Sweet-Home Project Coding Logic

---

1. Create a Folder "Sweet-Home"

Create a folder named "Sweet-Home" to contain all the services: API Gateway, Booking Service, Payment Service, and Eureka Server.

2. Service Dependencies

Booking Service:
Dependencies : Spring Cloud Netflix Eureka Client, Spring Boot Web, Spring Boot Data JPA

Payment Service:
Dependencies: Spring Cloud Netflix Eureka Client, Spring Boot Web, Spring Boot Data JPA

API Gateway:
Dependencies: Spring Boot Actuator, Spring Cloud Netflix Eureka Client

Eureka Server:
Dependencies: Spring Cloud Netflix Eureka Server

---

3. Booking Service

3.1 Model Classes

Create an entity class named `BookingInfoEntity`:

- Fields: `fromDate`, `toDate`, `aadharNumber`, `numOfRooms`, `roomNumbers`, `roomPrice`, `transactionId`, `bookedOn`
- Define appropriate data types for each field. Use `Date` for dates and `String` for textual data.
- Use annotations to map the class to a database table.
- Set default value 0 for `transactionId`.

3.2 Repository Layer

Create a repository interface:
  - Define a repository interface extending `JpaRepository` to handle CRUD operations for `BookingInfoEntity`.

3.3 Controller Layer

Endpoint 1: Create Booking
- URI: /booking
- HTTP Method: POST
- Request Body: fromDate, toDate, aadharNumber, numOfRooms
- Logic:
  - Generate random room numbers based on `numOfRooms`.
  - Calculate the `roomPrice` using the formula: `roomPrice = 1000  numOfRooms  (number of days)`.
  - Save booking details in the database with a default `transactionId` of 0.
  - Return the created booking information.

Endpoint 2: Handle Payment
- URI: /booking/{bookingId}/transaction
- HTTP Method: POST
- Path Variable: bookingId
- Request Body: paymentMode, upiId, cardNumber
- Logic:
  - Validate `paymentMode` (must be 'UPI' or 'CARD').
  - If invalid, return an error message.
  - If valid, send payment details to the Payment Service and retrieve `transactionId`.
  - Update the `transactionId` in the booking table.
  - Return a confirmation message.

3.4 Configuration

- Run on Port: 8081
- Configure as Eureka Client:
  - Set up Eureka client properties to register the booking service with the Eureka server.

---

4. Payment Service

4.1 Model Classes

- Create an entity class named `TransactionDetailsEntity`:

  - Fields: `transactionId`, `bookingId`, `paymentMode`, `upiId`, `cardNumber`, `transactionDate`
  - Define appropriate data types for each field.
  - Use annotations to map the class to a database table.

 4.2 Repository Layer

- Create a repository interface:
  - Define a repository interface extending `JpaRepository` to handle CRUD operations for `TransactionDetailsEntity`.

 4.3 Controller Layer

 Endpoint 1: Process Payment
- URI: /transaction
- HTTP Method: POST
- Request Body: bookingId, paymentMode, upiId, cardNumber
- Logic:
  - Generate a unique `transactionId`.
  - Save payment details in the database.
  - Return the generated `transactionId`.

 Endpoint 2: Get Transaction Details
- URI: /transaction/{transactionId}
- HTTP Method: GET
- Path Variable: transactionId
- Logic:
  - Retrieve transaction details based on `transactionId`.
  - Return the transaction details.

 4.4 Configuration

- Run on Port: 8083
- Configure as Eureka Client:
  - Set up Eureka client properties to register the payment service with the Eureka server.

---

 5. API Gateway (Optional)

 5.1 Configuration

- Run on Port: 9191
- Configure as Eureka Client:
  - Register the API Gateway with Eureka server.

  - Configure routes to forward requests to Booking and Payment services using their respective IDs and URIs.
  - Define path predicates to route specific requests to the appropriate service.

 5.2 Routes Configuration

- Booking Service:
  - ID: BOOKING-SERVICE
  - URI: lb://BOOKING-SERVICE
  - Path: /hotel/
- Payment Service:
  - ID: PAYMENT-SERVICE
  - URI: lb://PAYMENT-SERVICE
  - Path: /payment/

---

 6. Eureka Server

 6.1 Configuration

- Run on Port: 8761
- Enable Eureka Server:
  - Annotate the main class with the appropriate annotation to enable Eureka server.
  - Configure properties for standalone Eureka server.

 6.2 Test and Run

- Start the Eureka server and services:
  - Ensure the Eureka server is running.
  - Start the API Gateway, Booking service, and Payment service.
  - Verify that services are registered and visible on the Eureka dashboard.

By following these steps, you can set up the Booking Service, Payment Service, API Gateway, and Eureka Server for the "Sweet-Home" project, ensuring proper integration and functionality.