# Doctor–Patient–Appointment Microservices: End‑to‑End Case Study

A progressive case study demonstrating how to build a healthcare appointment booking system using Spring Boot microservices. Covers REST endpoints, Spring validation, OpenFeign, Eureka Service Discovery, API Gateway,Actuator. and observability.

## 1) Problem Statement & Goals

We want to design a healthcare platform where **Patients** can view **Doctors** and book **Appointments**. The architecture should:

* Use microservices (Doctor, Patient, Appointment, API‑Gateway, Discovery).
* Apply Spring validators for clean input handling.
* Enable inter‑service communication with OpenFeign.
* Provide service discovery using Eureka and routing with Spring Cloud Gateway.

## 2) High‑Level Architecture

**Services**

- **doctor‑service** → manages doctor profiles, specialties, schedules. - **patient‑service** → manages patient details, contacts, preferences.

- **appointment‑service** → handles availability, bookings, cancellations, rescheduling. - **api‑gateway** → unified entry point; routing, filters, authentication.

- **eureka‑server** → service registry for discovery.

**Data Ownership** - Each service owns its own database.

**Ports (example)** - eureka‑server: 8761 - api‑gateway: 8080 - doctor‑service: 8011 - patient‑service: 8012 - appointment‑service: 8013

## 

## 3) Domain Models

**Doctor**

{  
 "id": 101,  
 "fullName": "Dr. Aisha Verma",  
 "specialty": "Cardiology",  
 "email": "aisha.verma@care.com",  
 "phone": "+91-98xxxxxxx",  
 "locations": ["Delhi, GK-II"],  
 "workingHours": {  
 "mon": ["09:00-13:00", "15:00-17:00"],  
 "tue": ["09:00-13:00"]  
 }  
}

**Patient**

{  
 "id": 501,  
 "fullName": "Ashu Jauhari",  
 "email": "ashu@example.com",  
 "phone": "+91-9xxxxxxxxx",  
 "dob": "1995-05-12"  
}

**Appointment**

{  
 "id": 9001,  
 "doctorId": 101,  
 "patientId": 501,  
 "start": "2025-09-03T09:30:00+05:30",  
 "end": "2025-09-03T09:45:00+05:30",  
 "status": "BOOKED", // BOOKED | CANCELLED | COMPLETED  
 "notes": "Follow-up"  
}

## 4) API Endpoints

### 4.1 Beginner (CRUD Basics)

**doctor‑service** –

POST /dapi/doctors → create doctor

- GET /dapi/doctors/{id} → fetch doctor

- PUT /dapi/doctors/{id} → full update

- PATCH /dapi/doctors/{id} → partial update

- DELETE /dapi/doctors/{id} → delete

- GET /dapi/doctors?specialty=Cardiology&city=Delhi&page=0&size=20 → search doctors

**patient‑service**

– POST /papi/patients

- GET /papi/patients/{id}

- PUT /papi/patients/{id}

- PATCH /papi/patients/{id}

- DELETE /papi/patients/{id} - GET /papi/patients?name=ashu&page=0&size=10

**appointment‑service**

- POST /aapi/appointments → create booking

- GET /aapi/appointments/{id} → get appointment

- DELETE /aapi/appointments/{id} → cancel appointment

- POST /aapi/appointments/search → search by doctor, patient, or date

### 4.2 Intermediate

* GET /aapi/availability?doctorId=101&date=2025-09-03 → available slots
* POST /aapi/appointments/bulk → bulk booking

### 4.3 Advanced

* POST /aapi/appointments/{id}:reschedule → reschedule

Request : {

"newDate": "2025-09-05",

"newTime": "15:30",

"reason": "Doctor not available at original time"

}

Response:

{

"appointmentId": 101,

"status": "Rescheduled",

"oldDateTime": "2025-09-03T14:00",

"newDateTime": "2025-09-05T15:30",

"doctorId": 12,

"patientId": 45

}

Or

{

"error": "Appointment with ID 101 not found"

}

* POST /aapi/appointments/{id}:complete → mark complete
* POST /dapi/doctors/{id}/schedule:block → block slots

## 5) Spring Validation

**DTO with constraints**

public record CreateAppointmentRequest(  
 @NotNull Long doctorId,  
 @NotNull Long patientId,  
 @NotNull OffsetDateTime start,  
 @NotNull @Future OffsetDateTime end,  
 @Size(max = 500) String notes  
) {}

## 6) Eureka Service Discovery

**eureka‑server config**

server:  
 port: 8761  
spring:  
 application:  
 name: eureka-server  
  
eureka:  
 client:  
 fetch-registry: false  
 register-with-eureka: false

**client config**

spring:  
 application:  
 name: appointment-service  
server:  
 port: 0  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/

## 7) API Gateway

spring:  
 application:  
 name: api-gateway  
 cloud:  
 gateway:  
 routes:  
 - id: doctor-service  
 uri: lb://doctor-service  
 predicates:  
 - Path=/dapi/\*\*  
 - id: patient-service  
 uri: lb://patient-service  
 predicates:  
 - Path=/papi/\*\*  
 - id: appointment-service  
 uri: lb://appointment-service  
 predicates:  
 - Path=/aapi/\*\*  
server:  
 port: 8080  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://localhost:8761/eureka/

## 

## 8) Feign Clients

**1. appointment‑service → doctor‑service**

@FeignClient(name = "doctor-service", path = "/dapi")  
public interface DoctorClient {  
 @GetMapping("/doctors/{id}")  
 DoctorDto getDoctor(@PathVariable Long id);  
}

2. List all patients of a Doctor.

3. Doctor wants to see how many appointments are still in queue.

4. Doctor wants to see all cancelled appointments.

5. Show the total appointment for each doctor in a day.

## 9) Persistence & Transactions

* Each service manages its own database.
* appointment‑service enforces unique (doctorId, start, end).
* Idempotency keys can prevent duplicate bookings.

## 11) Observability

* Distributed tracing (Zipkin).

## 12) Testing Strategy

### .**Unit Tests** (Business Logic & Validators)

* **Tool:** JUnit + Mockito
* **Write Mockito test cases for Controller and Service layer.**

## 13) Show the health of each microservices.

14) Push the logs of each microservice in respective file.

## 15) Example Workflow

**Book Appointment** 1. Client → Gateway → POST /aapi/appointments 2. appointment‑service validates request and checks doctor/patient via Feign. 3. Availability confirmed → appointment stored. 4. Response returned with appointment details.