

Hotel room booking application

****Run all the applications separately as IntelliJ Community Edition is used by me ****

Important Note: Database used is H2. ***

1. **Booking service** - This service is responsible for collecting all information related to user booking and sending a confirmation message once the booking is confirmed.
2. **Payment service** - This is a dummy payment service; this service is called by the booking service for initiating payment after confirming rooms.
3. **Eureka-service** – Setup of the eureka server for registering both the application.

Step 1: Start Booking-service, Payment-service application and Eureka-server application.

Eureka server:

```
2022-07-25 13:13:09.747 INFO 7920 --- [ Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : isAws returned false
2022-07-25 13:13:09.747 INFO 7920 --- [ Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Initialized server context
2022-07-25 13:13:09.747 INFO 7920 --- [ Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 instances from neighboring DS node
2022-07-25 13:13:09.747 INFO 7920 --- [ Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew threshold is: 1
2022-07-25 13:13:09.747 INFO 7920 --- [ Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing status to UP
2022-07-25 13:13:09.757 INFO 7920 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8761 (http) with context path ''
2022-07-25 13:13:09.762 INFO 7920 --- [ main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
2022-07-25 13:13:09.779 INFO 7920 --- [ Thread-10] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
```

Payment Server:

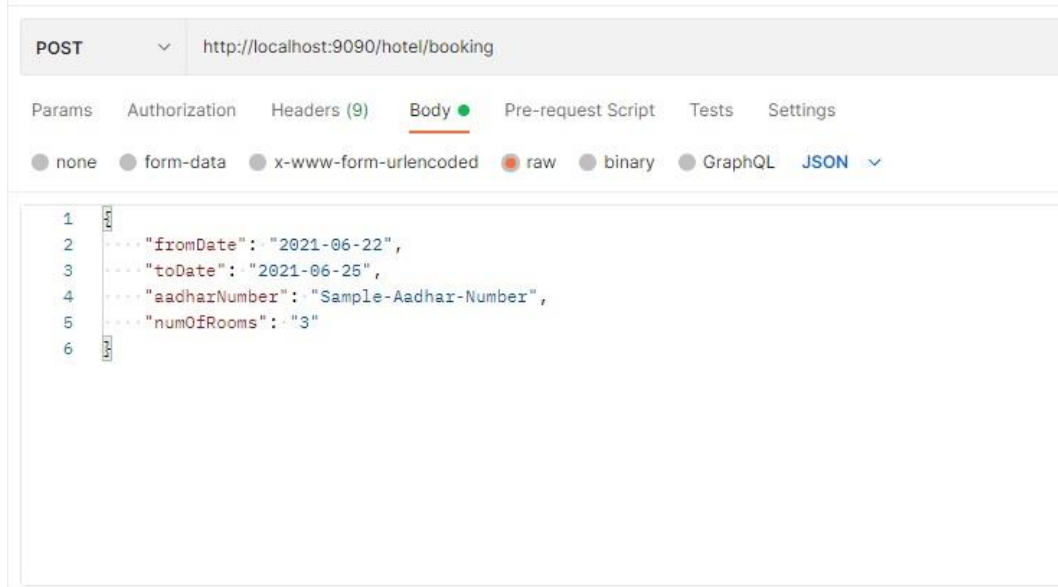
```
2022-07-25 12:54:13.417 INFO 8490 --- [ main] com.netflix.discovery.DiscoveryClient : The response status is 200
2022-07-25 12:54:13.422 INFO 8490 --- [ main] com.netflix.discovery.DiscoveryClient : Starting heartbeat executor: renew interval is: 30
2022-07-25 12:54:13.426 INFO 8490 --- [ main] com.netflix.discovery.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4
2022-07-25 12:54:13.430 INFO 8490 --- [ main] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1658733853435 with initial instances count: 0
2022-07-25 12:54:13.438 INFO 8490 --- [ main] o.s.c.n.e.s.EurekaServiceRegistry : Registering application PAYMENT-SERVICE with eureka with status UP
2022-07-25 12:54:13.443 INFO 8490 --- [ main] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1658733853443, current=UP, previous=STARTING]
2022-07-25 12:54:13.447 INFO 8490 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_PAYMENT-SERVICE/host.docker.internal:payment-service:9091: registering service...
2022-07-25 12:54:13.538 INFO 8490 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9091 (http) with context path ''
2022-07-25 12:54:13.538 INFO 8490 --- [ main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 9091
2022-07-25 12:54:13.555 INFO 8490 --- [ main] o.e.p.PaymentServiceApplication : Started PaymentServiceApplication in 11.361 seconds (JVM running for 13.398)
```

Booking Server:

```
2022-07-25 12:57:04.482 INFO 22508 --- [ main] c.n.discovery.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4
2022-07-25 12:57:04.490 INFO 22508 --- [ main] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1658734024489 with initial instances count: 1
2022-07-25 12:57:04.493 INFO 22508 --- [ main] o.s.c.n.e.s.EurekaServiceRegistry : Registering application BOOKING-SERVICE with eureka with status UP
2022-07-25 12:57:04.494 INFO 22508 --- [ main] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1658734024494, current=UP, previous=STARTING]
2022-07-25 12:57:04.498 INFO 22508 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_BOOKING-SERVICE/host.docker.internal:booking-service:9090: registering service...
2022-07-25 12:57:04.584 INFO 22508 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9090 (http) with context path ''
2022-07-25 12:57:04.587 INFO 22508 --- [ main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 9090
2022-07-25 12:57:04.591 INFO 22508 --- [ main] o.e.b.BookingServiceApplication : Started BookingServiceApplication in 11.235 seconds (JVM running for 13.46)
2022-07-25 12:57:04.609 INFO 22508 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_BOOKING-SERVICE/host.docker.internal:booking-service:9090 - registration status: 204
```

API Number 1 (Booking Service):

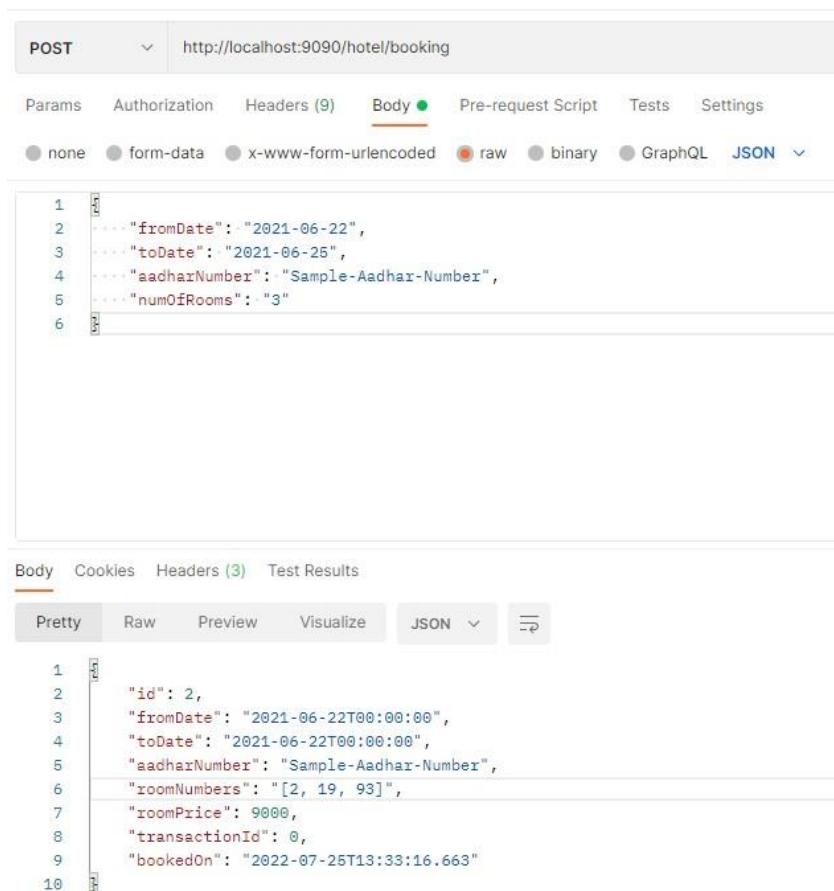
API: <http://localhost:9090/hotel/booking>



Response for booking service amount generated with particular set of code as shown below

```
(1000 * request.getNumOfRooms() * numberOfDays))
```

Response initial transactionId is 0 at the start as per the requirement.



API number 2: To get the transaction details.

This API makes a call to payment service to store the values into transaction table, which will help in generating transactionId which can we use to check whether payment against particular booking is completed or is incomplete.

POST API <http://localhost:9090/hotel/booking/2/transaction>

The screenshot shows a REST client interface with a POST request to `http://localhost:9090/hotel/booking/2/transaction`. The 'Body' tab is selected, showing a JSON payload. Below the request, the 'Test Results' tab is active, displaying a successful response in JSON format.

```
POST http://localhost:9090/hotel/booking/2/transaction

{
  "paymentMode": "CARD",
  "bookingId": "2",
  "upiId": "Sample-UPI-Number",
  "cardNumber": "card-Details"
}
```

```
{
  "id": 2,
  "fromDate": "2021-06-22T00:00:00",
  "toDate": "2021-06-22T00:00:00",
  "aadharNumber": "Sample-Aadhar-Number",
  "roomPrice": 9000,
  "transactionId": 2,
  "bookedOn": "2022-07-25T14:08:38.311"
}
```

There are few checks involved.

1. Payment Mode should be only in **"CARD"** or **"UPI"**, if payment Mode is done using any other way error will be displayed stating `"message": "Invalid mode of payment"`.

The screenshot shows the same REST client interface, but the request body has been modified to use `"CARD1"` as the payment mode. The response now shows an error message and a 404 status code.

```
POST http://localhost:9090/hotel/booking/2/transaction

{
  "paymentMode": "CARD1",
  "bookingId": "2",
  "upiId": "Sample-UPI-Number",
  "cardNumber": "card-Details"
}
```

```
{
  "message": "Invalid mode of payment",
  "statusCode": 404
}
```

2. Add on check for Invalid bookingId, if we have any invalid booking id response generated will be as follows for the particular API

<http://localhost:9090/hotel/booking/34/transaction>

The screenshot displays a REST client interface. At the top, the method is set to **POST** and the URL is `http://localhost:9090/hotel/booking/34/transaction`. Below the URL bar, there are tabs for **Params**, **Authorization**, **Headers (9)**, **Body** (which is selected), **Pre-request Script**, **Tests**, and **Settings**. Under the **Body** tab, there are radio buttons for **none**, **form-data**, **x-www-form-urlencoded**, **raw** (selected), **binary**, and **GraphQL**. A dropdown menu shows **JSON** is selected. The request body is a JSON object with the following content:

```
1 {
2   "paymentMode": "CARD",
3   "bookingId": "2",
4   "upiId": "Sample-UPI-Number",
5   "cardNumber": "card-Details"
6 }
```

Below the request body, there are tabs for **Body** (selected), **Cookies**, **Headers (3)**, and **Test Results**. Under the **Body** tab, there are buttons for **Pretty**, **Raw**, **Preview**, **Visualize**, and a **JSON** dropdown menu. The response body is shown in the **Pretty** view:

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

Payment Service:

Get API to get the transaction details.

For a valid transaction, following will be the response.

UpGrad / transaction/2

GET

▼

http://localhost:9091/payment/transaction/1

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Query Params

	KEY
	Key

body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Visualize

JSON ▼

↺

```
1  {
2    "id": 1,
3    "paymentMode": "CARD",
4    "bookingId": 1,
5    "upiId": "Sample-UPI-Number",
6    "cardNumber": "card-Details"
7  }
```

If any invalid data is passed response will be null and if we have a transaction with respect to booking it will respond with a transactionId.

UpGrad / transaction/2

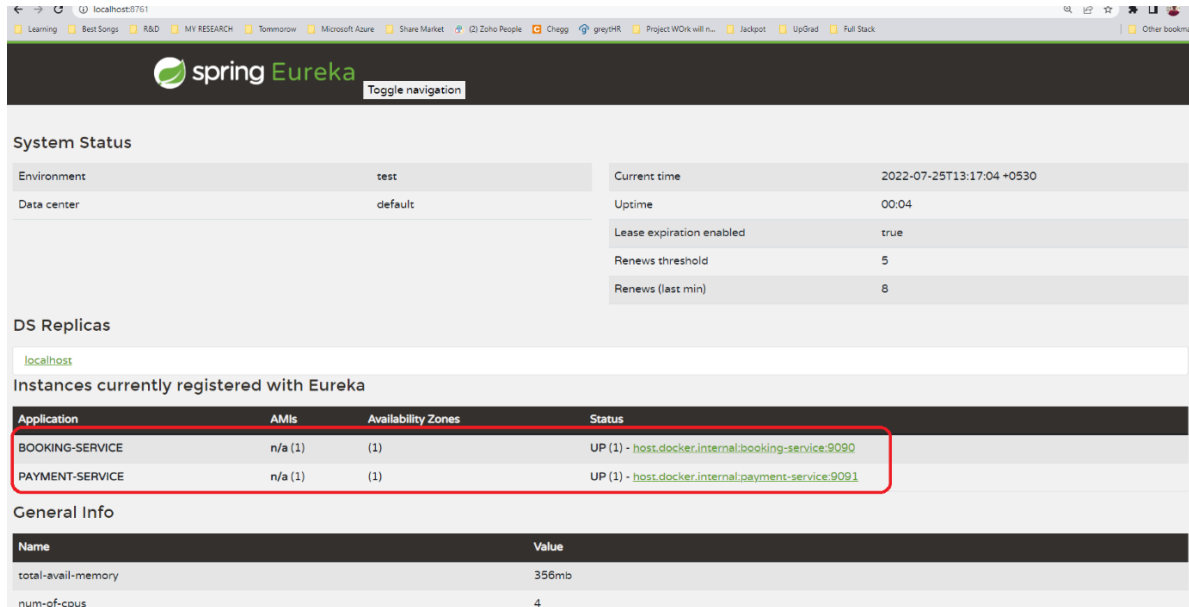
GET	▼	http://localhost:9091/payment/transaction/1800
Params	Authorization	Headers (7)
Body	Pre-request Script	Tests
Settings		
Query Params		
	KEY	VALUE
	Key	Value

Body	Cookies	Headers (3)	Test Results
Pretty	Raw	Preview	Visualize
JSON	▼		
1			
2	"id": null,		
3	"paymentMode": null,		
4	"bookingId": 0,		
5	"upiId": null,		
6	"cardNumber": null		
7			

We have another API which is called using booking Application, where in we generate the Transaction Id with respect to the payment done.

Eureka Service:

Post all the configuration both the application booking and payment application will be registered in eureka server.



The screenshot displays the Spring Eureka web interface. At the top, the 'spring Eureka' logo is visible with a 'Toggle navigation' button. Below the header, the 'System Status' section provides details about the environment (test), data center (default), current time (2022-07-25T13:17:04 +0530), uptime (00:04), lease expiration (enabled), renew threshold (5), and renew interval (8 minutes). The 'DS Replicas' section shows 'localhost'. The 'Instances currently registered with Eureka' section contains a table with the following data:

Application	AMIs	Availability Zones	Status
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:booking-service:9090
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:payment-service:9091

The table is highlighted with a red border. Below this, the 'General Info' section shows system metrics: total-avail-memory (356mb) and num-of-cpus (4).