Project Name : SWEET-HOME **Student Name** : Omdev Dalsaniya

Project Walkthrough :

a) Eureka Server (Configured Port : 8761)

- Spring Boot Project with Eureka dependency

b) API GateWay (Configured Port : 9191)

- Actuator, Gateway dependency
- Add path property in application.properties for BOOKING-SERVICE and PAYMENT-SERVICE
- Eureka Client registered in YAML file

c) Booking Service (Configured Port : 8081)

- WebClient, Spring dependency
- Eureka Client registered in YAML file
- Controller :
  i. /hotel/booking/{id}
  ii. /hotel/booking
  iii. /hotel/booking/{id}/transaction
- DAO : BookingDAO
- DTO :
  i. BookingDTO
  ii. TransactionDTO
- Service :
  i. BookingService
  ii. BookingServiceImpl
- Entity/model :
  i. BookingInfoEntity
  ii. TransactionDetailEntity

d) Payment Service (Configured Port : 8083)

- WebClient, Spring dependency
- Eureka Client registered in YAML file
- Controller :
  i. /transaction
  ii. /transaction/{id}
- DAO : PaymentDAO
- DTO :
  i. BookingDTO
  ii. TransactionDTO
- Service :
  i. PaymentService
  ii. PaymentServiceImpl
- Entity/model :
  i. BookingInfoEntity
  ii. TransactionDetailEntity

STEPS FOLLOWED TO SOLVE THE PROJECT

BOOKING SERVICE

Step 1: I first created BookingInfoEntity class for Booking Service Project

Step 2: Required properties and yml files added with configuration with eureka client and port number

Step 3: I added configuration for h2 database

Step 4: I created BookingDAO and BookingService along with BookingServiceImpl

Step 5: Then I created BookingController to implement Standalone API's. To implement Api's, I created DTO to send as response. Helping methods were added to Util class.

VERIFICATION

1. Eureka client server and standalone api first to check connectivity with API-GATEWAY

PAYMENT SERVICE

Step 1: I created TransactionDetailEntity class for Payment Service Project

Step 2: Required properties and yml files added with configuration with eureka client and port number

Step 3: I added configuration for h2 database

Step 4: I created paymentDAO and paymentService along with PaymentServiceImpl

Step 5: Then I created PaymentController to implement dependent API's. Created DTO for same, used same DTO as BookingService (i.e. BookingDTO). Used "BookingInfoEntity" class to deal with the response of Booking Service's API.

Step 6: Added same DTO and Entity to BookingService also.

Step 7: As endpoint 1 of PaymentService will be called by endpoint 2 of BookingService, RestTemplate was autowired to both the Application file in both the project.

SYNC CALLS

Step 1: Make Payment (/hotel/booking/{id}/transaction)

a) Added validation for request body for payment mode
b) Added validation to check booking id -> if exists in db or not
c) Saved transaction with the help of Payment Service /payment/transaction
d) Added required HttpEntity and postForObject method with restTemplate
e) Get required transactionId from Payment Service
f) Set the same id to BookingService
g) Used Booking Service to save the booking
h) Updated entity is returned in Response Entity

Step 2: Get booking call from Payment Service

a) Required method is added using RestTemplate with getForObject
b) Logged the same

VERIFICATION

1) POST : http://localhost:9191/hotel/booking
   Hit this API to save the booking details in BOOKING SERVICE (db).
2) GET : http://localhost:9191/hotel/booking/1
   Hit this API to get booking with booking Id.
3) POST : http://localhost:9191/hotel/booking/1/transaction
   It internally hit API in this order
   a) POST: http://localhost:9191/payment/transaction
      It stores the transaction in the PAYMENT SERVICE (db).
      This api returns the transactionId, with is than added to BookingEntity
   b) GET: http://localhost:9191/hotel/booking/1
      API a) logs the detail of bookingInfo and logs the same.
4) GET: http://localhost:9191/payment/transaction/1
   It fetches the transaction with Id returns the same