Group Members:

1.Anshul Dogra - 20csu019

2.Harshit Arora- 20csu041

3.Kartik Shukla- 20csu048

4.Manan Sethi- 20csu059

**Detailed Walkthrough of Hotel Room Booking Microservices**

* This report meticulously outlines a detailed, step-by-step walkthrough of the logic executed within each microservice constituting the Hotel Room Booking application.
* It intricately covers client interaction, microservice communication, and database enhancements, providing a comprehensive understanding of the entire operational framework.
* The emphasis on a point-by-point breakdown ensures clarity in comprehending the intricacies of each microservice, elucidating the processes involved in client engagement, seamless communication between microservices, and the crucial updates made to the underlying database infrastructure.
* By incorporating client interactivity, the report underscores the user-centric design of the application, highlighting the seamless flow of logic and information exchange. Furthermore, the detailed exploration of microservice communication sheds light on the intricate network of operations, fostering a holistic comprehension of the application's functionality.
* The thorough examination of database upgrades ensures that readers gain insights into the foundational enhancements that contribute to the application's efficiency and scalability.

**Microservice Registration:**

* To initiate seamless communication and orchestration within the Hotel Room Booking application, an initial step involves the registration of microservices on the Eureka server.
* Eureka, functioning as a service registry, plays a pivotal role in enabling dynamic service discovery. As the API Gateway, Booking service, and Payment service register themselves on the Eureka server, they become discoverable entities within the distributed system.
* This registration facilitates an organized and efficient interaction between microservices, ensuring that the API Gateway can intelligently route requests to the relevant services. The use of Eureka enhances the overall scalability and maintainability of the system, allowing for easy adaptation to changes in the microservices architecture and promoting a robust foundation for the Hotel Room Booking application.

**User Requests a Room Booking:**

* **User initiates booking:** The client enters check-in (fromDate), check-out (toDate), Aadhaar number (aadharNumber), and wanted number of rooms (numOfRooms) through the Booking service(API Gateway optional).
* **Booking service validation**: The Aadhaar number, room availability, and input dates are verified by the service. It raises the appropriate exception and notifies the user of the error if any input is found to be invalid.
* **Room generation and pricing:** If the input is legitimate, the service creates a random list of room numbers based on numOfRooms and calculates the total room price depending on the number of days between fromDate and toDate.

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

* **Payment details input:** If the user wants to confirm a booking, they furnish payment information like payment method (paymentMode), UPI ID (upiId), or card number (cardNumber).

Explanation: User confirms booking with payment details.

* **Booking confirmation:** The Booking service examines the given paymentMode and raises an exception if it's not a valid choice ("UPI" or "CARD").

Explanation: Payment mode is validated; exceptions for invalid options.

* **Synchronous communication:** When paymentMode is valid, the Booking service employs REST Template to send an asynchronous request to the Payment service, encompassing booking and payment details.

Explanation: Valid payment mode triggers asynchronous request to Payment service.

**3. Payment Service Processing:**

* **Payment information reception:** Upon receiving the request from the Booking service, the Payment service meticulously extracts crucial details encompassing both booking information and payment particulars.
* **Dummy transaction simulation:** Engaging in a simulated transaction, the service ingeniously generates a distinct transactionId, meticulously abstaining from any actual execution of payment processing.
* **Database update:** With precision, the service executes an update in its database, storing comprehensive transaction details, notably the transactionId and pertinent booking information within its designated transaction table.
* **Response to Booking service:** Swiftly responsive, the Payment service promptly communicates back to the Booking service, furnishing the distinctive transactionId as part of the comprehensive response.

**Explanation**: Payment service handles the request intricately, simulates the transaction smartly, updates the database meticulously, and responds promptly to the Booking service.

**4. Booking Confirmation and Completion:**

* **Transaction ID update:** Upon receipt of the transactionId from the Payment service, the Booking service meticulously updates its booking table, incorporating the received ID.
* **Confirmation message:** The Booking service generates and outputs a confirmation message to the console, encompassing essential details such as the user's Aadhaar number and other pertinent booking information.
* **User experience:** Enhancing user engagement, the confirmation message provides flexibility by being displayable to the user or transmitable through various channels such as email or SMS notification.

**Explanation**: Booking service manages transaction ID update efficiently, prints confirmation message with essential details, and enhances user experience through versatile notification options.

**5. Dependency Significance:**

**Booking Service Dependencies:**

* Spring Cloud Netflix Eureka Client: Enables the Booking service to register with the Eureka server, facilitating dynamic service discovery.
* Spring Boot Web: Supports the development of web applications, essential for handling user interactions and API requests.
* Spring Boot Data JPA: Provides seamless integration with databases, allowing the Booking service to interact with and store booking-related information.
* Spring Boot Dev tools: Enhances development productivity by providing tools for automatic application restarts and other development-focused features.

**Payment Service Dependencies:**

* Spring Cloud Netflix Eureka Client: Similar to the Booking service, facilitates registration with the Eureka server for efficient service discovery.
* Spring Boot Web: Essential for handling web-related functionalities within the Payment service.
* Spring Boot Data JPA: Allows seamless communication with databases, aiding in the storage and retrieval of payment transaction details.
* Spring Boot Dev tools: Boosts development efficiency by providing helpful development tools.

**API Gateway Dependencies**:

* Spring Boot Actuator: Enables monitoring and management of the application.
* Spring Cloud Netflix Eureka Client: Facilitates interaction with the Eureka server, ensuring dynamic routing of requests to the relevant microservices.
* Spring Boot Dev tools: Enhances development experience through automatic application restarts and other productivity features.
* Configure properties: Custom configuration to fine-tune the behavior of the API Gateway.

**Eureka Server Dependencies:**

* Spring Cloud Netflix Eureka Client: Essential for implementing Eureka server-client interaction, enabling microservice registration and discovery.
* Eureka Discovery Server: Provides the core functionality for the Eureka server to act as a service discovery and registry.
* Spring Boot Dev tools: Supports development by providing tools for automatic restarts and other developer-friendly features.

**Additional Points To Consider:**

* The interaction between services is intentionally synchronous, prioritizing accuracy in the booking confirmation message before it is presented to the user.
* The adherence to synchronous communication underscores the commitment to delivering precise and reliable information at every stage of the booking process. Essential to this seamless interaction are the database schemas, which define the model classes governing the functionality of both the Booking and Payment services.
* This structural foundation ensures a cohesive and standardized approach, facilitating smooth data exchange between the services. Furthermore, this walkthrough places significant emphasis on the primary logic flow, providing a comprehensive understanding of the fundamental processes involved, while concurrently addressing the necessity for robust error handling and exception implementation at each critical step.
* This meticulous attention to detail contributes to the overall reliability and resilience of the system, ensuring a robust and error-resistant booking experience for users.

**HIGH LEVEL DIAGRAM**

A diagram of a service

Description automatically generated

Endpoint 1: This endpoint is responsible for collecting information like fromDate, toDate,aadharNumber,numOfRooms from the user and save it in its database.

• URI: /booking

• HTTP METHOD: POST

• RequestBody: fromDate, toDate,aadharNumber,numOfRooms

• Response Status: Created

• Response: ResponseEntity<BookingInfoEntity>

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Endpoint 2: This endpoint is responsible for taking the payment related details from the user and sending it to the payment service. It gets the transactionId from the Payment service in respose and saves it in the booking table. Please note that for the field 'paymentMode', if the user provides any input other than 'UPI' or 'CARD', then it means that the user is not interested in the booking and wants to opt-out.

URL: booking/{bookingId}/transaction

HTTP METHOD: POST

PathVariable: int

RequestBody: paymentMode, bookingId, upiId,cardNumber

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Endpoint 1: This endpoint is used to imitate performing a transaction for a particular booking. It takes details such as bookingId, paymentMode,upiId or cardNumber and returns the transactionId automatically generated while storing the details in the ‘transaction’ table. Note that this 'transactionId' is the primary key of the record that is being stored in the 'transaction' table. After receiving the transactionId from 'Payment' service, confirmation message is printed on the console. Message String: String message = "Booking confirmed for user with aadhaar number: " + bookingInfo.getAadharNumber() + " | " + "Here are the booking details: " + bookingInfo.toString();

A screenshot of a computer

Description automatically generated

EndPoint 2: This endpoint presents the transaction details to the user based on the transactionId provided by the user.

• URL: /transaction/{transactionId}

• HTTP METHOD: GET

• RequestBody: (PathVariable) int

• Response Status: OK

• Response: ResponseEntity<TransactionDetailsEntity>

A screenshot of a computer

Description automatically generated

Service Discovery- EUREKA

A screenshot of a computer

Description automatically generated