

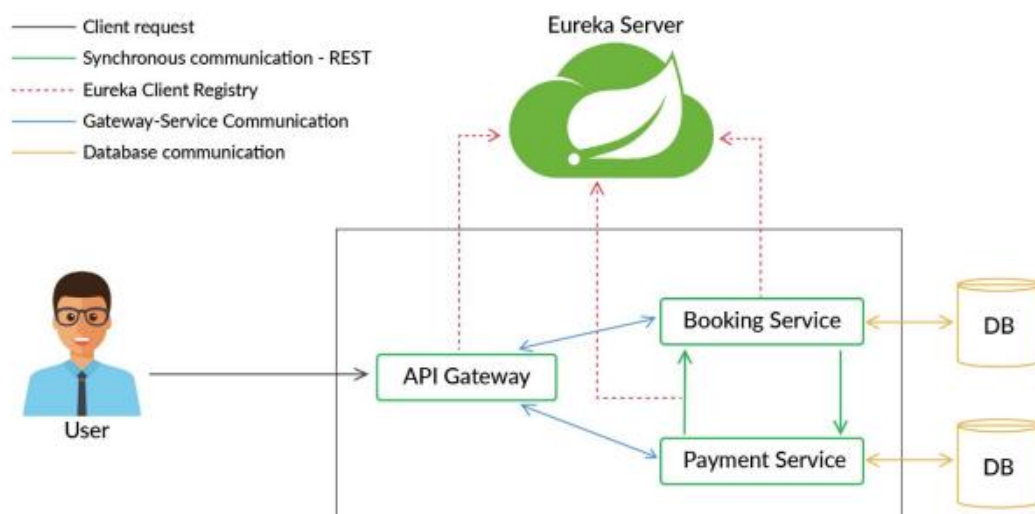
Hotel Room Booking Application

Project Overview

We are breaking the Hotel room booking application into three different microservices, which are as follows:

1. API-Gateway - This service is exposed to the outer world and is responsible for routing all requests to the microservices internally.
2. Booking service - This service is responsible for collecting all information related to user booking and sending a confirmation message once the booking is confirmed.
3. Payment service - This is a dummy payment service; this service is called by the booking service for initiating payment after confirming rooms.

Application Workflow



1. Booking Service

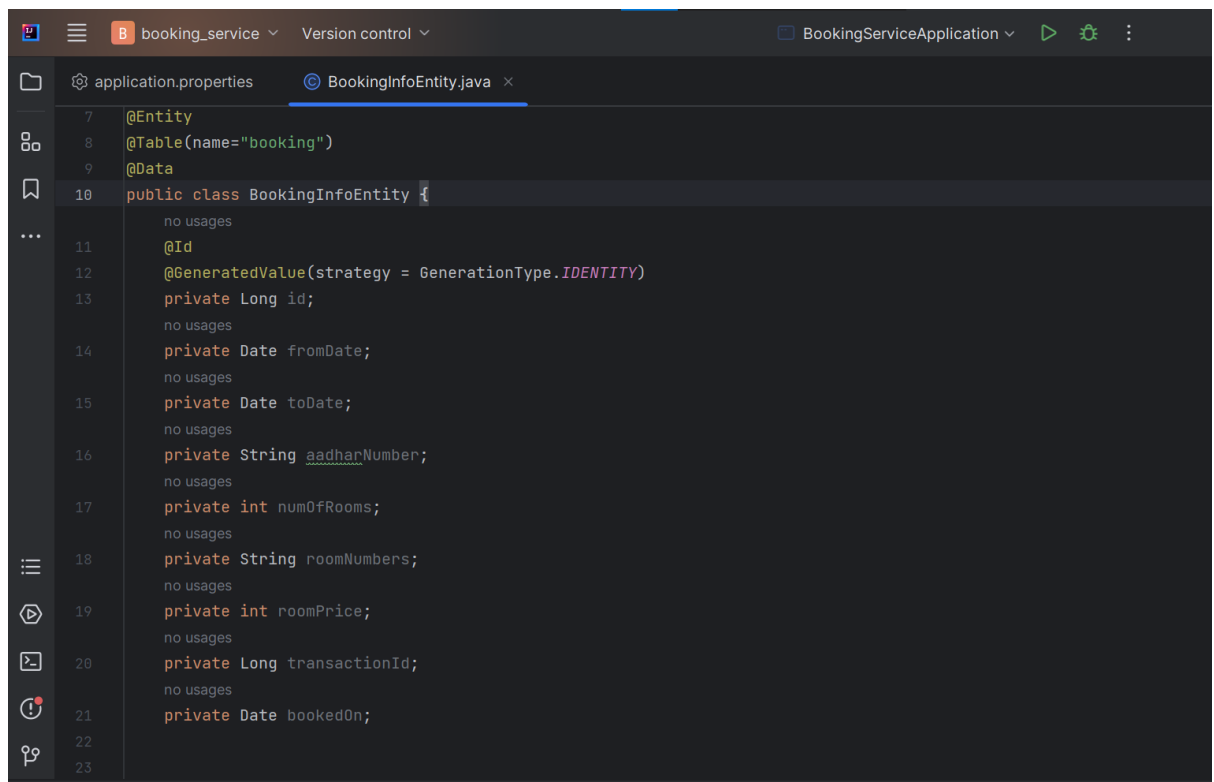
This service is responsible for taking input from users like- toDate, fromDate, aadharNumber and the number of rooms required (numOfRooms) and save it in its database. This service also generates a random list of room numbers depending on 'numOfRooms' requested by the user and returns the room number list (roomNumbers) and total roomPrice to the user.

The logic to calculate room price is as follows:

$\text{roomPrice} = 1000 * \text{numOfRooms} * (\text{number of days})$

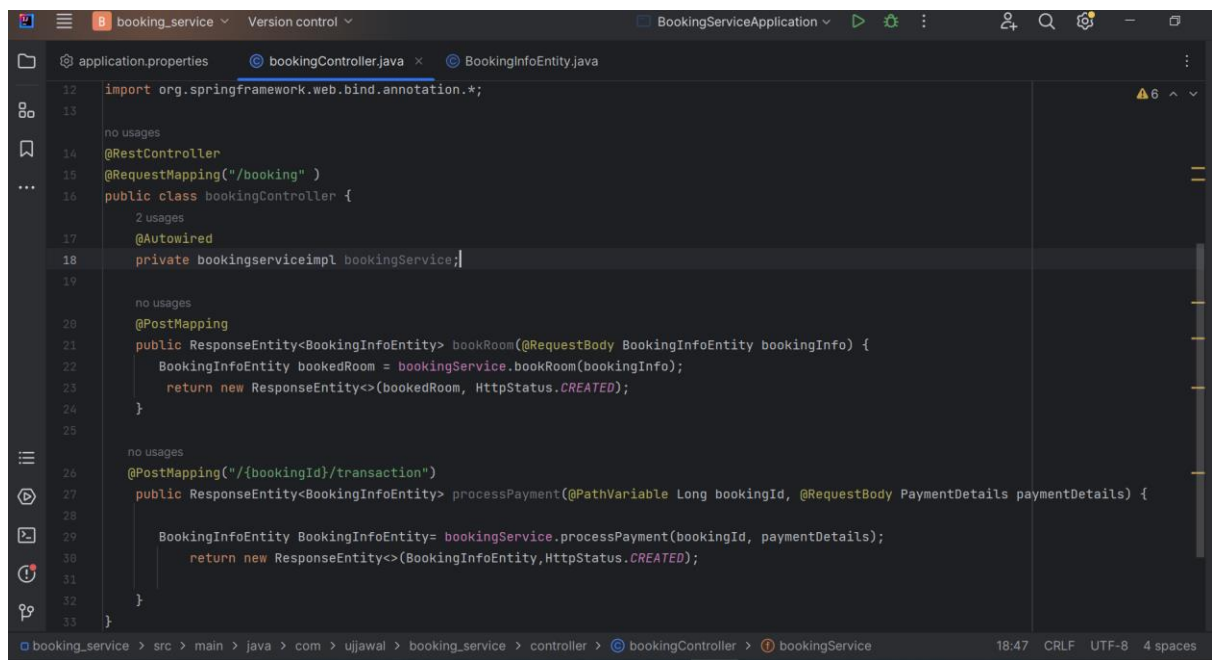
Here, 1000 INR is the base price/day/room. If the user wishes to go ahead with the booking, they can provide the payment related details like bookingMode, upild / cardNumber, which will be further sent to the payment service to retrieve the transactionId. This transactionId then gets updated in the Booking table created in the database of the Booking Service and a confirmation message is printed on the console.

1.1 Model Classes: Refer to the “booking” table in the schema to create the entity class named “BookingInfoEntity”.



```
7  @Entity
8  @Table(name="booking")
9  @Data
10 public class BookingInfoEntity {
11     no usages
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15     no usages
16     private Date fromDate;
17     no usages
18     private Date toDate;
19     no usages
20     private String aadharNumber;
21     no usages
22     private int numOfRooms;
23     no usages
24     private String roomNumbers;
25     no usages
26     private int roomPrice;
27     no usages
28     private Long transactionId;
29     no usages
30     private Date bookedOn;
```

1.2 Controller Layer:

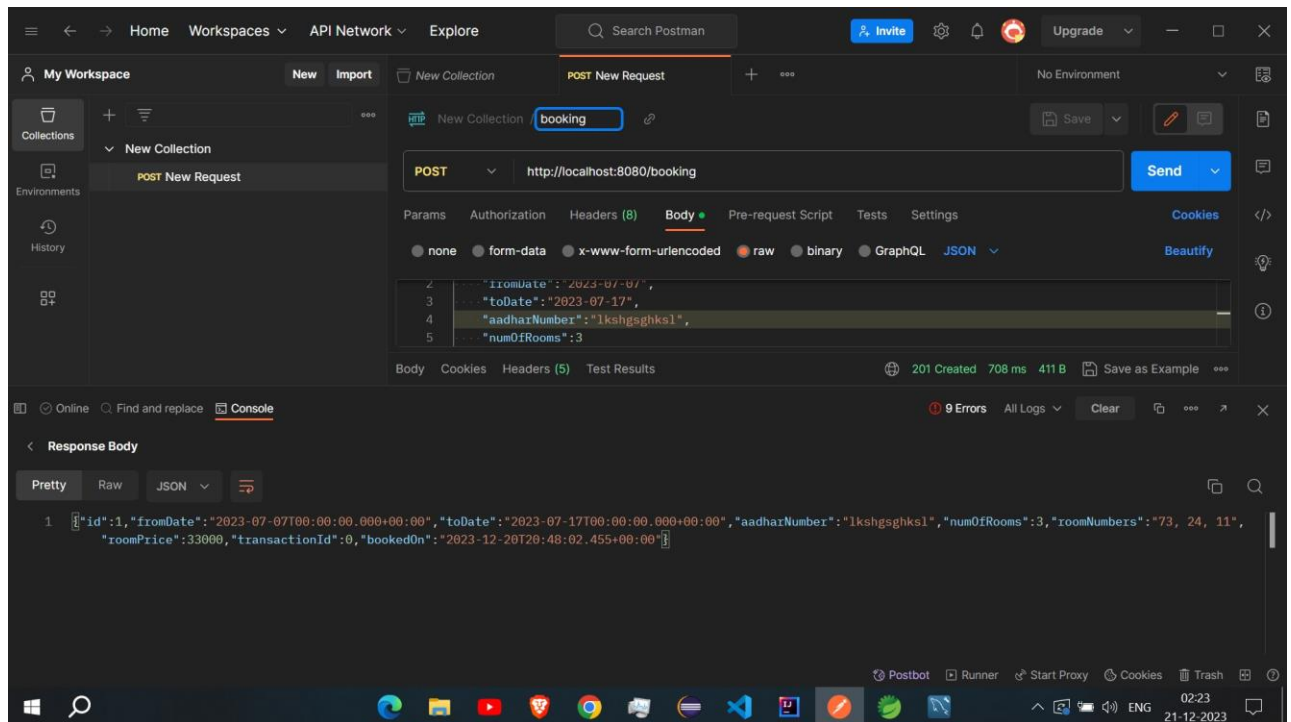


```
12 import org.springframework.web.bind.annotation.*;
13
14 @RestController
15 @RequestMapping("/booking")
16 public class bookingController {
17     @Autowired
18     private bookingServiceimpl bookingService;
19
20     @PostMapping
21     public ResponseEntity<BookingInfoEntity> bookRoom(@RequestBody BookingInfoEntity bookingInfo) {
22         BookingInfoEntity bookedRoom = bookingService.bookRoom(bookingInfo);
23         return new ResponseEntity<>(bookedRoom, HttpStatus.CREATED);
24     }
25
26     @PostMapping("/{bookingId}/transaction")
27     public ResponseEntity<BookingInfoEntity> processPayment(@PathVariable Long bookingId, @RequestBody PaymentDetails paymentDetails) {
28
29         BookingInfoEntity BookingInfoEntity= bookingService.processPayment(bookingId, paymentDetails);
30         return new ResponseEntity<>(BookingInfoEntity,HttpStatus.CREATED);
31     }
32 }
33 }
```

Endpoint 1:

This endpoint is responsible for collecting information like fromDate , toDate ,
aadharNumber , numOfRooms from the user and save it in its database.

- URI: /booking
- HTTP METHOD: POST
- RequestBody: fromDate, toDate,aadharNumber,numOfRooms
- Response Status: Created
- Response: ResponseEntity<BookingInfoEntity>



Endpoint 2:

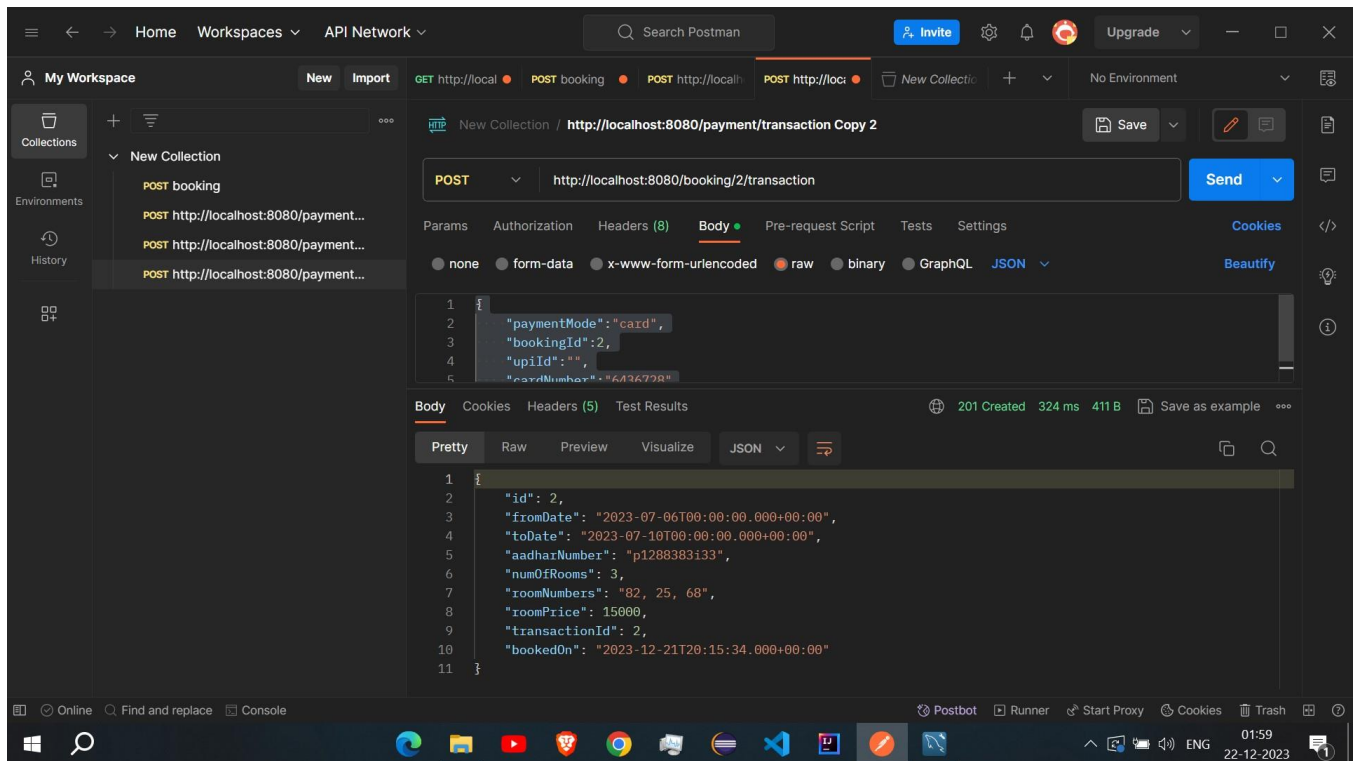
This endpoint is responsible for taking the payment related details from the user and sending it to the payment service. It gets the transactionId from the Payment service in response and saves it in the booking table. Please note that for the field 'paymentMode', if the user provides any input other than 'UPI' or 'CARD', then it means that the user is not interested in the booking and wants to opt-out.

URL: booking/{bookingId}/transaction

HTTP METHOD: POST

PathVariable: int

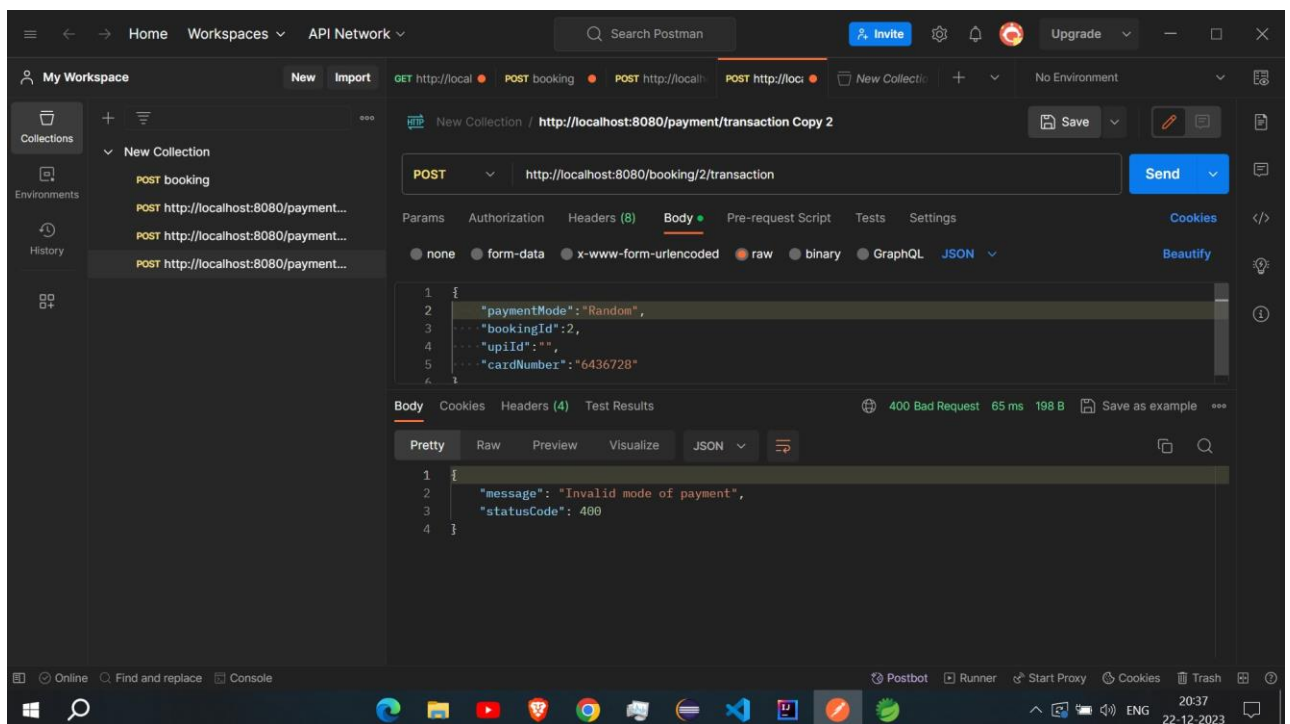
RequestBody: paymentMode, bookingId, upid, cardNumber



Exception 1:

If the user gives any other input apart from “UPI” or “CARD”, the response message should look like the following:

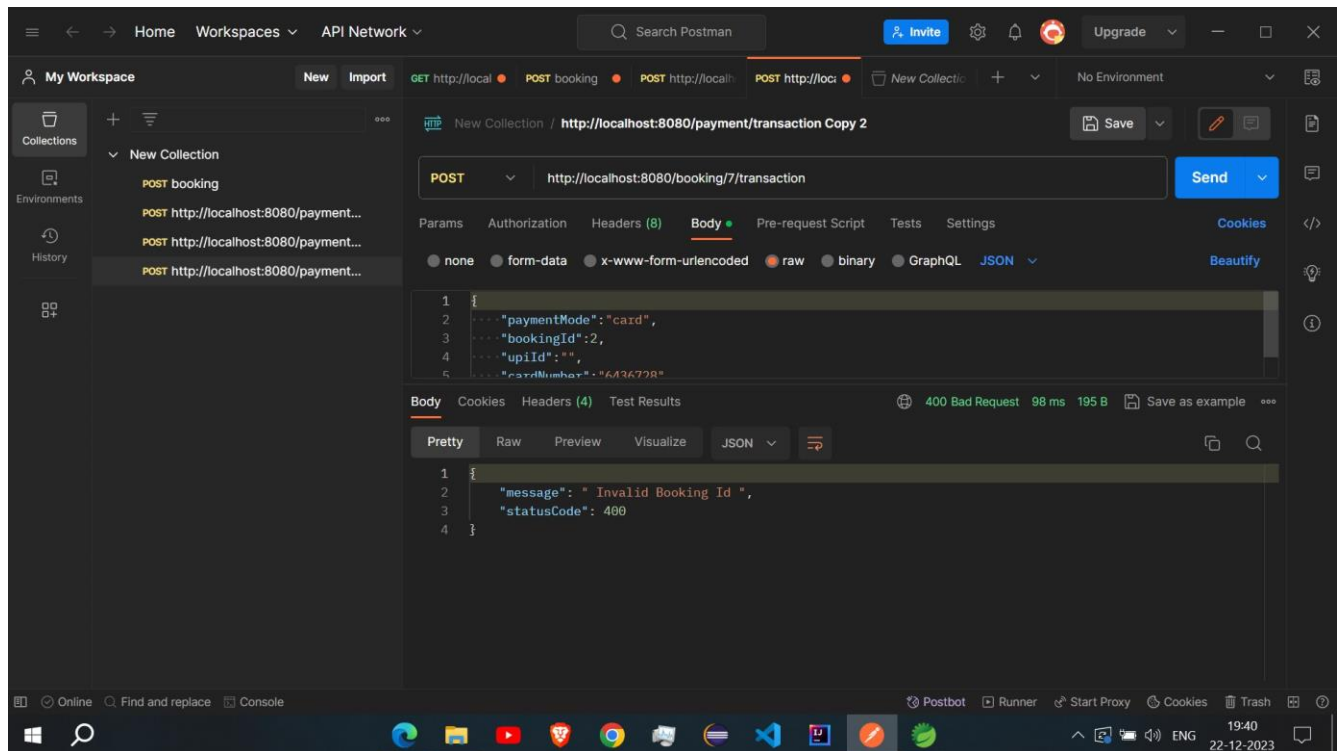
```
1. { 2.   "message": "Invalid mode of payment", 3.   "statusCode": 400 4. } 5.
```



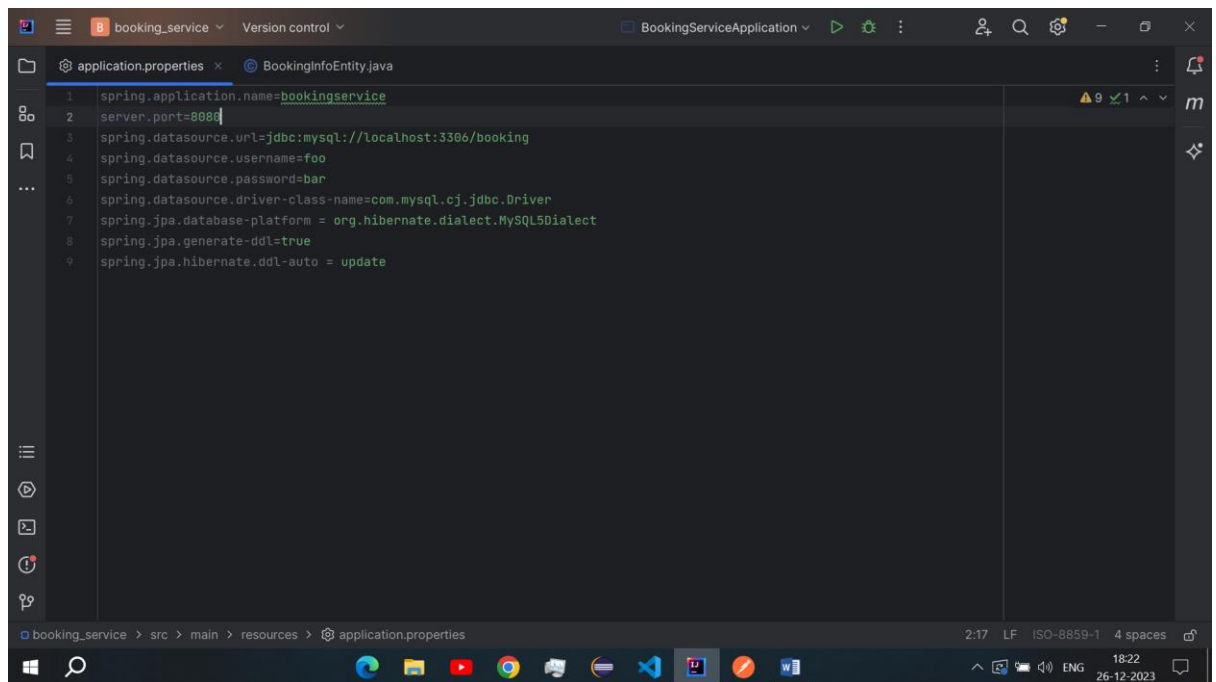
Exception 2:

If no transactions exist for the Booking Id passed to this endpoint then the response message should look like the following:

```
1. {  
2.   "message": " Invalid Booking Id ",  
3.   "statusCode": 400  
4. }  
5.
```



Application.properties



Bokingserviceimpl

```
@Service
public class bookingserviceimpl implements bookingservice{

    @Autowired
    private bookingrepository bookingrepositor;

    @Autowired
    paymentinterface paymentinterface;

    @Override
    public BookingInfoEntity bookRoom(BookingInfoEntity bookingInfo) {

        bookingInfo.setRoomNumbers (generateRandomroom(bookingInfo.getNumOfRooms ()) );

        bookingInfo.setRoomPrice (1000*(bookingInfo.getNumOfRooms ()) *calculatedays (bookingInfo.getFromDate (), booking
        Info.getToDate ()) );

        bookingInfo.setTransactionId ((long) 0);
        Date today= new java.util.Date ();
        bookingInfo.setBookedOn ( today);

        return bookingrepositor.save (bookingInfo);
    }

    private int calculatedays (Date fromDate, Date toDate) {
        // TODO Auto-generated method stub
        int daysdiff = 0;
        long diff = toDate.getTime () - fromDate.getTime ();
        long diffDays = diff / (24 * 60 * 60 * 1000) + 1;
        daysdiff = (int) diffDays;
        return daysdiff;
    }

    private String generateRandomroom (int numofRooms) {
        // TODO Auto-generated method stub
        Random random = new Random ();
        StringBuilder roomNumbers = new StringBuilder ();

        for (int i = 0; i < numofRooms; i++) {
```

```
        int roomNumber = random.nextInt(100) + 1;
        roomNumbers.append(roomNumber);

        if (i < numOfRooms - 1) {
            roomNumbers.append(", ");
        }
    }

    return roomNumbers.toString();
}

public BookingInfoEntity processPayment(Long bookingId, PaymentDetails paymentDetails) {
    String card="CARD";
    String upi="UPI";

    if(!card.equals(paymentDetails.getPaymentMode()) || !upi.equals(paymentDetails.getPaymentMode())) {
        throw new InvalidPaymentException("Invalid mode of payment");
    }

    Long paymentId=paymentInterface.performTransaction(paymentDetails).getBody();

    BookingInfoEntity bookingInfoEntity = bookingRepository.findById(bookingId)
        .orElseThrow(() -> new RecordNotFoundException(" Invalid Booking Id "));

    bookingInfoEntity.setTransactionId(paymentId);

    bookingRepository.save(bookingInfoEntity);

    return bookingInfoEntity;

    // TODO Auto-generated method stub
}
}
```


2. Payment Service:

This service is responsible for taking payment-related information- paymentMode, upiId or cardNumber, bookingId and returns a unique transactionId to the booking service. It saves the data in its database and returns the transactionId as a response.

Model Classes: Refer to the “transaction” table in the schema to create the entity class named “TransactionDetailsEntity”.

```
package com.ujjawal.payment_service.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

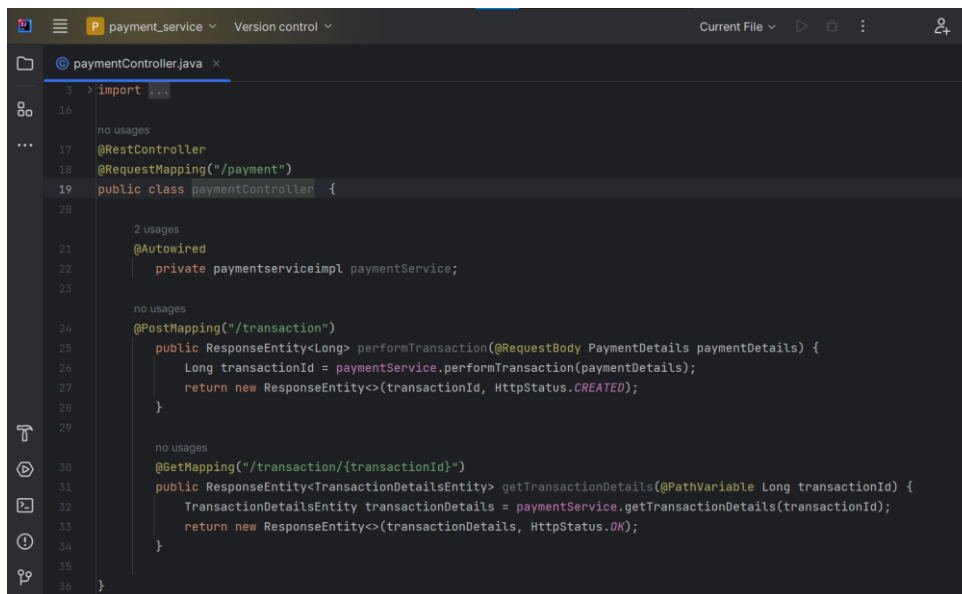
@Entity
@Table(name="payment")
@Data
public class TransactionDetailsEntity {

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

    private String paymentMode;
    private Long bookingId;
    private String upiId;
    private String cardNumber;

}
```

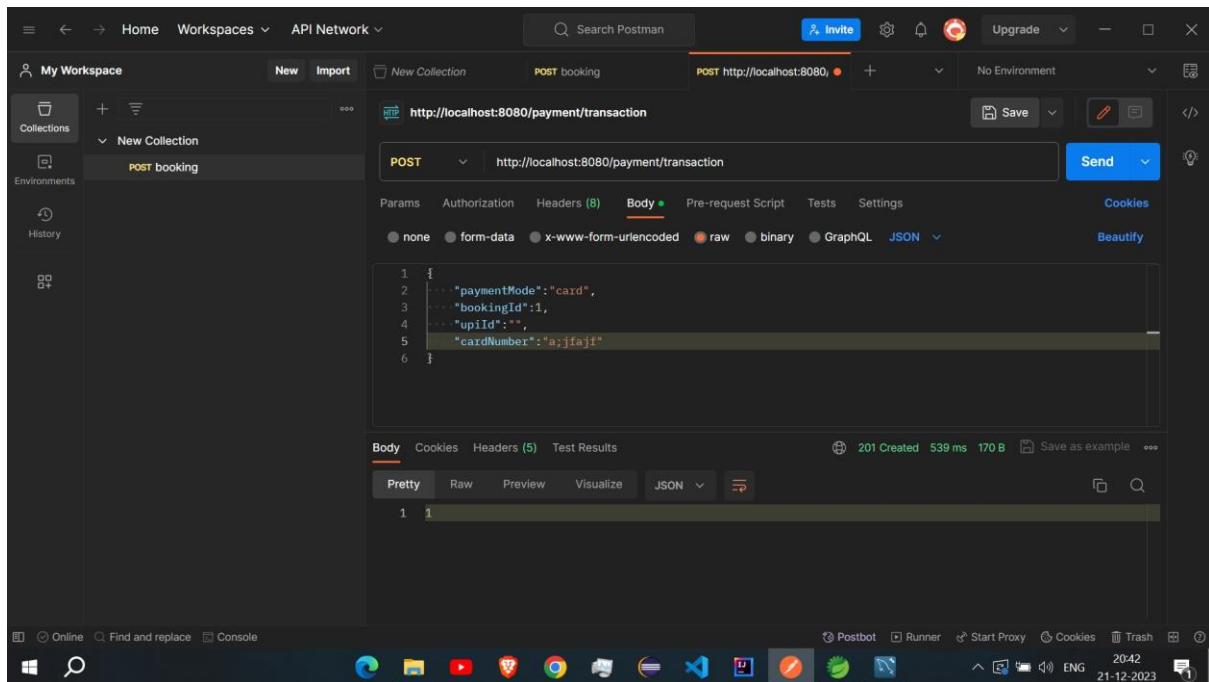
2.1.1 Controller Layer:



Endpoint 1:

This endpoint is used to imitate performing a transaction for a particular booking. It takes details such as bookingId, paymentMode, upild or cardNumber and returns the transactionId automatically generated while storing the details in the 'transaction' table. Note that this 'transactionId' is the primary key of the record that is being stored in the 'transaction' table. After receiving the transactionId from 'Payment' service, confirmation message is printed on the console. Message String: String message = "Booking confirmed for user with aadhaar number: " + bookingInfo.getAadharNumber() + " | " + "Here are the booking details: " + bookingInfo.toString(); Note: This endpoint will be called by the 'endpoint 2' of the Booking service.

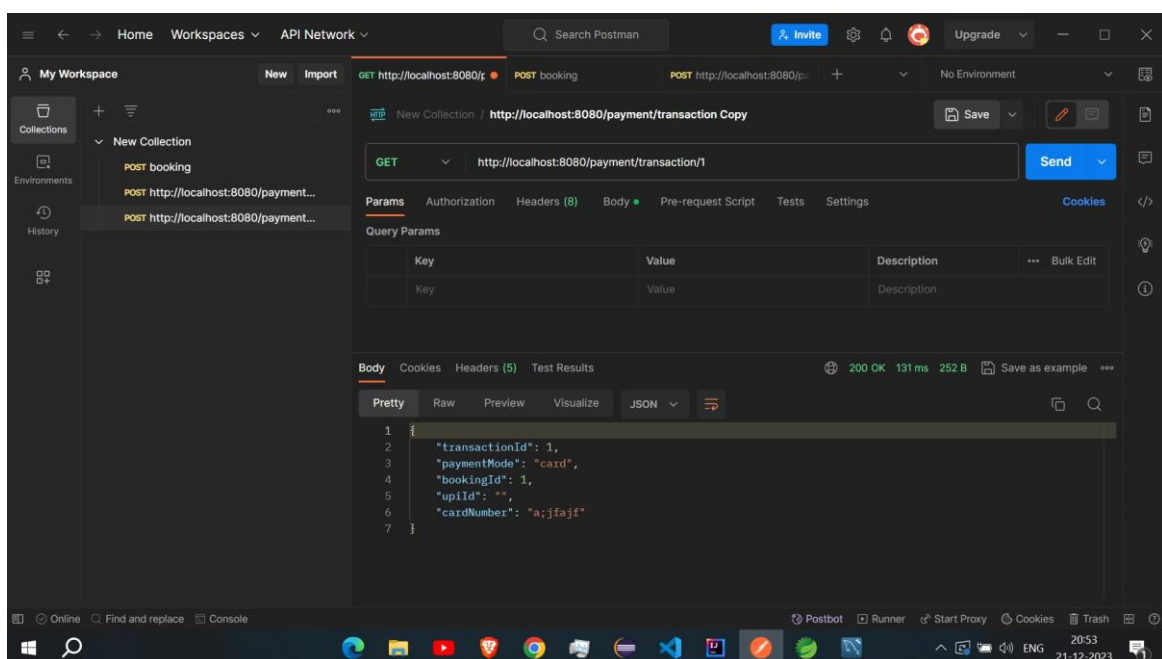
- URL: /transaction
- HTTP METHOD: POST
- RequestBody: paymentMode, bookingId, upild, cardNumber
- Response Status: Created
- Response: ResponseEntity



EndPoint 2:

This endpoint presents the transaction details to the user based on the transactionId provided by the user.

- URL: `/transaction/{transactionId}`
- HTTP METHOD: GET
- RequestBody: (PathVariable) int
- Response Status: OK
- Response: ResponseEntity



paymentserviceimpl

```
@Service
public class paymentserviceimpl implements paymentservice {

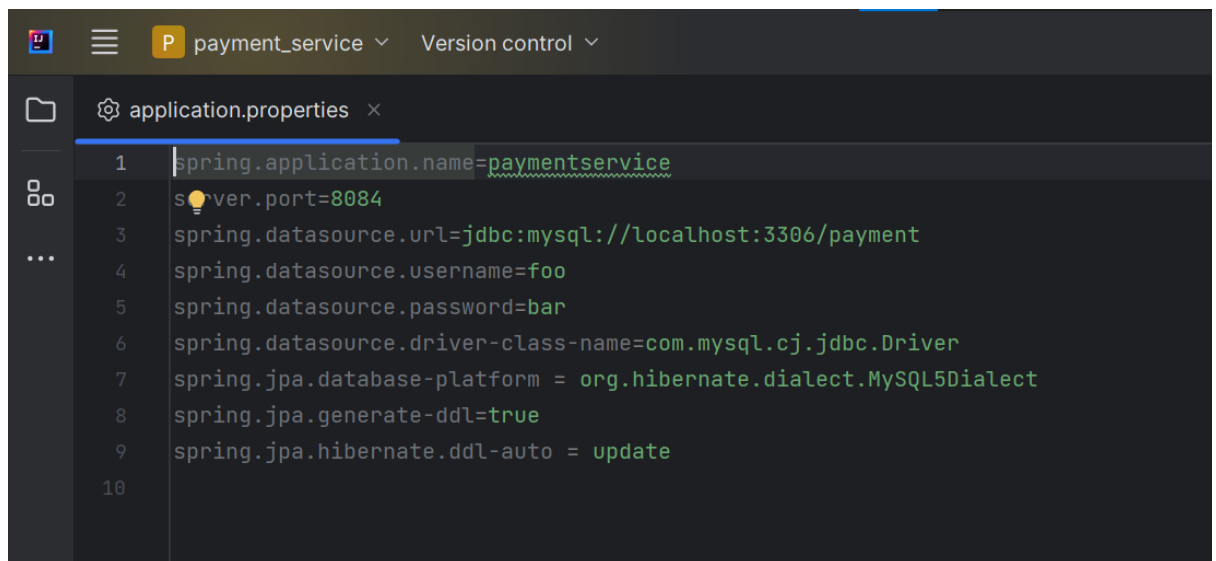
    @Autowired
    private paymentrepository paymentrepository;

    public Long performTransaction(PaymentDetails paymentDetails) {
        // TODO Auto-generated method stub
        TransactionDetailsEntity transactionDetails = new
TransactionDetailsEntity() ;
        transactionDetails.setBookingId(paymentDetails.getBookingId());
        transactionDetails.setCardNumber(paymentDetails.getCardNumber());
        transactionDetails.setPaymentMode(paymentDetails.getPaymentMode());
        transactionDetails.setUpiId(paymentDetails.getUpiId());
        TransactionDetailsEntity
EntitySaved=paymentrepository.save(transactionDetails);
        Long id=EntitySaved.getTransactionId();
        return id;
    }

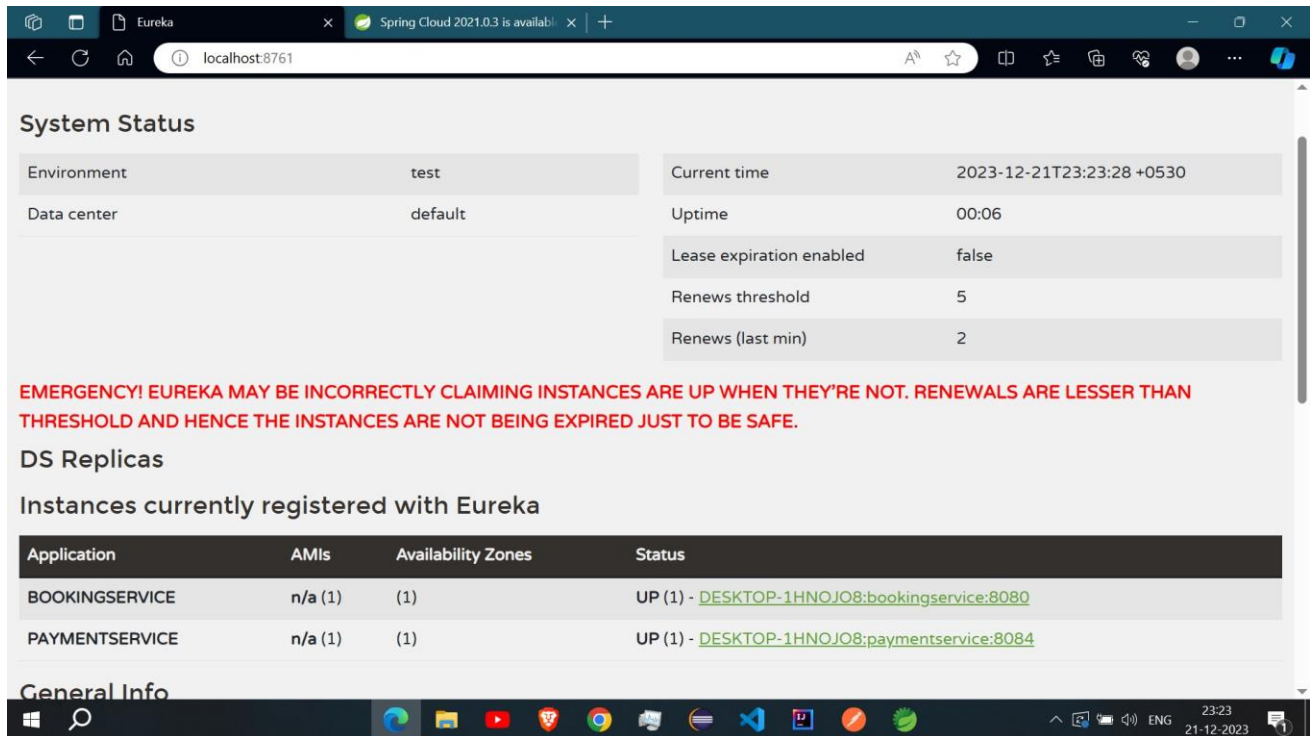
    public TransactionDetailsEntity getTransactionDetails(Long
transactionId) {
        // TODO Auto-generated method stub
        Optional<TransactionDetailsEntity> detailsEntityOptional =
paymentrepository.findById(transactionId);
        return detailsEntityOptional.get();
    }

}
```

Application.properties



Eureka Server



The screenshot shows the Eureka Server web interface in a browser window. The address bar shows 'localhost:8761'. The page title is 'Eureka'. The browser tabs include 'Eureka' and 'Spring Cloud 2021.0.3 is availabl...'. The interface displays the following information:

System Status

Environment	test	Current time	2023-12-21T23:23:28 +0530
Data center	default	Uptime	00:06
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	2

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BOOKINGSERVICE	n/a (1)	(1)	UP (1) - DESKTOP-1HNOJO8:bookingservice:8080
PAYMENTSERVICE	n/a (1)	(1)	UP (1) - DESKTOP-1HNOJO8:paymentservice:8084

General Info

The Windows taskbar at the bottom shows the Start button, search icon, and several application icons including Edge, File Explorer, YouTube, Steam, Chrome, and others. The system tray on the right shows the date and time as 23:23 on 21-12-2023.