Assignment

I have contributed JavaScript to a site that gets a collection of game cards from an API and displays them on a front page. Each card has a button to view details, which makes a request for a specific card form a different endpoint in the same API.

The site has another page with game rules and a contact form. In these I have replaced a given word with "something", and made a dropdown text container toggle based on a mouse event. And lastly the contact form has a validator for the four input fields upon submission.

Introduction

To begin with I skimmed the assignment to get a quick overview. My first mistake was starting coding without a through second reading. It was not before i was done with the first page that i discovered that I had missed the provided files, and that a days work on wrestling with flexbox and file structuring for a whole new site were potentially wasted. Luckily most of the code I had written was salvagable, and after reading through the assignment properly, I set out to do what i was supposed to.

The Index page

To start out I was unsure of how I should approach the API call. I initially did a XMLHttpRequest, but after getting "Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user’s experience" in the console, I investigated other approaches.

I landed on the fetch() function as it is sleek and asynchronous with Promises and all the new bells and whistles.

At the top environment variables are declared. References to HTMLElements and API endpoint. I also initiate the fetch here immediately when the script is loaded.

I set up the handling of the outcomes from the fetch inside it, but in separate functions. I could have squeezed everything into the .then(), but it felt cramped and unnecessary.

processSuccess() is the meat of the script, so to speak. It creates the objects I use, and calls the methods. I also instantiate a class declared later to contain the card array and methods.

One of these methods print all cards in the Library, and the other prints cards that match a name criteria. This latter method feels a bit "janky" as the .includes() string method is case sensitive. I made everything uppercase inside the logic, even though I could have used Regular Expressions for something more "professional".

The constructCard function is called whenever one of the previous methods needs to produce HTML, by means of a string literal which is appended to a running total. This is to keep my code DRY and neat.

I have taken the liberty of using a cardback instead of the given placeholder image. I think it looks better and fits thematically. If the placeholder would be wanted by any customer, replacing the source of the image would be easy.

The Specific card page

Here I copied over the basic structure of the fetch, but I handle the outcomes a bit differently.

As I don't need to store any card, the printing is handled immediately when the fetch is resolved. In the constructDetails I handle the case of a card having more than one color, or if it is colorless with a simple for-loop and a ternary operator - A really handy operator I have fallen in love with for simple inline logic.

The three functions that produce the HTML-strings feels a bit repetitive to me, and if this project were to be expanded any further I think I would try to compact them all into a single template.

The About page

The About page was a (to me) simple task, so I thought I'd spice it up with an IIFE (Immediately Invoked Function Expression) for the substring replacements. It was only supposed to run once, and no part of it was up for reuse later. I tried it compacted down to one line, but it got ugly, so I kept it at as two.

The toggle display for the drawer is achieved using the getComputedStyle that gets the CSS rules affecting an element. By checking if the display is "none", I can set it to "block", and if else it's set to "none". Another common way to do this is to change the classes of the element, but that involved checking for duplicates, appending and/or replacing, so I thought this was the most elegant solution.

The Contact page

On the Contact page I may have gone a bit overboard on the declaring of references, but in doing so I could make a neat and reusable evaluation function later down the line.

On final review I see that I could have gone the way of arrays here, but as this is a form with only four fields I would probably not save much space, and just make my code more cryptic.

Conclusion

All in all I think I solved this assignment in a good way. I have some inconsistencies in how i approach what is essentially the same problem throughout - as for example using RegExp sometimes, and .includes() at other times, or IIFE somewhere, and not elsewhere.

During the development of the code I made use of Postman to validate the URLs I was using and the structure of the responses. It was mostly out of curiosity and because I already had it installed.