

Port Numbers

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

Database Configuration

I have used the in-memory database h2.

By default, Spring Boot configures the application to connect to an in-memory store with the username sa and an empty password. The in-memory database is volatile, and data will be lost when we restart the application.

Configuration in Application.properties file

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
```

Eureka Configuration

For Eureka Clients (BookingService, PaymentService, API Gateway in application.yml file)

```
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/

  instance:
    hostname: localhost
```

Eureka Server (in application.yml file)

```
server:
  port: 8761

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

API Gateway Configuration

```
server:
  port: 9191

spring:
  application:
    name: API-GATEWAY
  cloud:
    gateway:
      routes:
        - id: BOOKING-SERVICE
          uri: lb://BOOKING-SERVICE
          predicates:
            - Path=/hotel/**

        - id: PAYMENT-SERVICE
          uri: lb://PAYMENT-SERVICE
          predicates:
            - Path=/payment/**

  discovery:
    enabled: true
```

Structure of Booking Service

Controller @RequestMapping(value = "/hotel")

Has the following dependencies

@Autowired

```
private BookingService bookingService;
```

This class has two methods for the two endpoints of Booking Service.

1. **bookingDetails** with @PostMapping(value = "/booking")
invokes bookingService.acceptBookingDetails(DTO)
2. **bookingConfirmation** with @PostMapping(value =
"/booking/{id}/transaction")
invokes bookingService.acceptPaymentDetails(DTO)

The second Controller method handles the following two exceptions -

1. "Invalid mode of payment" : checks if payment mode = "UPI" || "CARD", if true, only then invokes the bookingService. Otherwise sends back appropriate ResponseEntity

2. "Invalid Booking Id" : catches **BookingIdNotPresentException** thrown by `bookingService` and sends back appropriate `ResponseEntity`

BookingServiceImpl

Has the following dependencies

```
@Autowired
```

```
private BookingInfoDao bookingInfoDao;
```

```
@Autowired
```

```
private RestTemplate restTemplate;
```

```
@Value("${paymentService.url}") //in applications.properties
```

```
private String paymentServiceUrl;
```

This class contains the following two utility methods

1. **stringToLocalDate** : for converting `String` to `LocalDate`
2. **getRandomNumbers** : to generate room numbers based on count

This class contains the following two service methods

1. **acceptBookingDetails** takes `fromDate`, `toDate`, `aadharNumber`, `numOfRooms` from DTO. Calculates Room Price, Gets random room numbers from `getRandomNumbers`, sets `bookedOn` to `now()`, `transactionId` is set to 0 by default.
2. **acceptPaymentDetails** uses `bookingInfoDao` to find the `bookingInfoEntity` stored in the database. Throws `BookingIdNotPresentException` if not found. Uses `restTemplate` to call Payment Service through API Gateway. Sets the `transactionId`. Prints booking confirmation message on console.

DTO Classes

- Booking DTO : To map request body of endpoint 1

```
1 {
2     .... "fromDate": "2021-06-20",
3     .... "toDate": "2021-06-25",
4     .... "aadharNumber": "Sample-Aadhar-Number",
5     .... "numOfRooms": 3
6 }
```

- Payment DTO : To map request body of endpoint 2

```

1  {
2      "paymentMode": "CARD",
3      "bookingId": 1,
4      "upiId": "",
5      "cardNumber": "Card details"
6  }

```

- Exception DTO : To send back appropriate ResponseEntity in case of Exceptions

```

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

```

Structure of Payment Service

Controller @RequestMapping(value = "/payment")

Has the following dependencies

@Autowired

private PaymentService paymentService;

This class has two methods for the two endpoints of Payment Service.

1. **bookingDetails** with @PostMapping(value = "/transaction")
invokes paymentService.acceptPaymentDetails(DTO)
2. **bookingConfirmation** with @GetMapping(value = "/transaction/{id}")
invokes paymentService.getTransactionDetails(id)

PaymentServiceImpl

Has the following dependencies -

@Autowired

private TransactionDetailsDao transactionDetailsDao;

@Autowired

ModelMapper modelMapper;

This class contains the following two service methods

1. **acceptPaymentDetails** uses DTO to create a new transactionDetailsEntity and saves it in the database using transactionDetailsDao.
2. **getTransactionDetails** uses transactionDetailsDao to find the transactionDetailsEntity and then converts it to TransactionDTO using ModelMapper

DTO Classes

- Payment DTO : To map request body of endpoint 1

```
1 {  
2   .... "paymentMode": "CARD",  
3   .... "bookingId": 1,  
4   .... "upiId": "",  
5   .... "cardNumber": "Card details"  
6 }
```

(Does not contain Transaction Id)

- Transaction DTO : To send back appropriate ResponseEntity (endpoint 2)

```
1 {  
2   "id": 1,  
3   "paymentMode": "CARD",  
4   "bookingId": 1,  
5   "upiId": "",  
6   "cardNumber": "Card details"  
7 }
```

(Contains Transaction Id)