PAYMENT HANDLING MICROSERVICE DOCUMENTATION

Introduction

The Payment Handling microservice is a crucial component of the Grab Your Tickets platform, responsible for managing payment transactions related to user bookings. This documentation serves as a comprehensive guide for setting up, integrating dummy payment functionality, and connecting the Payment Handling microservice with the existing Grab Your Tickets project.

Table of Contents

---

1. Overview

The Payment Handling microservice plays a vital role in ensuring smooth and secure payment processing for bookings made by users on the Grab Your Tickets platform. It enhances user experience by providing reliable payment transactions.

---

2. Technologies Used

The Payment Handling microservice utilizes the following technologies:

- Java Spring Boot: Provides a robust framework for building microservices.
- Spring Web MVC: Facilitates the development of RESTful APIs.
- MySQL: Stores payment records efficiently.
- JSON: Standard format for data interchange.

---

3. Setting Up Payment Handling Microservice

3.1 Project Structure

- Create a new Spring Boot project for the Payment Handling microservice.
- Implement controllers, services, repositories, and models for handling payment transactions.
- Configure database connection and entity mappings for storing payment records.
- Implement necessary security measures for authentication and authorization.

3.2 Dependencies

Include necessary dependencies in the `pom.xml`, such as Spring Web MVC, Spring Data JPA, MySQL driver, etc.

```xml
<!-- Spring Boot Starter Web -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
```

</dependency>

<!-- Spring Boot Starter Data JPA -->

<dependency>

  <groupId>org.springframework.boot</groupId>

  <artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- MySQL Connector -->

<dependency>

  <groupId>mysql</groupId>

  <artifactId>mysql-connector-java</artifactId>

</dependency>

```

---

4. Integrating Dummy Payment Functionality

4.1 Payment Controller

Create a controller to handle payment-related endpoints. The controller calculates the payment amount based on the ticket price and the number of seats selected by the user.

```java

@RestController

@RequestMapping("/api/payments")

public class PaymentController {

  @Autowired

  private PaymentService paymentService;

```
    @PostMapping("/process")
    public ResponseEntity<String> processPayment(@RequestBody PaymentRequest paymentRequest) {
        // Calculate payment amount based on the ticket price and number of seats
        double paymentAmount = paymentRequest.getTicketPrice() * paymentRequest.getNumSeats();

        // Process payment
        boolean paymentSuccess = paymentService.processPayment(paymentAmount);

        if (paymentSuccess) {
            // Payment successful
            return ResponseEntity.ok("Payment processed successfully. Amount: $" + paymentAmount);
        } else {
            // Payment failed
            return ResponseEntity.badRequest().body("Payment failed. Please try again.");
        }
    }
}
```

4.2 Payment Request Model

Define a model class to represent the payment request data. The model includes fields for the ticket price and the number of seats.

```java
public class PaymentRequest {
    private double ticketPrice;
    private int numSeats;

    // Getters and setters
```

}

```

---

5. Integrating with Grab Your Tickets

Certainly! Let's fill up section 5.1 with detailed documentation and provide the complete documentation again:

5.1 RESTful API

The Payment Handling microservice exposes RESTful endpoints to facilitate communication with the Grab Your Tickets application. These endpoints handle various aspects of payment processing, including initiating payment transactions, handling callbacks, and retrieving payment status.

5.1.1 Initiating Payment Transaction

Endpoint: /api/payments/process

Method: POST

Description: Initiates a payment transaction based on the provided payment request data.

Request Body:

json

Copy code

{

"ticketPrice": 10.5,

"numSeats": 2

}

ticketPrice: The price of each ticket for the booking.

numSeats: The number of seats booked by the user.

Response:
```

Successful Response (HTTP 200 OK):

json

Copy code

```
{
  "message": "Payment processed successfully. Amount: $21.0"
}
```

Error Response (HTTP 400 Bad Request):

json

Copy code

```
{
  "error": "Payment failed. Please try again."
}
```

5.1.2 Handling Payment Callbacks

Endpoint: /api/payments/callback

Method: POST

Description: Handles payment callbacks from external payment gateways.

Request Body:

json

Copy code

```
{
  "paymentId": "123456789",
  "status": "success"
}
```

paymentId: The unique identifier of the payment transaction.

status: The status of the payment transaction (success/failure).

Response: No response required.

5.1.3 Retrieving Payment Status

Endpoint: /api/payments/{paymentId}

Method: GET

Description: Retrieves the status of a payment transaction by its ID.

Path Parameter:

paymentId: The unique identifier of the payment transaction.

Response:

Successful Response (HTTP 200 OK):

json

Copy code

```
{
  "paymentId": "123456789",
  "status": "success",
  "amount": 21.0
}
```

Error Response (HTTP 404 Not Found):

json

Copy code

```
{
  "error": "Payment transaction not found."
}
```

5.2 Secure Communication

Implement secure communication between the Grab Your Tickets application and the Payment Handling microservice using HTTPS. Utilize authentication tokens or API keys for authentication and authorization.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
```

```
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/api/payments/").authenticated()
            .and()
            .oauth2ResourceServer().jwt();
    }
}
```

---

6. Payment Processing Flow

6.1 Initiating Payment

- When a user completes a booking on the Grab Your Tickets platform, the application sends a request to the Payment Handling microservice to initiate payment processing.

- The Payment Handling microservice calculates the payment amount based on the ticket price set by the admin and the number of seats selected by the user.

6.2 Payment Result

- Upon processing the payment, the Payment Handling microservice sends a response indicating the success or failure of the payment transaction to the Grab Your Tickets application.

---

7. Error Handling

Handle errors gracefully within the Payment Handling microservice. Implement error handling mechanisms to handle validation errors, payment processing errors, and communication errors effectively.

---

8. Testing

Test the payment functionality by sending requests to the `/api/payments/process` endpoint with appropriate payment data. Verify that the Payment Handling microservice calculates the payment amount correctly and returns appropriate responses.

---

9. Documentation

Update the documentation of the Payment Handling microservice to include details about the payment amount calculation. Document the updated endpoint, request/response structures, and usage instructions for initiating payment transactions. Provide examples and explanations to guide developers on how to integrate payment functionality into the Grab Your Tickets project.
