

# 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, 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.

## 4. Payment Service

This service is responsible for taking payment-related information- paymentMode, upild 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.

### Following are the code snippets of the microservices in this project: -

#### 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().getAadhaarNumber()
                + " | "
                + "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);  
        }  
    }  
}
```

```

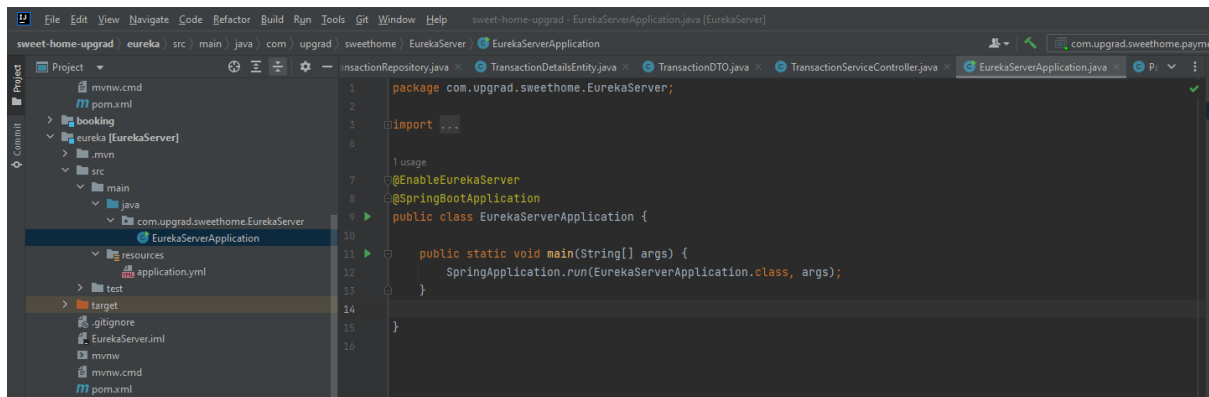
        } else
    if (transactionDTO.getPaymentMode().equalsIgnoreCase(PaymentMode.CARD)) {
        isValidRequest =
        !(Objects.isNull(transactionDTO.getCardNumber()) ||
        transactionDTO.getCardNumber().isEmpty());
        transactionDTO.setUpiId(null);
    } else{
        isValidRequest = false;
    }
    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/>

**System Status**

Environment	test	Current time	2023-05-21T15:27:44 +0530
Data center	default	Uptime	00:08
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	12

**DS Replicas**

localhost

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - surjeetk27-API-GATEWAY-9191
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - surjeetk27-BOOKING-SERVICE-8081
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - surjeetk27-PAYMENT-SERVICE-8083

**General Info**

Name	Value
total-avail-memory	68mb
num-of-cpus	12
current-memory-usage	37mb (54%)
server-uptime	00:08
registered-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
unavailable-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
available-replicas	

**Instance Info**

Name	Value
------	-------

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

Upgrad / Request1 Save

POST http://localhost:9191/hotel/booking

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON

```
1 {
2   ... "fromDate": "2023-04-20",
3   ... "toDate": "2023-05-20",
4   ... "aadharNumber": "8465-2457-4875",
5   ... "numOfRooms": 3
6 }
```

Body Cookies Headers (3) Test Results Status: 201 Created Time: 40 ms Size: 478 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "bookingId": 2,
3   "fromDate": "2023-04-20T00:00:00.000+00:00",
4   "toDate": "2023-05-20T00:00:00.000+00:00",
5   "aadharNumber": "8465-2457-4875",
6   "numOfRooms": 3,
7   "roomNumbers": "31, 56, 49, 27, 36, 54, 92, 61, 41, 7, 1, 93, 45, 77, 19, 87, 14, 13, 64, 81, 79, 74, 42, 27, 10, 61, 45, 62, 41, 7",
8   "roomPrice": 90000,
9   "transactionId": 0,
10  "bookedOn": "2023-05-21T10:00:24.924+00:00"
11 }
```

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

Upgrad / Request2 Save

POST http://localhost:9191/hotel/booking/2/transaction

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL ☒ JSON

```
1 {
2   ... "paymentMode": "card",
3   ... "bookingId": 2,
4   ... "upiId": "",
5   ... "cardNumber": "2405-6587-1845-3258"
6 }
```

Body Cookies Headers (3) Test Results Status: 201 Created Time: 38 ms Size: 478 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "bookingId": 2,
3   "fromDate": "2023-04-20T00:00:00.000+00:00",
4   "toDate": "2023-05-20T00:00:00.000+00:00",
5   "aadharNumber": "8465-2457-4875",
6   "numOfRooms": 3,
7   "roomNumbers": "31, 56, 49, 27, 36, 54, 92, 61, 41, 7, 1, 93, 45, 77, 19, 87, 14, 13, 64, 81, 79, 74, 42, 27, 10, 61, 45, 62, 41, 7",
8   "roomPrice": 90000,
9   "transactionId": 6,
10  "bookedOn": "2023-05-21T10:00:24.924+00:00"
11 }
```

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

Upgrade / Request3 Save

**POST** http://localhost:9191/hotel/booking/10/transaction

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   ... "paymentMode": "card",
3   ... "bookingId": 10,
4   ... "upId": "",
5   ... "cardNumber": "2485-6587-1845-3258"
6 }
```

Body Cookies Headers (3) Test Results Status: 400 Bad Request Time: 21 ms Size: 178 B

**Pretty** Raw Preview Visualize **JSON**

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

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

Upgrade / Request4 Save

**POST** http://localhost:9191/hotel/booking/2/transaction

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   ... "paymentMode": "wrong mode",
3   ... "bookingId": 1,
4   ... "upId": "",
5   ... "cardNumber": "2485-6587-1845-3258"
6 }
```

Body Cookies Headers (3) Test Results Status: 400 Bad Request Time: 10 ms Size: 181 B

**Pretty** Raw Preview Visualize **JSON**

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

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

Upgrade / Request5 Save

**POST** http://localhost:9191/payment/transaction ...

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   ... "paymentMode": "card",
3   ... "bookingId": 2,
4   ... "upId": "",
5   ... "cardNumber": "2485-6587-1845-3258"
6 }
```

Body Cookies Headers (3) Test Results Status: 201 Created Time: 19 ms Size: 122 B

**Pretty** Raw Preview Visualize **JSON**

```
1 7
```



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

Upgrad / Request6 Save

GET ▼ http://localhost:9191/payment/transaction/7

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

1

**Body** Cookies Headers (3) Test Results Status: 200 OK Time: 19 ms Size: 218 B

Pretty Raw Preview Visualize JSON ▼ ≡

```
1  {
2    "transactionId": 7,
3    "paymentMode": "card",
4    "bookingId": 2,
5    "upId": null,
6    "cardNumber": "2485-6567-1845-3258"
7  }
```