**Introduction**

This document provides an overview of the implementation of the Caffein project. It explains the components (its states and methods), the challenges faced, feedback from experts and future steps.

**Products Component**

The `Products` component is responsible for displaying the product listing and managing a shopping cart.

**States:**

* Products: An array of product information from the server
* Cart: An array of items in the shopping cart
* Orders: An array of order details
* OrderID: Integer of the following order’s ID

**Methods:**

* handleDelete(id): A function to delete a product by its ID and reload the page.
* addToCart(product): A function to add a product to the shopping cart.
* removeFromCart(productid): A function to remove a product from the shopping cart.
* calculateTotalPrice(): A function to calculate the total price of items in the shopping cart.
* handleCheckout(): A function to handle the checkout process and create a new order.

This component renders the product listing, each displaying the product name, category, roast level, amount, price, purpose and flavor. It also shows buttons for deleting, editing and adding the product to the cart. The shopping cart section displays the cart’s contents, the amount of each product type, and the total price and allows the user to remove the items and checkout. The registered orders are shown in a separate section, assigning a unique ID for each order and showing the order information.

**New Component**

The `New` component is responsible for adding new products to the app.

**States:**

* Product: an object containing information about the new product, which are the product name, category, roast level, purpose, flavour, amount, and image URL.

**Methods:**

* handleChange(e): A function to update the product state from the data in the form fields inputted by the user.
* handleClick(e): A function to send an HTTP POST request to add a new product. The user will be redirected back to the products page if it is successful.
* navigate: A function provided by the `useNavigate` hook that will redirect the user upon the successful of adding a new product.

This component renders a form where the user can fill in the product information of the new product. The fields are the product name, category, roast level, purpose, flavour, amount, and image URL. When the submit button is clicked, the user will be redirected to the products page if successful.

**Edit Component**

The `Edit` component is responsible for editing the data of an existing product in the app.

**States:**

* Product: an object containing information about the existing product, which are the product name, category, roast level, purpose, flavour, amount, and image URL.

**Methods:**

* handleChange(e): A function to update the product state from the data in the form fields inputted by the user.
* handleClick(e): A function to send an HTTP PUT request with the updated product details to modify the existing product. The user will be redirected back to the products page if it is successful.
* navigate: A function provided by the `useNavigate` hook that will redirect the user upon successfully modifying an existing product.

This component renders a form where the user can fill in the product information to change an existing product’s data. The fields are the product name, category, roast level, purpose, flavour, amount, and image URL. When the submit button is clicked, the user will be redirected to the products page if successful.

**Challenges**

During the making of the project, I encountered some challenges. The most challenging part for me is implementing the shopping cart.

1. **The `addToCart` Function**

A computer code on a black background

Description automatically generated

1The `addToCart` function snippet

**Explanation:**

This function is used to add the products to the cart. First, it checks whether the product already exists in the cart. If it already exists, it updates the cart by mapping through it and adds the quantity of the matching product by 1. If it doesn’t exist, it adds the product to the cart and sets the quantity to 1.

**Initial Solution:**

Initially, there was no “quantity” implemented. It directly adds the product to the cart, which causes the shopping cart to be a wall of text. For example, if you add a few products such as A, B, and C, then A again, it will appear as A, B, C, A. It will be very inconvenient for the users to know how many of each item they have in their cart, especially if it is a significant amount.

1. **The `removeFromCart` Function**

A computer screen with colorful text

Description automatically generated

2The `removeFromCart` function snippet

**Explanation:**

This function is used to remove a product from the cart. First, it checks whether the item id matches with the product id. If it matches, it will reduce the quantity of the product by 1. However, if the quantity of the product is already 1, it will be set to null. Then, the cart will be filtered, and the items whose quantity is null will be deleted.

**Alternative:**  
An alternative way is setting the quantity to 0 instead of null. In the end, there is no significant effect of using this method instead of the final solution. However, I prefer using null instead of 0. In my opinion, setting the quantity to null is a more apparent indicator that the item needs to be removed. This can be useful if a “selected items only checkout” will be added in the future. This enables the user to proceed to checkout only with the selected items but not delete the other items in the cart.

**Initial Solution:**  
Initially, like the `addToCart` function, quantity was not used. In this function, in the absence of quantity, deleting a product won’t work correctly. Instead of decreasing the product quantity by one or deleting it, what happened was that all of the products with the same id were removed from the cart despite the different quantities.

1. **The `handleCheckout` Function**

A screenshot of a computer program

Description automatically generated

3The `handleCheckout` function snippet

**Explanation:**

This function does the “checkout” by creating a new order object, which consists of orderID (unique id, increments by one), items (which are in the cart) and total price. It then adds the created order to the array of orders and clears the current cart.

A spreader function was used for the order items to copy the cart array so that the array remained unchanged. Without a spreader, the order’s items and cart will reference the same array. This will lead to a situation where when the cart is cleared, the orders will also be removed.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Feedback from Expert:**

To check the quality of my work, I did a validation meeting with an expert (Patrick). The feedback that I received mainly focused on the documentation, such as explaining some of the challenging parts, how I came up with it, and the issues that I encountered. I have incorporated that feedback in this document.

**Future Steps:**

Many improvements can be made for Caffein, but these are not possible right now due to time constraints. Here are some of the crucial ones that I composed:

* Fix the validation for the “new product” and “edit product” forms. Currently, the validation only checks whether there are any empty boxes and redirects the user back to the “products” page if there is an error. It would be beneficial to have a text showing the error on the page, something like “Error: fill in all the blanks”.
* Modify the “amount” column in the “products” table in the database by changing it from VARCHAR to INT. Currently, because it uses VARCHAR, the user is able to type gibberish and still be able to add a product. By changing it to INT and adding a dropdown menu on the form (gr/ml/pcs), the data will be much more accurate.
* Restructuring the code. As Caffein was made mainly for self-study, it is not quite organised. In the future, it would be advisable to restructure the code by separating the functions related to product management, cart, and orders to make it more readable and maintainable.

**Conclusion**

The main objectives of this project (learning new technologies and e-commerce fundamentals) have been completed. This project became a great learning experience in web development, From implementing the frontend with React, the backend using Node.js, and the database using MySQL. The main e-commerce features, such as the CRUD (create, read, update, delete) operations and the shopping cart, were also implemented. The knowledge gained from this project could serve as a solid foundation for future learning.