Stripe - Payments API

Friction Log

Tester / @GitHub	Shivank Rai / @rai9126
Date	May 21, 2020
Use Case Description	Accept Payments Using Stripe
OS Environment	OS Platform and Distribution: Mac OS X Catalina 10.15.4
Color Coding	This is awesome! This is a little annoying, I'm frustrated. This is awful, I feel miserable.

Note : I have leveraged the code samples found on Github for this entire project.

Step 1 : Create Payment Method for initiating Payment Intent on the server

```
 post("/create-payment-intent", (request, response) -> {
            response.type("application/json");

            CreatePaymentBody postBody = gson.fromJson(request.body(),
CreatePaymentBody.class);
            long amt = new Long(calculateOrderAmount(postBody.getItems()));
            PaymentIntentCreateParams createParams = new
PaymentIntentCreateParams.Builder()
                    .setCurrency(postBody.getCurrency()).setAmount(amt)
                    .build();
            // Create a PaymentIntent with the order amount and currency
            PaymentIntent intent = PaymentIntent.create(createParams);
            // Send publishable key and PaymentIntent details to client
            return gson.toJson(new
CreatePaymentResponse(dotenv.get("STRIPE_PUBLISHABLE_KEY"), intent.getClientSecret()));
        });
```

Observations :

- I had clear instructions about the suggested path. However, it would have been easier for me to understand the process if there had been a diagrammatic representation of the payment flow. On searching online, I found this example by Razorpay which has a similar path for setting up payment flow.
- Amount accepted is in the smallest currency unit and the minimum amount is $0.50. I did not know this and had to find the API Reference document to learn about it. This initially led to a bit of confusion.
- The naming convention used for parameters (such as `amount` and `currency`) made the API easy to understand. It was simple and easy to use.
- I can accept payments based on dynamic pricing, discounts and other internal logic, so this API allows my business to make changes as per market conditions without having to worry about downstream impact on the payment flow.

Step 2 : Collect Payment method details on client

```
var setupElements = function(data) {
    stripe = Stripe(data.publishableKey);
    var elements = stripe.elements();
    var style = {
        base: {
            color: "#32325d",
            fontFamily: '"Helvetica Neue", Helvetica, sans-serif',
            fontSmoothing: "antialiased",
            fontSize: "16px",
            "::placeholder": {
                color: "#aab7c4"
            }
        },
        invalid: {
            color: "#fa755a",
            iconColor: "#fa755a"
        }
    };

    var card = elements.create("card", { style: style });
    card.mount("#card-element");

    return {
        stripe: stripe,
        card: card,
        clientSecret: data.clientSecret
    };
};
```

Observations :

- Payment UI elements to collect card details have a nested structure. This makes the API extensible and easy to understand and use.
- As a developer, I would love to combine the above 2 steps into 1, thereby making the implementation process even more efficient. However this may create complexity in the code and would create issues where malicious users could abuse the system.
- I would have liked to modify the checkout process. I found this was possible in the API reference documents using Stripe's UI Libraries.

Step 3 : Submit the Payment to Stripe from client

```
var pay = function(stripe, card, clientSecret) {
    changeLoadingState(true);

    // Initiate the payment.
    // If authentication is required, confirmCardPayment will automatically display a
modal
    stripe
        .confirmCardPayment(clientSecret, {
            payment_method: {
                card: card
            }
        })
        .then(function(result) {
            if (result.error) {
                // Show error to your customer
                showError(result.error.message);
            } else {
                // The payment has been processed!
                orderComplete(clientSecret);
            }
        });
};
```

Observations :

- Management of credit card authentication was done within the Stripe interface, which reduces the effort required for setting up a payments gateway.
- Customer payment information does not hit my backend, so I do not have to worry about the management of this sensitive information.
- The API consists of simple unambiguous parameters that makes it easy to use for a global audience.
- Using the API is a truly delightful experience!

Step 4 : Asynchronously fulfill the customer's order (using Webhooks)

```
        post("/webhook", (request, response) -> {
            String payload = request.body();
            String sigHeader = request.headers("Stripe-Signature");
            String endpointSecret = dotenv.get("STRIPE_WEBHOOK_SECRET");

            Event event = null;

            try {
                event = Webhook.constructEvent(payload, sigHeader, endpointSecret);
            } catch (SignatureVerificationException e) {
                // Invalid signature
                response.status(400);
                return "";
            }

            switch (event.getType()) {
                case "payment_intent.succeeded":
                    // Fulfill any orders, e-mail receipts, etc
                    // To cancel the payment you will need to issue a Refund
                    // (https://stripe.com/docs/api/refunds)
                    System.out.println("💰Payment received!");
                    writeToLog("💰Payment received!","paymentlog.txt");
                    break;
                case "payment_intent.payment_failed":
                    System.out.println("❌ Payment failed.");
                    break;
                default:
                    // Unexpected event type
                    response.status(400);
                    return "";
            }

            response.status(200);
            return "";
        });
    }
```

Observations :

- Adding web hooks reduced the amount of communication required between client and server.
- If Stripe could provide a capability to instantly log transactions to a file it would reduce the amount of code required and add value to integrating with stripe.
- Bug : In the samples posted, loading the script.js file in the header wouldn't always work as sometimes

elements were not loaded before the javascript was executed. Moving this to the footer fixes the issue. However, this file was kept in the header to enable fraud detection in the background and thus moving it to the footer may defeat that purpose.

- Overall API Design is simple, composable, and predictable. Reading the API Changelog, I also found that it was backwards compatible. I also appreciate that the API avoids industry jargon like payment card number (PAN), thus making it highly scalable for all kinds and sizes of businesses worldwide.

Additional Observations :

- The CLI tool that posts webhooks to the local server is an awesome addition to the platform. I was delighted to learn that this was a recommendation that had been made by a developer(Stripe customer). The fact that Stripe built it, is a sign of high user empathy.

- Long term scalability : I came across a tweet from a developer that had not updated to using a new version of the API in 4 years and yet things did not break at their end (also a great sign of user empathy).

- Payments are a sensitive matter for both the merchant and the buyer. With unreliable networks worldwide, it is possible that buyers get charged multiple times or merchants are stuck waiting for payments to go through. The API allows for the use of idempotent requests that lead to safe rerunning of transactions. This in turn enables scalability, helps minimize effort and provides a great experience to both merchants and their buyers/customers. I was happy to find this since I have had first hand experiences of such issues.

- Properties as enum instead of booleans : Since it is difficult to predict all future states for each property, using enum enables easy refactorings in the future. This makes the API highly scalable.

- The API reference docs page had a smart night and day mode option. The Ctrl+F search was also integrated instead of being browser dependent. It’s the smallest of details that end up making a significant positive impact on the user experience. I can see why developers love using Stripe.
