Assignment 2: Typescript and React

COMP 315: Cloud Computing for E-Commerce

May 3, 2024

1 Introduction

A reactive website will scale and reorder elements to suit the screen size of the device being used to access it. A dynamic website will update the elements of the page without the need for a full reload. In this assignment you will expand upon a skeleton program in order to build a dynamic and reactive e-commerce front end website.

2 Learning outcomes

By the end of this assignment, you will:

- Be able to implement functions using Typescript.
- Be familiar with how to use React components and hooks.
- Have a template website that you could expand upon for a portfolio piece.

3 Problem description

For this task, you have been provided with a skeleton website, as well as the assets to populate it with. At the moment the site displays the: name, picture, rating, and price of a collection of items for sale. If you type into the search bar, it will only display items that have your search term within their name. You must add the following functionality:

- An indicator showing the number of search results or products available.
- Sorting the items by: name, price, or rating.
- The ability to show only in stock items in the search results.
- Adding or removing items from the shopping basket.
- A total cost of products in the shopping basket.

4 Initial setup

The skeleton code has been provided for you, which is a basic e-commerce website similar to the one shown in lectures. Ensure that you have *Node.js* installed on your computer, this should come with *Vite*. Download the zip file of this code and extract it to a suitable place on your computer. Navigate to that folder in your terminal, and type *npm install*. Once this installation has completed, type in *npm run dev*, which should host the website locally for you. Take the localhost address shown in the output and type it into your browser to see the website. There are 4 JSON files included in the 'Assets' folder, which are *random_products* 1, 100, 150, and 175. Each of these JSON files contains a list of products to be shown on the website. Each product has the attributes outlined in Table 1, with the images being generated using Adobe Firefly.

Attribute name	Note
ID	This is a unique identifier for each product, and is an integer
name	The name for each product.
price	The price of the product in pounds.
category	This is the general category of the product.
quantity	The number of this product that is currently available in stock. This is a non-negative integer.
rating	This is a real number rating of the product between 0 and 5.
image_link	The file location of the promotional image.

Table 1: The attributes that are stored for each product

5 Developing the website

5.1 The results indicator

When searching for products, it is often useful to know exactly how many products the current search has returned. This can help make the website feel more reactive. In the $results - indicator$ paragraph tag, add a notification about how many results or products the current search query has returned. If the search bar is empty, then the the output should be n *Products* where n is the number of products. If there is only a single product then the output should be 1 *Product*. If the search bar is not empty, then the output should be m *Results* where m is the number of products returned by the search query. If there is only a single product returned by the query, then it should say 1 *Result*. If there are no results returned by a query then the output should be $No\, search\, results\, found$.

5.2 Enhance search functionality

When looking at a list of products, a useful feature is being able to sort them by some attribute such as price or rating. Add functionality to the *select* tag inside of the $search - bar$, so changing the selected option will result in that form of sorting being applied to the results. By default the product list should be sorted by name alphabetically from A to Z. Once this task has been completed, add the following functionality to the *inStock* checkbox input. When this checkbox is ticked, the results should only include products that have a quantity larger than 0. Unticking this checkbox should reverse that behaviour. Hint: this can be accomplished by using a combination of a state and a hook.

5.3 Adding to the shopping basket

Each product currently has a button underneath that says 'Add to basket'. Update this code so that if the quantity of available product is 0, the button instead says 'Out of stock' and is disabled. Add a function to the 'Add to basket' button that passes the information to a shopping basket variable in App.tsx. This variable should be a list of type *BasketItem*. Adding multiple instances of the same product should increase the quantity property of the relevant basket instance. Do not worry about disabling the product's button if the quantity added to the basket is more than the quantity available. Hint: The parent/child example given in Tutorial 4 - question 7 can give you a good starting point.

5.4 Visualising the basket

Now that the data about the basket is being collected, we should visualise it for the user. If there are no items in the basket then the $shopping - area$ div should contain a paragraph text saying 'Your basket is empty'. If the shopping basket variable contains a product, then the $shopping - area$ div should contain that information. Each item in the basket should be surrounded by a div with the class 'shopping-row', and a suitable key such as the name of the item. Inside of that div there should be another div with the class 'shopping-information', and a button with 'Remove' text. The 'shopping-information' div should contain a paragraph tag which shows the information about the product in the format $[Productname](£[Productprice]) - [Productquantity]$. When the 'Remove' button is pressed, then the quantity of that product in the basket should be reduced by 1. If pressing that button reduces the quantity of the product to 0, then that item should be removed from the shopping basket. At the bottom of the $shopping - area$ div should be a paragraph tag with the total cost of the shopping basket. This should be in the form of $Total\ :£[Totalbasketcost]$. This value should be shown to 2 decimal places.

6 Marking

The completed assignment should be compressed into a zip file, and then submitted through the Canvas submission system. This will account for 10% of your overall module score. You may use any library that comes with a default installation of Node.js. Each variable should have the appropriate type, if the 'any' type is required than a comment must be included that justifies it's use. This rule is in place to encourage the use of Typescript type checking instead of just Javascript. Your work will be submitted to an automatic plagiarism/-collusion detection system, and those exceeding a threshold will be reported to the Academic Integrity Officer for investigation regarding adhesion to the university's policy https://www.liverpool.ac.uk/media/livacuk/tqsd/code-of-practice-on-assessment/appendix_L_cop_assess.pdf.

7 Deadline

The deadline is 23:59 GMT Monday the 13th of May 2024. Late submissions will have the typical 5% penalty applied for each day late, up to 5 days. Submissions after this time will not be marked. https://www.liverpool.ac.uk/aqsd/academic-codes-of-practice/code-of-practice-on-assessment/

8 Expansions for a portfolio piece

If you decided to expand upon this short example in order to create a portfolio piece, which I stress is not something you have to do for the assignment, then you will need to make several changes. My suggestion for these changes would be:

- Redesign the site to be more visually appealing, focus on the reactivity and showing off that it works on all form factors of device.
- Use Next.js to add individual pages for each product. This will also mean that you'll have to look into how to allow the user to return to the search results.
- Connect to a database, such as Neo4j, as this is crucial in showing you understand role of the front end.