Code Link: https://fluffstuff.netlify.app/
Live Page Link: https://github.com/ingan274/fluff-stuff

Executive Summary

In this homework, I added functionality to my Fluff Stuff website and transferred my code structure to React. Through the switch, I recreated my previous static website through the use of components. I could break up my pages to reuse elements, and I was successful in dynamically rendering different detailed product pages through the user the same page component. I used React Router to assist in switching between pages and provide a natural web experience. By changing to React and creating functional elements, I had moments where I was stuck understanding nested links in react routine, passing parameters and best ways to use stateful components, and sorting through local storage information to render on the page. Through this process, I understood React and the connection between components and tips that I will utilize on my next React project.

Reflection

Approaching this project, I knew I wanted to challenge myself and transfer my static HTML /CSS files and create my website in React. Through the process of switching, I encountered problems related to React Routing, managing stateful components, deleting from local storage, and GitHub Pages.

Before this class, I had experience with React Router within the context of only having the navigation. However, I have never created buttons or links within the page that required Routing as well. I got stuck in conceptualizing how <Links> from within components would connect to the Router, but through the exploration of other GitHub projects that utilizes React Router, I was able to understand that as long at the path exits within the < Switch> of the Router, all paths would be able to be accessed. By understanding how paths can be access through the < Router>, I can easily incorporate what I learned into my next project. Through the routing process, I also discovered its complexity with GitHub Pages when using React Router. Due to the nature of the set routes of the paths within React Router, the compatibility between GitHub Pages and React Router gets a bit confusing due to the initial natural pathing of GitHub Pages. GitHub pages URLs always include the name of the repository that it is accessing, but that also means setting the initial URL pathing to have the repository's start path. Additionally, creating the build folder of the project needed to be

adjusted to become a “docs” file. While I ended up hosting my site on Netlify, as it provided an easier way to hosting my site with a little adjustment to my build code, I now know how I would be able to host my react project on GitHub Pages.

By coding my website in React, I kept running into figuring out if the component I am creating should be a stateful one. By developing the feature of having added cart items update the number of cart items within the navigation, I kept thinking through the ways I would get the quantity from local storage to update the navigation. Initially, I was placing the navigation within each page component, but I quickly realized that it caused me to copy the same code onto every page component. I decided to move the navigation bar component to the App.js file. In moving this component to the App.js file, I made the App component stateful and created states that would track any changes within the quantity of the local storage. My journey into finding the solution now allows me to think about components from a global level and build my site top-down by considering how components are affected from a full website level down to a page level.

One major issue that I worked through was using the same page component but rendered by different URL paths. The initial render would be correct; however, the moment I navigated to another product page, that page's contents would not update. I explored potentially splitting up the pages into different components and different ways to nest my different paths. Through many hours of googling, reading through the documentation, and exploring different ways other have approached similar problems, I was able to use the URL parameters to update the state of the page so that when the parameter is updated, the states of that page would also update, ultimately updating the contents of the Detail page. Moving forward, I now have a better understanding of how I can utilize the same component for differently pathed pages.

Lastly, I ran into an issue when I was creating the functionality to delete an item from my cart. I considered the situation where a consumer adds the same product into the cart at different times. I wanted to make sure that my cart would reflect the number of unique items as one line item and not repeat individual items. I initially created a way to re-organize the products within the DOM; however, that was not reflected in the local storage. I had to figure out a way to delete items from the local storage that was not organized the way it was presented on the display. My solution was to re-update that cart in local storage once a cart was

restructured. From there, this made deleting items from a cart easier by only needed to capture the pillow type, color, and fill.

Programming Concepts

By creating a functional detail page, I gained more experience with creating stateful components, getting and setting local storage, passing props to children components, creating class-based components, and creating functional components.

When creating the Detail page component, I broke down the different areas of the page to better understand which components would be functional components and which ones would be class components. I created 4 different components:

- The sub-navigation (orange buttons across the top to help navigate between products).
- The photos component (to hold all the product images).
- The product details (that reflects all the details of the product and selection options).
- High-level detail banner (that sits below the fold of the page and provides high-level details about the product).

When creating these components, I understood what I needed from the parent component and if the components needed to be class components or functional components.

The sub-navigation I was able to create as a functional component. This is because the rendering of the component was straight forward and simple that it didn't require the passing of props. For the other components, I created class-based components. This is because, my image sources had to be passed back to the styling sheets rather than just to an image tag. Since my images were not cropped to the correct dimensions, I created divs that I could style and placed the image URLs as the background property. I had to pass the image source, image size, and image position from the parent to the child component for image elements. From there, I created object variables that I then used as the styling of the different divs. Reflecting on it now, I do not think I needed to create classes as many of these components were not stateful. Still, I found it easier for me to monitor how I was styling and getting different props from the parent component.

My Detail Page component was a stateful one. It required me to create a constructor function to determine the initial state of the page. I was able to decide on the URL

parameter and make a switch case that would update the page's state. I also created functions that would update the state of the selected color, fill, and product quantity. This allowed me to quickly and easily create a reusable function that could save cart and favorited items into the local storage. Since the button functions within my children components required grabbing certain states, I created all my functions on my Detail Page Components that I passed into Product Details components so that the buttons in the event within the child component could trigger the function. I found that it was easier to create a child component first to know what props I would need to pass. From there, I would import that component into the Parent component and make sure that I am grabbing all the necessary elements for that child component.