JavaScript 1 Course Assignment Report

Nancy Y. Bolstad

Introduction

This report reflects on how different JavaScript techniques have been implemented during the process of building a simple web application. It begins with exploring two different technologies that can asynchronously retrieve data from a remote server by making an API call: namely, the XMLHttpRequest(XHR) and the Fetch API. It will then present my solution for making an API call reusable when executing multiple Ajax tasks. Further, it shows how I made use of the Document Object Model (DOM), the Browser Object Model (BOM) and JavaScript to control the display of the JSON object as well as to manipulate content on the webpage. Last but not least, it explains how I used regular expressions to validate user input and replace content on the webpage. It will also briefly describe the testing and debugging process.

A short note on using the API

I would like to briefly explain how I have been using the API. I first installed the JSON Viewer extension in my Chrome browser, which converts the JSON file into a more readable format with elements being highlighted. Then, I checked the API document on the official website at *magicthegathering.io*. The document has clearly listed all the properties as well as solutions for how to deal with errors and get specific information from the JSON file. The JavaScript SDK for the API is also available on Magic's Github documentation page, which can help developers to perform tasks such as finding a card's name. I could have installed the MTG API for this project, but in order to build and show my own JavaScript skills, I developed original approaches for retrieving data and filtering through cards, which will be presented in the next section.

Making asynchronous API call via XMLHttpRequest in a callback style

Both XHR and the Fetch API can make asynchronous requests across domains and create fast interactivity without having to let the user wait for the entire page to be loaded. However, they do so in completely different ways (MDN Web Docs, 2019a; Quigley, 2010). The Fetch API makes use of chained promises: after making an API call, it will return a promise which can be used to perform a sequence of asynchronous tasks via the help of the *then()* method. XHR, on the other hand, makes an HTTP request via the following steps: create an XMLHttpRequest object; =>open the URL;=>send the request (Rascia, 2017; Haverbeke, 2015).

In this project, I decided to use the XHR techniques, because it has a broader browser support than the Fetch API. (MDN Web Docs, 2019b) To insure fast interactivity on the webpage, I made the request asynchronous by assigning the third argument in the *open()* method as "true". This will make the loading happen in segments in the background instead of the whole page having to be loaded at once. Additionally, to deal with the different responses (status codes), I used a conditional statement to control the flow of the program. If the response is successful, the program will execute a function that takes the returned JSON data as an argument. If any error arises, the program will display a notification to the user.

Further, because the assignments requires us to make two API calls – one call for generating cards on the page, the other for searching for a specific card name and then display the results – and because it has been recommended that the XMLHttpRequest object should be recreated each time when making a new request to the API server (MicroHOWTO, 2018), I chose to follow w3school's (2019) suggestion to make use of a callback function instead of writing several blocks of repeating/duplicate code for each API call. This makes the code cleaner.

In my project, the callback function takes a function as a parameter and will execute different tasks each time after the API has been called (Haverbeke, 2015). To process the resulting data before displaying them as the actual cards on the web page, I created two functions. While both take the JSON data as the argument, the *generateCards()* function abstracts wanted data (card names, images, ids) from the object by using the *forEach()* array method and the *cardsFilter() function* first filters through the resulting data and then returns only the search result. I have also created a *showMessage()* which is reusable and has been used to display suitable information for users on the page.

Modifying the DOM and the BOM using JavaScript

JavaScript uses the DOM to access the HTM document on the webpage, and the BOM to control the behaviour of the browser window (Quigley, 2010). In my project, on the card specific page, I used the window object's location property to read and modify the URL and the URL's query string of the document being loaded in the window. I used the following properties and methods to modify the HTML document (W3Schools, 2019b; Haverbeke, 2015; Quigley, 2010):

- *the getElementById()* method to refer to a specific element in the HTML document;
- the *addEventListener()* method to handle mouse click events;
- the hasChildNodes() method to perform a boolean check to see if a node has any children;
- the *removeChild()* method to remove a child node;
- the *appendChild()* method and the *createElement()* method to create new child nodes;
- the *setAttribute()* method to set an attribute to an element in the HTML file;
- the *focus()* method to turn the focus to a specific input field on the form;
- the *innerHTML* property to modify content inside the HTML content;
- the *textContent* property to set content for each card's name and image as well as the *src* attribute of the view more button;
- the *style* property to modify the page's appearance on the CSS style sheet.

Using regular expressions

On the about page, I used a regular expression *("/Magic/g*") to identify a specific pattern and then used the *replace()* string method to search for the pattern and replace it a specified string. To make sure that the change would be made to all the instances of the pattern, I made use of the *g* flag to perform a global replacement.

On the contact page, I used regular expressions to ensure that the user input will be validated before the form is submitted. The variables *emailRex* and *phoneRex* are assigned with bracketed regular expression. I used the RFC2822 standard for the email's regular expression and applied metasymbols to represent digits in the regular expression for the phone number field. The regular expression *test()* method has been used to test input from the user.

Debugging and Testing

In order to control the quality of my JavaScript code, several tools and techniques have been used. First, all the submitted code follows *The Principle of the Good Parts* created by Douglas Crockford and have been validated using the JSlint(2019). Second, I used Chrome's developer tools to test the performance of the web application. Third, I installed Git Version

Control in Visual Studio Code (2019) which allows me to keep a timeline of all revisions and to go back as necessary. Fourth, I used RexExr (an online tool) for regular expression testing.

Conclusion

By completing this week's course assignment, I have applied several JavaScript techniques that are at the core of developing a dynamic and interactive web page. These techniques include: making API calls with JSON data, accessing or finding relevant information from arrays and objects, making use of the DOM to control the appearance on a web page, modifying or checking a page's URL and its query string with the window interfaces, handling mouse events, and using regular expressions to validate input from users prior to form submission as well as to execute global replacements. Making use of the benefits of several code testing and debugging tools, the final JavaScript code is error free, clean, easy to understand and maintainable.

Reference:

JSlint. (2019). *Help.* Retrieved 26 March, 2019, from http://www.jslint.com/help.html.

MDN Web Docs. (2019a). *Fetch API*. Retrieved 1 March, 2019, from https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest.

MDN Web Docs. (2019b). *XMLHttpRequest*. Retrieved 1 March, 2019, from https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest.

MicroHOWTO. (2018). *Make an AJAX request using XMLHttpRequest.* Retrieved 27 February, 2019, from http://www.microhowto.info/howto/make_an_ajax_request_using_xmlhttprequest.html.

Quigley, Ellie. (2010). *JavaScript by Example*. Boston: Pearson Education, Inc.

Haverbeke, Marijn. (2015). *Eloquent JavaScript*. San Francisco: No Starch Press, Inc.

Rascia, Tania. (2017). *How to Connect to an API with JavaScript*. Retrieved 26 February, 2019, from https://www.taniarascia.com/how-to-connect-to-an-api-with-javascript/.

Visual Studio Code. (2019). *Using Version Control in VS code*. Retrieved 1 March 2, 2019, from https://code.visualstudio.com/docs/editor/versioncontrol.

W3Schools. (2019a). *AJAX - Server Response*. Retrieved 28 February, 2019, from https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp.

W3Schools. (2019b). *JavaScript Tutorial*. Retrieved 28 February, 2019, from https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp.