

Diary Entry

Link to SCIS website: <http://student.athabascau.ca/~matsph/>

Work for this Unit:

Pre-coding Activities:

Skimmed the Unit 7 AJAX FAQ

Looked through APIs listed here: <https://www.programmableweb.com/category/all/apis>

Watched AJAX introductions and tutorials such as:

<https://youtu.be/3l13qGLTgNw>

<https://youtu.be/O8C86NHhFZg>

<https://youtu.be/iqNiINZ4Sxg>

Read introduction to AJAX:

http://www.w3schools.com/xml/ajax_intro.asp

Read introduction to JSON:

http://secretgeek.net/json_3mins

Step 1: External data use proposal

Enhancements:

1. Facebook 'share this' feature for each web page.
 - This would allow Janice, Sean, and Robert to bookmark the pages they want to look at later, for things such as further reading on a solving method, or searching through special patterns. It would also allow Janice and Sean to share pages such as puzzles like the Megaminx and solving methods.
2. YouTube video playlist loaded from YouTube API.
 - This would improve the front page video highlights of cubing, making it easier to update and add to.

Step 2: Learning

I tested some of the AJAX and JSON features in a scrap file before I implemented it for my site.

First I followed this tutorial:

<https://youtu.be/BJ0tyZg2zek>

And tried to copy what he did, except for the url I used:

http://gdata.youtube.com/feeds/api/standardfeeds/most_popular?v=2&alt=json

which was provided by Google, supposedly giving a list of the most popular videos on YouTube in JSON format.

I tried this, and it didn't load it. So I made my local JSON file, and tried to load it locally. That still didn't work. After some testing of different more simple code, I eventually realized I forgot to add the JQuery loading script in the header. After that, it worked for the local JSON file. I then replaced it with the YouTube API JSON file, and it didn't work yet.

This is where I hit a road block. All of the videos I could find on using the YouTube API were outdated; even some of the Google developer's website on using the YouTube API was outdated. I searched around to find a good tutorial, and I eventually found this:

<https://youtu.be/l2DkxzyZ4xk>

I followed his set-up of the JQuery '\$.get' function, including all the parameters he used (except the query), and set it up to execute on page ready. I also got my own API key from Google developers, and then uploaded the scrap file to my SCIS website location, and looked in the Firefox debugging console for the output of the object sent back by the YouTube API. It reported an error, saying that either the project hasn't been used before, or the project wasn't enabled. It also gave me a link to enable the project. I clicked on the link, activated the project, and it sent the data!

Now that I got that big road block out of the way, I started to play around with it. The first thing I wanted to do was instead of using '\$.get', I would use '\$.ajax', since I learned it's a 'good' method, and plus I thought by doing with AJAX, I might know what '\$.get' actually is, and know the actual inner-workings better.

I looked up the '.get()' and '.ajax()' methods on the JQuery site, and found out what the parameters were for each. I also looked up quite a few stackoverflow questions, and they were all a little helpful, but I think the most helpful one was this:

<http://stackoverflow.com/questions/1344303/jquery-ajax-vs-get-post>

It was then I saw how they were similar, and how they were different. This made it much easier to convert between the 2, and know what information was being sent and how I could send it for each.

I then tried to convert the call I made with get() into a call with ajax(), using this as a guide:

<http://api.jquery.com/jQuery.get/>

The portion of code that showed how it was a shorthand AJAX function solved all my problems, and I converted it to AJAX with ease. I uploaded it to the server, and it worked! It was a little strange to me

that it worked without me defining the `dataType` (which I heard was something you needed to do with AJAX), but then I read that AJAX will try to guess if it is not specified, so then it made sense.

The next thing I did was convert the API from a search results return to a playlist return. I looked at the API reference here:

<https://developers.google.com/youtube/v3/docs/>

I found the section for the 'search' resource of the API (the resource I used before), and I tried to find all of the parameters that were used in the video tutorial (e.g. `part`, `q`, `type`, `key`). I then found the 'list' method (accessed through GET) for 'search', and it listed all of the parameters for the method, including 'part', 'q', 'type', and 'key'. I then understood how to work with them, so I then looked at the parameters for the 'list' method of the 'playlist' resource.

After an hour or so, I realized that the 'playlist' resource doesn't give what videos it contains. I looked up on stackoverflow how to get the videos of a playlist from the YouTube API, and only found examples for PHP, Java, etc., but not for JavaScript. I then saw that the YouTube API had an additional resource called 'playlistItems'; I saw that it lists all of the videos of a playlist, so I used it instead. After a little trouble with specifying the ID of the playlist (I used 'id' instead of 'playlistId'), I got the API to send the list of videos from a test playlist. I then set up JavaScript to grab the IDs of each of the videos, make YouTube player boxes for each video, holding a link to each video with its video ID inserted. I then thought that was all I needed to learn for now, and started adding the creation of videos from a YouTube playlist to my website.

Step 3: Coding

2. YouTube video playlist loaded from YouTube API:

I planned to use this feature to replace the "Video Highlights of Cubing" with a list of videos loaded from a playlist I made on YouTube. This would allow me to easily add to or remove from the playlist on the website without ever changing the code.

First, I added a 'js' file named 'playlist.js' to be loaded into the index page. Then I copied the AJAX code from the test file into the 'playlist.js' file, changed the playlist to the custom Highlights of Cubing playlist I made, and then made a loop that created new YouTube video objects for each of the videos in the playlist, adding their ID to the url of each object. I also removed the YouTube video objects from the index.html.

1. Facebook 'share this' feature for each web page

I first looked up "Facebook share button API" on YouTube, and found no methods that used AJAX. I then searched "Facebook API" and "Facebook AJAX", and still got nothing useful to me. I then looked up "Facebook API" on Yahoo! Search, and found the Facebook developers site. I looked through their SDK,

and found a Share Dialog that could be loaded using JavaScript. I thought it would work for a good share button.

In order to make a share button, I first needed to make a section to place the button in. I decided to append the button to the end of each page, and in order to do that, I added a section of code in the 'header.js' file that appended a 'share' div to the end of the page, with a button with the id 'fbshare'.

I then tried to use their 'FB.ui' method, but it complained I didn't provide an 'app id', so I searched for a guide on using the Facebook SDK, and found this:

<https://developers.facebook.com/docs/javascript/quickstart>

A tough part was adding in all of the initialization into the beginning of the body tag in every page. I was able to do this by adding the initialization script into an html file called 'facebooksdkinit.html', and in the 'header.js' file, I prepended a div where I loaded the html file into.

An even tougher part about this was that in order to use the SDK, I needed an 'app id'; in order to get an application ID, I needed to register as a Facebook Developer; in order to register, I needed to get a confirmation code; and in order to get a confirmation code, I needed them to send it through a phone number, which had long delays, and plenty of errors. I even had to make a new Facebook account because they seemingly blocked my first one.

I first tried to use the 'FB.ui' command as the quickstart told me to without adding any external references, but JavaScript complained it didn't know what 'FB.ui' was. I then looked at a video tutorial of using the Facebook SDK to see if he added any script loading tags, and on one video I saw that he used this:

```
<script src="http://connect.facebook.net/en\_US/sdk.js"></script>
```

After adding it, it still didn't work, but it did load a pop-up window that complained that "The domain of this URL isn't included in the app's domain". I looked up this error on stackoverflow, and I found some answers to what was causing the problem on this question:

<http://stackoverflow.com/q/37063685>

According to the answers, I didn't have good settings for the app. I then went into the settings for the app, and changed the 'Site URL' box to my website URL. I then tried to test the share button by uploading my local website copy to my website location, and it worked!

I then removed the script loading tag I saw on the video tutorial, and the share button still worked on the online website. I then implemented the share button for the rest of the pages.

How I have met the Learning Outcomes:

be able to use JavaScript to access and use web services for dynamic content (AJAX, JSON, etc.)

In my website, I have used 2 APIs, the Google Developers API for YouTube data:

<https://developers.google.com/youtube/v3>

And the Facebook Graph API in tandem with the Facebook SDK:

<https://developers.facebook.com/docs/javascript/reference/v2.8>

With the YouTube API, I have retrieved a playlist and the video IDs of every video in it using JSON through the AJAX method.

For the Facebook API, I have created a share form that sends a Facebook post to be added to the timeline of the person who uses the share form.

Notes:

Went well:

The YouTube API usage through AJAX went the best. Once I understood how to use AJAX through JQuery to send/receive JSON objects, the YouTube API worked very smoothly with it, and I was able to customize it extensively.

Didn't go well:

The Facebook share button implementation. I was hoping for an easy API interface like there was for the YouTube App, but they had their API work in tandem with their SDK, which made it hard for me to implement it in a clean manner.

What was most difficult and why:

Getting the Facebook share button to work. I first had to set up an app ID (which was a big trouble on its own), then I had to load the initialization code at the beginning of every body of every page, and then I had to set up the share dialog pop-up for the button. What was difficult about all this is that I was trying to get it to work on a local copy, and I couldn't see what was really going wrong until I tried it on my online website. There were also different standards, different tutorials, and different methods that all showed different ways of implementing it, which got me confused on what I needed to implement.

If done again, would it be done differently and why:

Yes, I would have first tried to implement the Facebook SDK on the website before trying to get it to work offline, that way I could make sure that the reason it wasn't working was because I was offline, and could have skipped many steps of my debugging.

How did previous experience help/hinder completing the tasks:

Previous experience with JQuery was very helpful, as much of AJAX and the APIs worked through JQuery much more easily than they would without it. I was also able to make changes across all pages with ease, and grab and place data from the APIs using simple methods.

Most surprising thing learned:

Almost all popular websites have their own APIs, allowing data to be sent and received through a programming interface. This was much different than what I was expecting, which was getting and sending data through the DOM, accessing and editing forms, inputs, and retrieving element data.

Most useful thing learned:

Almost all data and forms that a website offers through the user-interface, can also be accessed using a website's API, if they have one. This could be very useful if I ever come across a situation in my job or in my life where I need to routinely retrieve data or make changes to a website. This way I won't need to use the website DOM visual interface every time, but I could instead use a program to automate the process for me.

Map to Course Outcomes

In the original site mock-up, the 'Home Page.png' description says that the home page should include videos to show the highlights of the site. This was partly accomplished in Unit 2 with the links to the highlight videos, and further implemented in Unit 5 with the YouTube object boxes that allow you to watch the video from within the page. Now, in this unit, it is even further implemented and made to be more easily managed by loading the videos from a custom playlist on YouTube, allowing new video highlights to be added and removed according to what I find to be the best video highlights for the site.

Also, the original plan for the site was for it to be an "information resource", allowing anyone interested in cubing all the information they need, and allow them to "keep coming back" for more information. This is partly assisted through sharing on Facebook, as it not only allows the information resource to be accessed by more people who are interested, but it also allows pages to be bookmarked, making it easier for people to "keep coming back".

Self-assessment:

Learning Outcome	Evidence of Meeting the Learning Outcome	Your Own Assessment of the Grade You Believe Would Be Appropriate	Tutor's Justification of Grading (optional)
Apply a structured approach to identifying needs, interests, and functionality of a website.	I have made 7 scenarios that give structured processes of how the anticipated audience will address their needs with the website. I also took into account the constraints users might have, such as different operating systems, slow computers, and browsing the site using mobile devices.	B	
Design dynamic websites that meet specified needs and interests.	The JavaScript code in the 'floater.js' and 'sorter.js' files give the website dynamic content, which meets the needs and interests of the personas as described in 'Other Requirements' in the diary entries of Unit 4 and Unit 5.	A	

Write well-structured, easily maintained, standards-compliant, accessible HTML code.	<p>Sample1.html and Sample2.html are edited to show my ability to write standards-compliant and structured HTML. The errors that used to exist in the code are listed in the learning diary under the section “Other Requirements”.</p> <p>The 11 pages I wrote for my website all have links easily clickable on a mobile device, and are all well-structured and standards-compliant.</p>	B	
Write well-structured, easily maintained, standards-compliant CSS code to present HTML pages in different ways.	<p>The CSS I wrote is either in the external CSS file, or in a one-time internal CSS in the ‘other_puzzles.html’ page. In both, the code is organized with comments and well-structured using proper spacing, character case, property ordering, and naming schemes.</p> <p>It is easily maintained as the external CSS file is less than 130 lines long, and the internal CSS only defines styling for one selector.</p> <p>It is standards compliant with WCAG 2.0 by having no obscure or hard to read text, and proper contrast.</p>	A	
Use JavaScript to add dynamic content to pages.	I have added the JavaScript file ‘floater.js’ which dynamically changes the positioning of the navigation field depending on where the user’s viewport is.	B	
Critique JavaScript code written by others, identifying examples of both good and bad practice.	I modified the code written by LearnWebCode as described in the ‘How I have met the Learning Outcomes’ section of the diary entry for Unit 4.	A	
Select appropriate HTML, CSS, and JavaScript code from public repositories of open source and free scripts that improves your site and that enhances the experience of site visitors.	I selected JavaScript code from LearnWebCode as described in the ‘How I have met the Learning Outcomes’ section of the diary entry for Unit 4, which meets the needs and interests of the personas as described in ‘Other Requirements’ in the diary entry for Unit 4.	C	

Modify existing HTML, CSS, and JavaScript code to extend and alter its functionality, and to correct errors and cases of poor practice.	I edited the templates given in Unit 2 as described in the Unit 2 learning diary under the first requirement under the section 'Other Requirements'. I also modified the code written by LearnWebCode as described in the 'How I have met the Learning Outcomes' section of the diary entry for Unit 4.	C	
Write well-structured, easily maintained JavaScript code following accepted good practice, including	I have evidence of this listed below, as well as in the Unit 5 diary entry under the 'How I have met the Learning Outcomes' section, which both describe how I checked, edited, and wrote the JavaScript code.	B	
<ul style="list-style-type: none"> general appearance and form: commented, properly laid out, appropriate capitalization 	Using my Unit 4 notes on JavaScript code conventions as well as the HTML comments section of my Unit 2 notes, I added comments and formatted the code according to standards. I also used CamelCase notation, with lowercase first letters of variables.	B	
<ul style="list-style-type: none"> structure: modular, using functions and objects effectively 	I structured the code according to notes on proper spacing given in The Landing FAQs and the Moodle study guide. I also used 2 functions, 1 custom function, and 29 variables in the 'sorter.js' program code.	B	
<ul style="list-style-type: none"> standards-compliant 	I tested the sorting and filter elements on Firefox, Opera, Internet Explorer, and Chrome browsers. I also ensured that the new HTML code and JavaScript were compliant with the XHTML 1.0 strict standards and the WCAG 2.0 standards.	B	
<ul style="list-style-type: none"> accessible 	The website should have an 'A' rating by the WCAG 2.0 standards for the most part, and the new changes to the website have also been tested on an Android device.	B	
Write JavaScript code that works in all major browsers (including IE, Mozilla-based browsers such as Firefox, Opera, Konqueror, Safari, Chrome).	I tested the 'sorter.js' JavaScript code in the Firefox, Chrome, and Opera browsers, as well as on an android tablet.	B	

Effectively debug JavaScript code, making use of good practice and debugging tools.	I used the built-in Firefox debugger, by going line-by-line through code until it reported an error, which I read and researched to find where my code had a bug. I used the breakpoint feature to stop the execution of the code at the point where I wanted to see the execution in detail. I also kept track of the variables in the function scope in the variables tab on the right side of the debugger.	B	
Use JavaScript libraries (e.g., JQuery) to create dynamic pages.	I have used JQuery in 5 different JavaScript files and functions, and used over 11 functions and properties of JQuery, as listed in the Unit 6 diary entry under the section "How I have met the Learning Outcomes".	D	
Use JavaScript to access and use web services for dynamic content (AJAX, JSON, etc.).	I have implemented 2 web services to give dynamic content to the users. This includes loading YouTube videos from a Playlist, and a share button to post a link to the web page of the website the user is currently viewing onto their timeline. This is described in the Unit 7 diary entry under the section "How I have met the Learning Outcomes".	C	
Overall	For the first 5 units, my work is about 'B' level, with full implementations of website design as described and required by the learning outcomes. In units 6 and 7, I didn't give as much extensive evidence of learning the learning outcomes, but they are still adequate.	B	

External Site Data Notes

iFrames can incorporate external site data.

AJAX: Asynchronous Javascript and Xml

AJAX is a set of methods allowing pages to be updated without reloading the whole page.

AJAX also allows data to be sent from one page to a receiving application without leaving the current page.

XMLHttpRequest: Allows the sending and receiving of XML including XHTML, along with other data types such as plain text and JSON.

JSON: JavaScript Object Notation. Standard format for working with JavaScript objects from other sites.

You can send/receive data from APIs.

XMLHttpRequest cannot access file content from your own computer (not HTTP).

JSONP is good for getting data from other pages, and is a good alternative to XMLHttpRequest.

AJAX:

Called from JQuery using:

```
$.ajax({  
    type: "GET",  
    url: "page.xml",  
    datatype: "xml",  
    success: successFunction  
});
```

successFunction should be as follows:

```
function successFunction(data, status) {  
    $(data).find('div'); //Retrieve array of elements of type div from data from page  
}
```

JSON:

JSON: JavaScript Object Notation

Data hierarchy only using {[:,"

A variable can store a JSON object like this:

```
var myPairs = {"name":"JSON", "type":"array", "usability":"serversend"};
```

```
var myJson = JSON.stringify(myObj);
```

JSON can be sent through a URL address like this:

```
var myUrl = "http://www.website.com/index.php?x=" + myJson;
```

JSON can be retrieved like this:

```
Var myObj = JSON.parse(myJson);
```

And read like this:

```
$("#content").innerHTML = myObj.name;
```

JSON can be passed around using AJAX