Sweet-Home

The project is called ‘Hotel room booking application’ which is used to book rooms for a single hotel.

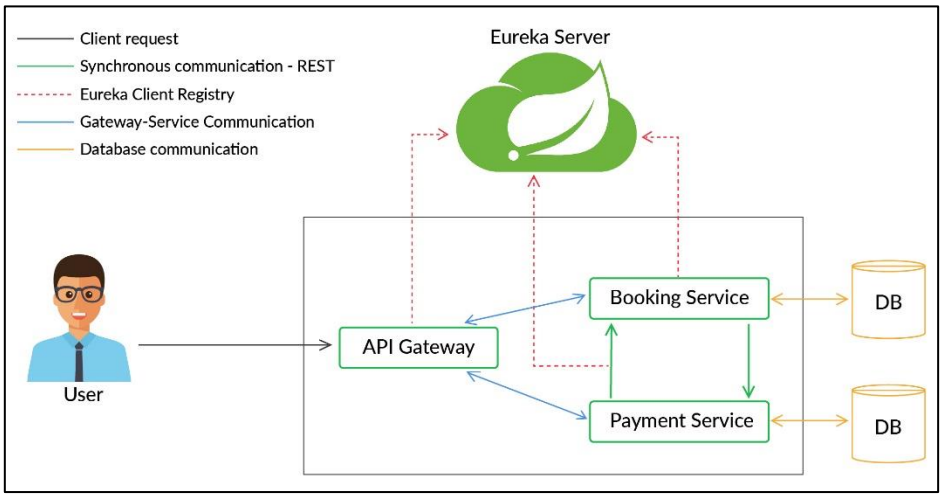


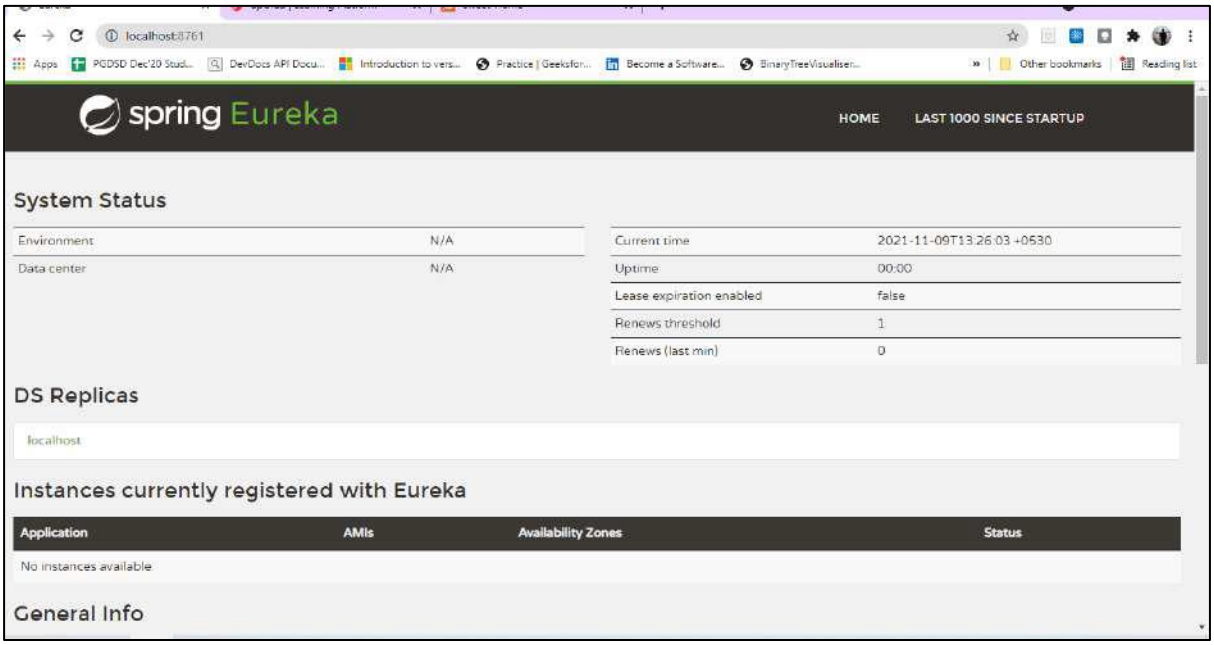
Figure 1 Schematic representation of architecture

1. API-Gateway

Initially, the API Gateway, Booking service and Payment service register themselves on the Eureka server. The user does not directly interact with Booking or Payment service. All requests are sent to API Gateway which then sends the requests to the relevant microservice.

Eureka: For self-registry

url: http://localhost:8761/



```
    <groupId>com.upgrad</groupId>
    <artifactId>Eureka</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>registry</name>
    <description>eureka registry</description>
    <properties>
        <java.version>11</java.version>
        <spring-cloud.version>2020.0.2</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
```

2. Booking and Payment Service

Booking service allows user to book, check availability, price, payment for a room

Steps Followed:

1. Created booking service database using command: **create database bookingdb ;**
   Similarly created payment service database using: **create database paymentdb;**
2. Open the Booking Service folder and direct to **src/main/resources/application.properties** and added following properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/bookingdb
spring.datasource.username=root
spring.datasource.password=Upgrad123
```

3. Run the application

4. Repeat the step 2 for payment service as well, navigate to Booking Service folder and direct to **src/main/resources/application.properties** and added following properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/paymentdb
spring.datasource.username=root
spring.datasource.password=Upgrad123
```

Refresh the Eureka server to see the PAYMENT-SERVICE and BOOKING-SERVICE instance

5. After adding application properties and configuring Eureka I created entities as per the sweet-home schema document given.
6. Created DAO layer by extending with JPA repository and also created DTO class

7. After which I implemented the service layer and applied logic:

Code for calculating number of days from the given from and to date , price is done as below in **addbooking** method in BookingServiceImpl.java

The code:

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. The logic to calculate room price is as follows:

```
public BookingInfoEntity addBooking(BookingInfoEntity bookingInfoEntity) {

    int totalDays;
    int noOfRooms = bookingInfoEntity.getNumOfRooms();
    int roomPrice;

    /**
     *  calculating number of days and price based on given room price as
Rs.1000 per room per day.
     */
    long difference = bookingInfoEntity.getToDate().getTime() -
bookingInfoEntity.getFromDate().getTime();
    totalDays = (int) (difference / (1000 * 60 * 60 * 24));
    roomPrice = 1000 * noOfRooms * (totalDays);

    ArrayList<String> roomNumbers =
getRandomNumbers(bookingInfoEntity.getNumOfRooms());
    String roomNumber = String.join(",", roomNumbers);

    bookingInfoEntity.setRoomNumbers(roomNumber);
    bookingInfoEntity.setRoomPrice(roomPrice);
    //Setting current Date as booked date
    bookingInfoEntity.setBookedOn(new Date());


    bookingDao.save(bookingInfoEntity);

    return bookingInfoEntity;

}
```

To generate a Room number used **getRandomNumber()** function to generate the random number according to the input user provided.

```
//Method to generate Random room numbers
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;
}
```

- Got Transaction ID from PAYMENT-SERVICE using RestTemplate in the below shown code:

```
@Override
public BookingInfoEntity getBookingData(Integer bookingID) {
    Optional<BookingInfoEntity> reqDetails =
bookingDao.findById(bookingID);
    if (reqDetails.isPresent()) {
        return reqDetails.get();
    }
    return null;
}


@Override
public BookingInfoEntity addPaymentData(int bookingId, PaymentDto
paymentDto) throws invalidBookingIdException, invalidPaymentModeException {
    String PaymentServiceURL = apiGateway + "/payment/transaction";
    if ("UPI".equals(paymentDto.getPaymentMode()) ||
"CARD".equals(paymentDto.getPaymentMode())) {
        if (!bookingDao.existsById(bookingId)) {
            throw new invalidBookingIdException();
        }
        // Getting Booking Data based on BookingID
        BookingInfoEntity bookingData = getBookingData(bookingId);
        // Getting transaction Id from Payment Service
        Integer transactionId =
restTemplate.postForObject(PaymentServiceURL,
                paymentDto, Integer.class);
        //Updating the transaction ID received from Payment service in
booking data
        bookingData.setTransactionId(transactionId);
        bookingDao.save(bookingData);
        String message = "Booking confirmed for user with aadhaar number: "
                + bookingData.getAadharNumber()
                + "    |    "
                + "Here are the booking details:    " +
bookingData.toString();

        //Printing message to console
        System.out.println(message);

        return bookingData;

    } else {
        throw new invalidPaymentModeException();
    }
}
```

Payment Service Code:

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 RDS database and returns the transactionId as a response.

```
@Service
public class PaymentServiceImpl implements PaymentService {
    @Autowired

    PaymentDao paymentDao;

     @Override
      public int addPaymentDetails(PaymentDto paymentDto) {
        TransactionDetailsEntity transactionDetailsEntity =new
TransactionDetailsEntity();

transactionDetailsEntity.setTransactionId(paymentDto.getTransactionId());

transactionDetailsEntity.setPaymentMode(paymentDto.getPaymentMode());
        transactionDetailsEntity.setCardNumber(paymentDto.getCardNumber());
        transactionDetailsEntity.setUpiId(paymentDto.getUpiId());
        transactionDetailsEntity.setBookingId(paymentDto.getBookingId());
                paymentDao.save(transactionDetailsEntity);

        return transactionDetailsEntity.getTransactionId();
    }

    //METHOD TO GET PAYMENT DETAILS FOR GIVEN TRANSACTION ID
    @Override
    public TransactionDetailsEntity getPaymentDetails(Integer
transactionId) {
        Optional<TransactionDetailsEntity> reqDetails =
paymentDao.findById(transactionId);
        if (reqDetails.isPresent()) {
            return reqDetails.get();
        }
        return null;
    }
}
```

8. Created the controller classes and added end points required for the project
9. Handled various exceptions using exception