Coding Logic For Sweet Home Project

Server Ports

Eureka Server: 8761
API Gateway: 9191
Booking Service: 8081
Payment Service: 8083

Datasource

Database: In-memory store-h2
Username: sa
Password: password

Sequence Of Deployment

1. Eureka Server
2. API Gateway
3. Booking Service/Payment Service
4. Payment Service/Booking Service

Booking Microservice Packages & Files

Aspects:

PostPaymentAspect: To check the validity of booking ID and mode of payment and throw respective errors.

Controllers:

BookingController: Controller layer for the booking microservice.

DAO:

BookingDAO: Data Access Object of the booking microservice.

DTOs:

BookingRequestDTO: Data Transfer Object for a booking request
BookingResponseDTO: Data Transfer Object for a booking response
ExceptionErrorResponseDTO: Data Transfer Object for a generating error response

PaymentRequestDTO: Data Transfer Object for a payment request

Entities:

BookingInfoEntity: Model class of booking microservice.

Exceptions:

InvalidBookingIdException & InvalidPaymentModeException: Java classes for extending the respective exception classes.

Handlers:

CustomExceptionHandler: Java class to handle the exceptions

Services:

BookingService: Interface for Booking Microservice
BookingServiceImple: Implementation of booking microservice interface

Coding Logic For Booking Microservice

Method to generate random room numbers:

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

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

AOP approach to handle the invalid booking ID and payment mode type done before rest of the logic:

```java
@Override
public BookingResponseDTO postPaymentInfo(int bookingId, PaymentRequestDTO
paymentRequestDTO) {
   //AOP approach to handle the invalid booking ID and payment mode type done
before rest of the logic

   //Retrieving transaction id by calling payment service:
   String paymentServiceUrl = apiGatewayBaseUrl + "/payment/transaction";
   int transactionId = restTemplate.postForObject(paymentServiceUrl,
paymentRequestDTO, Integer.class);

   //updating transaction ID on the booking
   BookingInfoEntity bookingInfoEntity = bookingDAO.getById(bookingId);
   bookingInfoEntity.setTransactionId(transactionId);
   BookingInfoEntity savedBookingInfoEntity =
bookingDAO.save(bookingInfoEntity);

   // Printing the confirmation message on successful booking
   String message = "Booking confirmed for user with aadhaar number: "
           + savedBookingInfoEntity.getAadharNumber()
           +    "    |    "
           + "Here are the booking details:    " +
savedBookingInfoEntity.toString();
   System.out.println(message);


   BookingResponseDTO bookingResponseDTO =
modelMapper.map(savedBookingInfoEntity, BookingResponseDTO.class);
   return bookingResponseDTO;
}
```

Dealing with edge cases during booking rooms and selecting the dates for the stay:

```
if (numOfRooms <= 0)  throw new RuntimeException("Number of rooms requested
must be greater than 0.");
if (numOfDays<=0) throw  new RuntimeException("To-date must be greater than
From-date");
```

Payment Microservice Packages & Files

Controllers:

TransactionsController: Controller layer for the payment microservice.

DAO:

TransactionDAO: Data Access Object for the payment microservice.

DTOs:

TransactionDetailsDTO: Data Transfer Object for the translation details.

Entities:

TransactionDetailsEntity: Model class for the payment microservice.

Services:

TransactionService: Interface for Payment Microservice
TransactionServiceImple: Implementation of the interface for payment microservice.

Testing the Sweet Home Project

The project is built as per the guidelines and the POSTMAN API Documentation, this, it can be tested in the same manner.