Collecting Payment

One of the most popular ways of collecting payment information is a service called Stripe. To use Stripe, you will need to sign up for an account with your business details, then take a look through the documentation on how to implement it on your website. Thankfully, they pretty much just give you the basic code you need to get it working. The code below was copied almost exactly from this page: https://www.npmjs.com/package/@stripe/react-stripe-js

You will also need API keys; these are secret keys that Stripe gives you so you can access their service. Without it, you won't be authorized to load/use Stripe.

App.js

```
import './App.css';
import CheckoutForm from './CheckoutForm';
import { CardElement, Elements, useStripe, useElements } from '@stripe/react-
stripe-js';
import { loadStripe } from '@stripe/stripe-js';
// This is where you would put your public key that Stripe gives you
const stripePromise = loadStripe('public_key_from_Stripe');

const options = {
  mode: 'payment',
  amount: 1099,
  currency: 'usd',
  // Fully customizable with appearance API.
  appearance: {
    /*...*/
  },
};

function App() {
  return (
    <div className="App">
      <Elements stripe={stripePromise} options={options}>
        <CheckoutForm />
      </Elements>
    </div>
  );
}

export default App;
```

CheckoutForm.js

```
import React, {useState} from 'react';
import {
  PaymentElement,
```

```
  useStripe,
  useElements,
} from '@stripe/react-stripe-js';

const CheckoutForm = () => {
  const stripe = useStripe();
  const elements = useElements();

  const [errorMessage, setErrorMessage] = useState(null);

  const handleSubmit = async (event) => {
    event.preventDefault();

    if (elements == null) {
      return;
    }

    // Trigger form validation and wallet collection
    const {error: submitError} = await elements.submit();
    if (submitError) {
      // Show error to your customer
      setErrorMessage(submitError.message);
      return;
    }

    // Create the PaymentIntent and obtain clientSecret from your server endpoint
    const res = await fetch('/create-intent', {
      method: 'POST',
    });

    const {client_secret: clientSecret} = await res.json();

    const {error} = await stripe.confirmPayment({
      //`Elements` instance that was used to create the Payment Element
      elements,
      clientSecret,
      confirmParams: {
        return_url: 'https://example.com/order/123/complete',
      },
    });

    if (error) {
      // This point will only be reached if there is an immediate error when
      // confirming the payment. Show error to your customer (for example, payment
      // details incomplete)
      setErrorMessage(error.message);
    } else {
      // Your customer will be redirected to your `return_url`. For some payment
      // methods like iDEAL, your customer will be redirected to an intermediate
      // site first to authorize the payment, then redirected to the `return_url`.
    }
  };

  return (
```

```
      <form onSubmit={handleSubmit}>
        <PaymentElement />
        <button type="submit" disabled={!stripe || !elements}>
          Pay
        </button>
        {/* Show error message to your customers */}
        {errorMessage && <div>{errorMessage}</div>}
      </form>
    );
};

export default CheckoutForm;
```

server.js

```
import express from "express";
import Stripe from "stripe";

const stripe = new Stripe("secret_key_from_Stripe"); // Initialize Stripe
const app = express();
app.use(express.json());

app.post('/create-intent', async (req, res) => {
  const { paymentMethodId } = req.body;
  try {
    const paymentIntent = await stripe.paymentIntents.create({
      amount: 1099, // $10.99, expressed in cents
      currency: 'usd',
      payment_method: paymentMethodId,
      confirm: true, // Confirm the payment at the same time
    });
    res.json({ success: true, paymentIntent });
  } catch (error) {
    res.json({ success: false, message: error.message });
  }
});

app.listen(8080, () => console.log("Server listening on port 8080"));
```