

1. Booking Service:

Added dependent libraries

- Eureka Discovery Client
- Spring Data JPA
- MySQL Driver
- Spring Web
- Kafka client

Added profile management, to select the database based on the selected profile (DEV, DEPLOY)

- DEV – this will enable H2 db (uses application-DEV.properties)
- DEPLOY – this will enable MySQL db hosted on AWS RDS (uses application-DEPLOY.properties)

Code currently submitted with DEPLOY profile.

To update the end point of MySQL RDS, uses `application-DEPLOY.properties` to be updated

Implemented the service with multi layered design consisting of Controller, Service, DAO, Exception, Model, Entity, DTO, Util layers

Exposed 2 Rest EndPoints

I. <http://localhost:8080/booking>

- This is a post call to persist the booking information and return the booking details
- Service layer implements the business logic of
 - i. Getting available rooms
 - ii. Calculating the room price based on no. of rooms, no. of days and room price
- Dao layer persists the entity into the DB
- Exceptions handled and logged as error
- Controller advice written to calculate the time taken for the service to be processed

II. <http://localhost:8080/booking/{{bookingId}}/transaction>

- This is a post call to update the transaction details into the booking info by connecting to Payment-Service
- Service layer implements the business logic of
 - i. Fetching the Booking based on Booking Id by calling DAO class
 - ii. call payment service using RestTemplate to get the transaction id for the transaction info
 - iii. use DAO layer to update the booking information based on the response from above step
 - iv. publish the notification on to kafka topic “message”
- Dao layer persists the entity into the DB
- Exceptions handled and logged as error

- Exception handler will respond with ErrorModel for exception cases
- Controller advice written to calculate the time taken for the service to be processed

2. **Payment Service**

Added dependent libraries

- Eureka Discovery Client
- Spring Data JPA
- MySQL Driver
- Spring Web

Added profile management, to select the database based on the selected profile (DEV, DEPLOY)

- DEV – this will enable H2 db (uses application-DEV.properties)
- DEPLOY – this will enable MySQL db hosted on AWS RDS (uses application-DEPLOY.properties)

Code currently submitted with DEPLOY profile.

To update the end point of MySQL RDS, uses application-DEPLOY.properties to be updated

Implemented the service with multi layered design consisting of Controller, Service, DAO, Exception, Model, Entity, DTO, Util layers

Exposed 2 Rest EndPoints

I. <http://localhost:8083/transaction>

- This is a post call to persist the booking information and return the booking details
- Service layer implements the business logic of
 - i. Map dto into entity and enrich the entity
 - ii. Calculating the room price based on no. of rooms, no. of days and room price
- Dao layer persists the entity into the DB
- Exception handler will respond with ErrorModel for exception cases
- Controller advice written to calculate the time taken for the service to be processed

II. <http://localhost:8083/transaction/{{transactionId}}>

- This is a get call to get the transaction details
- Service layer implements the business logic of
 - i. Fetching the transaction based on transaction Id by calling DAO class
- Dao layer persists the entity into the DB
- Exception handler will respond with ErrorModel for exception cases
- Controller advice written to calculate the time taken for the service to be processed

3. Eureka Server

Run on port 8761

Payment service registers as PAYMENT-SERVICE

Booking service registers as BOOKING-SERVICE

4. Notification Service

Simple java project with a main class written to consume messages from kafka topic “message” which is hosted on AWS EC2 instance

5. AWS EC2 created to host Kafka.

After installing java and kafka, server.properties updated with IP of the EC2 elastic IP.

Zookeeper server to be started

Kafka server to be started

Kafka topic created with replication factor 1 and partition 1

6. AWS RDS created

Schema created: sweethome

- create schema sweethome

Tables automatically created with help of spring jpa

ScreenShots:

Eureka – service registry

DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
BOOKING-SERVICE	n/a (1)	(1)	UP (1) - Coreus-BOOKING-SERVICE.8080
PAYMENT-SERVICE	n/a (1)	(1)	UP (1) - Coreus-PAYMENT-SERVICE.8083

Post on booking

POST

localhost:8080/booking

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1  {
2    "fromDate": "2021-06-16",
3    "toDate": "2021-07-25",
4    "aadharNumber": "Akash Sinha-Aadhar Number",
5    "numOfRooms": 2
6  }
```

BodyCookiesHeaders (5)Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "id": 1,
3    "fromDate": "2021-06-16T00:00:00",
4    "toDate": "2021-07-25T00:00:00",
5    "aadharNumber": "Akash Sinha-Aadhar Number",
6    "numOfRooms": 2,
7    "roomNumbers": "33,93",
8    "roomPrice": 78000,
9    "transactionId": 0,
10   "bookedOn": "2021-12-07T19:17:27.0177048"
11 }
```

7. Transaction with invalid booking id – exception handled

The screenshot displays a REST client interface with a tab for a POST request to `localhost:8080/booking/1/transaction`. The request body is a JSON object:

```
{  "paymentMode": "CARD",  "bookingId": 2,  "upiId": "MyUPIId",  "cardNumber": "Test Card 3"}
```

The response is shown in the 'Body' tab, formatted as JSON:

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

The interface includes tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, and the response is displayed in a 'Pretty' format. The status code 400 indicates a client error, specifically an invalid booking ID.

8. Transaction success

The screenshot shows a REST client interface with a POST request to `localhost:8080/booking/1/transaction`. The request body is a JSON object with the following fields:

```
{  "paymentMode": "CARD",  "bookingId": 1,  "upiId": "MyUPIId",  "cardNumber": "Test Card 3"}
```

The response is displayed in the 'Body' tab, showing a JSON object with the following fields:

```
{  "id": 1,  "fromDate": "2021-06-16T00:00:00",  "toDate": "2021-07-25T00:00:00",  "aadharNumber": "Akash Sinha-Aadhar Number",  "numOfRooms": 2,  "roomNumbers": "33,93",  "roomPrice": 78000,  "transactionId": 2,  "bookedOn": "2021-12-07T19:17:27.017705"}
```

9. Notification in Kafka

Message
offset = 20, key = Notification-2, value = Booking confirmed for user with aadhaar number: Akash Sinha-Aadhar Number | Here are the booking details: BookingInfoEntity(id=1, fromDate=2021-06-16T00:00, toDate=2021-07-25T00:00, aadhaarNumber=Akash Sinha-Aadhar Number, numOfRooms=2, roomNumbers=33,93, roomPrice=78000, transactionId=2, bookedOn=2021-12-07T19:17:27.017705)