FULL STACK DEV

spring boot

# Spring Boot Architecture

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

# SPRING BOOT ECOSYSTEM

- Allow developers to focus more on business logic rather than boilerplate setup.

- Key features of Spring Boot:
    - **Auto-configuration**: Automatically configures your Spring application based on the JARs on your classpath.

    - **Standalone**: Embeds a web server like Tomcat  directly, so you can just run your application as a JAR.

    - **No XML configuration**: Largely relies on annotations and convention over configuration.

# SPRING BOOT ANNOTATIONS

- **@SpringBootApplication** // This is the magic annotation!
  - Tells Spring to look for other components & configurations, allowing it to find your controllers, services, etc.

- **FrontEnd** <=> (Controller -> Service -> Repository -> Database)

- SpringApplication.run(.class, args)
  - This static method is responsible launching a Spring application from a Java main method

## ESSENTIAL DEPENDENCIES

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

- <dependency>
    - <groupId>org.springframework.boot</groupId>
    - <artifactId>spring-boot-starter-test</artifactId>
    - <scope>test</scope>
- </dependency>

# APPLICATION.PROPERTIES

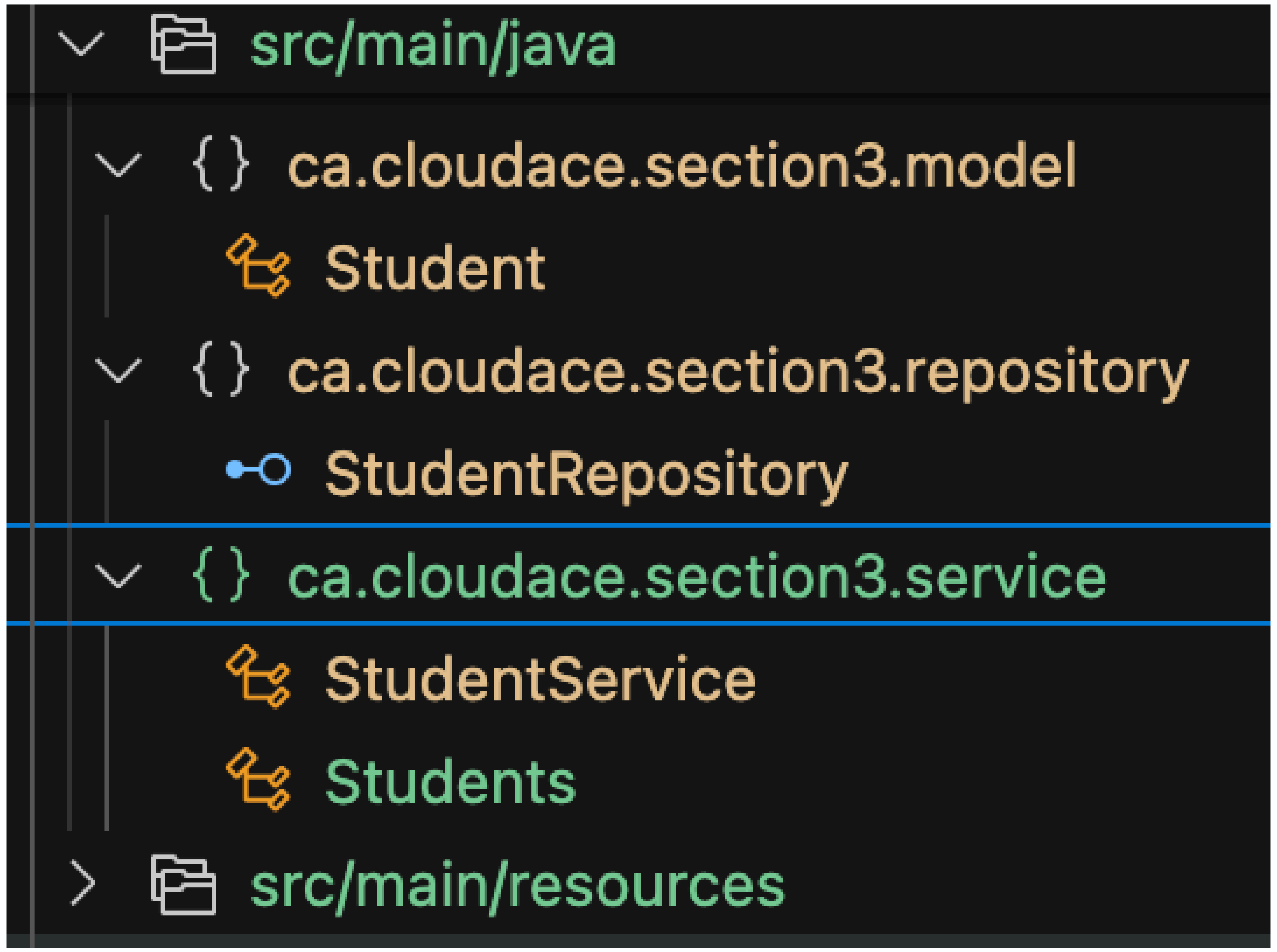
- Spring Boot uses **application.properties** to manage configuration.
  - In src/main/resources
    - application.properties

- When you work in different environment, you can have different versions of this file like :
  - application-dev.properties
    - for development
  - application-prod.properties
    - for production

## STRUCTURE OF YOUR APP

- controller
- service
- repository

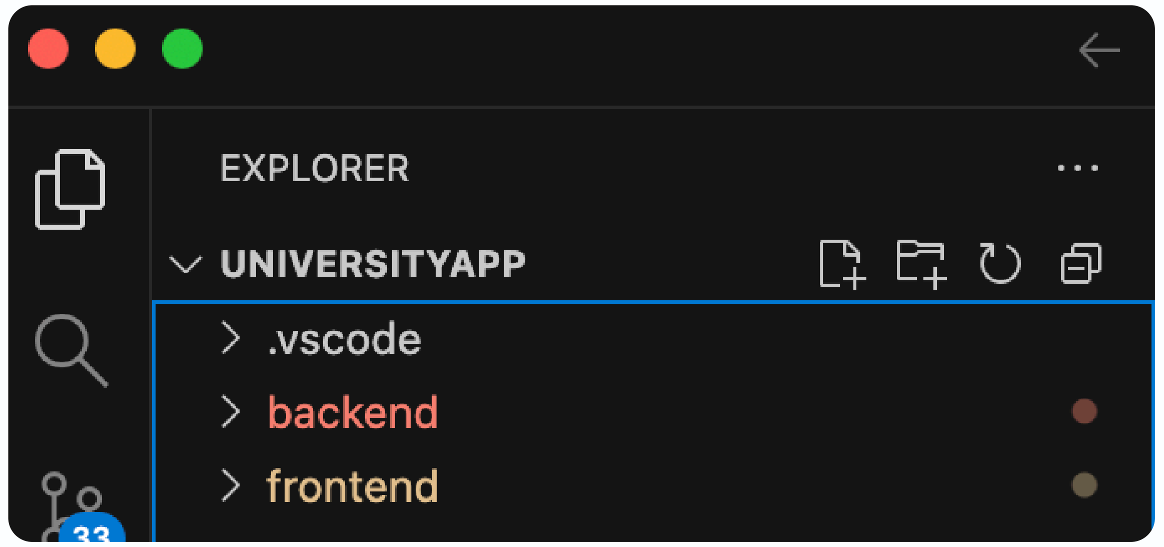
- package name :
  - ca.cloudace
- arficactId :
  - section3



```
src/main/java
  ca.cloudace.section3.model
    Student
  ca.cloudace.section3.repository
    StudentRepository
  ca.cloudace.section3.service
    StudentService
    Students
src/main/resources
```

## YOUR FIRST APP - PROJECT #1 : UNIVERSITYAPP

- Create a folder named : git
- Inside this you create another folder : universityapp
- Inside university app, there will be two projects :
  - backend
  - frontend



- So, overall :
- git
  - universityapp
    - **backend** (spring boot app) → a git repo
    - frontend (one of *angular*, *react*, *vue* → see later sections)

FULL STACK DEV

spring boot

# Writing REST Endpoints

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

## WHAT ARE CONTROLLERS ?

- Controllers handle incoming HTTP requests and send responses.

- Spring Boot offers two main controller types:
    - @Controller – For traditional web apps (HTML pages).
    - @RestController – For REST APIs (JSON / XML).

- Both are used to separate presentation logic from business logic.
    -

# MVC CONTROLLER

- @Controller (Web or MVC Controller)
- Used in traditional web applications that return HTML pages.
- Works with Thymeleaf, JSP, or other template engines.
- Returns view names, not raw data.

```
@Controller
public class StudentController {
@GetMapping("/students")
 public String listStudents(Model model){
 return "home"; // Resolved to home.html
}
}
```

**http://localhost:8080/students**

Rajeev Khoodeeram

# REST CONTROLLER

- Used in RESTful APIs that return JSON or XML responses.
- Commonly used for frontend-backend communication or mobile app APIs.
- Combines @Controller + @ResponseBody, so it returns data directly.

```
@RestController
@RequestMapping("/api/students")
public class StudentController {
@GetMapping
 public ResponseEntity<List<Student>> getAllStudents() {
 List<Student> students = studentService.getAllStudents();
 return new ResponseEntity<>(students, HttpStatus.OK);
}
}
```

**http://localhost:8080/api/students**

Rajeev Khoodeeram

## OTHER ANNOTATIONS

- @PostMapping
  - Create a new record in database
- @PutMapping
  - Update an existing record using its primary key
- @DeleteMapping
  - Delete an existing record using its primary key

FULL STACK DEV

spring boot

Using postman

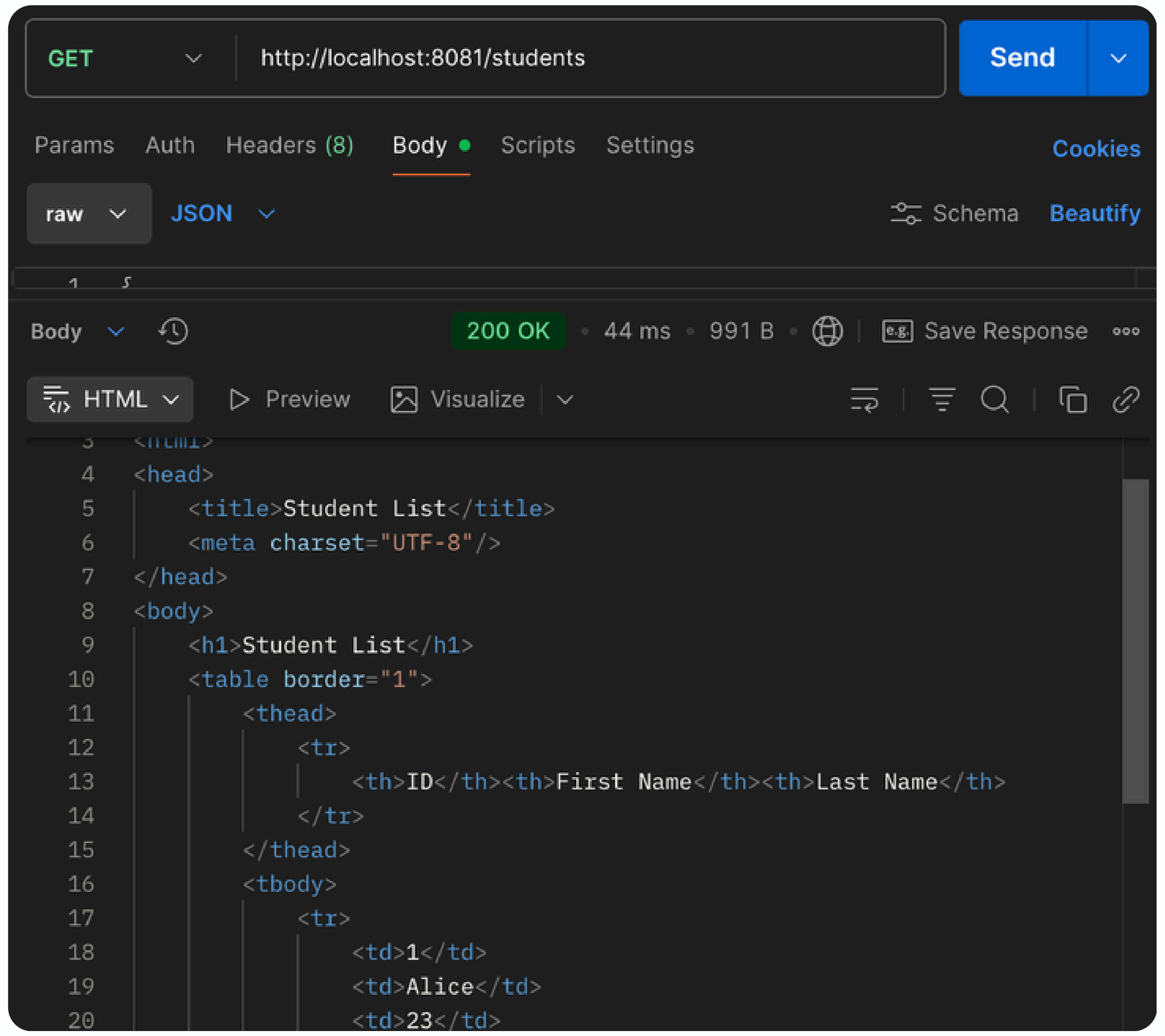
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

## USING POSTMAN FOR API TESTING

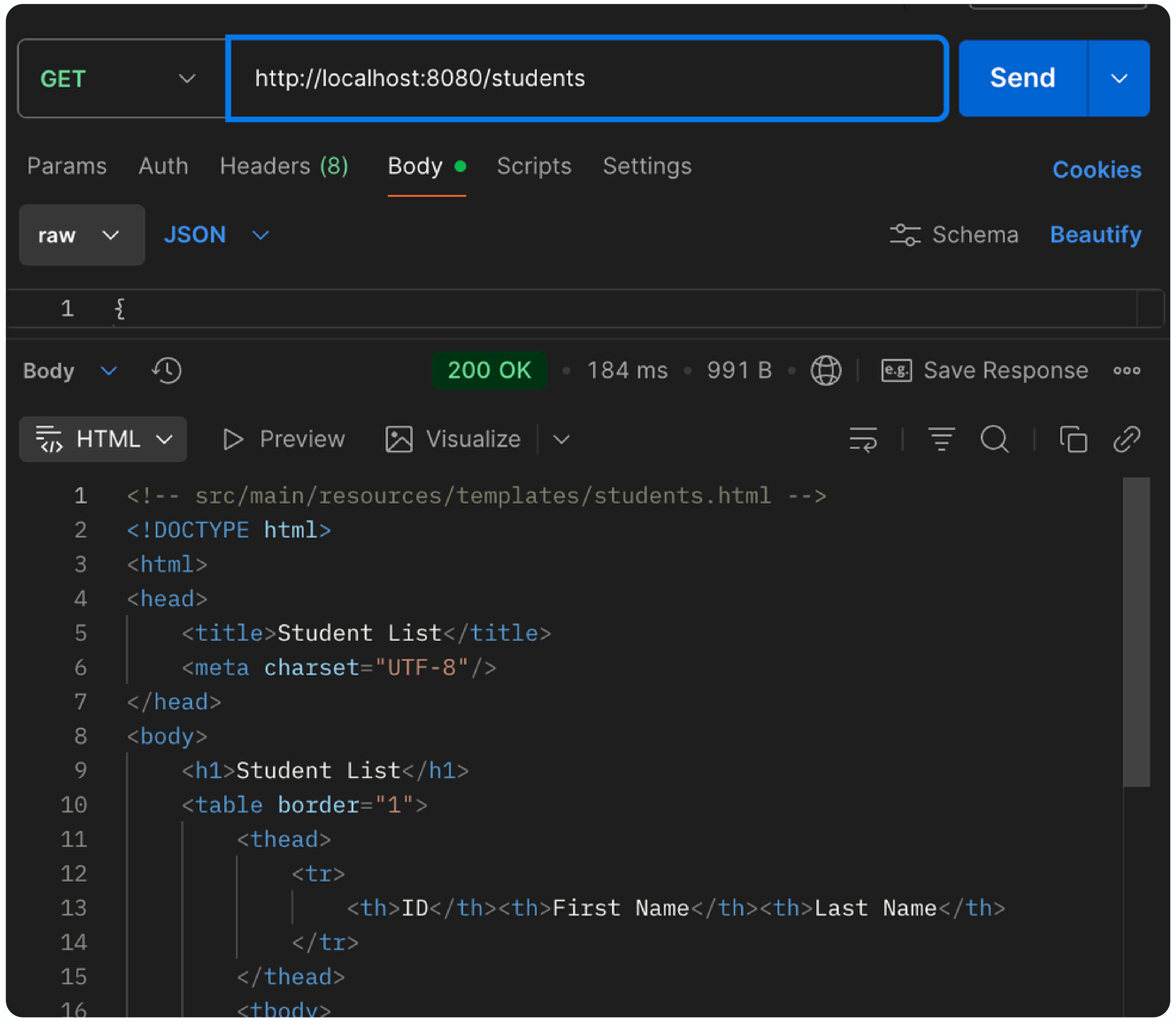
- Postman is a GUI tool to test and interact with APIs.
- Allows sending HTTP requests: GET, POST, PUT, DELETE, etc.
- Useful for testing REST APIs during development.
- Can test local apps (localhost) or deployed APIs.



http://localhost:8081/students

```
<head>
    <title>Student List</title>
    <meta charset="UTF-8"/>
</head>
<body>
    <h1>Student List</h1>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th><th>First Name</th><th>Last Name</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>Alice</td>
                <td>23</td>
```

# GET API CALLS



GET http://localhost:8080/students Send

Params Auth Headers (8) Body Scripts Settings Cookies

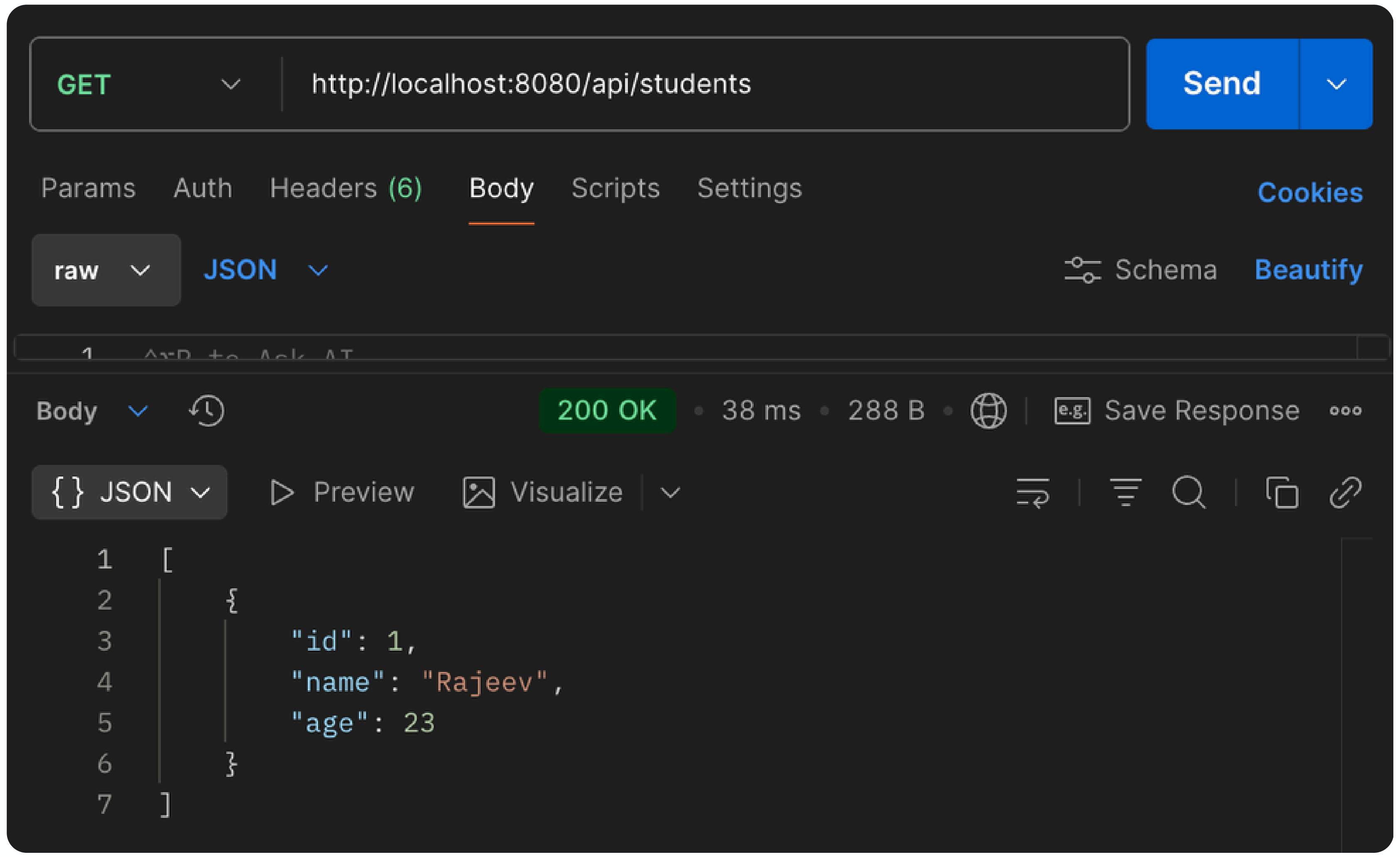
raw JSON Schema Beautify

```
{
```

Body 200 OK 184 ms 991 B Save Response

HTML Preview Visualize

```
<!-- src/main/resources/templates/students.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Student List</title>
    <meta charset="UTF-8"/>
</head>
<body>
    <h1>Student List</h1>
    <table border="1">
        <thead>
            <tr>
                <th>ID</th><th>First Name</th><th>Last Name</th>
            </tr>
        </thead>
        <tbody>
```



GET http://localhost:8080/api/students Send

Params Auth Headers (6) Body Scripts Settings Cookies

raw JSON Schema Beautify

Body 200 OK 38 ms 288 B Save Response

{ } JSON Preview Visualize

```
[
    {
        "id": 1,
        "name": "Rajeev",
        "age": 23
    }
]
```

FULL STACK DEV

spring boot

## Application properties

Presented by:

**Rajeev Khoodeeram**

OCTOBER 2025

## APPLICATION.PROPERTIES

- Spring Boot uses application.properties to manage configuration.
- We’ll learn to customize settings like server port, database connections, and logging.
- Understanding this file lets you tailor your app for different environments easily. By environments, we mean : developer, testing, etc
- application.properties (or application.yml) is the heart of Spring Boot's externalized configuration.

## SERVER CONFIGURATION (PORT, CONTEXT PATH)

- This is one of the most basic but crucial configurations.

- Scenario: You want your application to run on a port other than the default 8080, or you want to add a context path.

- # Server port (default is 8080)
- server.port=9090

- # Context path for the application (e.g.,http://localhost:9090/my-app/hello)
- server.servlet.context-path=/my-app

## PROFILE-SPECIFIC PROPERTIES

- Scenario: Different database settings or API endpoints for dev vs. prod.
- Files:
  - src/main/resources/application.properties (default/common properties)
  - src/main/resources/application-dev.properties
  - src/main/resources/application-prod.properties

- **application.properties (Default):**
- 
- # Default settings
- app.environment=Default
- server.port=8080
- spring.datasource.url=jdbc:h2:mem:defaultdb

# APPLICATION.PROPERTIES

- **application-dev.properties:**
- 
- # Development specific settings
- app.environment=Development
- server.port=8081 # Dev runs on a different port
- spring.datasource.url=jdbc:h2:mem:devdb

## APPLICATION.PROPERTIES

- **application-prod.properties:**

- # Production specific settings
- app.environment=Production
- server.port=8080 # Prod might use default or a specific external port
- spring.datasource.url=jdbc:mysql://prod-db:3306/prod_db
- spring.jpa.hibernate.ddl-auto=none
  - # Don't auto-create schema in prod

## WHICH ONE IS USED BY SPRING BOOT ?

- Spring Boot checks for the property:
  - **spring.profiles.active=dev**
- This can be set in several ways:
- **Option 1** – Inside application.properties
  - This will make Spring load application-dev.properties in addition to the default one.
- **Option 2** – Using Command Line
  - When you run your app:
    - java -jar myapp.jar --spring.profiles.active=prod
- **Option 3** – In your IDE (e.g., IntelliJ or VS Code)
  - Add it to Run Configuration → VM Options:
    - -Dspring.profiles.active=prod

FULL STACK DEV

spring boot

## Complete REST implementation

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

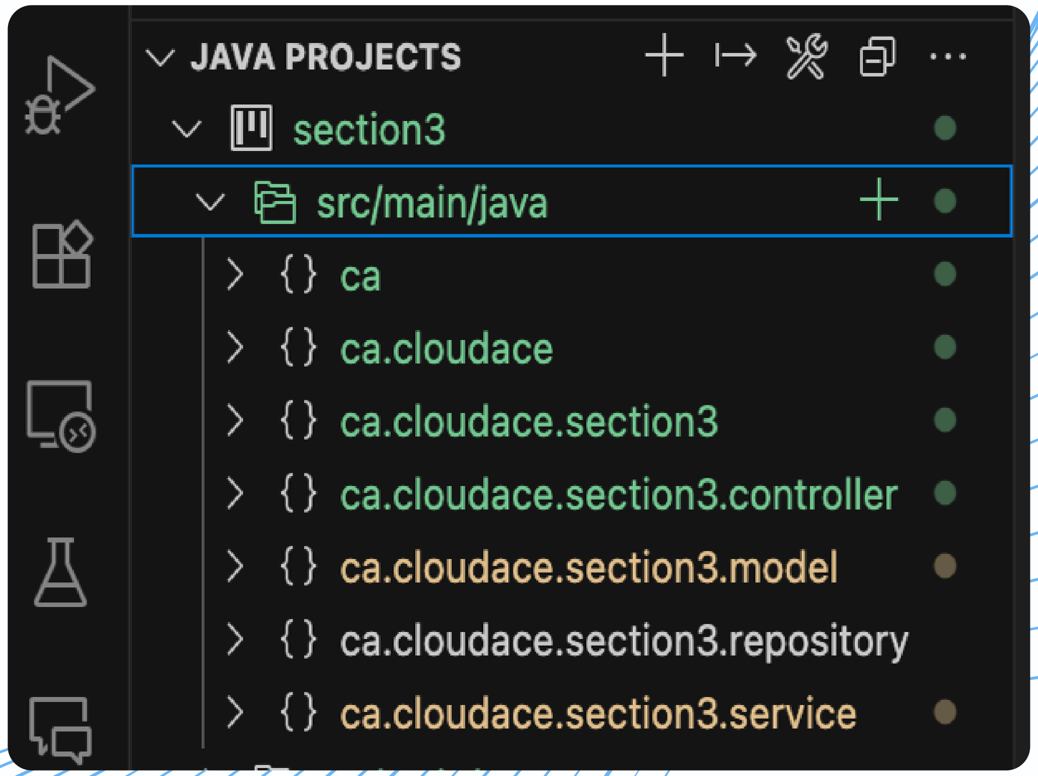
# SERVICE LAYER PATTERN

- **Controller**: Handles HTTP requests, delegates business logic to the service layer, and returns HTTP responses.

- **Service**: Contains the core business logic. It orchestrates operations, potentially involving multiple repositories.

- **Repository**: Handles direct database interaction.

- **Model** = Entity = Database table



```
JAVA PROJECTS
  section3
    src/main/java
      {} ca
      {} ca.cloudace
      {} ca.cloudace.section3
      {} ca.cloudace.section3.controller
      {} ca.cloudace.section3.model
      {} ca.cloudace.section3.repository
      {} ca.cloudace.section3.service
```

# CONTROLLER

- Handles HTTP requests from clients.
- Routes requests to the service layer.
- Can return views (HTML) or data (JSON/XML).
- 
- @RestController
- @RequestMapping("/api/students")
  - //@CrossOrigin(origins = "http://localhost:4200") // Allow CORS for frontend requests; for ANGULAR frontend
  - // use port 5175 for REACT or port 5176 for VUE as frontend

- public class StudentController {
- // If you are using a service layer, you can inject it here to handle database operations
- @Autowired
- private final StudentService studentService;

}

## SERVICE

- Contains business logic of the application.
- Processes data before sending it to the controller or repository.
- Keeps controllers thin and focused on request handling.

- @Service
- public class StudentService {

- @Autowired
- private StudentRepository studentRepository;

- public StudentService(StudentRepository studentRepository) {
- this.studentRepository = studentRepository;
- }

# REPOSITORY

- Handles data access (database operations).
- Uses Spring Data JPA or other persistence frameworks.
- Abstracts database queries from the service layer.

- @Repository
- public interface StudentRepository extends JpaRepository<Student, Long> {

- // Implements all default CRUD operations (see later)
- // Additional query methods can be defined here if needed

- }

# MODEL

- Represents the domain objects or database tables.
- Contains fields, getters/setters, and relationships.
- Maps to database structure via JPA/Hibernate.

- @Entity
- @Table(name = "students")
- public class Student {

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

- @NotBlank(message = "Name is required")
- private String name;

- @Min(value = 18, message = "Age must be at least 18")
- private int age;

- public Student() {
- // Default constructor
- }

## EXAMPLE

- Take a  student action (like Login or viewing his profile):
- The StudentController will be called (like http://localhost:8080/students/login

- This will call the StudentService class to determine what to do next
- If login, then we will handle database connection with the table students and verify its credentials.
- This will call Repository (which hide the SQL layer for us !!)
- This will also include invoking the model class which is Student (always singular)

- For courses : we will have
- CourseController → CourseService → CourseRepository → Course