

Stripe Payments Integration 101

For JavaScript Developers — Version 0.1.2 Created 2019-02-01 Fri 09:34

Roland Szőke

Table of Contents

1	Introduction	1
1.1	About The Author	1
1.2	About the Project	1
2	The Basics of Stripe Payments Integration....	2
2.1	Stripe Setup and Dashboard	2
2.1.1	Signup To Stripe.....	2
2.1.2	Create a New Coupon or Product	2
2.1.3	Integrate Stripe Into a Product	2
2.1.4	Stripe Developers Section	2
3	Creating a Webshop in React with Charges ..	4
3.1	Steps for Creating a Webshop.....	4
3.2	Terminal Commands To Build the React App.....	4
3.2.1	Create the React App and Install Stripe Elements and Axios..	5
3.2.2	Add <code>Stripe.js</code> Code to <code>index.html</code> File	5
3.3	Coding the App	5
3.3.1	Stripe Object in Root React Component <code>App.js</code>	5
3.3.2	Implementation of <code>Shop.js</code> Component	6
3.3.3	Explanation of <code>Shop.js</code>	8
3.3.4	POST Request to Stripe	9
3.3.5	Charge a Credit Card in <code>Shop.js</code>	9
4	Placing an Order With Stripe.....	11
4.1	Set Up an Express Server	11
4.1.1	Access Stripe in Server <code>index.js</code>	11
4.1.2	Stripe Customers	12
4.1.3	Use Card Token as ‘Source’	12
4.1.4	Implement CC Charges in <code>index.js</code>	12
4.2	Handle Orders on the Front End	12
4.2.1	Implement State of Shop Component	13
4.2.2	Create Input Fields	14
4.2.3	Create Products and Assign SKUs	15
4.2.4	Add Products to the Webshop	17
4.2.5	Send Orders to Stripe API on Submit	17
4.2.6	Create a Coupon for Testing	18
5	Setting up Stripe Webhooks to Verify Payments	21
5.1	Webhook Notification When Order Gets Paid	21
5.1.1	Webhook Setup using Stripe Dashboard	21
5.1.2	Decrypting Request Body Sent by Stripe	22

6 Wrapping It Up	23
LIST OF FIGURES	24
LIST OF CODE FRAGMENTS	25
CONCEPT INDEX	26
SOURCE INDEX	29

1 Introduction

In this article, I'll show how you can create a simple webshop using Stripe Payments integration, React and Express. We'll get familiar with the Stripe Dashboard and basic Stripe features such as charges, customers, orders, coupons and so on. Also, you will learn about the usage of webhooks and restricted API keys.

1.1 About The Author

A little bit about my Stripe experience and the reasons for writing this tutorial: At RisingStack (<https://risingstack.com/>) we've been working with a client from the US healthcare scene who hired us to create a large-scale webshop they can use to sell their products. During the creation of this Stripe based platform, we spent a lot of time with studying the documentation and figuring out the integration. Not because it is hard, but there's a certain amount of Stripe related knowledge that you'll need to internalize.

1.2 About the Project

We'll build an example app in this tutorial together—so you can learn how to create a Stripe Webshop from the ground up! The example app's frontend can be found at <https://github.com/RisingStack/post-stripe>, and its backend at <https://github.com/RisingStack/post-stripe-api>.

I'll use code samples from these repo's in the article below.

2 The Basics of Stripe Payments Integration

First of all, what is the promise of Stripe? It is basically a payment provider: you set up your account, integrate it into your application and let the money rain. Pretty simple right? Well, let your finance people decide if it is a good provider or not based on the plans they offer.

To show you how to use Stripe, we'll build a simple demo application with it together.

2.1 Stripe Setup and Dashboard

Before we start coding, we need to create a Stripe account. Don't worry, no credit card is required in this stage. You only need to provide a payment method when you attempt to activate your account.

2.1.1 Signup To Stripe

Go straight to the Stripe Dashboard (<https://dashboard.stripe.com/login>) and hit that **Sign up** button. Email, name, password... the usual. BOOM! You have a dashboard. You can create, manage and keep track of orders, payment flow, customers... so basically everything you want to know regarding your shop is here.

2.1.2 Create a New Coupon or Product

If you want to create a new coupon or product, you only need to click a few buttons or enter a simple curl command to your terminal, as the Stripe API Doc (<https://stripe.com/docs/api>) describes.

2.1.3 Integrate Stripe Into a Product

Of course, you can integrate Stripe into your product so your admins can set them up from your UI, and then integrate and expose it to your customers using Stripe.js (<https://github.com/stripe/stripe-node>).

2.1.4 Stripe Developers Section

Another important menu on the dashboard is the **Developers** section, where we will add our first *webhook* and create our *restricted API keys*. We will get more familiar with the dashboard and the API while we implement our demo shop below.

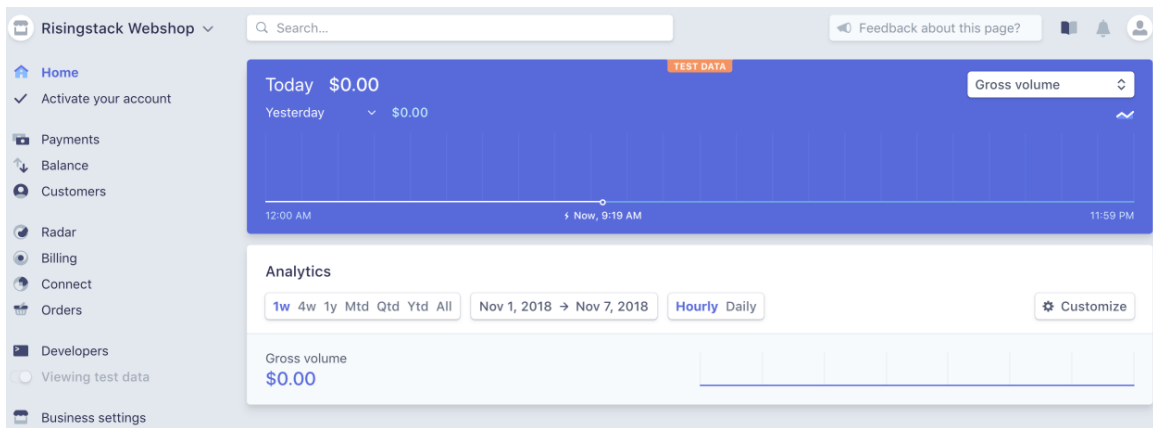


Figure 2.1: The Initial Stripe Dashboard View

3 Creating a Webshop in React with Charges

Let's create a React Webshop with two products: a Banana and Cucumber. What else would you want to buy in a webshop anyways, right?

3.1 Steps for Creating a Webshop

- We can use Create React App (<https://github.com/facebook/create-react-app>) to get started.
- We're going to use Axios (<https://github.com/axios/axios>) for HTTP requests;
- and query-string-object (<https://www.npmjs.com/package/query-string-object>) to convert objects to query strings for Stripe requests.
- We will also need React Stripe Elements (<https://github.com/stripe/react-stripe-elements>), which is a React wrapper for Stripe.js and Stripe Elements. It adds secure credit card inputs and sends the card's data for tokenization to the Stripe API.

WARNING: You should never send raw credit card details to your own API, but let Stripe handle the credit card security for you.

You will be able to identify the card provided by the user using the token you got from Stripe.

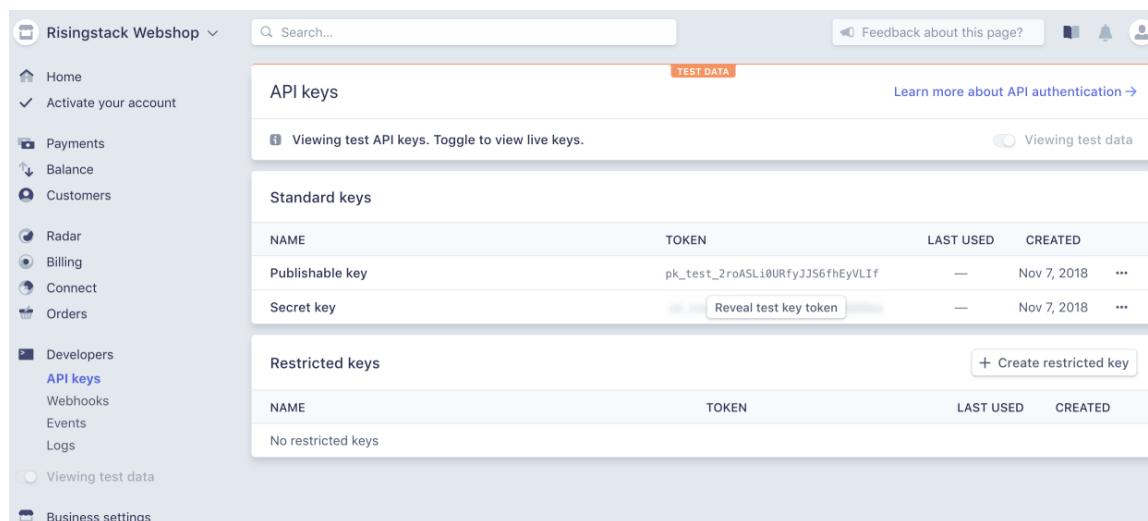


Figure 3.1: Stripe Payments Dashboard API Key

3.2 Terminal Commands To Build the React App

3.2.1 Create the React App and Install Stripe Elements and Axios

```
1 npx create-react-app webshop
2 cd webshop
3 npm install --save react-stripe-elements
4 npm install --save axios
5 npm install --save query-string-object
```

Listing 3.1: Create the Webshop Front End

3.2.2 Add Stripe.js Code to index.html File

After we're done with the preparations, we have to include `Stripe.js` in our application. Just add

```
<script src="https://js.stripe.com/v3/"></script>
```

to the head of your `index.html`.

3.3 Coding the App

3.3.1 Stripe Object in Root React Component `App.js`

First, we have to add a `<StripeProvider/>` from `react-stripe-elements` to our root React App component.

This will give us access to the Stripe object (<https://stripe.com/docs/stripe-js/reference#the-stripe-object>). In the props, we should pass a **public access key** (`apiKey`) which is found in the dashboard's **Developers** section under the *API keys* menu as *Publishable key*. (Figure 3.1)

```
1 // App.js
2 import React from 'react'
3 import {StripeProvider, Elements} from 'react-stripe-elements'
4 import Shop from './Shop'
5
6 const App = () => {
7   return (
8     <StripeProvider apiKey="pk_test_XXXXXXXXXXXXXXXXXXXXXXX">
9       <Elements>
10        <Shop/>
11      </Elements>
12    </StripeProvider>
13  )
14 }
```

Listing 3.2: `StripeProvider` Component in Root Component

The `<Shop/>` is the implementation of our shop form as you can see from `import Shop` from `'./Shop'`. We'll go into the details later.

3.3.2 Implementation of Shop.js Component

As you can see the `<Shop/>` is wrapped in `<Elements>` imported from `react-stripe-elements` so that you can use `injectStripe` in your components. To shed some light on this, let's take a look at our implementation in `Shop.js`.

```
1  // Shop.js
2  import React, { Component } from 'react'
3  import { CardElement } from 'react-stripe-elements'
4  import PropTypes from 'prop-types'
5  import axios from 'axios'
6  import qs from 'query-string-object'
7
8  const prices = {
9    banana: 150,
10   cucumber: 100
11 }
12
13 class Shop extends Component {
14   constructor(props) {
15     super(props)
16     this.state = {
17       fetching: false,
18       cart: {
19         banana: 0,
20         cucumber: 0
21       }
22     }
23     this.handleCartChange = this.handleCartChange.bind(this)
24     this.handleCartReset = this.handleCartReset.bind(this)
25     this.handleSubmit = this.handleSubmit.bind(this)
26   }
27
28   handleCartChange(evt) {
29     evt.preventDefault()
30     const cart = this.state.cart
31     cart[evt.target.name] += parseInt(evt.target.value)
32     this.setState({cart})
33   }
34
35   handleCartReset(evt) {
36     evt.preventDefault()
37     this.setState({cart:{banana: 0, cucumber: 0}})
38   }
39
40   handleSubmit(evt) {
41     // TODO
42   }
```

```

43
44   render () {
45     const cart = this.state.cart
46     const fetching = this.state.fetching
47     return (
48       <form onSubmit={this.handleSubmit}
49         style={{width: '550px', margin: '20px',
50           padding: '10px', border: '2px solid lightseagreen',
51           borderRadius: '10px'}}>
52         <div>
53           Banana {(prices.banana / 100).toLocaleString('en-US',
54             {style: 'currency', currency: 'usd'})}:
55           <div>
56             <button name="banana" value={1}
57               onClick={this.handleCartChange}>
58               +
59             </button>
60             <button name="banana" value={-1}
61               onClick={this.handleCartChange}
62               disabled={cart.banana <= 0}>
63               -
64             </button>
65             {cart.banana}
66           </div>
67         </div>
68         <div>
69           Cucumber {(prices.cucumber / 100).toLocaleString('en-US',
70             {style: 'currency', currency: 'usd'})}:
71           <div>
72             <button name="cucumber" value={1}
73               onClick={this.handleCartChange}>
74               +
75             </button>
76             <button name="cucumber" value={-1}
77               onClick={this.handleCartChange} disabled={cart.cucumber <= 0}>
78               -
79             </button>
80             {cart.cucumber}
81           </div>
82         </div>
83         <button onClick={this.handleCartReset}> Reset Cart </button>
84         <div style={{width: '450px', margin: '10px',
85           padding: '5px', border: '2px solid green',
86           borderRadius: '10px'}}>
87           <CardElement style={{base: {fontSize: '18px'}}}/>
88         </div>
89         {!fetching

```

```

90      ? <button type="submit"
91        disabled={cart.banana === 0 &&
92          cart.cucumber === 0}>Purchase</button>
93      : 'Purchasing...'
94    }
95    Price:
96      {((cart.banana * prices.banana + cart.cucumber * prices.cucumber) / 100
97        .toLocaleString('en-US',
98          {style: 'currency', currency: 'usd'}})}
99    </form>
100  )
101  }
102 }
103
104 Shop.propTypes = {
105   stripe: PropTypes.shape({
106     createToken: PropTypes.func.isRequired
107   }).isRequired
108 }

```

3.3.3 Explanation of Shop.js

Simple React Form

The `Shop` is a simple React form with purchasable elements: ‘Banana’ and ‘Cucumber’, and with a quantity ‘increase/decrease’ button for each. Clicking the buttons will change their respective amount in `this.state.cart`.

Presentation

There is a ‘submit’ button below, and the current total price of the cart is printed at the very bottom of the form. Price will expect the prices in cents, so we store them as cents, but of course, we want to present them to the user in dollars. We prefer them to be shown to the second decimal place, e.g. \$2.50 instead of \$2.5. To achieve this, we can use the built-in `toLocaleString()` function to format the prices.

Stripe Form for Card Details: `<CardElement/>`

Now comes the Stripe-specific part: we need to add a form element so users can enter their card details. To achieve this, we only need to add `<CardElement/>` from `react-stripe-elements` and that’s it. I’ve also added a bit of low effort inline ‘css’ to make this shop at least somewhat pleasing to the eye.

Pass Stripe Object As Prop to the Shop

We also need to use the `injectStripe` Higher-Order-Component in order to pass the `Stripe` object as a prop to the `<Shop/>` component, so we can call Stripe’s `createToken()` function in `handleSubmit` to tokenize the user’s card, so they can be charged. Once we receive the tokenized card from Stripe, we are ready to charge it.

```

1 // Shop.js
2 import { injectStripe } from 'react-stripe-elements'
3 export default injectStripe(Shop)

```

Listing 3.3: Inject Stripe from react-stripe-elements

3.3.4 POST Request to Stripe

For now let's just keep it simple and charge the card by sending a POST request to `'https://api.stripe.com/v1/charges'` with specifying the payment **source** (this is the token id), the charge **amount** (of the charge) and the **currency** as described in the Stripe API.

We need to send the API key in the header for authorization. We can create a restricted API key on the dashboard in the **Developers** menu. Set the permission for charges to “Read and write” as shown in Figure 3.2, below.

Do not forget:.. You should never use your swiss army Secret key on the client!

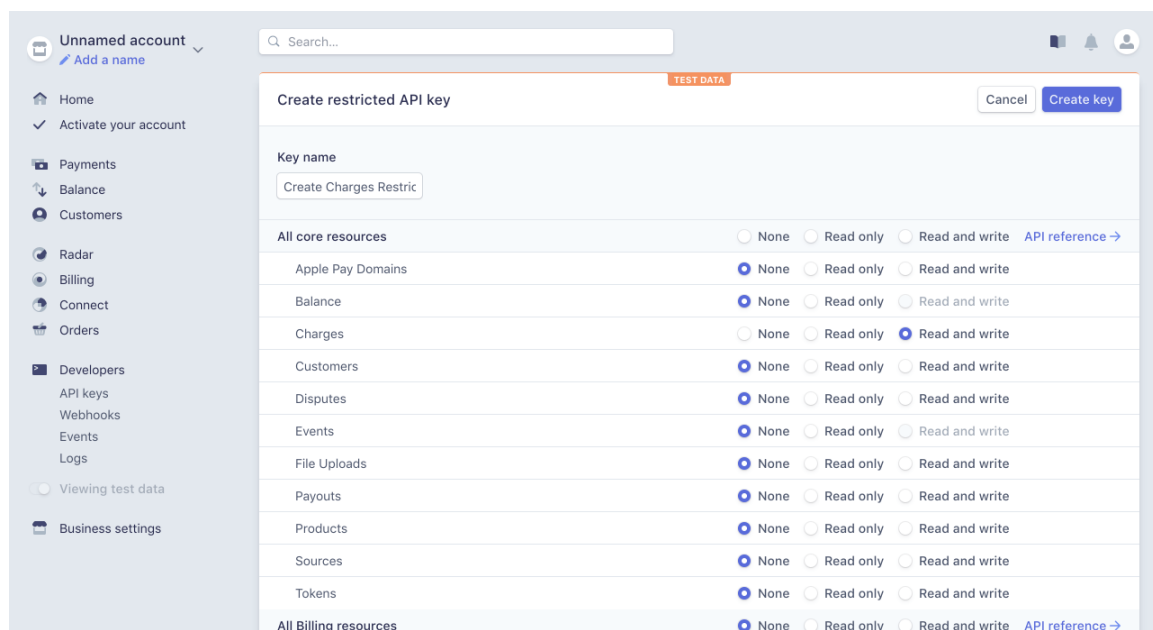


Figure 3.2: Restricted API Key in the Dashboard Developers Menu

3.3.5 Charge a Credit Card in Shop.js

```

1 // Shop.js
2 // ...
3 const stripeAuthHeader = {
4   'Content-Type': 'application/x-www-form-urlencoded',
5   'Authorization': 'Bearer rk_test_XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
6 }
7

```

```
8  class Shop extends Component {
9    // ...
10   handleSubmit(evt) {
11     evt.preventDefault()
12     this.setState({fetching: true})
13     const cart = this.state.cart
14
15     this.props.stripe.createToken().then(({token}) => {
16       const price =
17         cart.banana * prices.banana + cart.cucumber * prices.cucumber
18       axios.post('https://api.stripe.com/v1/charges',
19         qs.stringify({
20           source: token.id,
21           amount: price,
22           currency: 'usd'
23         }),
24         { headers: stripeAuthHeader })
25       .then((resp) => {
26         this.setState({fetching: false})
27         alert('Thank you for your purchase! Your card has been charged with: \
28           ${ (resp.data.amount / 100).toLocaleString('en-US',
29             {style: 'currency', currency: 'usd'}) }')
30       })
31       .catch(error => {
32         this.setState({fetching: false})
33         console.log(error)
34       })
35     }).catch(error => {
36       this.setState({fetching: false})
37       console.log(error)
38     })
39   }
40   // ...
41 }
```

For testing purposes you can use these international cards (<https://stripe.com/docs/testing#international-cards>) provided by Stripe.

4 Placing an Order With Stripe

Looks good; we can already create tokens from cards and charge them, but how should we know who bought what and where should we send the package? That's where products and orders come in.

Implementing a simple charging method is a good start, but we will need to take it a step further to create orders. To do so, we have to set up a server and expose an API which handles those orders and accepts webhooks from Stripe to process them once they got paid.

4.1 Set Up an Express Server

We will use express (<https://expressjs.com/>) to handle the routes of our API. You can find a list below of a couple of other node packages to get started. Let's create a new root folder and get started.

```
npm install express stripe body-parser cors helmet
```

The skeleton is a simple express "Hello World" using CORS (<https://www.npmjs.com/package/cors>) so that the browser won't panic when we try to reach our PI server that resides and Helmet (<https://www.npmjs.com/package/helmet>) to set a bunch of security headers automatically for us.

```

1  // index.js
2  const express = require('express')
3  const helmet = require('helmet')
4  const cors = require('cors')
5  const app = express()
6  const port = 3001
7
8  app.use(helmet())
9
10 app.use(cors({
11   origin: [/http:\/\/localhost:\d+$/],
12   allowedHeaders: ['Content-Type', 'Authorization'],
13   credentials: true
14 })))
15
16 app.get('/api/', (req, res) => res.send({ version: '1.0' }))
17
18 app.listen(port, () => console.log(`Example app listening on port ${port}!`))

```

Listing 4.1: Initial Framework for Server index.js

4.1.1 Access Stripe in Server index.js

In order to access Stripe, require `Stripe.js` and call it straight away with your Secret Key (you can find it in 'dashboard->Developers->Api keys'), we will use `stripe.orders.create()` for passing the data we receive when the client calls our server to place an order.

4.1.2 Stripe Customers

The orders will not be paid automatically. To charge the customer we can either use a ‘Source’ directly such as a **Card Token ID** or we can create a Stripe Customer (<https://stripe.com/docs/api/customers/create>).

The added benefit of creating a Stripe customer is that we can track multiple charges, or create recurring charges for them and also instruct Stripe to store the shipping data and other necessary information to fulfill the order.

You probably want to create Customers from Card Tokens and shipping data even when your application already handles users. This way you can attach permanent or seasonal discount to those Customers, allow them to shop any time with a single click and list their orders (<https://stripe.com/docs/api/orders/list>) on your UI.

4.1.3 Use Card Token as ‘Source’

For now let’s keep it simple anyway and use the Card Token as our ‘Source’ calling `stripe.orders.pay()` once the order is successfully created.

In a real-world scenario, you probably want to separate the order creation from payment by exposing them on different endpoints, so if the payment fails the Client can try again later without having to recreate the order. However, we still have a lot to cover, so let’s not overcomplicate things.

4.1.4 Implement CC Charges in `index.js`

```

1  // index.js
2  const stripe = require('stripe')('sk_test_XXXXXXXXXXXXXXXXXXXX')
3
4  app.post('/api/shop/order', async (req, res) => {
5    const order = req.body.order
6    const source = req.body.source
7    try {
8      const stripeOrder = await stripe.orders.create(order)
9      console.log('Order created: ${stripeOrder.id}')
10     await stripe.orders.pay(stripeOrder.id, {source})
11   } catch (err) {
12     // Handle stripe errors here: No such coupon, sku, ect
13     console.log('Order error: ${err}')
14     return res.sendStatus(404)
15   }
16   return res.sendStatus(200)
17 })

```

Listing 4.2: App Post Function

4.2 Handle Orders on the Front End

Now we’re able to handle orders on the backend, but we also need to implement this on the UI.

4.2.1 Implement State of Shop Component

First, let's implement the state of the `<Shop/>` as an object the Stripe API expects.

You can find out how an order request should look like here (<https://stripe.com/docs/api/orders/create>). We'll need an 'address' object with 'line1', 'city', 'state', 'country', 'postal_code' fields, a 'name', an 'email' and a 'coupon' field, to get our customers ready for coupon hunting.

```
1  // Shop.js
2  class Shop extends Component {
3    constructor(props) {
4      super(props)
5      this.state = {
6        fetching: false,
7        cart: {
8          banana: 0,
9          cucumber: 0
10       },
11       coupon: '',
12       email: '',
13       name: '',
14       address : {
15         line1: '',
16         city: '',
17         state: '',
18         country: '',
19         postal_code: ''
20       }
21     }
22     this.handleCartChange = this.handleCartChange.bind(this)
23     this.handleCartReset = this.handleCartReset.bind(this)
24     this.handleAddressChange = this.handleAddressChange.bind(this)
25     this.handleChange = this.handleChange.bind(this)
26     this.handleSubmit = this.handleSubmit.bind(this)
27   }
28
29   handleChange(evt) {
30     evt.preventDefault()
31     this.setState({[evt.target.name]: evt.target.value})
32   }
33
34   handleAddressChange(evt) {
35     evt.preventDefault()
36     const address = this.state.address
37     address[evt.target.name] = evt.target.value
38     this.setState({address})
39   }
40   // ...
```

```
41 }
```

4.2.2 Create Input Fields

Now we are ready to create the input fields. We should, of course, disable the submit button when the input fields are empty. Just the usual deal.

```
1  // Shop.js
2  render () {
3    const state = this.state
4    const fetching = state.fetching
5    const cart = state.cart
6    const address = state.address
7    const submittable =
8      (cart.banana !== 0 || cart.cucumber !== 0) &&
9      state.email &&
10     state.name &&
11     address.line1 &&
12     address.city &&
13     address.state &&
14     address.country &&
15     address.postal_code
16    return (
17      // ...
18      <div>
19        Name: <input type="text" name="name"
20          onChange={this.handleChange}/>
21      </div>
22      <div>
23        Email: <input type="text" name="email"
24          onChange={this.handleChange}/>
25      </div>
26      <div>
27        Address Line: <input type="text" name="line1"
28          onChange={this.handleChange}/>
29      </div>
30      <div>
31        City: <input type="text" name="city"
32          onChange={this.handleChange}/>
33      </div>
34      <div>
35        State: <input type="text" name="state"
36          onChange={this.handleChange}/>
37      </div>
38      <div>
39        Country: <input type="text" name="country"
40          onChange={this.handleChange}/>
41      </div>
```

```
42     <div>
43         Postal Code: <input type="text" name="postal_code"
44             onChange={this.handleAddressChange}/>
45     </div>
46     <div>
47         Coupon Code: <input type="text" name="coupon"
48             onChange={this.handleChange}/>
49     </div>
50     {!fetching
51     ? <button type="submit" disabled={!submittable}>Purchase</button>
52     : 'Purchasing...'}
53 // ...
```

We also have to define purchasable items. These items will be identified by a Stock Keeping Unit (https://en.wikipedia.org/wiki/Stock_keeping_unit) by Stripe, which can be created on the dashboard as well.

4.2.3 Create Products and Assign SKUs

First, we have to create the Products ('Banana' and 'Cucumber' on 'dashboard->Orders->Products') and then assign an SKU to them (click on the created product and 'Add SKU' in the **Inventory** group). An SKU specifies the products including its properties - size, color, quantity, and prices -, so a product can have multiple SKUs.

The screenshot shows the 'Create a product' form in the Risingstack Webshop. The form is divided into three main sections: Details, Images, and Shippable. The Details section contains fields for Name, Caption, Description, URL, Attributes, Active status, and ID. The Images section has an 'Image URL' field and an 'Add image' button. The Shippable section has a 'Shippable' checkbox and dimensions (Width, Height, Length, Weight) fields. The form is titled 'Create a product' and has 'Cancel' and 'Save product' buttons at the top right and bottom right. The left sidebar shows the navigation menu with 'Products' highlighted.

Risingstack Webshop Search... Feedback about this page?

Create a product TEST DATA Cancel Save product

Details

Name:

Caption:
A short one-line description of the product.

Description:

URL:
The URL of the webpage for the product.

Attributes:
A comma-separated list of attributes that define the SKUs for this product (e.g. color, size, gender). See the [API Reference](#) for sample attributes.

Active: ☒ This item is available for purchase

ID:
If an ID isn't provided, we'll generate one for you.

Images Image URL: Add image

No images
You can add up to 8 images using their URLs above.

☒ **Shippable**

Width: in Height: in Length: in Weight: oz
Values should be of the product when it is packed for shipping. Can be set per SKU later.

Cancel Save product

Figure 4.1: Create a Product

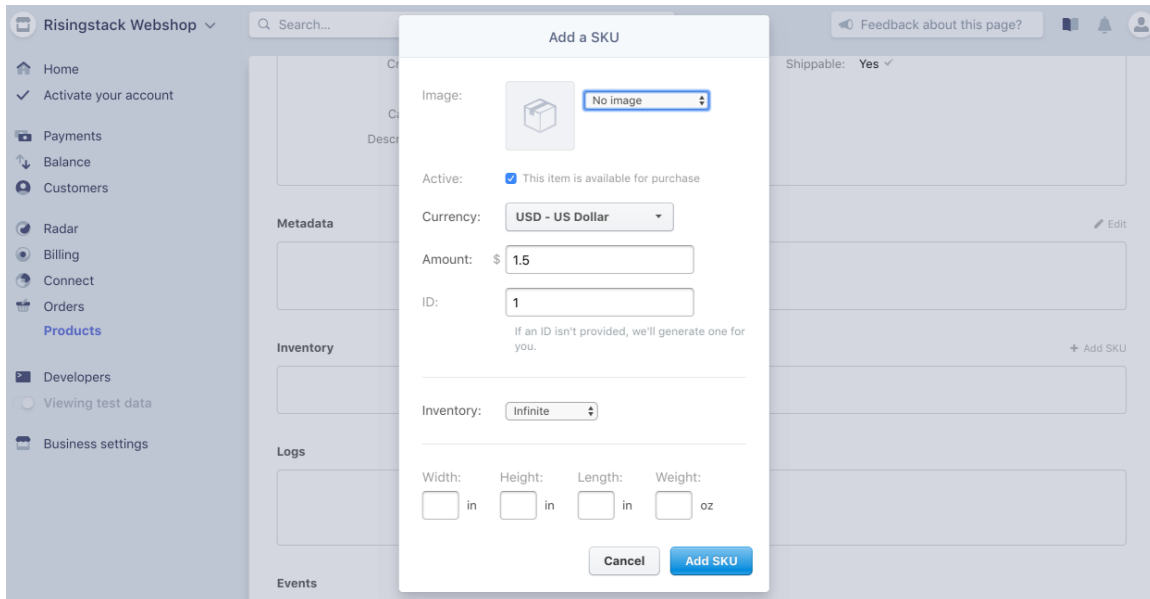


Figure 4.2: Create a SKU

4.2.4 Add Products to the Webshop

After we created our products and assigned SKUs to them, we add them to the webshop so we can parse up the order.

```
1 // Shop.js
2 const skus = {
3   banana: 1,
4   cucumber: 2
5 }
```

4.2.5 Send Orders to Stripe API on Submit

We are ready to send orders to our express API on submit. We do not have to calculate the total price of orders from now on. Stripe can sum it up for us, based on the SKUs, quantities, and coupons.

```
1 // Shop.js
2 handleSubmit(evt) {
3   evt.preventDefault()
4   this.setState({fetching: true})
5   const state = this.state
6   const cart = state.cart
7
8   this.props.stripe.createToken({name: state.name}).then(({token}) => {
9     // Create order
10    const order = {
11      currency: 'usd',
```

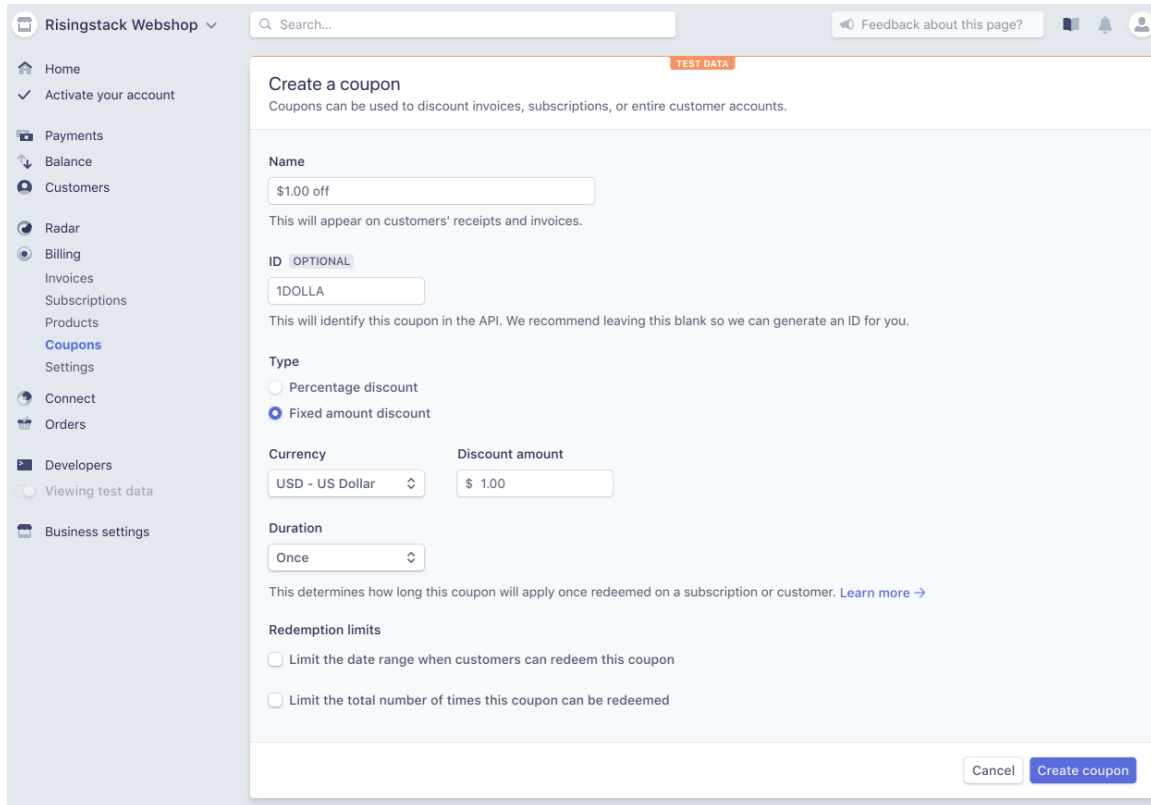
```
12     items: Object.keys(cart).filter((name) => cart[name] > 0
13       ? true : false).map(name => {
14         return {
15           type: 'sku',
16           parent: skus[name],
17           quantity: cart[name]
18         }
19       }),
20     email: state.email,
21     shipping: {
22       name: state.name,
23       address: state.address
24     }
25   }
26   // Add coupon if given
27   if (state.coupon) {
28     order.coupon = state.coupon
29   }
30   // Send order
31   axios.post('http://localhost:3001/api/shop/order', {order, source: token.id})
32     .then(() => {
33       this.setState({fetching: false})
34       alert('Thank you for your purchase!')
35     })
36     .catch(error => {
37       this.setState({fetching: false})
38       console.log(error)
39     })
40   }).catch(error => {
41     this.setState({fetching: false})
42     console.log(error)
43   })
44 }
```

4.2.6 Create a Coupon for Testing

Let's create a coupon for testing purposes. This can be done on the dashboard as well. You can find this option under the **Billing** menu on the **Coupons** tab.

There are multiple types of coupons based on their duration, but only coupons with the type *Once* can be used for orders. The rest of the coupons can be attached to Stripe Customers.

You can also specify a lot of parameters for the coupon you create, such as how many times it can be used, whether it is amount based or percentage based, and when will the coupon expire. Now we need a coupon that can be used only once and provides a reduction on the price by a certain amount.



The screenshot shows the Stripe dashboard for 'Risingstack Webshop'. The left sidebar contains navigation links: Home, Activate your account, Payments, Balance, Customers, Radar, Billing (selected), Invoices, Subscriptions, Products, Coupons, Settings, Connect, Orders, Developers, Viewing test data, and Business settings. The main content area is titled 'Create a coupon' with a 'TEST DATA' label. It includes a search bar, a feedback link, and a 'Name' field with '\$1.00 off'. Below this is an 'ID' field with '1DOLLA' and an 'OPTIONAL' label. The 'Type' section has radio buttons for 'Percentage discount' and 'Fixed amount discount' (selected). The 'Currency' is set to 'USD - US Dollar' and the 'Discount amount' is '\$ 1.00'. The 'Duration' is set to 'Once'. The 'Redemption limits' section has two unchecked checkboxes: 'Limit the date range when customers can redeem this coupon' and 'Limit the total number of times this coupon can be redeemed'. At the bottom right are 'Cancel' and 'Create coupon' buttons.

Create a coupon TEST DATA

Coupons can be used to discount invoices, subscriptions, or entire customer accounts.

Name

\$1.00 off

This will appear on customers' receipts and invoices.

ID OPTIONAL

1DOLLA

This will identify this coupon in the API. We recommend leaving this blank so we can generate an ID for you.

Type

☐ Percentage discount

☒ Fixed amount discount

Currency **Discount amount**

USD - US Dollar \$ 1.00

Duration

Once

This determines how long this coupon will apply once redeemed on a subscription or customer. [Learn more →](#)

Redemption limits

☐ Limit the date range when customers can redeem this coupon

☐ Limit the total number of times this coupon can be redeemed


[Cancel](#) [Create coupon](#)

Figure 4.3: Dashboard Coupon Creation

Great! Now we have our products, we can create orders, and we can also ask Stripe to charge the customer's card for us. But we are still not ready to ship the products as we have no idea at the moment whether the charge was successful. To get that information, we need to set up *webhooks*, so Stripe can let us know when the money is on its way.

Banana \$1.50:
 2

Cucumber \$1.00:
 1

 4242 4242 4242 4242 04 / 24 242 42424

Name:

Email:

Address Line:

City:

State:

Country:

Postal Code:

Coupon Code:

Price: \$4.00

Figure 4.4: Stripe Payments Shop Orders

5 Setting up Stripe Webhooks to Verify Payments

As we discussed earlier, we are not assigning cards but Sources to Customers. The reason behind that is Stripe is capable of using several payment methods (<https://stripe.com/docs/sources>), some of which may take days to be verified.

We need to set up an endpoint Stripe can call when an event — such as a successful payment — has happened. Webhooks are also useful when an event is not initiated by us via calling the API, but comes straight from Stripe.

Imagine that you have a subscription service, and you don't want to charge the customer every month. In this case, you can set up a webhook, and you will get notified when the recurring payment was successful or if it failed.

5.1 Webhook Notification When Order Gets Paid

In this example, we only want to be notified when an order gets paid. When it happens, Stripe can notify us by calling an endpoint on our API with an HTTP request containing the payment data in the request body. At the moment, we don't have a static IP, but we need a way to expose our local API to the public internet. We can use Ngrok (<https://ngrok.com/download>) for that. Just download it and run with `./ngrok http 3001` command to get an ngrok url pointing to our `localhost:3001`.

5.1.1 Webhook Setup using Stripe Dashboard

We also have to set up our webhook on the Stripe dashboard. Go to **Developers -> Webhooks**, click on 'Add endpoint' and type in your ngrok url followed by the endpoint to be called e.g. '`http://92832de0.ngrok.io/api/shop/order/process`'. Then under **Filter event** select 'Select types to send' and search for `order.payment_succeeded`.

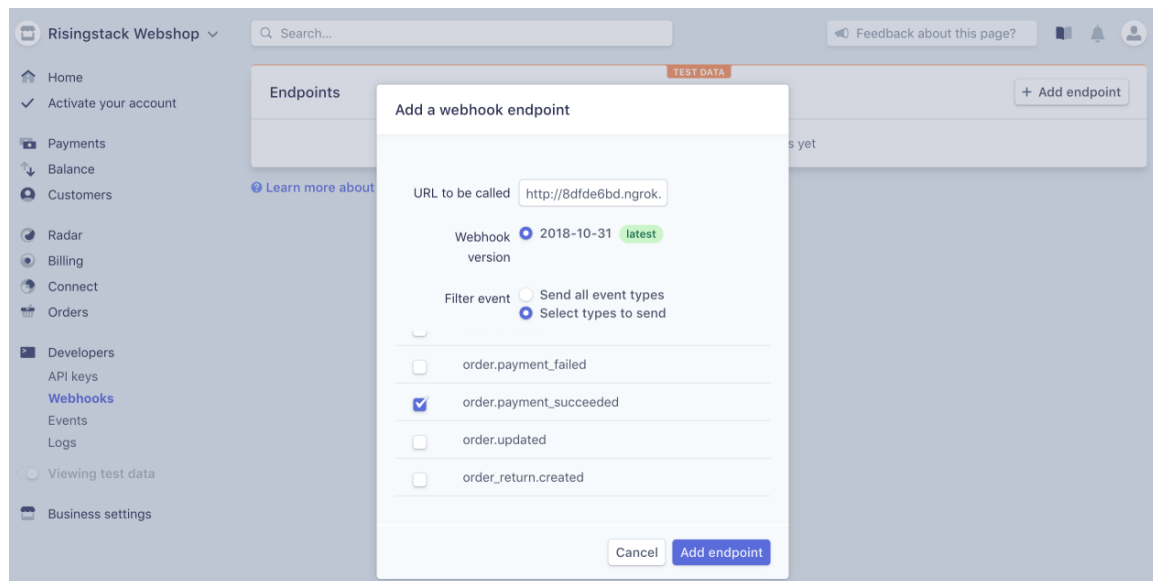


Figure 5.1: Stripe Dashboard Webhook Creation

5.1.2 Decrypting Request Body Sent by Stripe

The data sent in the request body is encrypted and can only be decrypted by using a signature sent in the header and with the webhook secret that can be found on the webhooks dashboard.

This also means that we cannot simply use `bodyParser` to parse the body, so we need to add an exception to `bodyParser` so it will be bypassed when the URL starts with `/api/shop/order/process`. We need to use the `stripe.webhooks.constructEvent()` function instead, provided by the Stripe SDK to decrypt the message for us.

```

1  // index.js
2  const bodyParser = require('body-parser')
3
4  app.use(bodyParser.json({
5    verify: (req, res, buf) => {
6      if (req.originalUrl.startsWith('/api/shop/order/process')) {
7        req.rawBody = buf.toString()
8      }
9    }
10  }))
11
12  app.use(bodyParser.urlencoded({
13    extended: false
14  }))
15
16  app.post('/api/shop/order/process', async (req, res) => {
17    const sig = req.headers['stripe-signature']
18    try {
19      const event = await
20        stripe.webhooks.
21          constructEvent(req.rawBody, sig, 'whsec_XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
22      console.log('Processing Order : ${event.data.object.id}')
23      // Process payed order here
24    } catch (err) {
25      return res.sendStatus(500)
26    }
27    return res.sendStatus(200)
28  })

```

After an order was successfully paid, we can parse and send it to other APIs like Salesforce or Stamps to pack things up and get ready to send out.

6 Wrapping It Up

My goal with this guide was to provide help to you through the process of creating a Stripe-based webshop using JavaScript. I hope you did learn from our experiences and will use this guide when you decide to implement a similar system like this in the future.

In case you need help with Stripe-based webshops, or Node & React development in general, feel free to reach out to us on info@risingstack.com.

LIST OF FIGURES

Figure 2.1: The Initial Stripe Dashboard View	3
Figure 3.1: Stripe Payments Dashboard API Key	4
Figure 3.2: Restricted API Key in the Dashboard Developers Menu	9
Figure 4.1: Create a Product	16
Figure 4.2: Create a SKU.....	17
Figure 4.3: Dashboard Coupon Creation	19
Figure 4.4: Stripe Payments Shop Orders	20
Figure 5.1: Stripe Dashboard Webhook Creation	21

LIST OF CODE FRAGMENTS

Listing 3.1: Create the Webshop Front End	5
Listing 3.2: StripeProvider Component in Root Component	5
Listing 3.3: Inject Stripe from react-stripe-elements	9
Listing 4.1: Initial Framework for Server index.js	11
Listing 4.2: App Post Function	12

CONCEPT INDEX

<

'<script>...</script>' 5

~

~react-stripe-elements~ 5

A

access key, public 5

Add Endpoint, Webhooks 21

address object 13

api keys 2

API endpoint for webhook notification 21

API key, create restricted 9

API key, Stripe Dashboard 4

API keys 5

API, expose to handle order 11

API, send orders to 17

apiKey 5

API 2, 13

App.js 5

author, about 1

authorization 9

axios 4, 5, 9

B

back end 11

backend, github 1

Billing menu, Dashboard 18

body-parser, node 11

bodyParser, exception to 22

C

Card Token 12

Card Token ID 12

CardElement 8

charge a customer 12

charges, multiple 12

charges, recurring 12

code samples, github 1

convert objects to query strings 4

cors, node 11

CORS 11

coupon 13

coupon, create 18

coupon, create new 2

coupon, parameters 18

Coupons tab, Dashboard 18

coupons, attached to Customers 18

coupons, duration 18

coupons, multiple types 18

create new coupon or product 2

create Stripe order 12

create-react-app 4, 5

createToken() function 8

credit card details, raw 4

credit card identification 4

credit card security 4

credit card, charge 8, 9

credit card, charge, POST request 9

credit card, details 8

credit card, secure inputs 4

credit card, tokenized 8

css, form style 8

Customer, create 12

D

dashboard 2

Dashboard 5, 9, 15, 18

Dashboard Developers API keys 11

Dashboard menu 5

Dashboard, define SKU in Stripe 15

dashboard, Developers section 2

Dashboard, Stripe 4

Dashboard, webhook integration 21

Dashboard, webhooks 22

data, shipping 12

data, store on Stripe 12

decrypt message using Stripe SDK 22

decrypt request body 22

Developers Dashboard Webhooks 21

Developers menu 9

Developers section, Dashboard 2, 5

discount 12

E

Elements 5, 6

endpoint for Stripe to call on event 21

endpoint, add to Dashboard 21

endpoints, different for order and payment 12

express 11

express server, set up 11

F

fields, address object 13

form, React 8

front end 5

front end, order handling 12

frontend, github 1

G

github sources 1

H

`handleSubmit` 8
 headers, security 11
 Helmet 11
`helmet`, node 11
 higher-order-component, `injectStripe` 8

I

identify credit card, how 4
`index.html` 5
`index.js` 11
`injectStripe` 6, 8
`injectStripe` hoc 8
`injectStripe(Shop)` function 8
 input fields, create 14
 integrate Stripe into product 2
 international cards 10
 Inventory group 15

K

key, public access 5

L

list orders 12

M

message, decrypt using Stripe SDK 22

N

ngrok endpoint, add 21
 ngrok service 21
 node packages, other needed 11
 notification upon payment, webhook for 21
`npm` 11
`npm install` 5
`npmx` 5

O

object, state 13
 order request, should look like 13
 order, create 11
 order, create Stripe 12
 order, parse 17
 order, send in code 17
 orders and products 11
 orders, handle on the UI 12
 orders, list on UI 12
 orders, send 17

P

parse order 17
 payment methods, several on Stripe 21
 post, axios 9, 17
 post, order 12
 POST request, credit card 9
 process orders, endpoint for,
 `/api/shop/order/process` 22
 product, create new 2
 products and orders 11
 products in webshop 4
 products, create 15
 project 1
 props 5
`props.stripe.createToken()` 9
 public access key 5
 Publishable key 5
 purchasable items, define 15

Q

query-string-object 4, 5

R

React app, setup 5
 React app, Webshop 4
 React form 8
 react props 5
 react root component 5
 React Stripe Elements 4
`react-stripe-elements` 5, 8
 recurring payments, webhook for 21
 restricted api keys 2
 root react component 5
 routes, API 11

S

Salesforce API	22
secret key	11
secret, webhook	22
security headers	11
send orders	17
separate order from payment	12
server code, skeleton	11
server, set up	11
Shop component	8, 13
Shop component, charge credit card	9
Shop object	13
Shop.js	6
signup	2
single click shopping	12
SKU, assign	15
SKU, define in Stripe	15
'Source'	12
Source, assigned to customers	21
Stamps API	22
state, implement Shop Component	13
static ip, ngrok	21
Stock Keeping Unit (SKU)	15
Stripe account, setup	2
Stripe API	13
Stripe API documentation, link to	2
Stripe Customer	12
Stripe Customer, benefit of using	12
Stripe Dashboard	4
Stripe dashboard	2
Stripe Elements	4
Stripe form element	8
Stripe object	5
Stripe object, as prop	8
Stripe payment methods	21
Stripe product integration	2
Stripe requests	4
Stripe signup	2
Stripe Webshop, create	1
Stripe, access from server	11

Stripe, in React form	8
Stripe, what it is	2
Stripe.js	4, 5
Stripe.js link	2
stripe.orders.create() function	11
stripe.orders.pay() function	12
stripe.webhooks.constructEvent() function ..	22
StripeProvider	5
StripeProvider component	5
submit button, disable	14
subscription service	21

T

testing	10, 18
token, from Stripe	4
tokenization, credit card data	4
tokenize, credit card	8
total price of orders, automatic calculation	17

U

UI, order handling	12
--------------------------	----

W

WARNING, raw credit card details	4
webhook	2
webhook for notification of payment	21
webhook for recurring payment	21
webhook secret	22
webhook, to process order	11
Webhooks Developers Dashboard	21
webhooks, set up	19
webhooks, usefulness	21
webshop	5
webshop creation	4
webshop products	4
Webshop, add products to	17

SOURCE INDEX

A

App.js..... 5

C

create-script..... 5

I

index.js..... 11, 12, 22

S

Shop.js..... 6, 8, 9, 13, 14, 17