Assignment 6B

Connie Ye constany@andrew.cmu.edu

→ Link to React code on Github

→ Link to live prototype

Note: My main github account, khanniie.github.io, is set up with react router for my portfolio site, which somehow broke the gh-pages functionality (meaning anything that has khanniie.github.io as the domain, even gh-page links, will route to my portfolio site automatically), so while the main code will be hosted using my khanniie account, I'm deploying the site on a different github account called khanniie-sites instead.

Reflection

The main challenge that I encountered was using React Router for the paths that were dynamically created instead of hardcoded. I encoded all of the product data for all 6 products into a json format so that one component (ProductPage.js) could be used to generate a detail page for any of the 6 products. Using this, it was also easier to make the product browsing page and the sale browsing page because there was only one data source. However, this meant that routes like "/cat-backpack" weren't hardcoded but rather pattern-matched from the "/" route and then sent to the ProductPage component. This meant that for any routes that weren't hardcoded or "/," I had to check the data for an element with a matching url, and then I would render the component accordingly. I suspect that there are better ways to do this, but all of them broke the application when I tried it.

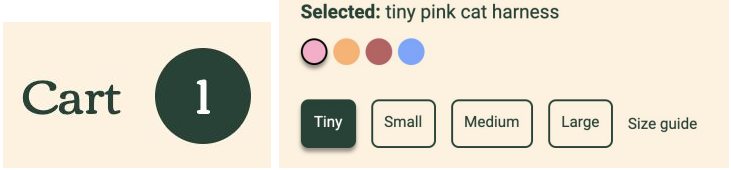
Another challenge that I encountered was enabling a link from the image carousel to link to a different product. Because it routed to the same component, ProductPage, the application didn't rerender the component when I clicked on a suggested product, so the page didn't update. To solve this, I put the element object into the state itself and then both routed the url to the new url and then also updated the element in the component's state.

Programming Concepts

1. **CSS transitions**
    a. I implemented a subtle hover state on the buttons using CSS transitions. I used the pseudo-class :hover to specify that buttons with a hover state on them should turn paler. I also specified a transition-duration so that it would fade over time instead of happening immediately.

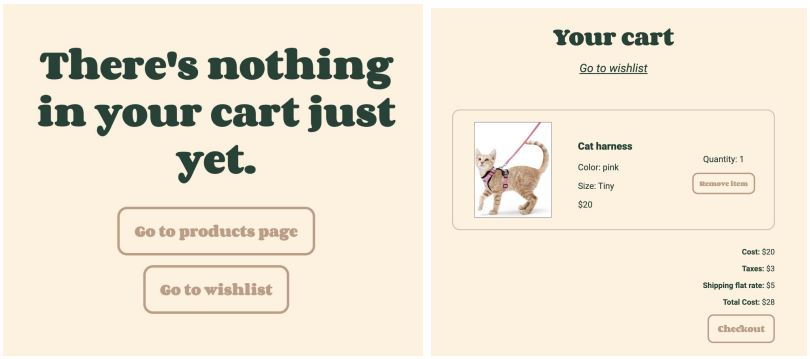
b. 

2. **Dynamic SVGs**
    a. I realized that if nothing happens to the button when you click on the “Add to cart” or “Add to wishlist” button, it’s hard to tell if it worked or not. Thus, I decided to add some fun confirmation feedback. I searched online for how to make a checkmark animation, and found a helpful answer that used an SVG html tag and classes to animate the properties of the svg tag’s children. I based my implementation off of that Stackoverflow answer and modified parts of it to fit what I wanted. I learned a lot about how to animate SVGs from playing around with the CSS animation properties and the styling on the SVG itself.
    b. 

3. **Stateful components**
    a. Parts of my UI were rendered based on the component state. One example of this was the cart indicator that showed the length of the array of cart items in the application state. Another example of this was the item color/size selectors. I stored the selected size and colors in my component state and used that to choose which preview image to show and which buttons to use an activated style for.
    b. 

4. **Event handlers in React**
    a. I implemented event handlers on some of the buttons that would change parts of the application state. One particularly tricky part of this was that the setState() function had to be wrapped into an anonymous function. Otherwise, the function would be called immediately and the returned value would be set into the onClick function rather than the actual function that you wanted to call.
    b. 

5. **Conditional rendering**
    a. For parts of my UI, I would return different JSX depending on the application state. For example, in the pictures below, I checked if there were any objects in the cart yet, and if not, I would display an “empty cart” message with helpful buttons instead of showing the cart at all. I also used this concept for the wishlist page.

b. 

6. Conditional styles
    a. In some cases, I styled the same part of the UI differently based on the product data. Specifically, some products had “cost” and “sale” values that were the same, indicating that they weren’t on sale - so I would style them like the picture on the left. However, if they were on sale, I would add the sale price in red and cross out the old price.
    b. 

Bonus

I implemented both of the extra credit functions.

Carousel

I implemented the image carousel for similar products by encoding similar products into the data json that I had. I would then grab the data for those similar objects and display them as an image carousel **at the bottom of all of the product pages**. The images are clickable and will take you to their respective products. The currently selected carousel object is stored in the component’s state.

Wishlist

I based my wishlist off of my cart component, but made modifications so that wishlist objects could be either removed from the wishlist or added to the cart. On the product detail page, I added an “Add to wishlist” button. Since I had a busy navbar already, I decided not to link the wishlist in the navbar, but as a link on the cart page.