Java Full-Stack Internship

14 Days · Spring Boot · MySQL · REST API · JPA · Firebase

Java 21 | Spring Boot | MySQL | JPA/Hibernate | REST API | Firebase | LLM Dev

---

Core2Cloud • core2cloud.in

Practical internship — 1 hour/day — real projects — industry workflow

Contents

DAY 1 · 1 HOUR

Git & Dev Tools

Set up VS Code, IntelliJ, and master Git — the one tool every tech company tests on Day 1 of onboarding.

0–5 min Real Hook	5–20 min Git Concepts	20–40 min Live Demo	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Imagine 5 developers all editing the same `App.java` on WhatsApp. Priya adds login. Ravi adds payment. Ravi sends his file to Priya. She overwrites it. **Ravi's payment feature is gone forever.**

This happened at a startup in 2019 — they lost 3 weeks of work. **Git is the solution.** It's a time machine for your code. Every tech company — Google, Infosys, Zoho — uses it on Day 1.

⚙️ Quick Setup (5 min)

- 1
  Install Git — `git --version` to verify. Download from git-scm.com if missing.
- 2
  Install VS Code — Add extensions: GitLens, Java Extension Pack, Live Server.
- 3
  Install IntelliJ IDEA Community — Free. Smarter Java autocomplete than VS Code.
- 4
  Create a GitHub account at github.com — use your real name. This is your developer portfolio.

🧠 Git — The 3 Zones (5–20 min)

💡 ANALOGY — GIT IS LIKE A PHOTO ALBUM

Working Directory = You're clicking photos right now (editing files).
Staging Area = You selected the best photos to print (git add).
Repository = The photos are printed and in the album forever (git commit).
GitHub = You shared the album online (git push).

```
# One-time setup
git config --global user.name  "Your Name"
git config --global user.email "you@gmail.com"

# Start a new project
git init                        # create .git folder
git clone https://github.com/x  # copy an existing repo

# Daily workflow — ALWAYS these 3 steps
git add .                       # stage all changes
git commit -m "add login form"  # save snapshot
git push origin main            # upload to GitHub

# Before you start coding each day
git pull                        # get teammate's latest changes
```

🌿 Branching — Work Without Fear (20–40 min)

> 🌍 **REAL EXAMPLE — FEATURE BRANCH**
>
> At Flipkart, no one codes directly on `main`. They create a branch like `feature/add-cart-button`, code there, then merge. If something breaks, `main` is always safe.

```
# Create and switch to a new branch
git checkout -b feature/login-page

# Code your feature, then commit
git add .
git commit -m "add login form UI"

# Merge back to main when done
git checkout main
git merge feature/login-page

# Useful inspection commands
git status          # what changed?
git log --oneline   # history in one line each
git diff            # exact line changes
```

🚫 .gitignore — What NOT to commit

> ⚠️ **NEVER COMMIT THESE**
>
> API keys, passwords, `.env` files — once pushed to GitHub (even if deleted later), they are compromised forever. Google has bots scanning GitHub for leaked keys.

```
# .gitignore file in your project root
.env
*.log
target/
node_modules/
.idea/
*.class
```

🎯 You Code It (40–55 min)

1. Create a folder `my-portfolio` on your desktop
2. Run `git init` inside it
3. Create `README.md` — write your name and 3 goals for this internship
4. Run `git add .` then `git commit -m "first commit"`
5. Create a new repo on GitHub named `my-portfolio`
6. Push: `git remote add origin <url>` then `git push -u origin main`
7. **Goal:** Show the instructor your GitHub repo URL

📄 Day 1 Cheat Sheet — Git & Dev Tools

SETUP ONCE

git config --global user.name "X"	Set name
git config --global user.email "x"	Set email
git init	New repo
git clone <url>	Copy repo

DAILY WORKFLOW

git pull	Get latest
git add .	Stage all
git commit -m "msg"	Save snapshot
git push	Upload

BRANCHING

git checkout -b name	Create + switch
git checkout main	Back to main
git merge name	Merge branch
git branch -d name	Delete branch

INSPECT

git status	What changed
git log --oneline	History
git diff	Line changes
git stash	Save dirty work

VS CODE SHORTCUTS

Ctrl + P	Open file
Ctrl + `	Terminal
Ctrl + Shift + P	Command palette
Ctrl + /	Toggle comment

GOOD COMMIT MESSAGES

add login form	✅ Clear
fix null pointer in UserService	✅ Specific
update stuff	❌ Vague
asdfgh	❌ Useless

DAY 3 · 1 HOUR

Java Fundamentals

Every ATM, every Android app, every Zomato backend runs Java — write your first real program.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 **REAL-WORLD HOOK (0–5 MIN)**

Every Android app. Every bank ATM. Every Zomato backend. Every Infosys enterprise system. They all run Java.

Java is 30 years old and still the #1 language for backend jobs in India. Today you write your first real Java program — an ATM simulation with real logic: balance checks, deposits, withdrawals.

☕ Core Java Concepts (5–20 min)

💡 **MENTAL MODEL**

Variable — a labelled box storing a value.
Method — a named set of instructions.
if/else — a decision (if balance enough, allow withdrawal).
for loop — repeat N times (process 100 transactions).

💻 Live Coding — ATM Simulation (20–40 min)

```
public class ATM {
    private String owner;
    private double balance;

    public ATM(String owner, double initialBalance) {
        this.owner   = owner;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount ≤ 0) { System.out.println("Invalid amount"); return; }
        balance += amount;
        System.out.println("Deposited ₹" + amount + ". Balance: ₹" + balance);
    }

    public void withdraw(double amount) {
        if (amount > balance) {
            System.out.println("Insufficient funds!");
        } else {
            balance -= amount;
            System.out.println("Withdrawn ₹" + amount + ". Remaining: ₹" + balance);
        }
    }

    public static void main(String[] args) {
        ATM atm = new ATM("Priya", 10000);
        atm.deposit(5000);
        atm.withdraw(3000);
        atm.withdraw(20000);  // Insufficient funds!
    }
}
```

🔄 Processing Multiple Transactions with for-each

```
double[] transactions = {500, -200, 1000, -50, -800};
double balance = 5000;
for (double t : transactions) {
    if (t > 0) balance += t;
    else       balance -= Math.abs(t);
    System.out.println("Balance: ₹" + balance);
}
```

🎯 You Code It (40–55 min)

1. Create a `GradeCalculator` class with method `getGrade(int marks)`
2. Logic: marks≥90="A", ≥75="B", ≥60="C", ≥50="D", else="F"
3. Add method `processClass(int[] marks)` — print grade for each student
4. Add method `getClassAverage(int[] marks)` — returns average as double
5. Test with: `int[] marks = {92, 45, 78, 63, 88, 31, 95}`
6. **Show the instructor:** Expected output shows all grades + class average

📄 Day 3 Cheat Sheet — Java Fundamentals

PRIMITIVE TYPES

int n = 42;	Whole number
double d = 3.14;	Decimal
boolean b = true;	True/false
char c = 'A';	Single char
String s = "hi";	Text (Object)
long l = 99999L;	Big whole number

STRING METHODS

s.length()	Character count
s.toUpperCase()	All caps
s.contains("x")	Check substring
s.replace("a","b")	Replace
s.trim()	Remove spaces
s.split(",")	Split to array

CONTROL FLOW

if (x>0) {}	Condition
else if (x==0) {}	Another branch
for (int i=0;i<n;i++) {}	Count loop
for (T x : list) {}	For-each
while (cond) {}	While loop
return value;	Exit method

USEFUL UTILS

Integer.parseInt("5")	String->int
String.valueOf(42)	int->String
Math.max(a,b)	Larger value
Math.abs(n)	Absolute value
Math.random()	0.0 to 1.0
System.out.printf("%.2f",x)	Formatted print

DAY 4 · 1 HOUR

Object-Oriented Programming

Why apps update without reinstalling — the power of blueprints and inheritance.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Why does your phone update Instagram without reinstalling the whole OS? Why can a Toyota mechanic understand a BMW engine?

Object-Oriented Design. Everything is built from reusable blueprints (classes). Change one blueprint — all objects update. This is how Netflix, Uber, and Amazon build software that lasts decades.

🌍 The 4 Pillars (5–20 min)

💡 ANALOGIES

Encapsulation — A capsule hides the chemical inside. You swallow it (call the method), not mix chemicals yourself.
Inheritance — A Manager IS an Employee. Child class gets everything from parent — no duplication.
Polymorphism — "Open" means different things: open a door, open a file. Same method name, different behaviour.
Abstraction — You drive a car without knowing the engine. Abstract class hides complexity.

💻 Live Coding — Employee Hierarchy (20–40 min)

```
public class Employee {
    protected String name;
    protected double salary;

    public Employee(String name, double salary) {
        this.name = name; this.salary = salary;
    }

    public void work() {
        System.out.println(name + " is working");
    }
}

public class Manager extends Employee {
    private int teamSize;

    public Manager(String name, double salary, int teamSize) {
        super(name, salary);
        this.teamSize = teamSize;
    }

    @Override
    public void work() {   // Polymorphism
        System.out.println(name + " is managing a team of " + teamSize);
    }
}

// Usage
Employee emp = new Employee("Priya", 50000);
Manager  mgr = new Manager("Ravi", 90000, 8);
emp.work();  // "Priya is working"
mgr.work();  // "Ravi is managing a team of 8"
```

🔗 Interface — A Contract Every Class Must Honour

```
interface Payable {
    double calculatePay();
}

class FullTimeEmployee implements Payable {
    public double calculatePay() { return 50000; }
}

class Contractor implements Payable {
    private int hours;
    public double calculatePay() { return hours * 500; }
}
```

🎯 You Code It (40–55 min)

1. Create a `BankAccount` class: fields `accountNo`, `balance`; methods `deposit()`, `withdraw()`, `getBalance()`
2. Create `SavingsAccount extends BankAccount` — override `withdraw()` to prevent balance going below ₹500
3. Create `LoanAccount extends BankAccount` — add field `interestRate`, method `calculateInterest(int months)`
4. In `main()`: deposit ₹10,000 then try withdrawing ₹9,800 from SavingsAccount
5. **Goal:** SavingsAccount prints "Cannot withdraw — minimum balance required"

📄 Day 4 Cheat Sheet — OOP

CLASS & OBJECT

class Car { }	Blueprint
Car c = new Car()	Create object
private int speed	Hidden field
public int getSpeed()	Getter
this.speed = speed	Own field

INHERITANCE

class Dog extends Animal	Inherit
super(name)	Call parent constructor
super.speak()	Call parent method
@Override	Replace parent method
instanceof	Check type

INTERFACE

ACCESS MODIFIERS

interface Flyable { }	Define contract
class Bird implements Flyable	Implement
default void method()	Optional override
implements A, B	Multiple OK
extends only ONE class	Single inheritance

public	Anyone can access
private	Only this class
protected	This + subclasses
abstract	Must be subclassed
final	Cannot override
static	Belongs to class

DAY 5 · 1 HOUR

Advanced Java

Streams, Lambdas, Optional — the tools that separate junior devs from senior devs.

0–5 min	5–20 min	20–40 min	40–55 min	55–60 min
Real Hook	Core Concept	Live Coding	You Code It	Cheat Sheet

🔥 **REAL-WORLD HOOK (0–5 MIN)**

Netflix shows you 200 recommended movies filtered from 10,000 — in under a second. How?

Not a manual for-loop. They use **Java Streams** — a pipeline that filters, transforms, and collects data in one fluent chain. Today you learn the tools that separate junior devs from senior devs.

⚙ Streams — Factory Assembly Line (5–20 min)

💡 **ANALOGY**

Raw materials (data) enter at one end. Each station (filter, map, sort) does one thing. Finished products (result) come out the other end. Stations don't know about each other — they just do their job.

💻 Live Coding — Student Pipeline (20–40 min)

```
import java.util.*;
import java.util.stream.*;

record Student(String name, int marks, String dept) {}

public class StreamDemo {
    public static void main(String[] args) {
        List<Student> students = List.of(
            new Student("Priya", 92, "CS"),
            new Student("Ravi",  45, "IT"),
            new Student("Anita", 78, "CS"),
            new Student("Meena", 88, "CS")
        );

        // CS students who passed, sorted by marks desc
        List<String> result = students.stream()
            .filter(s -> s.dept().equals("CS"))
            .filter(s -> s.marks() ≥ 60)
            .sorted((a,b) -> b.marks() - a.marks())
            .map(Student::name)
            .collect(Collectors.toList());

        System.out.println(result);  // [Meena, Anita, Priya]

        double avg = students.stream()
            .mapToInt(Student::marks).average().orElse(0);
        System.out.println("Avg: " + avg);
    }
}
```

🛡 Optional — Never NullPointerException Again

```
// Safe with Optional:
Optional<User> user = repo.findById(id);
user.ifPresent(u -> System.out.println(u.getName()));
String name = user.map(User::getName).orElse("Unknown");
User u = user.orElseThrow(() -> new RuntimeException("Not found"));
```

🎯 You Code It (40–55 min)

1. Create a `Product` record: name, price (double), inStock (boolean), category (String)
2. Add 8 products: mix of Electronics/Clothing, in-stock/out-of-stock, prices ₹200–₹5000
3. Use streams: filter in-stock only, then filter Electronics, sort by price ascending
4. Get the top 3 cheapest in-stock Electronics as a List of names
5. Calculate total value of all in-stock products using `mapToDouble().sum()`
6. **Show:** Print names list and total value

📄 Day 5 Cheat Sheet — Advanced Java

STREAM PIPELINE

list.stream()	Start stream
.filter(x -> cond)	Keep matching
.map(x -> transform)	Transform each
.sorted(comparator)	Sort
.limit(n)	First n items
.collect(toList())	End to list

STREAM TERMINALS

.count()	Count items
.findFirst()	First Optional
.anyMatch(pred)	At least one?
.mapToInt().sum()	Sum ints
.mapToDouble().average()	Average
Collectors.joining(", ")	Join to String

OPTIONAL

Optional.of(value)	Wrap value
Optional.empty()	Empty optional
opt.isPresent()	Has value?
opt.orElse(default)	Safe get
opt.orElseThrow()	Throw if empty
opt.map(fn)	Transform if present

EXCEPTIONS

try { } catch(E e) { }	Handle error
finally { }	Always runs
throw new RuntimeException()	Throw
class MyEx extends RuntimeException	Custom
e.getMessage()	Error text

DAY 6 · 1 HOUR

Database & SQL

Give your app a memory — store and query millions of records in milliseconds.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Zomato stores 50 million orders. When you press "Reorder" it finds your last order from millions in under 100ms. That's SQL with the right indexes.

HDFC Bank processes 10 million transactions daily. Every credit, debit, and balance check is a SQL query. Without databases, apps have no memory. Today you give your app a brain.

📊 Database = Excel with Superpowers (5–20 min)

💡 ANALOGY

A **Table** is like an Excel sheet. Each **row** is one record. Each **column** is a field. A **PRIMARY KEY** is like a unique student ID — no duplicates ever. But unlike Excel, you query millions of rows in milliseconds.

💻 Live Coding — Student Database (20–40 min)

```sql
-- Create the table
CREATE TABLE students (
    id      INT          PRIMARY KEY AUTO_INCREMENT,
    name    VARCHAR(100) NOT NULL,
    email   VARCHAR(150) UNIQUE,
    dept    VARCHAR(50),
    gpa     DECIMAL(3,2) DEFAULT 0.0
);

-- Insert
INSERT INTO students (name, email, dept, gpa)
VALUES ('Priya', 'priya@mail.com', 'CS', 3.9),
       ('Ravi',  'ravi@mail.com',  'IT', 3.2),
       ('Anita', 'anita@mail.com', 'CS', 3.7);

-- Read
SELECT * FROM students WHERE dept = 'CS' AND gpa > 3.5;
SELECT name, gpa FROM students ORDER BY gpa DESC LIMIT 3;

-- Update
UPDATE students SET gpa = 3.95 WHERE name = 'Priya';

-- Aggregate — stats dashboard
SELECT dept, COUNT(*) AS total, AVG(gpa) AS avg_gpa
FROM students
GROUP BY dept
ORDER BY avg_gpa DESC;
```

🔗 JOIN — Connecting Two Tables

```sql
SELECT s.name, c.title
FROM students s
INNER JOIN enrollments e ON s.id = e.student_id
INNER JOIN courses c     ON e.course_id = c.id
WHERE s.dept = 'CS';
```

🎯 You Code It (40–55 min)

1. Create the `students` table in MySQL Workbench
2. Insert at least 8 students from different departments (CS, IT, EC, ME)

3. Query 1: All students with GPA > 3.5, ordered by GPA descending
4. Query 2: Count students per department
5. Query 3: Department with the highest average GPA
6. Query 4: Students whose name contains the letter "a" (use LIKE)
7. **Show:** All 4 query results to instructor

📄 Day 6 Cheat Sheet — SQL

DDL

CREATE TABLE t (...)	New table
ALTER TABLE t ADD col type	Add column
DROP TABLE t	Delete table
PRIMARY KEY	Unique ID
NOT NULL	Required field
UNIQUE	No duplicates

DML

INSERT INTO t (cols) VALUES (vals)	Add row
SELECT * FROM t WHERE cond	Read rows
UPDATE t SET col=val WHERE cond	Edit row
DELETE FROM t WHERE cond	Remove
LIKE '%text%'	Pattern match

SELECT CLAUSES

WHERE col = val	Filter rows
ORDER BY col DESC	Sort
LIMIT 10	First 10
GROUP BY col	Group rows
HAVING COUNT(*) > 2	Filter groups
COUNT(*) AVG() MAX()	Aggregates

JOINS

INNER JOIN	Only matching rows
LEFT JOIN	All left + matching right
ON a.id = b.fk_id	Join condition
AS alias	Rename column/table
FOREIGN KEY	Reference another table

DAY 7 · 1 HOUR

Spring Boot

From 3 days of XML config to 30 seconds — build your first production REST API.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Before Spring Boot: setting up a Java web server took 3 days of XML config, WAR file deployment, and endless boilerplate.

After Spring Boot: **30 seconds**. One annotation and your app is running. Swiggy, Razorpay, Urban Company — all use Spring Boot backends. Today you build your first REST API.

🪝 What Spring Boot Does For You (5–20 min)

💡 ANALOGY

Raw Java is building a car from scratch — engine, seats, wheels, fuel system. Spring Boot is buying a car — it comes configured. You just turn the key (`@SpringBootApplication`) and drive.

💻 Live Coding — Student REST API (20–40 min)

```
// pom.xml dependency
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```java
// Model
public class Student {
    private Long id;
    private String name;
    private String email;
    private String dept;
    // constructors + getters + setters
}

// Controller — maps HTTP requests to methods
@RestController
@RequestMapping("/api/students")
public class StudentController {

    private List<Student> students = new ArrayList<>();
    private Long nextId = 1L;

    @GetMapping
    public List<Student> getAll() {
        return students;
    }

    @GetMapping("/{id}")
    public ResponseEntity<Student> getById(@PathVariable Long id) {
        return students.stream()
            .filter(s -> s.getId().equals(id))
            .findFirst()
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Student> create(@RequestBody Student s) {
        s.setId(nextId++);
        students.add(s);
        return ResponseEntity.status(201).body(s);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable Long id) {
        students.removeIf(s -> s.getId().equals(id));
        return ResponseEntity.noContent().build();
```

```
    }
}
```

🎯 You Code It (40–55 min)

1. Create a Spring Boot project at **start.spring.io** — add "Spring Web" dependency
2. Create a `Product` class: id, name, price (double), category
3. Create `ProductController` with: GET all, GET by id, POST create, DELETE
4. Add GET `/api/products/category/{cat}` — return products filtered by category
5. Test using Postman or `curl http://localhost:8080/api/products`
6. **Show:** All 5 endpoints working in Postman

📄 Day 7 Cheat Sheet — Spring Boot

CORE ANNOTATIONS

@SpringBootApplication	Main class
@RestController	HTTP endpoints
@Service	Business logic
@Repository	Data access
@Component	Generic bean
@Autowired	Inject dependency

HTTP MAPPINGS

@GetMapping	GET request
@PostMapping	POST request
@PutMapping	PUT request
@DeleteMapping	DELETE request
@RequestMapping("/base")	Base path
@PathVariable Long id	URL param

REQUEST/RESPONSE

@RequestBody T obj	JSON body -> object
@RequestParam String x	?x=val query param
ResponseEntity.ok(body)	200 OK with body
ResponseEntity.status(201)	Custom status
ResponseEntity.notFound()	404
ResponseEntity.noContent()	204 No Content

APPLICATION.PROPERTIES

server.port=8080	Change port
spring.datasource.url=...	DB URL
spring.datasource.username=...	DB user
spring.jpa.show-sql=true	Log SQL
spring.jpa.hibernate.ddl-auto=update	Auto schema

DAY 8 · 1 HOUR

Spring Data JPA

Let Spring write your SQL — method names become database queries automatically.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Writing SQL manually for every operation is like writing assembly code instead of Java. You'd write 500 lines where Spring writes 5.

Spring Data JPA automatically generates SQL from method names. `findByDeptAndGpaGreaterThan("CS", 3.5)` — that's your entire query. Used by Infosys, TCS, and every major Java backend team.

@ JPA = Java Persistence API (5–20 min)

💡 ANALOGY

JPA is a translator between Java objects and database tables. You work with **objects**. JPA automatically writes the SQL. `@Entity` tells JPA "this class is a database table". `@Id` marks the primary key.

💻 Live Coding — Student CRUD with JPA (20–40 min)

```
// Entity – maps to a database table
@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;

    @Column(unique = true)
    private String email;

    private String dept;
    private double gpa;
    // getters + setters
}

// Repository – JPA generates SQL automatically
public interface StudentRepository extends JpaRepository<Student, Long> {
    // Spring generates: SELECT * FROM students WHERE dept=? AND gpa>?
    List<Student> findByDeptAndGpaGreaterThan(String dept, double minGpa);

    // SELECT * FROM students ORDER BY gpa DESC
    List<Student> findAllByOrderByGpaDesc();

    // Custom JPQL query
    @Query("SELECT s FROM Student s WHERE s.gpa > :minGpa")
    List<Student> findTopStudents(@Param("minGpa") double minGpa);
}

// Service
@Service
public class StudentService {
    @Autowired private StudentRepository repo;

    public Student save(Student s) { return repo.save(s); }
    public List<Student> getAll()  { return repo.findAll(); }
    public void delete(Long id)    { repo.deleteById(id); }
}
```

🎯 You Code It (40–55 min)

1. Create a new Spring Boot project with dependencies: Spring Web, Spring Data JPA, MySQL Driver
2. Configure `application.properties` with your MySQL database details
3. Create a `Course` entity: id, title, credits, instructor, maxStudents
4. Create `CourseRepository extends JpaRepository` with methods: `findByInstructor()`, `findByCreditsGreaterThan()`
5. Create `CourseController` with full CRUD endpoints
6. **Show:** Create 3 courses via Postman, then GET all — verify they're persisted in MySQL

📄 Day 8 Cheat Sheet — Spring Data JPA

ENTITY ANNOTATIONS

@Entity	Mark as DB table
@Table(name="t")	Custom table name
@Id	Primary key
@GeneratedValue	Auto-increment
@Column(nullable=false)	NOT NULL
@Column(unique=true)	UNIQUE constraint

JPAREPOSITORY METHODS

repo.save(entity)	INSERT or UPDATE
repo.findById(id)	Returns Optional
repo.findAll()	Get all rows
repo.deleteById(id)	Delete by id
repo.count()	Total rows
repo.existsById(id)	Check exists

METHOD NAME QUERIES

findByName(String n)	WHERE name=n
findByNameAndDept	AND condition
findByGpaGreaterThan	WHERE gpa>x
findByNameContaining	LIKE %name%
findAllByOrderByGpaDesc	ORDER BY
countByDept(String d)	COUNT WHERE

CUSTOM QUERY

@Query("SELECT e FROM Entity e...")	JPQL
@Query(nativeQuery=true,"SELECT...")	Raw SQL
@Param("name")	Bind parameter
@Modifying + @Transactional	UPDATE/DELETE query
Page<T> findAll(Pageable p)	Pagination

DAY 9 · 1 HOUR

REST API Advanced

Validation, error handling, pagination — APIs that work like a senior developer wrote them.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 **REAL-WORLD HOOK (0–5 MIN)**

Every time you open Swiggy, 15 different APIs fire in parallel: auth check, restaurant list, your cart, promotions, location, payment methods.

Each one returns JSON. Each one is secured. Each one handles errors gracefully. Today you build APIs like a senior developer — validation, error handling, pagination, and security.

🔒 Professional REST API Design (5–20 min)

💡 **RULES OF GOOD APIS**

✓ Use nouns for resources: `/students` not `/getStudents`
✓ Use HTTP verbs for actions: GET=read, POST=create, PUT=update, DELETE=remove
✓ Return proper status codes: 200=OK, 201=Created, 404=Not Found, 400=Bad Request
✓ Validate input: never trust the client
✓ Return consistent error format

💻 Live Coding — Production-Grade API (20–40 min)

```
// DTO — only expose what clients need
public record StudentRequest(
    @NotBlank String name,
    @Email    String email,
    @Min(1) @Max(4) int year
) {}

// Global exception handler
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorResponse> handleNotFound(Exception ex) {
        return ResponseEntity.status(404)
            .body(new ErrorResponse("NOT_FOUND", ex.getMessage()));
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
handleValidation(MethodArgumentNotValidException ex) {
        String msg = ex.getBindingResult().getFieldErrors().stream()
            .map(e → e.getField() + ": " + e.getDefaultMessage())
            .collect(Collectors.joining(", "));
        return ResponseEntity.badRequest().body(new ErrorResponse("VALIDATION_ERROR",
msg));
    }
}

// Paginated endpoint
@GetMapping
public Page<Student> getAll(
    @RequestParam(defaultValue = "0") int page,
    @RequestParam(defaultValue = "10") int size
) {
    return repo.findAll(PageRequest.of(page, size));
}
```

🎯 You Code It (40–55 min)

1. Add `spring-boot-starter-validation` to your pom.xml
2. Create a `ProductRequest` DTO with: `@NotBlank name`, `@Positive price`, `@NotBlank category`
3. Add `@Valid` to your POST endpoint's `@RequestBody` parameter

4. Create a `GlobalExceptionHandler` class that returns JSON error messages
5. Test: POST with an empty name field — should return 400 with error message
6. Add pagination to GET all products: `?page=0&size=5`

📄 Day 9 Cheat Sheet — REST API

VALIDATION ANNOTATIONS

@NotBlank	Not null & not empty
@NotNull	Not null
@Email	Valid email format
@Min(n) @Max(n)	Number range
@Size(min=2,max=50)	String length
@Positive	Greater than 0

HTTP STATUS CODES

200 OK	Success
201 Created	Resource created
204 No Content	Deleted / empty
400 Bad Request	Invalid input
404 Not Found	Resource missing
500 Server Error	Unexpected crash

EXCEPTION HANDLING

@RestControllerAdvice	Global handler
@ExceptionHandler(Ex.class)	Handle specific
ResponseEntity.badRequest()	400 response
ResponseEntity.status(404)	404 response
ex.getMessage()	Error text

PAGINATION

extends JpaRepository	Already has it
PageRequest.of(page, size)	Create pageable
Page<T> findAll(Pageable p)	Get page
page.getTotalElements()	Total count
page.getTotalPages()	Total pages
?page=0&size=10	Query params

DAY 10 · 1 HOUR

Frontend Development

The UI that users actually touch — HTML, CSS, and JavaScript calling your API.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Your backend API is perfect — but users never touch it directly. They use the frontend. Swiggy's ₹100 crore revenue comes from a beautiful, fast React UI that makes ordering feel effortless.

Today you build a real student dashboard with HTML, CSS, and JavaScript that calls your Spring Boot API — the complete frontend of a full-stack app.

🖥️ The 3-Layer Frontend (5–20 min)

💡 ANALOGY

HTML — The skeleton (structure, bones).
CSS — The skin and clothes (appearance, layout).
JavaScript — The muscles (behaviour, interaction, API calls).

💻 Live Coding — Student Dashboard (20–40 min)

```
<!-- HTML structure -->
<div id="app">
  <h1>Student Dashboard</h1>
  <form id="addForm">
    <input id="nameInput"  placeholder="Name"  required>
    <input id="emailInput" placeholder="Email" type="email">
    <button type="submit">Add Student</button>
  </form>
  <div id="studentList"></div>
</div>
```

```
// JavaScript: fetch API to talk to Spring Boot
const API = 'http://localhost:8080/api/students';

async function loadStudents() {
    const res  = await fetch(API);
    const data = await res.json();
    document.getElementById('studentList').innerHTML =
        data.map(s => `
          <div class="student-card">
            <strong>${s.name}</strong> — ${s.dept}
            <button onclick="deleteStudent(${s.id})">🗑</button>
          </div>
        `).join('');
}

document.getElementById('addForm').addEventListener('submit', async e => {
    e.preventDefault();
    await fetch(API, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            name:  document.getElementById('nameInput').value,
            email: document.getElementById('emailInput').value
        })
    });
    loadStudents();
});

async function deleteStudent(id) {
    await fetch(`${API}/${id}`, { method: 'DELETE' });
    loadStudents();
}

loadStudents();
```

🎯 You Code It (40–55 min)

1. Create `index.html` with a form: product name, price, category
2. Write JavaScript that loads products from your Spring Boot API on page load
3. Display each product as a card with name, price, and a delete button
4. The form submit should POST to your API and reload the list
5. Add a search input that filters the displayed cards by name (client-side, no API call)
6. **Show:** Add 3 products via the form, search for one, delete one

Day 10 Cheat Sheet — Frontend

DOM BASICS

document.getElementById("id")	Get element
element.innerHTML = "..."	Set HTML content
element.value	Input field value
element.style.display	Show/hide
element.classList.add("x")	Add CSS class

FETCH API

fetch(url)	GET request
res.json()	Parse JSON body
method: 'POST'	POST request
headers: {Content-Type: 'application/json'}	JSON header
body: JSON.stringify(obj)	Serialize body
await / async	Handle promises

EVENTS

el.addEventListener('click', fn)	Click event
el.addEventListener('submit', fn)	Form submit
el.addEventListener('input', fn)	On typing
e.preventDefault()	Stop default
e.target.value	Input value

TEMPLATE LITERALS

`Hello ${name}`	Embed variable
arr.map(x => `<li>${x}</li>`).join('')	List to HTML
JSON.stringify(obj)	Object to JSON
JSON.parse(str)	JSON to object
localStorage.setItem(k,v)	Save locally

DAY 11 · 1 HOUR

Full-Stack Integration

Wire frontend to backend — from a collection of code to a working product.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

You have a beautiful frontend. You have a powerful Spring Boot API. They're not talking to each other yet.

This is what separates a *collection of code* from a *working product*. Today you wire them together: React/HTML frontend → Fetch API → Spring Boot controller → JPA → MySQL. Real full-stack.

🔗 The Full-Stack Data Flow (5–20 min)

💡 FLOW DIAGRAM

👤 User fills form → 📄 JavaScript fetch() → 🚀 Spring Boot Controller → ⚙️ Service → 🗄️ JPA Repository → 📊 MySQL Database

Response flows back the same way: Database row → Java object → JSON → Browser renders it.

💻 Live Coding — Fix CORS + Wire Frontend (20–40 min)

```
// CORS config - allow frontend to call your API
@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://localhost:3000", "http://localhost:5500")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowedHeaders("*");
    }
}
```

```
// Full-stack: frontend script calling real API
const API_BASE = 'http://localhost:8080/api';

async function loadDashboard() {
    const [students, courses] = await Promise.all([
        fetch(`${API_BASE}/students`).then(r => r.json()),
        fetch(`${API_BASE}/courses`).then(r => r.json())
    ]);

    document.getElementById('student-count').textContent = students.length;
    document.getElementById('course-count').textContent  = courses.length;

    document.getElementById('student-list').innerHTML =
        students.slice(0, 5).map(s => `
          <tr>
            <td>${s.name}</td>
            <td>${s.dept}</td>
            <td><span class="badge">${s.gpa}</span></td>
          </tr>
        `).join('');
}

loadDashboard();
```

🎯 You Code It (40–55 min)

1. Add the `CorsConfig` class to your Spring Boot project
2. Restart the backend — confirm it runs on port 8080
3. Create `dashboard.html` with 3 sections: stats cards, student table, add-student form
4. Load stats (total students, average GPA) by calling your API
5. The form should POST and instantly refresh the table without page reload
6. **Show:** Dashboard loading real data from your MySQL via Spring Boot

📄 Day 11 Cheat Sheet — Full-Stack

CORS FIX

@CrossOrigin("*")	Quick fix on controller
WebMvcConfigurer	Global CORS config
allowedOrigins("url")	Allowed frontend
allowedMethods("GET","POST"...)	HTTP verbs allowed

PARALLEL FETCH

Promise.all([...])	Run all in parallel
await Promise.all([f1,f2])	Wait for all
const [a,b] = await...	Destructure results
Promise.allSettled	Get all even if some fail

DEBUGGING TOOLS

F12 -> Network tab	See API calls
F12 -> Console	JS errors
Postman	Test API directly
spring.jpa.show-sql=true	See SQL queries
console.log(data)	Debug JS variables

LAYERED ARCHITECTURE

Controller	HTTP in/out, no logic
Service	Business rules here
Repository	Database access only
Entity	Database table model
DTO	What client sees

DAY 12 · 1 HOUR

Capstone Project

Build a complete Student Management System to show in interviews — from scratch.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 **REAL-WORLD HOOK (0–5 MIN)**

In every job interview, they ask: "Show me something you built." Not a tutorial. Not a copy-paste project. *Something you built from scratch.*

Today you build your capstone: a complete Student Management System with frontend + Spring Boot + MySQL. This goes on your GitHub. This is what you show in interviews.

🌟 What You'll Build (5–20 min)

💡 **CAPSTONE SPEC**

Student Management System

✓ Add / edit / delete students (name, email, dept, GPA, year)
✓ View all students in a table with search
✓ Dashboard: total students, average GPA, students per dept
✓ Spring Boot backend (port 8080) + MySQL + React or HTML frontend
✓ Deployed and accessible via URL

💻 Live Coding — Project Structure (20–40 min)

```
student-management/
├─ backend/
│   ├─ src/main/java/com/internship/
│   │   ├─ entity/Student.java
│   │   ├─ repository/StudentRepository.java
│   │   ├─ service/StudentService.java
│   │   └─ controller/StudentController.java
│   └─ src/main/resources/application.properties
└─ frontend/
    ├─ index.html      (student list + search)
    ├─ dashboard.html  (stats)
    └─ app.js          (all fetch calls)
```

```java
// StudentService.java – business logic layer
@Service
public class StudentService {
    @Autowired private StudentRepository repo;

    public Student createStudent(Student s) {
        if (repo.existsByEmail(s.getEmail()))
            throw new RuntimeException("Email already registered");
        return repo.save(s);
    }

    public Map<String, Object> getDashboardStats() {
        long total = repo.count();
        double avgGpa = repo.findAll().stream()
            .mapToDouble(Student::getGpa).average().orElse(0);
        return Map.of("total", total, "avgGpa", avgGpa);
    }
}
```

🎯 You Code It (40–55 min) — Build the Capstone

1. Create the folder structure above in IntelliJ
2. Copy your `Student` entity, `StudentRepository`, and `CorsConfig` from earlier days
3. Build `StudentService` with: createStudent (check duplicate email), getDashboardStats
4. Build `StudentController` with all CRUD + `GET /api/stats`
5. Build `index.html`: table of all students + search box + delete buttons
6. Build `dashboard.html`: stats cards pulling from `/api/stats`

7. **Goal:** Complete working app with at least 10 students in the DB

📄 Day 12 Cheat Sheet — Capstone Architecture

LAYERS

Controller	HTTP in/out only
Service	All business logic
Repository	DB queries only
Entity	DB table structure
DTO	API request/response shape

COMMON PATTERNS

repo.existsByEmail(e)	Unique check
repo.findByDept(d)	Filter query
repo.findAll(Sort.by("gpa").descending())	Sorted list
Map.of("k",v,"k2",v2)	Quick JSON map
@Transactional	DB transaction

PROJECT CHECKLIST

✅ Entity with @Id, @Column	DB model
✅ Repository extends JpaRepository	CRUD free
✅ Service with business rules	Logic layer
✅ Controller with validation	API layer
✅ CorsConfig for frontend	Allow browser
✅ Frontend calls API	Full stack

INTERVIEW QUESTIONS

Why separate Service layer?	Testable business logic
What is ORM?	Java objects <-> DB tables
What is REST?	Stateless HTTP API standard
What is CORS?	Browser security for cross-origin
Explain @Transactional	All-or-nothing DB ops

DAY 13 · 1 HOUR

Firebase Deployment

"It works on my machine" is a joke — today your app gets a real URL anyone can open.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Your app works on localhost — but no one can use it. "It works on my machine" is the biggest joke in software.

Today you deploy your app so anyone in the world can open it from their phone. Frontend goes to Firebase Hosting (free, fast CDN). Backend goes to Google Cloud Run. By the end you'll have a real URL to share.

🌐 Deployment Architecture (5–20 min)

💡 WHERE EACH PART LIVES

Frontend HTML/CSS/JS → Firebase Hosting (free, global CDN)
Spring Boot API → Docker container → Google Cloud Run (pay per request)
MySQL Database → Google Cloud SQL or PlanetScale (managed database)

💻 Live Coding — Deploy Frontend to Firebase (20–40 min)

```
# Step 1: Install Firebase CLI
npm install -g firebase-tools

# Step 2: Login
firebase login

# Step 3: Initialize project
firebase init hosting

# Step 4: firebase.json config
```

```
{
  "hosting": {
    "public": "frontend",
    "ignore": ["firebase.json", "**/.*"],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

```
# Step 5: Build & Deploy
firebase deploy --only hosting

# Step 6: Dockerize Spring Boot backend
```

```
# Dockerfile
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

```
# Build and push to Google Container Registry
mvn clean package -DskipTests
docker build -t gcr.io/YOUR_PROJECT/student-api .
docker push gcr.io/YOUR_PROJECT/student-api

# Deploy to Cloud Run
gcloud run deploy student-api   --image gcr.io/YOUR_PROJECT/student-api   --platform
managed   --region us-central1   --allow-unauthenticated
```

🎯 You Code It (40–55 min)

1. Install Firebase CLI: `npm install -g firebase-tools`
2. Run `firebase login` and authenticate with your Google account
3. In your frontend folder, run `firebase init hosting` and select your project
4. Update your frontend's API_BASE URL to your Cloud Run URL (or keep localhost for now)

5. Run `firebase deploy --only hosting`
6. Open the Firebase URL — your app is live!
7. **Show:** Share the Firebase URL with the instructor

📄 Day 13 Cheat Sheet — Deployment

FIREBASE CLI

npm install -g firebase-tools	Install
firebase login	Authenticate
firebase init hosting	Setup project
firebase deploy	Deploy all
firebase deploy --only hosting	Frontend only
firebase open hosting:site	Open deployed site

DOCKER COMMANDS

docker build -t name .	Build image
docker run -p 8080:8080 name	Run container
docker images	List images
docker ps	Running containers
docker push gcr.io/project/name	Push to registry

CLOUD RUN

gcloud run deploy name	Deploy service
--image gcr.io/project/name	Container image
--allow-unauthenticated	Public access
--region asia-south1	Mumbai region
gcloud run services list	View services

APPLICATION.PROPERTIES (PROD)

spring.datasource.url=${DB_URL}	Env variable
spring.profiles.active=prod	Activate profile
server.port=${PORT:8080}	Cloud Run port
logging.level.root=WARN	Reduce log noise

DAY 14 · 1 HOUR

LLM for Development

GitHub Copilot made devs 55% faster — learn to use AI as your pair programmer.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

GitHub Copilot made developers **55% faster** in Microsoft's study. Developers who use AI write better tests, catch more bugs, and deliver features faster.

But the developers who get replaced aren't those who use AI — they're those who don't. Today you learn to use LLMs as a pair programmer that never sleeps and knows every framework.

🤖 LLM as Your Coding Partner (5–20 min)

💡 MENTAL MODEL

Think of an LLM as a very senior developer who:
✓ Has read every Stack Overflow answer ever written
✓ Knows Java, Spring, SQL, React, Docker, Git
✓ Never judges your questions
✓ But can be wrong — always verify critical code

Your job: ask precise questions. Get precise answers.

💻 Live Coding — Build a Feature with AI (20–40 min)

```
-- Prompt technique 1: Precise context
"I have a Spring Boot app with a Student entity (id, name, email, dept, gpa).
Write a JPA repository method that finds all students in a given department
with GPA above a threshold, sorted by GPA descending."

-- Prompt technique 2: Explain existing code
"Explain this Spring Boot code line by line:
[paste your code]"

-- Prompt technique 3: Debug with AI
"I'm getting this error: [paste error]
Here's my code: [paste code]
What's wrong and how do I fix it?"

-- Prompt technique 4: Generate tests
"Write JUnit 5 tests for this Spring Boot service method:
[paste method]
Include edge cases: null input, empty list, duplicate email."
```

🔭 Build an AI-Powered Chat Feature

```java
// Spring Boot endpoint calling Groq API (free LLM)
@Service
public class AiService {
    @Value("${groq.api.key}")
    private String apiKey;

    private final RestTemplate rest = new RestTemplate();

    public String ask(String question) {
        var headers = new HttpHeaders();
        headers.setBearerAuth(apiKey);
        headers.setContentType(MediaType.APPLICATION_JSON);

        var body = Map.of(
            "model", "llama3-8b-8192",
            "messages", List.of(Map.of("role", "user", "content", question))
        );

        var resp = rest.postForObject(
            "https://api.groq.com/openai/v1/chat/completions",
            new HttpEntity<>(body, headers),
            Map.class
        );
        return ((Map) ((List) resp.get("choices")).get(0))
            .get("message").toString();
    }
}
```

🎯 You Code It (40–55 min)

1. Sign up at **console.groq.com** — get a free API key (faster than OpenAI)
2. Add the `AiService` class to your capstone project
3. Create an endpoint `POST /api/ask` that accepts a question and returns an AI answer
4. Add a chat box to your frontend HTML — input + submit button + answer area
5. Connect it to your `/api/ask` endpoint using fetch
6. Test: Ask "What are the top 3 Java interview questions?" — AI answers in your app!

📄 Day 14 Cheat Sheet — LLM for Dev

BEST PROMPT PATTERNS

FREE LLM APIS

Give context first	"I have a Spring Boot app..."
Be specific	"Write JPA method for..."
Include constraints	"...sorted by GPA desc"
Paste the error	"I get this error: ..."
Ask for tests	"Write JUnit tests for..."
Ask for explanation	"Explain line by line"

Groq API	Fastest, free tier, Llama3
Claude API	Best for code, Anthropic
Gemini API	Google, free tier
Ollama (local)	Run on your laptop
GitHub Copilot	IDE integration, student free

AI TOOLS FOR JAVA DEV

GitHub Copilot	Code autocomplete in IDE
Claude.ai	Architecture & debugging
ChatGPT	General questions
Cursor IDE	AI-first code editor
TabNine	ML autocomplete

WHEN NOT TO TRUST AI

Security-critical code	Review manually
Database migrations	Test in dev first
Payment logic	Write yourself + test
Cutting-edge APIs	AI may be outdated
Business logic	You know the domain

DAY 15 · 1 HOUR

Industry Simulation

Day 1 at a real company — pick up a ticket, fix the bug, write tests, open a PR.

0–5 min Real Hook	5–20 min Core Concept	20–40 min Live Coding	40–55 min You Code It	55–60 min Cheat Sheet

🔥 REAL-WORLD HOOK (0–5 MIN)

Your first day at Infosys, TCS, or a startup: your manager opens a Jira board, assigns you a ticket, and says "Fix this bug by EOD."

No one explains everything. No one holds your hand. You read the code, understand the system, fix the bug, write a PR, and get it reviewed. Today we simulate exactly that. This is Day 1 at a real company.

📋 The Real Developer Workflow (5–20 min)

💡 INDUSTRY WORKFLOW

1. **Pick up ticket** → Read description on Jira/GitHub Issues
2. **Create branch** → `git checkout -b feature/SMS-101-add-search`
3. **Write code** → Implement, test locally
4. **Write tests** → Unit tests pass
5. **Create PR** → Pull Request with description
6. **Code review** → Address reviewer comments
7. **Merge** → CI/CD deploys automatically

💻 Live Coding — Simulate Fixing a Bug (20–40 min)

```
// TICKET SMS-101: BUG – Search endpoint returns 500 when query is empty
// Expected: return all students when query is empty
// Actual: NullPointerException in StudentService.search()

// BROKEN code (current main branch):
public List<Student> search(String query) {
    return repo.findAll().stream()
        .filter(s → s.getName().contains(query))  // crashes if query=null
        .collect(Collectors.toList());
}

// FIXED code (your branch: feature/SMS-101-fix-search):
public List<Student> search(String query) {
    if (query == null || query.isBlank()) return repo.findAll();
    return repo.findByNameContainingIgnoreCase(query);
}
```

```
// Unit test you write alongside the fix
@SpringBootTest
class StudentServiceTest {
    @Autowired private StudentService service;

    @Test
    void searchWithNullQueryReturnsAll() {
        List<Student> result = service.search(null);
        assertThat(result).isNotNull();
    }

    @Test
    void searchWithValidQueryFiltersResults() {
        List<Student> result = service.search("Priya");
        assertThat(result).allMatch(s → s.getName().contains("Priya"));
    }
}
```

🎯 Industry Simulation (40–55 min)

1. **Ticket 1 — New Feature:** Add `GET /api/students/top?limit=5` — returns top N students by GPA
2. Create branch: `git checkout -b feature/add-top-students-endpoint`
3. Implement, write 2 unit tests, commit with message: "feat: add top N students endpoint"

4. **Ticket 2 — Code Review:** Review your partner's code — leave 2 improvement comments
5. **Ticket 3 — Deploy:** Merge your branch to main, run `firebase deploy`
6. **Final Show:** Demo your full capstone — live URL, working features, GitHub repo with commit history

📄 Day 15 Cheat Sheet — Industry Simulation

GIT WORKFLOW

git checkout -b feature/X	Start new feature
git add . && git commit	Save progress
git push origin feature/X	Push branch
gh pr create	Open pull request
git checkout main && git pull	Get latest main
git merge feature/X	Merge feature

COMMIT CONVENTION

feat: add search endpoint	New feature
fix: handle null query	Bug fix
refactor: extract service	Restructure
test: add search tests	Tests only
docs: update README	Documentation
chore: update dependencies	Maintenance

JUNIT 5 TESTS

@SpringBootTest	Full context test
@WebMvcTest	Controller layer only
@Test void name() {}	Test method
assertThat(x).isEqualTo(y)	Assert equality
assertThat(list).hasSize(n)	List size
assertThat(x).isNotNull()	Not null

WHAT INTERVIEWERS CHECK

GitHub profile	Real commit history
Project complexity	Full-stack preferred
Code organisation	Proper layers
Tests present	Even 2-3 tests
README.md	How to run it
Live URL	Firebase/Heroku link

Java Full-Stack — Real-World Projects

14 production-grade projects you can build with your Spring Boot + MySQL + REST API skills.

💡 HOW TO USE THIS LIST

Pick one project per difficulty level. Start Beginner → Intermediate → Advanced. Each project goes on GitHub with a README. That's your portfolio. Recruiters at product companies (Zoho, Freshworks, Razorpay) look for this exact combination.

▲ BEGINNER — BUILD CONFIDENCE (1–2 WEEKS EACH)

PROJECT 01

Student Grade Manager

CRUD portal — add students, record subject marks, auto-calculate grades and GPA. Includes admin dashboard with grade distribution chart.

Spring Boot | REST API | MySQL | JPA/Hibernate | + Thymeleaf UI

BEGINNER — 1 week

PROJECT 02

Library Book System

Track book inventory, borrow/return with due-dates, auto-calculate fine per day. Email receipt on borrow using JavaMailSender.

Spring Boot | MySQL | JPA | REST API | + JavaMailSender

BEGINNER — 1 week

PROJECT 03

Employee Directory

Department-wise employee listing with search, filter by role/city, pagination. Export to CSV. Role-based: HR admin vs. view-only.

Spring Boot | Spring Security | JWT | MySQL | + CSV export

BEGINNER — 1–2 weeks

▶ INTERMEDIATE — INDUSTRY-READY (2–3 WEEKS EACH)

PROJECT 04

Hospital Appointment Booking

Patients book slots with doctors by specialty. Doctors confirm/reschedule. Email notification on booking. JWT auth for 3 roles.

Spring Boot | JWT | JPA | REST API | MySQL | + Email notifications

INTERMEDIATE — 2–3 weeks

PROJECT 05

Online Quiz Platform

Create MCQ question banks by topic. Students attempt timed quizzes. Real-time scoring, attempt history, leaderboard by batch.

Spring Boot | JPA | REST API | MySQL | + WebSocket leaderboard

INTERMEDIATE — 2–3 weeks

PROJECT 06

E-commerce Product API

Product catalog with categories, cart management, order lifecycle (placed → shipped → delivered). Razorpay/Stripe webhook for payment.

Spring Boot | JPA | REST API | MySQL | JWT | + Payment gateway

INTERMEDIATE — 3 weeks

PROJECT 07

Food Delivery Tracker

Restaurant menus, customer orders, delivery partner assignment, live status updates (ordered → preparing → on-the-way → delivered).

Spring Boot | MySQL | JPA | JWT | + WebSocket status push

INTERMEDIATE — 2–3 weeks

PROJECT 08

Bill Splitting App

Create groups, add shared expenses, auto-split by equal/percentage/custom. View who owes whom. Settle up and mark paid.

Spring Boot | REST API | MySQL | JPA | + Settlement algorithm

INTERMEDIATE — 2 weeks

PROJECT 09

Job Portal Backend

Employers post jobs, candidates apply and upload resume. Filter by skill/location/salary. Application status pipeline with email alerts.

Spring Boot | JWT | JPA | REST API | MySQL | + File upload (S3)

INTERMEDIATE — 3 weeks

★ ADVANCED — PORTFOLIO SHOWCASERS (3–5 WEEKS EACH)

PROJECT 10

Bank Transaction System

Accounts, fund transfers with ACID transactions, mini-statement, fraud flag if >3 large txns/hour. Full audit log trail.

Spring Boot | JPA | MySQL | JWT | REST API | + ACID transactions

ADVANCED — 3–4 weeks

PROJECT 11

Hotel Room Booking

Room types, seasonal pricing, availability calendar, booking confirmation PDF, cancellation with refund policy logic.

Spring Boot | JPA | MySQL | JWT | + PDF generation

ADVANCED — 3–4 weeks

PROJECT 12

Real-time Chat API

One-on-one and group chats, message history stored in DB, read receipts, online/offline status using WebSocket (STOMP).

Spring Boot | MySQL | JPA | JWT | + WebSocket + STOMP

ADVANCED — 3–4 weeks

PROJECT 13

Social Media REST API

User profiles, posts with images, likes/comments, follow system, personalised feed (chronological + engagement score).

Spring Boot | JPA | MySQL | JWT | REST API | + Feed ranking algorithm

ADVANCED — 4 weeks

PROJECT 14

Multi-tenant SaaS Backend

Multiple organisations in one deployment, tenant-isolated data, subscription tiers, per-tenant role hierarchy, usage metering.

Spring Boot | JPA | MySQL | JWT | Spring Security | + Multi-tenancy pattern

ADVANCED

🕐 4–5 weeks

🏆 Skills You Practice Across All Projects

REST API DESIGN Every project	**JWT + SPRING SECURITY** Projects 3,4,5,6,7,9,10,11,12,13,14	**JPA + MYSQL** All projects
EMAIL / NOTIFICATIONS Projects 2,4,9	**FILE UPLOAD** Projects 9,11	**WEBSOCKET REAL-TIME** Projects 5,12
PAYMENT INTEGRATION Project 6	**PDF GENERATION** Project 11	