6B Reflection

Connor Shannon

Although I have experience with a lot of different programming languages, I only started learning javascript this semester and I've had a mixed relationship with it. While the flexibility of js can be convenient, it also makes it hard to choose the best way to approach a problem. I also really dislike the hacky way of injecting html through JS for dynamic content. While adding in the "remove from cart" button to the cart page I got stuck debugging a problem in the html string. I ended up dealing with this by moving the html into the html file while testing then moving it back into the javascript once I got it working.

I also had trouble updating the layout after adding dynamic content. This was especially a problem in the cart page. I overcame this by using the cart length to determine the height of the container then changing it in the javascript file.

Examples

Local Storage - Local storage provides an easy way to store and retrieve user-specific data while they are using the site. Without this, the data would be reset every time the user reloaded the page. Below is an example of how I removed items from the cart using local storage.

```
function removeFromCart(id){
  cart = JSON.parse(window.localStorage.getItem("cart"));
  cart = cart.filter((cartItem) => cartItem.id !==id);
  window.localStorage.setItem('cart', JSON.stringify(cart));
  window.localStorage.setItem('cartCount', cart.filter((cartItem) => cartItem.inCart === true).length);
  location.reload();
};
```

Filtering/ForEach/Map - I found that the filtering function was really helpful in a few different scenarios. It is a nice, concise way to get sort the list. I used it when implementing the wishlist. Rather than having two different lists, I just had one cart list and added an inCart boolean to signify which were in the cart and which were in the wishlist. In addition to filtering, I used the the .map and .forEach functions to loop through items.

```
function getItemCount(cart) {
  let q = 0;
  cart.filter((cartItem) => cartItem.inCart === true).forEach(cartItem => {
      q += parseInt(cartItem.quantity);
  });
  return q;
  }
```

Json/ Maps - I decided not to use a class or prototype for this project because I found it more convenient to keep the data in map objects. I like how javascript allows you to use both object.attribute and object["attribute"] to get specific attributes. Here is an example of how I set the cart object.

```
let selected = {
  "id":product.id,
  "name": product.name,
  "price": product.price,
  "color":selection[0].classList[1],
  "image":document.getElementById('center-img').src,
  "size":selection[1].classList[1],
  "quantity":document.getElementById('quantity').selectedIndex+1,
  "inCart":inCart,
};
```

DOM Manipulation - I also learned a number of concepts around how to changed the DOM of the html from javascript. For the products page, I created div elements using the document.createElement, added eventListeners to listen for on click events, and appended the elements to the parent html node.

```
productsList.forEach((product)=>{
      var productCard = document.createElement("div");
      productCard.className = "product-card";
      productCard.innerHTML=`
            <img src=${product.default} alt="dog">
          <h4>${product.name}</h4>
          <div class  = "items-row">
              <small>Avail in 4 colors</small>
              <span class="fa fa-star"> <b>${product.rating}</b></span>
          </div>
          <p class="product-price">${product.price}</p>
      `;
      // Add onClick listener for when the product is chosen
      productCard.addEventListener("click", ()=>{
        window.localStorage.setItem("activeProduct", JSON.stringify(product));
        window.location.href = "details.html";
      });
      productsHtml.appendChild(productCard);
```

I also used the "selected" name to change the css of selected elements and store the current selection within the dom. In order to make this work, I used getElementsByClassName to get the items and added or removed the selected tag from the classList to update selection.

```
let oldSelection = document.getElementsByClassName(`${optionType} selected`)[0]
if (oldSelection) oldSelection.classList.remove("selected");
newSelection.classList.add("selected");
```