CS385 Web Application Development

Project Report

Irish Gifts Marketplace

Team Vandium

Conor Walsh 20252982

Kim Winters 20252992

Jaymie O'Brien 20252991

Gareth McNaboe 20252984

Contents

Introduction / Instructions to install

With the exception of the running the command 'npm install' on the command line (to install the various dependencies), the app should not require any special steps in order to install.

The data (other than the product photos) is stored in a firebase real time database. The database, if required, can be accessed directly using the account information outlined below.

username: vandiumshannon@gmail.com **password:** cs385cjkg

The app is also hosted at: https://irish-gift-marketplace.netlify.app/

Chapter 1: Why did we develop this app?

We identified a need to develop a marketplace specifically for Irish products due to the following reasons.

1. Covid-19 restrictions has changed how Irish consumers purchase - accelerating the growth of online spending,
2. Customers demand fast, safe, easy to use websites to purchase from,
3. Only 23% of small Irish businesses use eCommerce in a meaningful way*,
4. 70% of online purchases made in Ireland is done in overseas markets*.

It is clear from the trending hashtags #supportirish, #supportlocal and #buyirish that Irish consumers want to support local businesses and keep their money circulating in Ireland this Christmas.

There is currently no marketplace for Irish made goods, so we decided to fulfil this need.

We discussed some other ideas also but believed that we could add infinite functionality to our marketplace (given enough time) so this would be the best test of our skills.

*https://www.localenterprise.ie/Tipperary/FinancialSupports/Trading%20Online%20Voucher%20Scheme%20for%20Small%20Irish%20Businesses/ - Retrieved 12:00 16th December 2020

Chapter 2: How did we design the app?

Following our first team meeting we agreed on the following key items:

- Application Programming Interface (API) Structure
- Pages/Routes needed
- To use GitHub for source control
- To use Microsoft Teams as the main communication channel.
- To use Trello to record work items to be completed.

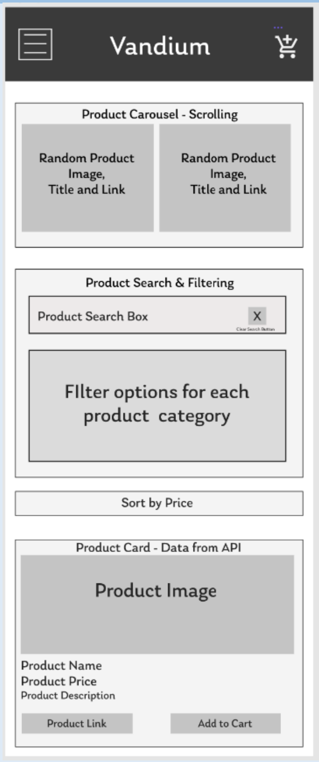
We decided to create our own product API for the application. 8 product categories (Beauty, Jewellery, Fashion, Sports, Home, Garden, Food & Drink and Art) were agreed on. We created 25 products each, based on a pre-agreed template, to have a total of 100 products in the API. The structure of the API product data was as follows:

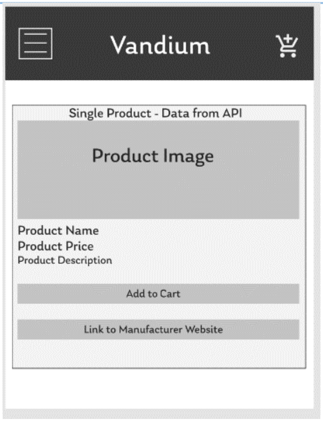
```
{
  "categoryID": 0,
  "deliveryCost": 4.5,
  "description": "Luxury Handcrafted Wooden Gift Box",
  "id": 1,
  "image": "https://raw.githubusercontent.com/Team-Vandium/data/main/1.JPG",
  "manufacturer": "baressentials",
  "manufacturer_website": "https://www.baressential.ie/product/luxury-gift-set-mini-three/",
  "name": "Body Oil & Soap Luxury Gift Box",
  "price": 84,
  "size": "small",
  "tags": ["parent", "female", "health and beauty"],
  "weight": 525
}
```

Our API was also to include an array of Categories, Delivery Costs and the Free Delivery threshold.

The 6 pages needed were the Homepage, Single Product Page, Basket, About, Newsletter Signup & a 404 Error Page Not Found. Wireframes for each page were created using the Figma design tool.

A navigation bar should appear at the top of each page, with links to the various pages and the shopping basket. The shopping basket should have the number of items currently in the basket listed beside it if possible.

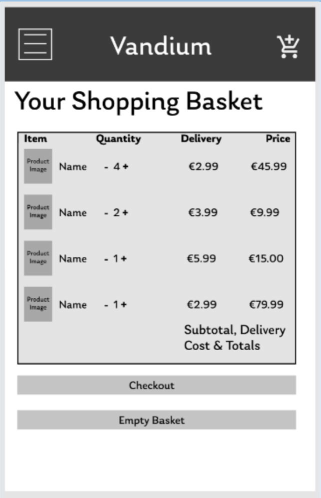
Wireframe	Description
 Vandium Product Carousel - Scrolling Random Product Image, Title and Link Random Product Image, Title and Link Product Search & Filtering Product Search Box X FIlter options for each product category Sort by Price Product Card - Data from API Product Image Product Name Product Price Product Description Product Link Add to Cart	**Homepage** The landing page for the website contains a scrolling product carousel. The carousel would be followed by a section where the user can narrow down the list of products on display via a search box or by using Product Category buttons. A button to allow the user sort the products listed according to their price should also appear here. Finally, the list of products should be displayed underneath with the product image, name, price and a shortened description on display to the user. Two buttons should also appear here. One to allow the user to click view the full product and the other to add the product to the basket.



Single Product Page

The single product should contain the data for each product - product image, name, price, full description.

This should be followed by two buttons – One to allow the user to add that product to their basket, the second button contains a link to the manufacturer's website.

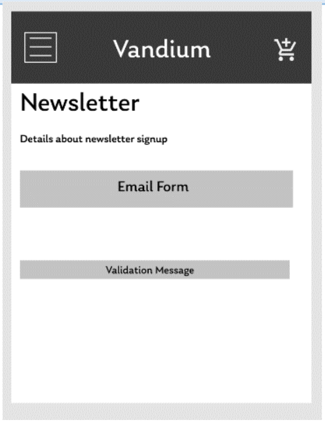


Item		Quantity	Delivery	Price
Product Image	Name	- 4 +	€2.99	€45.99
Product Image	Name	- 2 +	€3.99	€9.99
Product Image	Name	- 1 +	€5.99	€15.00
Product Image	Name	- 1 +	€2.99	€79.99

Basket

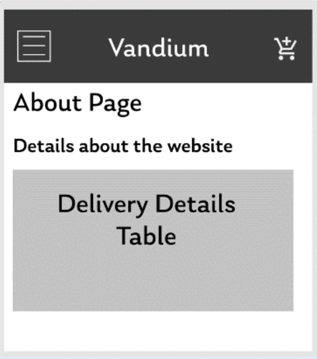
This wireframe shows how the Basket page should look. It contains a list of the items that the current user has in their shopping basket. A HTML5 table styled with bootstrap should be used for the list of products currently in the basket. A subtotal, delivery price and total price should display at the end of the table. Buttons to allow the user to checkout or empty their basket should appear at the end of the basket page.

If the user has nothing in their basket, a message should show encouraging the user to add some products.



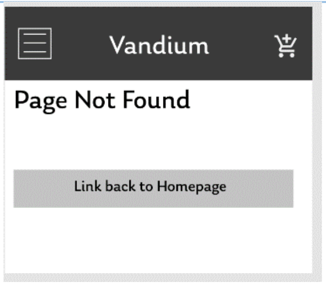
Newsletter Signup

The newsletter signup page should display an input that will allow the user to enter their email to be added to a marketing list. There should be a validation message displayed to the user if they haven't inputted a valid email address. The e-mail address is saved to the Realtime Database.



About

The about page should display some information about the website along with a table detailing on how the delivery charges are calculated.

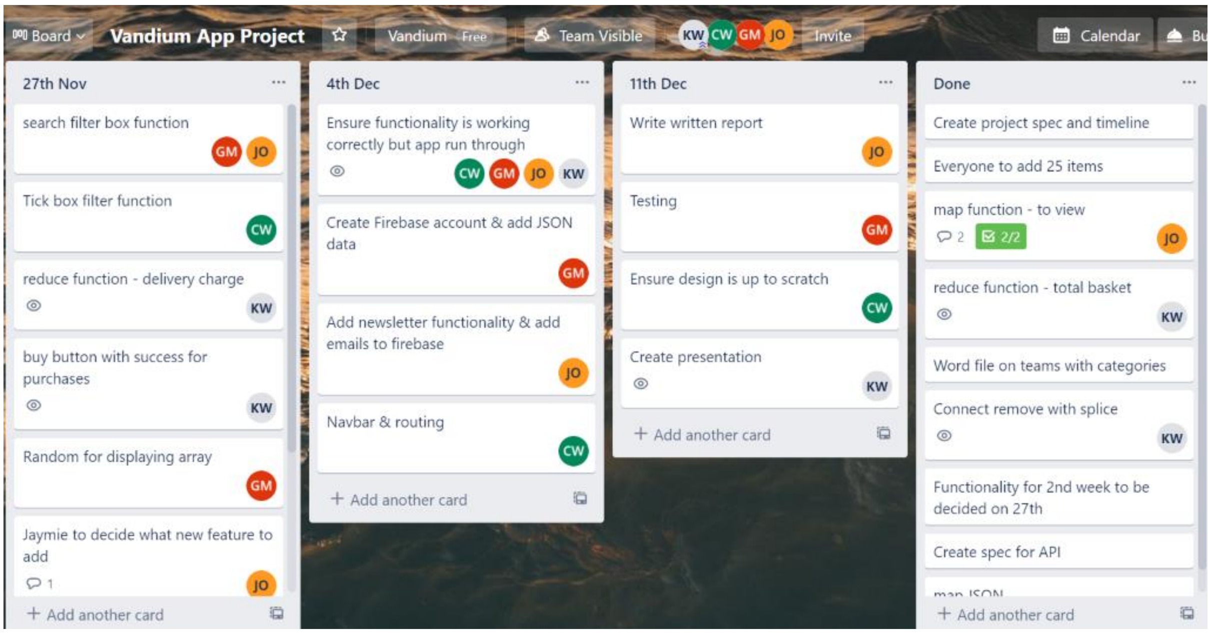
	**Page Not Found** This page should appear when the user tries to load a page that a route does not exist for. It should contain a link back to the homepage of the application.

Note:

The image file for each of the products is stored in the app in the /images folder. Ideally we would have arranged for separate hosting of these images so that the full details of products could be updated without amending the app itself. It was decided given the time and cost involved in undertaking this work it was not a priority. This decision was cleared with Peter Mooney as a part of one of our weekly lab updates. Moving the photos to a hosting services is one of the follow up work items identified in chapter 4 of this report.

Chapter 3: How did we implement the functionality in the app?

We organised our team using Trello throughout the project.



27th Nov

- search filter box function — GM JO
- Tick box filter function — CW
- reduce function - delivery charge — KW
- buy button with success for purchases — KW
- Random for displaying array — GM
- Jaymie to decide what new feature to add — 1 — JO
- + Add another card

4th Dec

- Ensure functionality is working correctly but app run through — CW GM JO KW
- Create Firebase account & add JSON data — GM
- Add newsletter functionality & add emails to firebase — JO
- Navbar & routing — CW
- + Add another card

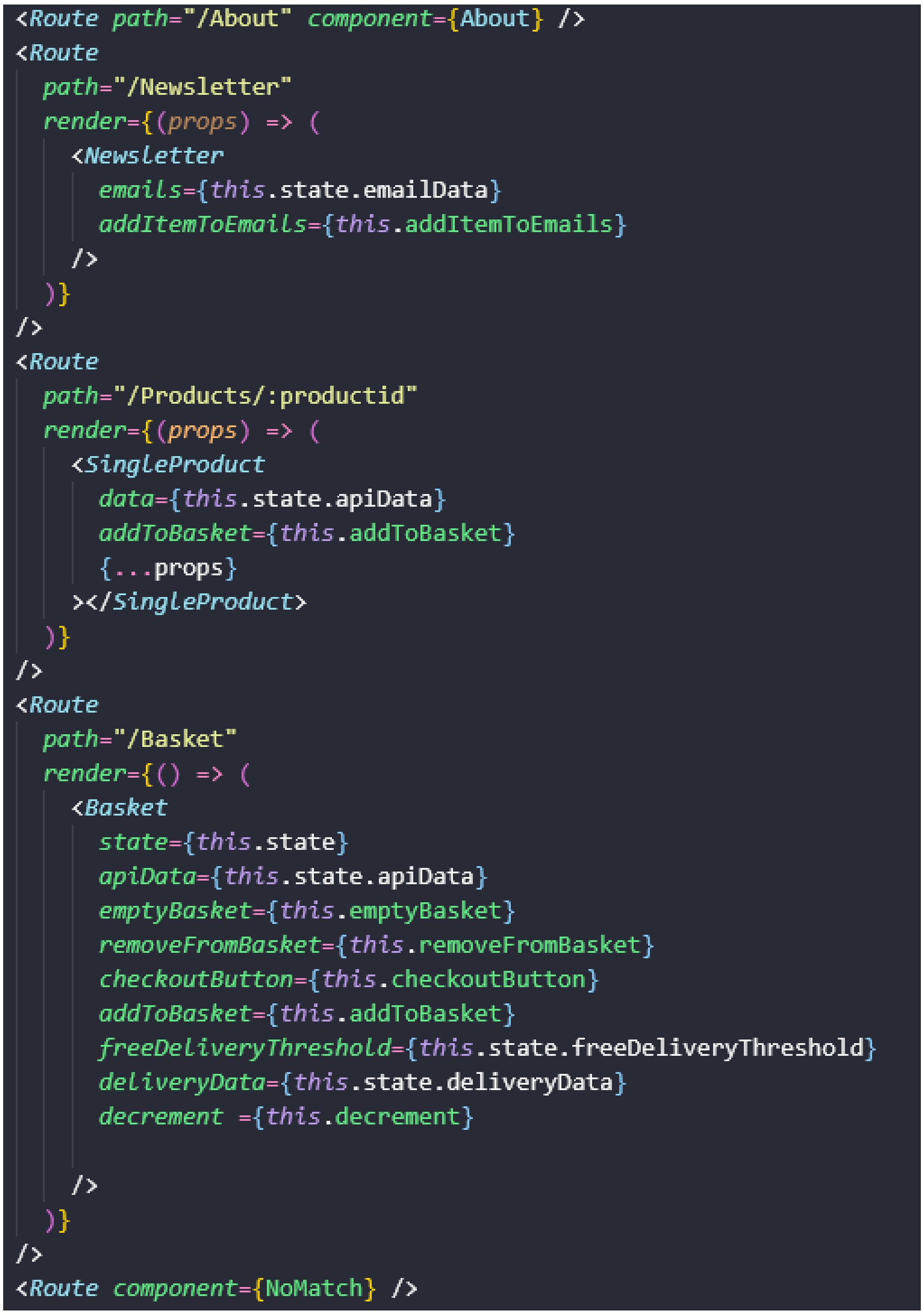
11th Dec

- Write written report — JO
- Testing — GM
- Ensure design is up to scratch — CW
- Create presentation — KW
- + Add another card

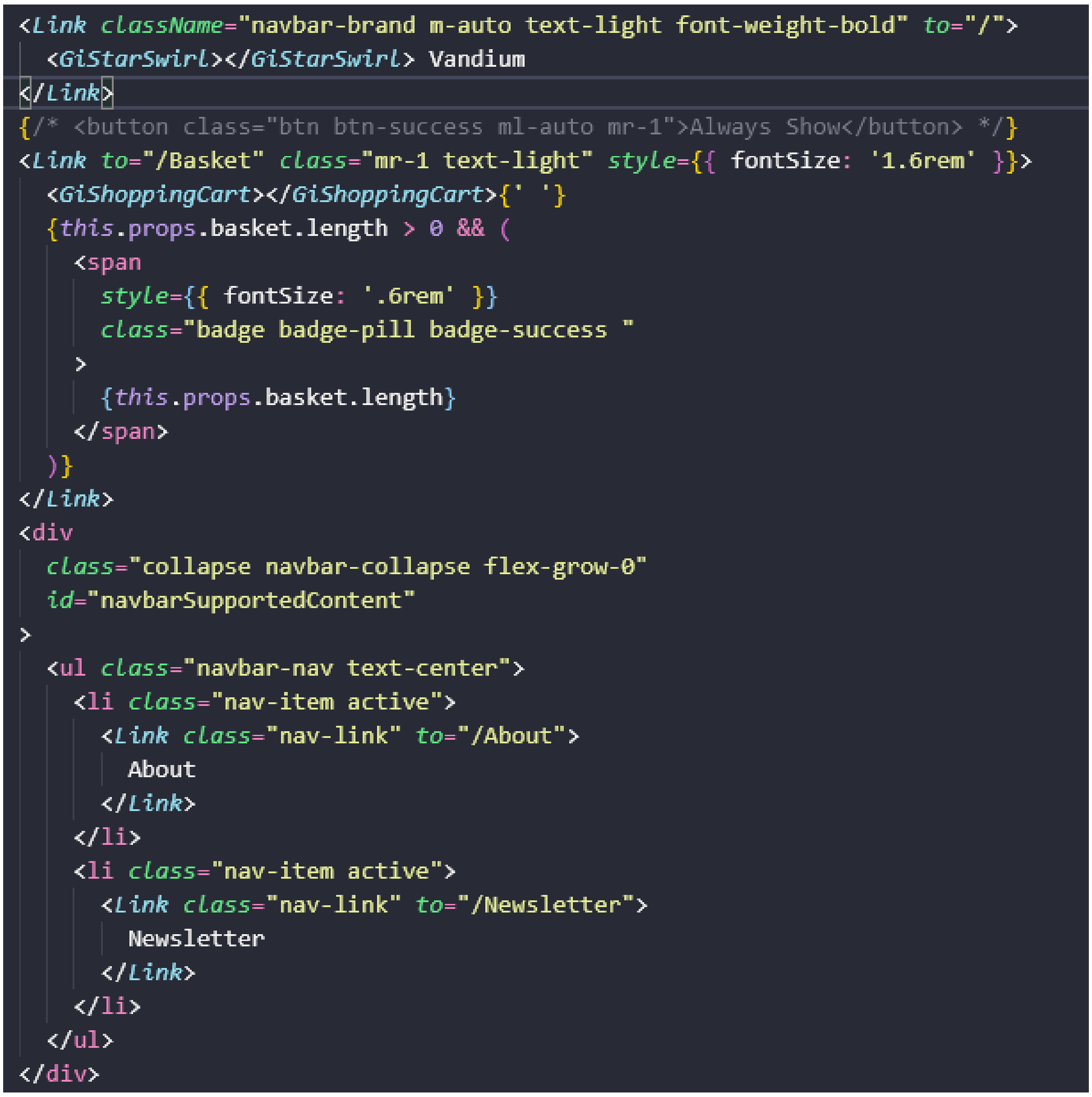
Done

- Create project spec and timeline
- Everyone to add 25 items
- map function - to view — 2 — 2/2 — JO
- reduce function - total basket — KW
- Word file on teams with categories
- Connect remove with splice — KW
- Functionality for 2nd week to be decided on 27th
- Create spec for API
- map JSON
- + Add another card

We used React Router to allow multiple pages to be rendered separately, while also maintaining a navigation bar at the top of the page and consistency of data throughout the components. It proved to be a very neat way of implementing a multi-page look. A series of <Link> tags in the navigation bar combined with a <Switch> tag in the App.js file allowed for smooth transitions between pages of the app.

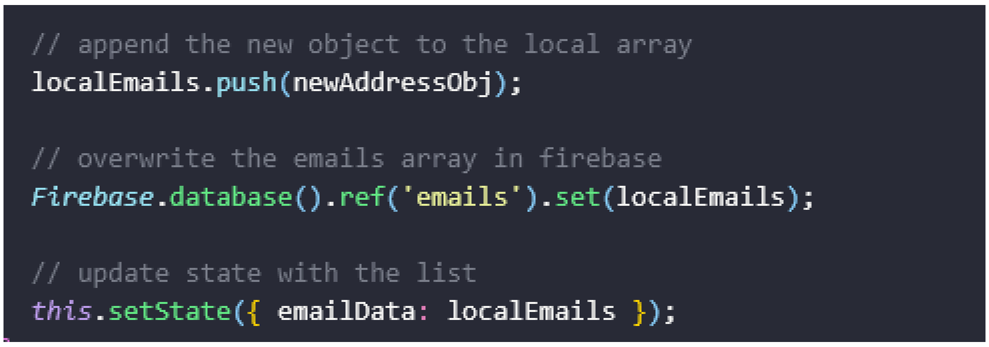


```
<Route path="/About" component={About} />
<Route
  path="/Newsletter"
  render={(props) => (
    <Newsletter
      emails={this.state.emailData}
      addItemToEmails={this.addItemToEmails}
    />
  )}
/>
<Route
  path="/Products/:productid"
  render={(props) => (
    <SingleProduct
      data={this.state.apiData}
      addToBasket={this.addToBasket}
      {...props}
    ></SingleProduct>
  )}
/>
<Route
  path="/Basket"
  render={() => (
    <Basket
      state={this.state}
      apiData={this.state.apiData}
      emptyBasket={this.emptyBasket}
      removeFromBasket={this.removeFromBasket}
      checkoutButton={this.checkoutButton}
      addToBasket={this.addToBasket}
      freeDeliveryThreshold={this.state.freeDeliveryThreshold}
      deliveryData={this.state.deliveryData}
      decrement ={this.decrement}

    />
  )}
/>
<Route component={NoMatch} />
```



```
<Link className="navbar-brand m-auto text-light font-weight-bold" to="/">
  <GiStarSwirl></GiStarSwirl> Vandium
</Link>
{/* <button class="btn btn-success ml-auto mr-1">Always Show</button> */}
<Link to="/Basket" class="mr-1 text-light" style={{ fontSize: '1.6rem' }}>
  <GiShoppingCart></GiShoppingCart>{' '}
  {this.props.basket.length > 0 && (
    <span
      style={{ fontSize: '.6rem' }}
      class="badge badge-pill badge-success "
    >
      {this.props.basket.length}
    </span>
  )}
</Link>
<div
  class="collapse navbar-collapse flex-grow-0"
  id="navbarSupportedContent"
>
  <ul class="navbar-nav text-center">
    <li class="nav-item active">
      <Link class="nav-link" to="/About">
        About
      </Link>
    </li>
    <li class="nav-item active">
      <Link class="nav-link" to="/Newsletter">
        Newsletter
      </Link>
    </li>
  </ul>
</div>
```

We used Firebase to store and maintain our data, including the product list, delivery charges and email addresses. Originally, we had used JSON data stored on GitHub, but Firebase allowed us to have more flexibility in updating the database to store data entered into the app.



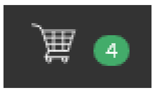
```
// append the new object to the local array
localEmails.push(newAddressObj);

// overwrite the emails array in firebase
Firebase.database().ref('emails').set(localEmails);

// update state with the list
this.setState({ emailData: localEmails });
```

Data was retrieved from Firebase in App.js and maintained in state, this allowed for relevant data to be passed to child components as needed.

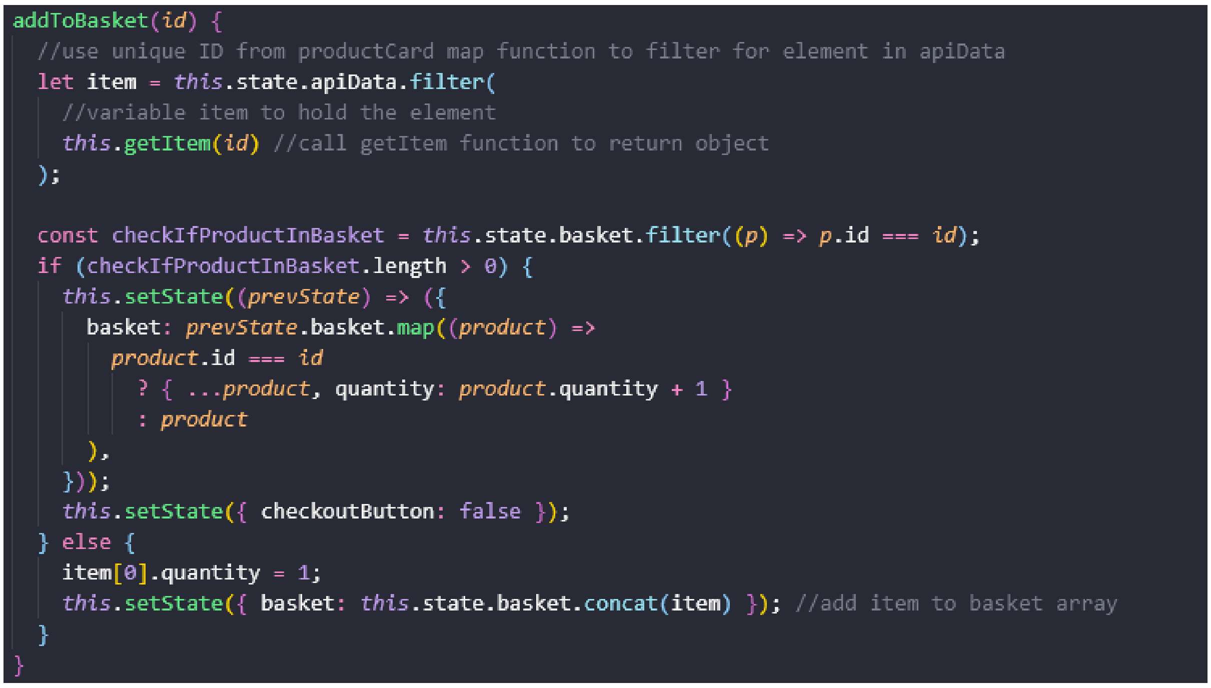
```
async componentDidMount() {
  try {
    this.getMessagesFromDatabase();
  } catch (error) {
    console.log(error);
    this.setState({ errorMsg: error });
  } // end of try catch
} // end of componentDidMount()

getMessagesFromDatabase() {
  //download and create json array of product data
  let ref1 = Firebase.database().ref('products');

  ref1.on('value', (snapshot) => {
    // json array
    let msgData = snapshot.val();
    let newMessagesFromDB1 = [];
    for (let m in msgData) {
      // create a JSON object version of our object.
      let currObject = {
        categoryID: msgData[m].categoryID,
        description: msgData[m].description,
        id: msgData[m].id,
        image: msgData[m].image,
        manufacturer: msgData[m].manufacturer,
        manufacturer_website: msgData[m].manufacturer_website,
        name: msgData[m].name,
        price: msgData[m].price,
        deliveryCost: msgData[m].deliveryCost,
        weight: msgData[m].weight,
        tags: msgData[m].tags,
      };
      // add it to our newStateMessages array.
      newMessagesFromDB1.push(currObject);
    } // end for loop
    // set state
    this.setState({ apiData: newMessagesFromDB1.sort(this.shuffle) });
  });
```

Maintaining state in App.js allowed for data to be shared between components when required. For example, the Basket component has a lot of functionality around adding or removing products, completely clearing the Basket, and calculating prices. On top of this, due to how the app is structured, changes to the contents of Basket are immediately reflected in the Navbar where there is a button showing the number of items currently in the user's basket.

```
<Navbar basket={this.state.basket} />
```



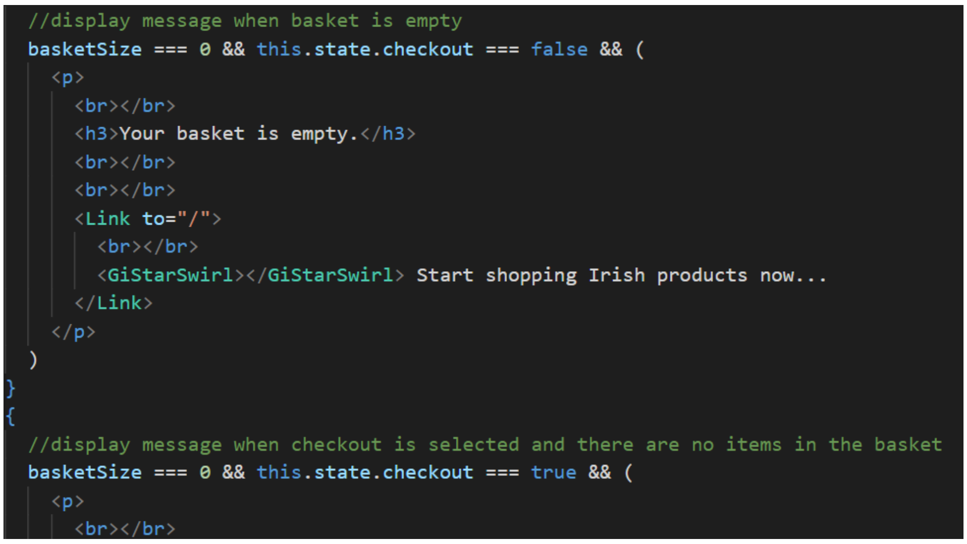
An interesting feature of the Basket component is that if the user adds the same item multiple times the quantity in the basket increases without simply adding another element to the basket array. This is achieved by checking for the existence of the item in the basket array. If the item is found, the 'quantity' value for the item is increased. To increment the quantity the existing state is retrieved, the existing product found, and its contents are spread into the new array along with the incremented quantity value. State is then updated with the new value. The basket can be decremented in the same way.



```
addToBasket(id) {
  //use unique ID from productCard map function to filter for element in apiData
  let item = this.state.apiData.filter(
    //variable item to hold the element
    this.getItem(id) //call getItem function to return object
  );

  const checkIfProductInBasket = this.state.basket.filter((p) => p.id === id);
  if (checkIfProductInBasket.length > 0) {
    this.setState((prevState) => ({
      basket: prevState.basket.map((product) =>
        product.id === id
          ? { ...product, quantity: product.quantity + 1 }
          : product
      ),
    }));
    this.setState({ checkoutButton: false });
  } else {
    item[0].quantity = 1;
    this.setState({ basket: this.state.basket.concat(item) }); //add item to basket array
  }
}
```

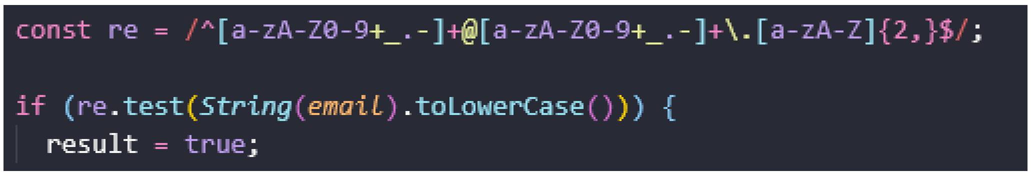
Conditional rendering is used often in the basket component as we want the user to see a relevant information when in basket regardless of their stage of purchase.

When you perform a checkout in your basket, the screen updates to display a thank you message. If you navigate to a new page and then return without making any more changes to basket, you will see the empty basket message.

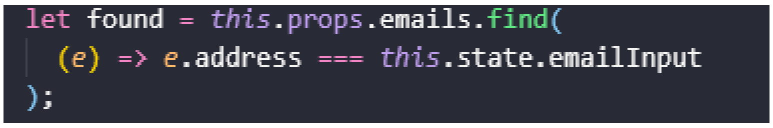


```
  //display message when basket is empty
  basketSize === 0 && this.state.checkout === false && (
    <p>
      <br></br>
      <h3>Your basket is empty.</h3>
      <br></br>
      <br></br>
      <Link to="/">
        <br></br>
        <GiStarSwirl></GiStarSwirl> Start shopping Irish products now...
      </Link>
    </p>
  )
}
{
  //display message when checkout is selected and there are no items in the basket
  basketSize === 0 && this.state.checkout === true && (
    <p>
      <br></br>
```

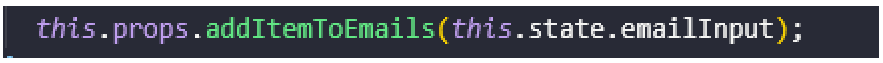
The Newsletter component allows the user to enter an email address to be signed up for a weekly newsletter from the app. The email input is validated using regex, disabling the submit button until validation is passed. The entered address is then checked against the existing mailing list in Firebase to ensure there is no duplication of email addresses. Valid email addresses are then added to the mailing list in Firebase. Conditional rendering is used here again, displaying a different message depending on if the submitted email was already on the mailing list or not.



```
const re = /^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9+_.-]+\.[a-zA-Z]{2,}$/;

if (re.test(String(email).toLowerCase())) {
  result = true;
```



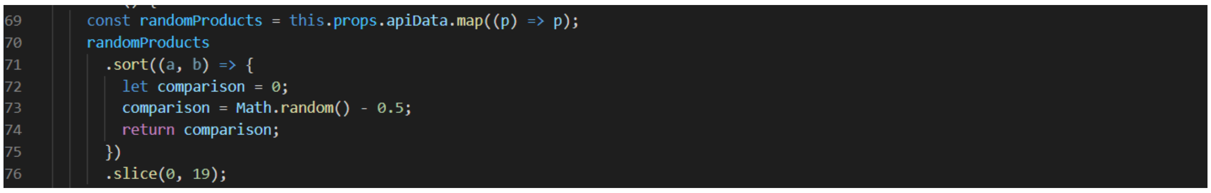
```
let found = this.props.emails.find(
  (e) => e.address === this.state.emailInput
);
```



```
this.props.addItemToEmails(this.state.emailInput);
```

The main Products page has several features to make it easier to find the items you need, including a search box, category buttons, carousel of products, and a sort by button to allow users to filter and rearrange the list of products.

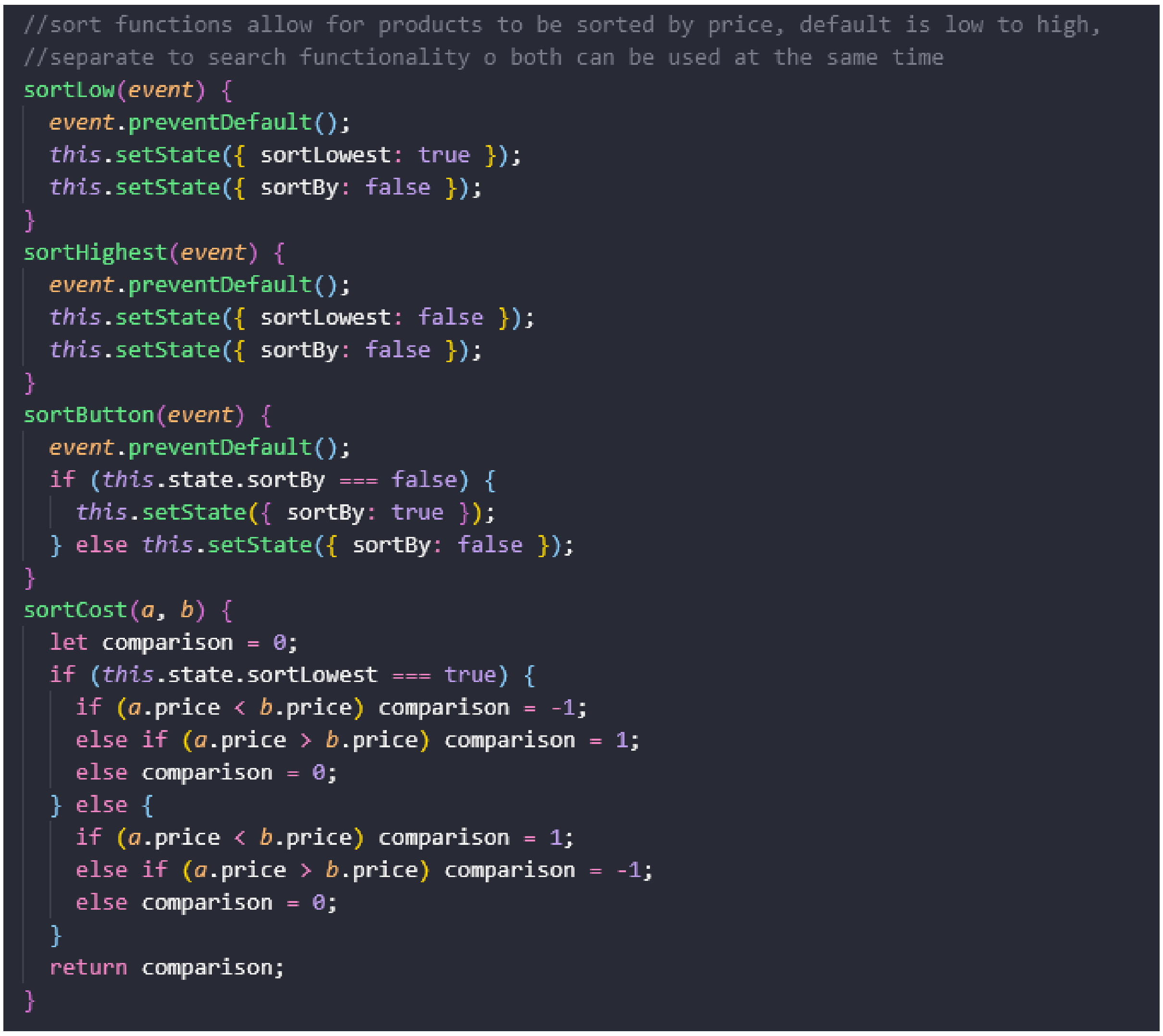
The carousel displays a random selection of the products used a randomised sort function.



```
const randomProducts = this.props.apiData.map((p) => p);
randomProducts
  .sort((a, b) => {
    let comparison = 0;
    comparison = Math.random() - 0.5;
    return comparison;
  })
  .slice(0, 19);
```

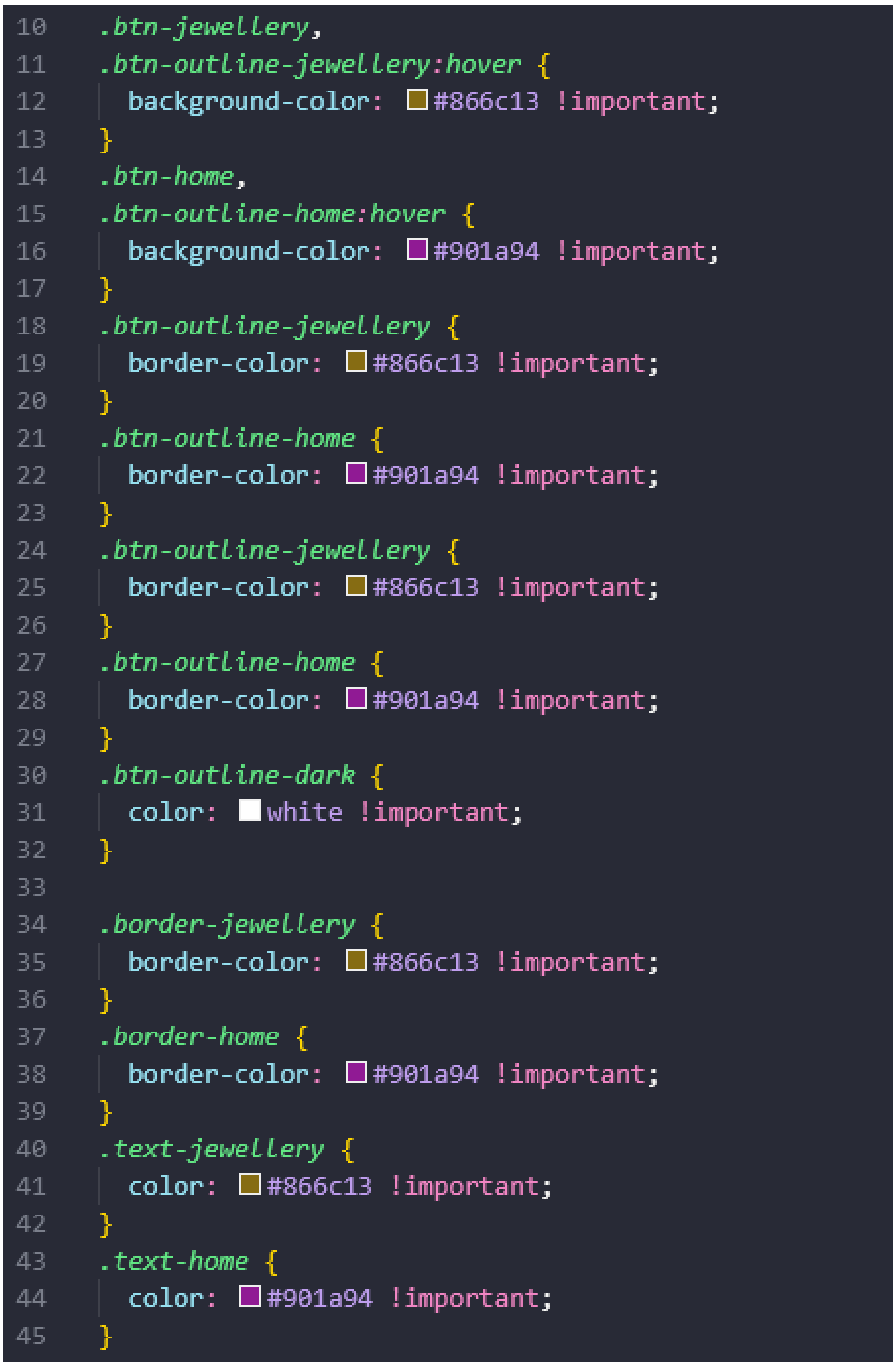
The search box and category buttons take advantage of the filter function when displaying the array of products, seamlessly updating the products shown to match the user's input.

The sort by button takes advantage of the sort function on the array to quickly allow the user to rearrange the products by price.



```
//sort functions allow for products to be sorted by price, default is low to high,
//separate to search functionality o both can be used at the same time
sortLow(event) {
  event.preventDefault();
  this.setState({ sortLowest: true });
  this.setState({ sortBy: false });
}
sortHighest(event) {
  event.preventDefault();
  this.setState({ sortLowest: false });
  this.setState({ sortBy: false });
}
sortButton(event) {
  event.preventDefault();
  if (this.state.sortBy === false) {
    this.setState({ sortBy: true });
  } else this.setState({ sortBy: false });
}
sortCost(a, b) {
  let comparison = 0;
  if (this.state.sortLowest === true) {
    if (a.price < b.price) comparison = -1;
    else if (a.price > b.price) comparison = 1;
    else comparison = 0;
  } else {
    if (a.price < b.price) comparison = 1;
    else if (a.price > b.price) comparison = -1;
    else comparison = 0;
  }
  return comparison;
}
```

Bootstrap allowed us to quickly customise the front end of the application and deliver a responsive, professional looking application. The Yeti theme from Bootswatch is the main theme used in the app. By implementing further custom CSS the app gained a unique look.



```css
.btn-jewellery,
.btn-outline-jewellery:hover {
  background-color: #866c13 !important;
}
.btn-home,
.btn-outline-home:hover {
  background-color: #901a94 !important;
}
.btn-outline-jewellery {
  border-color: #866c13 !important;
}
.btn-outline-home {
  border-color: #901a94 !important;
}
.btn-outline-jewellery {
  border-color: #866c13 !important;
}
.btn-outline-home {
  border-color: #901a94 !important;
}
.btn-outline-dark {
  color: white !important;
}

.border-jewellery {
  border-color: #866c13 !important;
}
.border-home {
  border-color: #901a94 !important;
}
.text-jewellery {
  color: #866c13 !important;
}
.text-home {
  color: #901a94 !important;
}
```

Chapter 4: Overall Evaluation

a. An honest appraisal on the app

We feel that the app provides a pleasant user experience. The products are clearly displayed. The user can easily search the products by category or a word search. More information and a link to the manufacturer's website are also easily accessible.

The shopping basket also calculates and clearly displays the overall cost including the delivery charges. Some sort functionality is quite basic in comparison to similar marketplaces but had we the tim e/skills already developed to connect to a database, sorting using SQL would provide a better user experience.

b. System Tests Undertaken

As a part of the completion of the project we undertook the following system tests in order to verify the functionality of the App.

Test No.	Description of Test	Complete
1.	Checked routing in navbar between main page, basket, about and newsletter.	✓
2.	Selected the more information button for multiple products to ensure individual product pages displayed and that website links on page operate correctly.	✓
3.	Selected all 8 product category radio buttons to ensure product displayed was filtered to show only products in that category. Also checked that in each case the show all button displayed and operated to revert to a view of all the products.	✓
4.	Checked that the scroll right and scroll left buttons in the carousel operated.	✓
5.	Input some search terms into the search box. Checked that search box operates while one of the product category radio buttons is selected. Checked that the clear search button operated correctly.	✓
6.	Selected price low to high and high to low buttons to ensure that the products displayed were relisted.	✓
7.	Selected add to basket to ensure product were added to basket / checked that remove item and empty basket buttons operate correctly.	✓
8.	Selected plus and minus buttons to ensure quantity of each product could be altered and that if reduced to zero that product would be removed from basket.	✓
9.	Checked that sub-total, delivery charge and total figure displayed in the basket are accurate.	✓
10.	Checked that the overall free delivery threshold was being applied correctly.	✓
11.	Checked that the overall number of items in the basket and displayed on the main page basket icon is accurate.	✓
12.	Selected checkout button when basket had items in it and checked that appropriate message displayed.	✓

Test No.	Description of Test	Complete
13.	Input of valid and invalid email addresses in the newsletter page and confirmed that submit button could only be pressed with a valid email address. Checked firebase real-time database to ensure it was updated for new email address.	✓
14.	Checked that appropriate loading message display.	✓

We were able to confirm that the app consistently passed these tests.

c. Are you happy with it?

Given the time and other constraints placed on us in undertaking this project we are pleased with the App which we were able to produce. We also believe that the code which we have written is easy to read, maintainable and would allow for the improvement of the App through the addition of additional features.

We completed the scope of work which we set out at the start of the project in good time. This allowed us to undertake thorough testing of the App and meet the other requirements and deadlines set out for the project.

Although not a specific requirement of the project scope we feel that the App is very well presented as well as easy to use.

d. Areas for Improvement

Given the purpose of the App is to provide a marketplace for the purchase of gifts, the App could be improved by providing additional filtering functions. A gift purchasing guide could take in details of the person you wish to purchase a gift for (e.g. their age, gender, or interests) and provide suggested products.

e. Additional features

If we have another 4 to 6 weeks to work on the project, we would add the following additional functionality to the app.

- Utilise firebase authentication which would allow customers to have user accounts and private data. This would facilitate the saving of baskets of products prior to the completion of orders and a wishlist.
- Arrange for hosting of images so they do not need to be stored in the app.
- Update the Checkout button to record orders to the firebase database.
- Additional filtering functionality could be achieved by connecting a database and uses SQL to allow guests choose their filters.
- Add an additional page to allow customers to remove their e-mail address from the mailing list with a corresponding update to the firebase database.
- Implementation of a payment method to the app.
- Allow for users to review products or import reviews from another website.

Appendix 1: List of Dependencies and their purposes

Dependency	Version	Purpose
@brainhubeu/react-carousel	2.0.1	Library to help implement Product Carousel
@testing-library/jest-dom	5.11.6	Testing
@testing-library/react	11.2.1	Testing
@testing-library/user-event	12.2.2	Testing
bootswatch	4.5.3	Bootstrap Theming Library
firebase	8.1.2	Realtime Database
react	17.0.1	Standard React Dependency
react-dom	17.0.1	Standard React Dependency
react-icons	4.1.0	Icon library
react-lazyload	3.1.0	Helps lazy load images to help with page load speed
react-router-dom	5.2.0	Routing
react-scripts	4.0.0	Scripts from the create-react-app
web-vitals	0.2.4	Measure performance