Islamic University
Faculty of Information Technology

PROJECT sys

Master Program

HOTEL Res

Advanced Software Eng- ICTS 6302

Prepared By :
Lama Abu Shawish
220253747

TO SUPERV
Dr. Abdelkareem Alashqar, IUG

DATE 2025
Semester: 1st 2025/2026
Date: December 2025

PROJECT OVERVIEW

ABOUT THE SYSTEM

This work presents a Hotel Reservation System for room booking,

customer registration, and agenda management in the hotel business. The system helps save time and makes work efficient in day-to-day transactions# It also protects against the possibility of misuse of fake currency.rded.

the bookings, the availability data and guest details).

The system is part of the Advanced Software Engineering course.

It reflects practical work in requirements, design, and implementation.

Product Vision

This project creates an easy and quick reservation system that allows the guest, receptionist and administration to find the available rooms immediately without confusion or paperwork. The system unifies the entire hotel business, arranges guest data, and reduces errors that occur in manual booking. The goal is for the service to be faster, more accurate, and run smoothly every day.

01

FOR

For hotel guests, receptionists and management, all of whose daily work is linked to the reservation system.

02

WHO

WHO needs a quick and organized way to see empty rooms, and to be able to make their reservations without delays or errors like those that occur with manual reservations.

03

IS A

Is online booking platform, equipped to organize all your booking work: rooms, guest records, and system updates.

04

THAT

THAT provides automatic reservations, real-time room updates, verifies payment, and collects all guest information in one place, with a clear interface that makes reservations easy for everyone.

05

UNLIKE

UNLIKE Manual booking and outdated systems that are confusing, inaccurate, and records are lost.

06

OUR PRODUCT

OUR PRODUCT provides an integrated, accurate, and fast solution… It reduces errors, speeds up hotel work, and helps management make its decisions easily because the system is stable and working properly.

.

persona

Sarah
Hotel Receptionist

Name: Sarah
Age: 29
Job: Receptionist
Location: Al Bahr Hotel
Experience: Two years working there

Background

Sarah does all her work with people from the beginning of the day to the end. The most tiring time is in the evening, because suddenly a few guests come in together and they want to finish quickly. She now knows the organization of the place and how it works, but the crowding that comes once causes her a little confusion.

Her goals

Get rid of receiving guests without taking too long.

Make sure that the rooms you give away are not reserved for anyone else.

He quickly goes to her empty rooms instead of searching a lot.

She arranges her work without the little papers that are wasted on her.

Obstacles

Crowding when guests enter at once.

Notes that you write quickly and forget what they were.

The system sometimes runs slowly and crashes.

The guest is standing and she is looking for a room among many options.

She gives away a room and leaves booked, and that's what she's most afraid of.

Her needs

A quick system shows her empty rooms immediately.

A place within the system where notes are written instead of paper.

An organized way to send orders to housekeeping without any hassle.

Omar
Frequent Hotel Guest

Name: Omar
Age: 35
His job: traveling from country to country
Number of trips: approximately two trips per month

Background

Omar travels frequently for work, and booking hotels has become a routine task for him.

His goals

Clear room information with no confusion.

A guaranteed payment method with no headache.

Instant confirmation… he can't wait.

Finishing the reservation in minutes because his time is always tight.

A system that stores his past bookings and reminds him before traveling.

Obstacles

When the booking website gets stuck or looks old.

When he sees a room marked "available" and then finds out it's not empty at all.
When he has to call reception and keep waiting, especially when he's traveling and needs to confirm something quickly.

His needs

A fast, clear, and uncomplicated system.

Just a few steps… he books, continues, and leaves.

Automatic notifications about his reservations without having to search for them.

persona

Lina
the hotel manager

Name: Lina
Age: 42
Position: Hotel Manager
Experience: Over 15 years working in this field

Background

Lina is the one who carries the hotel on her shoulders.

Every day she starts checking reservations, looking at empty and occupied rooms, and arranging the staff program.

The work needs to proceed clearly, and no mess will ruin the day.

Her goals?

Things are clear in front of her without philosophy.

The errors that occur from manual booking are reduced.

The system quickly shows her who has booked which room and when.

Dashboard tells her how things are going weekly and monthly.

Know the strong seasons early to prepare herself.

Long-term planning without continuing to extinguish daily problems.

Her needs?

A fast, organized system that updates data instantly.

Information ready in front of her without her searching for it.

A clear sign showing occupancy, movement, and booking directions.

Close monitoring of the condition of each room, who is entering and who is leaving.

A system that reduces its reliance on paper and storytelling among employees.

Obstacles?

When she opens the system and finds that the data is not up to date.

Errors that occur due to manual booking.

The slowness of the system that disrupts all work behind it.
She revolves around a simple piece of information between a million pages.

When employees find their data, they ask her again about every detail.

She doesn't know early when the strong seasons will come.

Stressful days when she does not have clear vision.

Ahmed
IT Support Technician

Name: Ahmed
Age: 31 years
Job: IT Support Technician in a large hotel
Experience: 4 years working in the hotel
Study: Master's in Delhi + IT Diploma

Background

Ahmed is the person they go to as soon as any system disrupts the hotel.

All his work is about maintaining computers, adjusting networks, and making sure that all digital platforms are working without problems, especially the reservation system.

He deals daily with receptionists, cleaning staff, and managers, and solves problems that suddenly appear and disrupt the day's work.

His goals

A stable reservation system that does not get agitated every now and then.

Reduces technical errors and downtime.

Maintains smooth work between departments through one clear system.

Has administrative powers that help him control problems quickly.

Train new employees easily without explaining every detail.

His needs

A stable, simplified, and easy-to-follow platform.

Obvious updates that cause data confusion.

A dashboard that shows him the system status and the faults if they occur.

Management tools that allow him to control problems from his place without a phone call every now and then.

A system that works throughout the hotel without conflicts or connection problems.

Obstacles

When the reservation system suddenly slows down or stops half of the work.

The calls that come to him during rush hour… and everyone needs an immediate solution.

Minor malfunctions that the system could have solved if it had been designed correctly.

Updates that cause new errors instead of fixing old ones.

Employees who use the system incorrectly, and he has to explain it to them again.

When departments get confused because the system is not unified between them.

Scenario

Sarah
Hotel Receptionist

Sarah was working at the evening reception.

A group of guests arrived at once and stood in line. She opened the reservation system immediately to see empty rooms and make sure there are no reservations in the same room.

She started distributing rooms, checking payments, and entering guest data quickly so they wouldn't be late.

As she saw them, she followed up on special requests, such as an extra bed or room preparation, and followed up with housekeeping to complete them

Live updates have made coordination between departments easier.

However, Sarah needs a clearer way to show her the pending matters that need to be done immediately

So that there is no delay for guests.

Omar
Frequent Hotel Guest

Omar, 35 years old, always travels for work, and from time to time he wants to book a hotel.

He travels about twice a month, looking for a fast and accurate booking system, payment is guaranteed, and confirmation comes all the time.

Before booking, Omar usually takes a look at the photos, compare prices and find out what services are available.

The hardest thing is when the reservation system is canceled, suspended, or non-empty rooms are offered.

Since he is always in a hurry, the system must complete the reservation within minutes, without having to contact anyone or wait for a long time

He also wants the records to be kept in his old reservations and send him reminders and updates about his stay

Scenario

Lina
the hotel manager

In

Lina, 42, is the hotel's general manager.

She has worked in this job for many years, and follow everything in the place: staff management, workflow, and level of service.

Every day she checks reservations and room status, and decide how the day is going.

She needs accurate and up-to-date information to make the right decision.

Lina wants the system to reduce manual booking errors and give her a clear picture: who blocked them, which room, and when.

What she wants most are dashboards that show the hotel's movement over weeks and months

So she can prepare for the seasons and see what needs improvement in front of you.

Ahmed
IT Support
Technician

Ahmed is the support officer for the reservation system.

In the morning, the employees started telling him: "The system is slow and does not want to work like others!"

Since these were days of pressure, Ahmed realized that such a delay would disrupt the entire hotel business.

Opening logs sees the problem: From the Internet? Is the server unrestricted? Is there no update in place?

When he knew why, he installed updates, ran services, and the system returned to normal.

Also make sure that the room and reservation information is correct and that there is no confusion.

Then he gives the new employees a quick explanation of how to behave and not cause delays.

Although the system makes them comfortable, Ahmed himself wants clearer tools and a diagnostic panel that detects the problem before it occurs, especially during busy times.

User Stories

Sarah
Hotel Receptionist

As a receptionist , I want to be able to see room availability so that I can assign rooms immediately

As a receptionist , I want to record unique guest requests so that I can forward them to housekeeping immediately

As a receptionist , I want to check in people quickly so I can deal with busy times effectively.

Omar
Frequent Hotel
Guest

As a guest , I want to be able to filter rooms based on my preferences (bed, price) so that I can easily find a room that matches my preferences.

As a guest , I want to simple and secure payment process so that I can receive my booking confirmation immediately

As a guest , I want to be notified of booking changes so that I can adjust my travel plans if anything changes.

User Stories

Lina
the hotel manager

As a manager , I want to view an operational dashboard so that I can see the hotel's current so that I can see the hotel's performance on a day-to-day basis

As a manager , I want to create reports on daily and weekly bases, so that I can take some strategic decisions.

As a manager, I want to clean and organized reservation information with correct phone number so that we can reduce front-desk errors such as incorrect arrivals.

Ahmed
IT Support
Technician

As a IT Support Technician , I want to be able to check system performance and spot trouble so that system issues do not affect hotel operations.

As a IT Support Technician , I want to administrative tools at my disposal to quickly figure out why there are system errors.

As a IT Support Technician , I want to access logs and diagnostic reports so that I can identify the causes of performance issues

Features

- Real-time room availability tracking
- Room search and filtering
- Automated booking workflow
- Secure online electronic payment
- Real time booking confirmation and notification system
- Guest profile and booking history management.
- Special requests and housekeeping alerts can be registered
- Operational dashboard for managers
- You will use a variety of experience factors when working across your reporting tools (daily, weekly, monthly)
- Trend analysis for Strategic Planning
- Role based access for reception, admin, user and IT support
- System performance monitoring
- Error notifications and Diagnostic Logging and Tracing Tools
- Tools for synchronization to avoid overbooking
- Solid and scalable infrastructure for minimizing system downtime

Microservice Architecture Design

a.1 Services, Containers, and Ports (as implemented)

This project runs using Docker Compose. Each component is deployed as a separate container with fixed ports:

1) discovery-server (Eureka) – port 9011

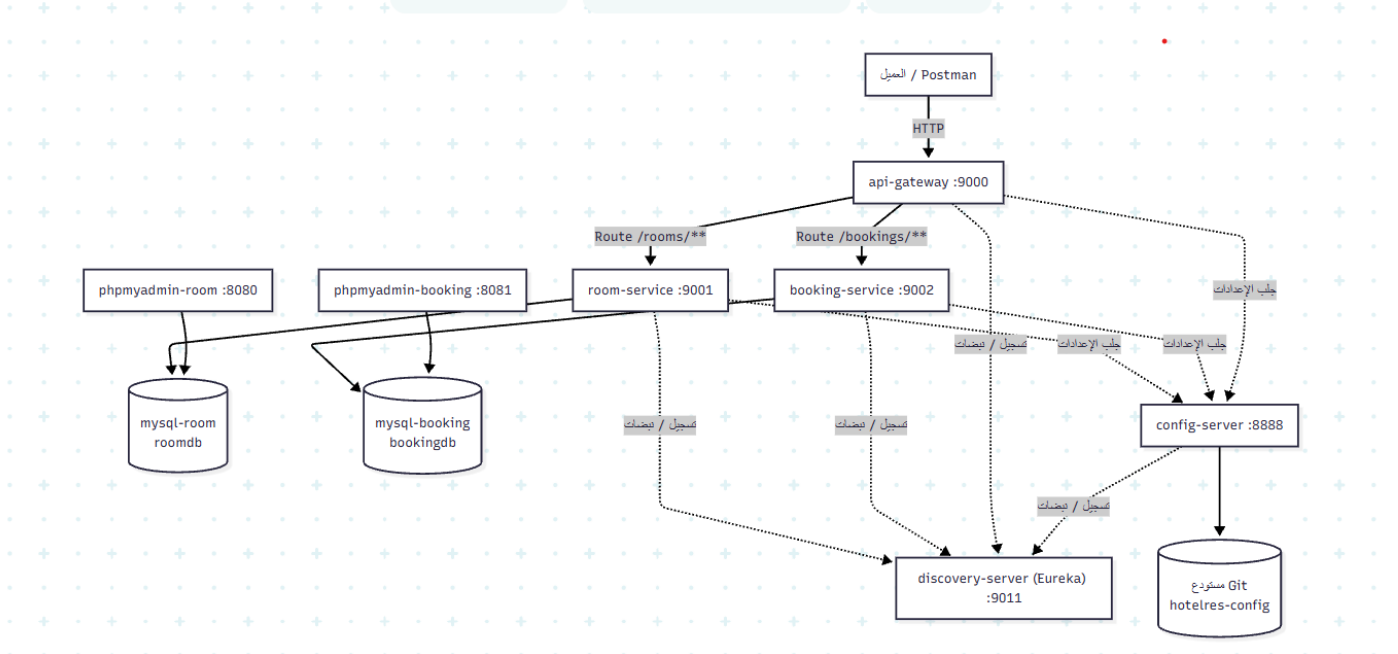
2) api-gateway – port 9000

3) config-server – port 8888 (Git-backed Config Server)

4) room-service – port 9001 + mysql-room (roomdb) + phpmyadmin-room (8080)

5) booking-service – port 9002 + mysql-booking (bookingdb) + phpmyadmin-booking (8081)

```
PS C:\Users\msi\OneDrive\Desktop\HotelRes-MS> docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED       STATUS                   PORTS                                                NAMES
c5819ae90998   hotelres-ms-config-server      "java -jar app.jar"      40 hours ago  Up 3 minutes             0.0.0.0:8888->8888/tcp, [::]:8888->8888/tcp          config-server
b11aad3380ed   hotelres-ms-discovery-server   "java -jar app.jar"      40 hours ago  Up 3 minutes             0.0.0.0:9011->9011/tcp, [::]:9011->9011/tcp          discovery-server
c1b555191831   hotelres-ms-api-gateway        "java -jar app.jar"      43 hours ago  Up 3 minutes             0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp          api-gateway
ef5812aa3afe   hotelres-ms-room-service       "java -jar app.jar"      43 hours ago  Up 3 minutes             0.0.0.0:9001->9001/tcp, [::]:9001->9001/tcp          room-service
725f36811435   hotelres-ms-booking-service    "java -jar app.jar"      43 hours ago  Up 3 minutes             0.0.0.0:9002->9002/tcp, [::]:9002->9002/tcp          booking-service
af8246e9afaa   phpmyadmin/phpmyadmin          "/docker-entrypoint.…"   43 hours ago  Up 3 minutes             0.0.0.0:8080->80/tcp, [::]:8080->80/tcp              phpmyadmin-room
e7aedfe2ccd3   phpmyadmin/phpmyadmin          "/docker-entrypoint.…"   43 hours ago  Up 3 minutes             0.0.0.0:8081->80/tcp, [::]:8081->80/tcp              phpmyadmin-booking
5c75e97d1856   mysql:8                        "docker-entrypoint.s…"   43 hours ago  Up 3 minutes (healthy)   0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp          mysql-room
6ee3939f2225   mysql:8                        "docker-entrypoint.s…"   43 hours ago  Up 3 minutes (healthy)   0.0.0.0:3308->3306/tcp, [::]:3308->3306/tcp          mysql-booking
```



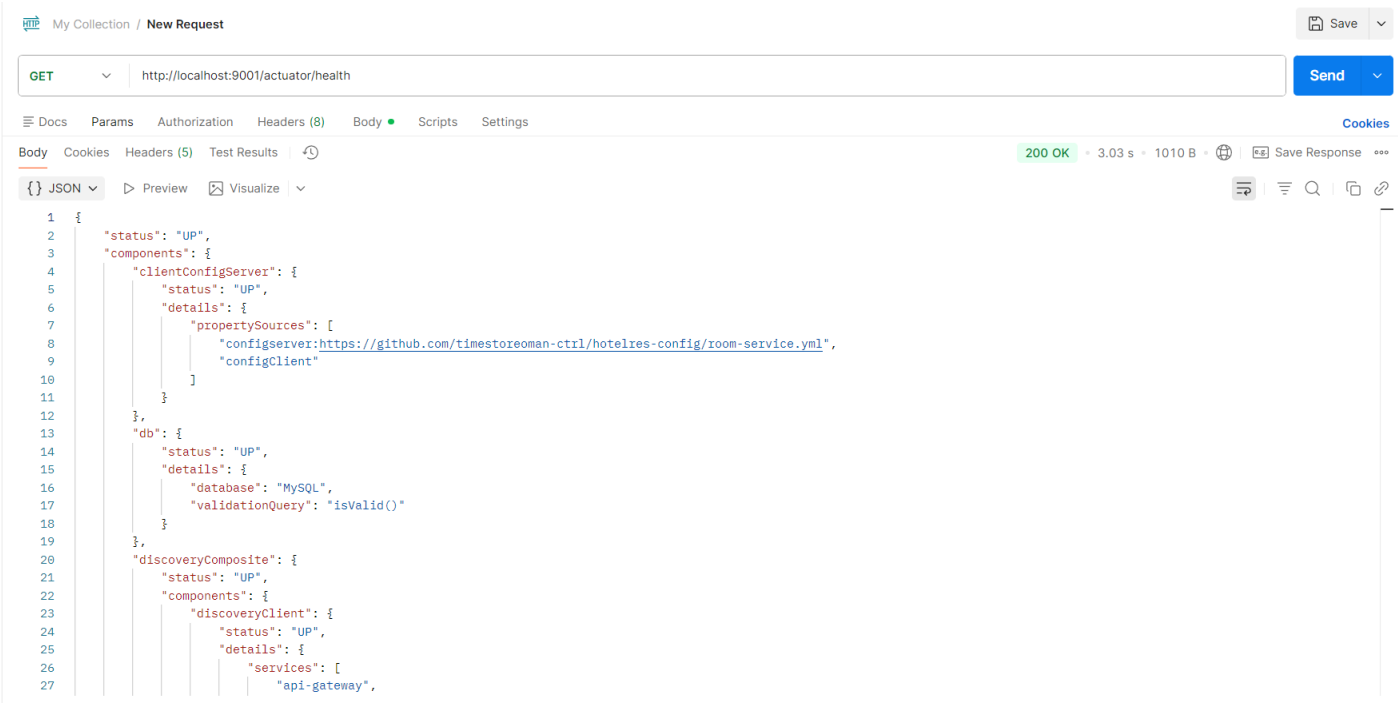
a.2 Health Checks (Actuator)

All services expose health endpoints for verification:

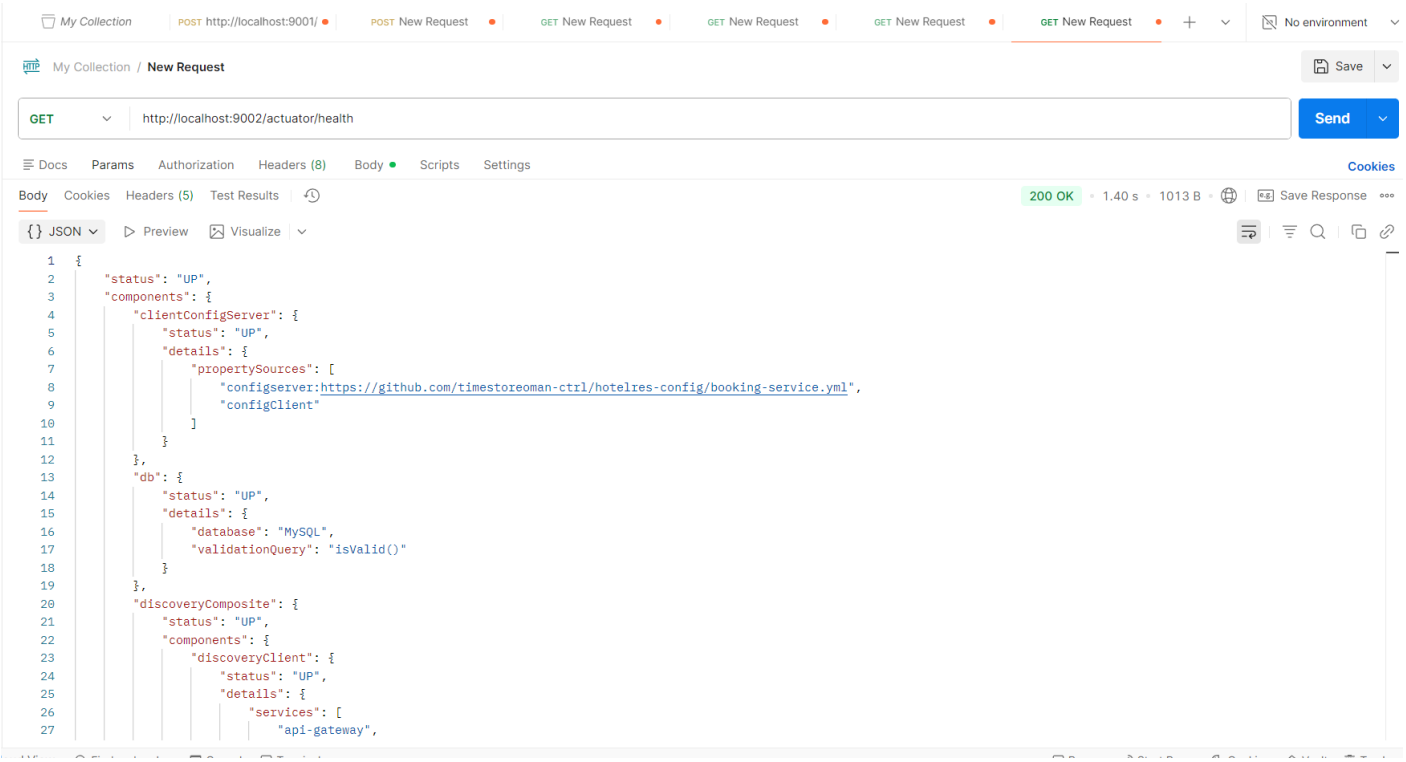
• GET /actuator/health (room-service, booking-service, api-gateway, config-server)

Health check response screenshots (status=UP) | Where to get it:

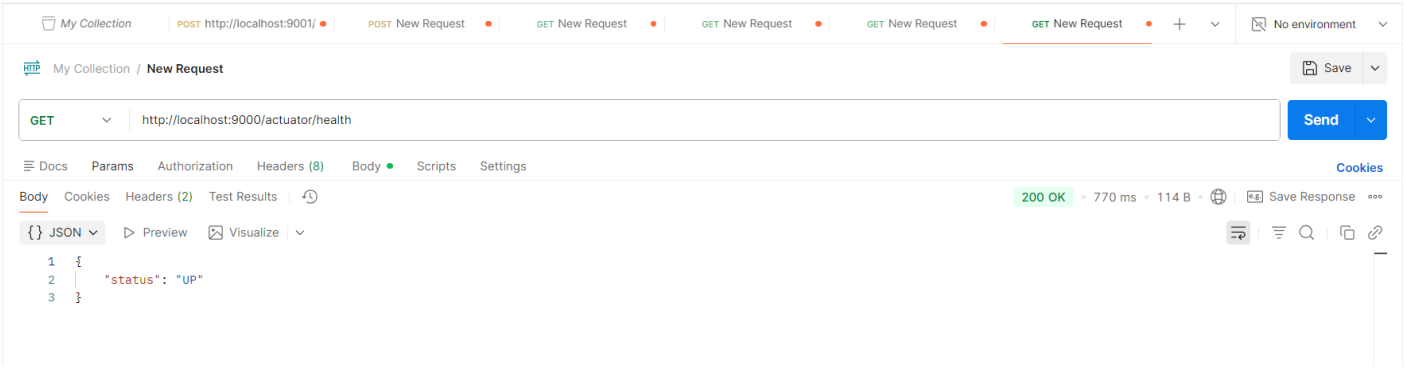
01. **room-service**: http://localhost:9001/actuator/health



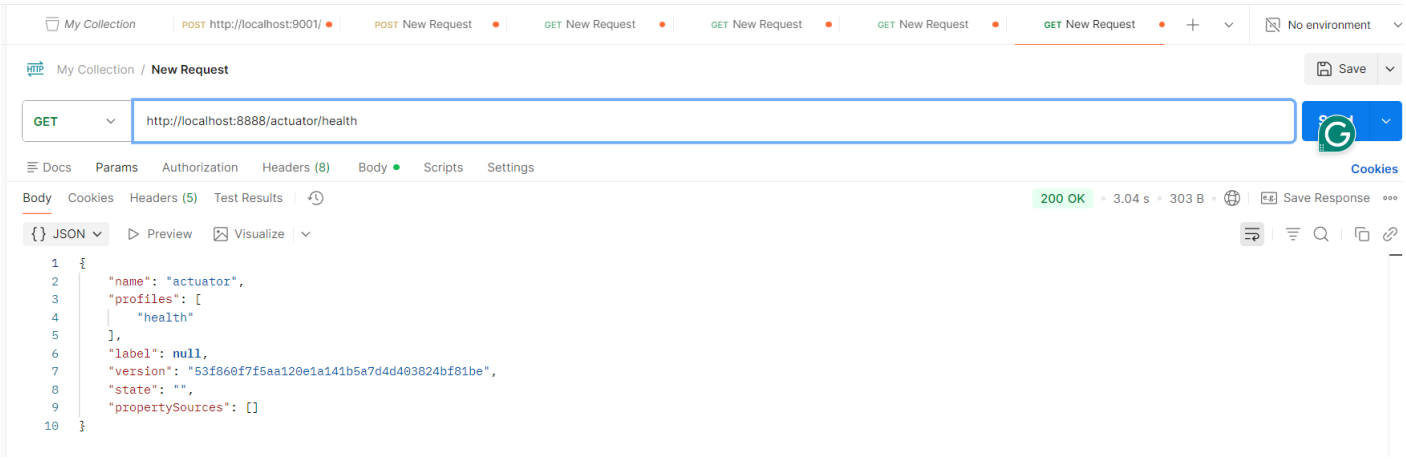
02. **booking-service**: http://localhost:9002/actuator/health



03. api-gateway: http://localhost:9000/actuator/health



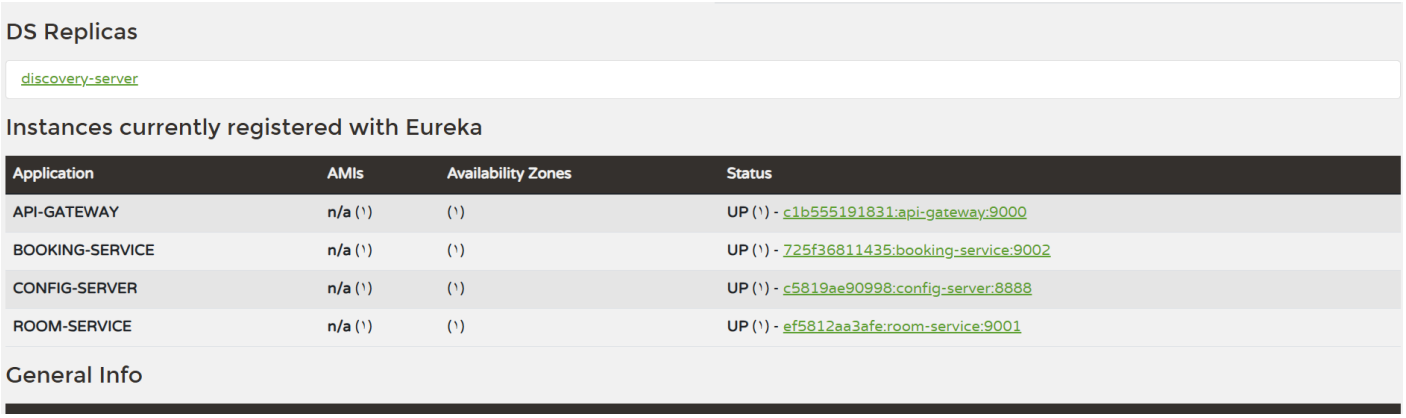
04. **config-server**: http://localhost:8888/actuator/health



a.3 Eureka Discovery Verification

Eureka Dashboard must show the following services as UP:

- API-GATEWAY
- ROOM-SERVICE
- BOOKING-SERVICE
- CONFIG-SERVER



b) Data Design (Database per Service + ERD)

b.1 Database per Service

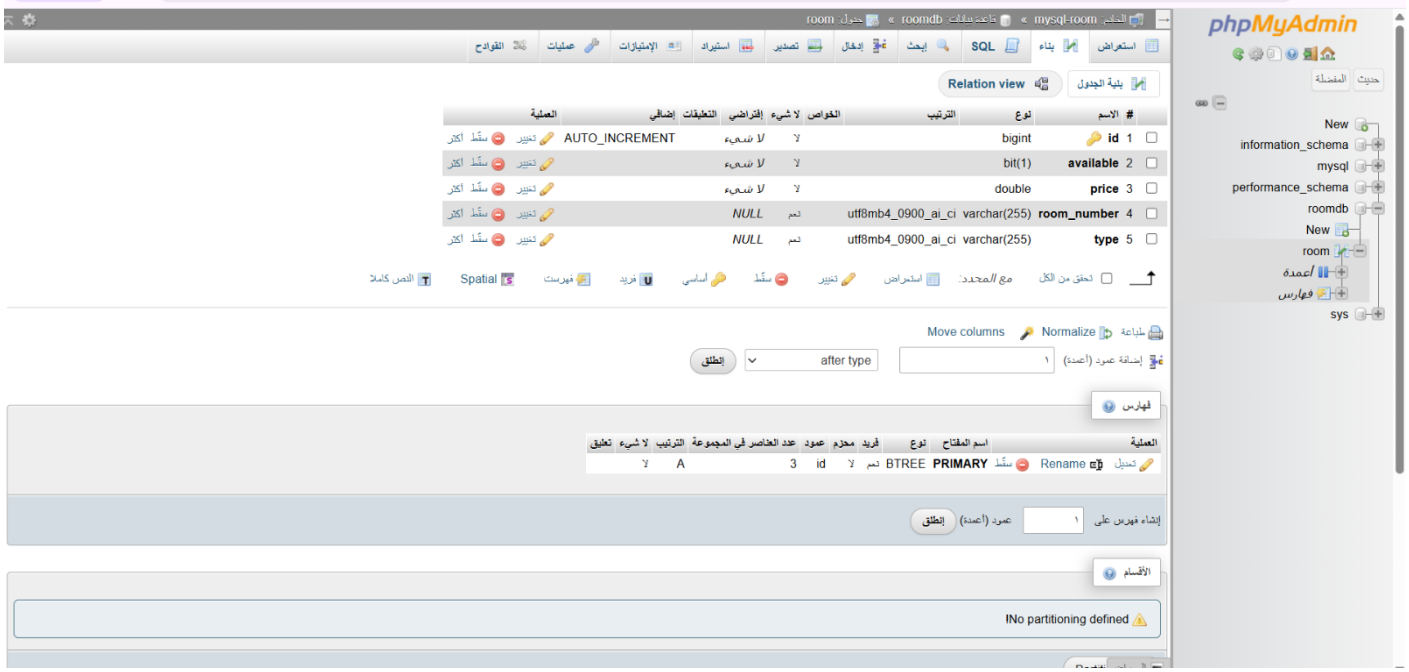
The system follows the Database-per-Service pattern. Each microservice owns its database schema and tables:

• room-service → roomdb (mysql-room)

• booking-service → bookingdb (mysql-booking)

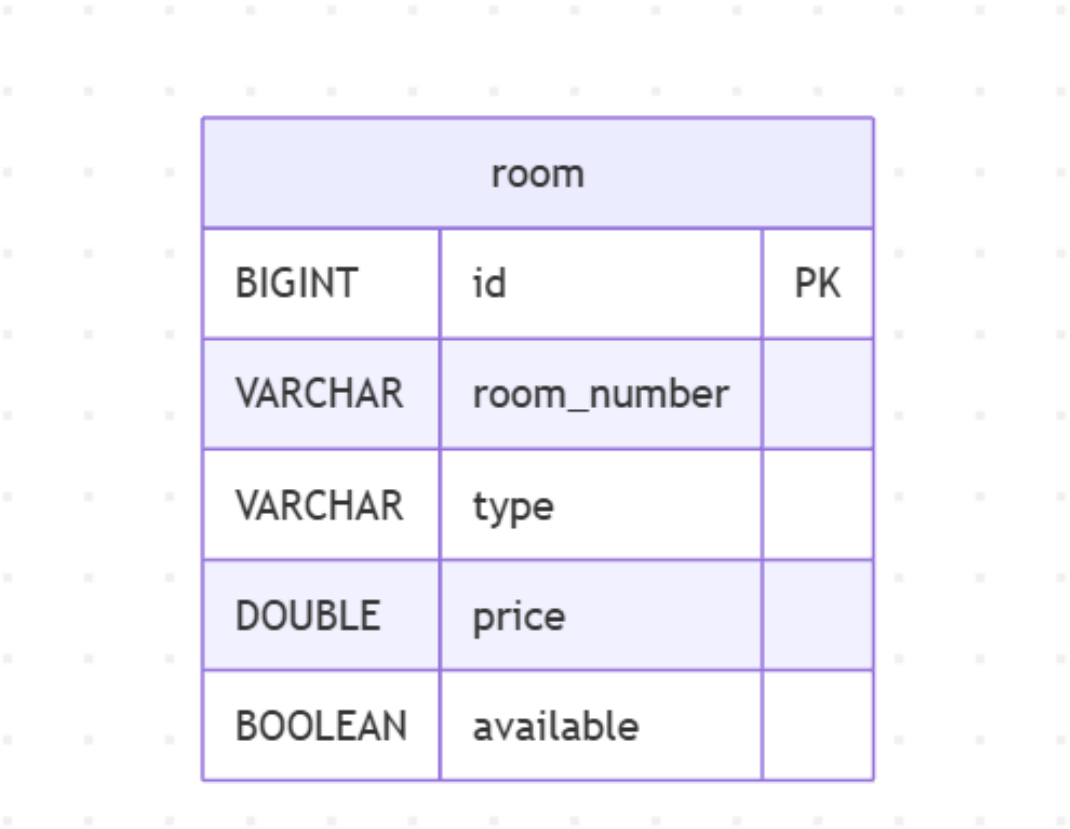
Cross-service references are logical only. The booking table stores roomId, and validity is guaranteed through REST calls to room-service, not through cross-database foreign keys.

01. **room-service**:



02. **booking-service**:

b.2 Room Database (roomdb) – ERD



b.3 Booking Database (bookingdb) – ERD

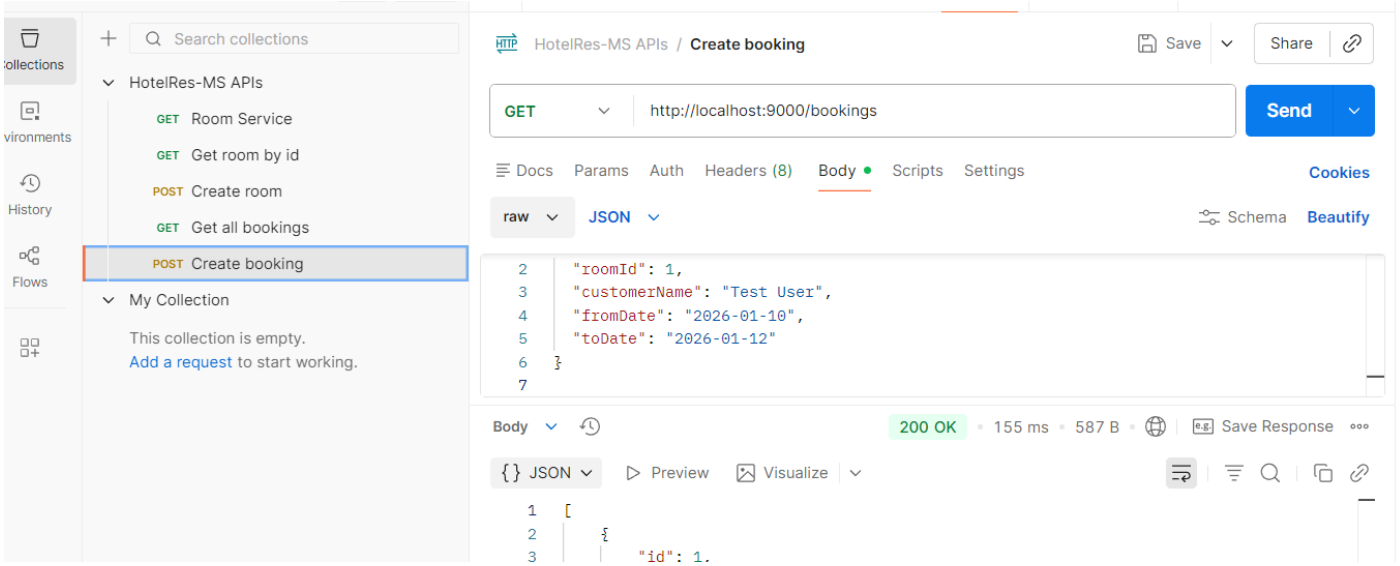
b.4 Seed Data (Room Service)

The following SQL inserts sample rooms for testing. The table name must match the actual table name in your database:

sql
INSERT INTO rooms (room_number, type, price, available) VALUES
('101', 'Single', 50.0, true),
('102', 'Double', 75.0, true),
('201', 'Suite', 120.0, false);

c) REST API Design (Room Service + Booking Service)

The client accesses the system through API Gateway at: http://localhost:9000. Direct service URLs are used only for troubleshooting.



c.1 Room Service Endpoints

Base URL via Gateway: http://localhost:9000

Direct URL (service): http://localhost:9001

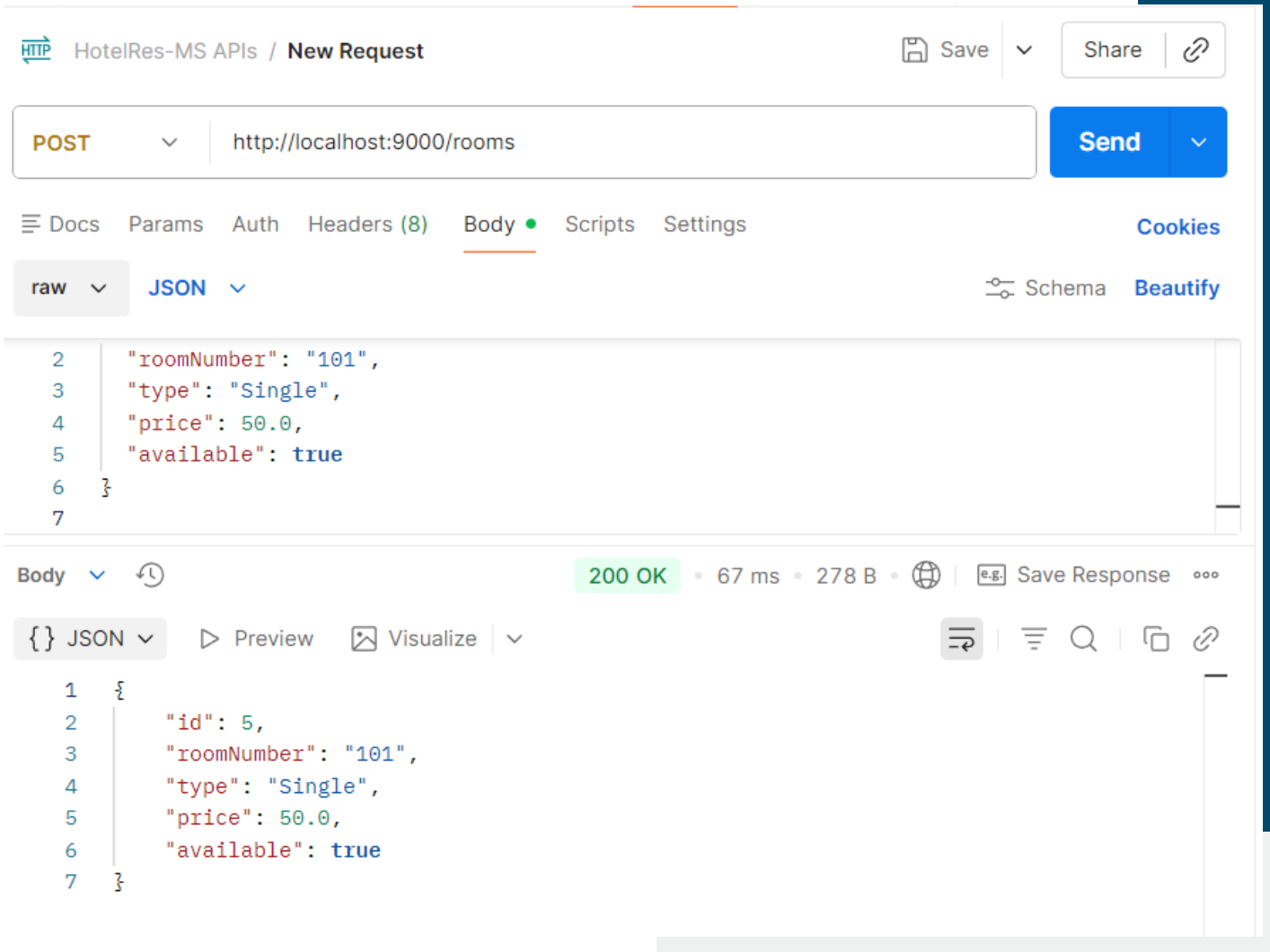
Create Room – POST /rooms

Request JSON:

json
```
{
  "roomNumber": "101",
  "type": "Single",
  "price": 50.0,
  "available": true
}
```

Response JSON (after save):

json
```
{
  "id": 5,
  "roomNumber": "101",
  "type": "Single",
  "price": 50.0,
  "available": true
}
```

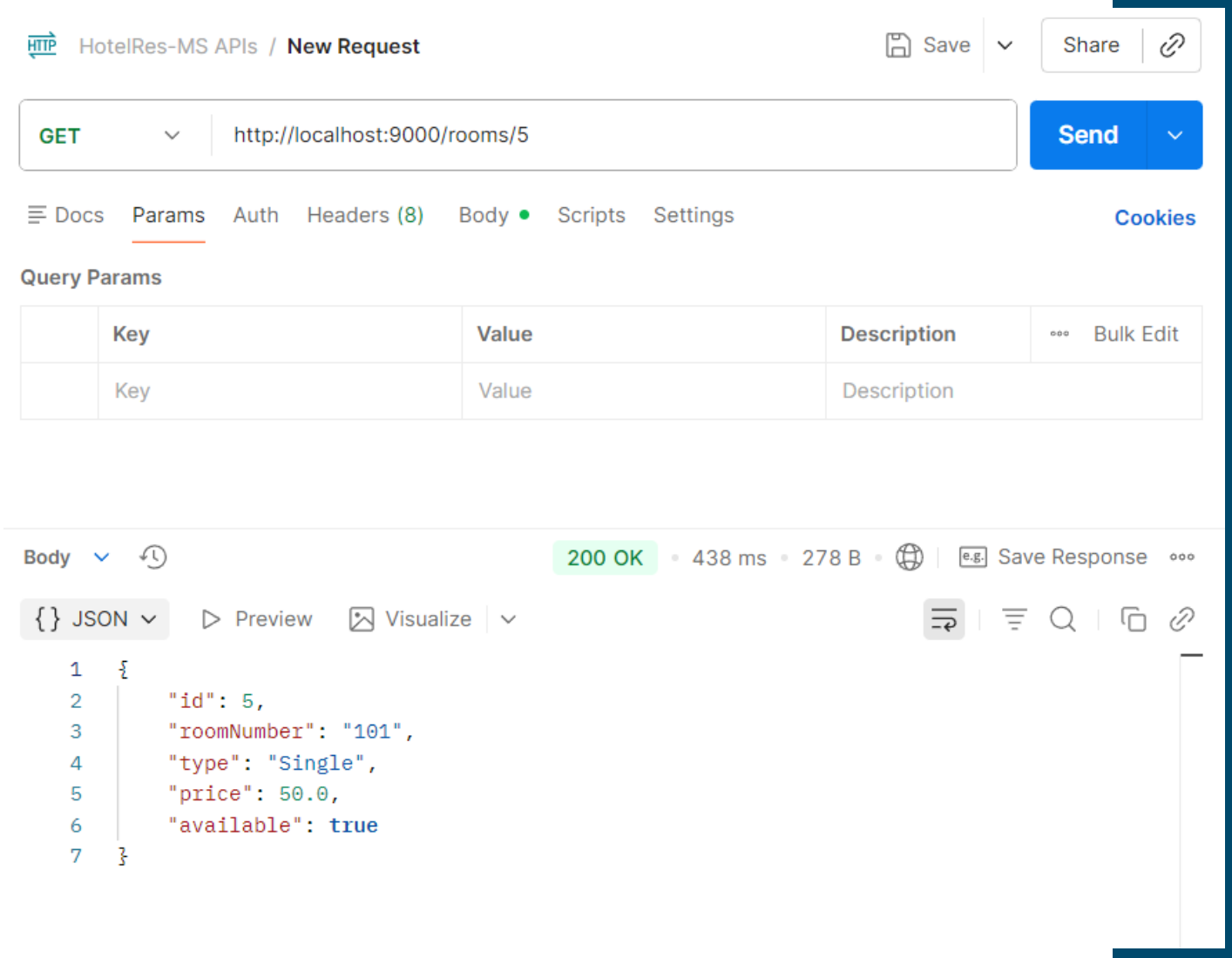


Read Room by ID – GET /rooms/{id}

Example: GET /rooms/5

Response JSON:

```
json
{
  "id": 5,
  "roomNumber": "101",
  "type": "Single",
  "price": 50.0,
  "available": true
}
```



c.2 Booking Service Endpoints

Base URL via Gateway: http://localhost:9000

Direct URL (service): http://localhost:9002

Create Booking – POST /bookings

Request JSON:

json

```
{
  "roomId": 1,
  "guestName": "Omar Saleh",
  "checkIn": "2026-01-20",
  "checkOut": "2026-01-23"
}
```

Response JSON (after save):

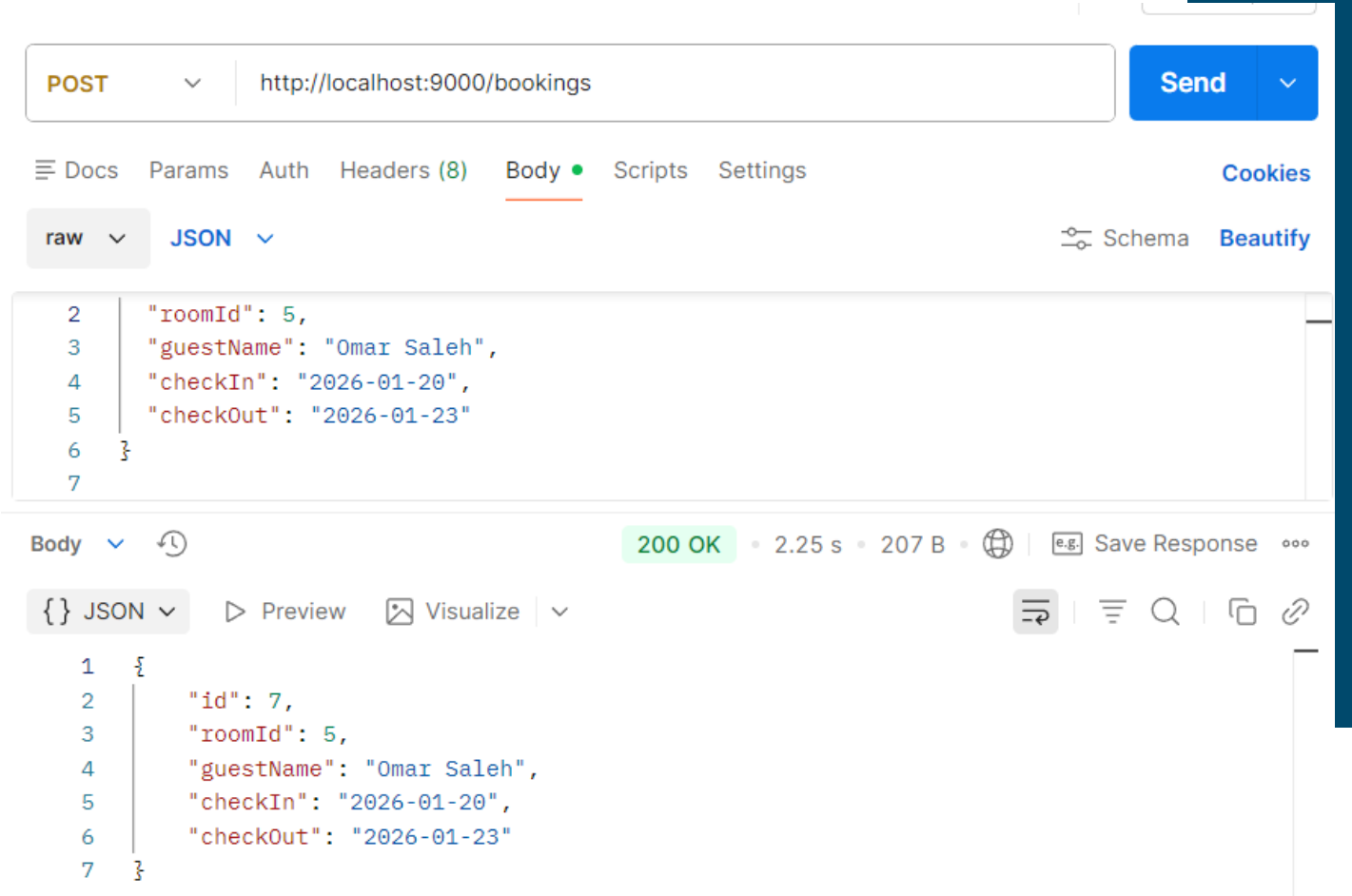
json

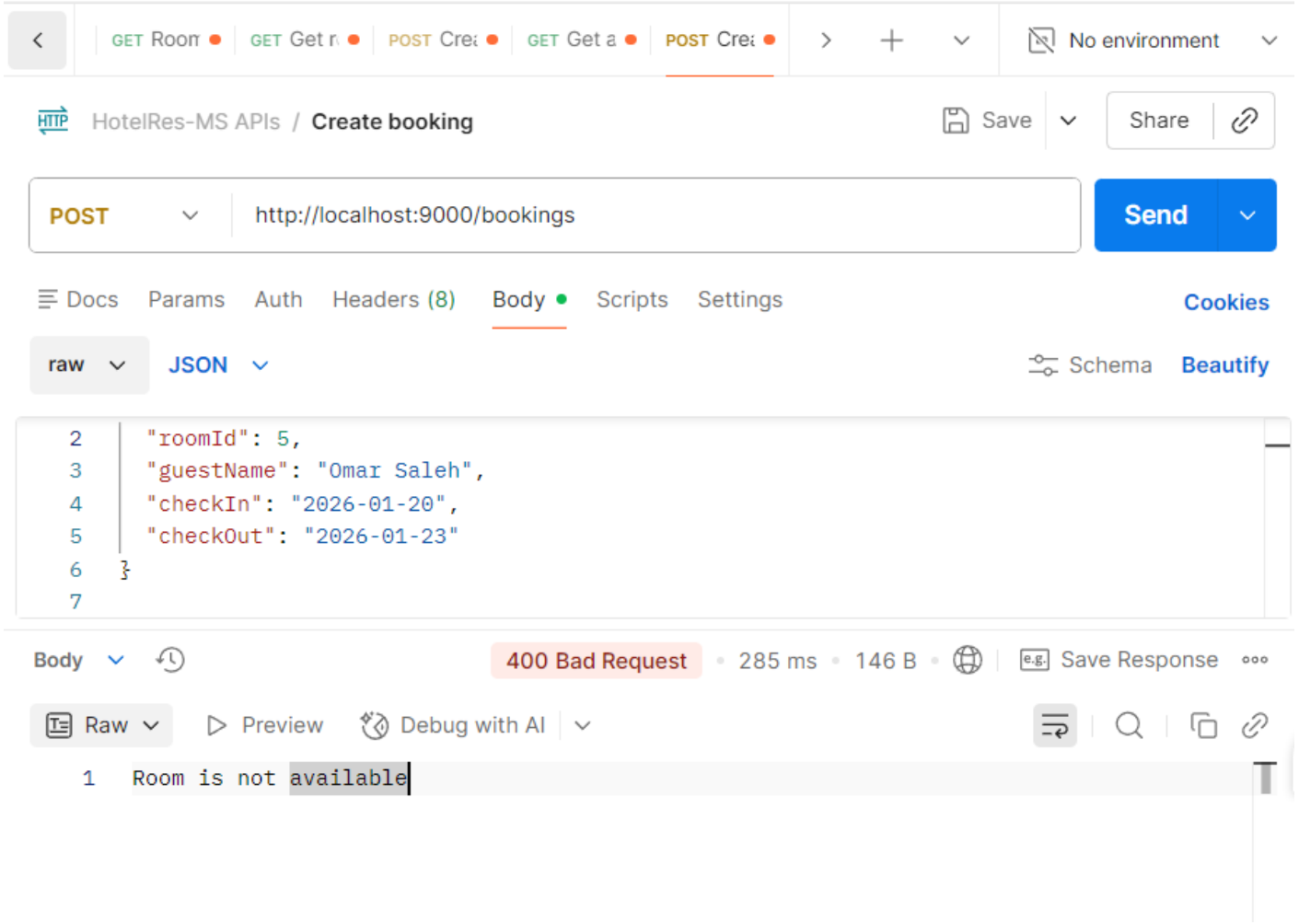
```
{
  "id": 1,
  "roomId": 1,
  "guestName": "Omar Saleh",
  "checkIn": "2026-01-20",
  "checkOut": "2026-01-23"
}
```

If the room is not available, booking-service returns

a clear error message (Bad Request) as implemented in BookingController

.





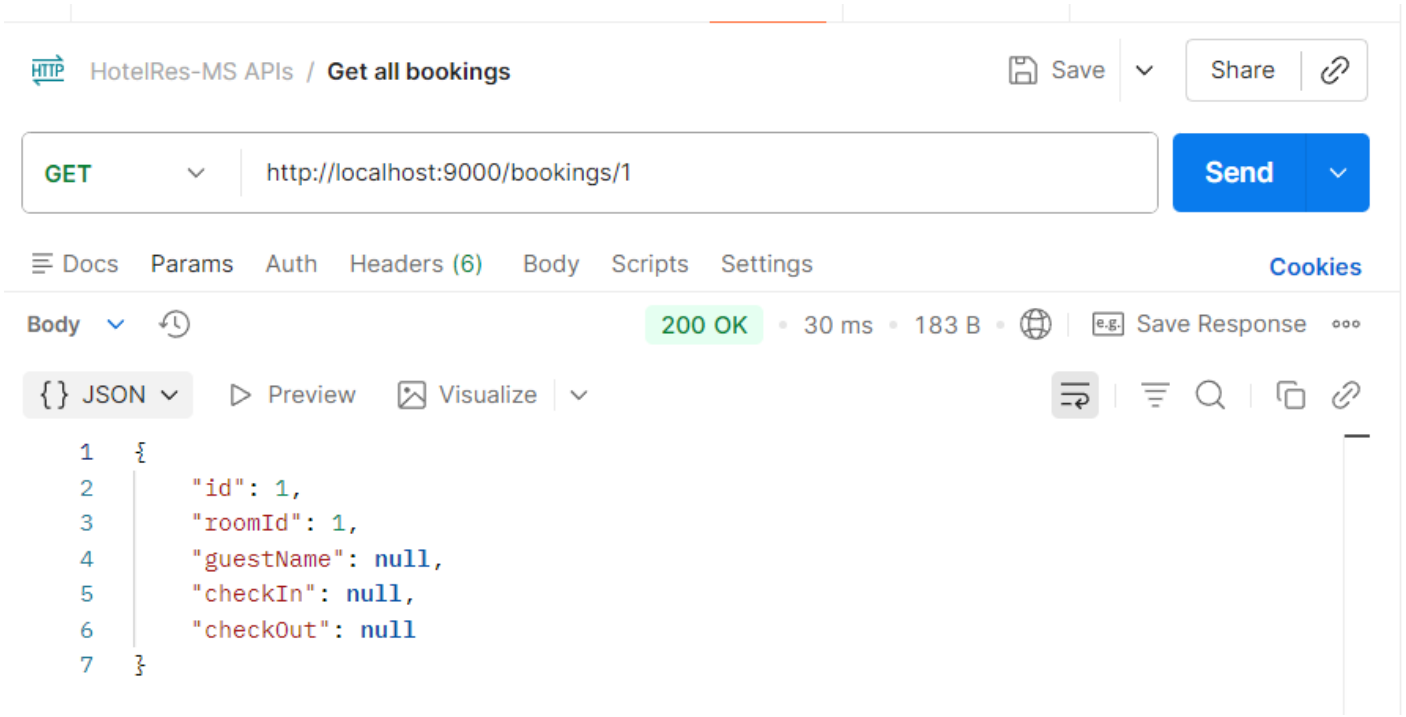
Read Booking by ID – GET /bookings/{id}

Example: GET /bookings/1

Response JSON:

json

```
{
  "id": 1,
  "roomId": 1,
  "guestName": "Omar Saleh",
  "checkIn": "2026-01-20",
  "checkOut": "2026-01-23"
}
```



http://localhost:9000/bookings/1 (screenshot)]

Read All Bookings – GET /bookings

Response JSON:

json

```
[
  {
    "id": 1,
    "roomId": 1,
    "guestName": "Omar Saleh",
    "checkIn": "2026-01-20",
    "checkOut": "2026-01-23"
  }
]
```

HotelRes-MS APIs / **Get all bookings**

Save | Share

GET http://localhost:9000/bookings **Send**

Docs Params Auth Headers (6) Body Scripts Settings Cookies

Body 200 OK • 3.30 s • 587 B Save Response

{} JSON Preview Visualize


```
[
    {
        "id": 1,
        "roomId": 1,
        "guestName": null,
        "checkIn": null,
        "checkOut": null
    },
    {
        "id": 2,
        "roomId": 1,
        "guestName": null,
        "checkIn": null,
        "checkOut": null
    },
    {
        "id": 3,
        "roomId": 1,
        "guestName": null,
        "checkIn": null,
        "checkOut": null
    },
    {
        "id": 4,
        "roomId": 2,
        "guestName": "Second Guest",
        "checkIn": "2026-01-20",
```

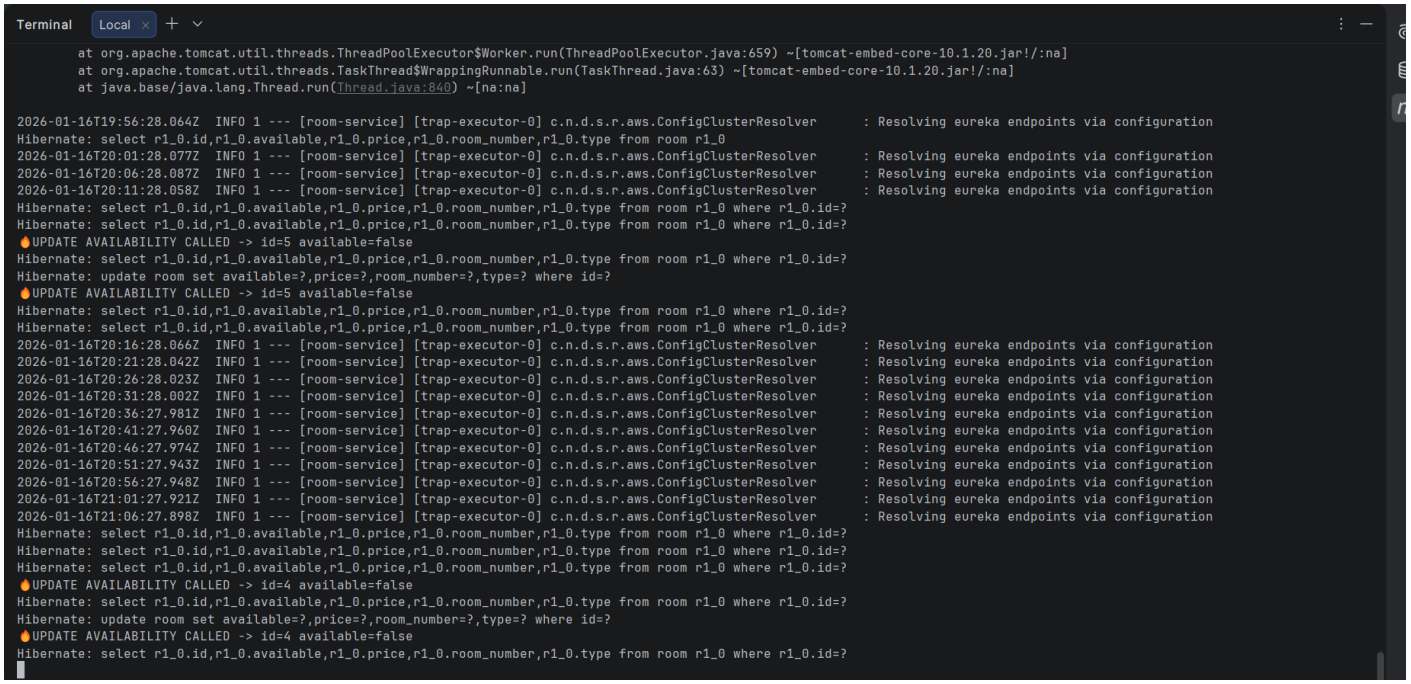
d) Inter-service Communication (booking-service ↔ room-service)

d.1 How booking-service reads room details and updates availability

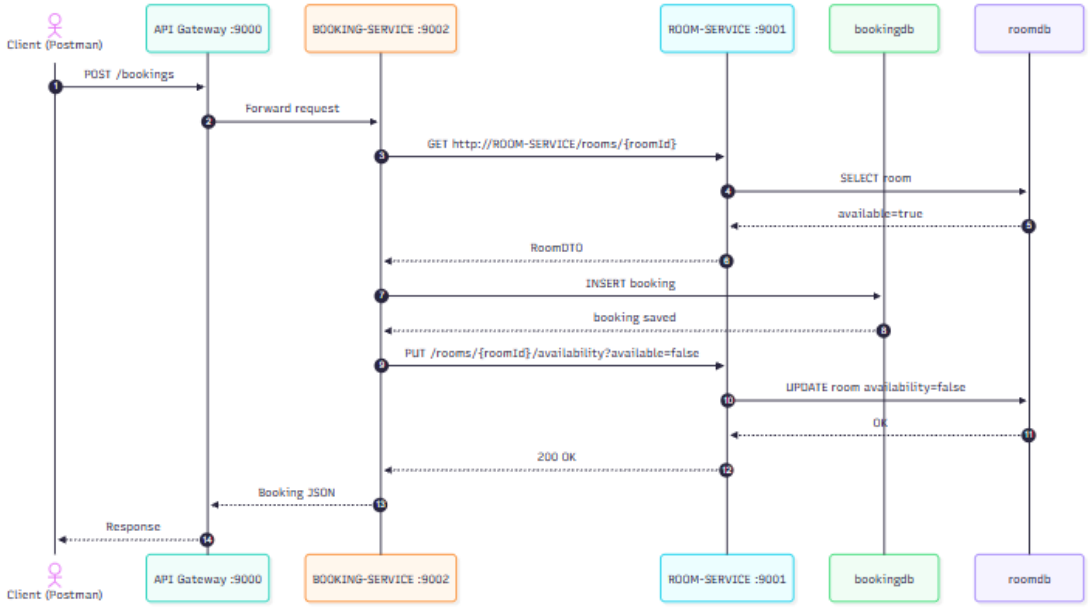
booking-service communicates with room-service using a Load-Balanced RestTemplate and Eureka service IDs. During booking creation, booking-service calls room-service to confirm the room exists and is available. After saving the booking, it updates the room availability to false. During booking deletion, it updates availability back to true.

Implemented calls (as required):

- GET http://ROOM-SERVICE/rooms/{roomId}
- PUT http://ROOM-SERVICE/rooms/{roomId}/availability?available=false (on create)
- PUT http://ROOM-SERVICE/rooms/{roomId}/availability?available=true (on delete)



d.2 Sequence Diagram – Create Booking



d.3 End-to-end proof (Before/After availability)

Verification steps:

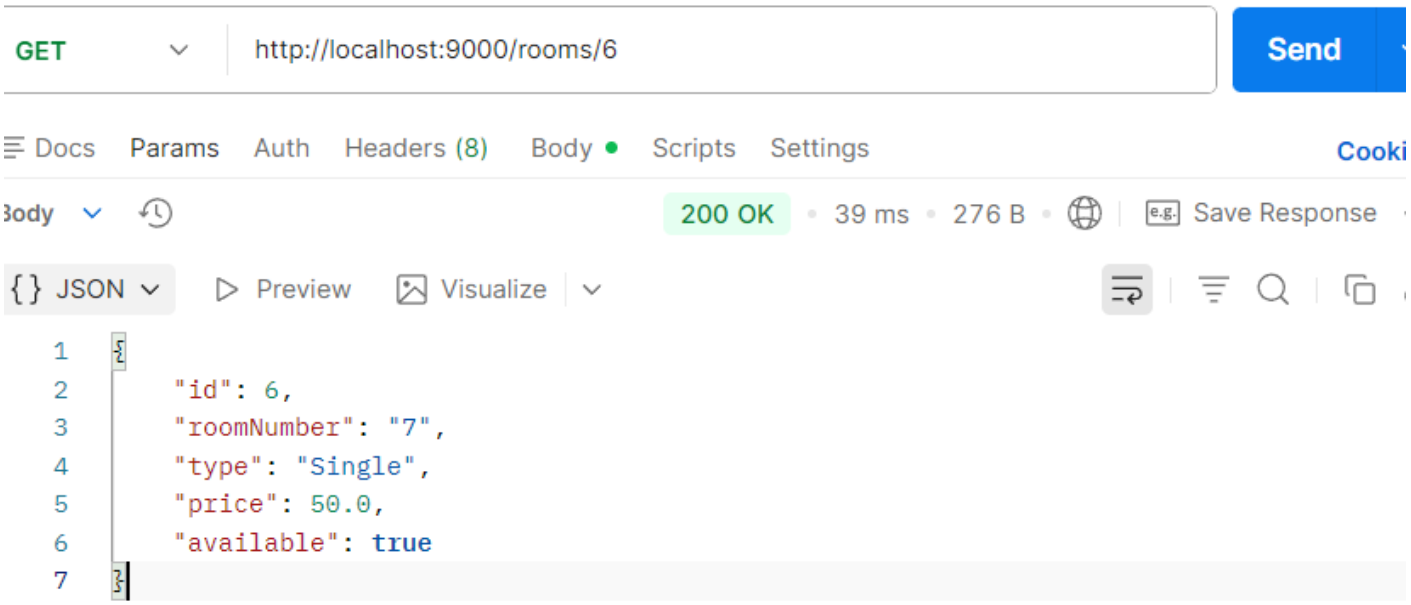
1) GET /rooms/{id} → available=true

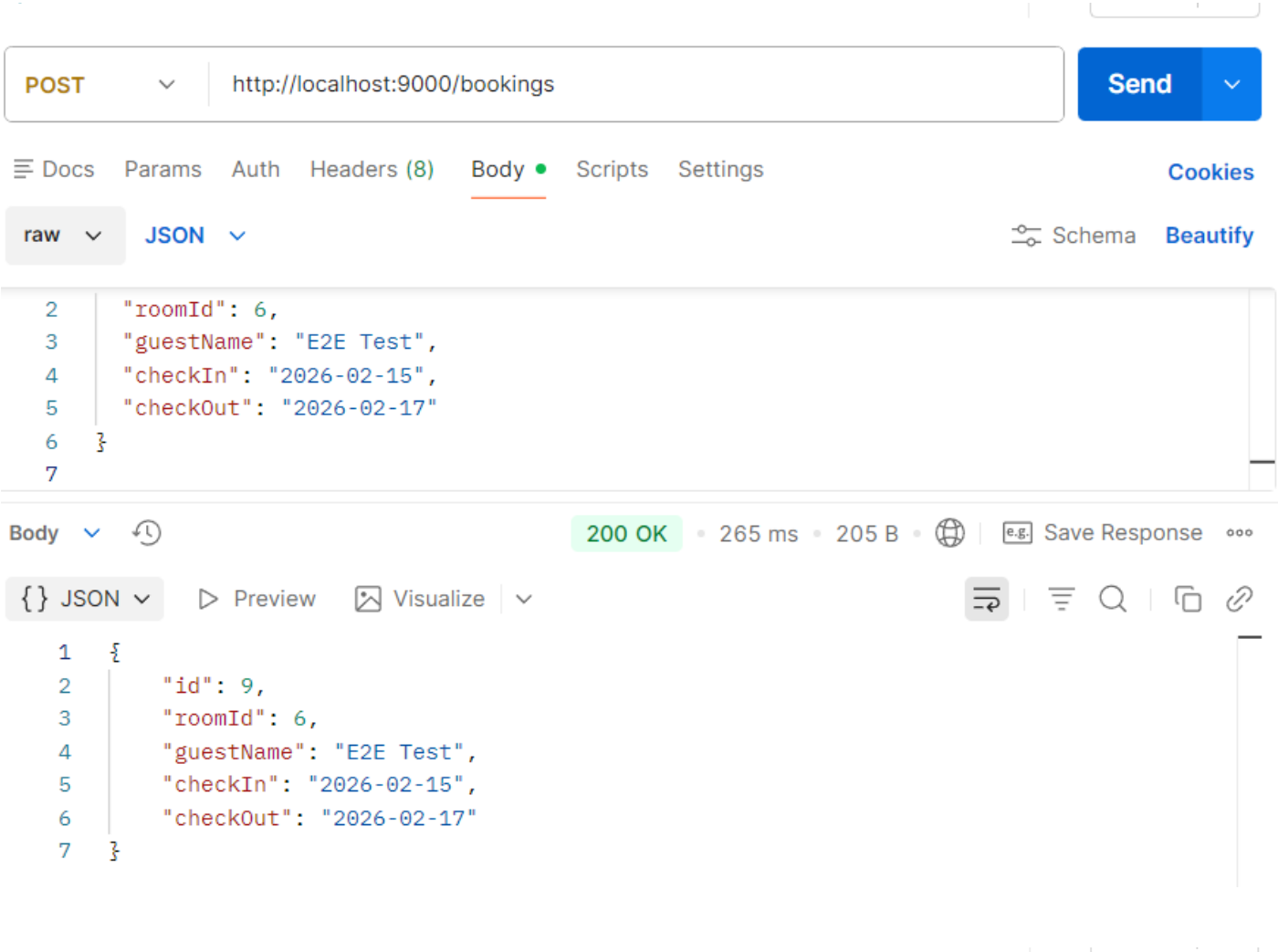
2) POST /bookings for the same room

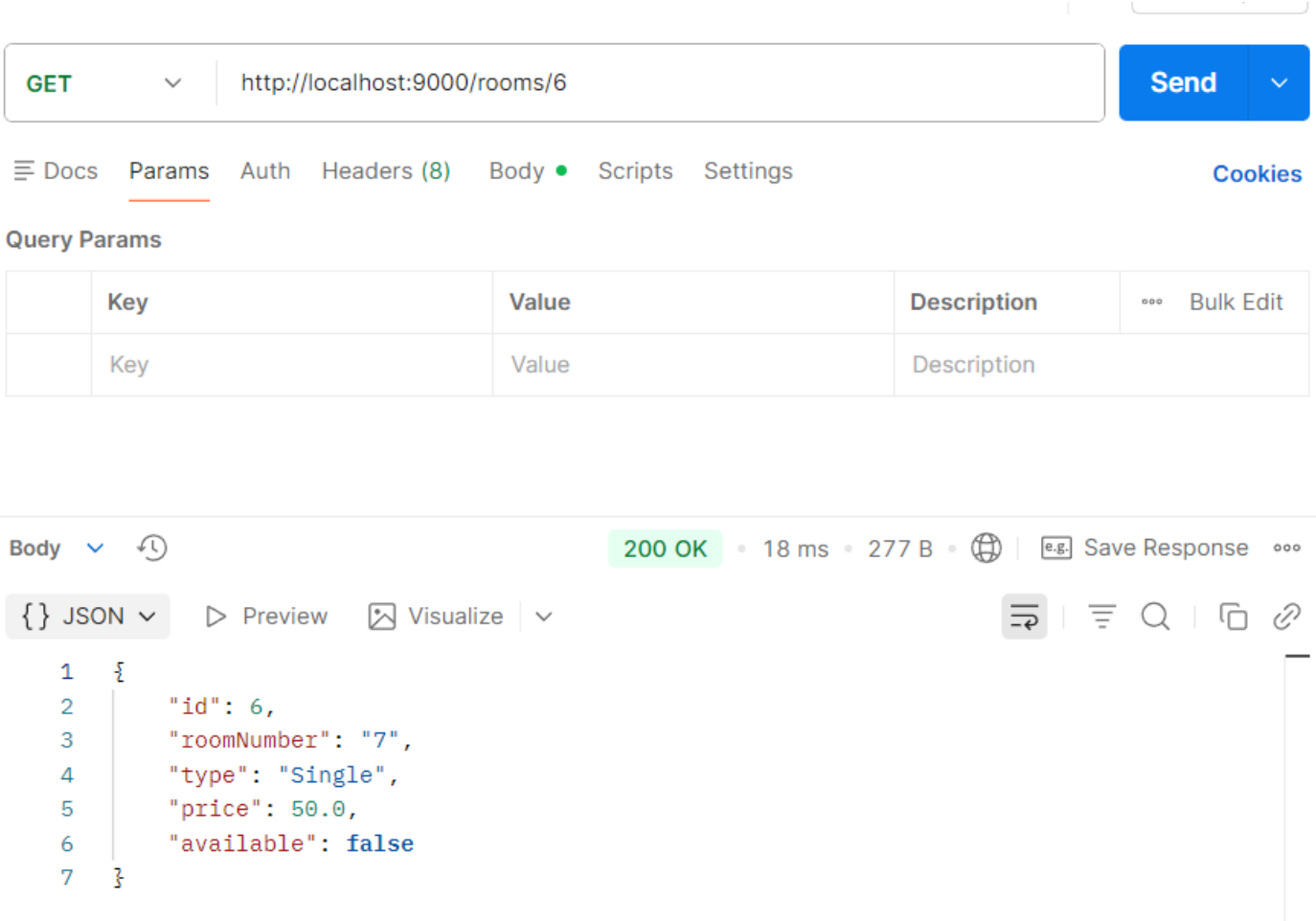
3) GET /rooms/{id} → available=false

4) DELETE /bookings/{id}

5) GET /rooms/{id} → available=true




POST
http://localhost:9000/bookings
Send
Docs
Params
Auth
Headers (8)
Body
Scripts
Settings
Cookies
raw
JSON
Schema
Beautify
2 "roomId": 6,
3 "guestName": "E2E Test",
4 "checkIn": "2026-02-15",
5 "checkOut": "2026-02-17"
6 }
7
Body
200 OK
265 ms
205 B
Save Response
JSON
Preview
Visualize
1 {
2 "id": 9,
3 "roomId": 6,
4 "guestName": "E2E Test",
5 "checkIn": "2026-02-15",
6 "checkOut": "2026-02-17"
7 }


GET
http://localhost:9000/rooms/6
Send
Docs
Params
Auth
Headers (8)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
200 OK
18 ms
277 B
Save Response
JSON
Preview
Visualize
1 {
2 "id": 6,
3 "roomNumber": "7",
4 "type": "Single",
5 "price": 50.0,
6 "available": false
7 }

DELETE http://localhost:9000/bookings/9 Send

Docs Params Auth Headers (8) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body 204 No Content 512 ms 64 B Save Response

Raw Preview Visualize

```
1
```

GET http://localhost:9000/rooms/6 Send

Docs Params Auth Headers (8) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body 200 OK 29 ms 276 B Save Response

{} JSON Preview Visualize


```
{
    "id": 6,
    "roomNumber": "7",
    "type": "Single",
    "price": 50.0,
    "available": true
}
```

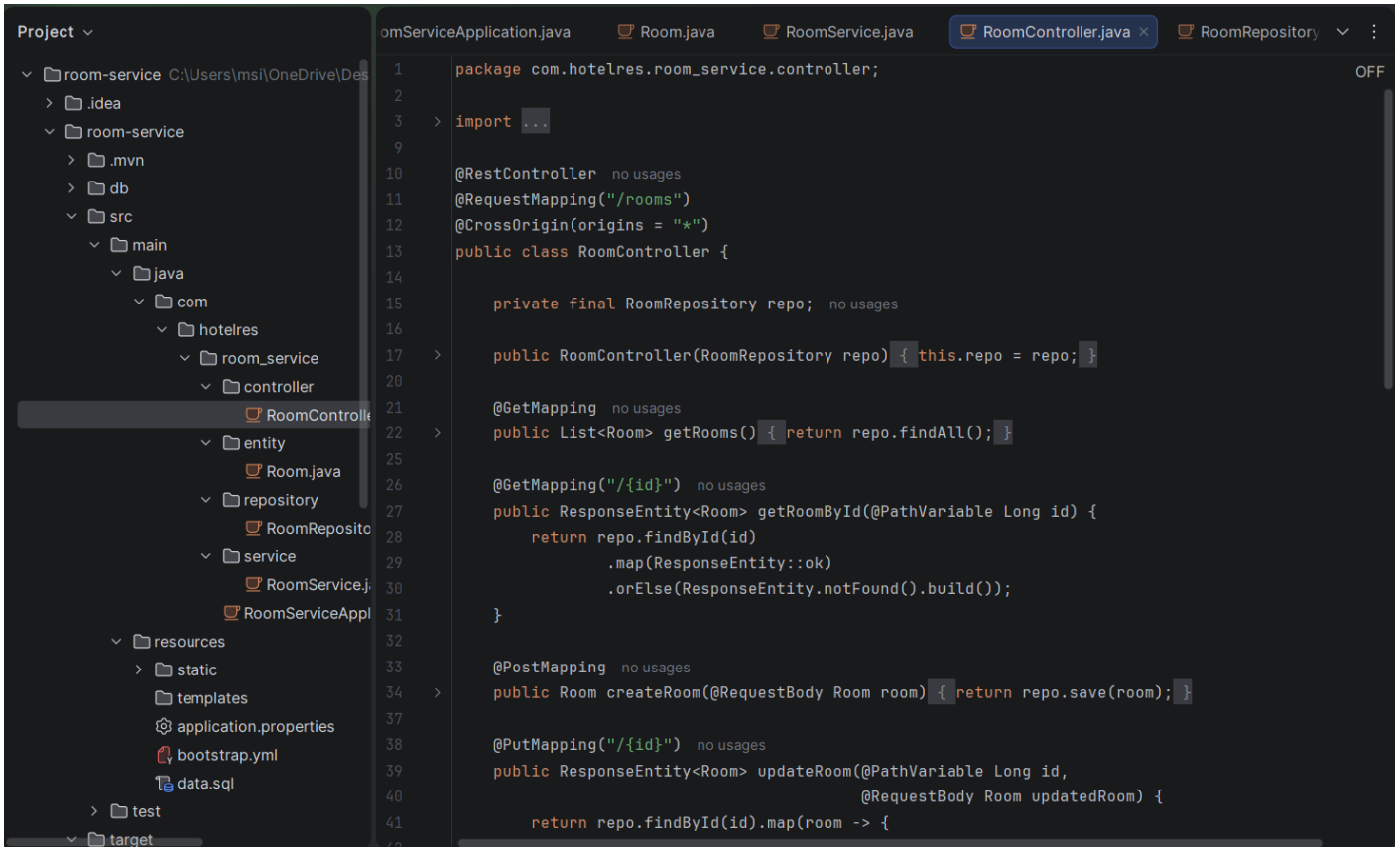
Implemntation Documantation

4.1 Room Service (room-service)

Purpose:
Manage room data in the system, including room number, type, price, and availability, and expose REST endpoints used by both clients and the booking-service.

4.1.1 RoomController.java

The **RoomController.java** class was implemented to handle all room management operations in the **room-service**.
It provides full REST-based CRUD functionality, allowing rooms to be created, retrieved, updated, and deleted.
A dedicated endpoint for updating room availability is also implemented using PUT /rooms/{id}/availability.
This endpoint is used by the **booking-service** during booking creation and deletion to ensure that rooms cannot be double-booked.
During update operations, availability is only modified when it is explicitly provided in the request, preventing unintended changes to the room state.
Additionally, a log statement is included in the availability update endpoint to help trace inter-service communication between the booking-service and the room-service during runtime.

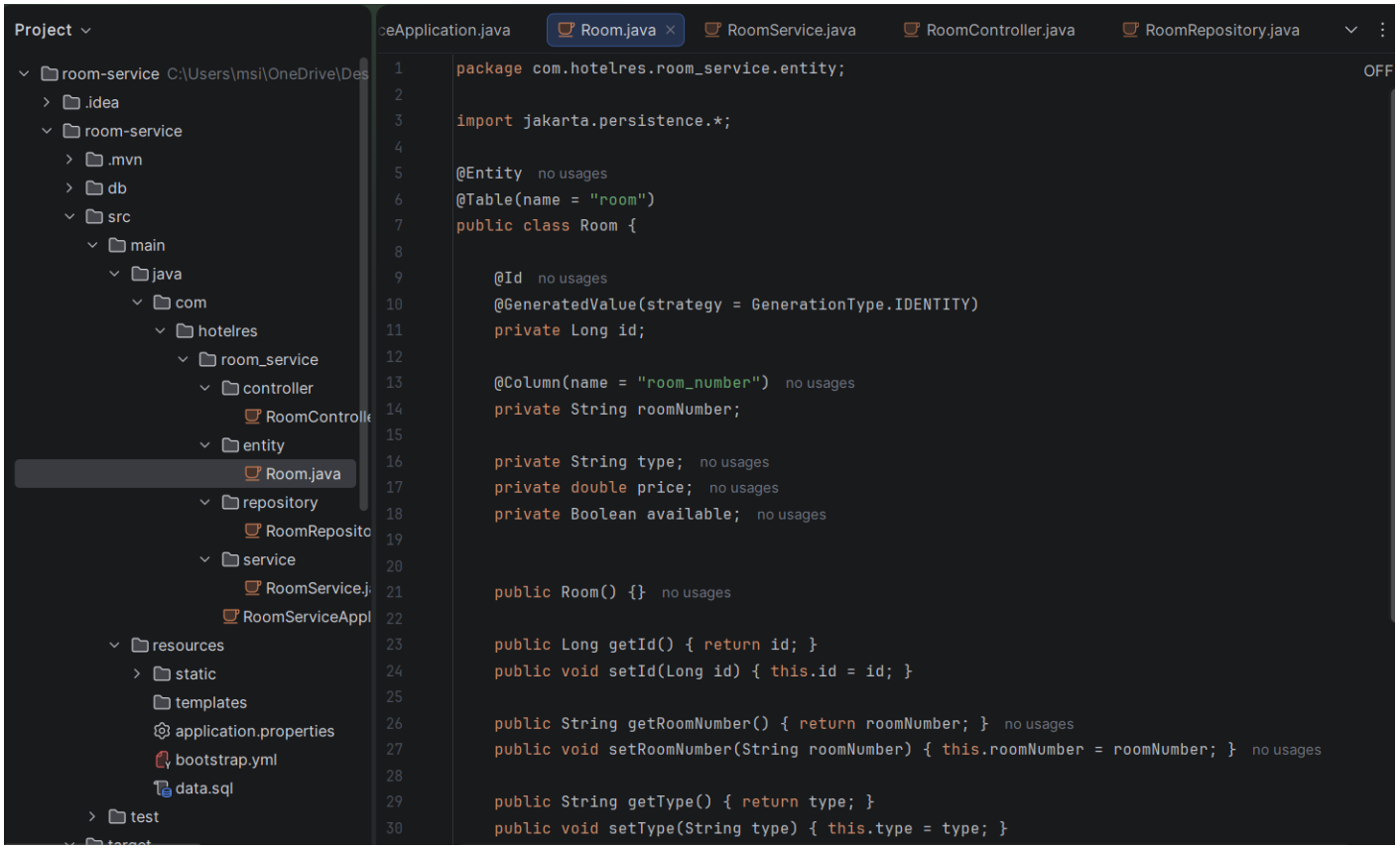


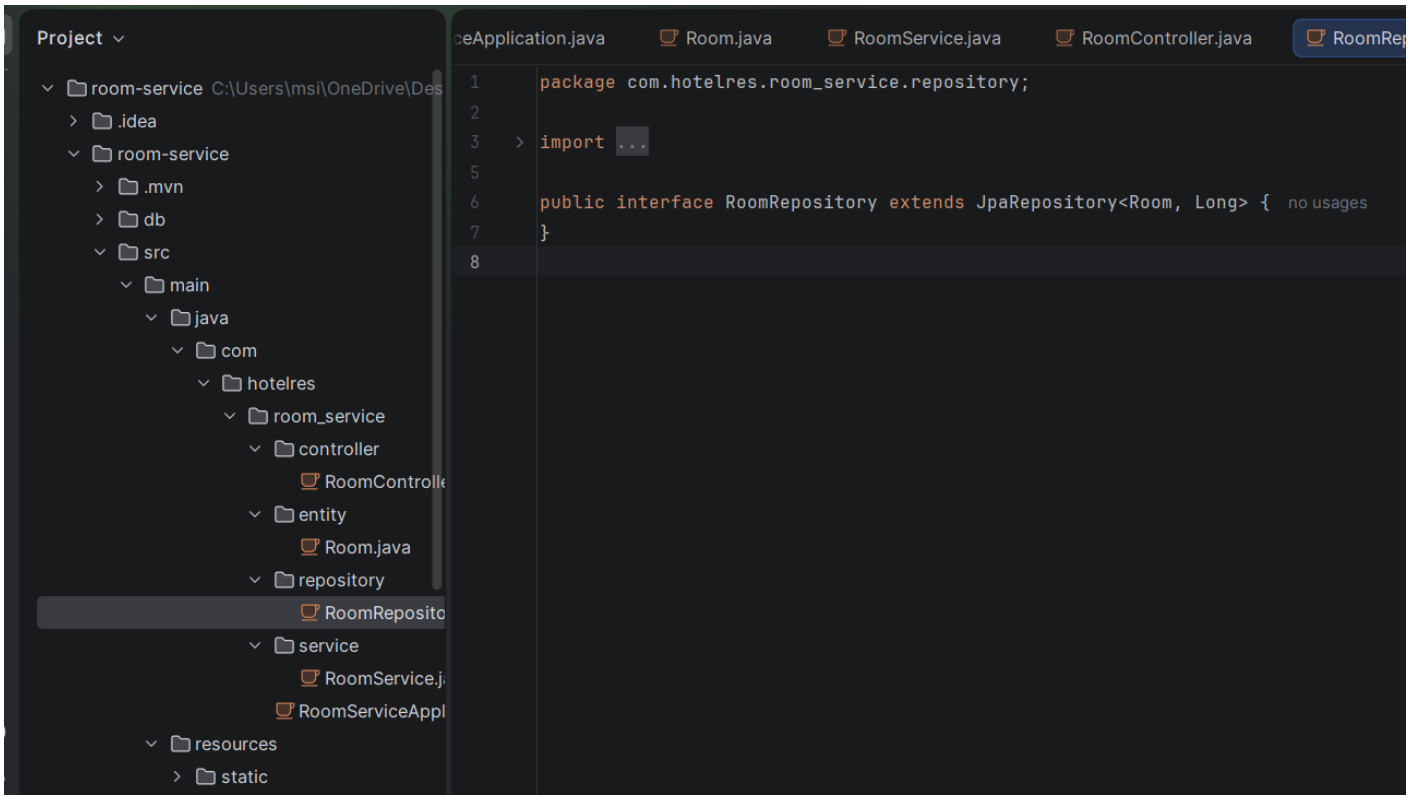
4.1.2 Room Entity + Repository

The **Room** entity class represents the room table in the database and defines the structure of room data used by the room-service.
It is annotated with **JPA annotations** to map the class and its fields to the corresponding database table and columns.

The @Entity and @Table annotations map the class to the room table, while the @Id and @GeneratedValue annotations define the primary key and auto-increment strategy.
Each field in the class corresponds to a column in the database, including room number, type, price, and availability.

The **RoomRepository** interface extends JpaRepository, providing built-in CRUD operations such as save, find, update, and delete without requiring manual SQL queries.
This repository is used by the controller to interact with the database in a clean and efficient way.


Project
room-service C:\Users\msi\OneDrive\Des
.idea
room-service
.mvn
db
src
main
java
com
hotelres
room_service
controller
RoomControlle
entity
Room.java
repository
RoomReposito
service
RoomService.j
RoomServiceAppl
resources
static
templates
application.properties
bootstrap.yml
data.sql
test
ceApplication.java
Room.java
RoomService.java
RoomController.java
RoomRepository.java
OFF
1 package com.hotelres.room_service.entity;
3 import jakarta.persistence.*;
5 @Entity no usages
6 @Table(name = "room")
7 public class Room {
9 @Id no usages
10 @GeneratedValue(strategy = GenerationType.IDENTITY)
11 private Long id;
13 @Column(name = "room_number") no usages
14 private String roomNumber;
16 private String type; no usages
17 private double price; no usages
18 private Boolean available; no usages
21 public Room() {} no usages
23 public Long getId() { return id; }
24 public void setId(Long id) { this.id = id; }
26 public String getRoomNumber() { return roomNumber; } no usages
27 public void setRoomNumber(String roomNumber) { this.roomNumber = roomNumber; } no usages
29 public String getType() { return type; }
30 public void setType(String type) { this.type = type; }


Project
room-service C:\Users\msi\OneDrive\Des
.idea
room-service
.mvn
db
src
main
java
com
hotelres
room_service
controller
RoomControlle
entity
Room.java
repository
RoomReposito
service
RoomService.j
RoomServiceAppl
resources
static
ceApplication.java
Room.java
RoomService.java
RoomController.java
RoomRe
1 package com.hotelres.room_service.repository;
3 import ...
6 public interface RoomRepository extends JpaRepository<Room, Long> { no usages
7 }
8

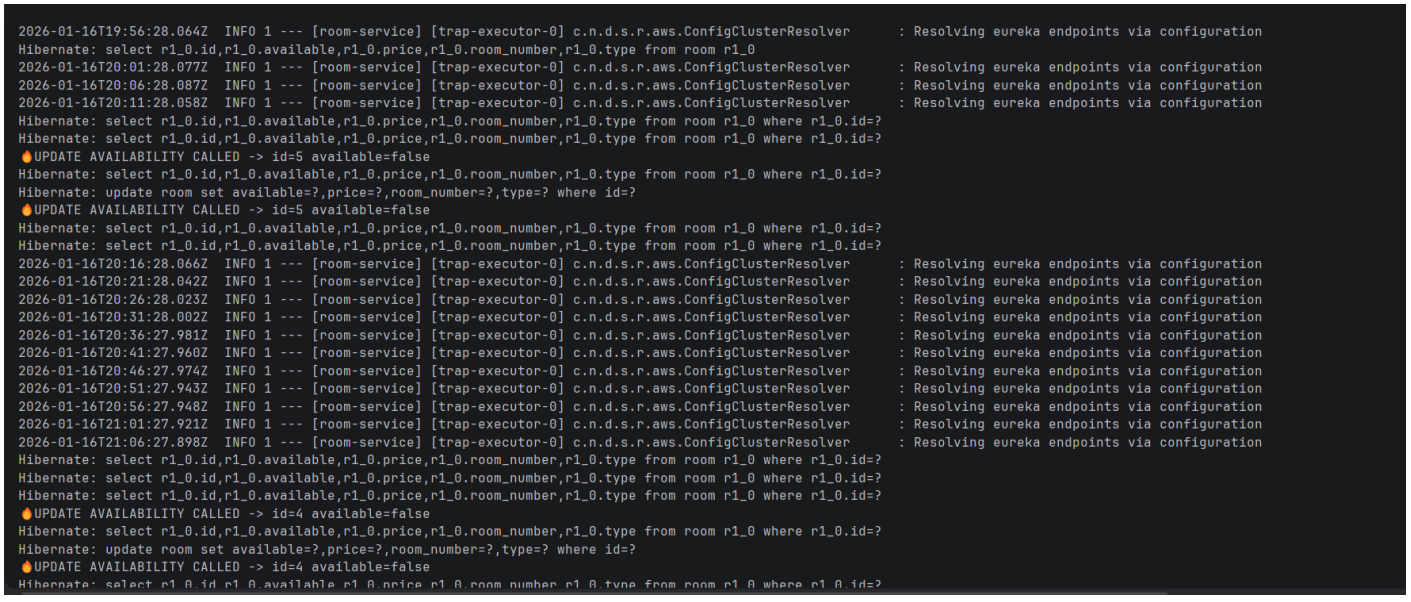
4.1.3 Discovery + Config Client Settings

The **room-service** is configured as a discovery client using **Eureka** and as a configuration client using **Spring Cloud Config**.
By annotating the main application class with @EnableDiscoveryClient, the service automatically registers itself with the Eureka Discovery Server at startup.

The property spring.application.name=room-service is used as the service identifier in Eureka and also as the key for loading external configuration from the **config-server**.
Configuration is loaded using the hostname config-server, which is resolved through Docker internal networking.

This setup allows the room-service to:

- Be discovered dynamically by other services (such as booking-service)
- Load centralized configuration without hardcoding environment-specific values



4.2 Booking Service (booking-service) – Implementation Details

Purpose:

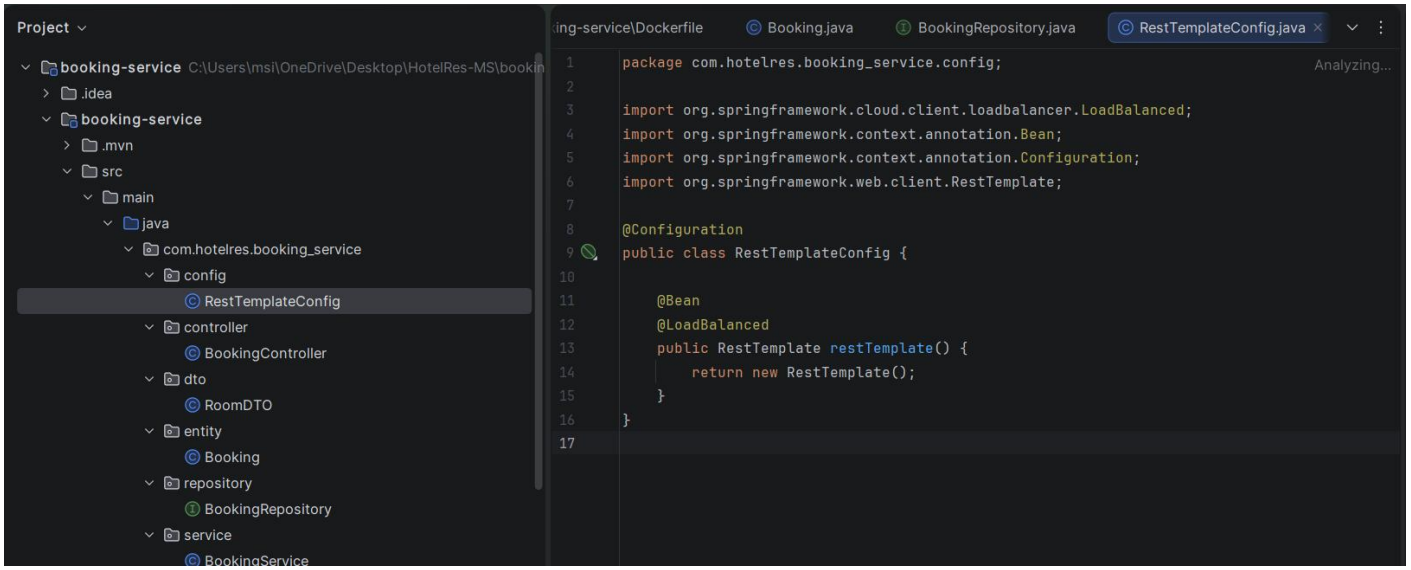
Manage booking operations and enforce room availability by communicating with the room-service during booking creation and deletion.

4.2.1 Load-Balanced RestTemplate (Eureka-based calls)

A **Load-Balanced RestTemplate** is configured in the booking-service to enable inter-service communication using **Eureka service discovery**.

By annotating the RestTemplate bean with @LoadBalanced, service names (such as ROOM-SERVICE) can be used instead of hardcoded hostnames or ports.

This allows the booking-service to dynamically resolve and communicate with room-service instances registered in Eureka, supporting loose coupling and scalability within the microservices architecture.



4.2.2 BookingController.java

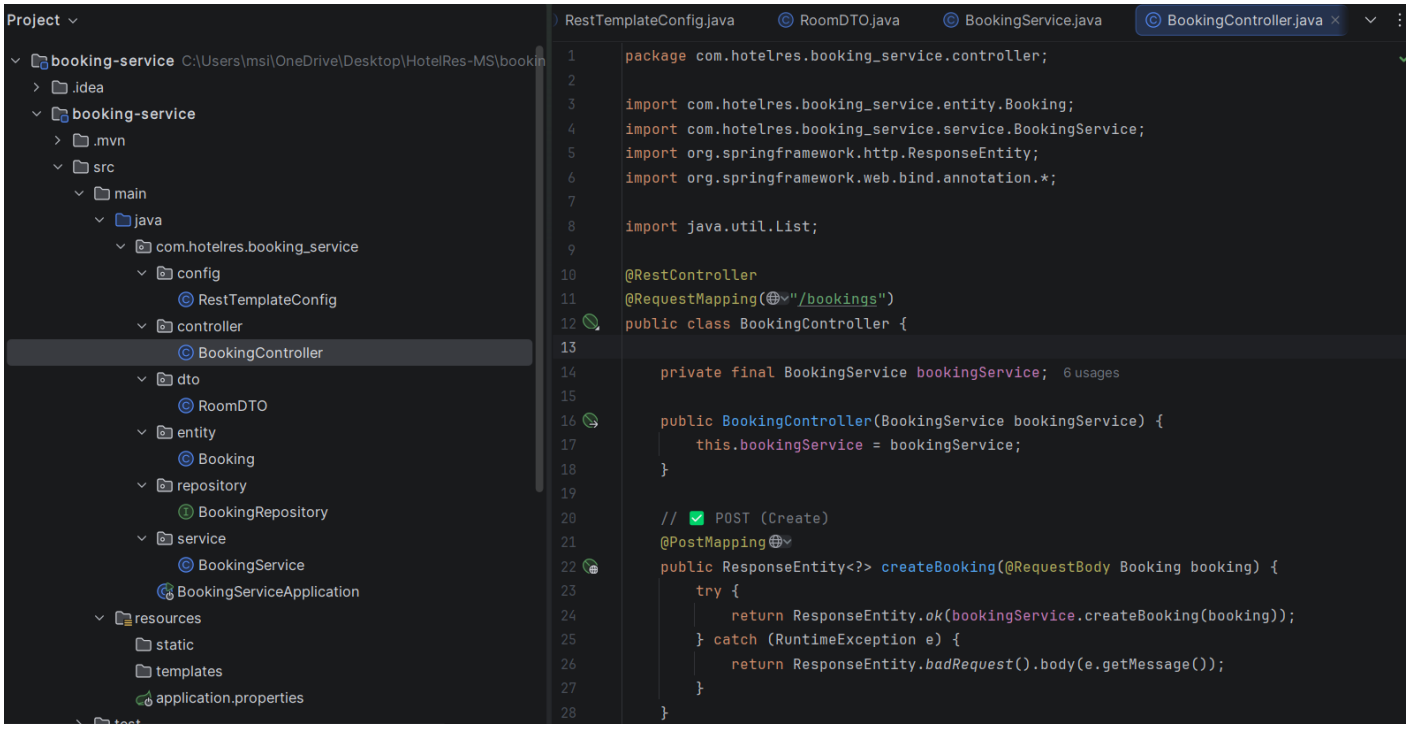
The **BookingController.java** class was implemented to handle all REST API requests related to booking management in the **booking-service**.
It acts as the entry point for client requests received through the API Gateway and delegates business logic execution to the **BookingService** layer.

The controller provides endpoints for:

- Creating new bookings
- Retrieving all bookings
- Retrieving a booking by its ID
- Updating existing bookings
- Deleting bookings

During booking creation and update, exceptions are handled explicitly to return a **Bad Request (400)** response when business rules are violated, such as attempting to book a room that is not available.
This ensures clear and user-friendly error handling for invalid booking operations.

The delete endpoint triggers the booking deletion logic, which also updates room availability through inter-service communication with the **room-service**.



4.2.3 DTO + Entity + Repository

The **RoomDTO** class is used as a data transfer object to receive room information from the **room-service** during inter-service communication.

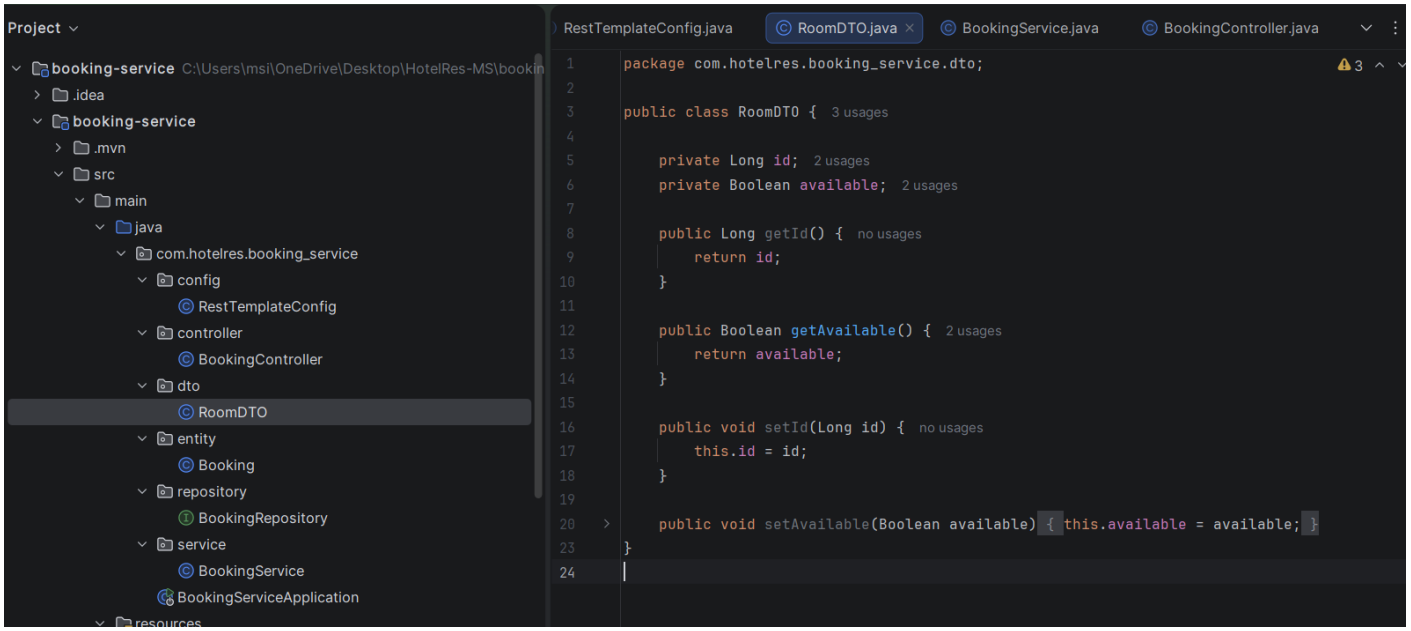
It contains only the fields required by the booking-service, such as the room ID and availability status, ensuring loose coupling between services.

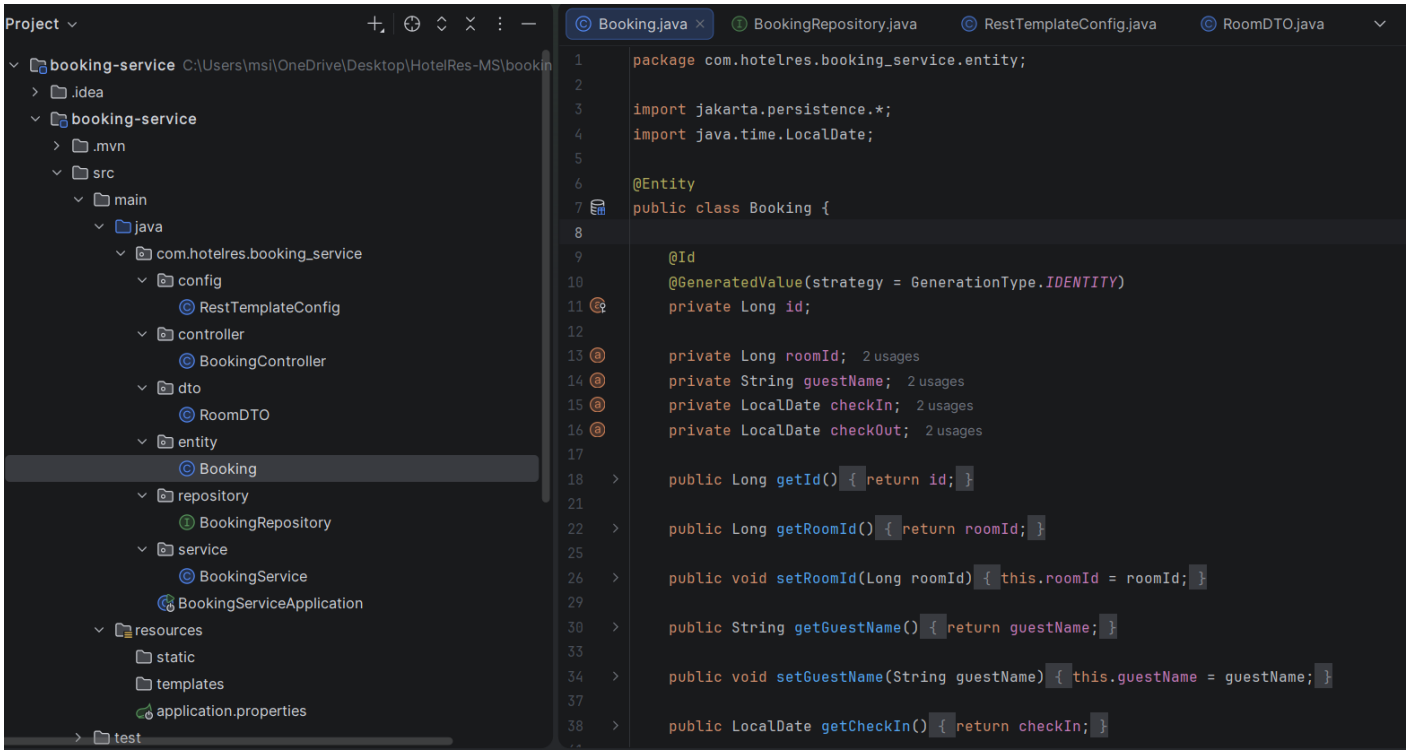
The **Booking** entity represents the booking table in the database and defines the structure of booking data managed by the booking-service.

It is annotated with JPA annotations to map the entity fields to database columns, including room ID, guest name, and check-in/check-out dates.

The **BookingRepository** interface extends JpaRepository, providing standard CRUD operations for the Booking entity without requiring explicit SQL queries.

This repository is used by the service layer to persist and retrieve booking data efficiently.





4.2.4 BookingService.java (Create/Delete business flow)

The **BookingService.java** class contains the core business logic for managing bookings in the **booking-service**.

It is responsible for enforcing room availability rules by communicating with the **room-service** during booking creation and deletion.

During **booking creation**, the service performs the following steps:

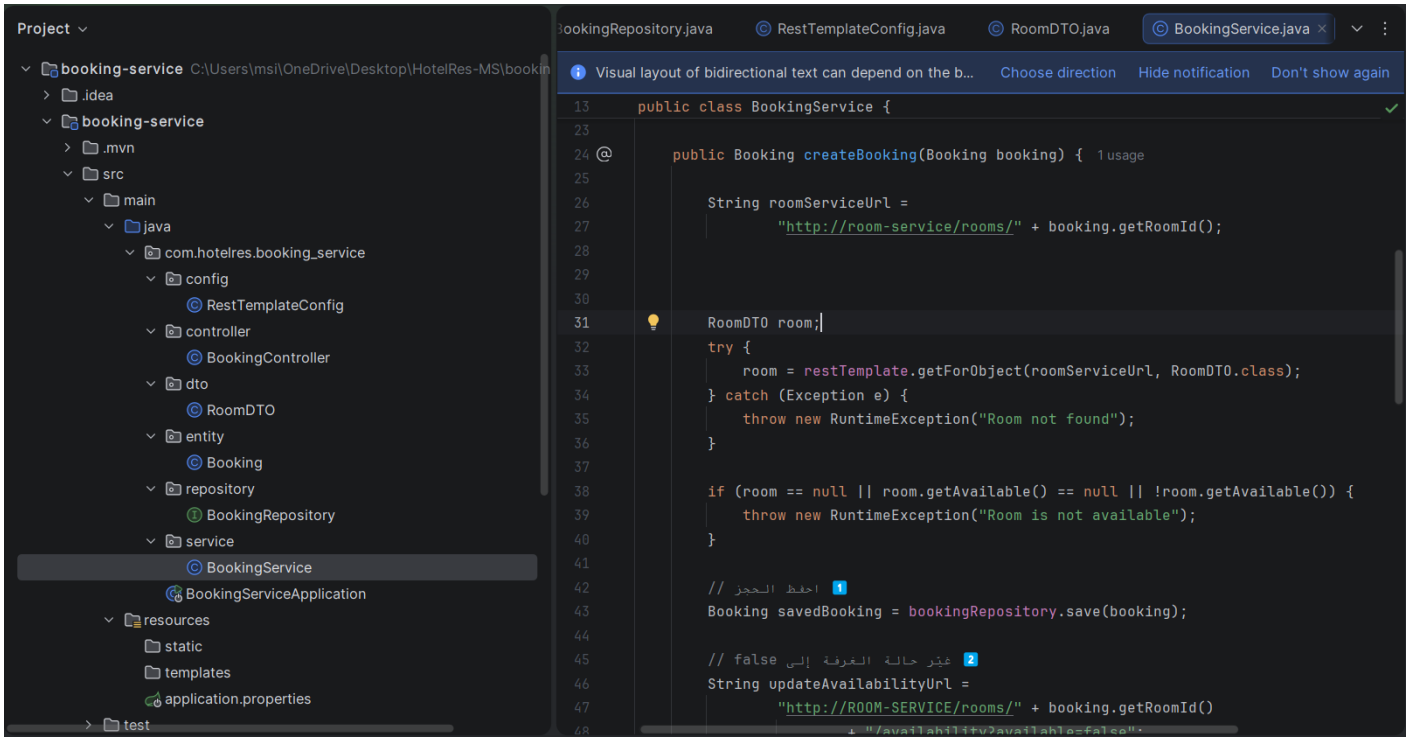
1. Calls the **room-service** using a load-balanced RestTemplate to verify that the room exists and is available.
2. Throws a runtime exception if the room is not found or not available.
3. Saves the booking in the booking database.
4. Sends a request to the **room-service** to update the room availability to false.

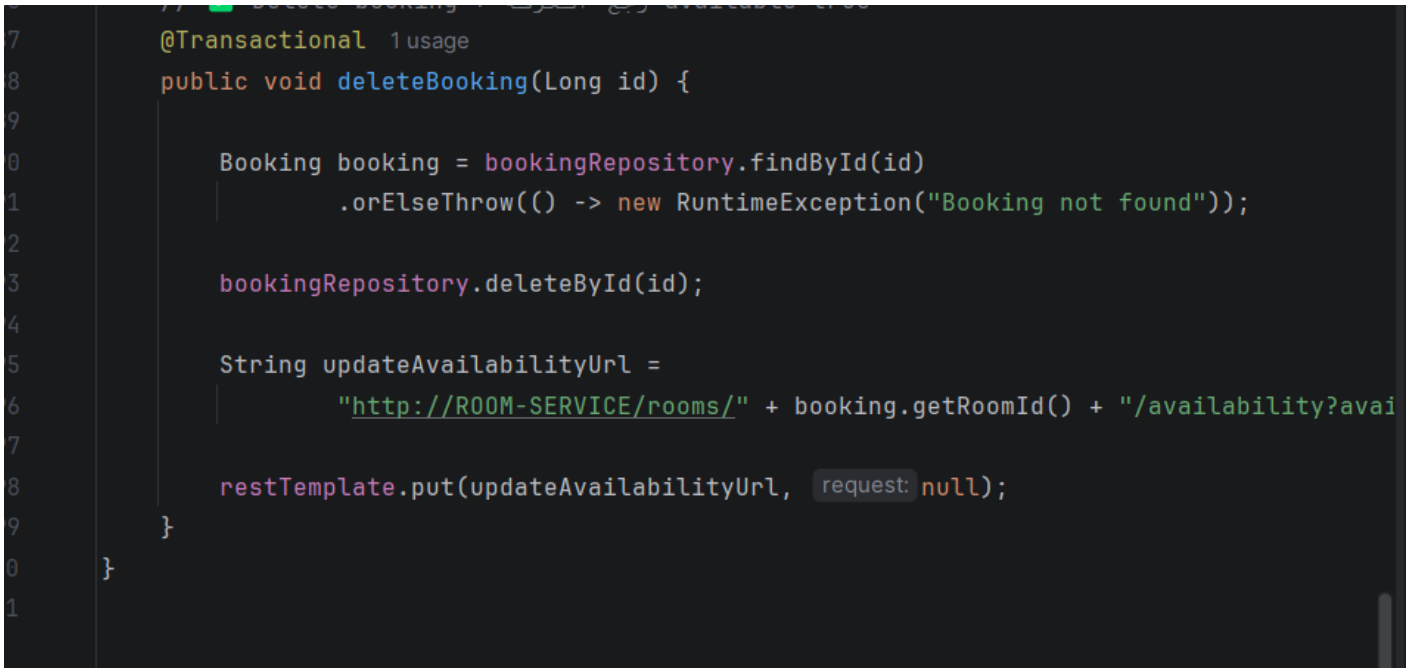
During **booking deletion**, the service:

1. Retrieves and deletes the booking from the database.
2. Calls the **room-service** to update the room availability back to true.

The delete operation is annotated with @Transactional to ensure data consistency between booking deletion and room availability update.

This implementation guarantees that room availability is always synchronized with booking state across services.





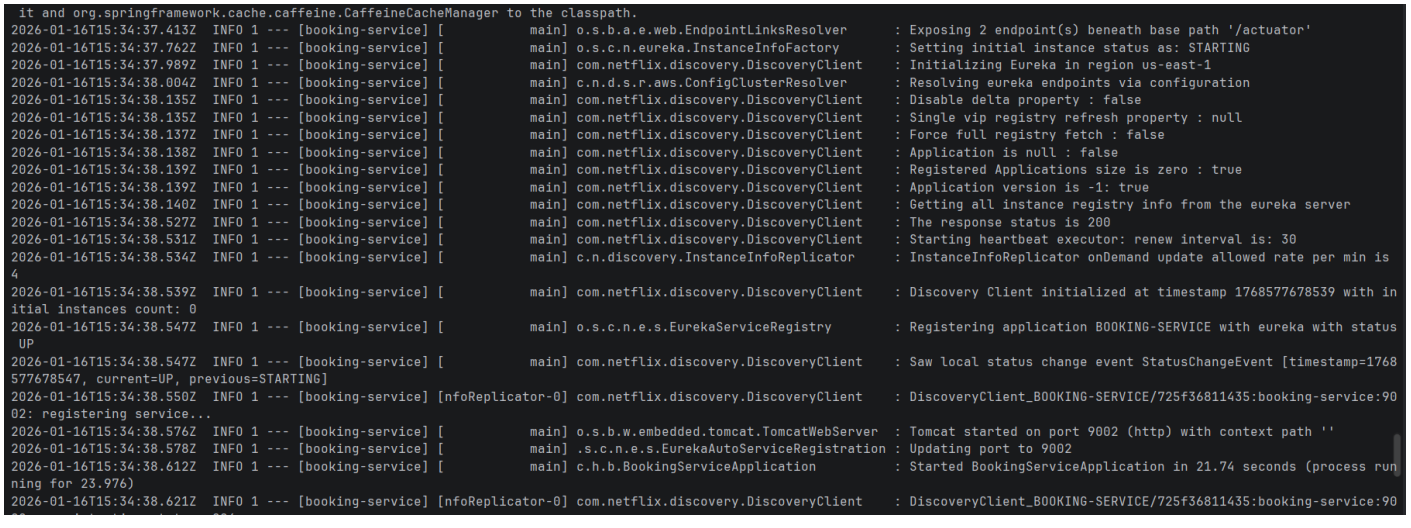
```
    @Transactional  1 usage
    public void deleteBooking(Long id) {

        Booking booking = bookingRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Booking not found"));

        bookingRepository.deleteById(id);

        String updateAvailabilityUrl =
                "http://ROOM-SERVICE/rooms/" + booking.getRoomId() + "/availability?avai

        restTemplate.put(updateAvailabilityUrl,  request: null);
    }
}
```

4.2.5 Discovery + Config Client Settings

The **booking-service** is configured as a discovery client using **Eureka** and as a configuration client using **Spring Cloud Config**.
By annotating the main application class with @EnableDiscoveryClient, the service automatically registers itself with the Eureka Discovery Server at startup.

The property spring.application.name=booking-service is used as the service identifier in Eureka and also to retrieve the corresponding configuration from the **config-server**.
External configuration is loaded from the config-server using Docker internal networking via the hostname config-server.

This configuration enables the booking-service to dynamically participate in service discovery and to load centralized configuration without hardcoding environment-specific values.



```
 it and org.springframework.cache.caffeine.CaffeineCacheManager to the classpath.
2026-01-16T15:34:37.413Z  INFO 1 --- [booking-service] [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 2 endpoint(s) beneath base path '/actuator'
2026-01-16T15:34:37.762Z  INFO 1 --- [booking-service] [           main] o.s.c.n.eureka.InstanceInfoFactory       : Setting initial instance status as: STARTING
2026-01-16T15:34:37.989Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Initializing Eureka in region us-east-1
2026-01-16T15:34:38.004Z  INFO 1 --- [booking-service] [           main] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2026-01-16T15:34:38.135Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Disable delta property : false
2026-01-16T15:34:38.135Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Single vip registry refresh property : null
2026-01-16T15:34:38.137Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Force full registry fetch : false
2026-01-16T15:34:38.138Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Application is null : false
2026-01-16T15:34:38.139Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Registered Applications size is zero : true
2026-01-16T15:34:38.139Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Application version is -1: true
2026-01-16T15:34:38.140Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Getting all instance registry info from the eureka server
2026-01-16T15:34:38.527Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : The response status is 200
2026-01-16T15:34:38.531Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Starting heartbeat executor: renew interval is: 30
2026-01-16T15:34:38.534Z  INFO 1 --- [booking-service] [           main] c.n.discovery.InstanceInfoReplicator     : InstanceInfoReplicator onDemand update allowed rate per min is
4
2026-01-16T15:34:38.539Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Discovery Client initialized at timestamp 1768577678539 with in
itial instances count: 0
2026-01-16T15:34:38.547Z  INFO 1 --- [booking-service] [           main] o.s.c.n.e.s.EurekaServiceRegistry        : Registering application BOOKING-SERVICE with eureka with status
 UP
2026-01-16T15:34:38.547Z  INFO 1 --- [booking-service] [           main] com.netflix.discovery.DiscoveryClient    : Saw local status change event StatusChangeEvent [timestamp=1768
577678547, current=UP, previous=STARTING]
2026-01-16T15:34:38.550Z  INFO 1 --- [booking-service] [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_BOOKING-SERVICE/725f36811435:booking-service:90
02: registering service...
2026-01-16T15:34:38.576Z  INFO 1 --- [booking-service] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 9002 (http) with context path ''
2026-01-16T15:34:38.578Z  INFO 1 --- [booking-service] [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 9002
2026-01-16T15:34:38.612Z  INFO 1 --- [booking-service] [           main] c.h.b.BookingServiceApplication          : Started BookingServiceApplication in 21.74 seconds (process run
ning for 23.976)
2026-01-16T15:34:38.621Z  INFO 1 --- [booking-service] [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_BOOKING-SERVICE/725f36811435:booking-service:90
```

4.2.6 Dockerfile + pom.xml

The **Dockerfile** defines how the booking-service is containerized using a Java 17 base image. It copies the packaged Spring Boot JAR into the container, exposes port **9002**, and runs the application using the Java runtime.

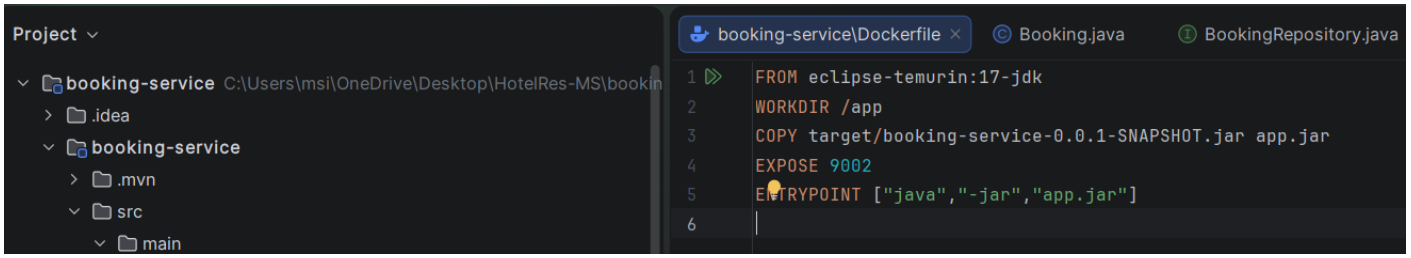
This ensures the booking-service can be deployed consistently inside Docker as an independent microservice.

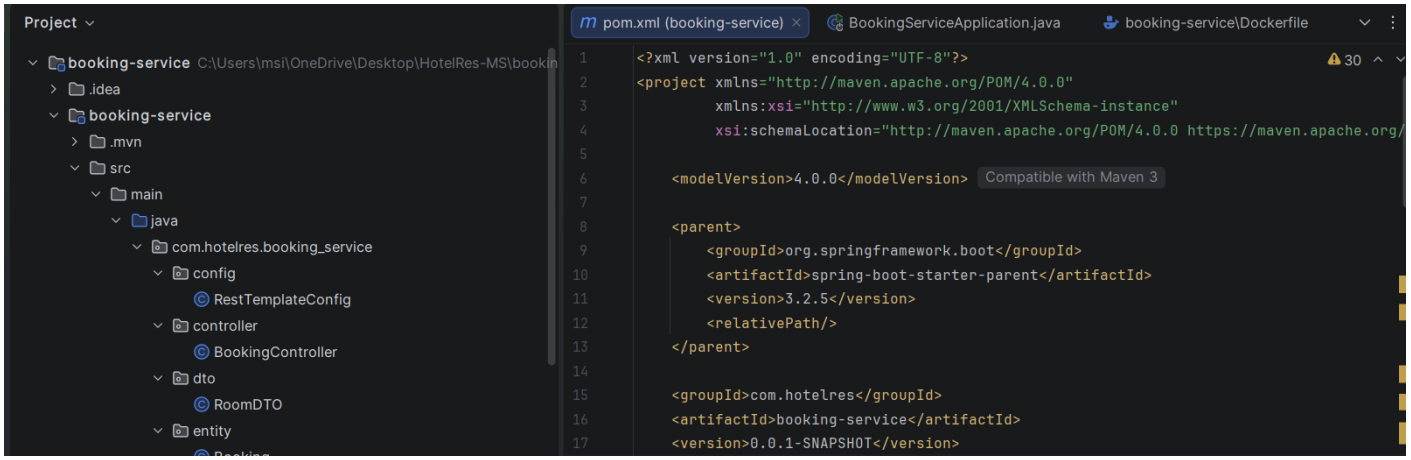
The **pom.xml** file defines the Maven build configuration and all required dependencies for the booking-service.

Key dependencies include:

1. **Spring Web** for building REST APIs
2. **Spring Data JPA** for database persistence
3. **MySQL Connector** for connecting to the MySQL database
4. **Eureka Client** for service discovery and registration
5. **Spring Cloud LoadBalancer** to support service-name-based calls (used with @LoadBalanced RestTemplate)
6. **Spring Cloud Config** to load centralized configuration
7. **Spring Boot Actuator** for monitoring and health endpoints

Together, these files enable the booking-service to be built, packaged, and executed reliably within the microservices environment.





4.3 API Gateway (api-gateway) – Implementation Details

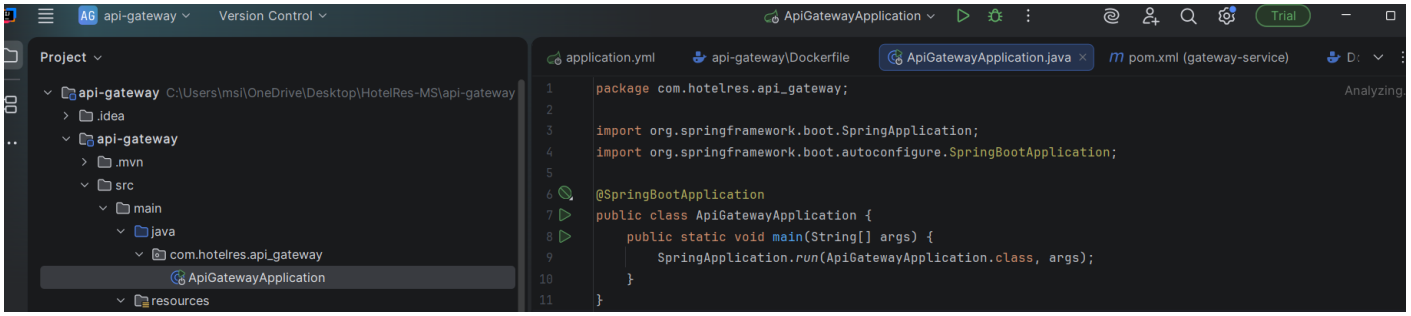
Purpose:

Provide a single entry point for clients and route incoming requests to internal microservices using service discovery.

4.3.1 Main Class + Configuration

The **ApiGatewayApplication** class represents the entry point of the API Gateway service. The gateway is configured as a Spring Boot application that retrieves its external configuration from the **config-server** and registers itself with the **Eureka Discovery Server**. The application configuration defines the service name as api-gateway, enabling it to be discovered by other services.

Eureka settings specify the discovery-server location, allowing the gateway to resolve service instances dynamically.



4.3.2 Gateway Routes (Rooms + Bookings)

The API Gateway routes are configured to forward client requests to the appropriate backend services.

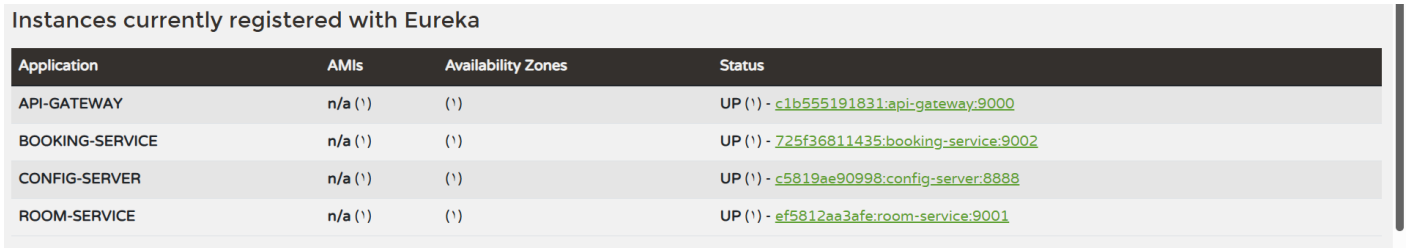
Requests matching /rooms/ are routed to **ROOM-SERVICE**, while requests matching /bookings/ are routed to **BOOKING-SERVICE**.

The routes use load-balanced URIs (lb://SERVICE-ID), which rely on Eureka service discovery to dynamically resolve service instances.

This approach removes the need for hardcoded service addresses and supports scalability and fault tolerance.

4.3.3 Dockerfile

The **Dockerfile** defines how the API Gateway is built and packaged into a Docker image.
It uses a multi-stage build to compile the application with Maven and then run it using a Java 17 runtime image.
The container exposes port **9000**, allowing clients to access the system through a single entry point.


Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - c1b555191831:api-gateway:9000
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - 725f36811435:booking-service:9002
CONFIG-SERVER	n/a (1)	(1)	UP (1) - c5819ae90998:config-server:8888
ROOM-SERVICE	n/a (1)	(1)	UP (1) - ef5812aa3afe:room-service:9001

4.4 Discovery Server (Eureka) – discovery-server

Purpose:

Register all microservices and enable service-to-service discovery using service IDs instead of fixed network addresses.

4.4.1 Main Class + Configuration

The **DiscoveryServerApplication** class initializes the Eureka Discovery Server.
The server is configured not to register itself or fetch the registry, as it acts as a standalone discovery component.
Eureka runs on port **9011** and provides a central registry where all microservices register themselves and discover other services dynamically.

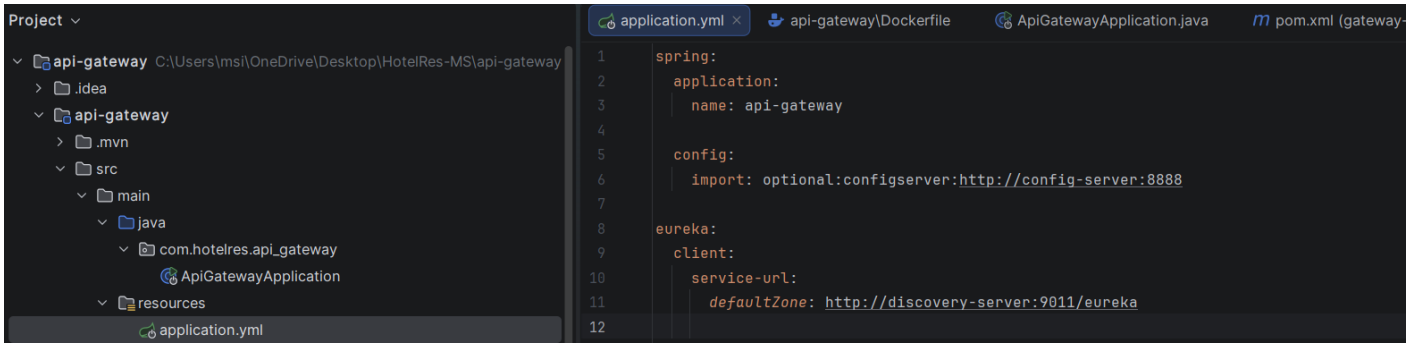
4.5 Config Server (Git-backed) – config-server

Purpose:

Provide centralized configuration management for all microservices. Configuration is loaded from a Git repository named **hotelres-config**, allowing services to externalize environment settings and avoid hardcoded values.

4.5.1 Main Class + Configuration

The **ConfigServerApplication** is implemented with @EnableConfigServer to enable Spring Cloud Config Server functionality.
It is also registered as a discovery client using @EnableDiscoveryClient, allowing the config-server itself to appear in Eureka.
The configuration runs on port **8888** and is connected to a Git repository (hotelres-config) as the backend configuration source.
Client services (room-service, booking-service, api-gateway) fetch their configuration from this server during startup.



4.5.2 Config Loading Verification (Located environment)

To verify that services successfully load configuration from the config-server, each client service prints a log line containing:

“Located environment”

This confirms that the service reached the config-server and loaded the remote configuration during startup.

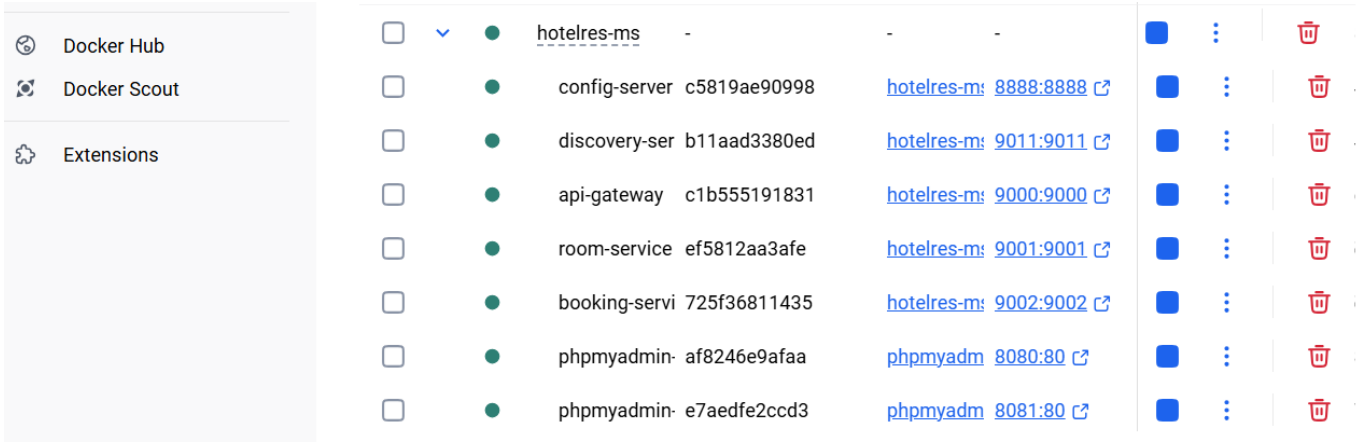
Example verification commands:

docker logs room-service | Select-String "Located environment"

docker logs booking-service | Select-String "Located environment"

docker logs api-gateway | Select-String "Located environment"

```
PS C:\Users\msi\OneDrive\Desktop\HotelRes-MS> docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED        STATUS                    PORTS                                                NAMES
c5819ae90998   hotelres-ms-config-server      "java -jar app.jar"      40 hours ago   Up 3 minutes              0.0.0.0:8888->8888/tcp, [::]:8888->8888/tcp          config-server
b11aad3380ed   hotelres-ms-discovery-server   "java -jar app.jar"      40 hours ago   Up 3 minutes              0.0.0.0:9011->9011/tcp, [::]:9011->9011/tcp          discovery-server
c1b555191831   hotelres-ms-api-gateway        "java -jar app.jar"      43 hours ago   Up 3 minutes              0.0.0.0:9000->9000/tcp, [::]:9000->9000/tcp          api-gateway
ef5812aa3afe   hotelres-ms-room-service       "java -jar app.jar"      43 hours ago   Up 3 minutes              0.0.0.0:9001->9001/tcp, [::]:9001->9001/tcp          room-service
725f36811435   hotelres-ms-booking-service    "java -jar app.jar"      43 hours ago   Up 3 minutes              0.0.0.0:9002->9002/tcp, [::]:9002->9002/tcp          booking-service
af8246e9afaa   phpmyadmin/phpmyadmin          "/docker-entrypoint.…"   43 hours ago   Up 3 minutes              0.0.0.0:8080->80/tcp, [::]:8080->80/tcp              phpmyadmin-room
e7aedfe2ccd3   phpmyadmin/phpmyadmin          "/docker-entrypoint.…"   43 hours ago   Up 3 minutes              0.0.0.0:8081->80/tcp, [::]:8081->80/tcp              phpmyadmin-booking
5c75e97d1856   mysql:8                        "docker-entrypoint.s…"   43 hours ago   Up 3 minutes (healthy)    0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp          mysql-room
6ee3939f2225   mysql:8                        "docker-entrypoint.s…"   43 hours ago   Up 3 minutes (healthy)    0.0.0.0:3308->3306/tcp, [::]:3308->3306/tcp          mysql-booking

PS C:\Users\msi\OneDrive\Desktop\HotelRes-MS> docker logs room-service | Select-String "Located environment"

2026-01-14T20:13:06.738Z  INFO 1 --- [room-service] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=room-service, profiles=[default],
label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=
2026-01-16T15:34:30.013Z  INFO 1 --- [room-service] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=room-service, profiles=[default],
label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=


PS C:\Users\msi\OneDrive\Desktop\HotelRes-MS> docker logs booking-service | Select-String "Located environment"

2026-01-14T20:13:10.823Z  INFO 1 --- [booking-service] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=booking-service,
profiles=[default], label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=
2026-01-16T15:34:25.989Z  INFO 1 --- [booking-service] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=booking-service,
profiles=[default], label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=


PS C:\Users\msi\OneDrive\Desktop\HotelRes-MS> docker logs api-gateway | Select-String "Located environment"

2026-01-14T20:13:10.123Z  INFO 1 --- [api-gateway] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=api-gateway, profiles=[default],
label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=
2026-01-16T15:34:27.615Z  INFO 1 --- [api-gateway] [           main] o.s.c.c.c.ConfigServerConfigDataLoader   : Located environment: name=api-gateway, profiles=[default],
label=null, version=53f860f7f5aa120e1a141b5a7d4d403824bf81be, state=
```



4.6.2 Ports Summary

The implementation uses the following ports:

- discovery-server (Eureka): 9011
- api-gateway: 9000
- config-server: 8888
- room-service: 9001
- booking-service: 9002
- phpmyadmin-room: 8080
- phpmyadmin-booking: 8081
- mysql-room: 3307 → 3306
- mysql-booking: 3308 → 3306

4.7 Testing / Verification

This section verifies that all system components are functioning correctly and that the microservices communicate as expected.

4.7.1 Eureka Registration Check

The Eureka Dashboard was used to verify that all microservices are successfully registered and running.

The following services were observed with status **UP**:

- API-GATEWAY
- ROOM-SERVICE
- BOOKING-SERVICE
- CONFIG-SERVER

This confirms that service discovery is correctly configured and that all services can be discovered dynamically.

4.7.2 Config Loading Check

To verify centralized configuration loading, service logs were checked for the message **"Located environment"**.

This message confirms that each client service successfully fetched its configuration from the config-server during startup.

Logs from room-service, booking-service, and api-gateway all contained this message, proving that Spring Cloud Config is functioning correctly.

4.7.3 Health Check (/actuator/health)

```
GET http://localhost:9000/actuator/health
GET http://localhost:9001/actuator/health
GET http://localhost:9002/actuator/health
GET http://localhost:8888/actuator/health
```

cmddddd