A report on the web app for Web and Mobile course. Do enjoy

WEB REPORT

Seaside Development report

Nathan Brathwaite, Zacahry Aird, Samuel Lowe, Khareen Proverbs

Contents

[Introduction 2](#_Toc74898651)

[Purpose 2](#_Toc74898652)

[Background 2](#_Toc74898653)

[Project Overview 2](#_Toc74898654)

[Development Process 2](#_Toc74898655)

[Contributions 3](#_Toc74898656)

[Application Details 3](#_Toc74898657)

[Build Style 3](#_Toc74898658)

[Public Directory 3](#_Toc74898659)

[SRC Directory 4](#_Toc74898660)

[Assets 4](#_Toc74898661)

[Components 5](#_Toc74898662)

[Firebase Folder 5](#_Toc74898663)

[Pages 6](#_Toc74898664)

[Redux 6](#_Toc74898665)

[Database Management 7](#_Toc74898666)

[Carts 8](#_Toc74898667)

[Collections 10](#_Toc74898668)

[Users 11](#_Toc74898669)

[PWA (Progressive Web App) 12](#_Toc74898670)

# Introduction

## Purpose

The purpose of this web application was to facilitate an online storefront that would allow users to purchase a variety of different clothing items along with other features. 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 had a better understanding of web development and took this as an opportunity to try new aspects of web design starting with the implementation of React and evolving by incorporating APIs to reduce workload. This Report will focus on the backend of the app, as the storefront is essentially the same as android and IOS and you can refer to that for the storefront.

A link is provide so access the PWA storefront here: [Seaside Development (seaside-live.herokuapp.com)](https://seaside-live.herokuapp.com/)

## Project Overview

### Development Process

The coding was done utilizing the React framework, this was done to allow for usages of the react library along with its functionalities. The code was a mixture of JavaScript for functionality and an extension of JavaScript called JSX, which deals with things such as the UI. Team members also needed to learn how to use and implement the firebase API into the software. This was the team’s first major usage of React and time was taken by some of its members to learn aspects and usages of the framework. PNGs were also used for things such as the main logo of the system.

### 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. Main and functional code was designed and implemented by Khareen Proverbs and Nathan Brathwaite

**GitHub link**

# Application Details

## Build Style

Since this is a storefront application there must a client component and a server component. The server component acts as the backend of the system and exists as a single file named server.js. The client component exists as a directory marked client and it contains most of the functionality such as front-end elements as well as assets such as the company logo. The client directory is broken down into two subdirectories “public” and “src” this report shall expand on these directories more.

## Public Directory

The public directory is a very basic directory containing various. PNG files, a. JSON file and an HTML file. The first notable file is “index.html” which is responsible for starting the application. The. The JSON file is called “manifest.json” and it is responsible for getting various. PNG files and displaying them for a web app.

## SRC Directory

The SRC (short for source) directory contains a multitude of folders and files, the folders will be discussed further but the individual files shall be addressed first. The files global.styles.js, index.css and Theme.js are responsible for the theming and style of the application. Servicer-worker.js and ServicerWorkerRegistration.js are the files responsible for making the application a Progressive Web App (PWA). App.js contains all the routing of the app. Within the src directory, there are main features that do the heavy lifting of the application.

1. Assets Folder
2. Components Folder
3. APIs
4. Pages Folder
5. Redux

### Assets

Assets is a very simple folder containing 3 files: crown.svg, shopping-bag.svg, iconmonstr-shopping-bag-7.svg. These files are .svg image files that are used by the application constantly while it is running.

### Components

This folder contains all the functionality of the system in all aspects. The amount of subdirectories is quite large and specific functionality will be discussed in a later part of the code.

### Firebase Folder

This directory is responsible for the utilization of the Firebase API within the PWA. As stated, prior we used the Firebase API for the secure storage of user information. Its files contain functions as to how this information is stored as well as a schema file, showing what information is to be stored. This API is used for the secure storage of user, store, and product information. The firebase API key is accessed within the firebase.util.js file.

Figure 1: Accessing Firebase with Credentials

Text

Description automatically generated

### Pages

The “Pages” sub-directory contains web functionality with regards to the various pages within the web application. Each subdirectory within the “Pages” folder is responsible for a different webpage and each of these contains a .jsx file for style and theming of the webpage and either 1 or 2 additional .jsx files responsible for specific functionality of that page.

### Redux

Redux is a state management container that exists within React, which mean keeping track of the states of the different components that exist within the system. It has its libraries and tools which are used in parts of our system. Through the use of redux allows our web app to maintain persistence and keep appropriate information for the user in unexpected events.

|  |  |
| --- | --- |
| import *{* combineReducers*}* from 'redux'; import *{*persistReducer*}* from "redux-persist"; import storage from 'redux-persist/lib/storage';  import userReducer from './user/user.reducer'; import cartReducer from './cart/cart.reducer'; import directoryReducer from './directory/directory.reducer'; import shopReducer from './shop/shop.reducer';  const persistConfig = *{* key: 'root',  storage,  whitelist: *[*'cart'*] }* const rootReducer = combineReducers*({* user: userReducer,  cart: cartReducer,  directory: directoryReducer,  shop: shopReducer, *})*;  export default persistReducer*(*persistConfig, rootReducer*)*;  This example of root reducer for the redux state on Web App, it helps bring all the redux states together and make them callable throughout the app | import categoryReducer from './store/reducers/category'; import productReducer from './store/reducers/products'; import cartReducer from "./store/reducers/cart"; import userReducer from "./store/reducers/user"; import ***thunk*** from "redux-thunk";  const persistConfig = *{* key: 'root',  storage: AsyncStorage,  whitelist: *[*'cart'*] }* const rootReducer = combineReducers*({* categories: categoryReducer,  products: productReducer,  cart: cartReducer,  user: userReducer, *})*;  const persist = persistReducer*(*persistConfig, rootReducer*)*; const store = ***createStore****(*persist, composeWithDevTools*()*, applyMiddleware*(****thunk****))*; const persistor = persistStore*(*store*)*;  Here is a comparison of redux state reducer on the React native. Almost identical in nature |

## Database Management

How data is accessed and managed within the system is multifaceted, we decided to you a NoSQL database due to simplicity and ease of integration. The Firebase folder gets the information from the database, the Redux folders create a state, and the firebase API will then use this to perform the CRUD operations such as to either, access, edit, delete or enter information into the database.

|  |
| --- |
| //ADD NEW ITEMS TO THE DATABASE DOCUMENT INSIDE COLLECTIONS DOCUMENT export const addCollectionAndDocuments = async *(* collectionKey,  objectsToAdd *)* => *{* const collectionRef = ***firestore***.collection*(*collectionKey*)*;   const batch = ***firestore***.batch*()*;  objectsToAdd.forEach*((*obj*)* => *{* const newDocRef = collectionRef.doc*()*;  batch.set*(*newDocRef, obj*)*;  *})*;   return await batch.commit*()*; *}*;  //CALLING TO THE OBJECT AND MAPPING TO THE APPROPRIATE CONTAINERS export const convertCollectionsSnapshotToMap = *(*collections*)* => *{* const transformedCollection = collections.docs.map*((*doc*)* => *{* const *{* title, items *}* = doc.data*()*;   return *{* routeName: encodeURI*(*title.toLowerCase*())*,  id: doc.id,  title,  items,  *}*;  *})*;   return transformedCollection.reduce*((*accumulator, collection*)* => *{* accumulator*[*collection.title.toLowerCase*()]* = collection;  return accumulator;  *}*, *{})*; *}*;  Although the store front does use the create operation. We still added a function addCollectionAndDocuments as means for better Content Management if need be. It also allows developers to easily add a new collection to the Firebase Firestore. |

The is an admin portal on the storefront that can only be accessed by the master user or any other authorized user. There are 3 main collections for its database system, and they are as follows:

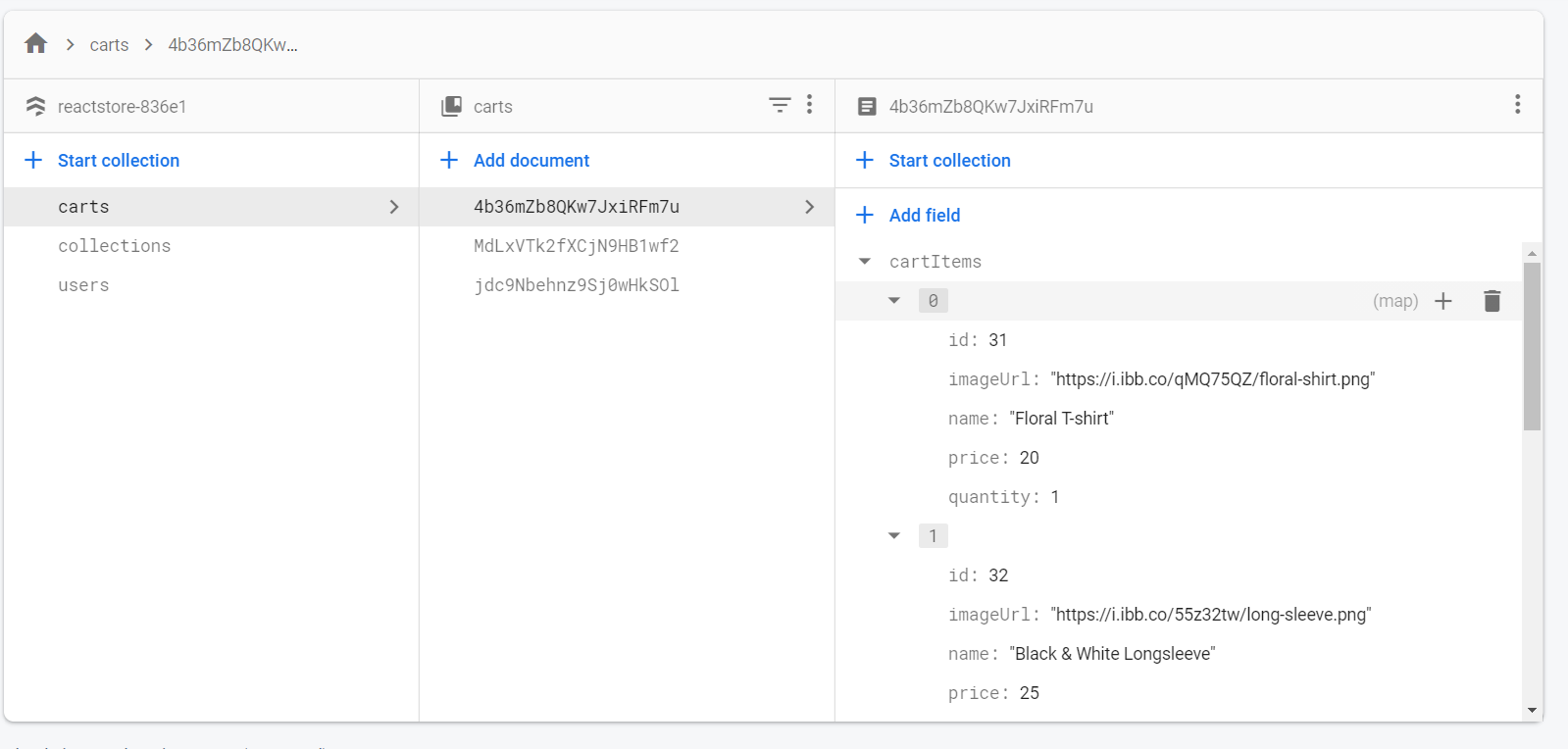
1. Carts
2. Collections
3. Users

### 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.

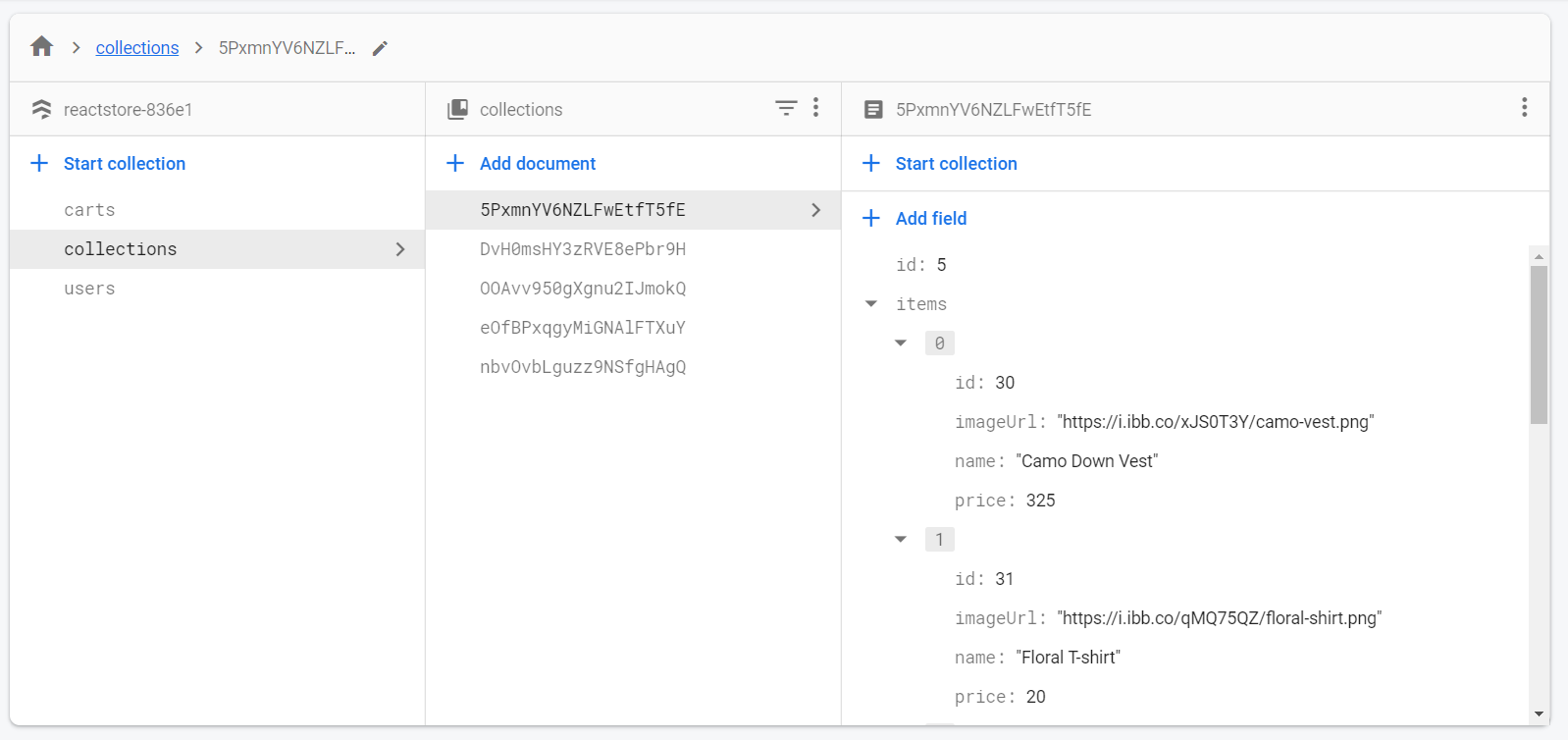


### Collections

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.

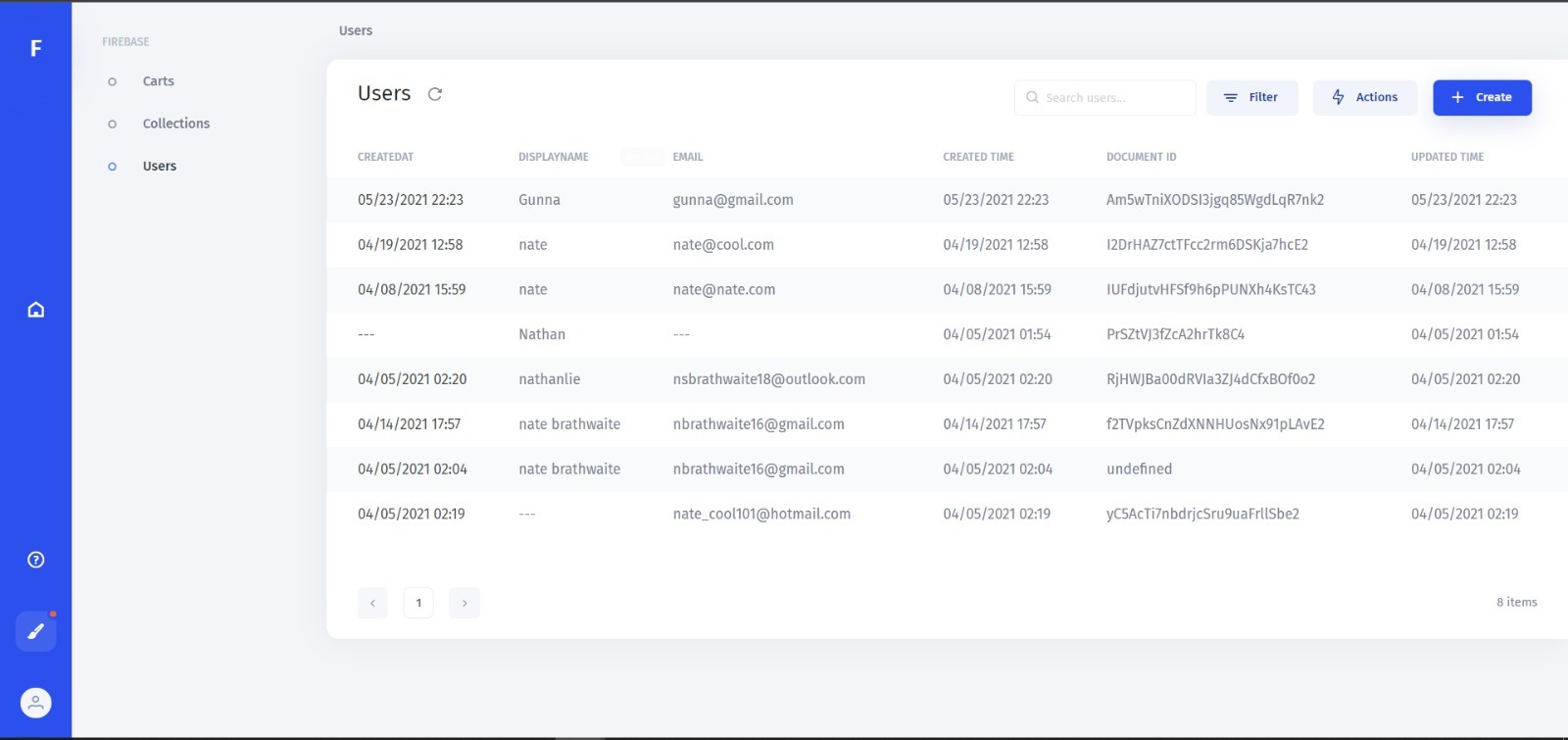


### Users

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

Actual look at the admin panel below



# PWA (Progressive Web App)

To make sure the application could work on both web and mobile platforms as a standalone app we add PWA abilities to the app. This was achieved by giving the app a service worker file. The Service worker is what is allows for the ability to download and sideload the app. Additionally there is a manifest file in the public directory, what it does is inform the operating system of the app name, icon, and splash for the app when it is downloaded.

# Express Server

To facilitate a better app experience when hosting on sites like Heroku. We add an express server in the app. This is in the server.js file. What is the server intended to do?

1. Compress the app in production.
2. Enforce the https protocol – Express-sslify
3. Handle Stripe API functions
4. Separate the secret keys from the frontend.
5. Register the app with a service worker so it can run as a PWA and loaded as a standalone app.

The backend server is great for handling token request for any API we made add further on.