one project, `admission-api`. Each session adds files to it or replaces files from the session before. Every listing carries its path inside the project as a first-line comment. Nothing in these four sessions was executed here (no JDK, Maven or MySQL on this machine); every listing was checked by reading against the Spring Boot 3.3 and Spring Security 6 APIs.

Question 29

Problem Statement

■ Write in lab record

Create a Spring Boot application using Spring Initializer. Add the following dependencies manually:

1. Spring MVC
2. Hibernate
3. JPA
4. Thymeleaf
5. DevTool
6. Actuator
7. MySQL/MSSQL/Oracle/MongoDB (as per your choice) driver.

Solution

■ Write in lab record

Steps

1. Open start.spring.io in a browser.
2. Fill the left panel: Project `Maven`, Language `Java`, Spring Boot `3.3.4`, Group `in.ignou`, Artifact `admission-api`, Name `admission-api`, Package name `in.ignou.admission`, Packaging `Jar`, Java `17`.
3. Click `Add dependencies` and pick, one by one: `Spring Web`, `Spring Data JPA`, `Thymeleaf`, `Spring Boot DevTools`, `Spring Boot Actuator`, `MySQL Driver`.
4. Click `Generate`; unzip `admission-api.zip` into your workspace.
5. Eclipse: File, Open Projects from File System, Directory, select the `admission-api` folder, Finish. Wait until Maven finishes downloading (bottom-right progress bar).
6. Open `pom.xml` and compare with the listing below. If a dependency is missing, paste its block inside `<dependencies>`, save, then right-click the project, Maven, Update Project.
7. Run once: right-click the project, Run As, Spring Boot App (or `./mvnw spring-boot:run` in a terminal).

Program

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

pom.xml

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- admission-api/pom.xml : generated by Spring Initializr, Spring Boot 3.3
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.3.4</version>
    <relativePath/>
  </parent>

  <groupId>in.ignou</groupId>
  <artifactId>admission-api</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>admission-api</name>
  <description>MCSL-222 Student Admission API (Sessions 7 to 10)</descriptio

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>
    <!-- a. Spring MVC (embedded Tomcat, Jackson for JSON) -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- b. Hibernate and c. JPA: one starter brings hibernate-core and Spri
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <!-- d. Thymeleaf view templates -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!-- e. DevTools: automatic restart on save -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
```

```
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <!-- f. Actuator: /actuator/health, /actuator/metrics ... -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- g. MySQL driver -->
    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Output

Checked by reading, not executed. The first run stops before Tomcat starts because the data source is not configured yet; that is Question 30:

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no em

Reason: Failed to determine a suitable driver class
```

After Question 30 the same run ends with lines of this shape:

```
Tomcat initialized with port 8080 (http)
HikariPool-1 - Start completed.
Tomcat started on port 8080 (http) with context path '/'
Started AdmissionApiApplication in 4.1 seconds (process running for 4.6)
```

Explanation

The seven items in the question map to six Maven artifacts. Spring MVC is `spring-boot-starter-web` (it also brings embedded Tomcat and Jackson). Hibernate and JPA arrive together in `spring-boot-starter-data-jpa`; Hibernate is the JPA implementation, so there is no separate starter. Thymeleaf, DevTools and Actuator are their own starters. The driver is `mysql-connector-j` with `runtime` scope because our code never imports a MySQL class; only JDBC needs it at run time. `spring-boot-starter-parent` fixes every version, so no `<version>` tag appears under the dependencies. `@SpringBootApplication` turns on auto-configuration and scans `in.ignou.admission` and its sub-packages for `@Entity`, `@Repository`, `@Service` and `@Controller` classes.

Question 30

Problem Statement

Write in lab record

Configure Database settings through the property file in Spring Boot.

Solution

Write in lab record

Steps

1. Start MySQL and create the schema: `mysql -u root -p` then `CREATE DATABASE admission_db;`. (The `createDatabaseIfNotExist=true` flag in the URL does the same job if the account may create schemas.)
2. Open `src/main/resources/application.properties` (empty after Initializr) and paste the listing.
3. Change `spring.datasource.username` and `password` to your MySQL account.
4. Run the application; watch the console for `HikariPool-1 - Start completed`.

Configuration

application.properties

```
# src/main/resources/application.properties  (Session 7, Q30)
spring.application.name=admission-api
server.port=8080

# --- MySQL connection ---
# create the schema first:  CREATE DATABASE admission_db;
spring.datasource.url=jdbc:mysql://localhost:3306/admission_db?createDatabas
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# --- JPA / Hibernate ---
# update = create missing tables and columns from the @Entity classes, never
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# --- seed data (src/main/resources/data.sql) ---
# run data.sql after Hibernate has created the tables, on a real (non-embedd
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

Output

Checked by reading. Expected console lines on a successful start:

```
HikariPool-1 - Starting...
HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@5c3b1f2a
HikariPool-1 - Start completed.
HHH000412: Hibernate ORM core version 6.5.3.Final
HHH10001005: Database info:
    Database JDBC URL [jdbc:mysql://localhost:3306/admission_db]
    Database driver: com.mysql.cj.jdbc.Driver
    Database version: 8.0.36
```

A wrong password fails at the pool, not at the query:

```
HikariPool-1 - Exception during pool initialization.
java.sql.SQLException: Access denied for user 'root'@'localhost' (using pass
```

Explanation

Property	Effect
`spring.datasource.url`	JDBC URL; host, port, schema and flags. `serverTimezone` stops the “server time zone value is unrecognized” error.
`spring.datasource.username`, `password`	MySQL account. Boot creates a HikariCP pool from these.
`spring.datasource.driver-class-name`	Optional; Boot infers it from the URL. Kept so the driver name is visible in the record.
`spring.jpa.hibernate.ddl-auto=update`	Hibernate compares entities with tables at start and issues `CREATE TABLE` or `ALTER TABLE ADD` as needed. It never drops columns. Use `validate` in production.
`spring.jpa.show-sql=true`	Prints every SQL statement; useful evidence for the lab record.
`spring.jpa.defer-datasource-initialization=true`	Runs `data.sql` after Hibernate has created the tables. Without it the inserts run first and fail.
`spring.sql.init.mode=always`	Boot runs `data.sql` only for embedded databases by default; `always` enables it for MySQL.

Question 31

Problem Statement

Write in lab record

Create JPA Repositories for all entities used in the Student Admission lifecycle.

Solution

Write in lab record

Steps

1. Create the package `in.ignou.admission.entity` and add `Student`, `Programme`, `Course`, `Admission` and the enum `AdmissionStatus`.
2. Create the package `in.ignou.admission.repository` and add one interface per entity extending `JpaRepository<Entity, Long>`.
3. Run the application. With `ddl-auto=update` and `show-sql=true` the console prints the `create table` statements the first time.
4. Check in MySQL: `USE admission_db; SHOW TABLES;` lists `admission`, `course`, `programme`, `student`.

Program

The life cycle has four entities. A Student applies to a Programme; the application is an Admission whose status moves APPLIED, VERIFIED, APPROVED or REJECTED; a Programme owns Courses.

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

Student.java

Student.java

```java
// src/main/java/in/ignou/admission/entity/Student.java
package in.ignou.admission.entity;

import java.time.LocalDate;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

/** Applicant. Maps to table `student` (Hibernate default: snake_case of the
@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 80)
    private String name;

    @Column(nullable = false, unique = true, length = 120)
    private String email;

    @Column(length = 15)
    private String phone;

    @Column(length = 60)
    private String city;

    private LocalDate dateOfBirth;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
```

```
    public LocalDate getDateOfBirth() { return dateOfBirth; }
    public void setDateOfBirth(LocalDate dateOfBirth) { this.dateOfBirth = d
}
```

One repository per entity:

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

StudentRepository.java

StudentRepository.java

```java
// src/main/java/in/ignou/admission/repository/StudentRepository.java
package in.ignou.admission.repository;

import java.util.List;
import java.util.Optional;

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

import in.ignou.admission.entity.Student;

/** No implementation needed: Spring Data generates it at start-up from the
public interface StudentRepository extends JpaRepository<Student, Long> {

    Optional<Student> findByEmail(String email);

    List<Student> findByCityIgnoreCase(String city);
}
```

Output

Checked by reading. Hibernate 6.5 with the MySQL dialect generates DDL of this shape on the first start (order may differ):

```sql
create table programme (
    duration_years integer not null,
    id bigint not null auto_increment,
    code varchar(10) not null,
    name varchar(100) not null,
    primary key (id)
) engine=InnoDB;
alter table programme add constraint UK_programme_code unique (code);

create table student (
    date_of_birth date,
    id bigint not null auto_increment,
    phone varchar(15),
    city varchar(60),
    name varchar(80) not null,
    email varchar(120) not null,
    primary key (id)
) engine=InnoDB;
alter table student add constraint UK_student_email unique (email);

create table course (
    credits integer not null,
    id bigint not null auto_increment,
    programme_id bigint not null,
    code varchar(10) not null,
    title varchar(120) not null,
    primary key (id)
) engine=InnoDB;
alter table course add constraint FK_course_programme foreign key (programme

create table admission (
    applied_on date,
    id bigint not null auto_increment,
    programme_id bigint not null,
    student_id bigint not null,
    status enum ('APPLIED','VERIFIED','APPROVED','REJECTED') not null,
    primary key (id)
) engine=InnoDB;
```

And the start-up log names the repositories it built:

```
1 Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2 Finished Spring Data repository scanning in 41 ms. Found 4 JPA repository in
```

Explanation

`JpaRepository<T, ID>` already declares `save`, `findById`, `findAll`, `existsById`, `count`, `deleteById` and paging variants. Spring Data creates a proxy class for each interface at start-up, so no implementation is written. Extra methods are derived from their names: `findByCityIgnoreCase(String)` becomes `select s from Student s where upper(s.city) = upper(?1)`; `findByProgrammeCode(String)` walks the `programme` association and compares `programme.code`. A misspelt property name fails at start-up with `No property 'citty' found for type 'Student'`, which is the check that the name is right.

Hibernate default naming converts `dateOfBirth` to `date_of_birth` and the class name `Student` to table `student`. The `@ManyToOne` fields become foreign-key columns named by `@JoinColumn`. `@Enumerated(EnumType.STRING)` stores `APPROVED` as text; without it Hibernate stores the ordinal 2, which breaks the moment someone reorders the enum.

Question 32

Problem Statement

Write in lab record

Create Rest Controller to fetch Student Information using JPA Repository; the response should display in JSON format.

Solution

Write in lab record

Steps

1. Create the package `in.ignou.admission.web` and add `StudentRestController`.
2. Add `data.sql` under `src/main/resources` so the table has rows to fetch (the properties from Question 30 already enable it).

3. Restart the application. Handler mappings are logged only at DEBUG level, so the request test below is the real check.
4. Run the curl commands in a terminal, or open `http://localhost:8080/api/students` in a browser.

Program

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

StudentRestController.java

StudentRestController.java

```java
// src/main/java/in/ignou/admission/web/StudentRestController.java   (Sessio
package in.ignou.admission.web;

import java.util.List;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import in.ignou.admission.entity.Student;
import in.ignou.admission.repository.StudentRepository;

/**
 * @RestController = @Controller + @ResponseBody: every return value is writ
 * to the HTTP body as JSON by Jackson instead of being resolved as a view n
 */
@RestController
@RequestMapping("/api/students")
public class StudentRestController {

    private final StudentRepository students;

    // one constructor: Spring injects the repository, no @Autowired needed
    public StudentRestController(StudentRepository students) {
        this.students = students;
    }

    /** GET /api/students  or  GET /api/students?city=Jaipur */
    @GetMapping
    public List<Student> all(@RequestParam(required = false) String city) {
        return city == null ? students.findAll() : students.findByCityIgnore
    }

    /** GET /api/students/1 → 200 with the student, or 404 with an empty bo
    @GetMapping("/{id}")
    public ResponseEntity<Student> one(@PathVariable Long id) {
        return students.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
```

```
    }
}
```

Output

Checked by reading. Every endpoint with its curl command and expected response:

```bash
curl -s http://localhost:8080/api/students
```

```
[{"id":1,"name":"Asha Verma","email":"asha@example.com","phone":"9876543210"
 {"id":2,"name":"Ravi Kumar","email":"ravi@example.com","phone":"9123456780"
```

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

```
{"id":1,"name":"Asha Verma","email":"asha@example.com","phone":"9876543210",
```

```bash
curl -s "http://localhost:8080/api/students?city=patna"
```

```
[{"id":2,"name":"Ravi Kumar","email":"ravi@example.com","phone":"9123456780"
```

```bash
curl -i http://localhost:8080/api/students/99
```

```
HTTP/1.1 404
Content-Length: 0
```

The console shows the SQL Hibernate ran for the first call:

```
Hibernate: select s1_0.id,s1_0.city,s1_0.date_of_birth,s1_0.email,s1_0.name,
```

Explanation

`@RestController` is `@Controller` plus `@ResponseBody`: the returned `List<Student>` is not a view name, it is handed to Jackson, which walks the getters and writes JSON with `Content-Type: application/json`. `LocalDate` is written as `2002-03-14` because Boot registers the `JavaTimeModule` and turns off timestamp output. The constructor takes `StudentRepository`; with one constructor Spring injects it without `@Autowired`. `ResponseEntity` is used only where the status code varies: `findById` returns an `Optional`, `map(ResponseEntity::ok)` gives 200 with a body, `orElse(notFound())` gives 404 with none. `@RequestParam(required = false)` lets the same method serve the full list and the filtered list.

Viva Questions

× Do not copy. Read for understanding and the viva

- **Q:** What does `@SpringBootApplication` combine? **A:** `@Configuration`, `@EnableAutoConfiguration` and `@ComponentScan` on the package of the class.
- **Q:** Why is there no `<version>` under each dependency? **A:** `spring-boot-starter-parent` manages versions through its dependencyManagement section, so all starters agree.
- **Q:** What does `ddl-auto=update` do and why not use it in production? **A:** It adds missing tables and columns from the entities at start-up. It never drops or renames, so the schema drifts; production uses `validate` with migration scripts.
- **Q:** How does Spring Data implement `findByCityIgnoreCase` without code? **A:** It parses the method name into a JPQL query at start-up and generates a proxy that runs it.
- **Q:** What is the difference between `@Controller` and `@RestController`? **A:** `@RestController` adds `@ResponseBody`, so return values are written to the response as JSON instead of resolved as view names.
- **Q:** Why does `data.sql` need `defer-datasource-initialization`? **A:** Boot runs SQL scripts before JPA starts by default; deferring runs them after Hibernate created the tables.
- **Q:** Why is `mysql-connector-j` scoped `runtime`? **A:** Our code compiles against JDBC interfaces only; the driver is needed only when the application runs.
- **Q:** What does `Optional.map(ResponseEntity::ok).orElse(notFound().build())` return for a missing id? **A:** A `ResponseEntity` with status 404 and an empty body.

Common Mistakes

× Do not copy. Read for understanding and the viva

- Choosing Spring Boot 2.x on Initializr and then writing `jakarta.persistence` imports; 2.x uses `javax.persistence` and nothing compiles.
- Placing entity or controller packages outside `in.ignou.admission`; component scan never finds them and the endpoint returns 404 with no error in the log.
- Forgetting `spring.sql.init.mode=always`, so `data.sql` is silently skipped on MySQL and every GET returns `[]`.
- Using `@Enumerated` without `EnumType.STRING`, storing ordinals that break when the enum changes.
- Adding a getter-less field to an entity and wondering why it is missing from the JSON; Jackson serialises getters.
- Running before MySQL is up. The pool fails with `Communications link failure`, which students misread as a code error.

Session Summary

Write in lab record

- Initializr settings and the `pom.xml` with the six starters plus the MySQL driver
- `application.properties` with the datasource, `ddl-auto=update`, `show-sql` and the `data.sql` switches
- Entity classes `Student`, `Programme`, `Course`, `Admission` and the `AdmissionStatus` enum, with the generated `CREATE TABLE` statements
- The four `JpaRepository` interfaces with their derived query methods
- `StudentRestController` with the four curl calls and their JSON responses, including the 404 case

Edit this page

Previous
Session 6

Next
Session 8