### \*\*Chapter 4: Cashfree API Integration\*\*

In this chapter, we will integrate the Cashfree Payment Gateway into your Spring Boot application by making API calls to Cashfree's endpoints. This involves setting up the payment process, handling responses, and integrating webhooks for real-time updates.

---

#### \*\*Topic 1: Overview of Cashfree API Documentation\*\*

The Cashfree API is designed for easy integration with multiple payment methods. You can find their official API documentation [here](https://dev.cashfree.com/).

\*\*Key API Endpoints:\*\*

1. \*\*Order Creation:\*\* `/api/v2/cftoken/order`

- This API call generates a unique token for each payment order, which is required to initialize the payment.

2. \*\*Order Status Check:\*\* `/api/v2/order/info/status`

- Use this to verify if a payment was successful or failed.

3. \*\*Refund API:\*\* `/api/v2/refund/create`

- Initiates refunds for a specific transaction.

---

#### \*\*Topic 2: Integrating Cashfree Payment Gateway with Spring Boot\*\*

Let’s start by integrating the \*\*Order Creation API\*\*. This will generate a payment token, which is essential for processing the payment.

---

##### \*\*Step 1: Create a Payment Request\*\*

You’ll first need to create an endpoint in your Spring Boot application to initiate a payment. The frontend will call this API to generate a payment token.

1. \*\*Model for Payment Request:\*\*

- Create a model to represent the payment request.

```java

public class PaymentRequest {

private String orderId;

private String orderAmount;

private String orderCurrency;

private String customerEmail;

private String customerPhone;

// Getters and setters

}

```

2. \*\*Service for Payment Integration:\*\*

- This service will call Cashfree’s order token API to create a payment request.

```java

@Service

public class PaymentService {

private final CashfreeConfig cashfreeConfig;

private final RestTemplate restTemplate;

@Autowired

public PaymentService(CashfreeConfig cashfreeConfig, RestTemplate restTemplate) {

this.cashfreeConfig = cashfreeConfig;

this.restTemplate = restTemplate;

}

public String generatePaymentToken(PaymentRequest paymentRequest) throws Exception {

// Construct request payload

Map<String, String> requestBody = new HashMap<>();

requestBody.put("orderId", paymentRequest.getOrderId());

requestBody.put("orderAmount", paymentRequest.getOrderAmount());

requestBody.put("orderCurrency", paymentRequest.getOrderCurrency());

requestBody.put("customerEmail", paymentRequest.getCustomerEmail());

requestBody.put("customerPhone", paymentRequest.getCustomerPhone());

// Prepare headers

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.APPLICATION\_JSON);

headers.set("x-client-id", cashfreeConfig.getAppId());

headers.set("x-client-secret", cashfreeConfig.getSecretKey());

HttpEntity<Map<String, String>> request = new HttpEntity<>(requestBody, headers);

// Send POST request to Cashfree API

String url = cashfreeConfig.getBaseUrl() + "/cftoken/order";

ResponseEntity<Map> response = restTemplate.postForEntity(url, request, Map.class);

if (response.getStatusCode() == HttpStatus.OK) {

// Extract token from response

Map<String, String> responseBody = response.getBody();

return responseBody.get("cftoken");

} else {

throw new Exception("Failed to create payment token");

}

}

}

```

---

##### \*\*Step 2: Create a REST Controller for Payment API\*\*

Now that we have the service to generate the payment token, let's create a REST API that your frontend can call to initiate the payment process.

```java

@RestController

@RequestMapping("/api/payment")

public class PaymentController {

private final PaymentService paymentService;

@Autowired

public PaymentController(PaymentService paymentService) {

this.paymentService = paymentService;

}

@PostMapping("/create")

public ResponseEntity<String> createPayment(@RequestBody PaymentRequest paymentRequest) {

try {

String paymentToken = paymentService.generatePaymentToken(paymentRequest);

return ResponseEntity.ok(paymentToken);

} catch (Exception e) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Failed to create payment");

}

}

}

```

This API will be called by the frontend to initiate the payment. It generates the \*\*payment token\*\*, which will be used by the frontend to complete the payment process on Cashfree’s hosted payment page.

---

##### \*\*Step 3: Example API Call from Frontend\*\*

The frontend will need to call the API to create the payment token. Here’s an example of how the request would look from a frontend (using Fetch API):

```javascript

fetch('http://localhost:8080/api/payment/create', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

orderId: 'ORDER12345',

orderAmount: '500',

orderCurrency: 'INR',

customerEmail: 'customer@example.com',

customerPhone: '9876543210'

})

})

.then(response => response.json())

.then(data => {

console.log('Payment Token:', data);

// Redirect to Cashfree payment page with token

})

.catch(error => {

console.error('Error:', error);

});

```

---

#### \*\*Topic 3: Cashfree Webhook Integration\*\*

Cashfree provides webhooks to notify your server when a payment succeeds or fails. Let’s set up a webhook endpoint in Spring Boot to listen for these notifications.

1. \*\*Create a Webhook Controller:\*\*

- This controller will handle incoming webhook notifications from Cashfree.

```java

@RestController

@RequestMapping("/api/payment")

public class WebhookController {

@PostMapping("/webhook")

public ResponseEntity<String> handleWebhook(@RequestBody Map<String, String> payload) {

// Validate and process webhook payload

String status = payload.get("txStatus");

if ("SUCCESS".equalsIgnoreCase(status)) {

// Handle successful payment

return ResponseEntity.ok("Payment success");

} else {

// Handle failed payment

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Payment failed");

}

}

}

```

2. \*\*Configuring Webhook in Cashfree Dashboard:\*\*

- Log in to your Cashfree dashboard.

- Under the "Developers" section, go to \*\*Webhook Settings\*\*.

- Add your webhook URL (e.g., `https://yourdomain.com/api/payment/webhook`).

3. \*\*Webhook Payload Example:\*\*

- When a payment is successful, Cashfree will send a payload like this to your webhook:

```json

{

"orderId": "ORDER12345",

"txStatus": "SUCCESS",

"paymentMode": "UPI",

"orderAmount": "500",

"txTime": "2023-01-01T12:34:56"

}

```

---

#### \*\*What We’ve Covered So Far:\*\*

1. \*\*Created an API to generate a payment token\*\* using Cashfree’s `/cftoken/order` endpoint.

2. \*\*Set up a REST controller\*\* to handle payment requests.

3. \*\*Webhook Integration:\*\* Implemented a webhook endpoint to listen for payment status updates from Cashfree.

---

In the comment area `// Handle successful payment`, you would typically write code to update your system based on the successful payment. This could involve various actions, depending on the requirements of your application. Here's a breakdown of what you might implement there:

### 1. \*\*Update Payment Status in Database\*\*

- You might have a table in your database that tracks orders or payments. In this section, you would update the order status to indicate that the payment was successful.

```java

String orderId = payload.get("orderId");

String paymentStatus = "SUCCESS";

// Assume you have a service to update order/payment status

orderService.updatePaymentStatus(orderId, paymentStatus);

```

### 2. \*\*Send Confirmation Email\*\*

- Upon successful payment, you may want to send an email confirmation to the customer, acknowledging that the payment was processed.

```java

String customerEmail = payload.get("customerEmail");

emailService.sendPaymentConfirmation(customerEmail, orderId);

```

### 3. \*\*Generate Invoice or Receipt\*\*

- You might generate an invoice or receipt after successful payment. This could involve creating a PDF document or generating a receipt in your database.

```java

receiptService.generateReceipt(orderId, payload);

```

### 4. \*\*Update Inventory (for E-commerce)\*\*

- If this is an e-commerce system, you could reduce the inventory count of the purchased items.

```java

inventoryService.reduceStock(orderId);

```

### 5. \*\*Trigger Further Business Logic\*\*

- Depending on the type of application, you may want to trigger other business logic, like enabling access to a service, delivering a digital product, or starting a subscription.

```java

subscriptionService.activateSubscription(customerId);

```

---

### Example Implementation:

```java

@PostMapping("/webhook")

public ResponseEntity<String> handleWebhook(@RequestBody Map<String, String> payload) {

String status = payload.get("txStatus");

String orderId = payload.get("orderId");

if ("SUCCESS".equalsIgnoreCase(status)) {

// Update payment status in the database

orderService.updatePaymentStatus(orderId, "SUCCESS");

// Send payment confirmation email

String customerEmail = payload.get("customerEmail");

emailService.sendPaymentConfirmation(customerEmail, orderId);

// Generate receipt

receiptService.generateReceipt(orderId, payload);

return ResponseEntity.ok("Payment success");

} else {

// Handle failed payment (you can update status, send failure notifications, etc.)

orderService.updatePaymentStatus(orderId, "FAILED");

return ResponseEntity.status(HttpStatus.BAD\_REQUEST).body("Payment failed");

}

}

```

In this way, you're handling various actions upon receiving a successful payment notification from Cashfree.