Lab Week 6 - Web Components, Local Storage, and Fetch

Submit Assignment

Due Friday by 11:59pm **Points** 0 **Submitting** a website url
Available Feb 9 at 3pm - Feb 13 at 11:59pm 4 days

It is important that you read through this lab carefully

The previous lab had you learn how JavaScript interacts with the DOM and some of its APIs. This week, we'll be adding on top of what you learned by fetching data from a 3rd party vendor, plugging it into custom web components that you will create yourself, and then inserting these custom elements into the DOM when the page loads. This might sound a bit daunting initially, but it's a lot more straightforward than you might expect. We'll also be taking a look at localStorage so that we only need to fetch the data once (very handy for those who are performance minded) as well as remembering your previous state.

Those of you familiar with Object Oriented style programming might be pleased to find out that custom elements is very similar using JavaScript classes. If you're unfamiliar or need to brush up, we'll link some resources below.

When creating custom components in this lab you might come across some new terms that sound a bit odd or confusing, but I promise they only sound that way. One of those terms is the **Shadow DOM**. What is the Shadow DOM? It is helpful to think of it as a scoped subtree of the regular DOM. Scoped here meaning that it is encapsulated from the rest of the DOM. Each component you create will have its own Shadow DOM all to itself. It's handy in this sense as you can add as much CSS & JS to a component as you please without having to worry that that style or code will bleed into the regular DOM. It also follows then that since the regular DOM is a tree and has a root element (<html>) that the Shadow DOM has a root as well referred to as the Shadow Root.

Resources

We will only be linking to more academically minded resources here as they are more likely to have accurate information, but if you find the concept of custom elements a bit confusing there are tons of other resources and videos you're free to look up and use.

- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
- https://developer.mozilla.org/en-US/docs/Web/Web_Components
- https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements
- https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

Set Up

- Fork https://github.com/CSE110-W2021/Lab6_CSE110Shop to get the starter code

Instructions

Like last time - DO NOT MODIFY ANY FILES OUTSIDE OF **script.js** AND **product-item.js**

Taking a look at the starter code you're provided, we've already done a lot of the heavy lifting with completed HTML & CSS. Looking at the HTML page in your browser initially is a bit sparse, but if you browse over to the code for that page, you can see a list item underneath a "Sample Product" comment.

That list item, which we'll refer to as a card, is the custom component you are going to be implementing. The card has four parts: And image, a title, a price, and an "Add to Cart" button. Adding more list items (which you can see by duplicating the starter one) you can see that they automatically start populating the page and wrapping around. Go ahead and delete any duplicates you made and comment back out that initial item.

What we'll be doing in this part: First, we will fetch a JSON file that contains many unique items, each with all of the information you need to plug in. Then, we'll create the custom component for the card. Finally, we'll create a custom component for each of the unique items from our JSON file and append it to the page.

1. We will start by fetching data from an endpoint to get product items. Inside your **script.js** file, you'll see a DOMContentLoaded event listener. This fires as soon as the webpage is ready. Inside here, create a fetch request to the following URL: https://fakestoreapi.com/products - This URL will return an array of JavaScript objects.
   - If you have never used JavaScript's Fetch API before, you can read about it here (Be careful - fetch is asynchronous)
   - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
   - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

2. Once you have fetched this array, add it to local storage. You'll want to go back and add a check before your initial fetch request now to see if it is in local storage first. Local storage will only store strings, so **watch out**.
   - If you've never used local storage before, you can read about it here
   - https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

3. This will likely be the trickiest part. You will be creating the custom component **<product-item>** for the card. We've given you a small start in **product-item.js**, but make sure to read up on web components using the resources below. Thankfully, as far as the HTML goes we've already given you a sample **<li>** that you can just copy and modify. Make sure to change the **img src**, **img alt**, and the inner text of both **p** elements. Since you will also need the styles for the card since the page styles won't work in the Shadow DOM, if you look in the **styles.css** file we've marked out the start and end of all the CSS you will need to include.
   - https://developer.mozilla.org/en-US/docs/Web/Web_Components
   - https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements
   - https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

4. Once you have a working custom component, upon page load back in **script.js** you will need to create a **<product-item>** element for each of the items you previously fetched and append them to the **<product-list-container>** element.
   - Note that when you are first fetching all of the data from the URL, it can take a few seconds for it to populate your page

5. Next, we will be implementing a cart item counting functionality. When an **Add to Cart** button is clicked, add 1 to the total count of cart items (displayed on the top right) and change that button to now say **Remove from Cart**. When a **Remove from Cart** button is clicked, subtract 1 to the total count of cart items and reset the button back to **Add to Cart**. The cart-count span element should change according to the number of items in the cart.

6. Finally, we want to make sure our cart is remembered when we refresh the page. When you add an item to your cart, make note of which items were added (this could be implemented many ways - a suggestion might be to make a list of items that were added in your cart and store this in localStorage. Hint: Each product should have a unique ID in the array that your fetched. When you render the Add to Cart button, you can check if the id of the current item that you're rendering is in the list of item ids that are in the cart).

Canvas Submission:

- Link to your repository
  - Please publish your site through Github Pages and include the link of the published site in a README.md file.

FAQ:

- **Q: I'm having some trouble using fetch**
  - A: It is asynchronous, so you have to make sure that you are waiting for the request to be finished before trying to access the data

- **Q: I'm having some trouble using local storage**
  - A: It only handles strings, so make sure you aren't trying to directly store a JSON object. There are ways to easily convert between the two if you look around a bit.