Code Logic of Sweet Home - Hotel Room Booking Application

This project facilitates user to book a room in a hotel.

It comprises of four micro services that are as follows: -

1. **API-gateway**

This service acts as a gateway for all user requests. Instead of exposing the Booking and Payment services, this gateway interacts with the users and re-routes the requests to the relevant service internally.

2. **Eureka Server**

Eureka server provides service registration and discovery capabilities for microservices-based architectures. It is a service registry that allows microservices to register themselves and discover other services within the system.

3. **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.

If the user wishes to go ahead with the booking, they can provide the payment related details like bookingMode, upiId / 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.

4. **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.

<u>Following are the code snippets of the microservices in this project: -</u>

1. BookingInfoEntity class:-

```
@Entity
@Table(name = "booking")
public class BookingInfoEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int bookingId;
    private Date fromDate;
    private Date toDate;
    private String aadharNumber;
    private int numOfRooms;
    private String roomNumbers;
    @Column(nullable = false)
    private int roomPrice;
    @Column(columnDefinition = "integer default 0")
```

```
    private int transactionId;

    @Column(updatable = false)
    @CreationTimestamp
    private Date bookedOn;
//.. Getters and Setters
//.. ToString
}
```

2. BookingDTO class:-

```
public class BookingDTO {

    private int id;
    private Date fromDate;
    private Date toDate;
    private String aadharNumber;
    private int numOfRooms;
    private String roomNumbers;
    private int roomPrice;
    private int transactionId;
    private Date bookedOn;

//.. Getters and Setters

}
```

3. TransactionDTO class:-

```
public class TransactionDTO {

    private int transactionId;
    @Pattern(regexp = "UPI|CARD", flags = Pattern.Flag.CASE_INSENSITIVE,
message = "Invalid mode of payment")
    private String paymentMode;

    private int bookingId;

    private String upiId;
    private String cardNumber;
}
```

4. BookingController:-

```
@RestController
@RequestMapping(value = "/hotel")
public class BookingController {
    @Autowired
    BookingService bookingService;

    @Autowired
    ModelMapper modelMapper;

    @Autowired
    RestTemplate restTemplate;

    @RequestMapping(value = "/booking", method = RequestMethod.POST,
consumes = MediaType.APPLICATION_JSON_VALUE)
```

```
    public ResponseEntity<BookingInfoEntity> createBooking(@RequestBody
BookingDTO bookingDTO) {
        BookingInfoEntity bookingInfoEntity = modelMapper.map(bookingDTO,
BookingInfoEntity.class);
        return
ResponseEntity.status(201).body(bookingService.createBooking(bookingInfoEnt
ity));
    }

    @RequestMapping(value = "/booking/{bookingId}/transaction", method =
RequestMethod.POST)
    public ResponseEntity<Object> createBookingWithTransaction(@Valid
@RequestBody TransactionDTO transactionDTO,

@PathVariable(name = "bookingId") int bookingId) {


        if(!bookingService.isBookingExists(bookingId)){
            Map<String, String> errors = new HashMap<>();
            String fieldName = "message";
            String errorMessage = " Invalid Booking Id ";
            errors.put(fieldName, errorMessage);
            errors.put("statusCode", "400");

            return ResponseEntity.status(400).body(errors);
        }

        HttpEntity<TransactionDTO> request = new
HttpEntity<>(transactionDTO);
        ResponseEntity<Integer> response = restTemplate
                .exchange("http://localhost:8083/payment/transaction",
HttpMethod.POST, request, Integer.class);

        int transactionId = response.getBody().intValue();

        Optional<BookingInfoEntity> bookingInfoEntity =
bookingService.updateTransaction(transactionId, bookingId);

        String message = "Booking confirmed for user with aadhaar number: "
                + bookingInfoEntity.get().getAadharNumber()
                +    "    |    "
                + "Here are the booking details:    " +
bookingInfoEntity.get().toString();

        System.out.println(message);

        return ResponseEntity.status(201).body(bookingInfoEntity.get());
    }

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Map<String, String> handleValidationExceptions(
            MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach((error) -> {
            String fieldName = "message";
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
            errors.put("statusCode","400");
        });
        return errors;
```

```
    }
}
```

5. BookingRepository class:-

```
@Repository
public interface BookingRepository extends JpaRepository<BookingInfoEntity,
Integer> {

    @Modifying
    @Query("update BookingInfoEntity b set b.transactionId = :transactionId
where b.bookingId = :bookingId")
    int updateTransaction(@Param(value = "transactionId") int
transactionId,
                          @Param(value = "bookingId") int bookingId);
}
```

6. TransactionDetailsEntity class:-

```
@Entity
@Table(name = "transaction")
public class TransactionDetailsEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int transactionId;
    @Column(nullable = false)
    private String paymentMode;
    @Column(nullable = false)
    private int bookingId;
    @Column()
    private String upiId;
    @Column()
    private String cardNumber;
```

7. TransactionServiceController:-

```
@RestController
@RequestMapping(value = "/payment")
public class TransactionServiceController {

    @Autowired
    PaymentService paymentService;
    @Autowired
    ModelMapper modelMapper;


    @RequestMapping(value = "/transaction", method = RequestMethod.POST,
consumes = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Integer> createTransaction(@RequestBody
TransactionDTO transactionDTO) {

        boolean isValidRequest;

if(transactionDTO.getPaymentMode().equalsIgnoreCase(PaymentMode.UPI)){
            isValidRequest = !(Objects.isNull(transactionDTO.getUpiId()) ||
transactionDTO.getUpiId().isEmpty());
            transactionDTO.setCardNumber(null);
```

```
        } else
if(transactionDTO.getPaymentMode().equalsIgnoreCase(PaymentMode.CARD)){
            isValidRequest =
!(Objects.isNull(transactionDTO.getCardNumber()) ||
transactionDTO.getCardNumber().isEmpty());
            transactionDTO.setUpiId(null);
        } else{
            isValidRequest = false;
        }
        if(isValidRequest){
            TransactionDetailsEntity transactionDetailsEntity =
modelMapper.map(transactionDTO, TransactionDetailsEntity.class);
            return new
ResponseEntity<>(paymentService.createTransaction(transactionDetailsEntity)
, HttpStatus.CREATED);
        } else{
            return new ResponseEntity<>(0, HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value = "/transaction/{transactionId}", method =
RequestMethod.GET)
    public ResponseEntity<TransactionDetailsEntity>
getTransaction(@PathVariable(name = "transactionId") int transactionId) {
        TransactionDetailsEntity transactionDetailsEntity =
paymentService.getTransaction(transactionId);
        return Objects.isNull(transactionDetailsEntity) ? new
ResponseEntity<>(null, HttpStatus.NOT_FOUND)
                : new ResponseEntity<>(transactionDetailsEntity,
HttpStatus.OK);
    }
}
```

Steps to Run the Solution: -

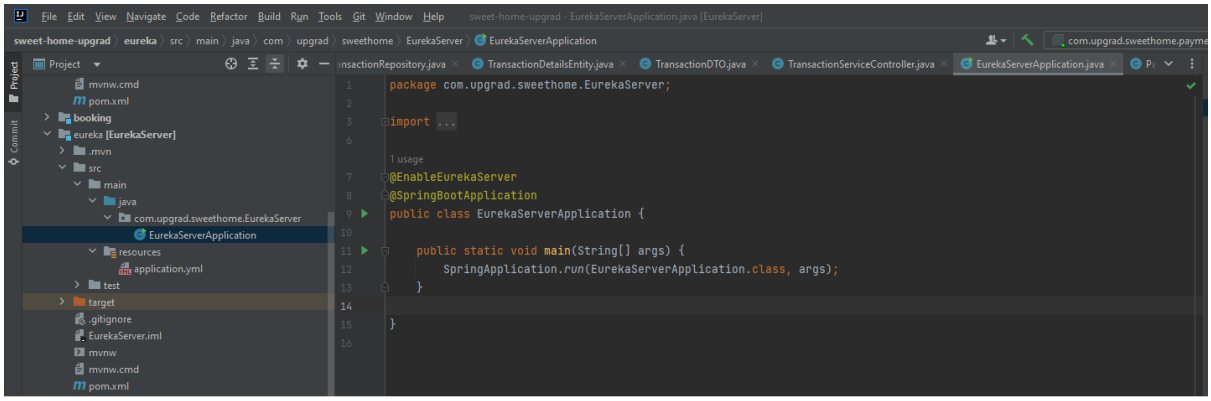
1. Install IntelliJ IDEA

2. Navigate to the folder where all four microservices folders are saved.

3. Right-click on the folder and select "Open Folder as Intellij IDEA".

4. Navigate to eureka [EurekaServer] > src > main > java > com.upgrad.sweethome.EurekaServer > EurekaServerApplication.java.

5. Click on the "Run" button located on the left side of the public class EurekaServerApplication statement.

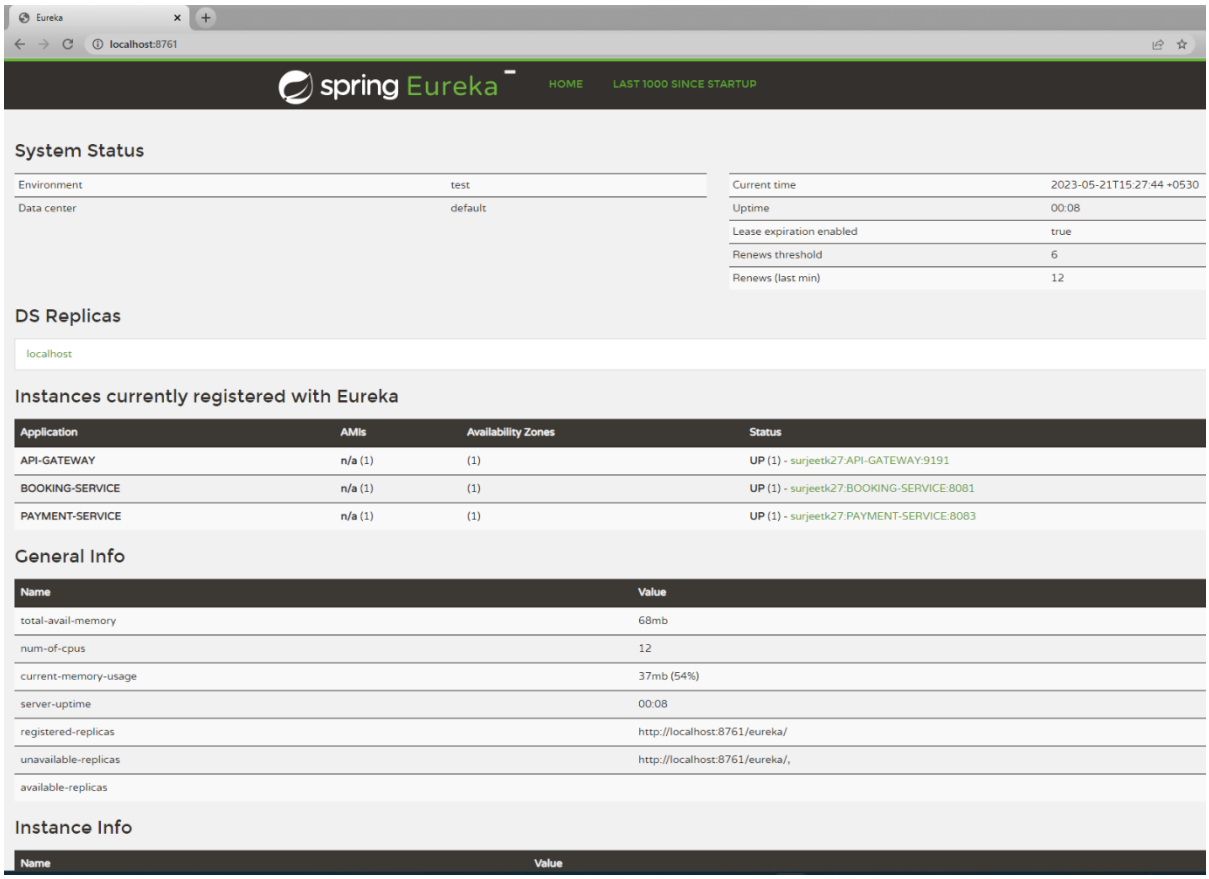


- If you cannot see the "Run" button, right-click on the main folder that contains all the microservices folders and select"Mark Directory as" and then "Source Root".

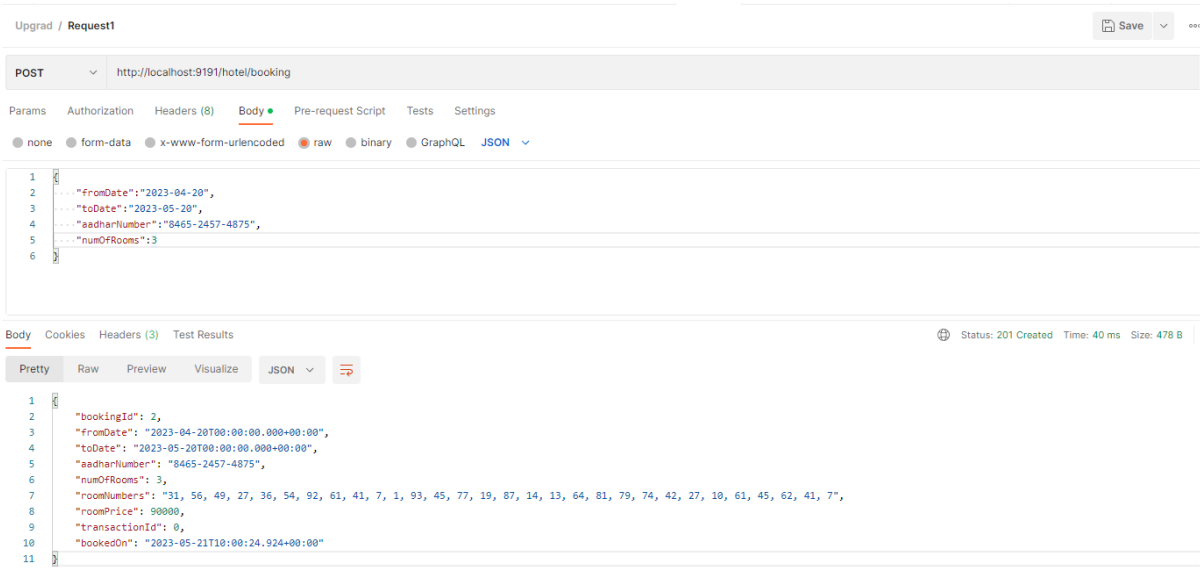
6. Once the Eureka server is live, navigate to each microservice's folder > src > java > com.upgrad.sweethome.ServiceName > ServiceNameApplication. Click on the "Run" button, located on the left side of the "public class ServiceNameApplication" statement, as you did for the Eureka server.

Screenshots of the Implemented solution: -

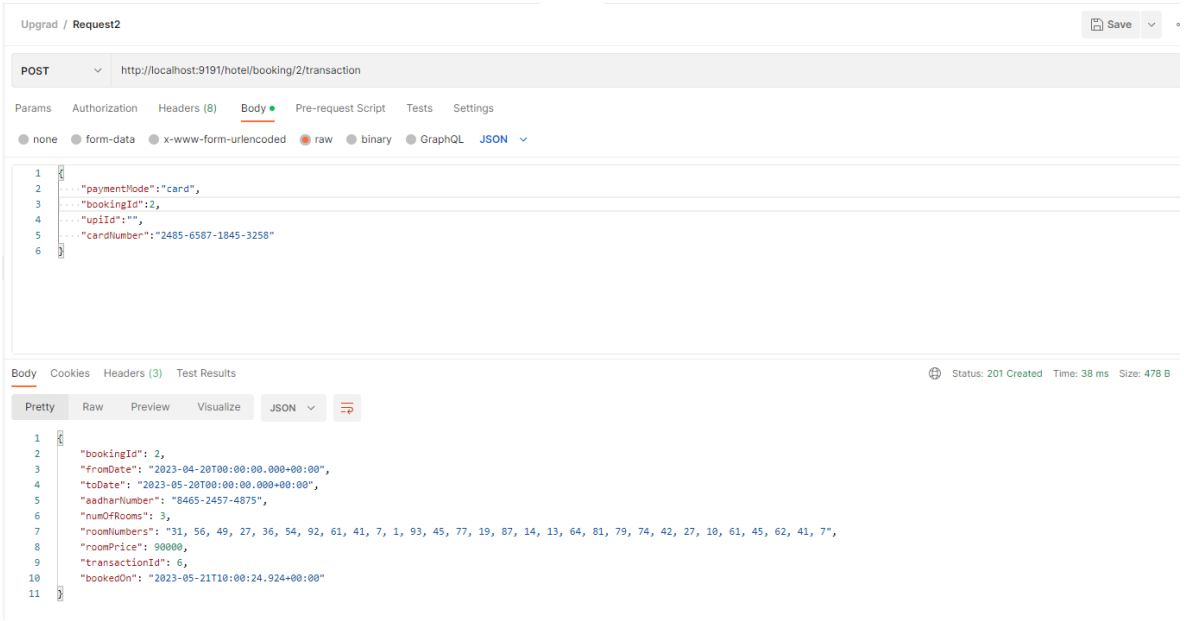
1. **Eureka Server - http://localhost:8761/**



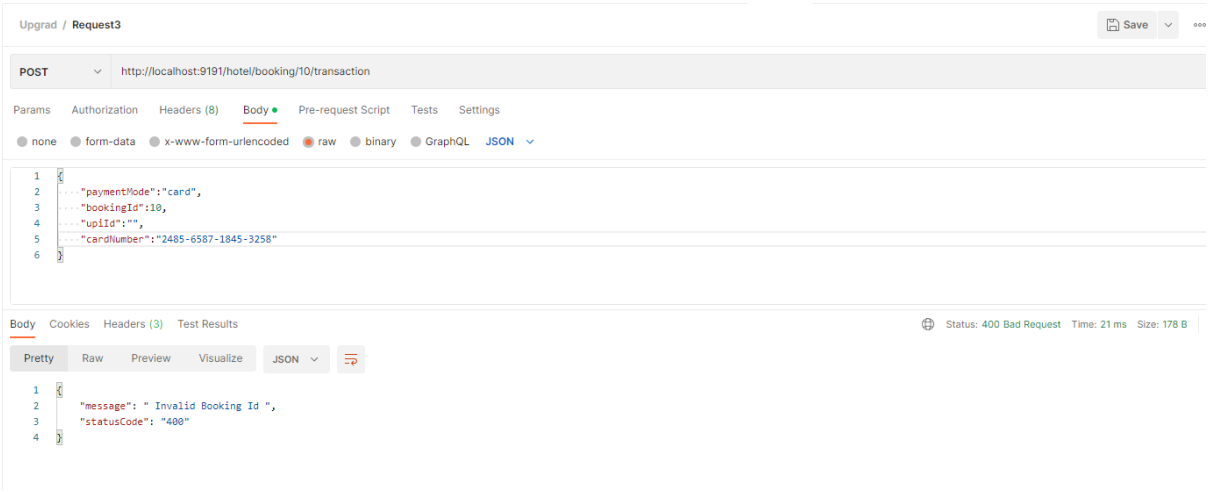
2. Request 1 = POST localhost:9191/hotel/booking



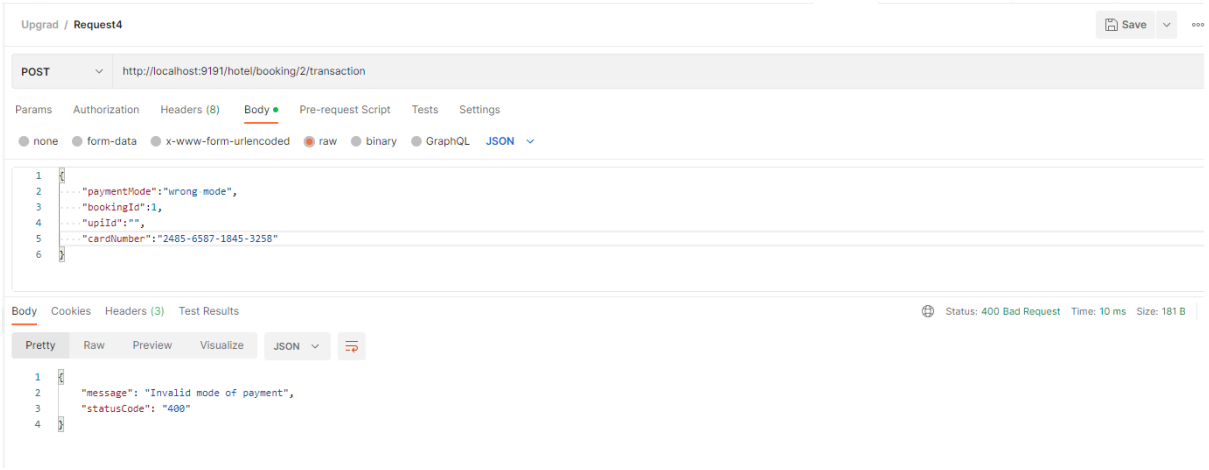
3. Request 2 = POST localhost:9191/hotel/booking/2/transaction



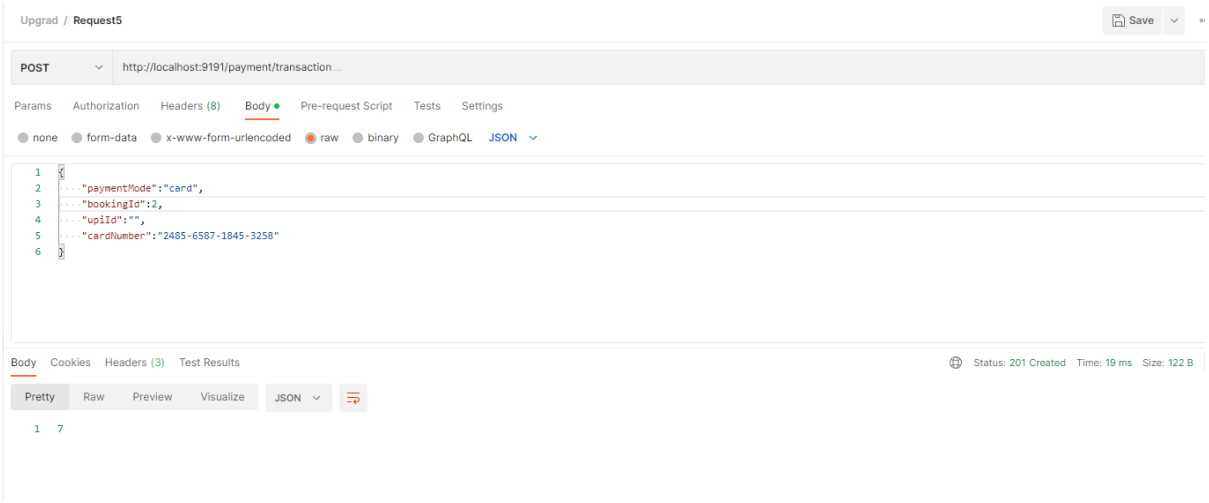
4. **Request 3 (Invalid booking ID) = POST localhost:9191/hotel/booking/10/transaction**



5. **Request 4 (Invalid payment mode) = POST localhost:9191/hotel/booking/2/transaction**



6. **Request 5 = POST localhost:9191/payment/transaction**



7. Request 6 = GET localhost:9191/payment/transaction/7