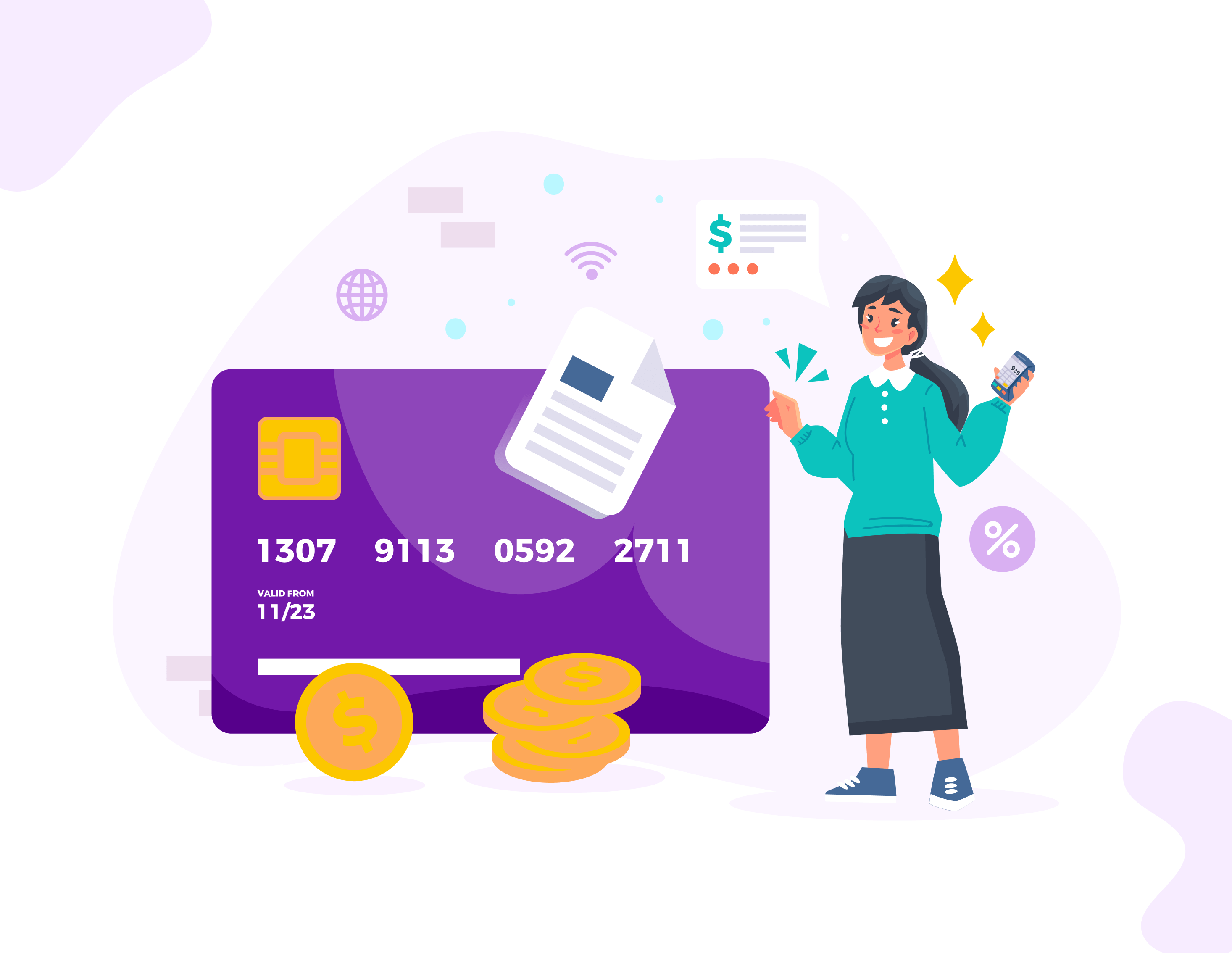
Stripe Integration

Documentation

8/2/2024



Niket Patel

Email: niket.patel@irishtaylor.com

Contents

[Stripe Integration Flow 2](#_Toc173524984)

[Step 1: Create a Stripe Account and Obtain API Keys 3](#_Toc173524985)

[Step 2: Install Stripe Library in Node.js 3](#_Toc173524986)

[Step 3: Add a secret key in the .env file 3](#_Toc173524987)

[Step 4: Create Stripe Configuration File 3](#_Toc173524988)

[Step 5: Create stripe Controller 3](#_Toc173524989)

[Initial Setup: 3](#_Toc173524990)

[Customer Retrieval or Creation: 3](#_Toc173524991)

[Check for Active Subscriptions: 4](#_Toc173524992)

[Create Checkout Session: 4](#_Toc173524993)

[Return Response: 5](#_Toc173524994)

[Step 6: Stripe Service Implentation 5](#_Toc173524995)

[Step 7: Route Configuration 6](#_Toc173524996)

[Step 8: Handle Webhooks 6](#_Toc173524997)

[Step 9: Integrate with Angular 7](#_Toc173524998)

[Figure 1 Top-level stripe flow 3](#_Toc173525840)

[Figure 2 Stripe account creation (encountered limitation) 4](#_Toc173525841)

[Figure 3 Customer activate portal 5](#_Toc173525842)

# Stripe Integration Flow In this document we are going to show the stripe integration flow from the initial, The steps to set up Stripe for payments gateway, including API endpoints and webhooks are as follows:

**Top Level Stripe flow**Some top-level stripe flows are illustrated in the figure below:

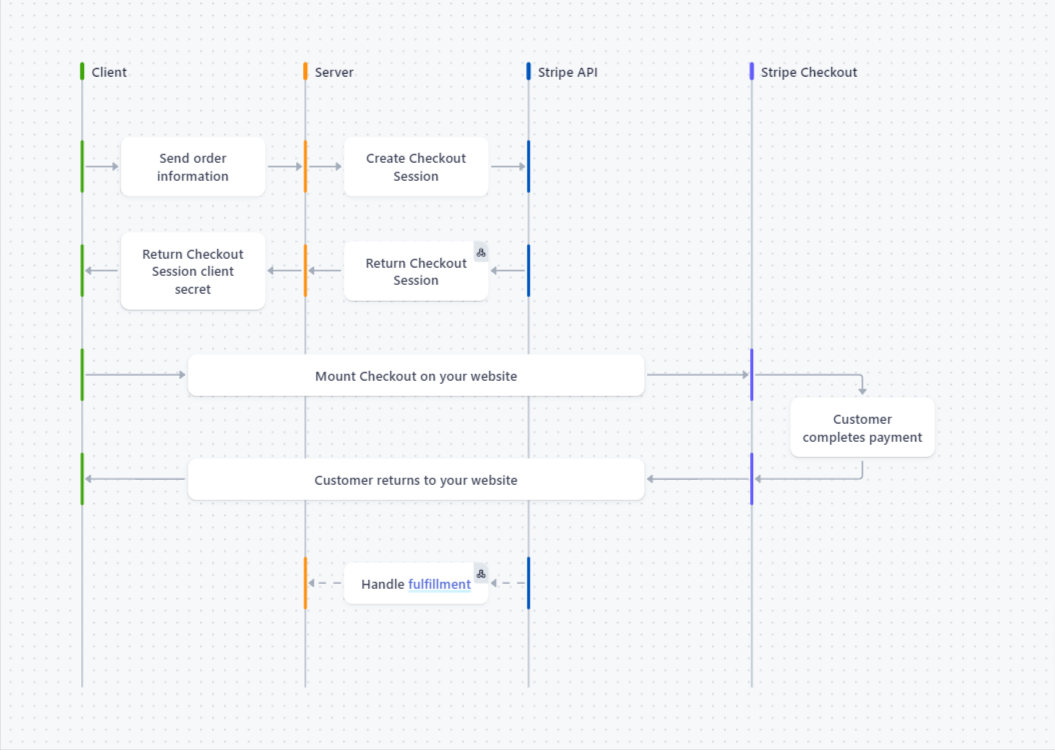


Figure 1 Top-level stripe flow

## Step 1: Create a Stripe Account and Obtain API Keys

1. Go to the Stripe website and create an account.
2. Navigate to the Stripe dashboard and click on the "Developers" tab.
3. Click on the "API keys" tab and generate a new set of API keys.
4. You will receive a publishable key and a secret key. Save these keys securely.

A screenshot of a computer

Description automatically generated

Figure 2 Stripe account creation (encountered limitation)

## Step 2: Install Stripe Library in Node.js

Run the following command in your terminal to install the Stripe library:

npm install stripe

## Step 3: Add a secret key in the .env file

STRIPE\_SECRET\_KEY=sk\_test\_51PhrJSEIG…….

## Step 4: Create Stripe Configuration File

require('dotenv').config();

const STRIPE\_CONFIG = {

    SECRET\_KEY: process.env.STRIPE\_SECRET\_KEY,

    CURRENCY: 'INR',

    // Success URL for checkout

    SUCCESS\_URL: 'http://localhost:4200/',

    // Cancel URL for checkout

    CANCEL\_URL: 'http://localhost:4200/',

    // Webhook secret key

    WEBHOOK\_SECRET: 'STRIPE\_WEBHOOK\_SECRET\_KEY',

}

module.exports = {

    STRIPE\_CONFIG

}

## Step 5: Create stripe Controller

### Initial Setup:

const userName = req.user.firstName + " " + req.user.lastName;

const userEmail = req.user.email;

* The controller extracts the user's name and email from the request object. This assumes that user information is attached to the request, likely by an authentication middleware.

### Customer Retrieval or Creation:

let customer = await stripeService.getCustomerByEmail(userEmail);

if (!customer) {

customer = await stripeService.createCustomer(userEmail, userName);

}

* It first tries to retrieve an existing Stripe customer using the user's email.
* If no customer is found, it creates a new Stripe customer with the user's email and name.
* To enable this flow, go to the Stripe portal and Active Customer Portal, as shown screenshot below.( https://dashboard.stripe.com/test/settings/billing/portal)

A screenshot of a computer

Description automatically generated

Figure 3 Customer activate portal

### Check for Active Subscriptions:

const subscriptions = await stripe.subscriptions.list({

customer: customer.id,

status: 'active',

limit: 1,

});

if (subscriptions.data.length > 0) {

// user already have subscription, then send user to manage subscription page

const stripeSession = await stripeService.createBillingPortalSession(customer.id, "http://localhost:4200/");

return res.json({ redirectUrl: stripeSession.url });

}

* If a customer already exists, it checks for any active subscriptions.
* If an active subscription is found, it creates a Billing Portal session and returns the URL to redirect the user to manage their existing subscription.

### Create Checkout Session:

const session = await stripeService.createCheckoutSession(

customer,

{

price\_data: {

currency: 'inr',

product\_data: { name: 'Standard', description: '...' },

unit\_amount: 79900,

recurring: { interval: 'month' },

},

quantity: 1,

},

"http://localhost:4200/", *// Success URL*

"http://localhost:4200/" *// Cancel URL*

);

* If no active subscription is found, it creates a new Checkout session.
* It sets up the product details, including price, currency, description, and subscription interval.
* Success and cancel URLs are provided for after the checkout process.
* Here we need to update the database subscription entries based on payment success or failure.

### Return Response:

res.json({ redirectUrl: session.url });

* Finally, it returns the URL of the created Checkout session to the client.

This flow ensures that:

* Existing customers are recognized.
* Users with active subscriptions are directed to manage their subscriptions.
* New subscribers or those without active subscriptions are directed to a new Checkout session.
* The client receives a URL to either manage an existing subscription or create a new one.

## Step 6: Stripe Service Implentation

const stripeConfig = require("../config/stripe.js");

const stripeApiKey = stripeConfig.STRIPE\_CONFIG.SECRET\_KEY;

const stripe = require('stripe')(stripeApiKey);

const createCustomer = async (email, name) => {

  return await stripe.customers.create({ email : email, name : name });

};

const getCustomerByEmail = async (email) => {

  const customers = await stripe.customers.list({ email, limit: 1 });

  return customers.data.length > 0 ? customers.data[0] : null;

};

const createCheckoutSession = async (customer, productDetails, successUrl, cancelUrl) => {

   return await stripe.checkout.sessions.create({

    customer: customer.id,

    payment\_method\_types: ['card'],

    mode: 'subscription',

    billing\_address\_collection: 'auto',

    line\_items: [productDetails],

    success\_url: successUrl,

    cancel\_url: cancelUrl,

  });

};

const createBillingPortalSession = async (customerId, returnUrl) => {

  return await stripe.billingPortal.sessions.create({ customer: customerId, return\_url: returnUrl });

};

## Step 7: Route Configuration

const express = require('express');

const router = express.Router();

const stripeController = require('../controller/stripeController');

const { isAuthenticatedUser } = require("../middleware/AuthMiddleware");

const userRouter = express.Router();

router.post('/create-stripe-session-subscription',isAuthenticatedUser, stripeController.createStripeSession);

router.post('/webhook', isAuthenticatedUser, stripeController.handleWebhook);

module.exports = router;

## Step 8: Handle Webhooks

Set up webhooks in Stripe to receive notifications when a user's subscription is updated or cancelled. Use the following code to handle webhooks:

const handleWebhook = async (req, res) => {

  // const db = client.db("subDB");

  // const subscriptions = db.collection("subscriptions");

  const sig = req.headers["stripe-signature"];

  let event;

  try {

    event = stripeService.handleStripeWebhook(req.body, sig, config.endpointSecret);

  } catch (err) {

    return res.status(400).send(`Webhook Error: ${err.message}`);

  }

  if (event.type === "invoice.payment\_succeeded") {

    const invoice = event.data.object;

    const subscription = await stripeService.retrieveSubscription(invoice.subscription);

    const customer = await stripeService.retrieveCustomer(invoice.customer);

    if (invoice.billing\_reason === "subscription\_create") {

      const subscriptionDocument = {

        userId: customer.metadata.userId,

        subId: subscription.id,

        endDate: subscription.current\_period\_end \* 1000,

      };

      await subscriptions.insertOne(subscriptionDocument);

      console.log(`First subscription payment successful for ${customer.email}`);

    } else if (invoice.billing\_reason === "subscription\_cycle") {

      await subscriptions.updateOne(

        { userId: customer.metadata.userId },

        { $set: { endDate: subscription.current\_period\_end \* 1000, recurringSuccessful\_test: true } }

      );

      console.log(`Recurring subscription payment successful for ${invoice.id}`);

    }

  }

  if (event.type === "customer.subscription.updated") {

    const subscription = event.data.object;

    if (subscription.cancel\_at\_period\_end) {

      console.log(`Subscription ${subscription.id} was canceled.`);

    } else {

      console.log(`Subscription ${subscription.id} was restarted.`);

    }

  }

  res.status(200).end();

};

## Step 9: Integrate with Angular

In your Angular application, create a new service to handle Stripe payments. Use the following code to create a new service:

onClick(action:String){

    if(action == 'Sign up')

      window.open('/signup',"\_self")

    else

    //  window.open("https://forms.office.com/r/FJcf5RQs4y", "\_blank");

     this.initiateCheckout();

  }

  initiateCheckout() {

    this.stripeService.createStripeSession().subscribe(

      response => {

        window.location.href = response.redirectUrl;

      },

      error => {

        console.error('Error creating Stripe session:', error);

        this.router.navigate(['/error']);

      }

    );

  }