STRIPE PAYMENT PART

1. Set Up Stripe

Sign up or log in to your Stripe Dashboard.

Get your Publishable Key and Secret Key from the "API keys" section.

2. Install Required Packages

In your backend project (Node.js), install the required dependencies:

bash

npm install express stripe body-parser cors mongoose dotenv

In your frontend project (React), install the Stripe package for React:

bash

npm install @stripe/react-stripe-js @stripe/stripe-js axios

3. Configure Your Environment Variables

In both your backend and frontend, create a .env file (for security) and add your keys.

Backend (.env):

STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key

MONGO\_URI=your\_mongo\_db\_uri

Frontend (.env):

REACT\_APP\_STRIPE\_PUBLISHABLE\_KEY=your\_stripe\_publishable\_key

4. Backend: Set Up Stripe Payment Intent

In your backend code, create an index.js file for the main server setup. Use Express and Stripe to create a payment intent.

Backend Code (index.js):

javascript

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const Stripe = require('stripe');

const cors = require('cors');

const bodyParser = require('body-parser');

const stripe = Stripe(process.env.STRIPE\_SECRET\_KEY);

const app = express();

// Middleware

app.use(cors());

app.use(bodyParser.json());

// MongoDB Connection

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.error('MongoDB connection error:', err));

// Endpoint to create a payment intent

app.post('/create-payment-intent', async (req, res) => {

const { amount } = req.body; // Amount in the smallest currency unit (e.g., cents for USD)

try {

const paymentIntent = await stripe.paymentIntents.create({

amount: amount,

currency: 'usd',

payment\_method\_types: ['card'],

});

res.status(200).send({

clientSecret: paymentIntent.client\_secret,

});

} catch (error) {

res.status(500).send({ error: error.message });

}

});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

5. Frontend: Set Up Stripe Payment Form

In your React app, create a payment form component to handle payments with Stripe’s React library.

PaymentForm.js (React Component)

import React, { useState } from 'react';

import { CardElement, useStripe, useElements } from '@stripe/react-stripe-js';

import axios from 'axios';

const PaymentForm = () => {

const stripe = useStripe();

const elements = useElements();

const [succeeded, setSucceeded] = useState(false);

const [error, setError] = useState(null);

const [processing, setProcessing] = useState(false);

const [clientSecret, setClientSecret] = useState('');

// Fetch client secret from the backend

const getClientSecret = async () => {

const response = await axios.post('http://localhost:5000/create-payment-intent', {

amount: 1000, // Specify amount here (in cents)

});

setClientSecret(response.data.clientSecret);

};

// Initialize payment intent when component mounts

React.useEffect(() => {

getClientSecret();

}, []);

const handleSubmit = async (event) => {

event.preventDefault();

setProcessing(true);

const payload = await stripe.confirmCardPayment(clientSecret, {

payment\_method: {

card: elements.getElement(CardElement),

},

});

if (payload.error) {

setError(`Payment failed: ${payload.error.message}`);

setProcessing(false);

} else {

setError(null);

setProcessing(false);

setSucceeded(true);

}

};

return (

<form onSubmit={handleSubmit}>

<CardElement />

<button disabled={processing || succeeded} type="submit">

{processing ? 'Processing...' : 'Pay Now'}

</button>

{error && <div>{error}</div>}

{succeeded && <div>Payment succeeded!</div>}

</form>

);

};

export default PaymentForm;

6. Integrate Stripe in the Main Component

Use Stripe's Elements and loadStripe components to integrate the payment form in your main React component.

App.js (Main React Component)

import React from 'react';

import { Elements } from '@stripe/react-stripe-js';

import { loadStripe } from '@stripe/stripe-js';

import PaymentForm from './PaymentForm';

const stripePromise = loadStripe(process.env.REACT\_APP\_STRIPE\_PUBLISHABLE\_KEY);

const App = () => {

return (

<Elements stripe={stripePromise}>

<PaymentForm />

</Elements>

);

};

export default App;

7. Run the Application

Backend: Start the server with node index.js or using nodemon if installed.

Frontend: Run the React app with npm start.

8. Save Payment Information to MongoDB

To save payment information, add additional logic to your backend’s /create-payment-intent endpoint to store data in MongoDB after a successful payment. Define a Mongoose model, for example, Payment.js:

Payment.js (Mongoose Model)

const mongoose = require('mongoose');

const paymentSchema = new mongoose.Schema({

amount: Number,

paymentIntentId: String,

createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('Payment', paymentSchema);

Then, in index.js, modify the endpoint to save payment info:

const Payment = require('./models/Payment');

app.post('/create-payment-intent', async (req, res) => {

const { amount } = req.body;

try {

const paymentIntent = await stripe.paymentIntents.create({

amount: amount,

currency: 'usd',

payment\_method\_types: ['card'],

});

// Save to MongoDB

const payment = new Payment({

amount,

paymentIntentId: paymentIntent.id

});

await payment.save();

res.status(200).send({

clientSecret: paymentIntent.client\_secret,

});

} catch (error) {

res.status(500).send({ error: error.message });

}

});

Mongo part

Setting up the MongoDB connection.

Defining the Payment schema and model.

Creating the /create-payment-intent endpoint to handle payment intent creation and save payment details to MongoDB.

**Running the Application**

1. Make sure MongoDB is running and accessible with the URI you’ve specified in .env.
2. Start the backend with:

bash

node index.js

Extra detail .

**Backend (Server Side)**

The backend will handle the server logic. It’s responsible for:

1. Connecting to MongoDB.
2. Setting up Stripe to process payments.
3. Exposing a REST API endpoint that the frontend can call to initiate a payment.

Create a new folder for your backend, for example: server.

**1. server/index.js**

This is the main server file, where you set up Express, MongoDB, and Stripe, and create an endpoint for initiating payments.

**File Location**: server/index.js

**Code:**

import express from "express";

import mongoose from "mongoose";

import dotenv from "dotenv";

import Stripe from "stripe";

import cors from "cors";

import bodyParser from "body-parser";

import Payment from "./models/Payment.js";

dotenv.config();

const app = express();

const stripe = new Stripe(process.env.STRIPE\_SECRET\_KEY);

// Middleware

app.use(cors());

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect(process.env.MONGO\_URI, { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log("MongoDB connected"))

.catch((err) => console.error("MongoDB connection error:", err));

// Payment endpoint

app.post("/create-payment-intent", async (req, res) => {

const { amount } = req.body;

try {

// Create Stripe payment intent

const paymentIntent = await stripe.paymentIntents.create({

amount,

currency: "usd",

payment\_method\_types: ["card"],

});

// Save payment data to MongoDB

const payment = new Payment({

amount,

paymentIntentId: paymentIntent.id,

});

await payment.save();

// Send client secret to frontend

res.status(200).send({ clientSecret: paymentIntent.client\_secret });

} catch (error) {

console.error("Error creating payment intent:", error);

res.status(500).send({ error: error.message });

}

});

// Start server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**2. server/Payment.js**

This file defines the MongoDB schema and model for a payment, representing each payment record in the database.

**File Location**: server/models/Payment.js

**Code:**

import mongoose from "mongoose";

const paymentSchema = new mongoose.Schema({

amount: Number,

paymentIntentId: String,

createdAt: { type: Date, default: Date.now },

});

const Payment = mongoose.model("Payment", paymentSchema);

export default Payment;

**Explanation**:

* Defines a schema with fields amount, paymentIntentId, and createdAt.
* Exports the Payment model to use it for creating and saving payment records.

**3. .env**

Create an .env file in the root of your server directory to store sensitive information.

**File Location**: server/.env

**Contents:**

plaintext

STRIPE\_SECRET\_KEY=your\_stripe\_secret\_key

MONGO\_URI=your\_mongo\_connection\_uri

PORT=….

**Frontend (Client Side)**

The frontend will display the UI for the user to enter their payment details and initiate the payment.

Create a new folder for your frontend, for example: client.

**1. client/App.js**

The main file for setting up your React application and routes. In this file, you'll include:

1. The Stripe Elements provider for handling payments.
2. React Router for navigation.
3. Importing the PaymentForm component (which you’ll create next) to process payments.

**File Location**: client/src/App.js

**Code:**

import React from "react";

import { Routes, Route } from "react-router-dom";

import { Elements } from "@stripe/react-stripe-js";

import { loadStripe } from "@stripe/stripe-js";

import PaymentForm from "./components/PaymentForm";

const stripePromise = loadStripe(process.env.REACT\_APP\_STRIPE\_PUBLISHABLE\_KEY);

function App() {

return (

<div className="App">

<Elements stripe={stripePromise}>

<Routes>

<Route path="/payment" element={<PaymentForm />} />

</Routes>

</Elements>

</div>

);

}

export default App;

**File Location**: client/src/components/PaymentForm.js

**Code:**

import React, { useState, useEffect } from "react";

import { CardElement, useStripe, useElements } from "@stripe/react-stripe-js";

import axios from "axios";

function PaymentForm() {

const [clientSecret, setClientSecret] = useState("");

const stripe = useStripe();

const elements = useElements()

useEffect(() => {

// Create payment intent on component mount

axios.post("http://localhost:5000/create-payment-intent", { amount: 1000 }) // Amount in cents

.then(response => setClientSecret(response.data.clientSecret))

.catch(error => console.error("Error:", error));

}, []);

const handleSubmit = async (event) => {

event.preventDefault();

if (!stripe || !elements) return;

const cardElement = elements.getElement(CardElement);

const { error, paymentIntent } = await stripe.confirmCardPayment(clientSecret, {

payment\_method: {

card: cardElement,

},

});

if (error) {

console.error("Payment failed:", error.message);

} else if (paymentIntent.status === "succeeded") {

console.log("Payment succeeded!");

}

};

return (

<form onSubmit={handleSubmit}>

<CardElement />

<button type="submit" disabled={!stripe}>Pay</button>

</form>

);

}

export default PaymentForm;

**Explanation**:

* **axios.post()**: Calls the backend endpoint /create-payment-intent to get the clientSecret for the payment.
* **useStripe and useElements**: Hooks provided by Stripe to interact with payment elements.
* **handleSubmit**: Handles form submission. confirmCardPayment finalizes the payment using the clientSecret and card details.

**3. .env**

Create a .env file in the root of your client directory for the frontend environment variables.

**File Location**: client/.env

**Contents:**

plaintext

REACT\_APP\_STRIPE\_PUBLISHABLE\_KEY=your\_publishable\_key\_from\_stripe

**Running the Application**

1. **Start the Backend**: Open a terminal, navigate to server, and run:

bash

node index.js

1. **Start the Frontend**: Open another terminal, navigate to client, and run:

bash

npm start