**Detailed Walkthrough of Hotel Room Booking Microservices**

This document offers an in-depth, step-by-step guide to the intricacies implemented in each microservice of the Hotel Room Booking application. It covers user interactions, microservice communication, and database updates.

**1. User Requests a Room Booking:**

* *i. User initiates booking:* The booking process begins when users provide essential details, including check-in, check-out dates, Aadhaar number, and the desired number of rooms through the Booking service. The API Gateway, optional yet acting as a centralized entry point, directs all requests.
* *ii. Booking service validation:* Thorough validation occurs within the Booking service, checking date validity, Aadhaar number format, and room availability. Invalid inputs trigger exceptions, ensuring data integrity by displaying pertinent error messages to the user.
* *iii. Room generation and pricing:* Post successful validation, the Booking service generates a random list of room numbers based on the specified room quantity. Transparent pricing is calculated, considering the number of days between check-in and check-out dates.

**2. User Decides to Proceed with Booking:**

* *i. Payment details input:* Users, if inclined to proceed, input payment details such as payment mode (UPI or CARD) along with respective identifiers (UPI ID or card number).
* *ii. Booking confirmation:* The Booking service, upon user confirmation, verifies the payment mode and asynchronously communicates with the Payment service using REST Template. This non-blocking approach ensures that payment processing doesn't impede the booking confirmation process.

**3. Payment Service Processing:**

* *i. Payment information reception:* The Payment service receives 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's essential to emphasize that this step serves as a placeholder for actual payment processing and does not involve real financial transactions.
* *iii. Database update:* Post dummy transaction simulation, 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:* Following a successful update, the Booking service prints a confirmation message to the console, including the user's Aadhaar number and other pertinent 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 overall user experience.

**Additional Points To Consider:**

* *Synchronous communication:* The emphasis on synchronous communication between services ensures accurate booking confirmation messages are received before displaying them to users.
* *Database schemas:* The model classes for both Booking and Payment services strictly adhere to the established database schemas, promoting data consistency and structural integrity.
* *Exception Handling:* Rigorous error handling and exception management are implemented at each step, fortifying the system's robustness.

**Project Overview:**

* *API-Gateway:* Functions as the primary entry point, intelligently routing requests to internal microservices.
* *Booking Service:* Assumes responsibility for collecting user booking information, validating inputs, generating room details, handling payments, and confirming bookings.
* *Payment Service:* Acts as a dummy service, simulating payment processing and interacting with the Booking service for transaction details.

**Application Workflow:**

* Services register on the Eureka server, with the API Gateway acting as the intermediary for user requests.
* User initiates room booking through the Booking service, receiving detailed room information and pricing.
* Payment details are provided by the user, and a dummy transaction is simulated by the Payment service.
* Booking is confirmed, and a comprehensive confirmation message is printed.

**Service Configuration:**

* *Booking Service (port 8081):*
  + Endpoint 1: Responsible for collecting user information and persisting it in the database.
  + Endpoint 2: Manages payment-related details, sending them to the Payment service, and updating 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 microservices architecture detailed here provides not only modularity, scalability, and ease of maintenance but also a robust and adaptable system. Breaking down the Hotel Room Booking application into distinct services, each with its designated responsibilities, fosters reliability and efficiency. The integration of synchronous communication, meticulous error handling, and a commitment to microservices design best practices ensures a dependable and effective application architecture.