NAMASTE NODE.JS SEASON 3

Episode-7

Hand Written Notes

-By Shanmuga Priya

www.linkedin.com/in/shanmuga-priya-e-tech2

Episode - 7

Payment Gateway Integration (Razorpay)

Workflow:

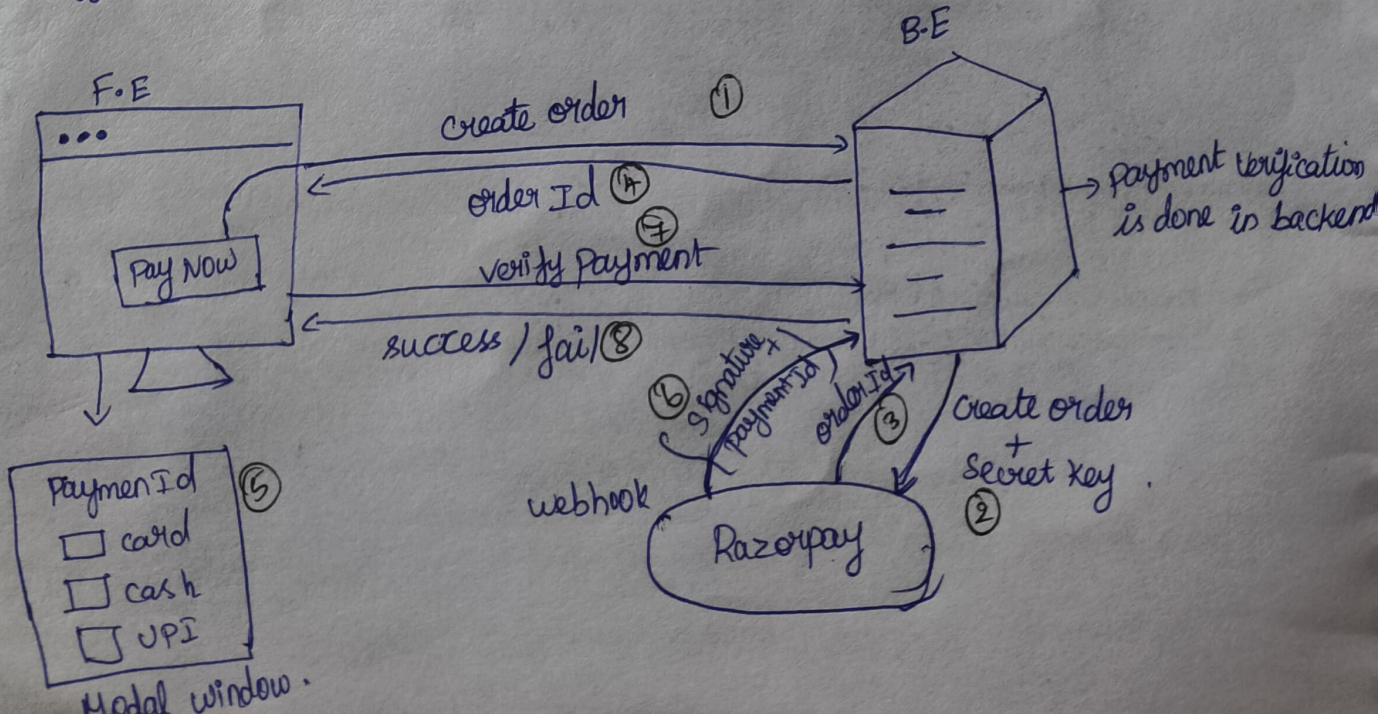
→ there is no connection between frontend and Razorpay whenever a button (Pay Now) is clicked it sends an api call to Backend then Backend will send it to Razorpay along with the secret key.

→ the Razorpay will generate an order and sends an orderId to Backend then Backend sends this order id as response to frontend.

→ Now, the pop up modal will appear along with the orderId and different mode of Payments. Once the payment is done Razorpay will automatically inform the backend that the payment is finished along with the signature & Payment Id using webhook.

→ After few minutes frontend will make an another API call to Backend to verify the Payment whether it is success (or) fail.

→ there is 2 API call made one to create (or) place an order, 2nd is to verify the Payment.



Steps Involved in Integrating Razorpay in a Project:

Step 1: Creation of Razorpay account:

→ Signup in a Razorpay using Id card such as aadhar card, website URL & so on. It will take 3-5 days for approval of the account.

Step 2: Including links in our Project:

→ In order for the approval of Razorpay, we need to create few components such as "Privacy Policy", "Terms of Use", "Refund and cancellation policy", "Contact Information", "About us" and include them as links in the footer section.

→ This step is mandatory as Razorpay explicitly mentions that business need to display these for approval. without, these links, Razorpay might reject your application (or) delay approval.

Step 3: Create a New page for Premium Membership:

→ In frontend we create a new component "Premium" which consits of diff Premium plans like Silver, gold with a button and link the Page to the Navbar for ease of access.

Step 4: Create an API for placing an Order:

Before creating an endpoint we need to create an instance for the Ra in utils folder create a new file "razorpay.js".

Step 1: Download Razorpay.

```
npm i razorpay
```

Step 2: creating instance.

```
const Razorpay = require("razorpay")
```

let instance = new Razorpay ({

key_id = "your key from razorpay dashboard",

key_secret = "your secret key from razorpay dashboard"

})

w to get a key_id & Key_Secret from dashboard?

→ go to dashboard → choose Test mode → go to Accounts & settings → click on API Keys → find key_id & key_secret.

Remember to place this in .env file.

5: Connecting the instance with the Backend API.

Create a new Route in "routes" folder as "payment.js" and add this ute in "app.js" like we do always.

→ import the razorpay instance in this file and connect it with Backend.

eg: const express = require ("express")

const {userAuth} = require ("../middleware/auth")

const razorpayInstance = require ("../utils/razorpay").

//creating Router

const paymentRouter = express.Router ()

↳ Post method as we are creating new order

paymentRouter.post ("/payment/create", userAuth, async (req, res) => {

try {

//connecting razorpayInstance.

const order = await razorpayInstance.orders.create ({

amount: 50000, → in Paise

currency: "INR",

receipt: "receipt #1",

notes: {
firstName: "value",
membershipType: "silver",
},
});

→ notes is the metadata that we need to send along with the order.

// once order is created save it in DB.
-> create a new Payment model and import it.
-> create a new Payment in the Payment model

```
const payment = new Payment({
  userId: req.user._id,   → we get user from userAuth.
  orderId: order.id,
  status: order.status,
  amount: order.amount,
  currency: order.currency
  ...
});
const savedPayment = await payment.save();

// Return back order detail to frontend.
res.json({ ...savedPayment.toJSON() });

} catch(err) {
  return res.status(500).json({ msg: err.message });
}
});
```

Step 6: Connecting this API in frontend.

-> attach a handler fn on btn and call this API whenever the button is clicked
-> this handler fn should send the membershipType selected. based on that backend data is dynamically generated.

Step 7: Displaying the Popup modal window for completing the Payment.

→ Once the API call is made successfull from frontend we need to display a modal. for that first we need to attach a razorpay modal script file to our index.html.

→ The script tag you will get from razorpay documentation in Add checkout options.

→ copy that script tag & Paste it in our "index.html".

→ extract the necessary data that we need to pass to the dialogue box from API response & pass it as "options".

eg: const { amount, keyId, currency, notes, orderId } = order.data ← api response

```
const options ={
        key : keyId,
        amount,
        currency,
        name : "Dev Tinder", → Name of the app to display in modal
        order_id: OrderId,
        prefill : {
            name : notes.firstname + " " + notes.lastname,
            email : notes.emailId,
        },
        theme : {
            color : "#F37254", → Color of the modal.
        }
    }
```

// code for display popup → passing the options.

const rzp = new window.Razorpay(options)

↳ It comes from the script tag we attached

rzp.open()

↳ it opens the modal

Step 8: Creating a new webhook in Razorpay.

→ webhook is used to communicate the backend whether the Payment is successfull (or) not.

→ in Razorpay Dashboard → go to Account & settings → then webhooks → click on Add New webhook. which ask for a "webhook URL" which is nothing but a what API should razorpay call if the payment is success (or) fail. and select the "payment Events" as "payment.failed" and "payment.captured". and add "Secret" this can be anything of our choice. and click on "create webhook".

Step 9: Creating an API for webhook

→ we need to create an API endpoint that we added in "webhook URL" while creating a webhook. for that we need to validate whether the webhook is valid (or) not.

eg:

```
Const { validateWebhookSignature } = require("razorpay/dist/utils/razorpay-utils")

//Route
PaymentRouter.post("/payment/webhook", async (req, res) => {
    try {
        // check whether webhook is valid
        Const webhookSignature = req.headers["x-Razorpay-Signature"]
                                 ↳ this will be automatically attached by RazorPay.

        Const isValidWebhook = validateWebhookSignature(
            JSON.stringify(req.body), webhookSignature,
            process.env.Razorpay_webhook_Secret
                                 ↳ it is the same Secret that we created while adding new web hook.
        )
```

```
if (!isValidWebhook) {
        return res.status(400).json({msg:"webhook signature invalid"})
}

// If there is valid webhook update the payment status in DB.
const PaymentDetails = req.body.payload.payment.entity
const payment = await Payment.findOne({orderId : PaymentDetails.
                                                    order_id})

payment.status = PaymentDetails.status
await payment.save()

//update the user as premium.
Before that update the user modal to include membershipType, ispremiu
validity & so on.
const user = await User.findOne({_id : payment.userId})
user.isPremium = true
user.membershipType = payment.notes.membershipType
await user.save()

//return success response to Razorpay.
return res.status(200).json({msg: "webhook received successfully"})
} catch(err){
        return res.status(500).json({msg: err.message})
}
```

Step 10: verify payment and update the UI.

→ create a router to check whether the user is premium (or) not.

eg:

```
PaymentRouter.get("/premium/verify", userAuth, async(req, res) => {
    const user = req.user.toJSON()
    if(user.isPremium){
        return res.json({isPremium: true})
    }
    return res.json({isPremium: false})
})
```

Step 11: update the UI

→ make an API call to this verify endpoint and update the UI based on isPremium status.

→ eg:

```
const [isPremium, setIsPremium] = useState(false)

useEffect(() => {
    verifyPremium()
}, [])

// handler fn
const verifyPremium = async() => {
    const res = await axios.get(BASE_URL + "/premium/verify", {
        withCredentials: true
    });
    if(res.data.isPremium){
        setIsPremium(true)
    }
}

// attach this handler fn to modal popup.
const options = {
    ...,
    handler: verifyPremium
}
```

, this fn will be called once the popup modal is closed

-> we return diff UI based on the isPremium state that we defined.

Step :12 Push the code to the instance

As a final step Push all the code to the instance and update the .env file.