HOTEL ROOM BOOKING APPLICATION (CodingLogic)

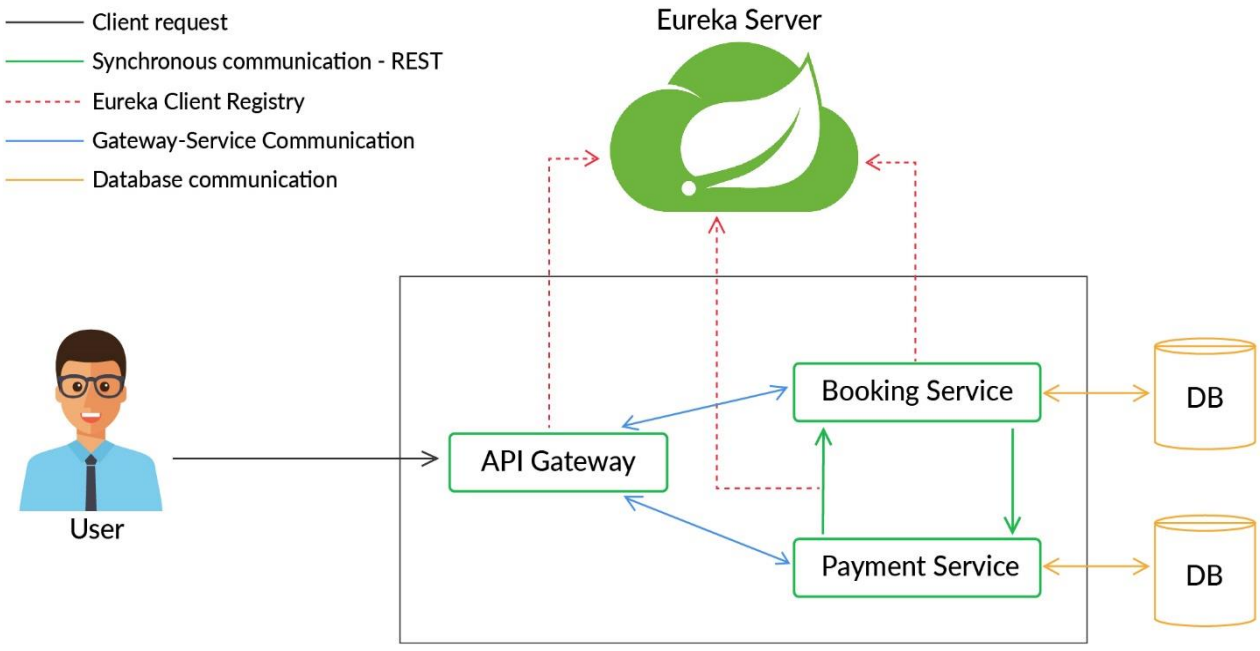
INTRODUCTION

I have created the three different microservice in this "Hotel room booking application", which are:

- **API-GATEWAY**: This service is exposed to the outer world and is responsible for routing all requests to the microservices internally.
- **BOOKING SERVICE**: This service is responsible for collecting all information related to user booking and sending a confirmation message once the booking is confirmed.
- **PAYMENT SERVICE:** This is a dummy payment service; this service is called by the booking service for initiating payment after confirming rooms.

APPLICATION WORKFLOW

The workflow of the application is this following Architecture diagram.



PROJECT SETUP

1. Create a folder "**Sweet-Home**".
2. Created the project structure using Spring Initializer for **'API Gateway'**, and microservices- **'Booking', 'Payment'**, and **'Eureka server'** inside "**Sweet-Home**".
3. These are the dependencies for each service:

Booking Service:

- Spring Cloud Netflix Eureka Client
- Spring Boot Web
- Spring Boot Data JPA
- Mysql-connector-java
- Lombok
- Model mapper
- Configured to run port 8081.

Payment Service:

- Spring Cloud Netflix Eureka Client
- Spring Boot Web
- Spring Boot Data JPA
- Mysql-connector-java
- Lombok
- Model mapper
- Configured to run port 8083.

API Gateway:

- Dependencies: Spring Boot Actuator, Spring Cloud Netflix Eureka Client
- Configured to run port 9191

Eureka Server:

- Dependencies: Spring Cloud Netflix Eureka Client
- Configured to run port 8761.

API- GATEWAY MICROSERVICE

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.

Application.yml file:

- In this file I have connfigured this service to run on port number 9191.
- Configured it as Eureka Client.
- Add Booking and Payment service as ids and URI. Also, add relevant paths for both services.
- **This is Booking service configuration:**
  id: BOOKING-SERVICE
  uri: lb://BOOKING-SERVICE
  predicates:
  - Path=/hotel/**
- **This is Payment service configuration:**
  id: PAYMENT-SERVICE
  uri: lb://PAYMENT-SERVICE
  predicates:
  - Path=/payment/**

After that, configuration is done properly for this service, we have run the Eureka server on this localhost.
http://localhost:8761/ (here we get our service name that is 'API Gateway')

BOOKING SERVICE MICROSERVICE

1. **Model Classes(Entity/Dto):**
   For this, taken Reference from the "booking" table in the schema to create the entity class named "BookingInfoEntity".

Service	Table Name	Columns	Datatypes	Constraints	Description
Booking Service	booking	bookingId	int (auto generated)	PRIMARY KEY	It refers to the "BookingId" of the user and is used to uniquely identify a booking.
		fromDate	Date	NULL	It refers to the date from which the user is looking for the room
		toDate	Date	NULL	It refers to the date untill when the user requires room
		aadharNumber	String	NULL	aadhar number of the user. It helps to uniquely identify the user.
		numOfRooms	int		It refers to the number of rooms required by user
		roomNumbers	String		It represents the list of room numbers allocated to the user
		roomPrice	int	NOT NULL	It refers to the total price of the allocated rooms for the requested days. Default value of a single room is Rs.1000, so if a user requests for 2 rooms for 2 days, then roomPrice will be Rs.4000.
		transactionId	int	DEFAULT=0	It refers to the transactionId which we get from the payment service.
		bookedOn	date	NULL	It refers to the current date
Payment Service	transaction	transactionId	int(auto generated)	PRIMARY KEY	It refers to the transaction Id and is used to uniquely identify a transaction.
		paymentMode	String		It refers to the user's mode of payment which can have values either as 'upi' or ' card'.
		booking Id	int	NOT NULL	It refers to the bookingId which we receive from the 'hotel booking service' when the payment service is called
		upi Id	String	NULL	If the user's mode of payment is 'upi', he has to provide the upiId and cardNumber must be null.
		cardNumber	String	NULL	If the user's mode of payment is 'card', he has to provide the cardNumber and upiId must be null.

2. **Booking Service:**
   I have created two class for Booking service.

- **BookingService;**

  This service is act like interface and there are two method getBookingInfo and addPaymentDetails.

  getBookingInfo: This is responsible for getbookingInfo information from users like- toDate, fromDate, aadharNumber and the number of rooms required (numOfRooms) and save it in its database.

  addPaymentDetails: This is responsible for taking the payment related details from the user and it sending to the payment service.

- **BookingServiceImpl:**
  This service is responsible for taking input from users. It 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:

  roomPrice = 1000* numOfRooms*(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, 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.

A sample code that you could refer to create random room numbers is as follows:
The following code snippet returns a random number list with upperbound of 100 and 'count' number of entries in the number list

```
public static ArrayList<String> getRandomNumbers(int count){
        Random rand = new Random();
        int upperBound = 100;
        ArrayList<String>numberList = new ArrayList<String>();

       for (int i=0; i<count; i++){
                numberList.add(String.valueOf(rand.nextInt(upperBound)));
        }
         return numberList;
}
```

3. Controller/BookingController:

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

- **addPaymentDetails:**
  This endpoint is responsible for taking the payment related details from the user and sending it to the payment service.

4. Dao:

This is JPA repository for database operation.

5. Exceptions

6. Handler:

This send proper error response with message and status code in case of exception.

PAYMENT SERVICE MICROSERVICE

1. Payment Service:

- **Transaction service**

  This service is act like interface and there are two method performingTransaction and getTransactionDetails.

- **TransactionServiceImple:**
  This service is responsible for taking payment-related information- paymentMode, upiId or cardNumber, bookingId and returns a unique transactionId to the booking service. After receiving the transactionId receiving from the payment service it will give a confirmation message.

2. **Entites/Dto:**
   For this, taken Reference from the "transaction" table in the schema to create the entity class named "TransactionDetailsEntity".

Service	Table Name	Columns	Datatypes	Constraints	Description
Booking Service	booking	bookingId	int (auto generated)	PRIMARY KEY	It refers to the "BookingId" of the user and is used to uniquely identify a booking.
		fromDate	Date	NULL	It refers to the date from which the user is looking for the room
		toDate	Date	NULL	It refers to the date untill when the user requires room
		aadharNumber	String	NULL	aadhar number of the user. It helps to uniquely identify the user.
		numOfRooms	int		It refers to the number of rooms required by user
		roomNumbers	String		It represents the list of room numbers allocated to the user
		roomPrice	int	NOT NULL	It refers to the total price of the allocated rooms for the requested days. Default value of a single room is Rs.1000, so if a user requests for 2 rooms for 2 days, then roomPrice will be Rs.4000.
		transactionId	int	DEFAULT=0	It refers to the transactionId which we get from the payment service.
		bookedOn	date	NULL	It refers to the current date
Payment Service	transaction	transactionId	int(auto generated)	PRIMARY KEY	It refers to the transaction Id and is used to uniquely identify a transaction.
		paymentMode	String		It refers to the user's mode of payment which can have values either as 'upi' or ' card'.
		booking Id	int	NOT NULL	It refers to the bookingId which we receive from the 'hotel booking service' when the payment service is called
		upi Id	String	NULL	If the user's mode of payment is 'upi', he has to provide the upiId and cardNumber must be null.
		cardNumber	String	NULL	If the user's mode of payment is 'card', he has to provide the cardNumber and upiId must be null.

3. Controller/ transactionController:

- **handleTransaction:** This endpoint take details such as bookingId, paymentMode, upiId or cardNumber and returns the transactionId automatically.

- **getTransactionDetails:** This endpoint provide transaction detail to the user based on the transaction id provided by the user.

4. **Dao/TransactionDetailsDao:**
   This is JPA repository for database operation.

EUREKA SERVER MICROSERVICE

This is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.

- Dependencies: Spring Cloud Netflix Eureka Client
- Annotate the main class with @EnableEurekaServer so that the Eureka server gets enabled.
- Port for Eureka server as 8761.

RUNNING INSTRUCTIONS

- Once the configuration is done properly for this service, run the Eureka server, Booking service, API Gateway and Payment service on your localhost.
- Hit the Eureka server IP (http://localhost:8761/) from the browser and then all application name appear on the eureka server.