Razorpay

* SDK - Software Development Kit

→ npm i razorpay

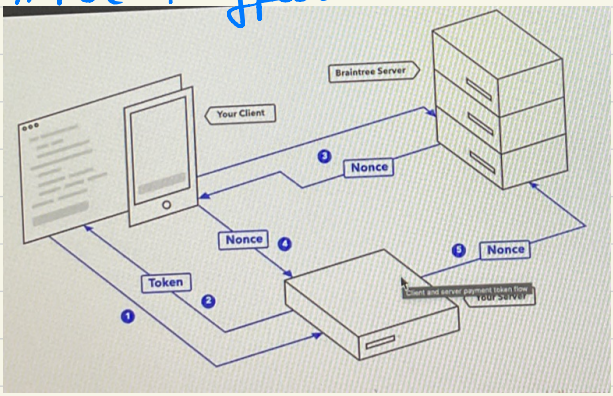
Sandbox account - Gives test credentials

* NONCE : number which cannot be regenerated

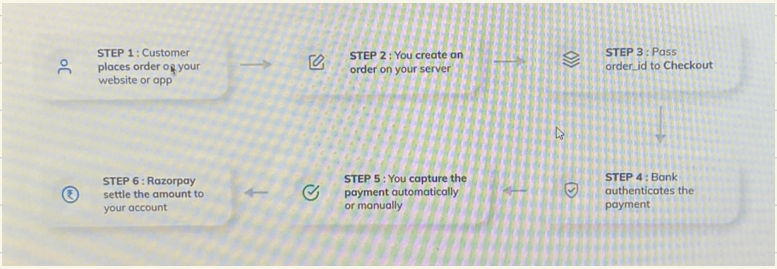
↓

as it is generated only once it helps us to verify the payment and also verifies multiple payments

// For Paypal



How Razorpay Works



* Razor_Key_ID
  Razor_Pay_Secret

*

```
import Razorpay from "razorpay"
import config from "./index.js"

const razorpay = new Razorpay({
    key_id: config.RAZORPAY_KEY_ID,
    key_secret: config.RAZORPAY_SECRET
})

export default razorpay
```

* When we want to deduct money: create a order
  When we want to refund money: generating a order

*

```
import CustomError from "../utils/customError.js";
import razorpay from "../config/razorpay.config.js"

export const generateRazorpayOrderId = asyncHandler(async (req, res) => {
    const {products, couponCode} = req.body

    if (!products || products.length === 0) {
        throw new CustomError("No product found", 400)
    }
    let totalAmount = 0

    const options = {
        amount: Math.round(totalAmount * 100),
        currency: "INR",
        receipt: `receipt_${new Date().getTime()}`
    }
    const order = await razorpay.orders.create(options)
```

100 paisa

});

* req.body : all the information we get from front end side

```
let totalAmount = 0
let discountAmount = 0

// TODO: DO product calculation based on DB calls

let productPriceCalc = Promise.all(
    products.map(async (product) => {
        const {productId, count} = product;
        const productFromDB = await Product.findById(productId)
        if (!productFromDB) {
            throw new CustomError("No product found", 400)
        }
        if (productFromDB.stock < count) {
            return res.status(400).json({ error: "Out of Stock Availability" })
        }
    })
)
```

```
        if (productFromDB.stock < count) {
            return res.status(400).json({
                error: "Product quantity not in stock"
            })
        }
        totalAmount += productFromDB.price * count
    })
)

await productPriceCalc;
```

```
// Todo: add order in database and update product stock

export const generateOrder = asyncHandler(async(req, res) => {
    //
})

//Todo: get only my orders
export const getMyOrders = asyncHandler(async(req, res) => {
    //
})

//Todo: get all my orders: Admin
export const getAllOrders = asyncHandler(async(req, res) => {
    //
})

//Todo: update order status: Admin
export const updateOrderStatus = asyncHandler(async(req, res) => {
    //
})
```

& Create routes also for these

```
import { Router } from "express";
import { generateOrder, generateRazorpayOrderId, getAllOrders, getMyOrders,
updateOrderStatus } from "../controllers/order.controller.js";
import { isLoggedIn, authorize } from "../middlewares/auth.middleware";
import AuthRoles from "../utils/authRoles.js";


const router = Router()


export default router;
```

* proxy : "https // digitalOcean /"

↑ will be added everytime a request is made in the route

So now when fetch (`/api/todo`);

request = https // digitalOcean / api / todos

OR

```
var corsOptions = {
  origin: 'http://example.com',
  optionsSuccessStatus: 200
}
```

// using cors for all the requests

Learn MongoDB Aggregation