Reflection

Supawat Vitoorapakorn

Reflections and Programming Concepts

Reflection

Issue 1: Improperly Using ID Chaining as Selectors
I seeked to replicate class with IDs chaining. Fo example, instead of using `.button` inside a nested `div`, I would be using multiple duplicated `#divA_divB_button` instead. This happened because it works well up until a certain complexity of the project. I did this for most of this project until it failed to work for my cards in my shopping cart.

How I resolved it
At the very end my chain ID method failed me when I tried to create my shopping cart. Because each card required a dynamically generated unique ID to bind an event handler, I was forced to not be reliant on chaining my IDs to style them properly. This is when I discovered the power of class.

How to mitigate it in the future
In a markup language, if a tag or attribute is created and it has remained over time, then it's probably there for a good reason. I initially thought that IDs and classes were just different syntaxes of doing the same thing, but its because I didn't fully grasp their difference that I made this mistake. In the future, I will try to be more deliberate in how I tag and attribute my elements/classes.

Issue 2: Hardcoding HTML vs Dynamic Generation via JS Objects
I didn't have the foresight and expereince of dynamic vs static content, I did not dynamically generate my HTML for all my products. It was only when I finished the whole project that I realized there was much more elegant route. I created a mock up for the experience by 2 dynamic product page instead.

How I resolved it
While I didn't dynamically generate HTML content for each product, I prototype the experience with inelegantly more HTML than I needed.

How to mitigate it in the future
Understand which content is dynamically repeating the same information in different structures. In the future, I could have an array of all the possible items as objects and have a javascript that dynamically generates all my HTML based on that rather than hardcoding it. I wished my product page was driven by objects dynamically rather than hard code HTML.

This whole thing made realize and appreciate how much work full-stack dev is.

Programming Concepts

1) Loose Coupling
My webpage is loosely coupled into three major components of the MVC model. In addition to MVC's loose coupling, my javascript is even further loosely coupled. My website contains 3 files: cart_default.js, product.js, and main.js, and these 3 javascript files are only active when they need to be. Product.js for example is only activated when product.html when is loaded. By mapping and organizing the functionality of my website to specific javascript files, I reduce the risk of breaking my system as a whole.
2) Model View Controller
I once asked my CS friend what is the biggest thing they learnt from their internship, and separating front end from my backend was their biggest lesson. Now I understand why. My data (model) is explicitly separated from the way I view them in my code, for example:

```
function onLoad() {
// Initialize Model
```

```
local_cart = JSON.parse(localStorage.getItem("user_cart"));
if (local_cart == null) {
    localStorage.setItem("user_cart", JSON.stringify([]));
}
// View Model
View_update_num_cart();
view_items_in_cart();
update_remover();
};
```

After my model is initialized and manipulated, I simply create functions to view...() and update...() them.

3) Event Based Programming

I created my controllers to manipulates my models directly by using J-query and vanilla javascript event handlers. These listening events served as the C for my MVC. I bind these to various element in mind HTML dom. I also used animations from J-query to increase the website's experience.

4) Client Side Storage and Arrays

I used `localStorage` to dynamically store and retrieve data across multiple webpages. An array is used to store my buns. After initializing a `cart = []` array in `localStorage`, I pulled my `cart` down as a reference, manipulate it, and pushed new buns() back into `localStorage`.

5) Classes and IDs

The cards in my shopping cart really helped me consolidate my understanding of classes vs IDs. While each card has a unique ID to identify and bind them to .listeningEvents in javascript, they are also part of a class that gets displayed all the same way in CSS.