Sweet-Home

This is an Overall Architecture for the 'Hotel room booking application' which is used to book rooms for a single hotel. The backend of the application comprises of two independent microservices: Booking Service and Payment Service.

Instructions to run application

1. Please Create database in mysql named "BookingService" and "PaymentService"

2. Please change user to "root" and password to "wajahath" or you can configure your existing user and password in applicaton.properties for both microservices i.e for payment and booking service.

Overall Architecture of Application

Application is divided into three microservices

1. **Booking service**
2. **Payment service**
3. **Eureka client service**

Booking Service

Booking service is divided into following layers

- **Entity**: In this we create booking schema including all details we want to store in database such as id, from date, to date etc
- **Dao layer**: this layer is responsible to performing crud operations. Here we use Booking Dao interface which extends Jpa Repository for performing all operations.
- **Dto layer:** This layer provides skeleton for json objects and we fetch json objects using this Dto class using getters and setters.
- **Exceptions**: this layer is responsible to catch exceptions and throw user friendly messages.
  Controller layer uses methods from service layer for logic implementation.
- **Service layer**: This layer is responsible to implement business logic, We're using BookinServiceImpl class in service layer which is implementation of Booking Service in which we used interface and Implemented methods are as follows.

  1. bookingDetails : We're using this method to implement logic for room booking. This method uses other two method which helps in generating random room numbers (getRoomNumbers method). This method fetches details from user input and performs business logic and saves in database. Here is the following logic we used in this method.

```
//logic for booking hotel room
public Booking bookingDetails(BookingDTO bookingRequest) throws
Exception {
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy/MM/dd");
    Date date = new Date();
    int requestedNumOfRooms = bookingRequest.getNumOfRooms();

    String roomNumbers = String.join(",",
getRoomNumbers(requestedNumOfRooms));
    Booking bookingInfo = new Booking();
    bookingInfo.setRoomNumbers(roomNumbers);
    bookingInfo.setFromDate(bookingRequest.getFromDate());
    bookingInfo.setToDate(bookingRequest.getToDate());
    bookingInfo.setBookedOn(date);
    bookingInfo.setAadharNumber(bookingRequest.getAadharNumber());
    long numberOfDays = (bookingInfo.getToDate().getTime() -
bookingRequest.getFromDate().getTime()) / (1000 * 60 * 60 * 24) %
365;
    System.out.println("Number of Days: " + numberOfDays);
    bookingInfo.setRoomPrice((int) (bookingRequest.getNumOfRooms() *
pricePerRoomPerDay * numberOfDays));
    System.out.println(bookingInfo.toString());
    return bookingDao.save(bookingInfo);
}
```

2. doPayment Method: This method is responsible for providing transaction details and here how it works
   Firstly, It checks whether correct payment method is used by using If method and takes booking Id from booking details.
   Secondly, It communicates with Payment micro service by using restTemplate and this microservice is responsible to provide with transaction Id.
   Lastly, Booking details are reverted back including transaction and booking confirmation.

   Following logic is used in this method.

```
public Booking doPayment(PaymentDTO paymentDetails) throws Exception {

    // call payment service and get paymentID to save in booking.
    System.out.println(paymentDetails.toString());
    String url = this.paymentServiceUrl + "/payment/transaction" ;

    if(!(paymentDetails.getPaymentMode().trim().equalsIgnoreCase("UPI") |
paymentDetails.getPaymentMode().trim().equalsIgnoreCase("CARD"))){
        throw new CustomException("Invalid mode of payment");
    }
    int bookingId = paymentDetails.getBookingId();
    Optional<Booking> bookingInfoOptional = bookingDao.findById(bookingId);
    if(bookingInfoOptional.isPresent()) {
        Booking bookingInfo = bookingInfoOptional.get();
        int trancationId = restTemplate.postForObject(url, paymentDetails,
Integer.class);
        bookingInfo.setTransactionId(trancationId);
        bookingDao.save(bookingInfo);
        String message = "Booking confirmed for user with aadhaar number: "
+ bookingInfo.getAadharNumber() +
                "     |     " + "Here are the booking details:     " +
bookingInfo.toString();
        //producer.send(new ProducerRecord<String,
String>("message","message", message));
        return bookingInfo;
    }else {
        throw new CustomException("Invalid Booking Id");
    }
}
```

- **Controller** : it is responsible for handling requests by intercepting http requests. In this case we're using post mappings to take user details and payment details.

```
@PostMapping("/booking")
public ResponseEntity<Booking> bookingDetails(@RequestBody BookingDTO
bookingRequest) throws Exception {
    Booking bookingInfo =
this.bookingService.bookingDetails(bookingRequest);
    return new ResponseEntity<Booking>(bookingInfo,
HttpStatus.CREATED);
```

API Documentation for the Booking Service

POST localhost:8081/hotel/booking

This an API defined in the Booking Service. This service is responsible to take input from user like- toDate, fromDate, aadharNumber and number of rooms required(count) and save it in its database and also generate a random list of room numbers depending on the count and display the room number list(roomNumbers) and roomPrice to the user, then if the user wants to go ahead with the booking, then he can simply provide the payment related details like bookingMode, upiId, cardNumber. These details which will be further sent to the payment service to and the booking will be confirmed.

This endpoint is responsible to collect information like fromDate, toDate,aadhar number,count from user and save it in its database and also returns the corresponding booking details.

Note that the transaction id returned in the response of this API will be '0' which represents that no transaction has happened so far for this booking. Only after the payment is done, transaction id will store the actual transaction id associated with the transaction.

HEADERS

Content-Type. application/json

```
{
    "fromDate": "2021-06-10",
    "toDate": "2021-06-15",
    "aadharNumber": "Sample-Aadhar-Number",
    "numOfRooms": 3
}
```

POST. localhost:8081/hotel/booking/1/transaction

This endpoint has to be defined in the Booking Service

This endpoint is responsible to take the payment-related details from the user and send it to the payment service; get the transactionId and save the corresponding transactionId in the booking table. Please note that for the paymentMode, if the user provides any other input apart from the 'UPI' or 'CARD', then it means that the user is not interested in the booking and wants to opt-out and this endpoint will throw a custom response with the message "Invalid mode of payment".

HEADERS. **Content-Type.** application/json

```
{
    "paymentMode": "UPI",
    "bookingId": 1,
    "upiId": "upi details",
    "cardNumber": ""
}
```

Exception 1: If the user gives any other input apart from “UPI” or “CARD”, the response message should look like the following:

```
{
   "message": "Invalid mode of payment",
  "statusCode": 400
}
```

Exception 2: If no transactions exist for the Booking Id passed to this endpoint then the response message should look like the following:

```
{
   "message": " Invalid Booking Id ",
  "statusCode": 400
}
```

Payment Service

This microservice is responsible to accept UPI and CARD payment and store transaction details and provide with transaction number.

Following logic is used in this service.

```
public Payment getPaymentById(int paymentId) throws Exception {

    Optional<Payment> paymentDetailDaoOp=
this.paymentDetailDao.findById(paymentId);

    if(paymentDetailDaoOp.isPresent()) {
      return paymentDetailDaoOp.get();
    }else {
      throw new Exception("Payment Id Not Found");
    }
  }
```

API Documentation for the 'Payment Service'

POST localhost:8083/payment/transaction

This is an endpoint defined in the payment service. The API to make transaction in the Booking service will interact with this endpoint to pass on the transaction details and get the transaction Id in return.

You can use this endpoint to unit-test the logic of the payment service.

HEADERS

Content-Type. application/json. **BODY.** raw

```
{
    "paymentMode": "UPI",
    "bookingId": 1,
    "upiId": "upi details",
    "cardNumber": ""
}
```

GET. localhost:8083/payment/transaction/1

This API is defined in the Payment Service.

This API is used to query the transaction details for a given transaction Id.

Eureka Client

Eureka Client is an application that holds the information about all client-service applications. Payment & Booking Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.