PUI HW6

JavaScript Functionality

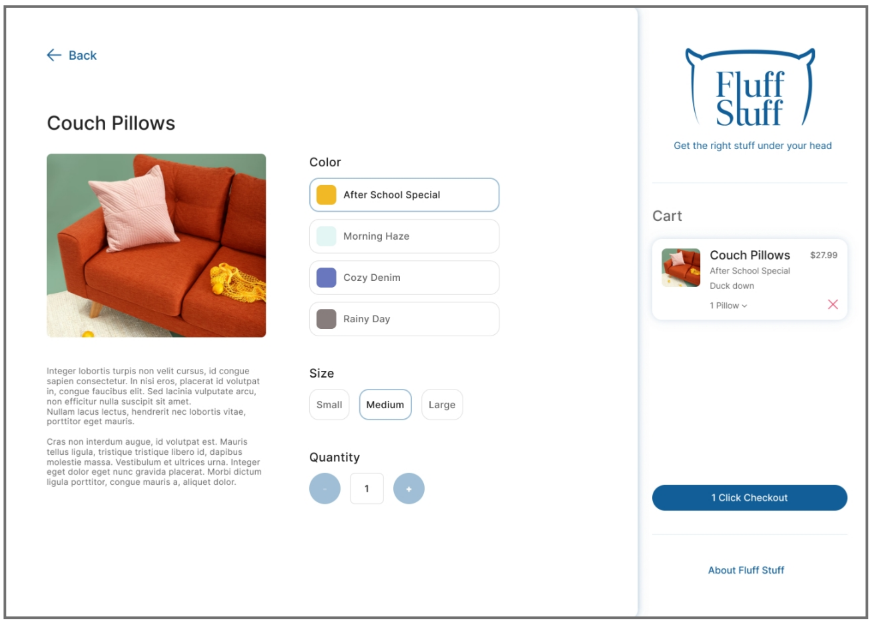
Sai Vinod Kota

Repo: https://github.com/vinod-r/homework-6b

Page: https://vinod-r.github.io/homework-6b/

Adapting my design to include the Cart

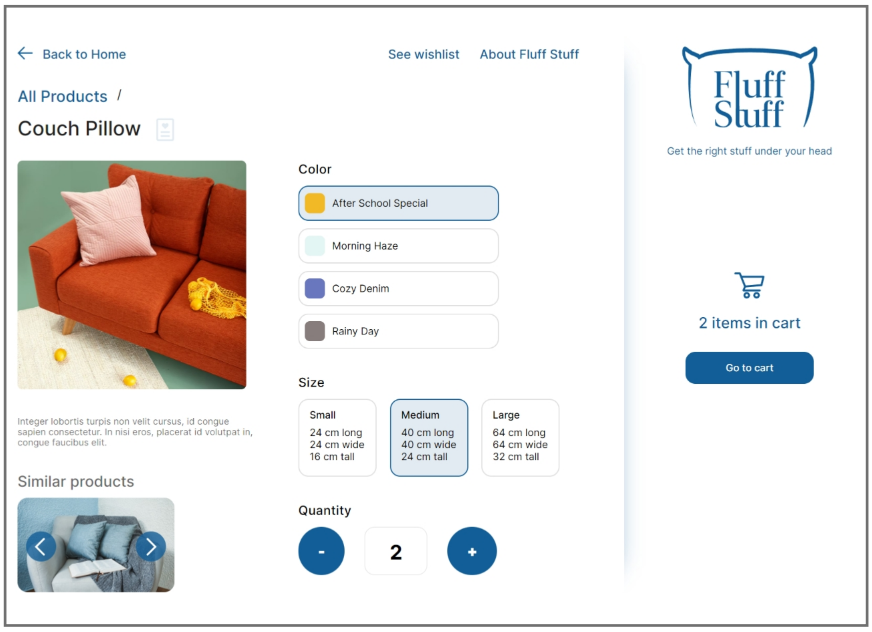
My initial design created early in this process incorporated the cart as a static element on all pages. It would represent the items in the cart on every page and give the users the option to immediately check out. However, to fit the requirements of the assignment, I adapted my design to make the cart display be a simple presentation of the number of items in the cart with a button to check out the actual cart page separately.



Integer lobortis turpis non velit cursus, id congue sapien consectetur. In nisi eros, placerat id volutpat in, congue faucibus elit. Sed lacinia vulputate arcu, non efficitur nulla suscipit sit amet.
Nullam lacus lectus, hendrerit nec lobortis vitae, porttitor eget mauris.

Cras non interdum augue, id volutpat est. Mauris tellus ligula, tristique tristique libero id, dapibus molestie massa. Vestibulum et ultrices urna. Integer eget dolor eget nunc gravida placerat. Morbi dictum ligula porttitor, congue mauris a, aliquet dolor.

Original Figma Protoype



Integer lobortis turpis non velit cursus, id congue sapien consectetur. In nisi eros, placerat id volutpat in, congue faucibus elit.

Final Implementation

Implementation

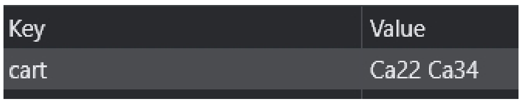
I did not face too many issues in implementing the remove functionality. It was rather straight forward as I had already worked on a lot of the dynamic functionality of the cart during Homework 6a.

While most people took the approach of saving their cart items in Local Storage as a JavaScript object, I took a slightly different approach. I felt that a JavaScript object was overkill for this simple functionality. It makes sense to have a JavaScript object where there large number or permutations or if any of the fields can be dynamic, like names or addresses.

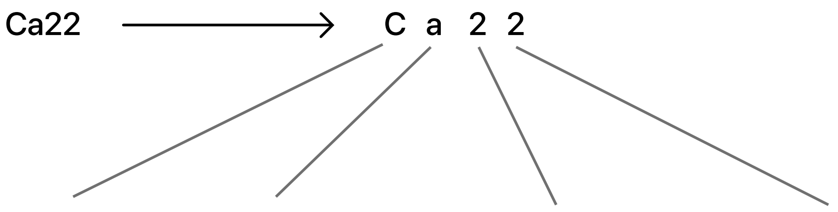
In this case, each cart item was a possible combination of a few options (Pillow type - 4 options, Size - 3 options, Color - 4 options and Quantity.) Rather than run a JavaScript object through Local Storage, which would require encoding and parsing since Local Storage can only store string values, I chose to form a simple hash system that uses a series of 4 or 5 (if the quantity is 10) characters. By using a series of hashes in a string rather than a complex JS object, I was able to keep my cart functionality quite simple and straight forward.

If the cart required a more complex set if data for it's functioning, I would certainly rely on a JS object to keep things straight forward but in this case, I feel my approach was simpler and easier.

Cart Item in Local Storage



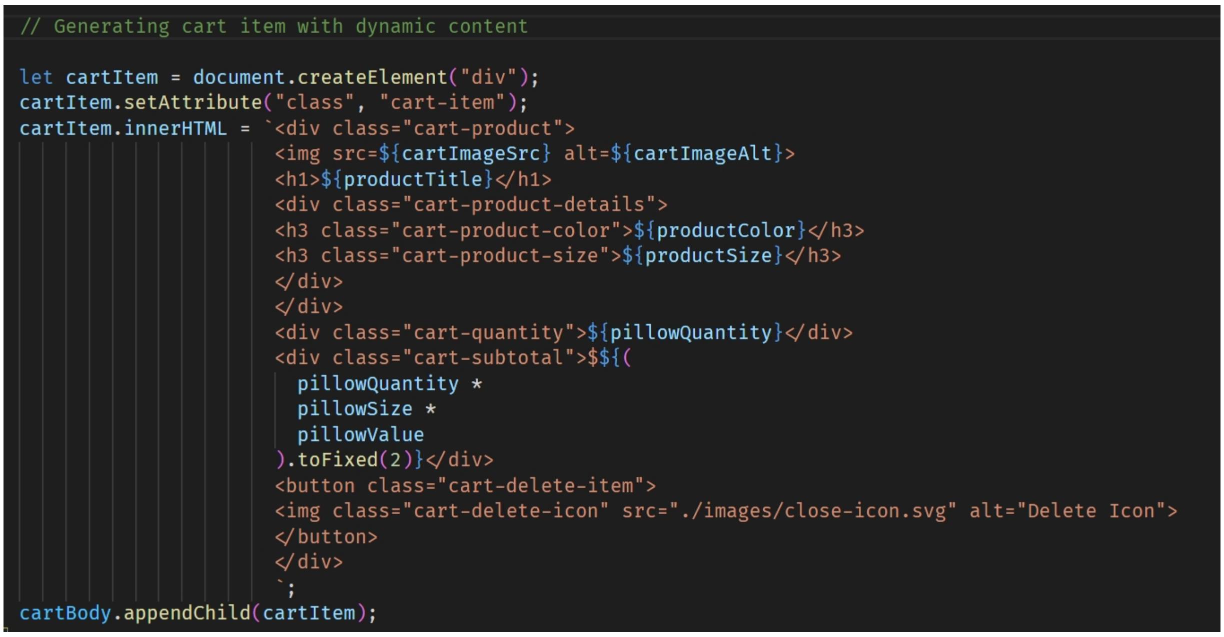
Key	Value
cart	Ca22 Ca34



1ST CHARACTER	2ND CHARACTER	3RD CHARACTER	4TH AND 5TH CHARACTERS
Type	Color	Size	Quantity
Couch	**After School Special**	**Medium**	**Medium**
C - Couch	A - After School Special	1 - Small	1 Digit for Qty less than 9.
R - Round	M - Morning Haze	2 - Medium	2 Digits for Qty of 10.
F - Floor Pouf	C - Cozy Denim	3 - Large	
B - Bed	R - Rainy Day		

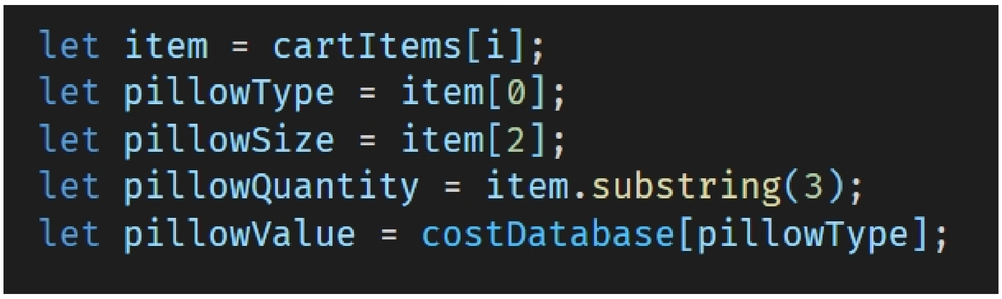
Storing the Cart Items as a string of hashes meant that I can easily split them up into an array of strings and easily access any of the characters to rebuild the cart item on the cart page.

On the cart page, I used a simple foreach loop to loop through all the items in the array and generate the HTML for each of them. Initially I had used the approach of creating each DOM element individually and manually appending them to their respective parents and then updating their inner HTML. But upon experimenting, I discovered that I could just write the entire cart item out in plain HTML as inner HTML for a parent div, much like I would with JSX in React. I had never considered this option prior to this point so it was a useful learning experience.



```
// Generating cart item with dynamic content

let cartItem = document.createElement("div");
cartItem.setAttribute("class", "cart-item");
cartItem.innerHTML = `<div class="cart-product">
                      <img src=${cartImageSrc} alt=${cartImageAlt}>
                      <h1>${productTitle}</h1>
                      <div class="cart-product-details">
                      <h3 class="cart-product-color">${productColor}</h3>
                      <h3 class="cart-product-size">${productSize}</h3>
                      </div>
                      </div>
                      <div class="cart-quantity">${pillowQuantity}</div>
                      <div class="cart-subtotal">$${(
                        pillowQuantity *
                        pillowSize *
                        pillowValue
                      ).toFixed(2)}</div>
                      <button class="cart-delete-item">
                      <img class="cart-delete-icon" src="./images/close-icon.svg" alt="Delete Icon">
                      </button>
                      </div>
                      `;
cartBody.appendChild(cartItem);
```

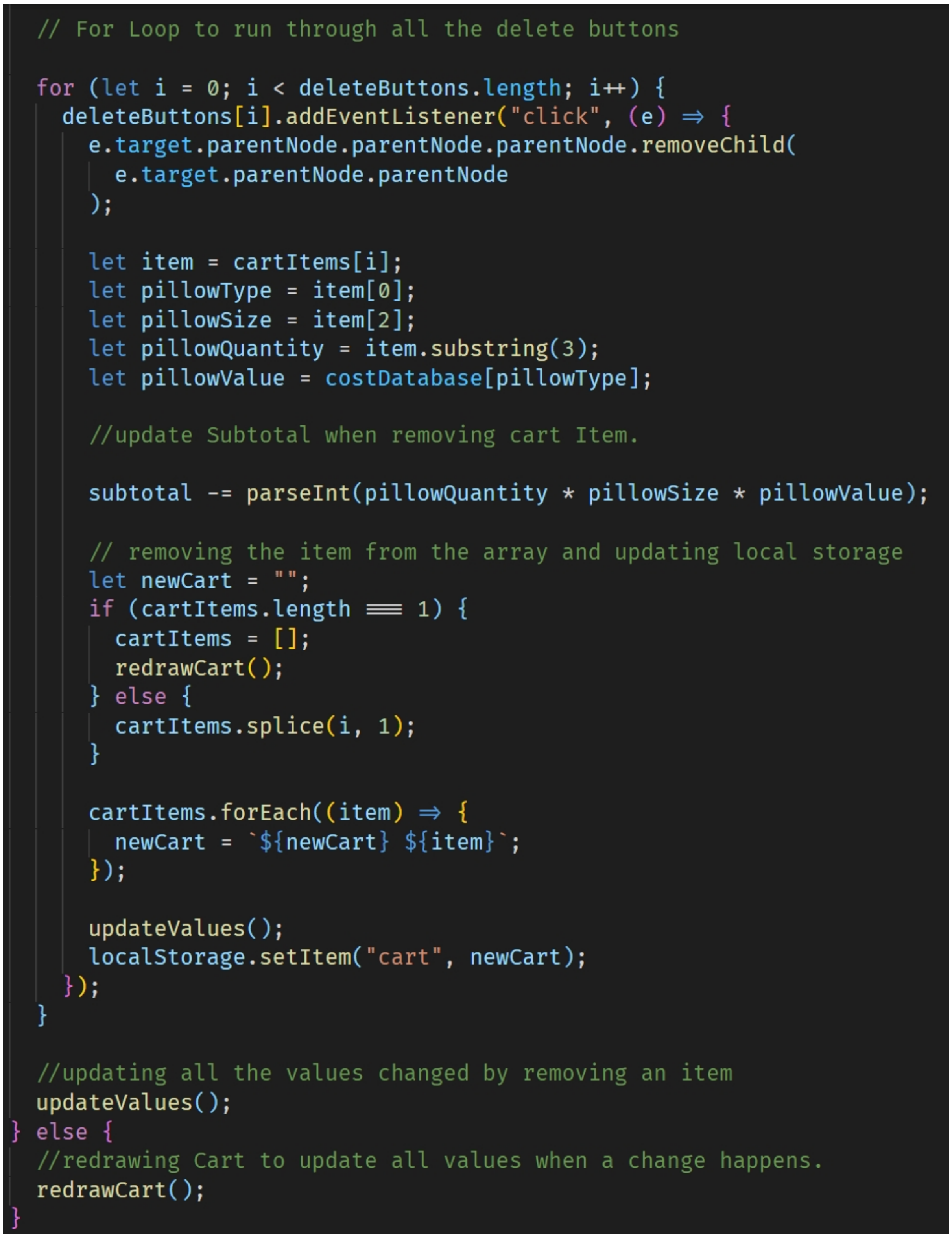
Code to generate cart item



```
let item = cartItems[i];
let pillowType = item[0];
let pillowSize = item[2];
let pillowQuantity = item.substring(3);
let pillowValue = costDatabase[pillowType];
```

Decoding the hash into product details. cartItems is the array derived from Local Storage.

After that, I used a for loop to add an Event Listener to all the remove buttons for each cart item. By using a regular for loop, I was able to keep a track of the iterator, which meant that I could also remove the item at that specific index from the cart item array. This kept the display on the page consistent with the actual state of the cart item in the Local Storage.



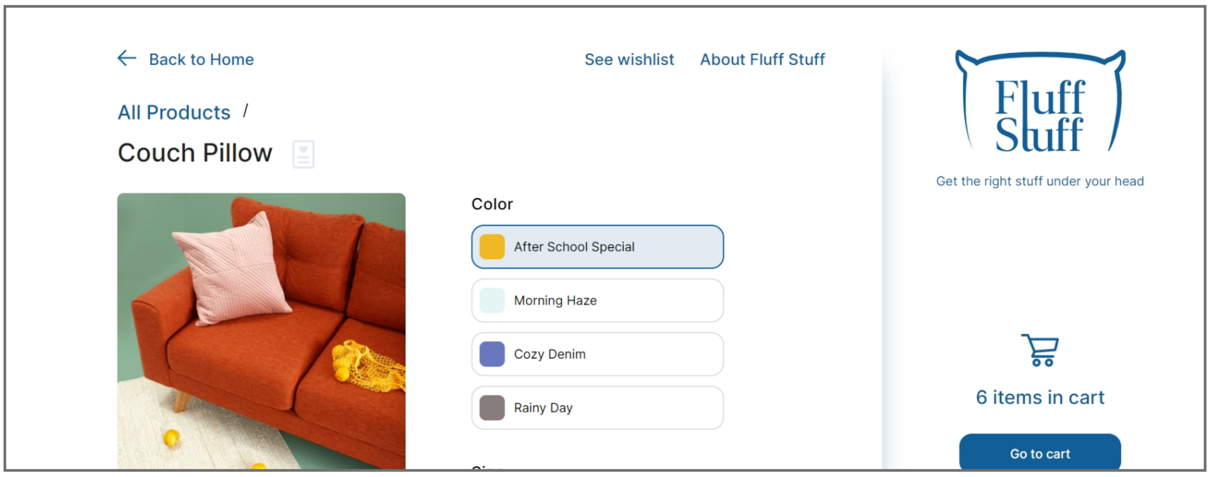
```
  // For Loop to run through all the delete buttons

  for (let i = 0; i < deleteButtons.length; i++) {
    deleteButtons[i].addEventListener("click", (e) => {
      e.target.parentNode.parentNode.parentNode.removeChild(
        e.target.parentNode.parentNode
      );

      let item = cartItems[i];
      let pillowType = item[0];
      let pillowSize = item[2];
      let pillowQuantity = item.substring(3);
      let pillowValue = costDatabase[pillowType];

      //update Subtotal when removing cart Item.

      subtotal -= parseInt(pillowQuantity * pillowSize * pillowValue);

      // removing the item from the array and updating local storage
      let newCart = "";
      if (cartItems.length === 1) {
        cartItems = [];
        redrawCart();
      } else {
        cartItems.splice(i, 1);
      }

      cartItems.forEach((item) => {
        newCart = `${newCart} ${item}`;
      });

      updateValues();
      localStorage.setItem("cart", newCart);
    });
  }

  //updating all the values changed by removing an item
  updateValues();
} else {
  //redrawing Cart to update all values when a change happens.
  redrawCart();
}
```

Code to update cart and all dependent elements when removing an item.

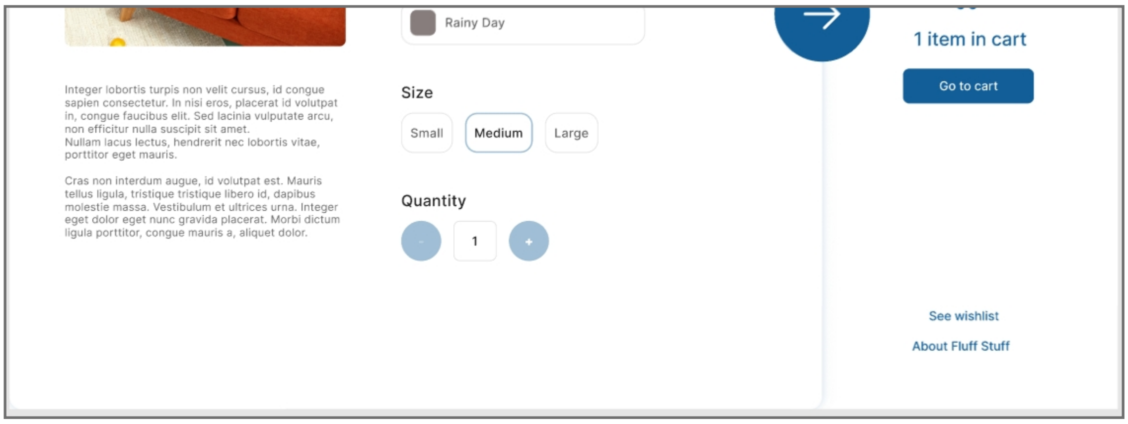
Addressing Usability Issues

During the in class presentation and critique session, I received some valuable feedback from my peers and my TA, pointing out some issues with my design.

The first of these was the positioning of the About and Wishlist (extra credit feature) at the bottom of the fixed, right-side navigation. It was common consensus that it would make more for users if this was at the top of the screen in some manner so that it was easily discoverable and usable.



Updated position of nav elements at the top of the page



Integer lobortis turpis non velit cursus, id congue sapien consectetur. In nisi eros, placerat id volutpat in, congue faucibus elit. Sed lacinia vulputate arcu, non efficitur nulla suscipit sit amet. Nullam lacus lectus, hendrerit nec lobortis vitae, porttitor eget mauris.

Cras non interdum augue, id volutpat est. Mauris tellus ligula, tristique tristique libero id, dapibus molestie massa. Vestibulum et ultrices urna. Integer eget dolor eget nunc gravida placerat. Morbi dictum ligula porttitor, congue mauris a, aliquet dolor.

Original position of the nav elements at the bottom of the side bar

Useful Programming Concepts

CONCEPT 1

Page Specific Javascript

Rather than have all my JavaScript functionality forced into one file (especially considering the functionality for the extra credit), I chose to break up the code and make it modular so that only the functionality that is required on specific pages is loaded. While this can be a quick way to save some loading time, that is often negligible. An added benefit was that any minor errors that could be thrown when a script looks for a specific DOM element that is not on a given page can be avoided.

CONCEPT 2

Using Hashes over JavaScript Objects

While the course was directed towards using JavaScript Objects, I found them to be overkill for the simple task of sharing cart state over multiple pages. I did use JavaScript objects as dictionaries to rebuild my hashes into full fledged objects. However, the actual data being sent through the Local Storage was still ultimately a series of hashes in a string - the only datatype Local Storage can hold.

CONCEPT 3

Using JSX style InnerHTML over Templates.

Rather than use the Template tag to generate cart items on the fly, I used a JSX style InnerHTML method. This allowed me to easily change out the dynamic content of an HTML block with variables while also being much easier to write than using specific DOM manipulation commands within JavaScript to build a complex HTML structure required to fit the styling needs of the Cart Item. I believe this greatly reduced the cognitive load required when constructing that HTML block inside of JavaScript.

CONCEPT 4

Adding event listeners in JavaScript

Rather than use inline function calls in the HTML file, I located DOM elements with JavaScript to add event listeners. Not only does this make the HTML file much cleaner and easier to read, it also lets me add interactivity to dynamically generated content like the delete buttons of cart items.

CONCEPT 5

Using JavaScript Objects as Dictionaries

As I was sending my cart items as a series of characters, the easiest and least error prone way to translate them into full-fledged details for the cart again was to take the individual characters and match them up to JavaScript objects with key-value pairs corresponding to the individual details (Type, Color, Price). By doing this, I was able to simplify a lot of the encoding and decoding work.

Extra Credit

EXTRA CREDIT 1

Wishlist

I added the ability for users to tap on an icon by the name of the product on the product details page and add it to a wish list. The icon on the product page is dynamic and updates to show if an item is wish listed or not and is persistent so it shows the correct state even when leaving the page and returning later. I also added the wish list page where users can go and see all the items on their wish list and remove them from there if desired.

The functionality for adding to and maintaining the wish list was similar to the cart but rather than require a whole hash, it only required 1 character per item to note its type and it does not support more than 1 instance of the same product type. A new wrinkle for the wish list feature as compared to the cart functionality is that the item can be removed from the wish list directly from the product details page.

EXTRA CREDIT 2

Carousel

I added a similar products carousel to every product page that showcases similar products. As per the requirements, it can be advanced manually by pressing a button but also advances by itself every 2 seconds. Based on critique for my website in the Lab critique session, I adjusted the visual prominence of the carousel.

Functionally, the carousel was simple to implement. It is an array of URLs to the image files which are looped through. Upon reaching the final index value, the function sets the index back to 0, making it a perfect loop. However, when I was implementing the automatic advancement feature, I incorporated it into the manual advance function which was creating a feedback loop of a new auto advance each time I clicked. To solve this issue, I separated the two into individual and unrelated functions so they do not end up making the cart advance too fast too quickly.

EXTRA CREDIT 3

Fully featured product pages

As I had already made product pages for all the pillow types, I ensured that I included the dynamic updating of the page based on user selections as well as add to cart functionality on all of those pages.

External Resources

All images used in the site are sourced from Unsplash.com
Unsplash Licence: https://unsplash.com/license