



Expertise
and insight
for the future

Son Chu Hoang

Shopify Upsell App: Using Next.js, React.js to boost sale

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

10 October 2020

Author(s) Title	Son Chu Hoang Shopify Upsell App: Using Next.js, React.js to boost sale
Number of Pages Date	34 pages 10 October 2020
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Information Technology
Specialisation option	Software Engineer
Instructor(s)	Janne Salonen, Head of Department
<p>This paper covers the build of a Shopify upsell app, with a purpose of increasing sales, specifically AOV (Average order value). The app is integrated and used by an ecommerce sneaker brand, Rens. Rens is making 700,000 euros in 2019 and looking forward to making 1 million euros in by the end of 2020. With the installment of the Shopify Upsell App, Rens seek to reach that goal faster.</p>	
Keywords	NextJs, ReactJS, Javascript, Shopify, eCommerce, Upsell

Contents


1. Introduction	1
1.1. Introduction to upsell in ecommerce	1
1.2. Introduction to Shopify	2
1.3. Introduction to Rens	2
1.4. Introduction to Shopify Upsell App	3
2. Problems and Solutions	3
2.1. Problems	3
2.2. Solutions	3
3. Core Technologies	4
3.1. Program Languages	4
3.2. Core Libraries	4
3.2.1. Node.js	5
3.2.2. Next.js	5
3.2.3. React	5
3.2.4. @shopify/polaris	5
3.2.5. @shopify/app-bridge-react	5
3.2.6. handlebars	5
3.2.7. dotenv	5
3.2.8. ngrok	5
4. Design and Implementation	6
4.1. Designs	6
4.1.1. Merchant UI	6
4.1.2. Upselling Section on the Shopify store	7
4.2. Create Upsell App	9
4.2.1. Basic Server Setup	9
4.2.2. Set up Upsell app on Shopify Partner	11
4.2.3. UI Interface of Upsell Up for Merchants	13
4.2.5. Modifying merchant's store	19
References	29
Appendices	30
Appendix 1. package.json	30

1. Introduction

1.1. Introduction to upsell in ecommerce

According to [1] Investopia, upselling is a sale technique where company's staff, merchants or sellers offer customers another product, an upgrade, a more expensive version of what they're buying, in order to increase the basket value of their purchase. Upselling is very commonly seen in online retail stores. Take Amazon for example, after a product is added to cart, they are shown other products that customers who bought the same item usually buy with, or sometimes, product bundles, which offer other items but save customers monetary value.

Frequently bought together

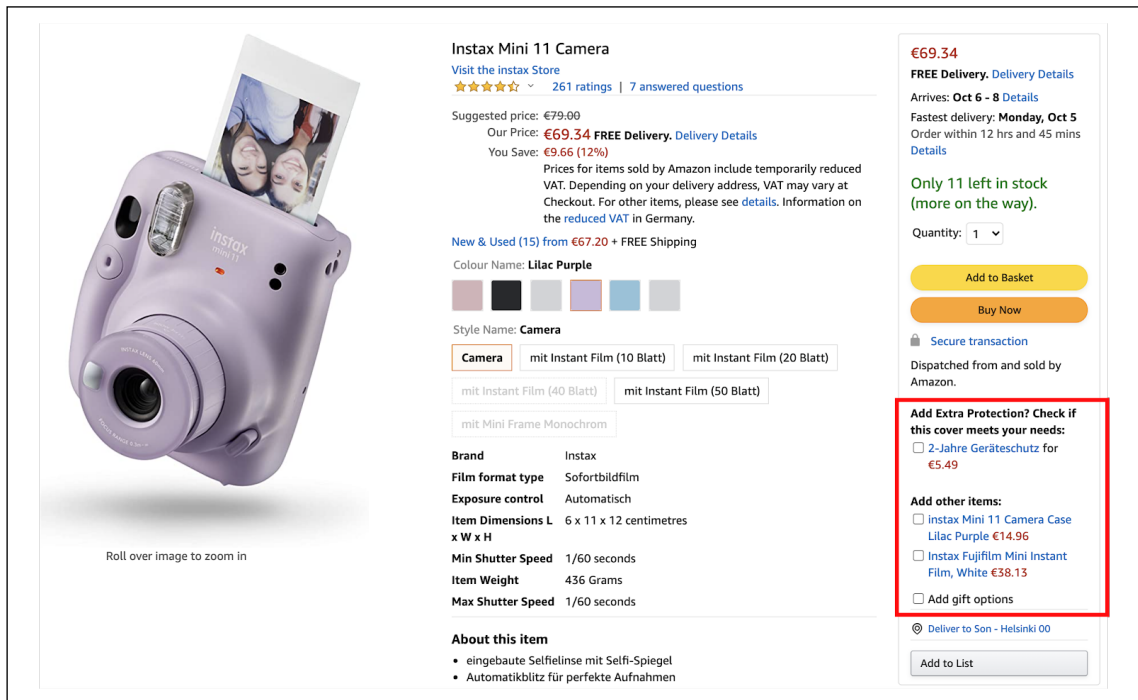


Total price: **€98.86**
[Add all three to Basket](#)

i Some of these items are dispatched sooner than the others. [Show details](#)

- This item:** Instax Mini 11 Camera **€69.34**
- Instax Fujifilm Mini Instant Film, White, 2 x 10 Sheets (20 Sheets) **€14.56**
- instax Mini 11 Camera Case Lilac Purple **€14.96**

Figure 1. Amazon bundle offer



Instax Mini 11 Camera
 Visit the instax Store
 ★★★★★ 261 ratings | 7 answered questions

Suggested price: €79.00
 Our Price: **€69.34** FREE Delivery. Delivery Details
 You Save: €9.66 (12%)

Prices for items sold by Amazon include temporarily reduced VAT. Depending on your delivery address, VAT may vary at Checkout. For other items, please see details. Information on the reduced VAT in Germany.

New & Used (15) from €67.20 + FREE Shipping

Colour Name: **Lilac Purple**

Style Name: **Camera**

Camera mit Instant Film (10 Blatt) mit Instant Film (20 Blatt)
 mit Instant Film (40 Blatt) mit Instant Film (50 Blatt)
 mit Mini Frame Monochrom

Brand: Instax
 Film format type: Sofortbildfilm
 Exposure control: Automatisch
 Item Dimensions L x W x H: 6 x 11 x 12 centimetres
 Min Shutter Speed: 1/60 seconds
 Item Weight: 436 Grams
 Max Shutter Speed: 1/60 seconds

About this item

- eingebaute Selfielinse mit Selfi-Spiegel
- Automatikblitz für perfekte Aufnahmen

€69.34
 FREE Delivery. Delivery Details
 Arrives: **Oct 6 - 8** Details
 Fastest delivery: **Monday, Oct 5**
 Order within 12 hrs and 45 mins
 Details

Only 11 left in stock (more on the way).

Quantity: 1

Add to Basket
 Buy Now

Secure transaction
 Dispatched from and sold by Amazon.

Add Extra Protection? Check if this cover meets your needs:

2-Jahre Geräteschutz for €5.49

Add other items:

instax Mini 11 Camera Case Lilac Purple €14.96
 Instax Fujifilm Mini Instant Film, White €38.13
 Add gift options

Deliver to Son - Helsinki 00
 Add to List

Figure 2. Amazon upselling extra items to go with the main product

1.2. Introduction to Shopify

[2] Shopify is an e-commerce online service that helps merchants launch their online businesses in a fast, reliable and scalable way. Until now, there have been over 1 million businesses powered by Shopify and Rens is no exception.

1.3. Introduction to Rens

[3] Founded by 2 founders, Hoang Son Chu and Bao Khanh Tran, in Helsinki, Finland in 2019, Rens aimed to make athleisure sneakers from eco-friendly materials, particularly used coffee grounds and recycled plastic. Rens's demographic is from 20 to 35 years old, living in the US, UK, Finland and Germany.

Main revenue channel of Rens is through online eCommerce Direct-To-Consumer store and the company is currently hosting its web store powered by Shopify.



Figure 3. Rens Product

1.4. Introduction to Shopify Upsell App

Shopify Upsell App allows merchants to pick their upsell strategy to optimize sales.

2. Problems and Solutions

2.1. Problems

Though Shopify provides different discount types, it natively doesn't provide options to post upsell offers on merchants' store. In order for Rens to boost sales, it wants to up-sell 1 extra pair of shoes every time a customer adds 1 product to their cart.

2.2. Solutions

Rens will install Shopify Upsell App and through the app, Rens can select what products to upsell once customers have added at least one item in their cart. The upsell suggestion will be placed at the bottom of the cart drawer.

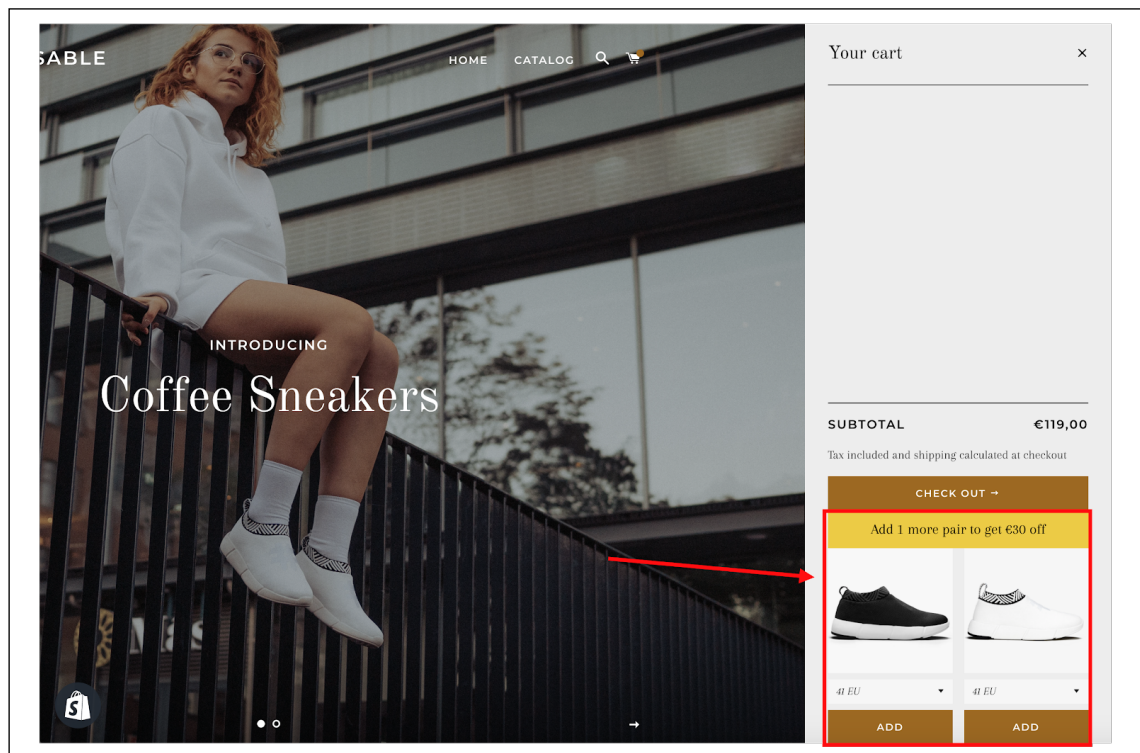


Figure 4. Upsell placement on a Shopify store

3. Core Technologies

This section provides insights into the techniques and technologies used during the development of Shopify Upsell App.

3.1. Program Languages

Shopify Upsell App is primarily a web app and written in Javascript, abbreviated as JS.

Javascript is initially used on web browsers, but it has been embedded in servers, usually via NodeJS. Javascript is commonly used with CSS and HTML.

3.2. Core Libraries

This section provides a brief description about different libraries and frameworks used during the development of Shopify Upsell App.

3.2.1. Node.js

Node.js is an event-driven and asynchronous JavaScript runtime environment built on top of Chrome V8 engine. Node.js compiles JavaScript to machine code in order to build the back-end in JavaScript. [4]

3.2.2. Next.js

Next.js is a React framework with a structure that allows you to build a frontend React application, and transparently handles server-side rendering. It solves the problem of React apps, which commonly load the content after javascript is loaded, resulting in poor SEO and customer experience. Next.js provides server rendering, allowing the app to display its initial state before any javascript script is loaded. [5]

3.2.3. React

React is a javascript library for building user interfaces. It allows developers to design simple views for each state and React will efficiently update components when data changes. [6]

3.2.4. @shopify/polaris

Polaris is a CSS Library with premade layout and components that goes with the Shopify design styling system. [7]

3.2.5. @shopify/app-bridge-react

Shopify App Bridge offers React component wrappers for some App Bridge actions. [8]

3.2.6. handlebars

Handlebars is a javascript library to build semantic templates effectively.

3.2.7. dotenv

Dotenv is a zero-dependency module that loads environment variables from a .env file into process

3.2.8. ngrok

ngrok secure introspectable tunnels to localhost webhook development tool and debugging tool.

4. Design and Implementation

This section provides the details about the development of the application including UI design for both the merchants and the customers.

4.1. Designs

4.1.1. Merchant UI

The Upsell App will have mainly 3 states:

State 1 - Initial state: This is when customers haven't selected any products to upsell, which will be the default state of the app.

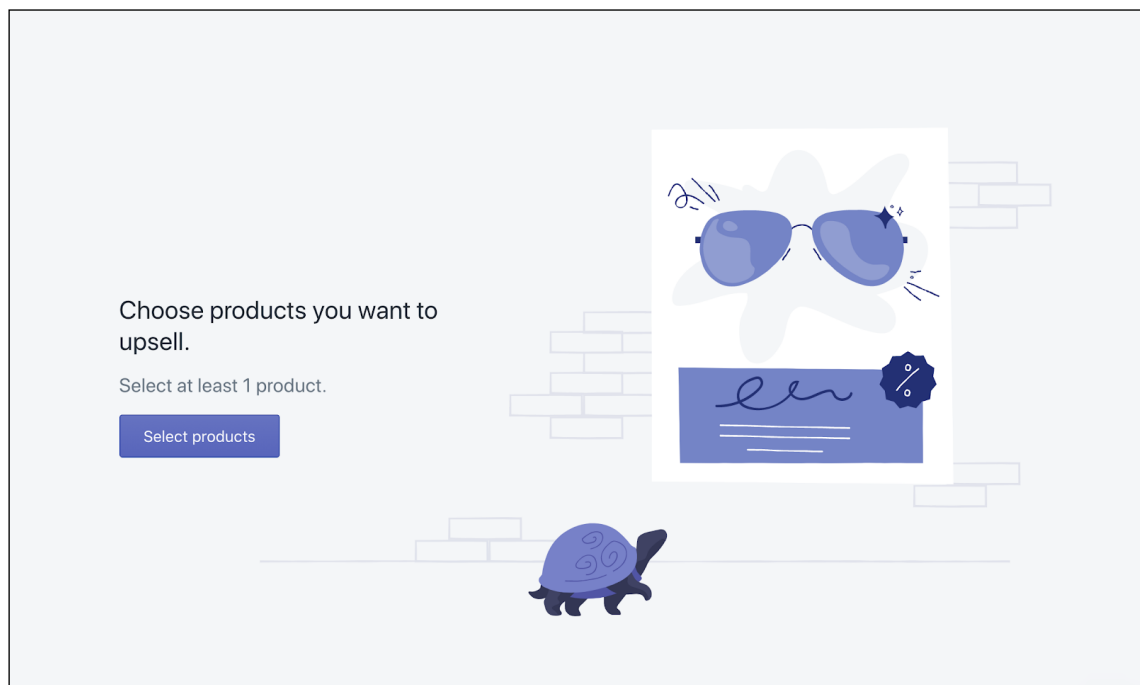


Figure 5. Upsell App initial state

State 2 - Select product state: This is when customers will have to select the products they would like to offer upsell.

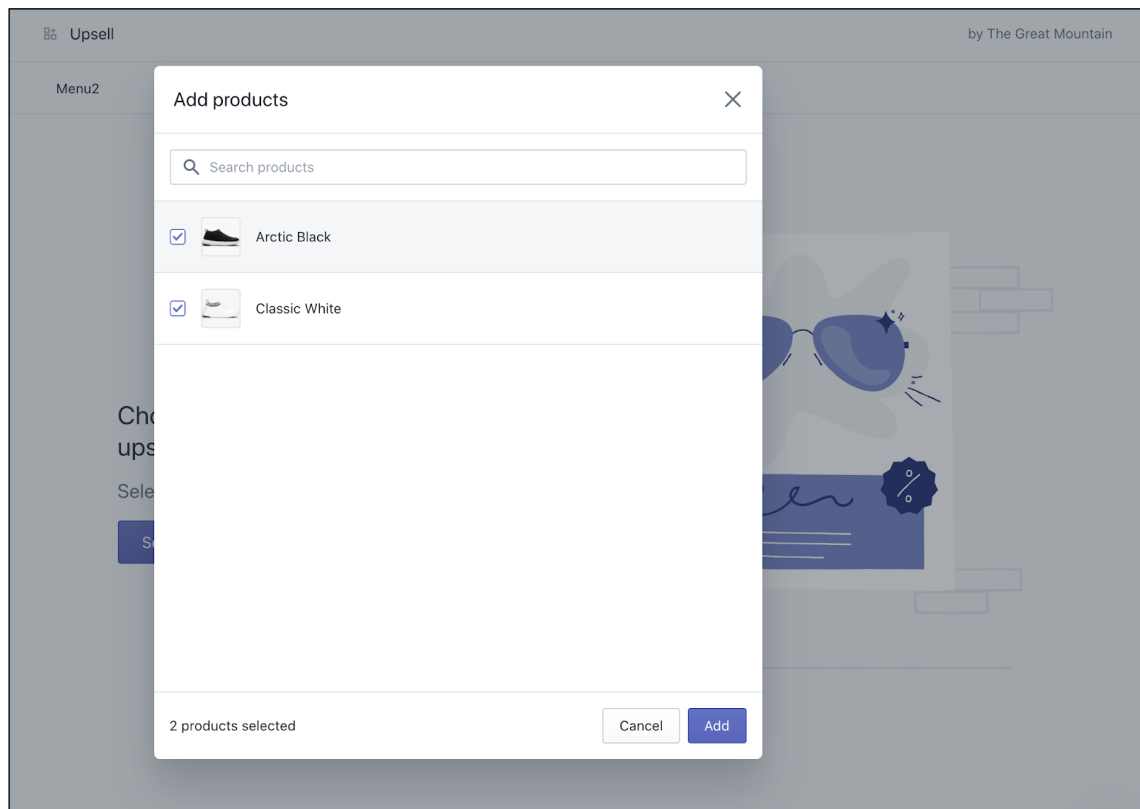


Figure 6. Select product state

State 3 - Publish state: This is when customers have selected at least 1 product to offer upsell.

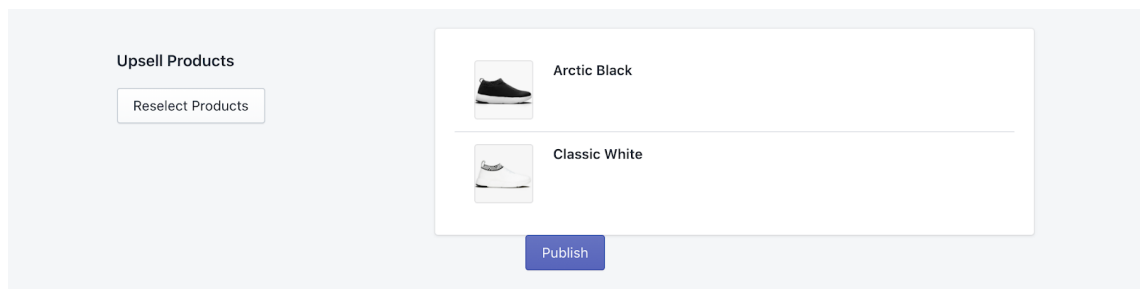


Figure 7. Publish state

4.1.2. Upselling Section on the Shopify store

After the merchants have selected and published the upselling products, when a customer adds 1 product to their cart and opens it, the upselling section will be shown to them at the bottom of the cart drawer.

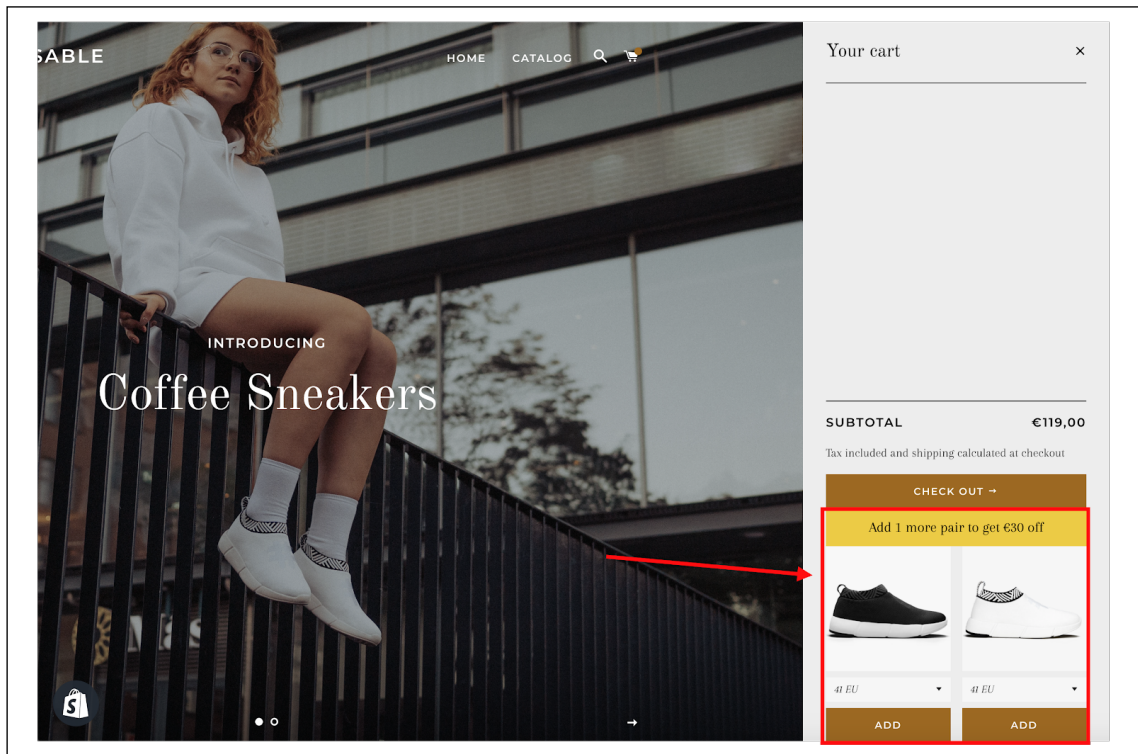


Figure 8. Upselling section on merchant's store

And once they click button ADD, the selected upsell product will automatically inserted to their cart and the corresponding discount will be applied.

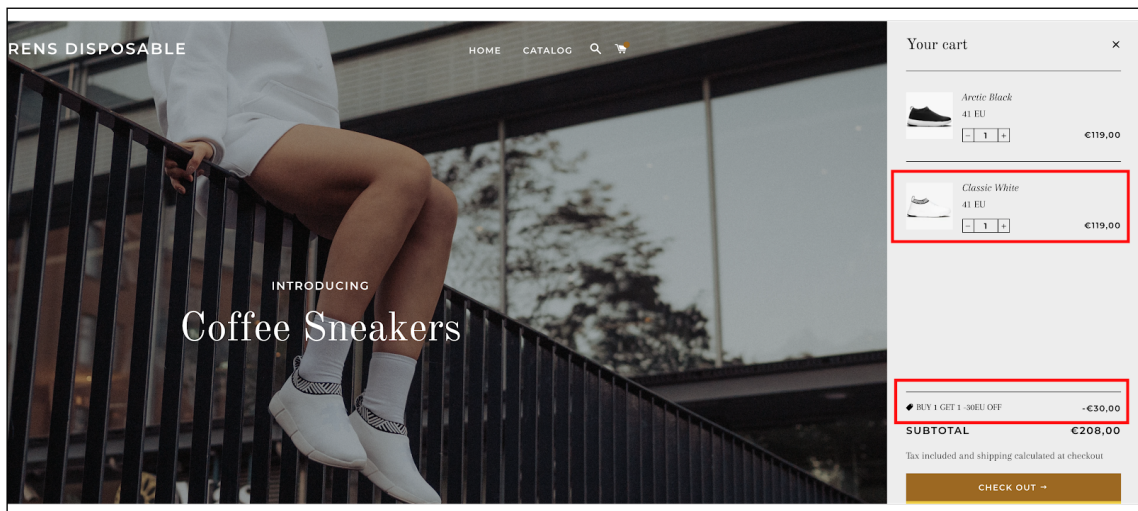


Figure 9. After customer select an upselling product

4.2. Create Upsell App

4.2.1. Basic Server Setup

Bootstrapped with NextJs and Koa, package.json is setup according to Appendix 1.

```

1 // server.js
2 const dotenv = require('dotenv');
3 const Koa = require('koa');
4 const next = require('next');
5 const { default: createShopifyAuth } = require('@shopify/koa-shopify-auth');
6 const { verifyRequest } = require('@shopify/koa-shopify-auth');
7 const { ApiVersion } = require('@shopify/koa-shopify-graphql-proxy');
8 const { default: graphqlProxy } = require('@shopify/koa-shopify-graphql-proxy');
9
10 const session = require('koa-session');
11
12 dotenv.config();
13
14 const port = parseInt(process.env.PORT, 10) || 3000;
15 const dev = process.env.NODE_ENV !== 'production';
16 const app = next({ dev });
17 const handle = app.getRequestHandler();
18
19 const { SHOPIFY_API_SECRET_KEY, SHOPIFY_API_KEY } = process.env;
20
21 app.prepare().then(() => {
22   const server = new Koa();
23   server.use(session({ secure: true, sameSite: 'none' }, server));
24   server.keys = [SHOPIFY_API_SECRET_KEY];
25
26   server.use(
27     createShopifyAuth({
28       apiKey: SHOPIFY_API_KEY,
29       secret: SHOPIFY_API_SECRET_KEY,
30       scopes: ['read_products', 'read_themes', 'write_themes'],
31       afterAuth(ctx) {
32         const { shop, accessToken } = ctx.session;
33         ctx.redirect('/');
34       },
35     }),
36   );
37
38   server.use(graphqlProxy({version: ApiVersion.October19}))
39   server.use(verifyRequest());
40   server.use(async (ctx) => {
41     await handle(ctx.req, ctx.res);
42     ctx.respond = false;
43     ctx.res.statusCode = 200;
44     return
45   });
46
47   server.listen(port, () => {
48     console.log(`> Ready on http://localhost:${port}`);
49   });
50 });

```

Figure 10. Server.js

When registering a private Shopify app, Shopify will give developers a `API_KEY` and a `API_SECRET_KEY` string. They are used to authenticate the app for the requested access right to merchants' stores.

```

1 SHOPIFY_API_KEY=f975581cd4e5194f3fdbe36d787e6c5e
2 SHOPIFY_API_SECRET_KEY=shps_1a8cea48aa18007fe143d054b3d06e09

```

Figure 11. .env

Being stored in .env file, while using `dotenv.config()` in `server.js`, `SHOPIFY_API_KEY` and `SHOPIFY_API_SECRET_KEY` can be retrieved as environment variables, particularly `process.env.SHOPIFY_API_KEY` and `process.env.SHOPIFY_API_SECRET_KEY`.

In Upsell App, it needs 3 access rights from the merchants:

1. `read_products`: to be able to retrieve product information from the app. The app will then use that information to display upselling product options for users to choose from.
2. `read_themes`: Upsell App updates merchant's theme, therefore, needs this right to properly modify the theme.
3. `write_themes`: Upsell App modify merchant's theme to inject component that offer the upsell to merchant's customers

These 3 rights are defined in the `scope` property in `createShopifyAuth` function

```

1 createShopifyAuth({
2   apiKey: SHOPIFY_API_KEY,
3   secret: SHOPIFY_API_SECRET_KEY,
4   scopes: ['read_products', 'read_themes', 'write_themes'],
5   // ...
6 })

```

Figure 12. Required fields of `createShopifyAuth`

4.2.2. Basic Frontend Setup

With NextJS, to define what to display in the root page of the app, simply create a React component in `pages/index.js`.

```

1 // pages/index.js
2 const Index = () => (
3   <div>
4     <p>Sample app using React and Next.js</p>
5   </div>
6 );
7 export default Index;
8 |

```

Figure 13. pages/index.js - First React UI Component

4.2.3. Set up Upsell app on Shopify Partner

To be able to run our app on Shopify Partner Platform, ngrok is needed. Ngrok will tunnel securely to the developer's localhost, making it public for internet users.

Running `npm run dev` will run the application at port 3000. Then run `ngrok http 3000` will tunnel the application from local to a random domain.

```

Session Status      online
Account             son.chu@rensooriginal.com (Plan: Free)
Version             2.3.35
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://690e291a7bbe.ngrok.io -> http://localhost:3000
Forwarding           https://690e291a7bbe.ngrok.io -> http://localhost:3000

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00  0.00  0.00  0.00

```

Figure 14. Result of running `ngrok http 3000`

In the setting of Upsell App in Shopify, the forwarding address is entered in App URL and Allowed Redirection URL(s):

← Upsell

App setup

App information

This information will be used to identify your app.

App name
Upsell

API contact email
son.chu@thegreatmountain.com

We'll use this email to contact you about API issues (such as Webhook failures).

URLs

Shopify merchants access your app through the URLs you specify here.

App URL
https://4d0e8d05de7a.ngrok.io/

Preferences URL (optional)
e.g. https://example.com/preferences

Allowed redirection URL(s)
https://4d0e8d05de7a.ngrok.io/auth/callback

You must include at least one redirect URL before making your app public. Merchants are redirected to these allowed URLs after the app is installed. [Learn more about redirection URLs.](#)

Figure 15. Upsell App Setup

In this paper, a Shopify development store, <https://rens-disposable.myshopify.com>, is created and installs the Upsell App. What merchants can see from the app now is this:

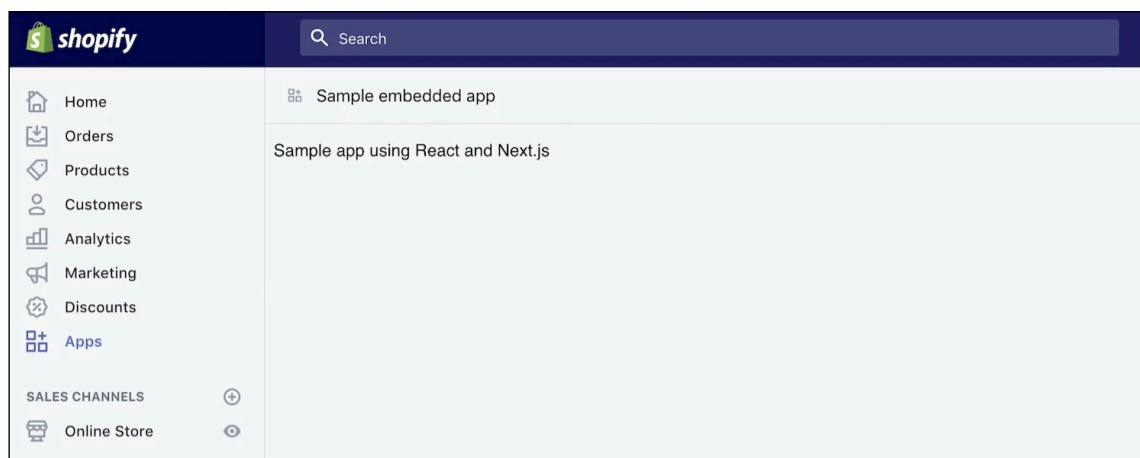


Figure 16. Upsell App with a simple view

4.2.4. UI Interface of Upsell Up for Merchants

By default, the merchants will have 0 products offering for upsell. Upsell will initially display an empty state with a button for users to click. To do this, we will use React. NextJS framework allows developers to render pages written with React. By creating a React component in pages/index.js, it will be rendered at the root page of the app. See the content of the file in Appendix 2.

When creating a Shopify app, it's very important to create a feel and look like a Shopify interface so that merchants would quickly get familiar with the tool. Shopify Polaris is a CSS library that provides pre-made styles following the Shopify styling guidelines.

To apply polaris styles, we create pages/_app.js. NextJs will use this file to define a React component, which plays a wrapper containing individual react components of each page.

```
1 // pages/_app.js
2 import App from 'next/app';
3 import Head from 'next/head';
4
5 class MyApp extends App {
6   render() {
7     const { Component, pageProps } = this.props;
8     return (
9       <React.Fragment>
10         <Head>
11           <title>Sample App</title>
12           <meta charSet="utf-8" />
13         </Head>
14         <Component {...pageProps} />
15       </React.Fragment>
16     );
17   }
18 }
19
20 export default MyApp;
21
```

Figure 17. pages/_app.js - Wrapper component of the whole React app

The hereby additional code will apply polaris styling to the Upsell App.

```

1  // pages/_app.js
2  import App from 'next/app';
3  import Head from 'next/head';
4  import { AppProvider } from '@shopify/polaris';
5  import '@shopify/polaris/dist/styles.css';
6
7  class MyApp extends App {
8    render() {
9      const { Component, pageProps } = this.props;
10     return (
11       <React.Fragment>
12         <Head>
13           <title>Sample App</title>
14           <meta charSet="utf-8" />
15         </Head>
16         <AppProvider>
17           <Component {...pageProps} />
18         </AppProvider>
19       </React.Fragment>
20     );
21   }
22 }
23
24 export default MyApp;

```

Figure 18. pages/_app.js with polaris integration

The layout of the app should now look like this:

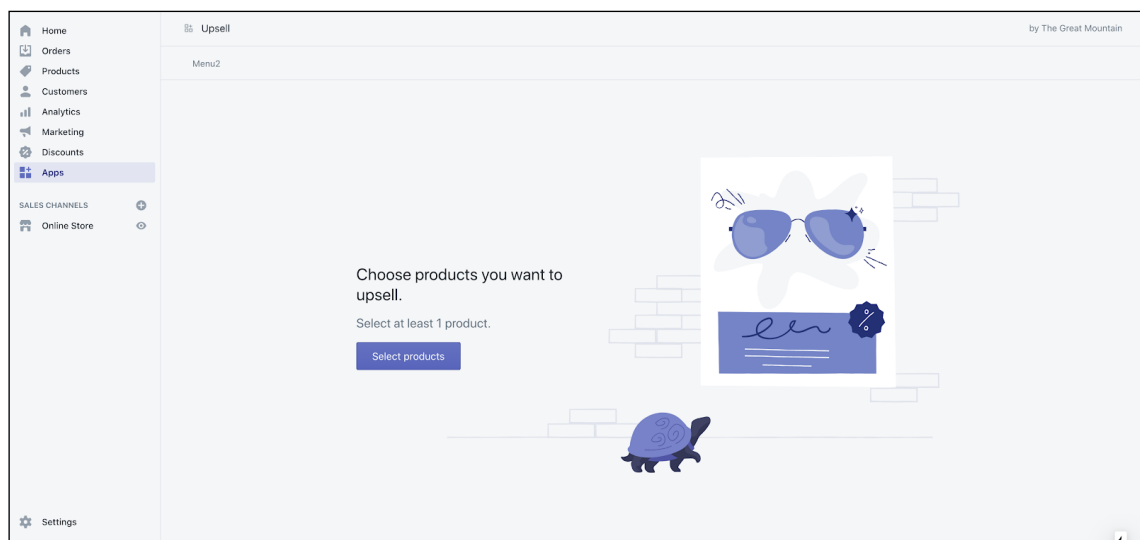


Figure 19. Upsell App initial layout

However, clicking the Select products button doesn't do anything yet. The app will need a bridge between the frontend and the merchant's shop information, like their products. `@shopify/app-bridge-react` will then be used for that purpose.

Shopify App Bridge is a JavaScript library that seamlessly integrates your app into Shopify user interfaces, including the web admin, mobile app, and POS. In this tutorial we've focused primarily on the web admin, but App Bridge will ensure your app name, logo, and navigation menu appears reliably across all of Shopify's interfaces. Keeping your look and feel consistent with Shopify's UI also makes it faster and easier for merchants to start using your app. When building with React, you can use the Shopify App Bridge React library to initialize the library by passing your app's Shopify API Key and the shop origin to the App Bridge Provider component.

To use Shopify App Bridge, in `server.js`, we need to share the shop's origin via cookie.

```
1 // server.js
2 server.use(
3   createShopifyAuth({
4     apiKey: SHOPIFY_API_KEY,
5     secret: SHOPIFY_API_SECRET_KEY,
6     scopes: ["read_products", "read_themes", "write_themes"],
7     afterAuth(ctx) {
8       const { shop } = ctx.session;
9       ctx.cookies.set("shopOrigin", shop, {
10        httpOnly: false,
11        secure: true,
12        sameSite: "none",
13      });
14       ctx.redirect("/");
15     },
16   })
17 );
```

Figure 20. Set up Shopify App Bridge in `server.js`

In `_app.js`, the hereby code will integrate Shopify App Bridge to our frontend

```

1 // pages/_app.js
2 import App from "next/app";
3 import Head from "next/head";
4 import Cookies from "js-cookie";
5 import { Provider } from "@shopify/app-bridge-react";
6
7 class MyApp extends App {
8   render() {
9     const { Component, pageProps } = this.props;
10    const config = {
11      apiKey: API_KEY,
12      shopOrigin: Cookies.get("shopOrigin"),
13      forceRedirect: true,
14    };
15    return (
16      <React.Fragment>
17        <Head>
18          <title>Sample App</title>
19          <meta charSet="utf-8" />
20        </Head>
21        <Provider config={config}>
22          <AppProvider>
23            <Component {...pageProps} />
24          </AppProvider>
25        </Provider>
26      </React.Fragment>
27    );
28  }
29 }
30 export default MyApp;

```

Figure 21. Set up Shopify App Bridge in pages/_app.js

Now that we have successfully integrated Shopify App Bridge, we should be able to have access to merchant's shop information, especially their products. Up next, Upsell App will allow merchants to select which products to offer upsell to. ResourcePicker will be used to serve that purpose.

```

1 // pages/index.js
2 import { ResourcePicker } from '@shopify/app-bridge-react';
3
4 // ...
5 <Page>
6   <ResourcePicker
7     resourceType="Product"
8     showVariants={false}
9     open={this.state.open}
10    onSelection={(resources) => { /* Handling selected products */ }}
11    onCancel={() => this.setState({ open: false })}
12  />

```

Figure 22. Use ResourcePicker from Shopify App Bridge in pages/index.js

As seen, ResourcePicker will only be displayed when the value of its open attribute is true, and that relies on this.state.open. In <EmptyState> component's action value, when EmptyState button is clicked, it will trigger onAction handler and make this.state.open true, therefore, open ResourcePicker. The UI then will be shown as the following:

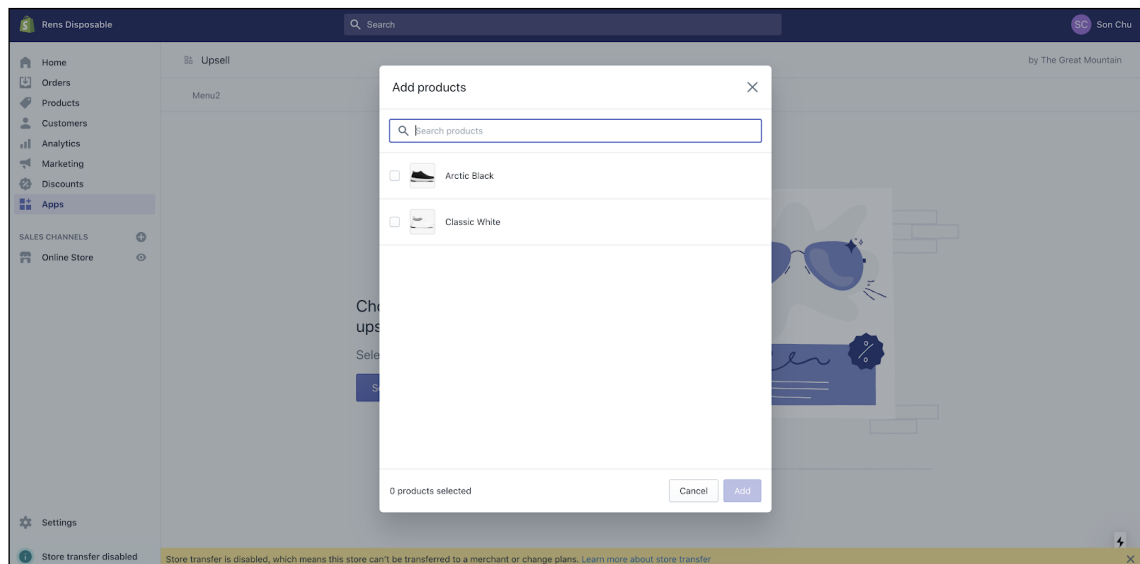


Figure 23. ResourcePicker UI

Some crucial steps to implement this UI are:

1. Only show <EmptyState> when this.state.selectedProducts is empty. Therefore, the following code will be added.

```
{!selectedProducts.length &&
```

```
<EmptyState ...
```

2. To display products once selected, the app will utilize a number of Polaris's components: Layout.AnnotatedSection, Card, ResourceList, ResourceItem. Next to <EmptyState>, as the following

```

1
2  !!selectedProducts.length && (
3
4    <Layout.AnnotatedSection
5      title="Upsell Products"
6      description={
7        <Button onClick={() => this.setState({ open: true })}>
8          Reselect Products
9        </Button>
10     }
11   >
12     <Card sectioned>
13       <ResourceList
14         resourceName={{ singular: "product", plural: "products" }}
15         items={selectedProducts}
16         renderItem={(item) => (
17           <ResourceItem
18             id={item.id}
19             media={
20               <Thumbnail
21                 alt={item.title}
22                 size="medium"
23                 source={item.images[0].originalSrc}
24               />
25             }
26             accessibilityLabel={`View details for ${item.title}`}
27             name={item.title}
28           >
29             <h3>
30               <TextStyle variation="strong">{item.title}</TextStyle>
31             </h3>
32           </ResourceItem>
33         )}
34       />
35     </Card>
36   </Layout.AnnotatedSection>
37   <Button primary onClick={this.publish}>
38     Publish
39   </Button>
40 </>
41 );
42

```

Figure 24. Define components in the page in Index component

The result display is like the following screenshot:

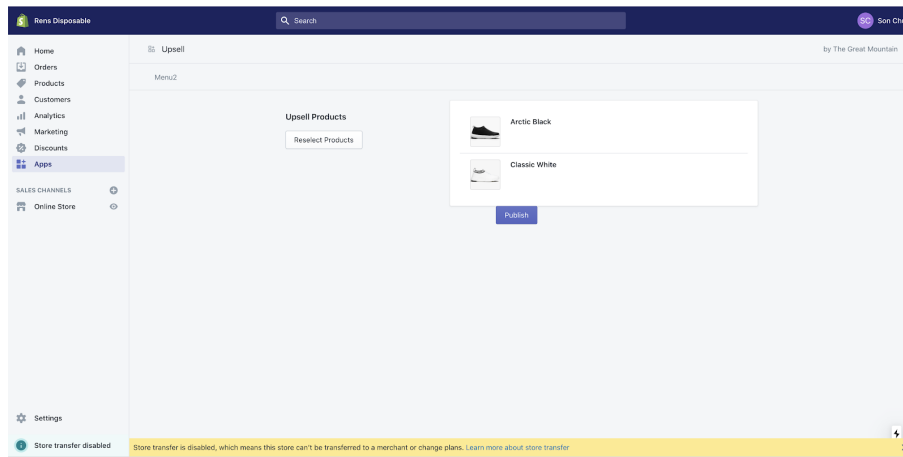


Figure 25. Screen after upsell products are selected

This is as far as the user interface of the Upsell App needs to be. From here, merchants would expect that after selecting the products and clicking the Add button, the upsell section will appear in their website's cart drawer.

4.2.5. Modifying merchant's store

To be able to modify merchant's store, the Upsell App must be granted `read_themes` and `write_themes` access, which has been defined in the scopes of shopify authentication in `server.js`

```

1 // server.js
2 // ...
3 server.use(
4   createShopifyAuth({
5     apiKey: SHOPIFY_API_KEY,
6     secret: SHOPIFY_API_SECRET_KEY,
7     scopes: ['read_products', 'read_themes', 'write_themes'],
8     afterAuth(ctx) {
9     // ...
10

```

Figure 26. Required scopes for the app to work defined in `createShopifyAuth` in `server.js`

After the user is authenticated, Upsell App server will receive `accessToken` via `ctx.session` in `afterAuth(ctx)` callback. It will then be stored in the cookie to be easily shared to all routes' handlers.

```

1 // server.js
2 // ...
3   afterAuth(ctx) {
4     const { shop, accessToken } = ctx.session;
5     ctx.cookies.set('shopOrigin', shop, {
6       httpOnly: false,
7       secure: true,
8       sameSite: 'none'
9     });
10    ctx.cookies.set('accessToken', accessToken, {
11      httpOnly: false,
12      secure: true,
13      sameSite: 'none'
14    });
15
16    ctx.redirect('/');
17  },
18 // ...

```

Figure 27. Share shopOrigin and accessToken across the whole application

Next.js's dynamic routes will be applied here to build a proxy to Shopify Admin APIs. For example, making a request to <https://rens-disposable.myshopify.com/admin/apps/upsell-57/api/shopify/themes/112204841120/assets.json> will be equivalent to make a request to <https://rens-disposable.myshopify.com/admin/api/2020-07/themes/112204841120/assets.json>. The only difference is that when making a request to <https://rens-disposable.myshopify.com/admin/api/2020-07/...>, it will attach the accessToken in the request's headers.

To implement the logic of dynamic routes or redirecting, in `pages/api/shopify/[...slug].js`, the app will have the code as the following:

```

1 // pages/api/shopify/[...slug].js
2 const axios = require('axios');
3 const url = require('url');
4 const path = require('path');
5
6 export default function handler(req, res) {
7   const body = req.body;
8   const shopifyPath = req.url.replace('api/shopify/', '');
9   const shopifyUrl = `https://rens-disposable.myshopify.com`;
10
11   const accessToken = req.cookies.accessToken;
12   const requestUrl = (new url.URL(path.join('/admin/api/2020-07', shopifyPath),
13     shopifyUrl)).href;
14
15   axios({
16     url: requestUrl,
17     method: req.method,
18     data: req.body,
19     headers: {
20       'X-Shopify-Access-Token': accessToken
21     }
22   }).then(({ data }) => {
23     res.status(200).json(data)
24   }).catch((error) => {
25     res.status(400).send(error)
26   })
27 }

```

Figure 28. Set up Shopify App Bridge in server.js

To test if the redirection works, open <https://rens-disposable.myshopify.com/admin/apps/upsell-57/api/shopify/themes/112204841120/assets.json> in the browser, the merchant will then see:

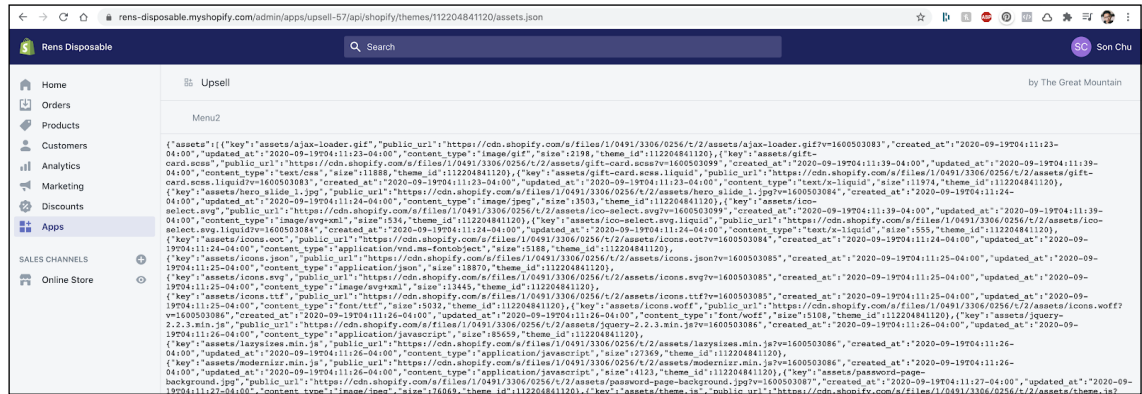


Figure 29. Result screen of <https://rens-disposable.myshopify.com/admin/apps/upsell-57/api/shopify/themes/112204841120/assets.json>

With the redirection working, the Upsell App can make requests from the frontend to Shopify Admin APIs without having to include the access token in the request's headers.

Upsell App will create a snippet in the merchant's published theme, containing the upselling section and its javascript to handle the event of adding an upselling item to the customer's cart.

components/upsell-in-cart.liquid.js is then created.

```

1  import { compile } from 'handlebars';
2
3  const upsellTemplate = `
4  <div id="upsell">
5    <div class="upsell-offer">
6      <p class="upsell-heading text-center">{{ upsellHeading }}</p>
7      <p class="upsell-subheading text-center">{{ upsellSubheading }}</p>
8    </div>
9    <div class="inCartUpsaleProduct--container">
10     {{#upsellProducts}}
11     <div class="inCartUpsaleProduct">
12       
14       <select class="inCartUpsaleProduct--variantSelect" data-product-id="{{id}}">
15         {{#variants}}
16         <option value="{{id}}">{{ title }}</option>
17         {{/variants}}
18       </select>
19       <button type="button" data-product-id="{{id}}"
20         class="inCartUpsaleProduct--submitBtn btn--secondary btn--full">Add</button>
21     </div>
22     {{/upsellProducts}}
23   </div>
24   <script>
25     $('inCartUpsaleProduct--submitBtn').on('click', function() {
26       console.log({ button: this })
27       const productId = $(this).data('productId');
28       const variantId = $('select[data-product-id="'+productId+'"]').val().match(/\\d+/g)
29       [0];
30
31       $.post('/cart/add.js', {
32         items: [{
33           quantity: 1,
34           id: variantId,
35         }]
36       }).complete(() => {
37         console.log('ajaxCart load')
38         window.ajaxCart.load();
39       })
40     </script>
41     <style>
42     // Styles for section layout
43     </style>
44 `
45 export default function generateUpsellSnippet(upsellProducts = []) {
46   const snippet = compile(upsellTemplate)({
47     upsellHeading: 'Add 1 more item',
48     upsellSubheading: '',
49     upsellProducts,
50     browseMoreUrl: '',
51     browseMoreUrlButtonText: 'Shop more'
52   })
53   return snippet;
54 }

```

Figure 30. components/upsell-in-cart.liquid.js

The syntax of the first highlighted block belongs to handlebars. The compile function from handlebars will render the template string `upsellTemplate` depending on variables appearing in the block. For example, `{{#upsellProducts}} ... {{/upsellProducts}}` will loop through array `upsellProducts` variable. Or `{{images.0.originalSrc}}` will interpolate an item of `upsellProducts`'s value of its `images[0].originalSrc`.

The second highlighted block is the snippet's javascript to handle the event of adding the selected upsell product to customers' carts. Notice that it makes requests to `/cart/add.js`, this belongs to Shopify Cart API and will add an item to the cart and activate various events in the background, for example, sending customers abandonment emails if they abandon their cart, etc... Shopify Cart API is universal and can be used in any Shopify theme.

In the callback of the request, `window.ajaxCart.load()` is called to refresh the customer's cart with the newly added item. The `window.ajaxCart.load` function, however, exists in the majority of Shopify's public themes, but not in all themes.

In the publish method in Index Component, it will do 2 things:

1. Modify `snippets/ajax-cart-template.liquid`, an existing snippet containing HTML of the theme's cart drawer, to include the snippet template `upsell-in-cart.liquid`, created by the Upsell App.
2. Create or update if exists `snippets/upsell-in-cart.liquid`

In order to do so, the app needs to know what theme it should update. The component will need to retrieve the published theme's id. The following code will do so:

```

1 // pages/index.js
2 class Index extends React.Component {
3   state = {
4     open: false,
5     selectedProducts: [],
6     themeId: ''
7   }
8
9   componentDidMount() {
10    axios.get(`/api/shopify/themes.json`)
11      .then(({ data }) => {
12        const themeId = data.themes.find((theme) => theme.role === 'main').id;
13        this.setState({ themeId });
14      })
15  }

```

Figure 31. Retrieve the live theme's ID

Now that **themeId** is retrieved, what method **publish** will do is the following:

1. Generate the template string for **snippets/upsell-in-cart.liquid** with the selected-Products injected in the template.

```

import generateUpsellSnippet from '../components/upsell-in-cart.liquid.js';
// ...
publish = () => {
  const { selectedProducts, themeId } = this.state;
  const newUpsellTemplate=(generateUpsellSnippet(selectedProducts));

```

Figure 32. Generate template for **snippets/upsell-in-cart.liquid**

2. Make request to Shopify Update Asset API to update **snippets/upsell-in-cart.liquid** in the live theme.

```

axios.put(`/api/shopify/themes/${themeId}/assets.json`, {
  "asset": {
    "key": "snippets/upsell-in-cart.liquid",
    "value": newUpsellTemplate
  }
})

```

Figure 33. Update snippets/upsell-in-cart.liquid

3. Retrieve snippets/ajax-cart-template.liquid

```

axios.get(`/api/shopify/themes/${themeId}/assets.json?asset[key]=snippets/
ajax-cart-template.liquid`)
  .then(({ data }) => {
    const originalTemplate = data.asset.value;

```

Figure 34. Retrieve snippets/ajax-cart-template.liquid

4. Make sure that a liquid snippet include exists in the right place, which is right before the ending `</form>` tag in `snippets/ajax-cart-template.liquid`.

If the template already contains the liquid snippet include `{% endraw %}{% include 'upsell-in-cart' %}{% raw %}`, the job here is done. The hereby code will implement so:

```
if (originalTemplate.includes(`{% include 'upsell-in-cart' %}`)) {
  return;
}
```

Figure 35. Retrieve the live theme's ID

5. If not, In `snippets/ajax-cart-template.liquid`, the file needs to have this exact code right before the closing tag `</form>`. Since what the app receive after the request to `/api/shopify/themes/${themeId}/assets.json?asset[key]=snippets/ajax-cart-template.liquid` is of type string, it's impossible to use html dom manipulation. Therefore, string queries manipulation and regular expression will be useful here.

`^<\div>[\n]+<\form>/g` is the regular expression that is used to detect the last closing div tag of any closing form tag. Noted that in most Shopify themes, `ajax-cart-template.liquid` only contain 1 form.

Assigned the regular expression to a variable, `regEx`, to find the the last closing div tag of the closing form tag, do `originalTemplate.match(regEx)[0]`. Concat with the liquid include string, ``{% endraw %}{% include 'upsell-in-cart' %}{% raw %}``, we will have a new string to replace the origin to form a whole new template for `snippets/ajax-cart-template.liquid`:

```
const regEx = /^<\div>[\n ]+<\form>/g;
const strToAppendToCartTemplate = `{% endraw %}{% include 'upsell-in-cart' %}{% raw %}`
const newTemplate = originalTemplate.replace(regEx, strToAppendToCartTemplate +
originalTemplate.match(regEx)[0])
axios.put(`/api/shopify/themes/${themeId}/assets.json`, {
  asset: {
    key: 'snippets/ajax-cart-template.liquid',
    value: newTemplate
  }
})
})
```

Figure 36. Use regular expression to check if the snippet is included in `ajax-cart-template.liquid`

And now, the Upsell App is completed and after the merchant hits the Publish button, it will create an upsell section in the cart drawer.

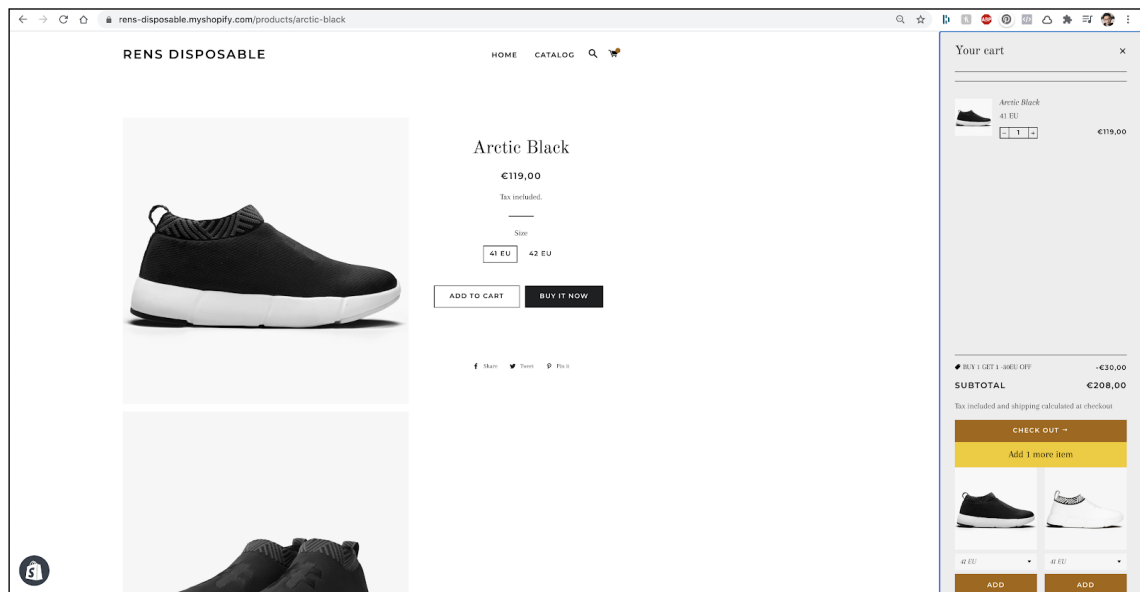


Figure 37. Merchant's store with the upselling section

5. Results

The upsell app has been installed on Rens store: rensoriginal.com since the beginning of September until the present (October 14, 2020), the brand witnessed a doubled amount of orders with 2 items.

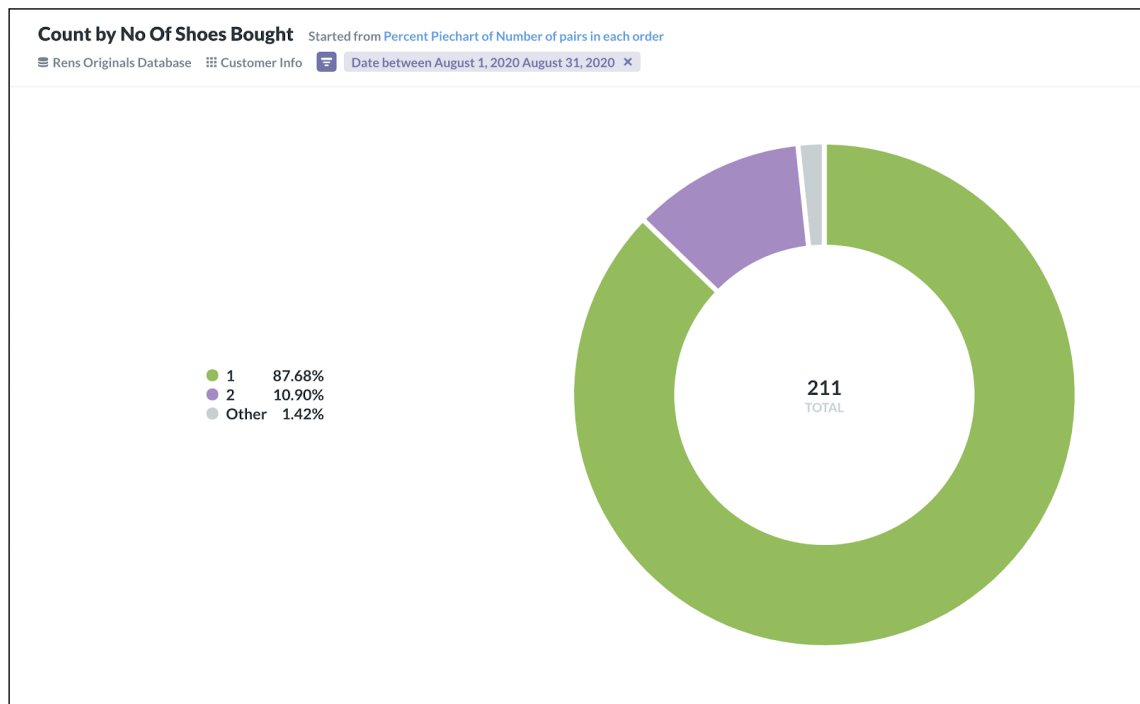


Figure 38. Pie chart of numbers of pairs in an order in August 2020

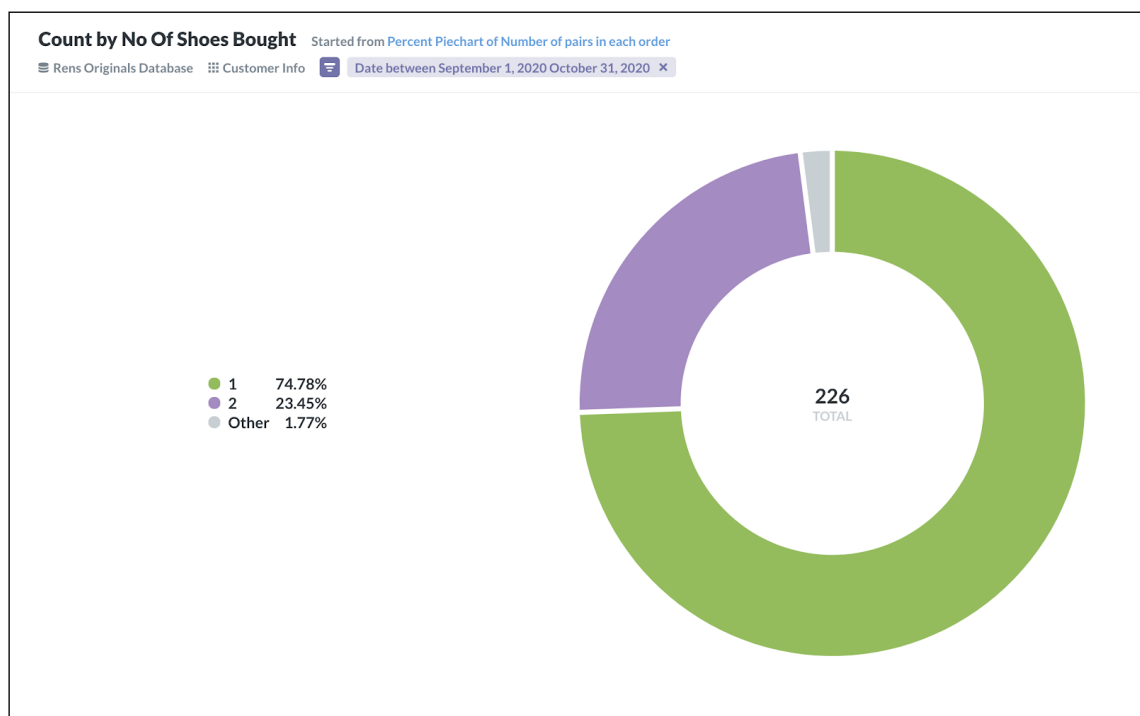


Figure 39. Pie chart of numbers of pairs in an order in September 2020

This has demonstrated that buyers are a lot more encouraged to buy multiple pairs at once when introduced with upsell items.

With the sale season coming up, especially Black Friday, Christmas and New Year, where consumers will be looking forward to bundle sales to prepare gifts for friends and families, Upselling App is proven to help Rens offer discounts for bundle, raise the chance of increasing the sales. And when Rens introduces new products, they can easily offer upsell items that go with the shoes, for example, offering socks while buying shoes, or offering track pants while buying hoodies.

6. Conclusion

The number of eCommerce businesses is increasing day by day and there are various ways to increase sales through websites. Upselling is only one of the tactics, but an easy one to gain significant revenue that is being missed out on by a lot of brands, especially small and medium ones. That's why there's a need for Upsell App to be introduced to the market, making it easier than ever for merchants to upsell products directly in customers' cart and increase their revenue.

For further development of the Upsell App, we could introduce more variations of combo discounts and different placements of where the discounts will be shown to merchants' stores. Those placements could be right on the product page, home page, or on the post-checkout page. The app could also be developed not only for the small and medium merchants but can be adapted by big retailers where they would have more than 20 SKUs in-store. Then the backend of the Upsell App could utilize more user data and personalize the upsell items based on customers' preferences. There's so much potential to what Upsell App can do and only the sky is the limit.

References

1. Suggestive Selling (Upselling) by Adam Hayes. Available from <https://www.investopedia.com/terms/s/suggestive-selling.asp>
2. Introduction to Shopify, by Shopify. Available from <https://www.shopify.com/about>
3. Rens's featured in Ilta-sanomat. URL: <https://www.is.fi/taloussanomat/art-2000006185488.html>. Accessed 10 Oct 2020.
4. Herron D. Node. Js Web Development. 4th edition. Birmingham: PACKT Publishing; 2018: 7-24
5. Next.js. URL: <https://nextjs.org>. Accessed 10 Oct 2020.
6. React.js. URL: <https://reactjs.org>. Accessed 10 Oct 2020.
7. Shopify polaris. URL: <https://polaris.shopify.com>. Accessed 10 Oct 2020.
8. Shopify App Bridge React. URL: <https://www.npmjs.com/package/@shopify/app-bridge-react>. Accessed 10 Oct 2020.

Appendices

Appendix 1. package.json

```
{
  "name": "upsell-metropolia",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "node server.js",
    "build": "next build",
    "start": "NODE_ENV=production node server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@shopify/app-bridge-react": "^1.27.2",
    "@shopify/koa-shopify-auth": "^3.1.70",
    "@shopify/koa-shopify-graphql-proxy": "^4.0.1",
    "@shopify/polaris": "^5.3.1",
    "@zeit/next-css": "^1.0.1",
    "apollo-boost": "^0.4.9",
    "axios": "^0.20.0",
    "dotenv": "^8.2.0",
    "graphql": "^15.3.0",
    "handlebars": "^4.7.6",
    "js-cookie": "^2.2.1",
    "koa": "^2.13.0",
    "koa-session": "^6.0.0",
    "next": "^9.5.3",
    "react": "^16.13.1",
    "react-apollo": "^3.1.5",
    "react-dom": "^16.13.1"
  }
}
```

Appendix 2. upsell-in-cart.liquid.js

```
import { EmptyState, Layout, Page, TextStyle, Card, Button, Resource-
List, ResourceItem, Thumbnail } from '@shopify/polaris';
import generateUpsellSnippet from '../components/upsell-in-cart.liq-
uid.js';
import axios from 'axios';

const img = 'https://cdn.shopify.com/s/files/1/0757/9955/files/empty-
state.svg';

class Index extends React.Component {
  state = {
    open: false,
    selectedProducts: []
  }

  render() {
    const { selectedProducts } = this.state;
    return (
      <Page>
        <Layout>
          {!selectedProducts.length &&
            <EmptyState
              heading="Choose products you want to upsell."
              action={{
                content: 'Select products',
                onAction: () => {
                  this.setState({ open: true })
                },
              }}
              image={img}
            >
            <p>Select at least 1 product.</p>
          </EmptyState>
        </Layout>
      </Page>
    )
  }
}
```

```
    }  
  }  
  
  export default Index;
```