Detailed Walkthrough of Hotel Room Booking Microservices

This document provides a detailed, step-by-step walkthrough of the logic implemented in each microservice of the Hotel Room Booking application. It includes user interactions, microservice communication, and database updates.

1. User Requests a Room Booking:

i. **User initiates booking:** Users initiate the booking process by providing essential details such as check-in, check-out dates, Aadhaar number, and the desired number of rooms through the Booking service. The API Gateway is optional but serves as a central entry point for all requests.

ii. **Booking service validation:** The Booking service validates the user input, including date validity, Aadhaar number format, and room availability. Any invalid input triggers appropriate exceptions with error messages displayed to the user, ensuring data integrity.

iii. **Room generation and pricing:** Upon successful validation, the Booking service generates a random list of room numbers based on the specified number of rooms. It also calculates the total room price based on the number of days between check-in and check-out dates, providing a transparent pricing mechanism.

2. User Decides to Proceed with Booking:

i. **Payment details input:** If the user decides to proceed, they input payment details such as payment mode (UPI or CARD) along with the respective identifiers (UPI ID or card number).

ii. **Booking confirmation:** The Booking service verifies the payment mode and asynchronously communicates with the Payment service using REST Template. This ensures that the payment processing does not block the booking confirmation process.

3. Payment Service Processing:

i. **Payment information reception:** The Payment service receives the booking information and payment details from the Booking service.

ii. **Dummy transaction simulation:** The Payment service simulates a dummy transaction, generating a unique transaction ID. It emphasizes that this is a placeholder for actual payment processing and does not involve real financial transactions.

iii. **Database update:** The Payment service updates its database with transaction details, including the generated transaction ID and booking information.

iv. **Response to Booking service:** The Payment service responds to the Booking service with the generated transaction ID.

4. Booking Confirmation and Completion:

i. **Transaction ID update:** The Booking service receives the transaction ID from the Payment service and updates its booking table with the received ID.
ii. **Confirmation message:** The Booking service prints a confirmation message to the console, including the user's Aadhaar number and other relevant booking details.
iii. **User experience:** The confirmation message can be displayed to the user or sent through additional channels such as email or SMS, enhancing the user experience.

Additional Points To Consider:

- **Synchronous communication:** Emphasizing synchronous communication between services ensures accurate booking confirmation messages before displaying them to users.
- **Database schemas:** The model classes for both Booking and Payment services adhere to the database schemas, promoting data consistency and structure.
- **Exception Handling:** Error handling and exceptions are implemented at each step to enhance the robustness of the system.

Project Overview:

API-Gateway:

Acts as the entry point, routing requests to internal microservices.

Booking Service:

Collects user booking information, validates, generates room details, handles payment, and confirms bookings.

Payment Service:

Dummy service simulating payment processing, interacting with the Booking service for transaction details.

Application Workflow:

Services register on the Eureka server, with the API Gateway serving as the intermediary for user requests.

User initiates room booking through the Booking service, receiving room details and prices.

Payment details are provided by the user, and a dummy transaction is simulated by the Payment service.

Booking is confirmed, and a confirmation message is printed.

Service Configuration:

Booking Service (port 8081):

Endpoint 1: Responsible for collecting user information and saving it in the database.

Endpoint 2: Takes payment-related details, sends them to the Payment service, and updates the booking table.

Configured as a Eureka client.

Payment Service (port 8083):

Endpoint 1: Simulates a transaction, returns a transactionId, and saves details in the 'transaction' table.

Endpoint 2: Presents transaction details based on the provided transactionId.

Configured as a Eureka client.

Dependencies:

Booking Service dependencies include Spring Cloud Netflix Eureka Client, Spring Boot Web, Spring Boot Data JPA, and Spring Boot Devtools.

Payment Service dependencies mirror those of the Booking Service.

API Gateway dependencies include Spring Boot Actuator and Spring Cloud Netflix Eureka Client.

Eureka Server dependencies include Spring Cloud Netflix Eureka Client and Eureka Discovery Server.

Conclusion:

The described microservices architecture provides modularity, scalability, and ease of maintenance. By breaking down the Hotel Room Booking application into distinct services, each with its responsibilities, the system becomes more robust and adaptable to future changes. The use of synchronous communication, proper error handling, and adherence to best practices in microservices design contribute to a reliable and efficient application.

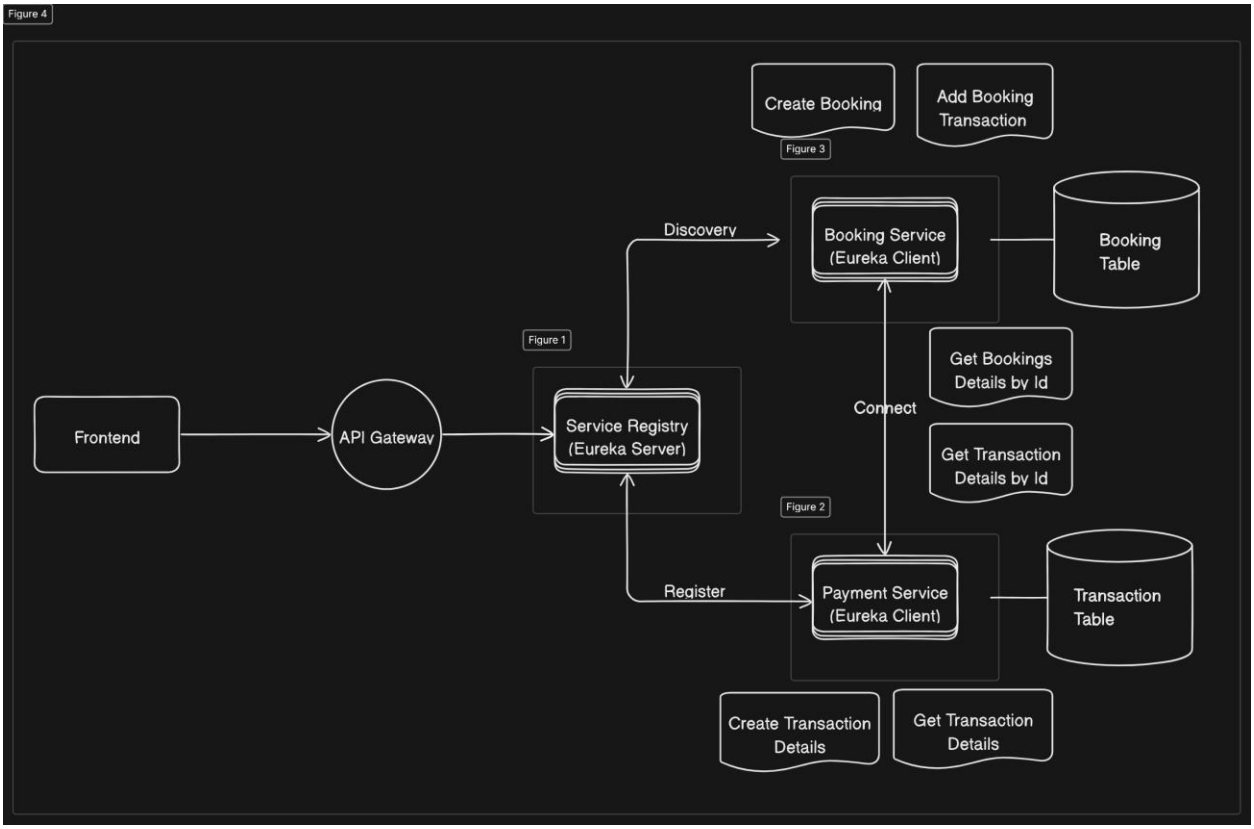

Figure 1: High Level Diagram