

CA-3

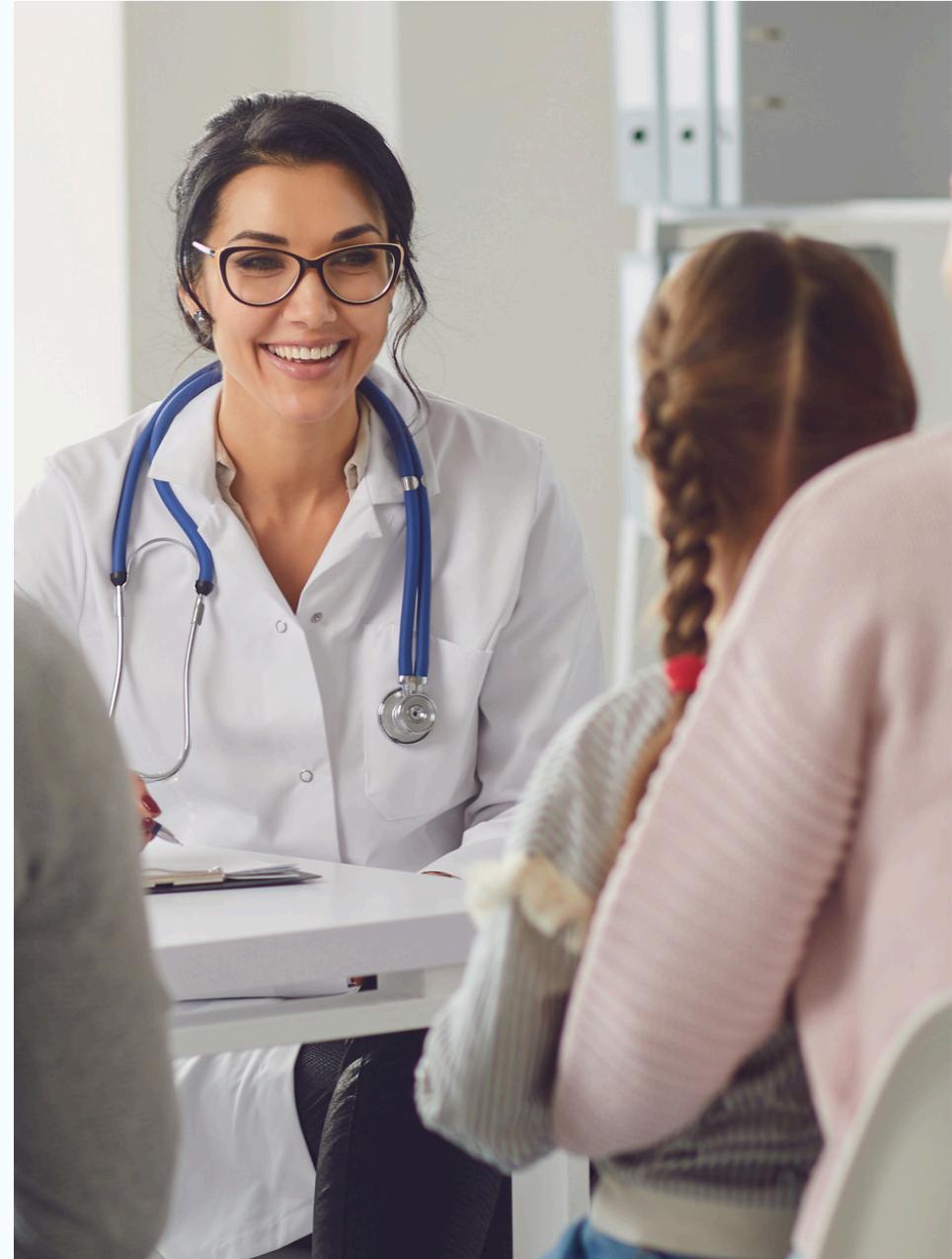
# HOSPITAL MANAGEMENT SYSTEM



# OUR TEAM

---

- AJAY
- DINESH
- GAUTAM
- VISHAL
- HARSHIT
- SUDHEER



SPRINGBOO  
T

# ABOUT PROJECT



**We, as a group, are tasked with developing a RESTful API for a Hospital Management System using Spring Boot. This API will manage operations related to patients, doctors, appointments, departments, and staff. Together, we will follow specific steps to complete the project effectively.**

---

[www.reallygreatsite.com](http://www.reallygreatsite.com)

# ENTITIES AND ATTRIBUTES



- 1. Patient: Attributes like id, name, age, gender, address, contact.
- 2. Doctor: Attributes like id, name, specialization, experience, contact.
- 3. Appointment: Attributes like id, appointmentDate, status, patientId, doctorId.
- 4. Department: Attributes like id, name, headOfDepartment.
- 5. Staff: Attributes like id, name, role, departmentId.

## 1. Patient

- ID: Unique identifier for the patient
- Name: Full name of the patient
- Age: Patient's age
- Gender: Patient's gender
- Address: Residential address of the patient
- Contact: Contact details of the patient

# ENTITY RELATIONSHIPS

## 1. Appointment and Patient

- Relationship: Many-to-One
- Description:
- A single patient can have multiple appointments, but each appointment is associated with only one patient.

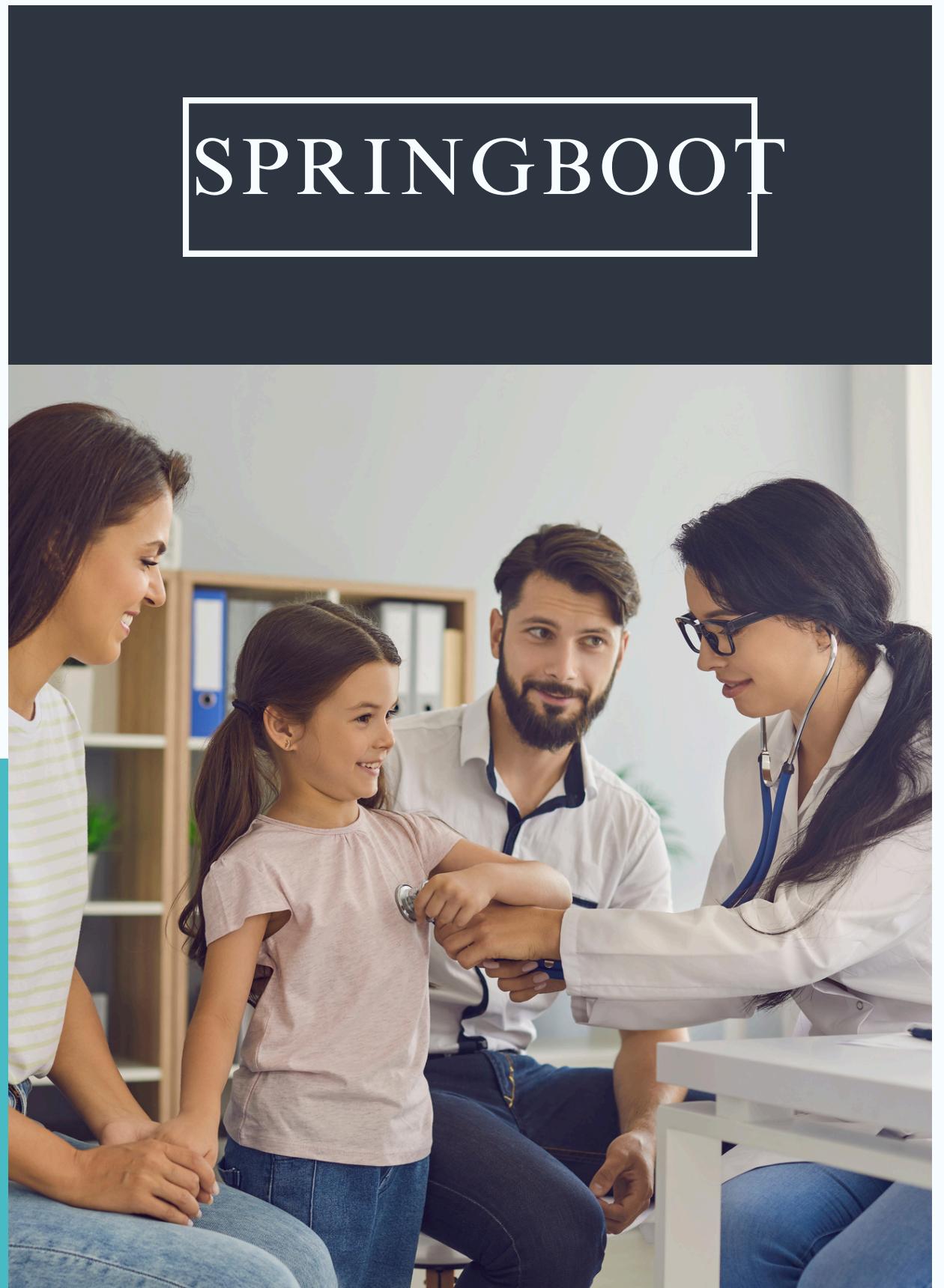


## . Appointment and Doctor

- Relationship: Many-to-One
- Description:
- A single doctor can have multiple appointments, but each appointment is associated with only one doctor.

## Department and Staff

- Relationship: One-to-Many
- Description:
- A single department can have multiple staff members, but each staff member belongs to only one department.



SPRINGBOOT

# FIELDS OF ENTITIES



- 1. Patient**
- id
  - name
  - age
  - gender
  - address
  - contact



**2. Doctor**

- id
- name
- specialization
- experience
- contact

**3. Appointment**

- id
- appointmentDate
- status
- patientId
- doctorId

**4. Department**

- id
- name
- headOfDepartment

**5. Staff**

- id
- name
- role
- departmentId



RESTful

# PATIENT RELATED ENDPOINTS

Patient-related endpoints:

- Retrieve all patients: GET /api/patients
- Get a patient by ID: GET /api/patients/{id}
- Add a new patient: POST /api/patients
- Update a patient: PUT /api/patients/{id}
- Delete a patient: DELETE /api/patients/{id}



RESTful

# DOCTOR RELATED ENDPOINTS

Doctor-related endpoints:

- Retrieve all doctors: GET /api/doctors
- Get a doctor by ID: GET /api/doctors/{id}
- Add a new doctor: POST /api/doctors
- Update a doctor: PUT /api/doctors/{id}
- Delete a doctor: DELETE /api/doctors/{id}



RESTful

# APPOINTMENT RELATED ENDPOINTS

Appointment-related endpoints:

- Retrieve all appointments: GET /api/appointments
- Get an appointment by ID: GET /api/appointments/{id}
- Add a new appointment: POST /api/appointments
- Update an appointment: PUT /api/appointments/{id}
- Cancel an appointment: DELETE /api/appointments/{id}



RESTful

# DEPARTMENT RELATED ENDPOINTS

Department-related endpoints:

- Retrieve all departments: GET /api/departments
- Get a department by ID: GET /api/departments/{id}
- Add a new department: POST /api/departments
- Update a department: PUT /api/departments/{id}
- Delete a department: DELETE /api/departments/{id}



RESTful

# STAFF RELATED ENDPOINTS

Staff-related endpoints:

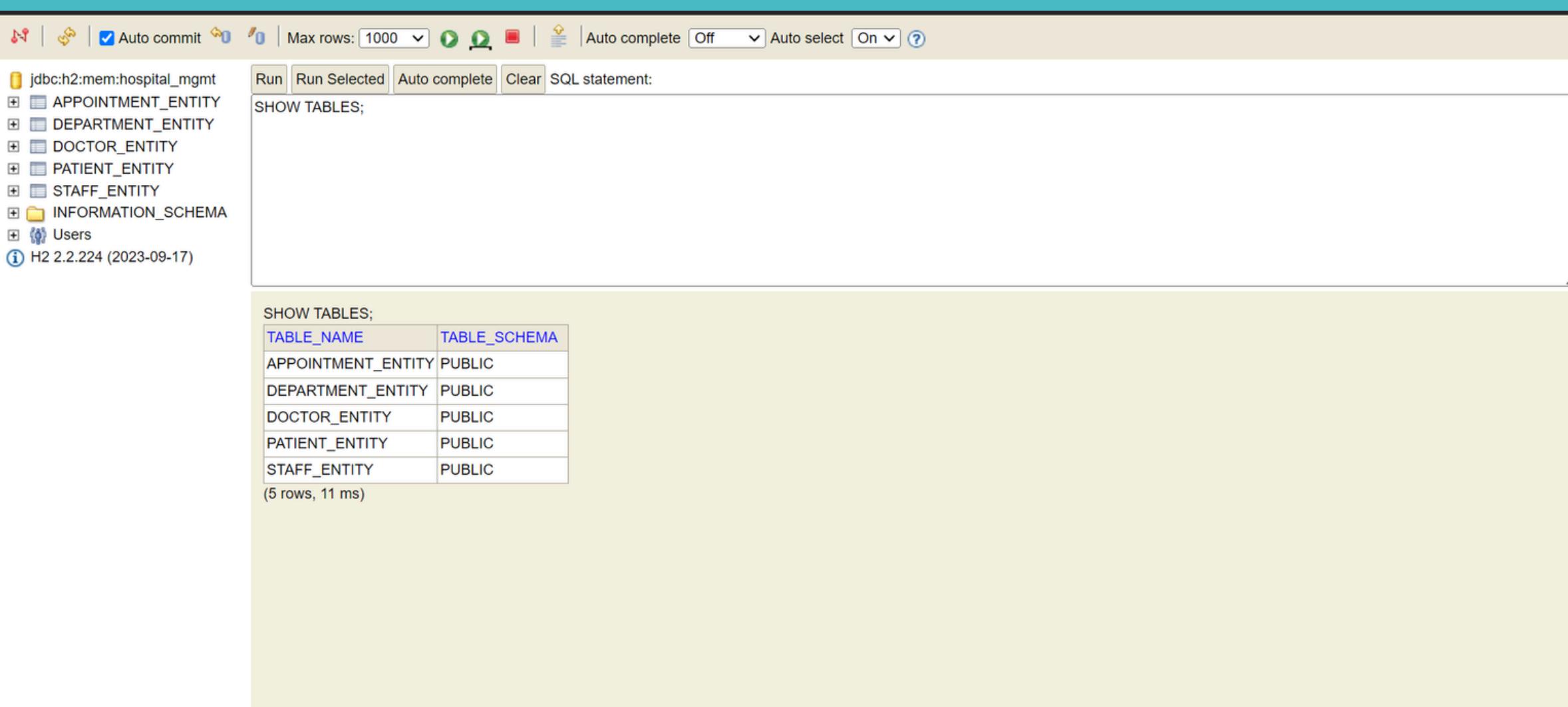
- Retrieve all staff members: GET /api/staff
- Get a staff member by ID: GET /api/staff/{id}
- Add a new staff member: POST /api/staff
- Update staff details: PUT /api/staff/{id}
- Delete a staff member: DELETE /api/staff/{id}

## DEPENDENCIES

- MYSQL CONNECTOR: DATABASE
- SPRING BOOT STARTER DATA JPA: ORM
- SPRING BOOT STARTER WEB: WEB
- SPRING BOOT DEVTOOLS: DEVELOPMENT
- H2 DATABASE: LIGHTWEIGHT DATABASE
- SPRING BOOT STARTER TEST: TESTING

- MYSQL CONNECTOR (MYSQL-CONNECTOR-JAVA)
- SPRING BOOT STARTER DATA JPA (SPRING-BOOT-STARTER-DATA-JPA)
- SPRING BOOT STARTER WEB (SPRING-BOOT-STARTER-WEB)
- SPRING BOOT DEVTOOLS (SPRING-BOOT-DEVTOOLS)
- H2 DATABASE (H2)
- SPRING BOOT STARTER TEST (SPRING-BOOT-STARTER-TEST)

# DATABASE



The screenshot shows the H2 Database Browser interface. The left sidebar lists the database schema:

- jdbc:h2:mem:hospital\_mgmt
- + APPPOINTMENT\_ENTITY
- + DEPARTMENT\_ENTITY
- + DOCTOR\_ENTITY
- + PATIENT\_ENTITY
- + STAFF\_ENTITY
- + INFORMATION\_SCHEMA
- + Users
- (i) H2 2.2.224 (2023-09-17)

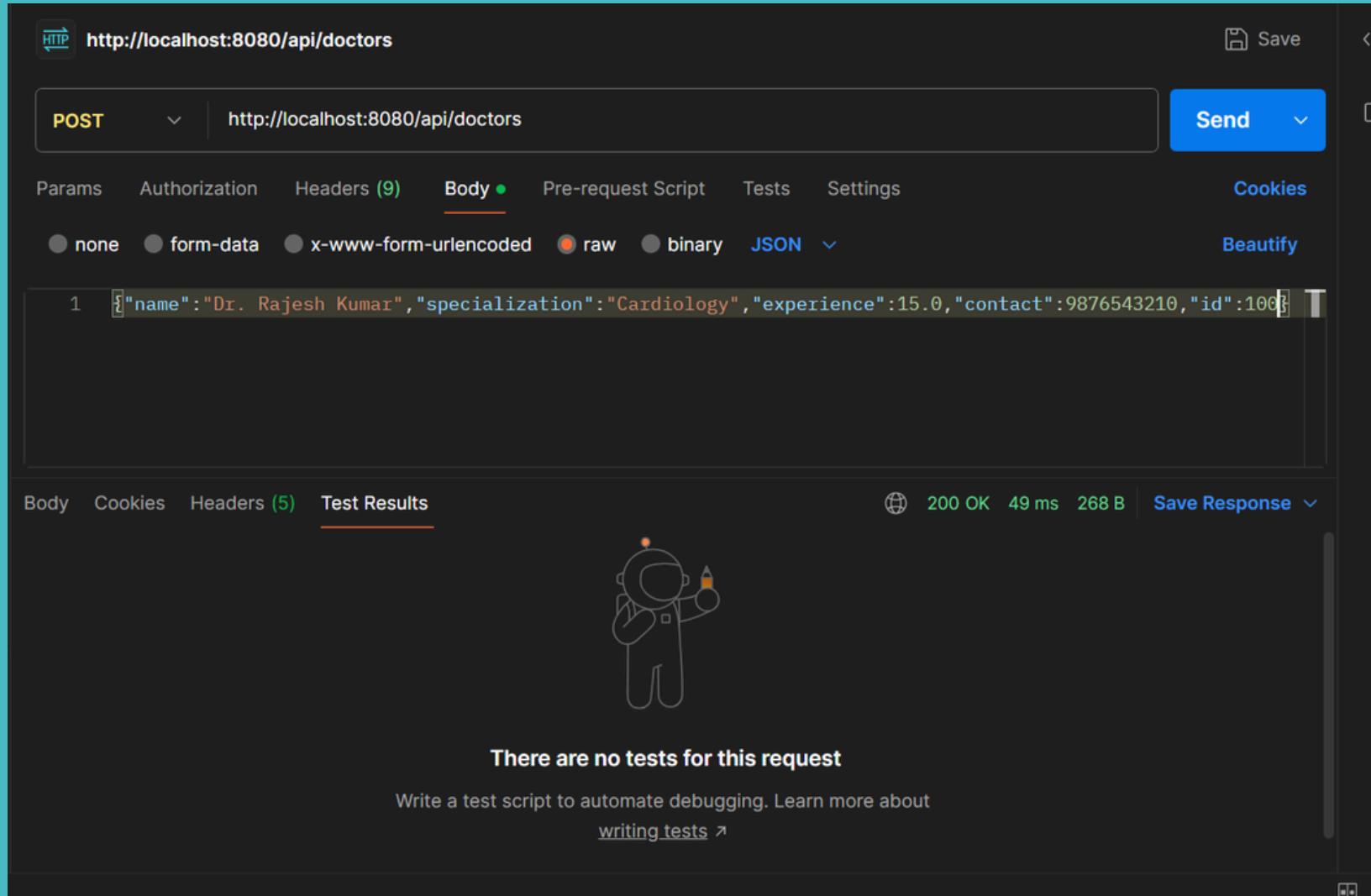
The main panel displays the results of the SQL command `SHOW TABLES;`. Below that, it shows the table structure for `APPPOINTMENT_ENTITY`:

TABLE_NAME	TABLE_SCHEMA
APPPOINTMENT_ENTITY	PUBLIC
DEPARTMENT_ENTITY	PUBLIC
DOCTOR_ENTITY	PUBLIC
PATIENT_ENTITY	PUBLIC
STAFF_ENTITY	PUBLIC

(5 rows, 11 ms)

**The H2 database contains tables such as `APPPOINTMENT_ENTITY`, `DOCTOR_ENTITY`, `PATIENT_ENTITY`, `STAFF_ENTITY`, and `DEPARTMENT_ENTITY`, each with specific fields like IDs, names, and relational attributes. These tables are interconnected using foreign key constraints, defining relationships such as appointments between doctors and patients, and staff assigned to departments.**

# CREATE OPERATION



The screenshot shows the Postman application interface. At the top, it displays the URL `http://localhost:8080/api/doctors`. Below the URL, there's a dropdown menu set to `POST` and another dropdown showing the same URL. The main area is titled "Body" with a green dot, indicating it's selected. The body content is a JSON object:

```
1  {"name": "Dr. Rajesh Kumar", "specialization": "Cardiology", "experience": 15.0, "contact": 9876543210, "id": 100}
```

Below the body, there are tabs for "Params", "Authorization", "Headers (9)", "Cookies", "Beautify", and "Test Results". The "Test Results" tab is currently active, showing a success message with a 200 OK status code, 49 ms response time, and 268 B size. A small icon of a doctor holding a stethoscope is displayed next to the test results. A note says "There are no tests for this request" and provides a link to "writing tests".

The Create Doctor API (POST /api/doctors) adds a new doctor by accepting a JSON payload with the doctor's details and returns the created doctor's information.

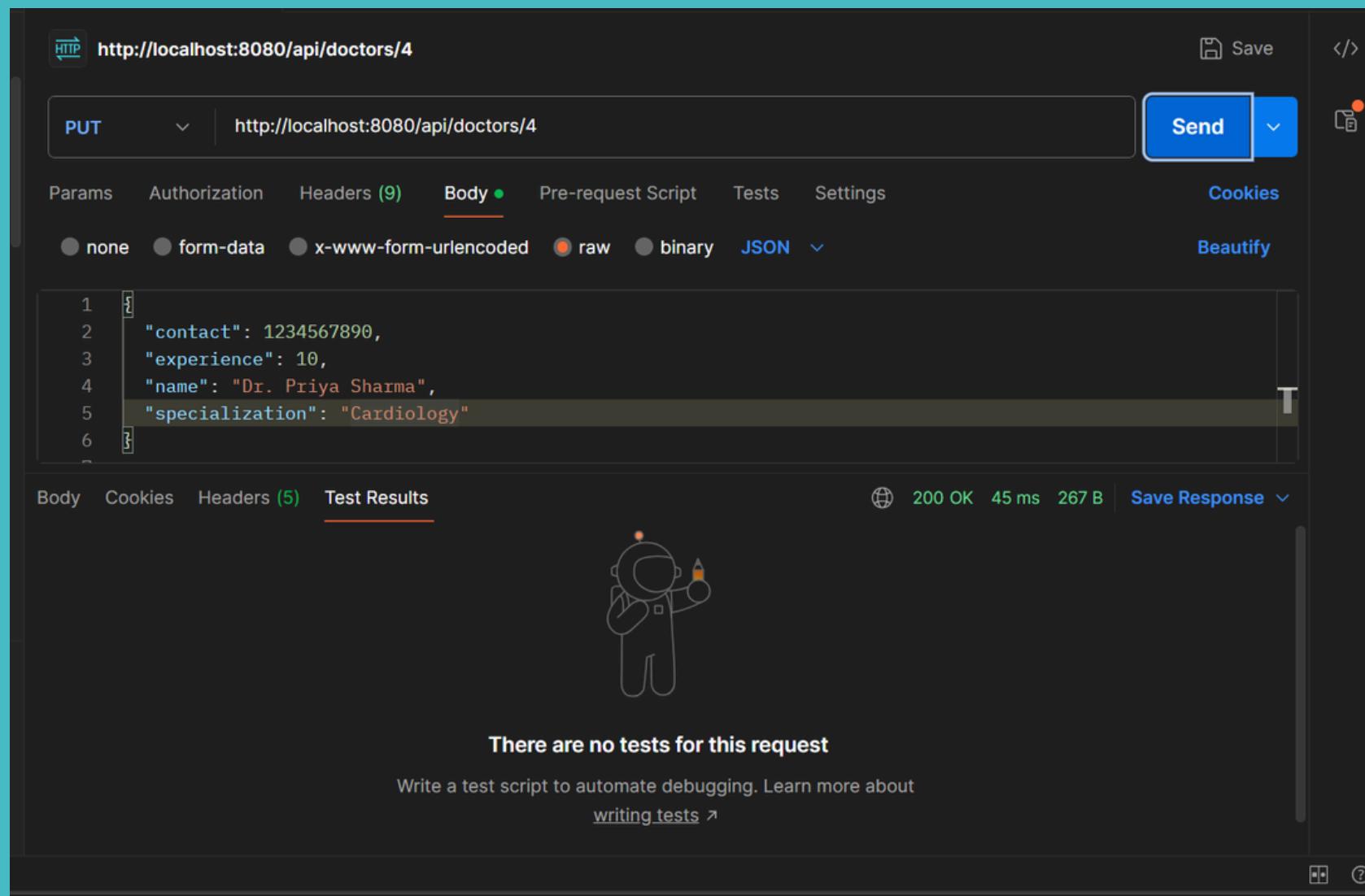
# READ OPERATION

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:8080/api/doctors` is entered into the address bar. Below it, a dropdown menu shows the method as `GET`. The `Body` tab is selected, and the `raw` radio button is checked. The response body is displayed as a JSON array of doctor profiles:

```
[{"name": "Dr. Rajesh Kumar", "specialization": "Cardiology", "experience": 15.0, "contact": 9876543210, "id": 1}, {"name": "Dr. Priya Patel", "specialization": "Dermatology", "experience": 8.0, "contact": 9876543211, "id": 2}, {"name": "Dr. Anil Sharma", "specialization": "Orthopedics", "experience": 10.0, "contact": 9876543212, "id": 3}, {"name": "Dr. Neha Verma", "specialization": "Pediatrics", "experience": 12.0, "contact": 9876543213, "id": 4}, {"name": "Dr. Ravi Singh", "specialization": "General Medicine", "experience": 20.0, "contact": 9876543214, "id": 5}, {"name": "Dr. Shalini Mehta", "specialization": "Neurology", "experience": 7.0, "contact": 9876543215, "id": 6}, {"name": "Dr. Sanjay Gupta", "specialization": "Gynaecology", "experience": 22.0, "contact": 9876543216, "id": 7}, {"name": "Dr. Rekha Nair", "specialization": "Oncology", "experience": 18.0, "contact": 9876543217, "id": 8}, {"name": "Dr. Amit Yadav", "specialization": "ENT", "experience": 14.0, "contact": 9876543218, "id": 9}, {"name": "Dr. Sneha Reddy", "specialization": "Dentistry", "experience": 5.0, "contact": 9876543219, "id": 10}]
```

We as a group tested the GET method of the `/api/doctors` API in Postman by sending a request to `http://localhost:8080/api/doctors` and verified the response data.

# UPDATE OPERATION



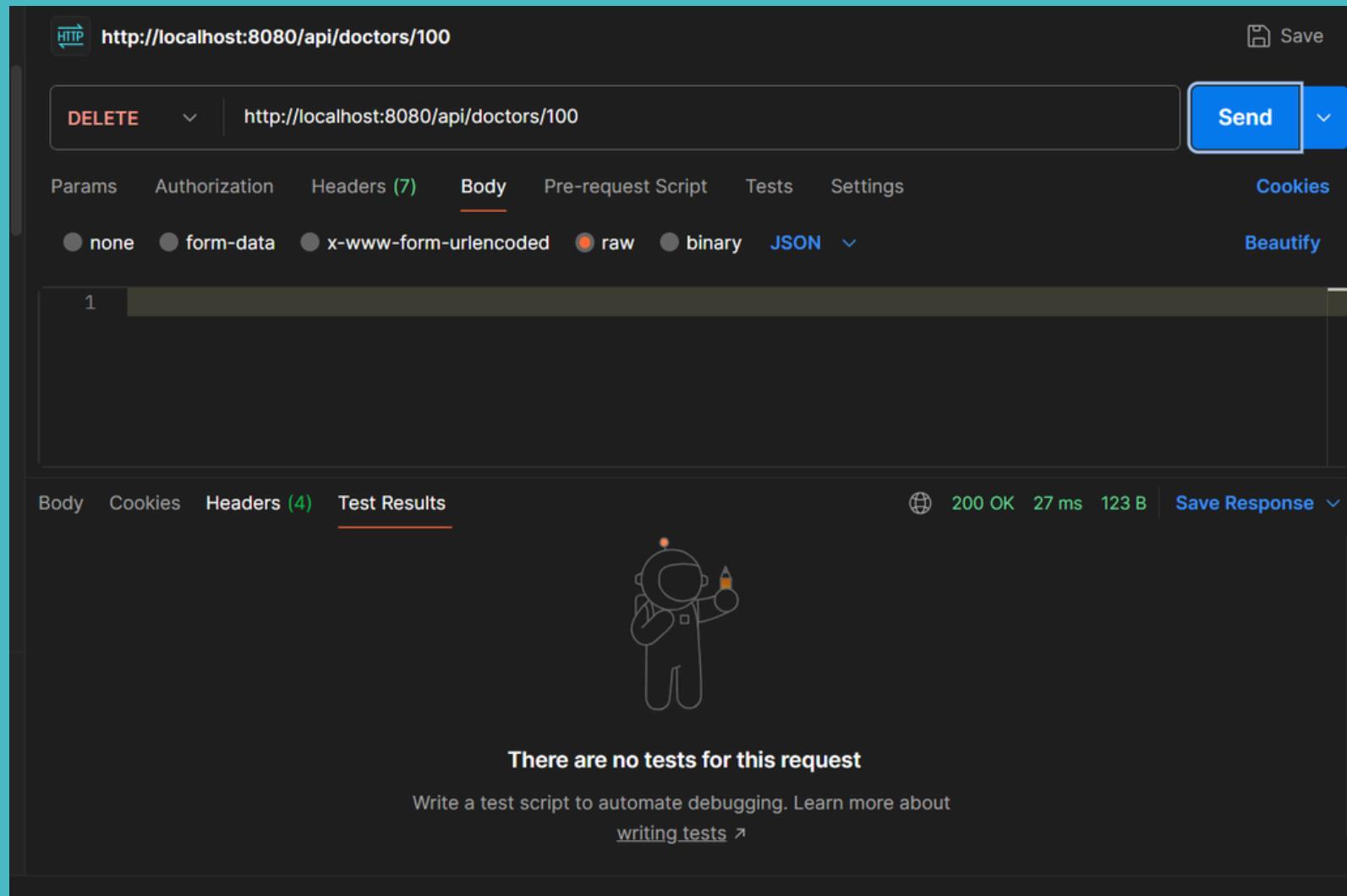
The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:8080/api/doctors/4`. The method is selected as `PUT`. Below the URL, there are tabs for `Params`, `Authorization`, `Headers (9)`, `Body`, `Pre-request Script`, `Tests`, and `Settings`. The `Body` tab is currently active, indicated by a green dot. Under the `Body` tab, the content type is set to `JSON`. The request body contains the following JSON data:

```
1 {
2   "contact": 1234567890,
3   "experience": 10,
4   "name": "Dr. Priya Sharma",
5   "specialization": "Cardiology"
6 }
```

At the bottom of the interface, there are tabs for `Body`, `Cookies`, `Headers (5)`, and `Test Results`. The `Test Results` tab is highlighted with an orange bar. Below these tabs, a message says `There are no tests for this request`. A note at the bottom encourages users to `Write a test script to automate debugging`.

The Update Doctor API (`PUT /api/doctors/{id}`) updates an existing doctor's details by their ID using the provided data in the request body.

# DELETE OPERATION



The Delete Doctor API (`DELETE /api/doctors/{id}`) removes a doctor's record from the database based on their unique ID.

# SERACH WITH ID

The screenshot shows the Postman application interface. At the top, there is a header bar with the URL "GET http://localhost:8080/api/doctors/1". Below this, the main interface has a "Send" button and tabs for "Params", "Authorization", "Headers (7)", "Body", "Pre-request Script", "Tests", and "Settings". The "Params" tab is selected. Under "Query Params", there is a table with two rows, both labeled "Key". In the "Body" section, there are tabs for "Pretty", "Raw", "Preview", and "Visualize", with "Pretty" selected. The response body is displayed as a JSON object: {"name": "Dr. Rajesh Kumar", "specialization": "Cardiology", "experience": 15.0, "contact": 9876543210, "id": 1}. The status bar at the bottom indicates a 200 OK response with 188 ms latency and 267 B size.

The Get Doctor by ID API (GET /api/doctors/{id}) retrieves the details of a doctor from the database based on their unique ID.

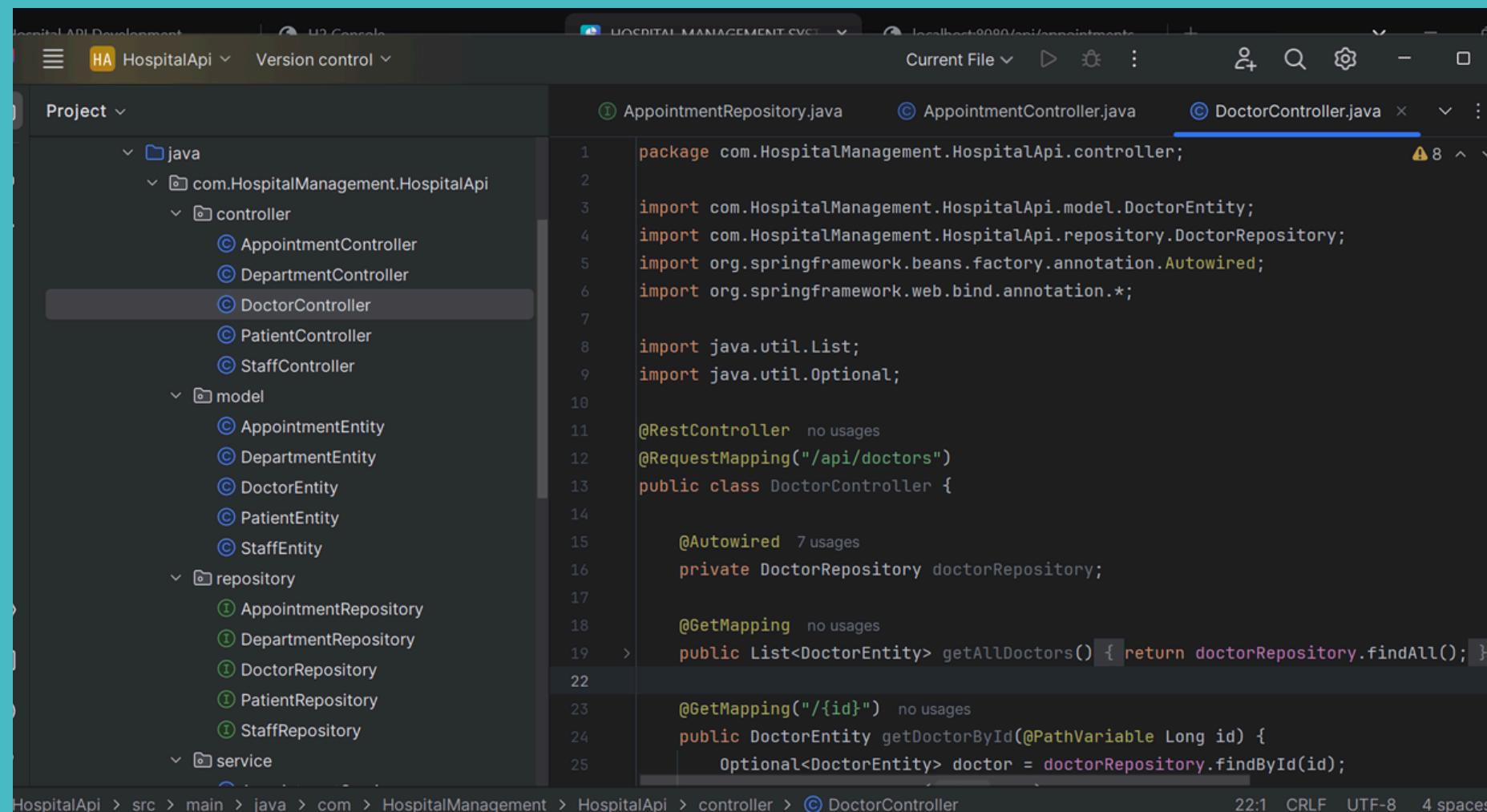
# CONTINUATION

We tested the Patient, Appointment, Department, and Staff APIs using Postman, similar to how we tested the Doctor APIs. We performed GET, POST, PUT, and DELETE requests to verify functionality for retrieving, adding, updating, and deleting records. Each entity's endpoints were tested with appropriate data and verified for correct responses. The API behavior was consistent with the expected outcomes for all entities.

SPRINGBOOT



# THANK YOU



A screenshot of a Java IDE (IntelliJ IDEA) showing the code for the `DoctorController.java` file. The project structure on the left shows packages like `com.HospitalManagement.HospitalApi`, `model`, `repository`, and `service`. The `DoctorController` class is selected in the code editor. The code implements a REST controller for doctors, using `@RestController` and `@GetMapping` annotations. It injects a `DoctorRepository` and provides methods to get all doctors and a specific doctor by ID.

```
package com.HospitalManagement.HospitalApi.controller;

import com.HospitalManagement.HospitalApi.model.DoctorEntity;
import com.HospitalManagement.HospitalApi.repository.DoctorRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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

@RestController no usages
@RequestMapping("/api/doctors")
public class DoctorController {

    @Autowired 7 usages
    private DoctorRepository doctorRepository;

    @GetMapping no usages
    public List<DoctorEntity> getAllDoctors() { return doctorRepository.findAll(); }

    @GetMapping("/{id}") no usages
    public DoctorEntity getDoctorById(@PathVariable Long id) {
        Optional<DoctorEntity> doctor = doctorRepository.findById(id);
    }
}
```