Razorpay is a popular payment gateway allowing businesses to accept online customer payments. In this article, we’ll walk through integrating Razorpay into a Node.js backend and a React frontend.

**Core Idea**

1. Create two endpoint in the backend.

* /create-order to create order on razorpay for the payment
* /verify-order to verify the payment details if the payment is done right or not

2. Make API call on this from frontend to create order and open a razorpay window to take the payment.

3. Verify the payment by making another API call to the backend

**Setting up the Node.js Backend**

Generating Razorpay Credentials

1. Create a Razorpay account if you haven’t already.
2. Log in to the Razorpay dashboard and navigate to the “Settings” section.
3. Generate your API key and API secret. These will be used to authenticate your Node.js application with Razorpay.

Installing the Razorpay Node.js SDK

1. Create a new Node.js project and initialize it with npm init -y.
2. Install the Razorpay Node.js SDK by running npm install razorpay.
3. Add a few dependencies by npm install cors dotenv express .
4. add "type": "module", in package.json as shown below.

**Creating the Backend Server**

1. Create a new file index.js and import the Razorpay SDK:

import Razorpay from 'razorpay';  
  
const razorpay = new Razorpay({  
 key\_id: 'YOUR\_RAZORPAY\_KEY\_ID',  
 key\_secret: 'YOUR\_RAZORPAY\_KEY\_SECRET',  
});

2. Create a route to handle the creation of a payment order:

app.post('/create-order', async (req, res) => {  
 try {  
 const options = {  
 amount: req.body.amount, // amount in the smallest currency unit  
 currency: 'INR',  
 receipt: 'receipt\_' + Math.random().toString(36).substring(7),  
 };  
  
 const order = await razorpay.orders.create(options);  
 res.status(200).json(order);  
 } catch (err) {  
 res.status(500).json({ error: err.message });  
 }  
});

3. Create a route to handle the payment verification:

app.post('/verify-payment', async (req, res) => {  
 try {  
 const { razorpay\_order\_id, razorpay\_payment\_id, razorpay\_signature } = req.body;  
  
 const sign = razorpay\_order\_id + '|' + razorpay\_payment\_id;  
 const expectedSign = crypto.createHmac('sha256', 'YOUR\_RAZORPAY\_KEY\_SECRET')  
 .update(sign.toString())  
 .digest('hex');  
  
 if (razorpay\_signature === expectedSign) {  
 // Payment is verified  
 res.status(200).json({ message: 'Payment verified successfully' });  
 } else {  
 res.status(400).json({ error: 'Invalid payment signature' });  
 }  
 } catch (err) {  
 res.status(500).json({ error: err.message });  
 }  
});

**Final Backend Code**

// .env  
  
RAZORPAY\_ID=""  
RAZORPAY\_KEY=""  
RAZORPAY\_KEY\_SECRET="Kaam25hai"

// index.js  
  
import Razorpay from 'razorpay';  
import dotenv from 'dotenv';  
import express from "express"  
import cors from "cors"  
const app = express()  
  
dotenv.config(); // access env values  
app.use(cors()) //allow cors so we can make request from react   
app.use(express.json()) // parse body  
  
const RAZORPAY\_KEY\_ID = process.env.RAZORPAY\_ID  
const RAZORPAY\_KEY = process.env.RAZORPAY\_KEY  
const RAZORPAY\_KEY\_SECRET = process.env.RAZORPAY\_KEY\_SECRET  
  
const razorpay = new Razorpay({  
 key\_id: RAZORPAY\_KEY\_ID,  
 key\_secret: RAZORPAY\_KEY,  
});  
  
// API Call for creating the order  
app.post('/create-order', async (req, res) => {  
 console.log("Create order")  
 console.log("body", req.body)  
 try {  
 const options = {  
 amount: req.body.amount, // amount in the smallest currency unit, in our case ( INR ) we will be using paisa ( RS \* 100)   
 currency: 'INR',  
 receipt: 'receipt\_' + Math.random().toString(36).substring(7), //Unique and random receipt ID  
 };  
  
 const order = await razorpay.orders.create(options);  
 console.log(order)  
 res.status(200).json(order);  
 } catch (err) {  
 res.status(500).json({ error: err.message });  
 }  
});  
  
// API Call for verifying the payment of the order  
app.post('/verify-payment', async (req, res) => {  
 console.log("Verify order")  
  
 try {  
 const { razorpay\_order\_id, razorpay\_payment\_id, razorpay\_signature } = req.body;  
  
 const sign = razorpay\_order\_id + '|' + razorpay\_payment\_id;  
 const expectedSign = crypto.createHmac('sha256', RAZORPAY\_KEY\_SECRET)  
 .update(sign.toString())  
 .digest('hex');  
  
 if (razorpay\_signature === expectedSign) {  
 // Payment is verified  
 console.log("Payment verified successfully")  
 res.status(200).json({ message: 'Payment verified successfully' });  
 } else {  
 console.log("Invalid payment signature")  
 res.status(400).json({ error: 'Invalid payment signature' });  
 }  
 } catch (err) {  
 res.status(500).json({ error: err.message });  
 }  
});  
  
  
app.listen(3001, console.log(`Server running in port 3001`))

// package.json {backend application}  
  
{  
 "name": "server",  
 "version": "1.0.0",  
 "description": "",  
 "main": "index.js",  
 "type": "module",  
 "scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1"  
 },  
 "keywords": [],  
 "author": "",  
 "license": "ISC",  
 "dependencies": {  
 "cors": "^2.8.5",  
 "dotenv": "^16.4.5",  
 "express": "^4.19.2",  
 "razorpay": "^2.9.4"  
 },  
 "devDependencies": {  
 "@types/express": "^4.17.21"  
 }  
}

**Integrating Razorpay with React**

**Creating the Frontend**

1. Create a new React project using npm create vite.
2. Install the Razorpay checkout script by running npm install react-razorpay.

**Initializing the Razorpay Checkout**

1. Import the Razorpay component from react-razorpay and use it in your React component:

Dont worry, I used to comment to explain the code.

Create a PaymentButton component

// \src\Components\PaymentButton.jsx  
  
  
import useRazorpay from "react-razorpay";  
  
export default function PaymentButton() {  
   
 const [Razorpay] = useRazorpay();  
   
 const RAZORPAY\_KEY\_ID = import.meta.env.RAZORPAY\_ID;  
  
 const handlePayment = async () => {  
 try {  
 // Make the API call to backend  
 const response = await fetch("http://localhost:3001/create-order", {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 },  
 body: JSON.stringify({ amount: 5000 }),  
 });  
  
 const order = await response.json();  
 console.log(order)  
 // add option for the payment gateway it can be dynamic if you want   
 // we can use prop drilling to make it dynamic  
 const options = {  
 key: RAZORPAY\_KEY\_ID,  
 amount: order.amount,  
 currency: order.currency,  
 name: "Your Company Name", // Add company details  
 description: "Payment for your order", // Add order details  
 order\_id: order.id,  
 // this is make function which will verify the payment  
 // after making the payment   
 handler: async (response) => {  
 try {  
 await fetch("http://localhost:3001/verify-payment", {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 },  
   
 body: JSON.stringify({  
 razorpay\_order\_id: response.razorpay\_order\_id,  
 razorpay\_payment\_id: response.razorpay\_payment\_id,  
 razorpay\_signature: response.razorpay\_signature,  
 }),  
 });  
 // Add onPaymentSuccessfull function here  
 alert("Payment successful!");  
 } catch (err) {  
 // Add onPaymentUnSuccessfull function here  
 alert("Payment failed: " + err.message);  
 }  
 },  
 prefill: {  
 name: "John Doe", // add customer details  
 email: "john@example.com", // add customer details  
 contact: "9999999999", // add customer details  
 },  
 notes: {  
 address: "Razorpay Corporate Office",  
 },  
 theme: {  
 // you can change the gateway color from here according to your  
 // application theme  
 color: "#3399cc",  
 },  
 };  
 const rzpay = new Razorpay(options);  
 // this will open razorpay window for take the payment in the frontend  
 // under the hood it use inbuild javascript windows api   
 rzpay.open(options);  
 } catch (err) {  
 alert("Error creating order: " + err.message);  
 }  
 };  
  
 return <button onClick={handlePayment}>Pay with Razorpay</button>;  
}

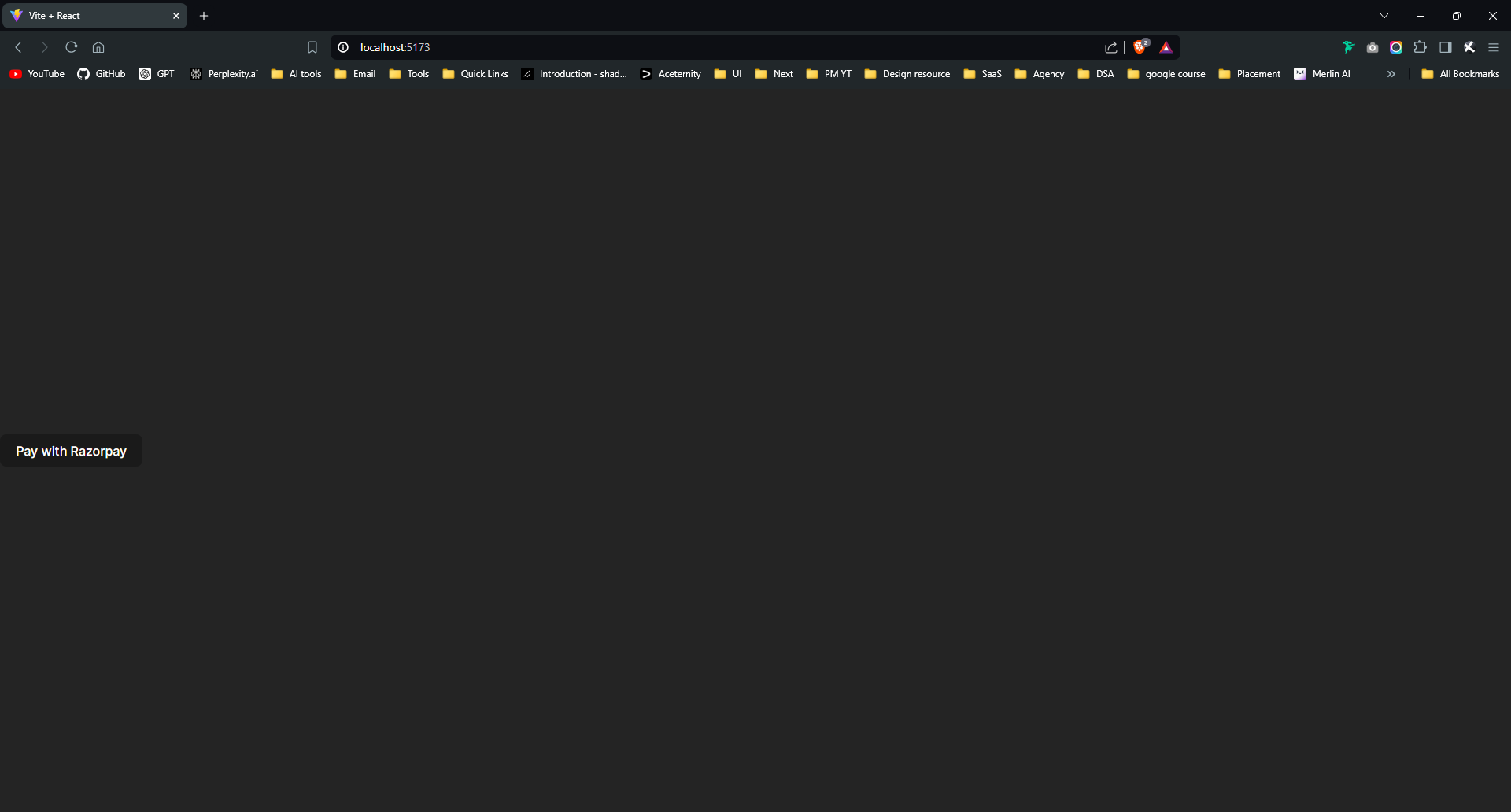
2. Use the PaymentButton component in your App:

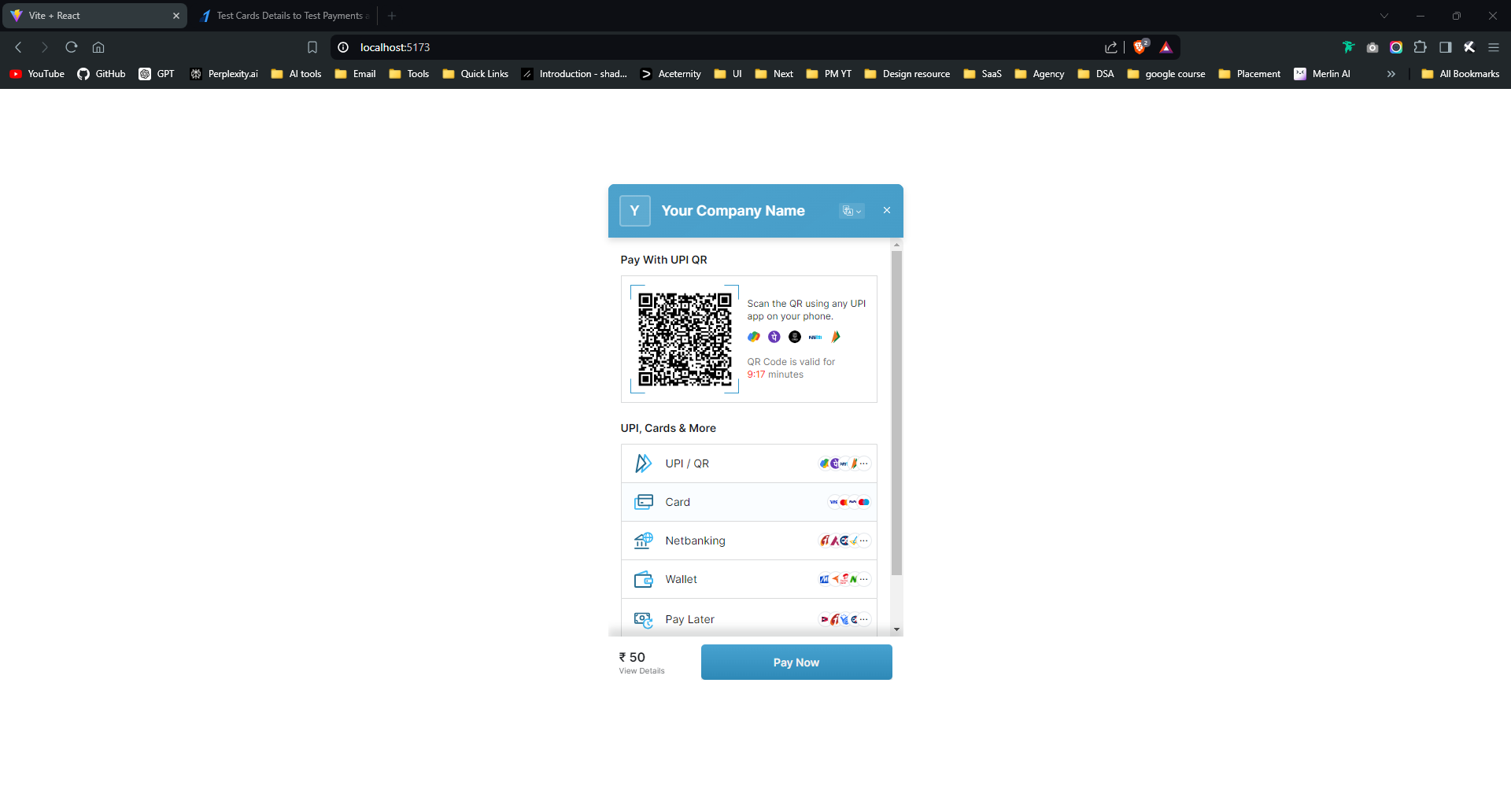
// App.js  
  
import PaymentButton from "./Components/PaymentButton"  
  
function App() {  
  
 return (  
 <>  
 <PaymentButton/>  
 </>  
 )  
}  
  
export default App

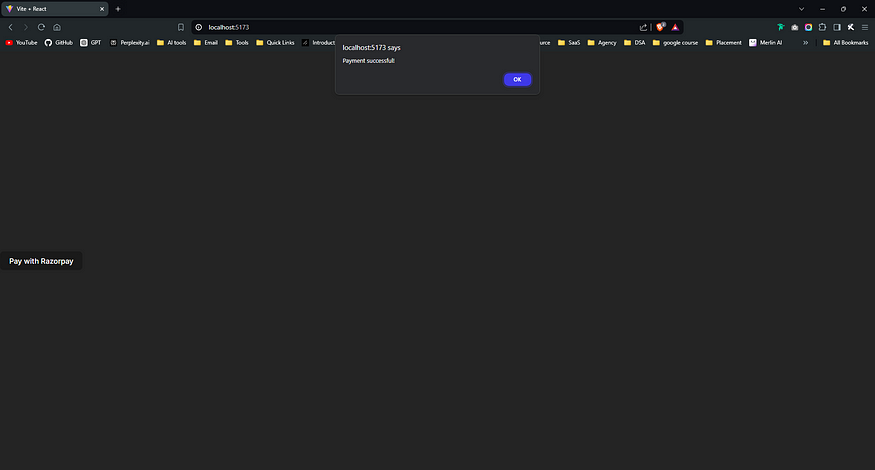
3. This code creates a new Razorpay order on the backend, then initializes the Razorpay checkout with the order details. When the user completes the payment, the handler function is called, which verifies the payment on the backend.

Here is few screenshots of sucessfull payment

For test credential you can [visit here](https://razorpay.com/docs/payments/payments/test-card-details/):







In this guide, we’ve covered how to integrate the Razorpay payment gateway with a Node.js backend and a React frontend. We set up a Node.js server to create and verify Razorpay orders, and a React component to handle the payment process. This integration allows you to securely accept payments in your web application.

**For Stripe:** [**https://docs.stripe.com/payments/checkout/how-checkout-works**](https://docs.stripe.com/payments/checkout/how-checkout-works)

Great question — let’s break this down in a practical way.

**Stripe vs Razorpay: when to use which?**

| **Criteria** | **Stripe** | **Razorpay** |
| --- | --- | --- |
| **Where your business is based** | Stripe is best suited if you are operating internationally, or in countries where Stripe has a strong presence (US, EU, Singapore, Australia, etc.). | Razorpay is excellent if your primary market is **India**, since it is fully optimized for Indian payment regulations and methods. |
| **Target customers** | Stripe supports global cards, wallets, bank debits, etc. Ideal if you have international customers paying in multiple currencies. | Razorpay is best for **Indian customers** using UPI, net banking, wallets, debit/credit cards, and even PayLater/EMI options. |
| **Regulatory compliance** | Stripe handles a lot of international compliance (PCI-DSS, GDPR, etc.) very smoothly. | Razorpay is fully compliant with RBI guidelines and local tax/GST requirements. |
| **Ease of setup** | Stripe has powerful APIs, great for developers, but can be complex for local Indian requirements like GST invoicing. | Razorpay is built for Indian small businesses, easier KYC, simpler to plug into Indian GST/tax norms. |
| **Payout speed** | Stripe payouts depend on country but usually 2–7 days. | Razorpay payouts in India can be same-day or next-day. |
| **Features** | Stripe is very feature-rich (subscriptions, billing, connect for marketplaces, fraud prevention, etc.). | Razorpay also has subscriptions, marketplace split payments, and supports features like NEFT settlement, UPI AutoPay, etc. |
| **Currencies** | Supports 135+ currencies | Supports INR and a few international currencies |
| **Marketplace scenario** | Stripe Connect is very advanced for managing multi-vendor marketplaces internationally. | Razorpay Route is simpler and great for Indian marketplaces (splitting payments to vendors, etc.). |

**So in short:**

✅ **Use Stripe**

* If your business is international
* If you have customers outside India
* If you need advanced global features (multi-currency, global marketplace payouts, etc.)

✅ **Use Razorpay**

* If your business is India-based
* If your customers are mostly Indian (because of UPI, wallets, etc.)
* If you want simpler Indian compliance, faster onboarding, easier GST handling

**👉 Example**

* A US-based SaaS selling subscriptions to worldwide customers → *Stripe*
* An Indian food delivery app taking UPI/card payments from Indian users → *Razorpay*
* An Indian e-commerce marketplace serving Indian sellers and Indian buyers → *Razorpay Route*
* An international marketplace with payouts to multiple countries → *Stripe Connect*