REPORT

CA MODULE 4

JAVASCRIPT

Introduction:

In this report I will try to paint a clear picture of my thought process throughout this assignment. I will also describe some of my functions as to clarify how I picture how they work.

Body:

Monday morning I read the assignment and made mental notes on how to tackle the task at hand.
I downloaded the files provided in the course assignment(CA), open up the index.html and script.js files; And read the assignment again.
This time i came halfway down the page before I made the decision to not read more than one “line” at the time.
I figured that it would make it easier to focus on getting one element of the code working; instead on fixating on the final result.

We had already done an assignment that required us to connect to an API.
So I could have copy/pasted in some code, However I wanted to understand how it worked.
So i check out MDN Web Docs[1] for more information.

```
1  // refer to question 1 before development starts for scope document
2  // connect to this api https://api.magicthegathering.io/v1/cards
3
4  fetch("https://api.magicthegathering.io/v1/cards").then(function (result) {
5      return result.json();
6  }).then(function (json) {
7      displayResults(json);
8      searchButton(json);
9  });
```

In the picture above:
On line one there is a fetch statement. It takes a URL variable; and it is a javascript promise. the fetch is followed by a .then, indicating that whatever comes after the it has to do with what came before.
So the .then is now the fetched URL and The Anonymous function with a parameter “result” that follow can be described as follows:

function(result) - result = value/content of fetched URL

The function use the return statement to return result.json();

.json() is used to get the json object from the result.
We need to do this in order to use it’s values in javascript.

The function is followed by yet another .then and another anonymous function.
This function has a “new” parameter (json).
The new parameter is not new at all. It is the result of the previous function. Only this time it is called json, because it makes it easier to see what is going on (remember that the result of the first function was converted to json in the return statement).
This function is used to call and pass along the json object to new functions outside of the fetch promise.

before I declared my “displayResult” function I tested to see if I successfully connected to the API by writing console.log(json); at the end of the fetch statement.

```
▼Object
  ▶cards: (100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…
  ▶__proto__: Object
```

It worked great, And I could continue with the code for the index.html file.

Getting the cards to show up was done in no time, I declared my function and made sure it was named the same as in the fetch statement.

```
fetch("https://api.magicthegathering.i
    return result.json();
}).then(function (json) {
    console.log(json);
    displayResults(json);
    searchButton(json);
});

function displayResults (magic) {
```

Then I decided I needed a for loop to get all the cards from the json object.
Then I wrote variables that used the dom to create elements.

```
// Create tags for text
var myh4 = document.createElement("h4"),
    image = document.createElement("img"),
    link = document.createElement("a");
```

I made the same elements, and gave them the same id as the template card used.
I continued to give each element a value matching the value of the template card.

```
// add value/text
myh4.textContent = magic.cards[i].name;
image.setAttribute("src", magic.cards[i].imageUrl);
link.setAttribute("href", "card-specific.html?id=" + magic.cards[i].id);
link.textContent = "View More"
image.style.width = "100%";
```

Then I appended each element to the container div, and the container div to the column div. and the cards were displayed on screen.

```
// Append to div
cardContainer.appendChild(myh4);
cardContainer.appendChild(image);
cardContainer.appendChild(link);
col_sm_4.appendChild(cardContainer);
cardsId.appendChild(col_sm_4);
```

Next task was the search bar.
The search bar however took forever to get working the right way.
I declared a function named search Button and passed in the same json parameter.
Adding the click event to the search button and deleting the all the cards was easy.

```
var allCards = document.getElementById("cards");
allCards.remove();
```

Until I realized that I did not want to remove that html element. Only wanted to remove everything inside it.
And if I used the my first approach I would have to do it 100 times to remove all the cards.
I did not want to do that. It would take a long time and it would not be effective and it certainly would not be the way you expect it to be done.

I did a quick google search and found this great example on stackoverflow.com[2]

```
var allCards = document.getElementById("cards");

while (allCards.firstChild) {
    allCards.removeChild(allCards.firstChild);
}
```

The example works as follows:
By using a while loop, I could check for firstChilds in allCards.
and as long as allCards had a firstChild. The firstChild would get removed.

I spent monday evening and the entire tuesday trying to get the search result to work. I figured I should use the filter method. But I just could not get it to work.
Finally I reached out to MJ Phillips to ask for a hint. He asked I had looked at the filter method.
I had looked at it for two days. but I decided to take another look.
I wrote my code again, and it worked. Somewhere in my old code was a typo that did not show up as an error. Or I was too frustrated to acknowledge it.

Now all I had to do was to return the filter result to a new array.
I did this by using the push method.

```
// Make new Array
var testArray = [];
testArray.push(test);
```

I used the array with and if statement to either give an error or show the matching cards on the webpage.
The error would show if the length of the array was lower than 1.
and for anything else the cards would be implemented like they did in the “displayResult“ function.

The search function is missing some finesse. I would like the search to ignore uppercase, lowercase numbers and special characters in the search. as it stand now “white” will not result in any cards; But “White” will display all cards with a color value of “White”.

Moving on to the card-specific.html and specific.js files.
This seemed like a fairly easy task. I fetched the API again. but this time I used the querystring of the clicked object, in the url. this was easily done with the included “getQueryStringValue” function.

```
// variable for the id
var id = getQueryStringValue("id");

// fetch API
fetch("https://api.magicthegathering.io/v1/cards/" + id).then(function (result) {
    return result.json();
}).then(function (json) {
    checkForQuery(json);
    displayCard(json);

});
```

The displayCard function works very much like the displayResult function in the script.js file. It use the API json object and grabs the values it need and append them to elements in the DOM.

I had some issue with checking for a querystring. I have a bad feeling that my decision is incorrect.

```
function checkForQuery(magic){
    // Get URL
    var url = new URL(window.location.href);
    // get everyting after "?id=" in URL
    var query = url.searchParams.get("id");

    // All cards have ID-Length = 36
    if(query.length < 36){
        var container = document.getElementById("cardDetails");
        var missingQuery = document.createElement("h2");
        missingQuery.innerHTML = "Missing Query String";
        container.appendChild(missingQuery);
    }
```

I decided that I needed to get the whole url. And a search on the good old mozilla MDN web docs was needed for this task.
Then I grabbed the search parameters[3] of that url. The search parameters is the same as the card Id. and all the cards have the same length of there ids.
So I said that if the query was shorter than 36, the query string was missing.

When I got to the About section of the CA I felt relieved.
I felt like I had a plan for this right away.
I grabbed the HTML element and played around with how to get the value of it.
It turned out I need to use "innerHTML" without any additional code.

```
// Replace every instance of "Magic" with "Something"
var aboutMain = document.getElementById("aboutText");
var text = aboutMain.innerHTML;
```

Now that I had the content I could start to manipulate it.
We had done something similar in on of the Lesson Assignments(LA).
I knew that i had to use the replace method. After a quick refresher the function took shape and the word "magic" was replaced with "something".

In the previous Lesson Assignment I had spent unnecessary amounts of time trying to hide unwanted to do cards with a click event.
I used the same function for this task.

```
var trigger = document.getElementById("moreInfoTrigger");
trigger.addEventListener('click', function(){
    var moreInfo = document.getElementById("moreInfoContent");
    if (moreInfo.style.display === "none") {
        moreInfo.style.display = "inline-block";
    } else {
        moreInfo.style.display = "none";
    }

})
```

It is an click event. when the event occurs the function executes.
Inside the function is an IF statement. It checks the display value of the hidden/shown div “moreInfoContent”.
if the style.display === “none”, then it is changed to “inline-block” making it visible.
if however style.display === anything other than “none”, it is changed to “none”, making it hidden.

Finally I had to start work on the contact.html and contact.js files.
I was not looking forward to this, as I knew it would require regular expressions(RegEx).
RegEx is hard to read, and understand.

I found a site to test RegEx, and used it when designs my expressions.
This made it easy to see if the RegEx was valid or not before testing, and it could easily be changed and tested over and over again.

The input fields for FirstName and LastName was the easiest to do.
I made a RegEx pattern that looks like this.

```
var namePattern = /^[a-zA-Z-]{1,30}$/;
```

It matches if the value of first or last Name equal to 2 or more characters. with a maximum of 20 characters. the “-” is also allowed.

The next pattern to tackle was the phone pattern.
The one I made looks like this.

```
var phonePattern = /^[\d]{3}[\-|\.|\ ][\d]{3}[\-|\.|\ ][\d]{4}$/;
```

This one was easier than I thought it would be.
I used \d to indicate any digit. and {3} and {4} to indicate how many digits that is required.
in between each “[\d]{3}” you will find “[\-|\.|\ ]” this is added to make the phone number use spaces, periods or white spaces between each group of numbers. Only one of them can be used between each number group.

The last and hardest was the email pattern.
It looks like this.

```
var emailPattern = /^[a-zA-Z0-9-._]+@+[a-zA-Z0-9-.]+[\.]+[a-zA-Z0-9]{2,6}$/;
```

I opted for a simple approach to this pattern. I did some snooping around and found out that a perfect pattern for emails does not exist. More on that later.
The first part “[a-zA-Z0-9-._]”, allows any alphabetic character, digit, period, underscore or hyphen.
the second part is the requirement of an at ”@” symbol.
followed by the third part witch is identical to the first part.
The fourth part require a period.
and finally the fifth part allows 2-6 characters and or digits.

Now for what to do about the elusive perfect email Regex pattern.
For a contact form like this, I would only require the email input field to contain a at”@” symbol. To make sure that the email is valid or at least make it more likely. I would ask the user to type it again in a new input field.
Then I would compare the two; If they matched, everything would be in order and the form would be submitted. Else the email had to be retyped.

For account registration I would do the same thing, only difference being that a confirmation email would be sent out. and without clicking on a link in that email the account would not be created.

Conclusion:

The code works, so I believe I have done a good job on this CA.
I feel like have been tested on all previous lessons in this CA. Some things I had to read up on. But to my suprise, I had learned a lot, and much of the code was written without additional research. I really enjoyed the fact that I had learned a lot during these weeks and are looking forward to future JavaScript projects.

References:

1. Mozilla MDN web docs - Fetch method
   https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

2. Remove first child of html div
   https://stackoverflow.com/questions/3955229/remove-all-child-elements-of-a-dom-node-in-javascript

3. Mozilla MDN web docs - searchParams
   https://developer.mozilla.org/en-US/docs/Web/API/URL/searchParams