A report of our app design and insight to the thought process

SEASIDE ANDROID/IOS REPORT

Report on React Native Version

Nathan Brathwaite, Zachary Aird, Samuel Lowe, Khareen Proverbs

Contents

[Introduction 2](#_Toc74861760)

[Purpose 2](#_Toc74861761)

[Background 2](#_Toc74861762)

[Project Overview 2](#_Toc74861763)

[Development Process 2](#_Toc74861764)

[Contributions 3](#_Toc74861765)

[Application Details 5](#_Toc74861766)

[Build Structure 5](#_Toc74861767)

[Components Folder 5](#_Toc74861768)

[Navigation Folder 6](#_Toc74861769)

[APIs 6](#_Toc74861770)

[Store Folder 8](#_Toc74861771)

[UI Folder 8](#_Toc74861772)

[Screens Folder 10](#_Toc74861773)

[Data Folder 11](#_Toc74861774)

[Database Management 14](#_Toc74861775)

[Carts 14](#_Toc74861776)

[Collections 15](#_Toc74861777)

[Users 17](#_Toc74861778)

[Application in Action 18](#_Toc74861779)

# Introduction

## Purpose

The purpose of this application was to facilitate an online storefront that would allow users to purchase a variety of different clothing items along with other features. This goal was to be achieved on both android and IOS devices. This was to be achieved with as little effort as possible and as such many APIs were implemented throughout the development cycle.

## Background

The team already had been exposed to android development and just recently exposed to IOS development. Though similar, android development and IOS development had unique elements to them. Since the team had been exposed to android development prior, members were more comfortable coding in that environment. It was decided here to utilize a multitude of APIs through React Native Framework to reduce development time and workload, as developers would not have to worry too much about the new syntax when it came to IOS development.

## Project Overview

### Development Process

The coding was based on a pre-existing set of code written by the team, a web application version of the system. It was this system that the design was based on, as it was to fulfil the same purpose, just in the form of a local application. For example, the firebase database that would be used in the application is the same one used in the pre-existing web app. Various, PNGs and other assets were ported from that system and brought to this one. Once the importing of the mandatory functionality of the web app was done, application-specific functionality was added to the android system. Once this was done the programmers that were more equipped to handle IOS were tasked with “translating” the android code to the necessary IOS code.

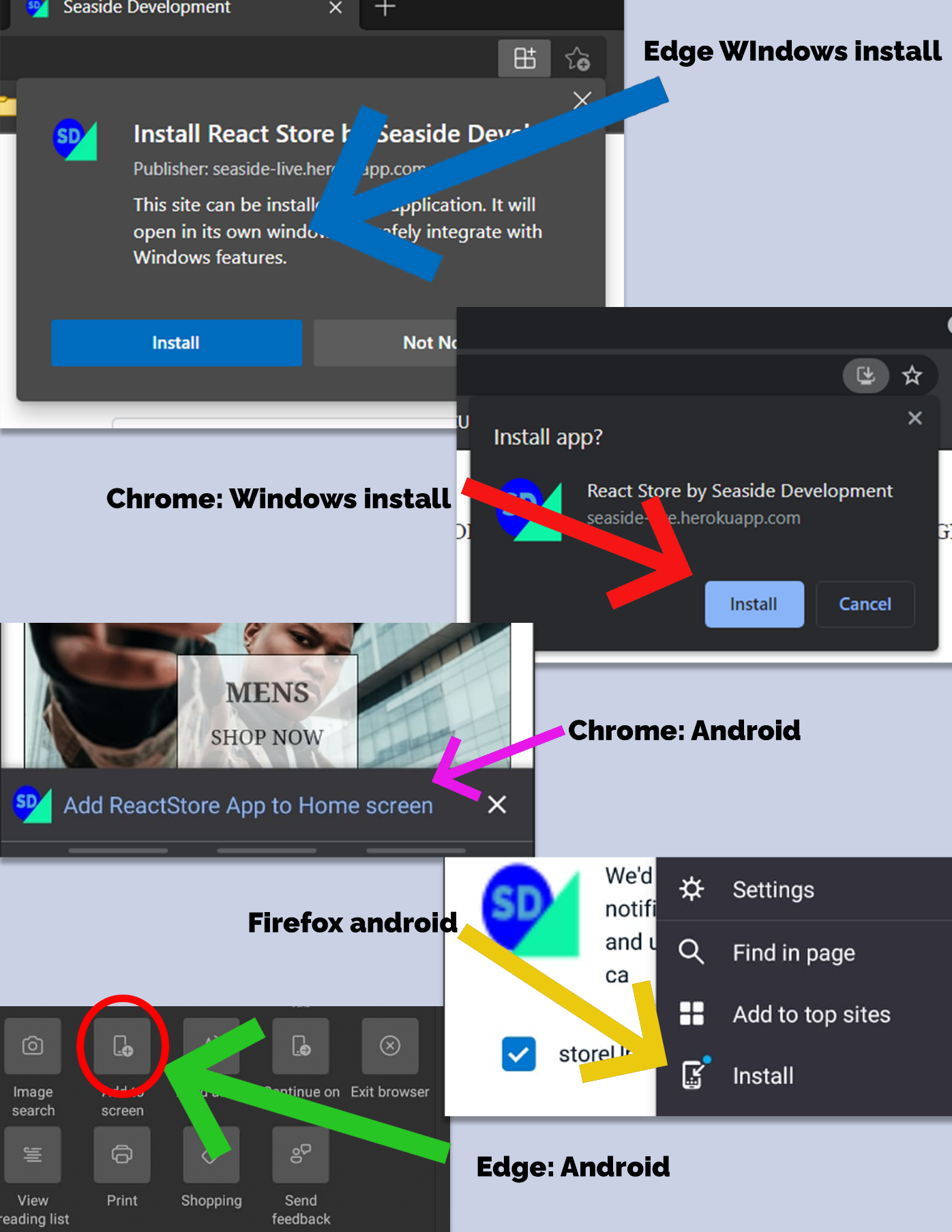
### Contributions

As stated, before the database was the one used in the previous application, meaning the members Samuel Lowe and Zachary Aird. The UI of the system was designed by all members of the group. Android coding with React Native was done by members Nathan Brathwaite and Zachary Aird. IOS coding with React Native was done by Kareen Proverbs and Samuel Lowe. The imported code was designed by Khareen Proverbs and Nathan Brathwaite.

**React Native GitHub link:** [Seaside-Development/Android-ios-store (github.com)](https://github.com/Seaside-Development/Android-ios-store)

**React JS GitHub Link**: [Seaside-Development/WebApp-store: A react eCommerce store (github.com)](https://github.com/Seaside-Development/WebApp-store)

**Do note. You can try a live version of our WebApp here:** <https://seaside-live.herokuapp.com/>. Below is how to install the Web App on your device to use natively.



**There is also an APK for android here**: <https://mycavehilluwi-my.sharepoint.com/:u:/g/personal/nathan_brathwaite_mycavehill_uwi_edu/EdaNd4C0cEpApiper8k-VqwBBPU1x0bKDU3JTlvsikrZOQ?e=e7268X> .

Do note due to Apple’s restriction we cannot provide a sideload for the app. But you can easily use the Web App just as you would with the native app.

# Application Details

## Build Structure

The system is designed to be multiplatform as such there are different folders meant for the different platforms. For this report, we shall be focusing on the android folder as well as the other folders and files that it interacts with. The way by which the program consolidates these platforms is using the file, App.json, which holds the various platforms the code runs on, and which folders are to be used.

## Components Folder

This folder contains the functions for and layouts for card and header components in the software. All the files associated with the folder import from react. It contains a multitude of files such as alert.components.js, which handles alerts whenever a purchase is made. Another example is within the file cart-item.component.js which contains the views of the checkout.

Figure : Purchase Alert Code

Text

Description automatically generated

## Navigation Folder

Navigation of the application is handled within the navigation folder. It was responsible for assigning titles to various screens and assigning the names of headers as well. Furthermore, it is responsible for the styling of headers used within the application.

Figure : Functionality within ShopNavigator.js

Text

Description automatically generated

## APIs

The application uses a variety of APIs for easier development for the team. The APIs used in this application are as follows: Firebase, Geolocation for React-Native, Stripe API and Google Analytics. Implementation of these APIs is the same for both IOS and Android platforms except for the Geolocation API.

1. **Firebase**: This API is used for the secure storage of user, store, and product information. The firebase API key is accessed within the App.json file. It also handles secure logins and sign-ups of the users in the database. This functionality is held within firebase.js.

Figure : Where the app accesses Firebase

Text

Description automatically generated

1. **Google Analytics**: The Google Analytics API is responsible for tracking activity within the android app, for example how long a user may stay on the home screen. This is found within the analytics.utils.js.

Figure : Google Analytics Code

Text

Description automatically generated

1. **Stripe**: The API is used in this application to handle the payments made by the users once they enter their information. The fire checkout.services.js is what handles the data entered.
2. **Geolocation**: React native handles the actual feat of location, however it requires slightly different means of implementation for each platform. For android, the following code is entered into the manfest.xml file:

Figure : Code for gaining android location permission.



For IOS the required code for location permissions is as follows:

Figure : Code for gaining IOS location Permissions.

Text

Description automatically generated

## Store Folder

This folder contains state management for the application itself through the use of redux. Redux is a state container for JavaScript apps. It controls the states of the major aspects of the apps, the user account states cart state and more.

## UI Folder

The UI folder contains the buttons, style, and layout of a particular part of the application. Most notable the signing-up form. Most of the layout information of this form is within the file AuthForm.js which imports elements from reacting as well as other elements from react-native. These elements include both functionalities as well as design and styles.

Figure : View and imports from AuthForm.js

Text

Description automatically generated

Figure : Styles used for AuthForm.js.

Text

Description automatically generated

## Screens Folder

This folder contains all the views that are visible to the end-user. Including views such as the login screen, the check-out screen or more, the home screen and more. Besides the layout and contents of the views themselves, the file also contains the style and design of the view ranging from the text of the colour to its size.

Figure : An example of a View in the screens folder

Text

Description automatically generated

Figure : Style options within CartScreen.js

Text

Description automatically generated

## Data Folder

For the application having functionality outside of an active internet connection, a folder containing data was created. This folder contained 3 files and are as follows: project.js, startingData.js and AllData.json.

1. **Producsts.js:** This contains the products of the store which imports a class called Products which contains the following variables: id, category, name, imageUrl and price. These products are constructed and stored into a constant list named STORE. It should be noted that the variable named categories reference the category class.

Figure : A snippet of the products.js file

Text

Description automatically generated

1. **startingData.js**: This file contains all the categories of various products which are imported from the Category class. A category object has the following variables: Id, imageURL, name, linkURL and a description. The file contains a single constant named CATEGORY which is a list of constructed category objects. It is from this list and a products category is selected.

Figure : Snippet of the startingData.js file

Text

Description automatically generated

1. **AllData.json**: This file is now the combination and utilization of the prior files. It is here that the products created and stored within the application. It is here the app grabs the data and displays the information to the user once offline.

Figure : A snippet of the AllData.json file

Text

Description automatically generated

# Database Management

How data is accessed and managed within the system is multifaceted. The firebase folder gets the information from the database, the Redux folders will then create a state and the firebase API will then use this to either, access, edit, delete or enter information into the database. There are 3 main collections for its database system, and they are as follows:

1. Carts
2. Collections
3. Users

## Carts

Carts contain information on what is within a user’s cart at a given time. Within the cart collection, each document is a unique cart belonging to a registered user and these documents contain the items that the associated user wishes to buy. Each item within the document has the following fields:

1. ID: The identification number of the item
2. imgURL: The associated image URL of the item
3. name: The name of the item.
4. price: The price by which it was sold.
5. quantity: The amount of the associated item the user has ordered.

It should be noted that these field values, except for the quantity ordered are retrieved from the “Collections” collection.

Graphical user interface, text, application, email

Description automatically generated

## Collections

The documents within this table contain all information with regards to the items. Each item within the documents has 4 fields:

1. ID: The identification number of the item
2. imgURL: The associated image URL of the item
3. name: The name of the item.
4. price: The price by which it was sold.

These are the values that are referenced in the cart documents.

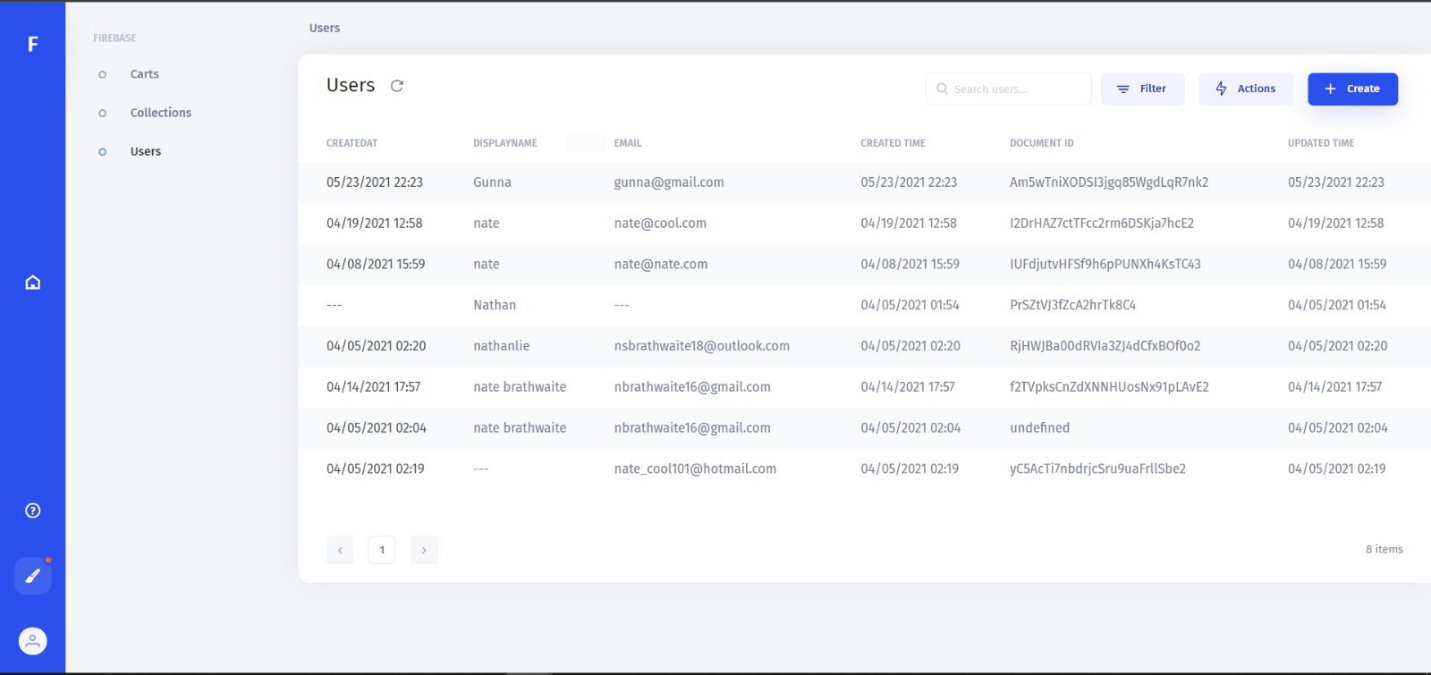
Graphical user interface, text, application, email

Description automatically generated

## Users

This table contains information about the users who are registered into the system. Its fields are as follows:

1. created: The date and time the user was created.
2. DisplayName: The name that the user registered themselves as
3. Email: The email of the user (primary key)
4. Created Time: The date and time the user’s cart was created.
5. DocumentID: The ID of the shopping cart that is associated with the user
6. Updated Time: The last time the user’s cart was changed



# Application in Action

Figure : Checkout Screen

Graphical user interface, text, application, email

Description automatically generated

Figure : Item Selection Screen

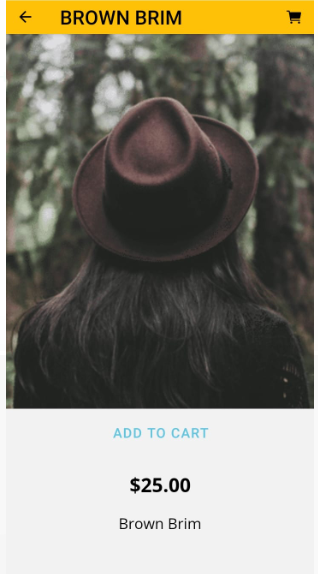


Figure : Home Screen

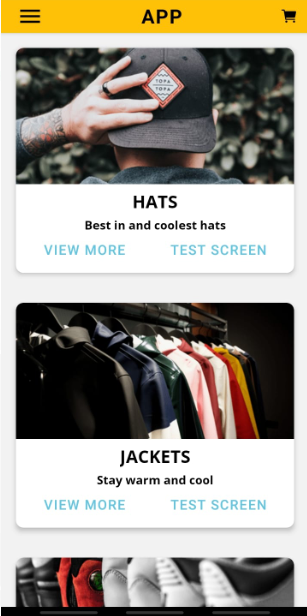


Figure : Payment Screen

Graphical user interface, application

Description automatically generated

Figure : Example of Products Screen

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure : Navigation Side Menu

Graphical user interface, text, application, chat or text message

Description automatically generated