Processing Payments with Square

Square is a payment service that emphasizes quick, secure payments as well as a user-friendly and affordable point of sale (POS) system. You may have already seen their tiny credit card readers, which are great for mobile businesses like those that sell via food trucks or craft fairs. This guide explains how to begin using Square with your Gatsby site.

Prerequisites

You'll need to set up a developer account to get started. Create a new application from your developer dashboard. The name of your application cannot include the word "square". Once that's done, you should see your new application on your dashboard. Each application has an application ID and access token associated with it.

Choosing the right product

Square can handle most of your payment-related needs including accepting payments, managing product catalogs, and managing payroll. Begin by determining which of these products you want to incorporate into your business. This guide focuses on accepting payments on your website but you may wish to use their in-person or in-app options as well.

You've got two options for integrating Square payments: redirecting to a hosted checkout page with the Checkout API *or* taking payments from your Gatsby site using the `SqPaymentForm` library with the Payments API.

Redirecting to a hosted checkout page

Redirecting to a Square-hosted page takes some of the pressure off since you don't need to build a checkout page. However, getting that functionality "for free" does come with some restrictions. You will not be able to customize the UI of the page users are sent to once they're ready to check out. Square only recommends this option for situations where accepting payments on your own site isn't feasible.

To use the Square Checkout API, you'll need to send a request that includes the relevant order information. In response, you'll get a URL with the format

`https://connect.squareup.com/v2/checkout?c={{CHECKOUT_ID}}&l={{LOCATION_ID}}` which you'll redirect your user to. From there, Square handles the payment form UI and processing.

Accepting Square payments

Square recommends using the Payments API instead because it offers much greater flexibility. You can customize not only the look and feel of the checkout process but also the checkout process itself, such as customizing the payment methods available.

This process is broken into two steps:

1. Generate a single-use token called a nonce.
2. Charge whatever payment source the user has provided (this could be a credit card, gift card, etc.) using the nonce.

To add a Square payment form to your Gatsby site, you'll need to load the JavaScript that runs Square's payment form into the `<head>` of your website. You can do this by creating a `<script>` element and appending it to the `<head>`. Wrapping this up into a function (such as `loadSquareSdk` in the following example) will allow you to load this script only when it's required (that is, only when you want to use the payment form).

```
export const loadSquareSdk = () => {
  return new Promise((resolve, reject) => {
    const sqPaymentScript = document.createElement("script")
    sqPaymentScript.src = "https://js.squareup.com/v2/paymentform"
    sqPaymentScript.crossorigin = "anonymous"
    sqPaymentScript.onload = () => {
      resolve()
    }
    sqPaymentScript.onerror = () => {
      reject(`Failed to load ${sqPaymentScript.src}`)
    }
    document.getElementsByTagName("head")[0].appendChild(sqPaymentScript)
  })
}
```

You'll also need to create some variation of a `PaymentForm` component. Square maintains a few payment form templates you can base your component on. Try starting with the "basic" JavaScript file. They also provide a running example using their "basic-digital-wallet" template with React.

Your `PaymentForm` component can be added to `paymentForm.js` along with your `loadSquareSDK` method. To check out a sample `PaymentForm` component, please see the example site using Square payments.

Once that's done, you can use the `SqPaymentForm` object available on the `window`

(you get this from the Square JS you called in the `<head>`) and pass it in via props whenever you want the form to show up! In the example below, the `PaymentForm` has been added to the homepage from the default starter.

```
import React, { useEffect, useState } from "react"
import { Link } from "gatsby"

import Layout from "../components/layout"
import SEO from "../components/seo"
import PaymentForm, { loadSquareSdk } from "../components/paymentForm" //highlight-line

const IndexPage = () => {
  const [squareStatus, setSquareStatus] = useState(null)

  useEffect(() => {
    loadSquareSdk()
      .then(() => {
        setSquareStatus("SUCCESS")
      })
      .catch(() => setSquareStatus("ERROR"))
  }, []) // on mount, add the js script dynamically

  return (
    <Layout>
      <SEO title="Home" />

      {squareStatus === "ERROR" &&
        "Failed to load SquareSDK. Please refresh the page."}
      {squareStatus === "SUCCESS" && (
        <PaymentForm paymentForm={window.SqPaymentForm} /> //highlight-line
      )}

      <Link to="/page-2/">Go to page 2</Link>
    </Layout>
  )
}

export default IndexPage
```

To charge payment source using nonce, a server-side implementation is required which will uses the Square Node.js SDK library to call the Square Payments API.

In the `PaymentForm` component, call `processPayment` end point once the nonce is generated. Use the nonce, amount and currency to capture payment. API can be served using netlify lambda function, but any form backend implementation will work.

```
axios
    .post("PROCESS_PAYMENT_API_END_POINT", {
        amount: 1 * 100 // charge 1 USD,
        currency: "USD",
        nonce, //highlight-line
    })
    .then((result) => {
        alert('Payment complete successfully!\nCheck browser developer console for more deta
    })
    .catch((error) => {
        console.log(`error in processing payment:${error}`);
        alert('Payment failed to complete!\nCheck browser developer console for more details
        this.setState({ error: true });
    });
```

Below is the Netlify lambda implementation. Create a PaymentsApi object and call the CreatePayment endpoint to charge the payment source. After receiving the `Payments.CreatePayment` endpoint request, Square processes the payment and returns a response.

```
import crypto from "crypto"
import { ApiClient, PaymentsApi } from "square-connect"

const headers = {
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Allow-Headers":
    "Origin, X-Requested-With, Content-Type, Accept",
}

exports.handler = async (event, context) => {
  try {
    // Set the Access Token (while testing use the sandbox one)
    const accessToken = "YOUR_ACCESS_TOKEN"

    // retrieves the payment data sent from the website
    const data = JSON.parse(event.body)

    // Set Square Connect credentials and environment
    const defaultClient = ApiClient.instance

    // Configure OAuth2 access token for authorization: oauth2
    const oauth2 = defaultClient.authentications["oauth2"]
    oauth2.accessToken = accessToken

    // Set 'basePath' to switch between sandbox env and production env
    // sandbox: https://connect.squareupsandbox.com
    // production: https://connect.squareup.com
```

```
    defaultClient.basePath = "https://connect.squareupsandbox.com"

    // generate a idempotency key for the payment
    // length of idempotency_key should be less than 45
    const idempotency_key = crypto.randomBytes(22).toString("hex")

    // instantiate the api
    const payments_api = new PaymentsApi()

    // request object to process the payment
    const request_body = {
      source_id: data.nonce,
      amount_money: {
        amount: data.amount,
        currency: data.currency,
      },
      idempotency_key: idempotency_key,
    }

    // calls the square payments api to process the payment issued
    const response = await payments_api.createPayment(request_body)
    return {
      statusCode: 200,
      headers,
      body: JSON.stringify({
        message: `Payment Successful`,
        paymentInfo: response,
      }),
    }
  } catch (error) {
    console.log(error)
    return {
      statusCode: 500,
      headers,
      body: "Something went wrong. Try again later",
    }
  }
}
```

Review the payment in the Sandbox Seller Dashboard. The payment is credited to the Sandbox test account whose access token is used in the application.

The Square sandbox

You can test your setup using the Square sandbox. To do so, you'll need to return to your developer dashboard. In the "Credentials" tab, you can toggle

back and forth between your production and sandbox credentials!

Other resources

- `SqPaymentForm` documentation
- Square's tutorial for online payment options
- Square's Payment Walkthrough
- Square's Test Cards
- Square's blog post on Online Payments with React + Square
- Example code for using Square Payments