PUI Assignment 6B

Reflection

During this assignment, I ran into the most difficulty in implementing the cart item remove button functionality. At first, I underestimated the complexity of the task. I started working on this portion of the assignment by creating a new JavaScript file and immediately starting to write code without first mapping out all of the steps that would be required to correctly implement the cart removal. However, I soon realized that the task was a bit more involved than I had anticipated. I started out by implementing the functionality to remove the item from the `cart` array variable, and the cart stored in localStorage. Afterwards, I realized that there were also a lot of visual changes that needed to take place across the page. Things like updating the number of items displayed in the basket, updating the total cost, and displaying an appropriate message if the cart was now empty. Because of this, I had to make sure that the adequate information was stored in each product, and added a `.price` field to the `Product` objects, so that the total could be easily updated. Each step in itself was not too difficult to implement, but it would have been more efficient to map out all of the steps required beforehand (in pseudocode perhaps) so that I could approach the task in a more methodical way.

For the rest of the assignment, I found that the best way to debug small errors was to use the browser developer tools. These tools were very helpful in doing things such as running tests in the console to check the values of different variables, looking at local storage, and seeing how HTML elements ended up being aligned on the page.

Programming Concepts

Throughout this assignment, I learned and applied many new programming concepts in JavaScript. One of the first major programming concepts I used was JS Objects. I initially used this to create `Product` objects to store information about products that customers added to cart. I found this to be an extremely useful construct, and also used it later on to add additional functionality to my website. Below is the constructor for the `Product` object that I implemented in my main.js file.

```
// From main.js - Constructor for Product object
function Product(flavor, glazing, quantity, price, image, itemId) {
   this.flavor = flavor;
   this.glazing = glazing;
   this.quantity = quantity;
   this.price = price;
   this.image = image;
   this.itemId = itemId;
}
// From product-details.js - instantiation of Product object
const item = new Product(flavor, glazing, quantity, price, image, itemId);
```

Another programming concept I implemented was storing the cart as an array variable. This made it easy to add and remove items from the cart using built-in JS functions I learned about

such as `push(elem)` (appends new element to end of array), `findIndex(function(){})` (returns index of element that the provided function returns true), and `splice(startIndex, n)` (removes n elements starting at the startIndex).

```
// From main.js - Instantiation of cart array
var storedCart = JSON.parse(localStorage.getItem("savedCart"));
var cart = storedCart ? storedCart : [];

// From product-details.js - Adding item to cart
cart.push(item);

// From cart.js - Removing item from cart
const index = cart.findIndex(function (item) {
   if ((item.flavor == product.flavor)
   && (item.glazing == product.glazing)
   && (item.quantity == product.quantity)) {
       return true;
   }
});
cart.splice(index, 1);
```

Another new programming concept that I implemented is using different JS files for page-specific functionality. As can already be observed from previous examples, I used multiple JS files to separate and organize page specific code, making everything modular and easy to keep track of. Each HTML file included the main.js script. In this file, I included variables and object definitions that would be used across all pages such as the cart and number of items in the cart. Page specific files such as cart.js, were included in the cart html page, and handled functionality such as rendering cart items.

```
// From basket.html - Including two scripts for cart page in <head>
<script defer src="main.js" type="text/javascript"></script>
<script defer src="cart.js" type="text/javascript"></script>

// From cart.js - Example of page specific functionality
function renderCart() {
   if (cart.length == 0) {
       const clone = emptyTemplate.content.cloneNode(true);
       basketItems.appendChild(clone);
       total.innerText = "Total: $0.00";
   } else {
       totalPrice = 0;
       for (let item of cart){
           showProductInCart(item);
```

```
            totalPrice = totalPrice + item.price;
        }
        total.innerText = `Total: $${totalPrice}.00`
    }
    // Render wishlist
    if (wishlist.length != 0) {
        for (let item of wishlist) {
            showProductInWishlist(item);
        }
    }
}
renderCart();
```

A fourth programming concept I used in this project was using <template> tags in HTML. Content within template tags is not initially rendered when the page loads. However, using scripting, you can create clones of templates, customize them, and dynamically insert them into the DOM to have them rendered on the page. In this assignment, this construct was particularly useful in rendering the cart page, since the items displayed in the cart depends on what is inside the cart variable. I defined the structure of a cart item and all of the information that would go inside of it inside template tags. Then, in the linked JavaScript file, in order to render the items, I used the `cloneNode()` function to clone the template, `querySelector()` function to modify the clone, and `appendChild()` function to add it to the page. I later on used this construct to dynamically render items on the menu page as well, so that menu items could be easily added, removed, and customized based on the bakery's offerings.

```
<!-- From basket.html - Basket item template definition -->
<template id="basket-item-template">
    <div class="basket-item">
        <div class="basket-item-left">
            <img class="basket-item-img" src="assets/original.png" alt="Original
Flavor">
        </div>
        <div class="basket-item-right">
            <div class="item-line-1">
                <p class="basket-quantity"></p>
                <button class="basket-name"></button>
                <p class="basket-price"></p>
            </div>
            <div class="item-line-2">
                <p class="basket-glazing"></p>
                <div class="basket-edit-buttons">
                    <button class="remove-item-btn" onclick="">X</button>
```

```
            </div>
        </div>
    </div>
</template>

// From cart.js - Example of cloning, manipulating, and adding new node to page
var basketItems = document.getElementById("basket-items");
const itemTemplate = document.getElementById("basket-item-template");
var clone = itemTemplate.content.cloneNode(true);
var basketName = clone.querySelector(".basket-name");
basketName.innerText = product.flavor;
basketItems.appendChild(clone);
```

A fifth programming concept I used was dynamically adding event listeners to buttons in JavaScript with `addEventListener()`. This was particularly helpful when creating the remove buttons for the shopping cart items. It made the HTML template code cleaner by not including the “onclick” attribute. Additionally, it allowed me to group all of the code for creating a new cart element together in one function. This also allowed me to use variables from outside of the onClick function such as `product` which contained the information about the specific product in the cart. This way, I could directly access values of that product like `product.quantity` instead of having to grab the information from parentNodes.

```
// From product-details.js - Adding event listener to remove button
var button = clone.querySelector(".remove-item-btn");
button.addEventListener("click", function () {
    // Remove item from stored cart list
    const index = cart.findIndex(function (item) {
        if ((item.flavor == product.flavor)
        && (item.glazing == product.glazing)
        && (item.quantity == product.quantity)) {
            return true;
        }
    });
    cart.splice(index, 1);
    localStorage.setItem("savedCart", JSON.stringify(cart));
    // Update displayed basket items
    basketItems.removeChild(this.parentNode.parentNode.parentNode.parentNode);
    // Update total price
    totalPrice = totalPrice - product.price;
```

```
    total.innerText = `Total: $${totalPrice}.00`;
    // Update number of items in header
    itemsInCart = itemsInCart - product.quantity;
    localStorage.setItem("itemsInCart", itemsInCart);
    if (itemsInCart != 0) {
        itemsInCartElement.innerText = itemsInCart;
    } else { // Display empty basket notice
        itemsInCartElement.innerText = "";
        const clone = emptyTemplate.content.cloneNode(true);
        basketItems.appendChild(clone);
    }
});
```