Web Technologies and Full Stack Development

Servlets, JSP and Spring Boot

The notes for the lesson “Servlets, JSP and Spring Boot”: chapters 4.7 to 4.9, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

3 chapters | Chapters 4.7 to 4.9

About these notes

These are the notes for the University free n easy video lesson **Servlets, JSP and Spring Boot** on Web Technologies and Full Stack Development: chapters 4.7 to 4.9, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

4.7 Java Servlets: life cycle, request and response

A servlet is a Java class that answers HTTP requests inside a servlet container such as Tomcat. The container creates one object, calls `init()` once, `service()` for every request, and `destroy()` at shutdown. You write `doGet()` or `doPost()` to read parameters, headers and sessions, then write HTML or JSON back. This design gives Java web applications a single shared object with container-managed setup and cleanup. Servlets are the base under JSP, JSF and many Spring Boot request paths.

> You need: Java classes and objects, the HTTP request and response model from an earlier lesson, Tomcat setup from 4.2, and session tracking from 4.6.

Life cycle: one object, many requests

A servlet is a Java object whose life the container controls. Tomcat loads the class and creates exactly one instance for a mapping. After construction, the container calls `init(ServletConfig)` once, before the first request. Put one-time setup there, such as a database pool or a configuration value. Then `service()` runs for every request. `HttpServlet.service()` reads the HTTP method and calls `doGet()`, `doPost()`, `doPut()` and so on. At shutdown or undeploy, `destroy()` runs once.

Map the servlet in code with `@WebServlet("/greet")`, or in `web.xml` with servlet and servlet-mapping entries. Older containers use `javax.servlet` imports. Newer containers use `jakarta.servlet`. A class with the wrong import package is not a servlet for that container and fails to load.

The container uses one thread per request on the one servlet object. Instance fields are shared by all those threads. A field such as `String userName` can show one user's data to another user. Keep per-request data in local variables or request attributes. Use a `synchronized` block only for a true shared counter, and expect lower throughput.

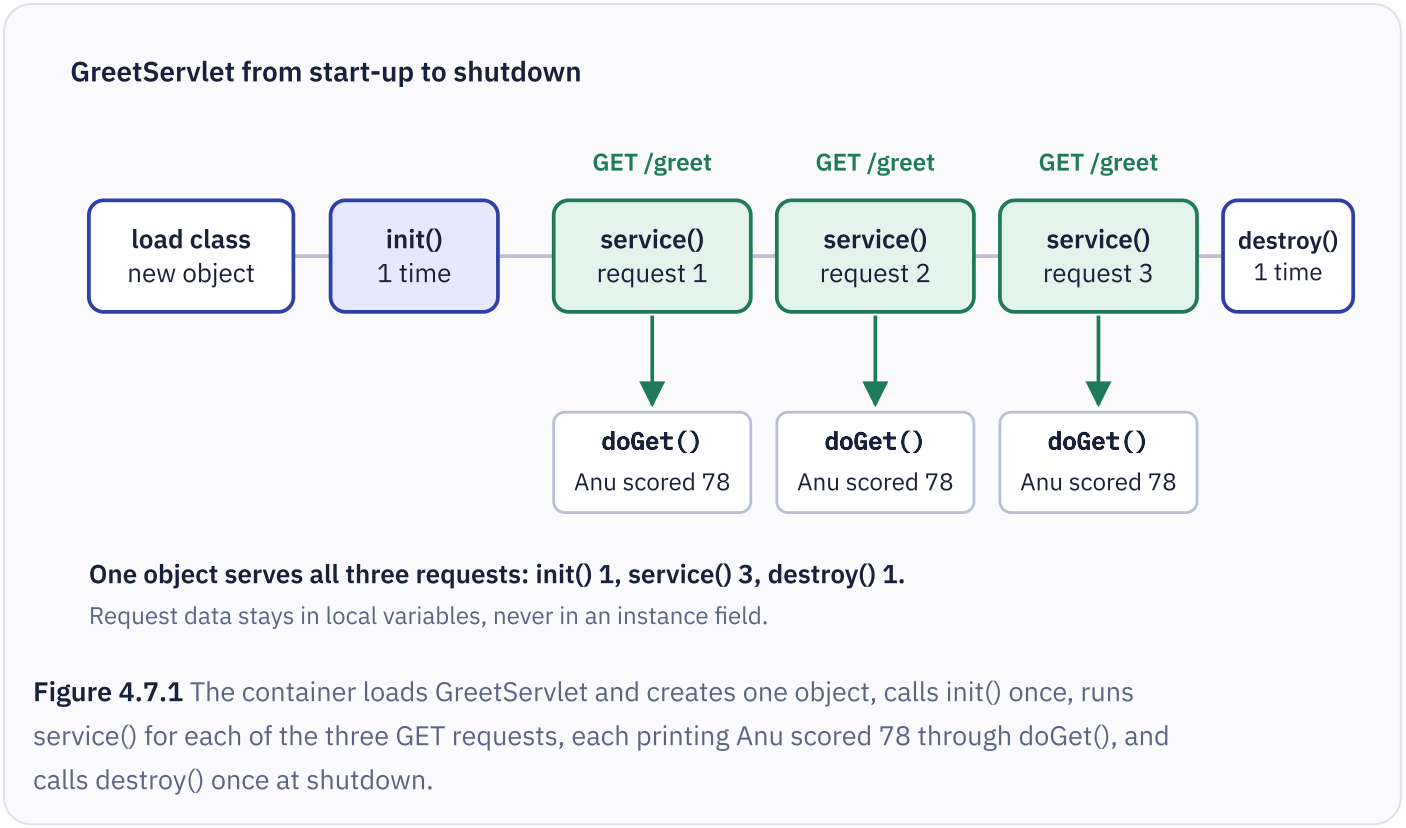


Figure 4.7.1 The container loads GreetServlet and creates one object, calls init() once, runs service() for each of the three GET requests, each printing Anu scored 78 through doGet(), and calls destroy() once at shutdown.

Reading a request and writing a response

Write a servlet by extending `HttpServlet`. The usual methods are `protected void doGet( HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException` and the same signature for `doPost`. `HttpServletRequest` gives `getParameter("name")`, which returns a `String` or `null`. For an `application/x-www-form-urlencoded` form POST, Tomcat parses the body and `getParameter` still works. For `multipart/form-data`, add `@MultipartConfig` or read parts with `getPart()`. Use `Integer.parseInt()` for numbers. `getParameter` returns `null` if the parameter is missing, so check for `null` and catch `NumberFormatException` before using `Integer.parseInt()`.

Read headers with `getHeader("User-Agent")` and the method with `getMethod()`. On the response, call `setContentType("text/html;charset=UTF-8")` before `getWriter()`. The

writer is a `PrintWriter`; `out.println()` writes the body. `setStatus(404)` or `sendError(404, "Not found")` sets the status. Do not write to the writer after a forward.

GET puts data in the URL, so it suits reads and bookmarkable pages. POST puts data in the body, so it suits form submissions and changes. If a servlet overrides only `doPost`, a GET reaches the default `doGet`, which returns 405 Method Not Allowed. The client sends the HTTP method in the request line. `service()` reads it and dispatches to the matching `doXxx()` method; your code never calls `doGet()` or `doPost()` itself.

Forward, redirect, parameters and session

A forward stays on the server: `req.getRequestDispatcher("/view.jsp").forward(req, res)` sends the same request and response objects to another resource. The browser URL does not change. A redirect sends `302` to the browser: `res.sendRedirect("next")`. The browser makes a new request, the URL changes, and request attributes are lost. Use forward for MVC views; use redirect after a POST so refresh does not resubmit.

Init-params belong to one servlet: `getServletConfig().getInitParameter("db")`. Context-params belong to the whole web app: `getServletContext().getInitParameter("db")`. Init-params are set in `web.xml` or with `@WebInitParam` inside `@WebServlet`. Context-params have no annotation; set them in `web.xml` with <context-param>, or at startup in a `ServletContextListener` using `getServletContext().setInitParameter()`. Both reach your code as strings.

For login state use `HttpSession session = req.getSession(true)`, then `session.setAttribute("user", name)`. The session ID travels in a cookie, as in 4.6. Session attributes are shared across requests from the same browser, so guard mutable objects. Remove with `removeAttribute` or `invalidate()` at logout.

WORKED EXAMPLE

Question. A `GreetServlet` is mapped to `/greet`. It reads `name=Anu` and `marks=78`. Give the `doGet` body and trace the life cycle for three identical GET requests. State the exact output for one request and the number of `init()`, `service()` and `destroy()` calls. The small class below simulates what the container does so you can see the call order. The real servlet extends `HttpServlet` and its `doGet` takes `HttpServletRequest` and `HttpServletResponse`, as shown in step 3.

Java program.

```
public class GreetServletSim {
    void init() {
        System.out.println("init() called once");
    }
    void service(String name, int marks) {
        doGet(name, marks);
    }
    void doGet(String name, int marks) {
        System.out.println(name + " scored " + marks);
    }
    void destroy() {
        System.out.println("destroy() called once");
    }
    public static void main(String[] args) {
        GreetServletSim s = new GreetServletSim();
        s.init();
        int calls = 0;
        for (int i = 0; i < 3; i++) {
            s.service("Anu", 78);
            calls++;
        }
        System.out.println("service() calls = " + calls);
        s.destroy();
    }
}
```

1. The container loads `GreetServlet` and creates one instance.
2. It calls `init()` once. No request data is stored in an instance field.
3. In `doGet`: `String name = req.getParameter("name"); int marks = Integer.parseInt(req.getParameter("marks")); res.setContentType("text/html;charset=UTF-8"); PrintWriter out = res.getWriter(); out.println(name + " scored " + marks);`.
4. Request 1: `name` is `Anu`, `marks` is `78`, output is `Anu scored 78`.

5. Requests 2 and 3 repeat the same `service()` and `doGet()` calls with the same values.
6. At shutdown `destroy()` runs once. Counts: `init()` 1, `service()` 3, `destroy()` 1.

Dry run

REQUEST	NAME	MARKS	PRINTED LINE
1	Anu	78	Anu scored 78
2	Anu	78	Anu scored 78
3	Anu	78	Anu scored 78

Output

```
init() called once
Anu scored 78
Anu scored 78
Anu scored 78
service() calls = 3
destroy() called once
```

Answer. One request prints `Anu scored 78`; across three requests `init()` runs 1 time, `service()` runs 3 times, and `destroy()` runs 1 time.

EXAM TRAPS

- Putting per-user data in an instance field of the servlet. The container has one servlet object, so all requests share that field.
- Expecting `doPost` to run for a GET. `service()` dispatches by HTTP method, and the default `doGet` returns 405.
- Calling `getWriter()` before `setContentType()`. Set the content type and charset first.
- Confusing forward with `sendRedirect`. Forward keeps the URL and request attributes; redirect starts a new request in the browser.
- Mixing `javax.servlet` and `jakarta.servlet` imports. A container accepts only one package name.

CHECK YOURSELF

– REMEMBER **Which servlet method runs exactly once after the object is created and before the first request?** 2 marks

`init()`.

– UNDERSTAND **Why does a servlet usually avoid mutable instance fields for per-user data?** 3 marks

The container has one servlet instance and serves requests on many threads. A shared field can mix one user's data into another user's response. Use local variables, request attributes or session attributes.

– APPLY **A servlet is mapped to `/hi`. A request is `/hi?name=Ravi`. Complete the line that prints `Hi Ravi`.** 3 marks

`out.println("Hi " + req.getParameter("name"));` prints `Hi Ravi`.

– ANALYZE **A servlet overrides only `doPost`. A browser sends `GET /save`. What response comes?** 4 marks

405 Method Not Allowed. `service()` dispatches a GET to `doGet`, and the default `doGet` sends 405.

– REMEMBER **State one difference between `forward` and `sendRedirect`.** 2 marks

Forward runs on the server with the same request and keeps the URL. Redirect sends 302 and the browser makes a new request with a new URL.

– APPLY **Write the call that reads an init-param named `db` from inside a servlet.** 4 marks

`getServletConfig().getInitParameter("db")` returns the value.

– ANALYZE **Two threads run a servlet method that does `count++` on an instance field without synchronization. What can go wrong?** 5 marks

The read-modify-write can interleave. Both threads may read the same old value, so one increment is lost and the count is too small. Use `AtomicInteger` or synchronize.

BEYOND THE SYLLABUS

Not for the exam. Spring Boot wraps a servlet container and routes requests through `DispatcherServlet`, but singleton beans still face the same shared-state rule. Servlet filters and listeners add cross-cutting work such as logging, compression and security. Virtual threads change the cost of blocking calls, not the HTTP contract. Search next: Jakarta Servlet 6, Tomcat threading, `RequestDispatcher`, servlet filters.

4.8 JSP and JSF for dynamic pages

A JSP is an HTML page with Java inside. The container translates it into a servlet on first request, compiles it once, then reuses that servlet. You write the page, not println lines. JSF declares inputs and buttons as components, binds them to Java beans, and writes and reads HTML for you. Only a few universities still teach JSP and JSF; most new work uses Spring or React.

You need 4.7 for the servlet life cycle and HTML forms.

From hello.jsp to a running servlet

hello.jsp is plain text. On first request the container translates it to hello_jsp.java, which extends a JSP base servlet, compiles it to hello_jsp.class, loads it, and creates one instance.

1. translate hello.jsp → hello_jsp.java
2. compile → hello_jsp.class
3. load, jspInit() once
4. _jspService() per request
5. jspDestroy() at shutdown

This is the life cycle of 4.7 with two renamed methods.

Directives configure the page. `<%@ page import="java.util.*" %>` sets imports, content type or session use. `<%@ include file="header.jsp" %>` copies a file at translation time; `<jsp:include page="..." />` runs at request time. `<%@ taglib %>` declares a tag library.



Figure 4.8.1 JSP reads name and marks straight from the query string with ${param...}; JSF first copies the typed values into marksBean on postback and then reads #{marksBean...}. Both print Anu scored 78.

Scriptlets, implicit objects, EL and JSTL

Three tags carry Java: scriptlet `<% %>`, expression `<%= %>`, declaration `<%! %>`.

Nine implicit objects exist. Main five: `request`, `response`, `out`, `session` (see 4.6), `application`; rest are config, page, pageContext, exception.

EL reads values: `${param.name}` is a parameter, `${sessionScope.user}` a session attribute. JSTL: `<c:forEach>` loops, `<c:if test="...">` decides, `<c:out value="..." />` prints with HTML escaping.

MVC and the JSF component tree

A JSP should only display. A servlet controller reads parameters, calls model classes, stores results as request attributes, then forwards with `request.getRequestDispatcher("/show.jsp").forward(request, response)`. The JSP prints `${student.marks}`.

JSF builds on servlets. A Facelet (.xhtml) uses `h:` tags and the framework builds a component tree, one object per input/button. `<h:inputText value="#{marksBean.name}"/>` calls getName() when rendering and setName() on submit.

A JSF request has six phases: restore view, apply request values, process validations, update model values, invoke application, render response. `required="true"` is checked in phase three; on failure phases four and five are skipped, so your action never runs and the page redraws with the message.

WORKED EXAMPLE

Question. A page must show `Anu scored 78` from URL `?name=Anu&marks=78`. Write the JSP with EL, then the JSF page and bean binding that show the same text.

```
show.jsp:
<p>${param.name} scored ${param.marks}</p>

marks.xhtml:
<h:form><h:inputText value="#{marksBean.name}"/><h:inputText value="#{marksBean.marks}"

MarksBean: String name; int marks; getName/setName/getMarks/setMarks.
```

1. JSP path: the browser sends name=Anu and marks=78; the request parameter map holds both pairs.
2. show.jsp prints `<p>${param.name} scored ${param.marks}</p>` as Anu scored 78.
3. JSF path: marks.xhtml holds the form in the code block. The user types Anu and 78 and presses Show; a first GET does not copy query parameters into input components.
4. On postback JSF converts the values and calls setName("Anu") and setMarks(78) in apply request values / update model values.
5. The button returns result; navigation opens result.xhtml with `#{marksBean.name} scored #{marksBean.marks}`.

Dry run

STEP	VALUE READ	OUTPUT SO FAR
first request: translate+compile	hello.jsp	
_jspService: read param.name	Anu	Anu scored
_jspService: read param.marks	78	Anu scored 78

Output

```
Anu scored 78
```

Answer. Both print Anu scored 78. The JSP reads the query string directly; JSF copies typed values into the bean after postback.

EXAM TRAPS

- JSP is not JavaScript: scriptlets run on the server; the browser receives only HTML.
- EL does not escape HTML: `${param.name}` is raw; use `<c:out value="${param.name}" />` against script injection.
- `<%@ include %>` merges at translation time; `<jsp:include>` runs at request time.
- Failed JSF validation skips update model values and invoke application; setter and action do not run.
- In JSF the container creates the bean; `new MarksBean()` gives a different object.

CHECK YOURSELF

– REMEMBER **Name the three JSP scripting elements and the symbol of each.** 2 marks

Scriptlet `<% %>`, expression `<%= %>`, declaration `<%! %>`.

– UNDERSTAND **Why is hello.jsp compiled once but _jspService run for every request?** 3 marks

Translation and compilation build the class once; that class is a servlet, so one instance serves every request.

– APPLY **A request carries `?city=Chennai&temp=34`. Write the EL that prints `Chennai is 34 C`.** 4 marks

`${param.city} is ${param.temp} C`.

– ANALYZE **A student writes `<% int hits = 0; hits++; %><%= hits %>` and expects a counter shared by all users. Explain the result and the fix.** 5 marks

hits is a local variable of _jspService, recreated as 0 on each request, so the page always shows 1. Use an application attribute with synchronization or an `AtomicInteger`; a declaration `<%! int hits = 0; %>` is shared but hits++ is not thread-safe.

– APPLY **Write the Facelet that binds a text field to property amount of billBean and submits to outcome total.** 4 marks

`<h:form><h:inputText value="#{billBean.amount}"/><h:commandButton value="Total" action="total"/></h:form>`.

– UNDERSTAND **What is the JSF component tree and when is it used?** 3 marks

A server-side tree of component objects, one per tag on the Facelet; it is restored at the start of a request and rendered at the end.

BEYOND THE SYLLABUS

Not for the exam. JSP is in maintenance mode: new Java services usually return JSON from Spring Boot and let a React or Angular front end draw the page. JSF survives in long-lived enterprise portals and is specified as Jakarta Faces. Search 'Jakarta Server Pages specification', 'Facelets templating' and 'JSF life cycle phases' to go deeper.

4.9 Spring Boot REST services with JPA

Spring Boot builds a web service you start with one command. It packs an embedded Tomcat inside your JAR, so you install no separate server. @RestController turns a Java method into a URL that answers JSON. Spring Data JPA turns a repository call into SQL against a real table. Teams use this stack for payment APIs, order services and mobile backends. Only a few universities teach it. You need servlet request and response and REST, JSON and databases (a later lesson).

You need: 4.7 Java Servlets, and a later lesson REST, JSON and databases.

Starters and the embedded server

A Spring Boot project is a Maven project with a parent that fixes library versions. You add a starter, not a library: a starter is a bundle of dependencies tested together.

- spring-boot-starter-web brings Spring MVC, Jackson for JSON, and embedded Tomcat.
- spring-boot-starter-data-jpa brings JPA and Hibernate; the H2 or MySQL driver is added separately.
- spring-boot-starter-test brings JUnit and Mockito.

The class carrying @SpringBootApplication is three annotations in one: @Configuration, @EnableAutoConfiguration and @ComponentScan. Its main method calls SpringApplication.run(App.class, args).

Auto-configuration reads application.properties. Set spring.datasource.url, spring.datasource. username and spring.datasource.password for the database. Add spring.jpa.hibernate.ddl-auto= update so Hibernate creates or alters tables.

Embedded Tomcat starts inside the same Java process, by default on port 8080. mvn package builds one runnable JAR, and java -jar app.jar starts the whole service on any machine with a

JVM.

@RestController: one method, one URL

@RestController marks a class whose methods return data, not page names. Spring MVC maps the URL to a method, then Jackson writes the returned object as JSON.

The mapping annotation fixes the HTTP method: @GetMapping reads, @PostMapping inserts, @PutMapping and @DeleteMapping handle the rest. A path such as /api/students/{id} holds a stand-in.

- @PathVariable Long id reads {id} from the URL and converts it to Long.
- @RequestBody Student s reads the JSON body and converts it to a Student.
- ResponseEntity<Student> carries body and status together: ResponseEntity.status(201).body(saved).

A plain returned object gets status 200. Use ResponseEntity when the status must differ, for example 201 after an insert.

Spring Data JPA: an interface that writes SQL

JPA maps a class to a table. @Entity marks the class, @Id marks the primary key, and @GeneratedValue(strategy = GenerationType.IDENTITY) lets the database supply the next id. The class needs a public no-arg constructor, because Hibernate builds instances by reflection.

A repository is an interface, not a class: interface StudentRepository extends JpaRepository<Student, Long>. The two type parameters are the entity and the id type. Spring writes the implementation at startup.

- findById(1L) returns Optional<Student> and runs a SELECT by primary key.
- findAll() runs SELECT * FROM student.
- save(s) runs INSERT when the id is null and UPDATE when it is set.

For learning use H2 in memory: jdbc:h2:mem:testdb. For a real service use MySQL: jdbc:mysql://localhost:3306/college.

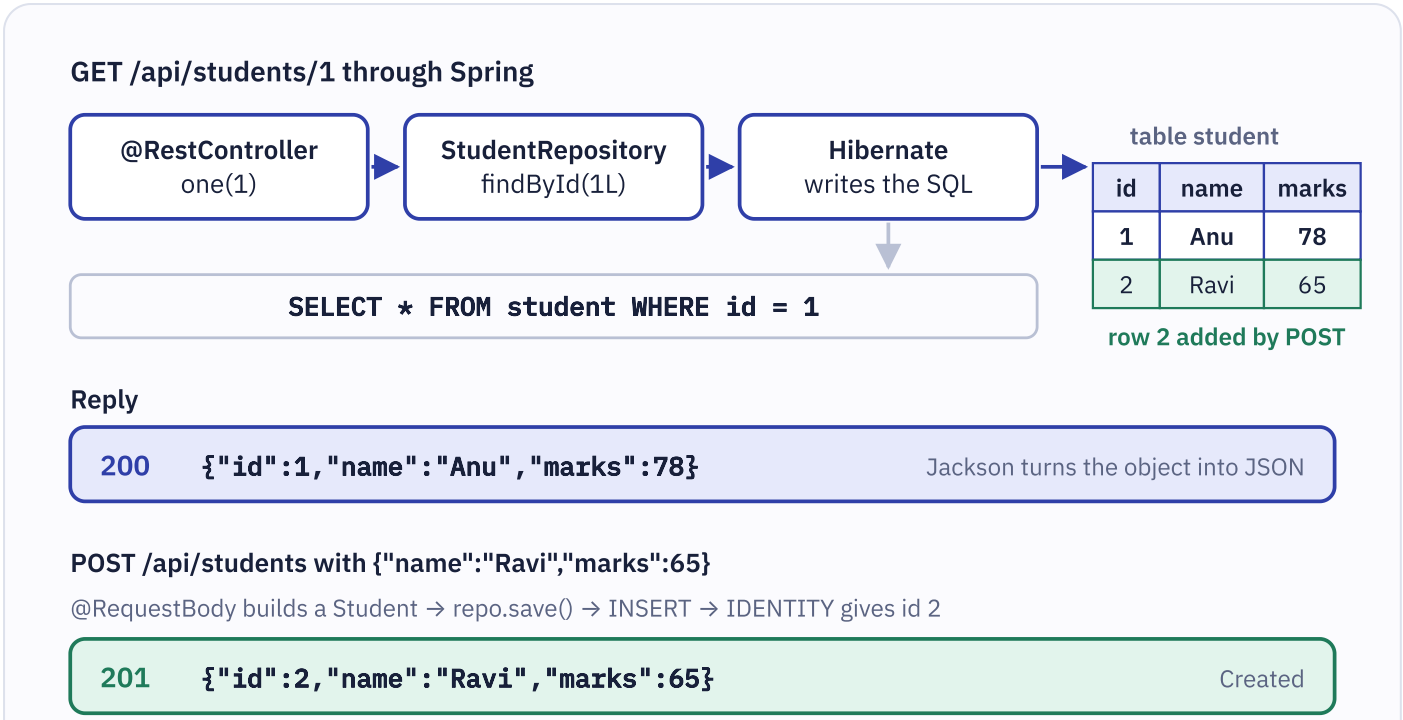


Figure 4.9.1 The controller calls findById(1L), Hibernate runs SELECT ... WHERE id = 1, and the row (1, Anu, 78) comes back as JSON with status 200. A POST of Ravi with 65 marks is saved as row 2 and answered with 201.

WORKED EXAMPLE

Question. The table student has columns id, name and marks, and row (1, Anu, 78) exists. Write the entity, repository and controller for GET /api/students/1 and give the JSON. Then give the POST that adds Ravi with 65 marks, and its response.

```
@Entity
class Student {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  Long id;
  String name;
  int marks;
  Student() {}
  // getters and setters
}

interface StudentRepository extends JpaRepository<Student, Long> {}

@RestController
@RequestMapping("/api/students")
class StudentController {
  private final StudentRepository repo;
  StudentController(StudentRepository repo) { this.repo = repo; }

  @GetMapping("/{id}")
  Student one(@PathVariable Long id) {
    return repo.findById(id).orElseThrow();
  }

  @PostMapping
  ResponseEntity<Student> add(@RequestBody Student s) {
    Student saved = repo.save(s);
    return ResponseEntity.status(201).body(saved);
  }
}
```

1. Entity Student: @Entity, @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Long id, String name, int marks, and a no-arg constructor. Hibernate maps it to table student.
2. Repository: interface StudentRepository extends JpaRepository<Student, Long>. Spring generates the class; findById(1L) becomes SELECT * FROM student WHERE id = 1.
3. Controller: @RestController with @GetMapping("/{id}") and method Student one(@PathVariable Long id) { return repo.findById(id).orElseThrow(); }.
4. Jackson writes the returned object as {"id":1,"name":"Anu","marks":78}. The method returned normally, so the status is 200.

5 POST /api/students with body {"name":"Ravi","marks":65}: @RequestBody builds a Student, repo.save inserts the row, IDENTITY returns id 2, and ResponseEntity.status(201) sends 201 with {"id":2,"name":"Ravi","marks":65}.

Dry run

STEP	CALL	SQL	RESULT
1	findById(1L)	SELECT WHERE id = 1	Student(1, Anu, 78)
2	Jackson writes body	none	200, id 1
3	save(Ravi, 65)	INSERT name, marks	id = 2
4	Jackson writes body	none	201, id 2

Answer. GET /api/students/1 returns 200 with {"id":1,"name":"Anu","marks":78}; POST returns 201 with {"id":2,"name":"Ravi","marks":65}.

EXAM TRAPS

- An @Entity without @Id, or without a no-arg constructor, fails at startup; Hibernate also needs getters or field access, so pick one style.
- JpaRepository is an interface. You never write new StudentRepository(); you inject it into the controller constructor.
- Returning the entity leaks every column and can trigger lazy loading while Jackson writes JSON; a DTO is safer.
- @Controller returns a view name, @RestController writes the body. Mixing them gives a rendered page or a 404, not JSON.
- @PathVariable reads the URL, @RequestBody reads the JSON body. Swapping them gives a null field or a 400.

CHECK YOURSELF

– REMEMBER **What does @RestController add to a class?** 2 marks

@Controller plus @ResponseBody, so return values are written as the response body instead of a view name.

– UNDERSTAND **Why does Spring Data JPA need only an interface for a repository?** 3 marks

Spring generates the implementing proxy at startup from JpaRepository's declared methods, so findById and save need no hand-written SQL.

– APPLY **The table already holds ids 1 and 2. A POST body is {"name":"Meera", "marks":91}. Give the saved id and the response.** 4 marks

IDENTITY gives id 3; status 201 with {"id":3,"name":"Meera","marks":91}.

– ANALYZE **findById returns Optional<Student>. What happens by default when the id is missing and the method calls orElseThrow()?** 5 marks

It throws NoSuchElementException, which Spring maps to 500; a 404 needs an explicit exception handler.

– REMEMBER **Which embedded server does Spring Boot start by default, and on which port?** 2 marks

Tomcat, on port 8080.

– UNDERSTAND **Give the difference between @PathVariable and @RequestBody.** 3 marks

@PathVariable reads one value from the URL path; @RequestBody deserializes the whole JSON body into an object.

BEYOND THE SYLLABUS

Not for the exam. Spring Boot is the default Java backend in Indian service companies, so the next steps are Spring Security with JWT for login tokens, Actuator for health and metrics endpoints, and Testcontainers for tests against a real MySQL. Spring Cloud adds service discovery and central configuration once one service becomes ten. Docker packaging and CI/CD pipelines follow. Search 'spring boot jpa crud example', 'spring boot actuator health' and 'testcontainers mysql junit'.

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
Servlet life cycle	init once, service on every request, destroy once — all called by the container	4.7
Managed bean	a Java object that JSF creates and links to page fields by name	4.8
@RestController	Spring annotation marking a class whose method return values become the response body, usually JSON	4.9
JPA repository	an interface you declare; Spring Data writes the SQL behind findAll(), save() and friends	4.9

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

– REMEMBER **Describe the servlet life cycle. State which method runs how many times and who calls it.** 10 marks

Container loads the class and creates one instance; init() once, after construction, for setup; service() for every request; it dispatches to doGet or doPost; destroy() once before the object is removed; Programmer overrides methods, never calls them; One instance may serve many requests on different threads, so instance fields are shared.

Apply

– APPLY **Write a servlet that reads two numbers from an HTML form and prints their sum, showing doGet and the response content type.** 12 marks

Class extends HttpServlet; doGet(HttpServletRequest, HttpServletResponse); setContentType("text/html") before writing; request.getParameter for both fields, parse with Integer.parseInt; PrintWriter out = response.getWriter(); Handle NumberFormatException with a message.

– APPLY **Write a Spring Boot REST controller with one endpoint that saves an entity and one that lists all rows, using a JPA repository.** 12 marks

@RestController and @RequestMapping paths; Repository interface extends JpaRepository<Entity, Long>; @GetMapping returns List, Spring serialises to JSON; @PostMapping with @RequestBody binds the JSON to the entity; save() and findAll() explained; Embedded Tomcat started by main().

Analyze and evaluate

– EVALUATE **Compare PHP with Java servlets and JSP as server-side technologies: compilation, threading, database access and hosting cost.** 13 marks

PHP: interpreted per request, quick to deploy, cheap shared hosting; Servlets: compiled bytecode, JVM, one instance on many threads; JSP compiles to a servlet on first use; Both talk to databases through a driver; PHP uses PDO or mysqli, Java uses JDBC or JPA; Java gives strong typing and a large library; PHP gives a shorter learning curve; Choose by team skills, expected load, hosting budget.

– ANALYZE **Explain how a JSP page becomes a servlet, how JSF adds components and beans, and how Spring Boot changes the deployment step.** 14 marks

JSP translated to Java source, then compiled to a servlet class; Life cycle is the servlet life cycle; jspService handles the request; JSF: reusable UI components, a component tree, managed beans bound by name, navigation rules; Spring Boot: starter dependencies, auto-configuration, embedded Tomcat; Build produces an executable JAR or a WAR for an external container; @RestController method returning an object becomes a JSON body.

Glossary (terms used in this lesson)

Cookie A small name=value string the server asks the browser to store and return on later requests to that site.

JPA	Jakarta Persistence API, the standard set of annotations and interfaces used for ORM in Java.
JSF	JavaServer Faces: a component-based Java web framework built on servlets.
JSP	JavaServer Pages: an HTML file with Java pieces that the container translates into a servlet.
ORM	Object-relational mapping: linking Java class fields to database table columns so you work in objects, not SQL strings.
Query string	The part of a URL after ?, holding name=value pairs joined by &.
Servlet	A Java class extending HttpServlet that the container calls to answer HTTP requests.
Session	Server-side storage tied to one visitor by a session ID, used for logins and carts.
Session ID	A long random string that links a browser's cookie to the right server-side session data.
Spring Boot	A Spring framework layer that auto-configures an application and starts an embedded server.

Sources and further reading

Textbooks

- Steven Holzner, *PHP: The Complete Reference*: Syntax, variables, data types, operators, control statements, arrays, strings, form handling and cookies.
- Marty Hall and Larry Brown, *Core Servlets and JavaServer Pages, Volume 1: Core Technologies*, 2nd: Servlet life cycle, request and response objects, JSP translation and scripting elements.
- Craig Walls, *Spring in Action*, 6th: REST controllers, Spring Data JPA repositories and Spring Boot auto-configuration.
- Lynn Beighley and Michael Morrison, *Head First PHP & MySQL*: Forms, sessions and cookies with worked PHP examples.

Open web resources

Each link below was opened by the book script on 11 Nov 2026 and returned a page; the pages were not read line by line.

- PHP Manual (official): www.php.net/manual/en/index.php
- XAMPP — Apache Friends: www.apachefriends.org

- Apache Tomcat (official site): tomcat.apache.org
- Spring Guides — Building a RESTful Web Service: spring.io/guides/gs/rest-service
- Spring Data JPA project page: spring.io/projects/spring-data-jpa
- MDN Web Docs — HTTP cookies: developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

Syllabus: the University free n easy universal syllabus, built from top universities' syllabi.

Built for the University free n easy YouTube channel from the universal syllabus of Web Technologies and Full Stack Development (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.