Integration Guide

Table of Contents

1. Document's Purpose

The purpose of this document is to propose the solution against the Client's requirement to ensure we at HDFC Collect Now have clearly understood the requirements from Client and are going to deliver the solution accordingly. This is required to ensure that Client's expectations, scope definitions are addressed clearly.

2. Overview

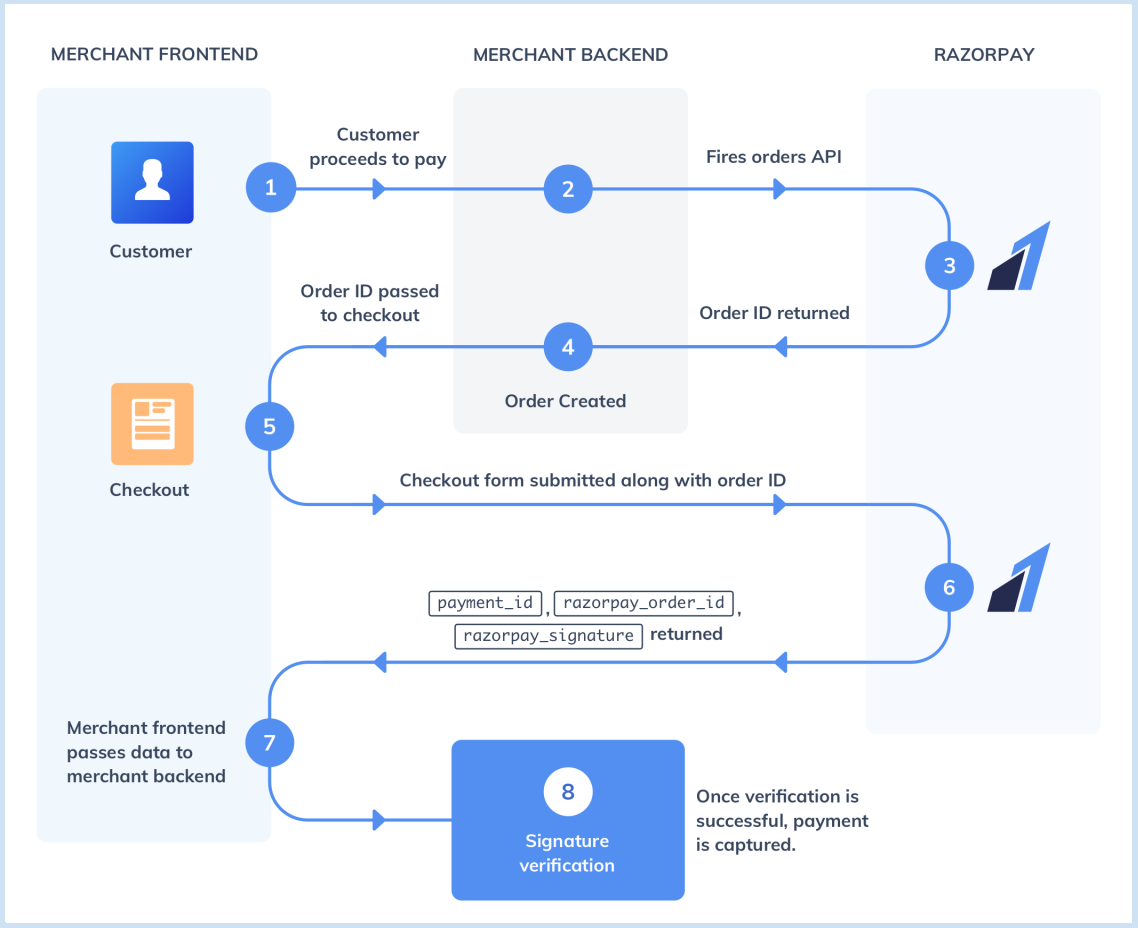
Before we dive into the integration, it is recommended that you understand how Razorpay Payments work to integrate faster.

2.1. Payments Flow

The process of accepting payments from your customers is as follows:

1. Razorpay creates a unique **order_id** based on the order created by the customer at the merchant's website.
2. The customer proceeds to make the payment for the order amount via **Razorpay Checkout**.
3. The **Razorpay Checkout** collects the customer's payment details and relays it to the bank for authorization.
4. On a successful authorization, Razorpay returns a Payment ID, Order ID, and a Razorpay Signature in the response. This response objects are consumed by the merchant's server for *capturing* the payment.
5. Once the payment is successfully captured, funds are settled into the merchant's account in *T+X* day ('T' being the day of payment creation and 'X' being the number of days as per merchant's settlement schedule).

The following



g illustration will help you picture the entire payment flow:

2.2. Payment Status

In its entire, a `payment_id` moves through following statuses based on the processing done at that stage.

`created`	Customer's payment details are submitted to Razorpay. No processing has been done on the payment at this stage.
`authorized`	The customer's payment details are successfully authenticated by the bank. At this stage, funds are debited from the customer's account. However, it is not transferred to the merchant's account.
`captured`	Merchant has verified the purchase as complete and funds have been transferred to the merchant's account in T+X day ('T' being the day of payment creation and 'X' being number of days as per merchant's settlement schedule).
`refunded`	A payment that has been successfully captured can be refunded to customer. Any authorized payment not followed by a capture within 5 days is automatically refunded.

3. Integration

Our set of RESTful APIs lets you communicate with Razorpay APIs in order to build a fully functional payments collection interface tailored for your merchants.

All API request are sent to
`https://<key-id>:<key-secret>@api.razorpay.com`

For example,
`https://rzp_test_Ad93zqidD9eZwy:69b2e24411e384f91213f22a@api.razorpay.com/v1/payments`

To start accepting payments using Razorpay, you need API keys that can be generated through your Razorpay Dashboard

***NOTE**: API keys should not be shared with anyone and should be used while making the client-side requests to the Razorpay's servers.*

To integrate Razorpay Payment Gateway on a merchant involves the following major steps:

1. Create an Razorpay Order
   This will generate an order Id. The payment will be made against this order_id.

2. Implement Razorpay Checkout.js
   This involves building an interface to collect the customer's payment details on the merchant's site. With our ready-to-use libraries, the required integration effort is minimal.

3. Capture the Payment
   Once the payment is authorized by the bank, the merchant can capture the payment either manually or automatically.

3.1. Razorpay Order

Whenever an order is created on the merchant's site Razorpay generates a unique **order_id** which has a 1:1 mapping with the order created at merchant's end. Razorpay Orders is the primary way to manage payments against an order and it can only be created via Orders API. However, the orders that are created can be viewed on the Dashboard. Order creation is not a mandatory steps, however we recommend to use Orders Api for security purpose and below are some of the advantages of implementation.

3.1.1. Advantages of Orders API

1. Restricts successful payments to 1 per order. This prevents multiple successful payments against the same order.
2. Allows you to automatically capture payments thereby reducing the number of requests required to complete a payment. This is explained in detail in the later sections.
3. Allows you to combine multiple payment attempts corresponding to one single order, making it easier to query Razorpay systems for an order.

3.1.2. Order Life Cycle

Similar to the payment entity, a Razorpay Order also moves through different states in its entire lifetime.

Status	Description
`created`	Indicates that the order has been created.
`attempted`	Indicates that the payment against this order has been attempted but not successful. The `attempts` field in the response will specify the number of attempts made.

`paid`	Indicates that the payment made against an order has been successfully captured.

3.1.3. API Reference

The following sections explain the various API request related to order with an example which will help you understand it quicker.

3.1.3.1. Create an Order

The following endpoint will create a Razorpay Order.

API Endpoint	HTTP Verb
`/orders`	POST

Attribute	Type	Description
`id`	string	The unique entity identifier. In this case, order_id has the following format: order_<14 digit hexadecimal string>
`amount`	integer	The order amount (in Paise) that the customer is charged with.
`currency`	string	The currency associated with this order's amount
`attempts`	integer	The number of payment attempts that have been made against this order
`status`	string	This status of the order as per the Order life cycle
`receipt`	string	Unique order identifier at merchant's end. Maximum character length is 40
`payment_capture`	boolean	Flag for automatically capturing the payment. `1` is to enable, `0` is to disable Learn more about capturing payments.
`created_at`	timestamp	The timestamp corresponding to order creation

notes	object	Object consisting of notes passed while creating an order entity

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X POST \
   --data "amount=50000" \
   --data "currency=INR" \
   --data "receipt=transaction_1234" \
   --data "payment_capture=1" \
https://api.razorpay.com/v1/orders
```

Response

```
{
  "id": "order_4xbQrmEoA5WJ0G",
  "entity": "order",
  "amount": 50000,
  "currency": "INR",
  "receipt": "transaction_1234",
  "status": "created",
  "attempts": 0,
  "created_at": 1455696638,
  "notes": {}
}
```

3.1.3.2. Fetch Single Order

Given an order id, Razorpay GET Order API can be used to fetch the order details.

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X GET \
   https://api.razorpay.com/v1/orders/order_7IZKKI4Pnt2kEe
```

Response

```
{
    "id": "order_7IZKKI4Pnt2kEe",
    "entity": "order",
    "amount": 50000,
    "currency": "INR",
    "receipt": "rcptid33",
    "status": "created",
    "attempts": 0,
    "notes": [],
    "created_at": 1455696913
}
```

3.1.3.3. Fetch Multiple Orders

Orders can also be fetched in bulk, using this API with support for `from`, `to`, `count` and `skip`.

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X GET \
   https://api.razorpay.com/v1/orders/
```

Response


```
{
  "entity": "collection",
  "count": 2,
  "items": [
    {
      "id": "order_7IZKKI4Pnt2kEe",
      "entity": "order",
      "amount": 50000,
      "currency": "INR",
      "receipt": "rcptid42",
      "status": "created",
      "attempts": 0,
      "notes": [],
      "created_at": 1487348538
    },
    {
      "id": "order_4xbSwsPABDJ8oK",
      "entity": "order",
      "amount": 50000,
      "currency": "INR",
      "receipt": "rcptid43",
      "status": "paid",
      "attempts": 1,
      "created_at": 1455696756,
      "notes": {}
    }
  ]
}
```


Refer the complete documentation for Orders API.

3.1.4. Supported Webhooks

Following are the list of webhooks can be utilized to get notified about the events on Razorpay orders.

order.paid	Triggered whenever a payment against a particular order gets paid.

Example Webhook Payload


```
{
  "entity": "event",
  "event": "order.paid",
  "contains": [
    "payment",
    "order"
  ],
  "payload": {
    "payment": {
      "entity": {
        "id": "pay_8B33XWu170gVbv",
        "entity": "payment",
        "amount": 29935,
        "currency": "INR",
        "status": "captured",
        "order_id": "order_9A33XWu170gUtm",
        "invoice_id": null,
        "international": false,
        "method": "upi",
        "amount_refunded": 0,
        "refund_status": null,
        "captured": true,
        "description": null,
        "card_id": null,
        "bank": null,
        "wallet": null,
        "vpa": "gauravkumar@okexmpl",
        "email": "gaurav.kumar@example.com",
        "contact": "+919123456780",
        "notes": {
          "loginID": "1234567890",
```

```
          "lobID": "1",
          "txnID": "123456789012"
        },
        "fee": 598,
        "tax": 3592,
        "error_code": null,
        "error_description": null,
        "created_at": 1536137632
},
    "order": {
      "entity": {
        "id": "order_9A33XWu170gUtm",
        "entity": "order",
        "amount": 29935,
        "amount_paid": 29935,
        "amount_due": 0,
        "currency": "INR",
        "receipt": "123456789012",
        "offer_id": null,
        "status": "paid",
        "attempts": 1,
        "notes": [],
        "created_at": 1536137630
      }
    }
  },
  "created_at": 1536137660
}
```


Learn how to setup a Razorpay webhook

3.2. Razorpay Checkout

Razorpay checkout library integration provides merchants with all the necessary tools in order to build and customize a fully-functional payment collection system on their site.

Razorpay raw form POST API enables you to directly create payments. To use this, you need to post data on the following URL: https://api.razorpay.com/v1/checkout/embedded

For mobile integration, you'll need to call SDK method with same parameters:

3.2.1. Checkout Parameters

Following are the parameters required for collecting payments from the customer

`key_id`	Your authentication key Id. You can access your authentication keys from the Razorpay Dashboard
`amount`	Payment amount (in Paise)
`callback_url`	The URL to which the payment (POST) response will be sent
`name`	Customer Name
`description`	Descript
`prefill[email]`	Customer's email address
`prefill[contact]`	Customer's contact number
`order_id`	Unique order_id associated with this payment.
`notes[]`	Custom notes for the customer For example, `notes[abc]=123` in form POST like `<input name="notes[abc]" value="123">`

3.2.2. Example Checkout Script

The following checkout script can be executed in a merchant's site to collect from the customer.

```
<form method="POST" action="https://api.razorpay.com/v1/checkout/embedded">
<input type="hidden" name="key_id" value="API_KEY">
<input type="hidden" name="order_id" value="razorpay_order_id">
<input type="hidden" name="amount" value="500 in paisa">
</form>

<input type="hidden" name="name" value="HDFC Collect Now">
<input type="hidden" name="description" value="Enter description">
<input
```

```
 type=”hidden” name=”prefill[email]” value=”gaurav.kumar@example.com”>
<input type=”hidden” name=”prefill[contact]” value=”9999999999”>
<input type=”hidden” name=”notes[transaction_id]” value=”transaction_1234”>
<input type=”hidden” name=”callback_url” value=”YOUR_CALLBACK_URL”>
<button>Submit</button>
```

Android:

3.3. Installation

Add the following lines of code to your app's `build.gradle` file:

```
dependencies {
  implementation "com.razorpay:customui:3.9.8"
}
```

This will add the dependencies for the SDK.

3.4. Initialization

You can add your Razorpay API key to **`AndroidManifest.xml`**.

The sample **`AndroidManifest.xml`** file is shown below:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application>
    <meta-data
        android:name="com.razorpay.ApiKey"
        android:value="YOUR_API_KEY"
        />
  </application>
</manifest>
```

3.5. Override min SDK version

The SDK min version is 19, so the app should have minSDK version 19 or higher. If you want to support the devices below API level 19, then you may override minSDK version. See below code for the example:

```
<uses-sdk android:targetSdkVersion="27" android:minSdkVersion="14"
            tools:overrideLibrary="com.razorpay"/>
```

3.5.1. Proguard Rules

If you are using proguard for your builds, you need to add the following lines to your proguard file:

Java

```
-keepclassmembers class * {
    @android.webkit.JavascriptInterface <methods>;
}

-keepattributes JavascriptInterface
-keepattributes *Annotation*

-dontwarn com.razorpay.**
-keep class com.razorpay.** {*;}
```

3.6. Instantiating Razorpay

To instantiate Razorpay, you need to pass a reference of your activity to the `Razorpay` constructor, as shown below:

```
import com.razorpay.Razorpay


Razorpay razorpay = new Razorpay(activity);
```

If you want to set your API key at runtime instead of statically defining it in your AndroidManifest.xml, you can pass it as a parameter to the Razorpay constructor, as shown below:

```
Razorpay razorpay = new Razorpay(activity, "YOUR_API_KEY");
```

3.7. Setting up the Webview

The bank ACS pages will be displayed to the user in a webview. You need to define a webview in your layout and pass the reference to our SDK using **`setWebView`**, as shown below:

```
webview = findViewById(R.id.payment_webview);
// Hide the webview until the payment details are submitted
webview.setVisibility(View.GONE);
razorpay.setWebView(webview);
```

3.8. Submitting Payment Data

Once you have received the customer's payment information, it needs to be sent to Razorpay's API to complete the "creation" step of the payment flow. You can do this by invoking **`openCheckout`** method. Before invoking this method, you have to make your webview visible to the customer. The data that needs to be submitted in the form of a **`JSONObject`**, as shown below:

```
JSONObject  payload = new JSONObject();
payload.put("order_id",  "razorpay_order_id"); //Created using Razorpay Orders API
payload.put("amount",  "500"); //Amount in Paisa. 500 = Rs 5
payload.put("name",  "HDFC Collect Now");
payload.put("description",  "Enter description");
payload.put("prefill[email]",  "gaurav.kumar@example.com");
payload.put("prefill[contact]",  "9999999999");
payload.put("notes[transaction_id]",  "transaction_1234");

// Make webview visible before submitting payment details
```

```
razorpay.openCheckout(payload, new PaymentResultListener() {
                @Override
                public void onPaymentSuccess(String razorpayPaymentId) {
              // Razorpay payment ID is passed here after a successful payment
                }

                @Override
                public void onPaymentError(int code, String description) {
                        // Error code and description is passed here
                }
        });

} catch (Exception e) {
        Log.e(TAG, "Error in submitting payment details", e);
}
}
```

3.9. Handling onBackPressed

The SDK needs to be notified if the user presses the Back button. Through this method, if the back button is pressed before the payment is complete, the payment is marked as **Failed** on the Merchant Dashboard.

```
/**
 * In your activity, you need to override onBackPressed
 */
@Override
public void onBackPressed() {
        if(razorpay != null){
                razorpay.onBackPressed();
        }
        super.onBackPressed();
}
```

iOS:

Integration Steps :-

Note:

1. We have moved our SDKs to Swift framework. This framework only supports iOS version **10.0 and later**.
2. We no longer support Swift 3, moving forward only the latest version of Swift will be supported.

Following is the latest version of our framework:

- Razorpay - 1.0.26

SDK	Framework	Framework Compiled With	Download Link
Razorpay(with bit code enabled)	1.0.26		Click Here

Step 1: Importing the Library

For cocoapod users, our cocoapod page is [CocoaPod] (bitcode enabled).

Follow the instruction given below to import the SDK library to your Swift or Objective-C project:

For Swift:

1. Download the SDK from above and unzip it.
2. Open your project in XCode and go to **file** under **Menu**. Select **Add files to "yourproject"**.
3. Select **Razorpay.framework** in the directory you just unzipped.
4. Select the **Copy items if needed** check-box.
5. Click **Add**.
6. Navigate to **Target settings > General** and add the **Razorpay.framework** in both **Embedded Binaries** and **Linked Frameworks and Libraries**.

For Objective-C:

1. Download the SDK from above and unzip it.
2. Open your project in XCode and go to **file** under **Menu**. Select **Add files to "yourproject"**.

3. Select **Razorpay.framework** in the directory you just unzipped.
4. Select the **Copy items if needed** check-box.
5. Click **Add**.
6. Navigate to **Target settings > General** and add the **Razorpay.framework** in both **Embedded Binaries** and **Linked Frameworks and Libraries**.
7. Go to **Project Settings**..
8. Select **Build Settings - All and Combined** from the Menu of the project settings.
9. Under **Build Options** set **Always Embed Swift Standard Libraries** option to **TRUE**.

Note:

For both iOS and Objective C projects - Make sure the framework is added in both the **Embedded Binaries** and **Linked Frameworks and Libraries** under **Target settings - General**.

Step 2: Initialize the Razorpay SDK

To initialize the Razorpay SDK, you will need the following:

- API key.
- A delegate that implements `RazorpayPaymentCompletionProtocol` or `RazorpayPaymentCompletionProtocolWithData`.

Note:

The following snippets are first written in Swift and then in OBJC

```
import Razorpay

class ViewController: UIViewController,
RazorpayPaymentCompletionProtocolWithData {

      var razorpay: Razorpay!
      .
      .
      override func viewDidLoad() {
        super.viewDidLoad()
        .
        .
        razorpay = Razorpay.initWithKey(razorpayTestKey, andDelegate: self)
    }
    .
    .
}
```

```
#import <Razorpay/Razorpay.h>

@interface ViewController () <RazorpayPaymentCompletionProtocolWithData> {
  Razorpay *razorpay;
      .
      .
      - (void)viewDidLoad {
            [super viewDidLoad];
            .
            .
            razorpay = [Razorpay initWithKey:@"YOUR_PUBLIC_KEY"
andDelegate:self];
      }
}
```

Step 3: Pass Payment Options and Display Checkout Form

Add the following code to your `ViewController` or where ever you want to initialize payments:

```
internal func showPaymentForm(){
      let options: [String:Any] = [
                        "amount" : "100" //mandatory in paise
                        "order_id" : "razorpay_order_id" //Created using
Razorpay Orders API
                        "name": "business or product name"
                        "description": "purchase description"
                        "image": "https://url-to-image.png",
                        "prefill": [
                        "contact": "9797979797",
                        "email": "foo@bar.com"
                        ],
                        "theme": [
                        "color": "#F37254"
                        ]
            ]
```

```
      razorpay.open(options)
}
```

```
- (void)showPaymentForm { // called by your app
  NSDictionary *options = @{
                            @"amount": @"1000", // mandatory, in paise
1000 = Rs 10
                            @"order_id": @"razorpay_order_id",// Created
using Razorpay Orders API
                            @"image": @"https://url-to-image.png",
                            @"name": @"business or product name",
                            @"description": @"purchase description",
                            @"prefill" : @{
                                @"email": @"foo@bar.com",
                                @"contact": @"9797979797"
                            },
                            @"theme": @{
                                @"color": @"#F37254"
                            }
                        };
```

NOTE:

We now support another optional parameter to the open method - displayController
When this parameter is specified , the razorpay controller will be pushed on to this controller's navigation controller if present or presented on this controller if absent.

For example the swift call is

```
razorpay.open(options, displayController: self)
```

You can find the list of all supported options [here].

Note:

Progress Bar: To support theme color in progress bar, please pass HEX color values only.

Step 4: Handle Success and Errors

You can handle success/errors when payment is completed by implementing `onPaymentSuccess` and `onPaymentError` methods of the `RazorpayPaymentCompletionProtocol`. Alternatively, you can also implement `onPaymentSuccess` and `onPaymentError` methods of `RazorpayPaymentCompletionProtocolWithData`.

```
public func onPaymentError(_ code: Int32, description str: String){
    let alertController = UIAlertController(title: "FAILURE", message: str,
preferredStyle: UIAlertControllerStyle.alert)
    let cancelAction = UIAlertAction(title: "OK", style:
UIAlertActionStyle.cancel, handler: nil)
    alertController.addAction(cancelAction)
    self.view.window?.rootViewController?.present(alertController,
animated: true, completion: nil)
}

public func onPaymentSuccess(_ payment_id: String){
    let alertController = UIAlertController(title: "SUCCESS", message:
"Payment Id \(payment_id)", preferredStyle: UIAlertControllerStyle.alert)
    let cancelAction = UIAlertAction(title: "OK", style:
UIAlertActionStyle.cancel, handler: nil)
    alertController.addAction(cancelAction)
    self.view.window?.rootViewController?.present(alertController,
animated: true, completion: nil)
}
```

```
- (void)onPaymentSuccess:(nonnull NSString*)payment_id {
      [[[UIAlertView alloc] initWithTitle:@"Payment Successful"
message:payment_id delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil] show];
}

- (void)onPaymentError:(int)code description:(nonnull NSString *)str {
      [[[UIAlertView alloc] initWithTitle:@"Error" message:str
delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil] show];
}
```

Here you have to put necessary actions after payment is completed based on success/error.

Possible values for failure code are:
- 0: Network error
- 1: Initialization failure / Unexpected behaviour
- 2: Payment cancelled by user

Success handler will receive `payment_id` which you can use later for capturing purposes.

iOS has higher requirements for secure URLs. As many Indian banks do not comply with the requirements, you can implement the following in your info.plist as a workaround:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

Add the above to your `info.plist` file. For more information click [here].

Step 5 : Collect Now

Now that you have completed your integration , to use the Collect Now feature just change the payload that you pass with the one or more of following parameters as required.

```
let dictOptions = [
"amount":"100",
"name":"abcd",
"description":"test description",
"prefill[email]":"abc@cde.com",
"prefill[contact]":"1234567890",
"notes[transaction_id]":"this is a test note",
"order_id":"(Razorpay_Order_ID)"
]

razorpay.openCheckout(dictOptions)
```

3.9.1. Response

In case of successful payment, Razorpay Checkout will return the following response objects which you need to store to perform further actions:

`razorpay_order_id`	The `order_id` generated at the time of order creation.
`razorpay_payment_id`	The `payment_id` returned by the Razorpay's API when the payment is successful.
`razorpay_signature`	Used for verifying the payment. Refer the Payment verification step.

Once payment is complete, we will redirect to the given `callback_url`, which will receive the result as POST parameter.

For example callback_url: https://your-site.com/callback?order_id=ordr_12346

- s

Learn more about of Error Response3.9.2. API Reference

Razorpay provides APIs to the merchant where they can check the payment details. Below mentioned are the various API details:

3.9.2.1. Fetch Single Payment

Given a payment id, Razorpay GET Payment API can be used to fetch the transaction details.

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \ -X GET \
https://api.razorpay.com/v1/payments/pay_29QQoUBi66xm2f
```

Response

```
{
```

```
    "id": "pay_29QQoUBi66xm2f",
    "entity": "payment",
    "amount": 50000,
    "currency": "INR",
    "status": "captured",
    "order_id": null,
    "invoice_id": null,
    "international": false,
    "method": "wallet",
    "amount_refunded": 0,
    "refund_status": null,
    "captured": true,
    "description": "Purchase Description",
    "card_id": null,
    "bank": null,
    "wallet": "freecharge",
    "vpa": null,
    "email": "gaurav.kumar@example.com",
    "contact": "9123456780",
    "notes": {
        "merchant_order_id": "order id"
    },
    "fee": 1438,
    "tax": 188,
    "error_code": null,
    "error_description": null,
    "created_at": 1400826750
}
```


3.9.2.2. Fetch Multiple Payments

Payments can also be fetched in bulk. The response is paginated, with support for from, to, count and skip parameters. Payments fetched can further be filtered by email, contact and user-defined notes using the relevant query parameters as shown below:

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \ -X GET \
https://api.razorpay.com/v1/payments?count=50&skip=10
```

Response


```
{
  "count": 2,
  "entity": "collection",
  "items": [
    {
      "id": "pay_7IZD7aJ2kkmOjk",
      "entity": "payment",
      "amount": 50000,
      "currency": "INR",
      "status": "captured",
      "order_id": null,
      "invoice_id": null,
      "international": false,
      "method": "wallet",
      "amount_refunded": 0,
      "refund_status": null,
      "captured": true,
      "description": "Purchase Description",
      "card_id": null,
      "bank": null,
      "wallet": "freecharge",
      "vpa": null,
      "email": "gaurav.kumar@example.com",
      "contact": "9123456780",
      "notes": {
        "merchant_order_id": "order id"
      },
      "fee": 12,
      "tax": 2,
      "error_code": null,
      "error_description": null,
      "created_at": 1487348129
    },
    {
      "id": "pay_19btGlBig6xZ2f",
      "entity": "payment",
      "amount": 29935,
      "currency": "INR",
      "status": "captured",
      "order_id": null,
      "invoice_id": null,
      "international": false,
```

```
      "method": "card",
      "amount_refunded": 0,
      "refund_status": null,
      "captured": true,
      "description": "Purchase Description",
      "card_id": "card_12abClEig3hi2k",
      "bank": null,
      "wallet": null,
      "vpa": null,
      "email": "gaurav.kumar@example.com",
      "contact": "9123456780",
      "notes": {
        "merchant_order_id": "rcptid#2"
      }
    },
    {
      "fee": 12,
      "tax": 2,
      "error_code": null,
      "error_description": null,
      "created_at": 1400826750
    }
  ]
}
```

3.9.2.3. Fetch Payments for an Order

The payments corresponding to an order can be obtained from the API.

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \ -X GET \
https://api.razorpay.com/v1/orders/order_29QQoUBi66xm2f/payments
```

Response

```
{
  "count": 1,
  "entity": "collection",
  "items": [
```

```
    {
      "id": "pay_29QQoUBi66xm2f",
      "entity": "payment",
      "amount": 50000,
      "currency": "INR",
      "status": "captured",
      "amount_refunded": 0,
      "refund_status": null,
      "email": "gaurav.kumar@example.com",
      "contact": "9364591752",
      "error_code": null,
      "error_description": null,
      "notes": {},
      "created_at": 1400826750
    },
  ]
}
```


3.9.3. Supported Webhooks

Razorpay offers following webhook events for the Payment entity:

`payment.authorized`	Triggered when a payment is successfully authorized by the bank.
`payment.captured`	Triggered when a payment is successfully captured by the merchant
`payment.failed`	Triggered when a payment processing has failed

Example Webhook Payload


```
{
 "event": "payment.authorized",
 "entity": "event",
 "contains": [
   "payment"
 ],
 "payload": {
   "payment": {
```

```
    "entity": {
      "id": "pay_6X6jcHoHdRdy79",
      "entity": "payment",
      "amount": 50000,
      "currency": "INR",
      "status": "authorized", // for payments that have failed, the status
would be "failed".
      "amount_refunded": 0,
      "refund_status": null,
      "method": "card",
      "order_id": "order_6X4mcHoSXRdy79",
      "card_id": "card_6GfX4mcIAdsfDQ",
      "bank": null,
      "captured": true,
      "email": "gaurav.kumar@example.com",
      "contact": "9123456780",
      "description": "Payment Description",
      "error_code": null,
      "error_description": null,
      "fee": 200,
      "service_tax": 10,
      "international": false,
      "notes": {
        "reference_no": "848493"
      },
      "vpa": null,
      "wallet": null
    }
  },
  "created_at": 1400826760
 }
}
```

Learn how to setup a webhook.

3.10. Capturing Payments

The payment capture mechanism is a security step which enables the merchants to verify the payments made towards their account.

> ***NOTE**: The captured amount must be same as authorized amount.*

Depending on the capture mechanism being used, the payment flow differs. There are two ways in which you can capture payments:

1. Manual Capture
2. Auto Capture

3.10.1. Manual Capture

If you want to manually capture the payment, you need to make an API request to Razorpay as shown in the below example.

Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X POST \
   --data "amount=500" \
   https://api.razorpay.com/v1/payments/pay_29QQoUBi66xm2f/capture
```

In this capture method, there is no need for a verification step since you are manually making a capture request.

3.10.2. Auto Capture

Razorpay Orders allows us to auto-capture payments on your behalf after getting authorized from the bank. In such cases, the payment automatically moves to captured state. In such cases, you do not need to make an extra network request.

You can enable auto-capture of payments by passing `payment_capture=1` in the create order request as shown below.

3.10.2.1. Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X POST \
   --data "amount=500" \
   --data "currency=INR" \
   --data "receipt=Receipt #20" \
   --data "payment_capture=1" \

 https://api.razorpay.com/v1/orders
```

> ***NOTE**: This capture mechanism will be enabled by default. To disable this refer the Standard Capture section*

In this capture mechanism a payment verification is required explained in the following section.

3.10.3. Verifying the Payment

To verify a payment, a signature needs to be generated at your end which is calculated as HMAC hex digest using SHA256 algorithm and the fields `razorpay_payment_id`, `Razorpay_order_id` and `razorpay_signature`. The `razorpay_payment_id` and `razorpay_order_id` fields are concatenated using the | (pipe) symbol.

Following is a pseudo code for the signature:

```
= hmac_sha256(razorpay_order_id + "|" + razorpay_payment_id, secret);

if (generated_signature == razorpay_signature) {
  payment is successful
}
```

3.12. Refunds

Razorpay provides API for issuing refunds. The refunds can be made for full amount or partial amount. All refunds are credited back to the same source with timelines that varies with the payment method.

3.12.1. Full Payment Refund API

The API for creating a refund against a payment Id is shown below:

3.12.1.1. Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X POST \
   https://api.razorpay.com/v1/payments/pay_29QQoUBi66xm2f/refund
```

3.12.1.2. Response

```
{
  "id": "rfnd_5UXHCzSiC02RBz",
  "entity": "refund",
  "amount": 1000,
  "currency": "INR",
  "payment_id": "pay_5UWttxtCjkrldV",
  "notes": {},
  "created_at": 1462887226
}
```

3.12.2. Partial Refund API

The API for creating a partial refund against a payment Id is shown below:

3.12.2.1. Example Request

```
curl -u <YOUR_KEY>:<YOUR_SECRET> \
   -X POST \
   https://api.razorpay.com/v1/payments/pay_29QQoUBi66xm2f/refund --data "amount=500" \
```

3.12.2.2. Response

```
{
  "id": "rfnd_5UXHCzSiC02RBz",
  "entity": "refund",
  "amount": 500,
  "currency": "INR",
  "payment_id": "pay_5UWttxtCjkrldV",
  "notes": {},
  "created_at": 1462887226
}
```

Learn more about Refunds.

3.12.3. Batch Refunds

Batch Refunds feature allows refunds to be made in bulk by uploading an Excel file.

Learn how to create a batch payment refund

3.12.4. Supported Webhooks

Razorpay also provides a webhook for refunds that you can use to get notified when a refund is created against a payment on a merchant account.

Learn how to setup a webhook.

3.13. Dispute

A dispute is a situation that arises when your customer or the issuing bank questions the validity of a payment. It could arise due to reasons such as unauthorised charges, failure to deliver the promised merchandise, excessive charges and many others.

A dispute can be of the following different types:

- **Fraud**: It is a dispute that gets raised when the issuing bank suspects a transaction to be fraudulent based on the risk analysis.
- **Retrieval**: A retrieval is a request, initiated by the customer with their issuer bank for more information about a transaction. This is essentially a soft chargeback.
- **Chargeback**: Chargeback is a claim raised against a transaction, initiated by the customers with their issuer bank, for a refund. The bank starts an official enquiry in this case.
- **Pre-Arbitration**: When a chargeback that you have won is challenged by the customer for the second time.

- **Arbitration**: When a chargeback that you have won is challenged for a third time by the customer and the card networks directly get involved. These disputes are usually very expensive.

3.13.1. Disputes Process Flow

The overall flow in which disputes are raised is given below:

1. A dispute is always raised by the issuing bank. However, it can be initiated by the bank or the customer.
    1. **Initiated by the issuing bank**: The issuing bank suspects a fraudulent transaction and asks the merchant for justification.
    2. **Initiated by the customer**: The customer claims that the transaction was unauthorized and raises it with the issuing bank.
2. The merchant gets notified about the dispute and can do one of the following:
    1. **Accept**: If the merchant feels that the dispute is legitimate, the dispute request can be accepted. In the case of a fraud, the merchant must refund the amount. In other cases, Razorpay will auto-refund the amount.
    2. **Contest**: If the merchant feels that the dispute is not legitimate, the dispute can be contested. In this case, the merchant submits the relevant documents to prove that the transaction was fair.
3. If the merchant contests, the documents are sent to the customer's bank. The bank reviews the case and provides a verdict.
4. If the merchant loses, the amount would be deducted from the merchant's account and is sent to the customer.

3.13.2. Supported Webhooks

You can use webhooks to receive alerts whenever a dispute is created or there is a change in status. The available webhook events for disputes are:

payment.dispute.created	Triggered when a dispute against a payment is raised.
payment.dispute.lost	Triggered when the contested dispute is lost by the merchant.
payment.dispute.won	Triggered when the contested dispute is won by the merchant.
payment.dispute.closed	Triggered when the contested dispute has been closed.

3.13.2.1. Example Webhook Payload

```
{
 "entity": "event",
 "account_id": "acc_8Pl5TfyzwTDtt3",
 "event": "payment.dispute.created",
```

```
"contains": [
 "payment",
 "dispute"
],
"payload": {
 "payment": {
  "entity": {
   "id": "pay_9EnFZTBmQcxoUA",
   "entity": "payment",
   "amount": 50000,
   "currency": "INR",
   "status": "captured",
   "order_id": null,
   "invoice_id": null,
   "international": false,
   "method": "netbanking",
   "amount_refunded": 0,
   "amount_transferred": 0,
   "refund_status": null,
   "captured": true,
   "description": "Add Funds to Account",
   "card_id": null,
   "bank": "HDFC",
   "wallet": null,
   "vpa": null,
   "email": "gaurav.kumar@example.com",
   "contact": "9123456789",
   "notes": {
    "dashboard": "true"
   },
   "fee": 1476,
   "tax": 226,
   "error_code": null,
   "error_description": null,
   "created_at": 1513598384
  }
 },
 "dispute": {
  "entity": {
   "id": "disp_9bg0Jxor3LpROR",
   "entity": "dispute",
   "payment_id": "pay_9EnFZTBmQcxoUA",
```

```
    "amount": 45000,
    "amount_deducted": 0,
    "currency": "INR",
    "gateway_dispute_id": "dfghj",
    "reason_code": "chargeback_authorization",
    "respond_by": 1518853863,
    "status": "open",
    "phase": "chargeback",
    "created_at": 1518594688
   }
  }
 },
 "created_at": 1518594688
}
```


Learn how to setup Webhooks.

Refer the complete documentation on Disputes Process.

3.14. Webhooks

Webhooks allow you to build or set up integrations which subscribe to certain events on Razorpay API. When one of those events is triggered, we send an HTTP POST payload in JSON to the webhooks configured URL.

You can set up a webhook from your Razorpay Dashboard and configure separate URLs for Live and Test mode. A Test mode webhook will only receive events for your test transactions.

When setting up the webhook, you will be asked to specify a *secret*. Using this *secret*, you can validate that the webhook is from Razorpay. Entering the *secret* is optional but recommended. The *secret* should never be exposed anywhere publicly.

3.14.1. Setting up Webhooks

To setup webhook for various events

1. Go to **Dashboard** > **Settings** > **Webhooks**.
2. Click **Setup your Live Webhook**.
3. Enter the **Webhook URL** where you will receive the webhook payload when the event is triggered.
4. Enter **Secret**. This field is optional.

> ***NOTE:***
> *The secret that you enter here can be used to validate that the webhook is from Razorpay. This should not be exposed publicly.*

5. Select events from the list of **Active Events** that you would like to activate.
6. Click **Save** to enable the webhook.

Refer the complete documentation on Webhooks.

3.14.2. Validation

When your webhook `secret` is set, Razorpay uses it to create a hash signature with each payload. This hash signature is passed along with each request under the header `X-Razorpay-Signature` which you need to validate at your end. The hash signature is calculated using HMAC with SHA256 algorithm; with your webhook secret set as the key, and the webhook request body as the message.

You can validate the webhook signature yourself using an HMAC calculated as shown below:

```
key                = webhook_secret
message            = webhook_body
received_signature = webhook_signature

expected_signature = hmac('sha256', message, key)

if expected_signature != received_signature
    throw SecurityError
end
```

4. Late Authorization of Payments

Late authorization is a situation that arises when a payment that is near to completion is interrupted by external factors such as network issues, technical errors at customer's/bank's end, manual intervention, and many others. In such cases, funds may get debited from the customer's bank account and Razorpay does not receive a payment status from the bank.

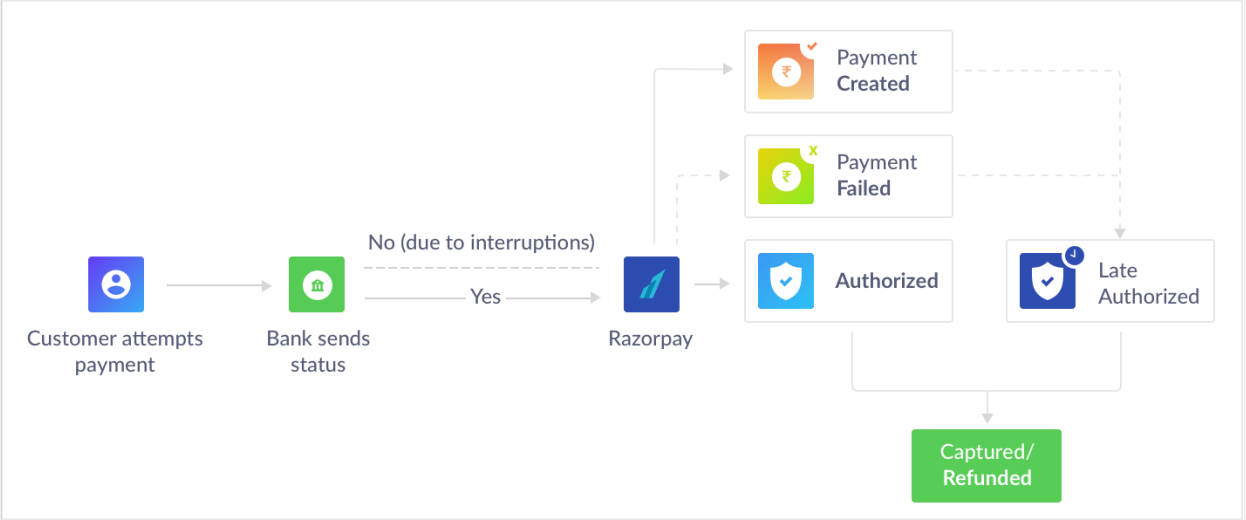
If there is no response from the bank and the payment status will remain in 'Created' for the first 10 minutes, after that it will be marked as 'Failed' due to timeout. Thereafter, Razorpay will query the bank at various intervals for 5 days, from the day of payment creation. During this time, if our system receives the payment status from the bank as Successful, the payment is marked as authorized like any other payment and is considered as Late authorized.

4.1. Handling Late Authorization of Payments

Razorpay provide multiple ways to handle these type of payments:

- If the payment is delayed by X time from the payment creation time, we can mark that payment as **Success** or **Fail**
- If there is any payment is processed in the time interval defined by the merchant, we can mark that payment as Success or Fail
- Merchant can consume **payment.authorized** webhook and then decide on a per-payment basis whether to capture them or not.

[Refer the complete documentation on Late Authorization](#).



5. Error Description

There could be situations when the API responses are not as expected due to invalid API request parameters, network errors and so on. A few of the commonly found errors along with their descriptions are described below:

- **Payment was not completed on time**

  This usually happens when a payment couldn't get completed within a stipulated time set by the banks and the session expires and we don't receive a callback from the bank. This may happen due to various reasons, for instance, the user does not receive an OTP or there is network fluctuation at the user's end.

- **Payment failed due to insufficient balance in wallet**

The preferred wallet does not has sufficient balance while making a payment and hence the above error occurs when the payment fails.

- **Payment failed**

  Payment is declined by the bank gateway where the exact reason is not disclosed by the bank. Banks usually decline the payments when the transaction is not verified, suspecting a certain risk factor (fraudulent transaction) or the customer's transaction deviates from his current purchasing pattern.

- **Payment processing cancelled by user**

  The error "Payment processing cancelled by the user" explains that the user has aborted the ongoing transaction due to which the payment couldn't get completed. This is possible due to various reasons, for instance, *OTP was not received/ delayed *Ongoing Bank's downtime which is not allowing a transaction to go through *User changed his mind *Slow/flaky internet at customer's end *Customer closed the 3D secure page during transaction etc.

- **The card number provided is not a legitimate one**

  This is quite explanatory that the card number entered by the user was incorrect and the authentication could not be completed due to which the payment failed with the above error.

- **Payment was blocked by bank because it failed certain risk checks**

  There are various risk parameters which banks work on to ensure a safe and legitimate transactions, and checks are performed at the banks' end during the authentication process, failing which results in a failed transaction with the above mentioned error.

- **Payment declined**

  This is synonymous with "Payment Failed" error description, as different banks tend to pass different codes for a same set of scenarios.

- **Payment declined by gateway**

  Most probably due to customer clicking the cancel button on 3D Secure page. While during the payment is processing, if the customer clicks on the cancel button after reaching to the 3D secure page and due to which the authentication could not get completed and end up getting aborted mid-way. This is quite similar to "Payment processing cancelled by user" and may happen due to various reasons like: *OTP was not received/ delayed *Ongoing Bank's downtime which is not allowing a transaction to go through *User changed his mind *Payment processing declined. The card issuing bank has prevented the payment from being authorized:** During the process of authentication, the bank rejects the verification mainly due to risk/ fraud suspect or change in buying pattern of the customer (a transaction of unusually high amount) and hence the payment fails with this error.

- **Payment processing failed due to 3dsecure or OTP authentication failure**

During the process of authentication, banks prompt for an E-secure password or an OTP to be entered by the customer. If the customer enters a wrong password/ OTP, the authentication could not be successfully completed and fails returning the above error.

- **There is a problem with the gateway causing the payment to fail**

  This is a rare scenario and occurs when the gateway is facing some technical issue which is eventually restricting the payment to go through.

- **Payment declined because it didn't pass all risk checks**

  This is synonymous to “Payment was blocked by bank because it failed certain risk checks” as different banks passes different error codes for the similar set of scenarios.

- **Invalid VPA. Please enter a valid Virtual Payment Address**

  This error occurs when user enters the wrong VPA while transacting because of which the payment couldn’t go through and it fails.

- **No Wallet Account is associated with the given email and mobile number combination**

  While making a payment, if the customer enters a wrong combination of credentials (Mobile number and Email ID), the payment fails returning the above error. There is also a probability that the customer is not registered with that particular wallet account and trying to make a payment through it which eventually is failing the payment.

- **Card declined by bank**

  During the process of authentication, the bank rejects the verification mainly due to risk/ fraud suspect or change in buying pattern of the customer (a transaction of unusually high amount). The bank, however, doesn’t disclose the exact reason for declining the payment to us.

- **Payment processing cancelled by user**

  While making a payment through netbanking, when the user clicks on cancel/close button on 3D secure page due to any possible reason for instance: *OTP was not received/ delayed

  *Ongoing Bank's downtime which is not allowing a transaction to go through *User changed his mind *Slow/flaky internet at customer's end The payment fails with the above error.

- **Payment failed due to incorrect card CVV**

  While making a transaction through card, if the customer enters a wrong CVV the payment fails due to unsuccessful completion of authentication at the bank’s end.

- **Payment failed because card holder couldn't be authenticated**

  This error happens with the customer hasn’t completed two factor authentication.

- **The payment failed most probably due to an invalid card number**

  While making a payment, if the customer enters a wrong card number, the payment fails due to unsuccessful authentication at the bank's end.

- **Card is expired**

  When the bank identifies that the payment is made through an expired card, the authentication fails and so the payment, returning the above error.

- **Payment declined by bank**

  This is synonymous to "Payment Failed" error description, as different banks tend to pass different codes for a same set of scenarios.

- **Payment processing failed due to insufficient balance**

  When a customer is trying to make a payment worth greater than the balance amount in his account, the bank fails the payment with the given error.

- **Payment declined due to not receiving timely response from bank**

  Banks have a time frame for processing the payments and this gets expires as soon as the timeline breaches and the callback is not received. Hence the payment gets declined by the gateway in this case with the above error.

- **The gateway request to submit payment information timed out. Please submit your details again**

  This is a rare case when a request for capturing a payment is timed out at the acquirer bank's end and so the above error occurs asking to re-attempt the transaction.

- **Expiry date is not valid**

  While making a payment, if the customer enters an invalid date of the card, the payment fails returning the above error.

- **Transaction not permitted to cardholder**

  Any transaction which is failed by the customer's issuing bank due to fraud suspect, Card blocked by the bank or due to any risk factor which is not disclosed by the banks. In this case, the customer need to contact his bank for the resolution and Razorpay has no control over this.

- **Payment failed because given card is inactive**

  When a payment is attempted using an expired card, the above error is thrown by the bank.

- **Payment was blocked because of fraud**

When the bank identifies a possible fraudulent activity (stolen card etc) during the transaction authentication process, they block the same with the above error description. Typically, this error description is thrown when the payment is attempted internationally, where the possibility of fraudulent transactions increases, as there is no 2 factor authentication for international transactions.

- **Payment processing failed by bank due to risk**

  This is synonymous to “Payment was blocked by bank because it failed certain risk checks” as different banks passes different error codes for the similar set of scenarios.

- **Payment processing failed because of expired OTP**

  OTP sent by the banks are live for a certain time limit after which it gets inactive. If a customer enters an inactive OTP on the 3D secure page, the transaction fails with this error code.