Wolfgang C. Strack

CS175: From JavaScript to AJAX

15 December 2015

Running HTML:	http://toolkit.cs.ohlone.edu/~gen246/project/
JavaScript:	http://toolkit.cs.ohlone.edu/~gen246/project/js/index.js
Main CSS:	http://toolkit.cs.ohlone.edu/~gen246/project/css/index.css

To view the HTML for the project, use the browser's "View Source" tools.

Onward, the site design is based off of a (very) rough-draft design of another personal web-development project I am currently working on. That project is called "AID: Assist Improved" and more information on it can be found at: https://github.com/vulfgang/AID.

In the JavaScript file linked above, everything is split up into functions called by the main() functions. There is some documentation above each function to explain what its doing. The rest of this presentation will consist of breaking down each JavaScript/jQuery function individually and explaining in further detail what is going on as well as my thoughts on the code when I was implementing it, difficulties I encountered, and what I have learned.

function main($)

I felt the need to have a main function just so I could keep things more organized in the code. JavaScript tends to get very messy and repetitive so I try to organize and format as much of it as I can and increase readability. There isn't much to explain here—the main method calls every other method in the script (besides defer_jquery()) and methods that involve jQuery are wrapped in a $(document).ready to ensure those methods work properly. The $ parameter is passed in as the jQuery variable.

function insert_common_includes()

This function probably deserves the most explaining out of all the other functions. When coding this function, I was trying to solve the problem of not having to type out multiple different `<script>` and `<link>` tags on multiple pages. In other words, if I had a bunch of style-sheets I wanted to include in all of the pages of my website, I wanted a way to dynamically insert tags linking them instead of hard-coding them all by myself on **each** page. While this can be done with a server-side language, I felt that I could still find a way to do it with just front-end JavaScript—what this class focuses on.

And indeed, I did find a neat way to do it—I used AJAX. Specifically, I used jQuery's AJAX helper method to retrieve the style-sheets I wanted with a GET request to the server. Then, in a callback function argument of the AJAX helper method, I would append the CSS data that was received to the `<head>` of the page as a `<style>` tag.

While this function appends CSS style-sheets dynamically, it can also append JavaScripts dynamically as well. In my case, however, the only JavaScripts I wanted included (besides index.js itself) were jQuery tools. In order for this function to work, jQuery needs to already be loaded so that's why I could not dynamically insert using this function.

function execute_jquery_ui()

The first thing that might be noticed is that this function is very short. It does only one thing, to be exact, and that's to activate the jQuery UI tabs on the page. I had originally planned to include some implementations of jQuery UI effects here. In fact, I did previously implement these examples but realized that they didn't really do anything good for the site. Rather, the adding of different effects—whether they be animations, style changes, other UI elements—made the site seem cluttered and more unprofessional. With that said, I opted to leave that jQuery UI functionality out in favor of a cleaner look for the site.

Anyway, the jQuery tabs here can all be seen in the red navigation bar near the top of the page. These pages are the

Home, Favorites, About, and Contact pages. Originally, the links in this navigation bar were supposed to lead to separate HTML pages, but I felt it might be useful to turn this functionality of the site into tabs.

function execute_jquery_validation()

When I first learned jQuery validations in this class, I realized how very useful and handy the methods can be. Hand-coding the validations was a good learning experience as well and made me appreciate the jQuery validation methods even more since they abstract away all of the small details.

With all of that said, I implemented the use of jQuery form validations in this function. The implemented validations come with custom CSS that styles error messages to appear red and next to the invalid field. In the process of implementing this particular example, I learned that jQuery automatically detects email fields if you set the form like so: `<input type="email"...>`. This is as opposed to setting the type to just “text” and then having to specify sub-options for your email field when calling the jQuery validation method.

function execute_jquery_other()

This function executes any other jQuery functionality that doesn't involve jQuery UI or validation. I did not want to go overboard with the animations, style changes, etc. because, again, I had actually implemented several examples using `$().animate` or other combinations of jQuery methods with `$().mouseover, $().click, $().toggle,` etc. but it all ended up just making the site look worse rather than better.

In place, I kept two jQuery calls that are both combinations of `$().hover` and `$().add/removeClass`. These calls go with their own custom CSS which can be found in the link specified above (index.css). The first call makes it such that whenever the mouse hovers over a “search result” on the home page, the table row it is organized in will become bold so as to give it a “focused” look. The second call to $().hover makes the navigation-bar links' text focus whenever the cursor mouses over them.

function defer_jQuery(method)

The inspiration for this function sprung from my frustration of getting errors just because jQuery wasn't loaded into the page. In essence, my custom scripts would load and execute before jQuery was loaded and executed, resulting in a long list full of errors that a bunch of functions are not defined. The most frustrating part about this was that it would only happen sometimes. Since my scripts were loading asynchronously, sometimes jQuery would load before everything else and sometimes it would not, giving me the errors.

After getting fed up with the errors and inconsistency of it, I eventually set out to implement this function. Thankfully, this resulted in just what I wanted: no more inconsistent errors. The function does what it describes—it is given a method as a parameter and will only call that method once jQuery has finished loading successfully into the page. Once jQuery is loaded, this function will then call the parameter and pass it jQuery like so: `method(jQuery)`. In this particular implementation, defer_jQuery is called and passed the `main($)` function.

Final Thoughts

For all the explanation as well as total time spent on the code for this project, I feel that more of the time and effort went into the HTML rather than the JavaScript/jQuery itself. I do not see this as a bad thing, however. Earlier, I mentioned that I grew much more appreciation for the jQuery validation methods after basically implementing the methods myself. I feel that I can say the same for the rest of the jQuery I have used throughout the project and also throughout the course. We started learning and coding from raw JavaScript DOM manipulation; jQuery provides a library of functions that make all of that DOM manipulation we learned be far easier and shorter to implement. I think that in itself speaks volumes about the usefulness of the concepts within the examples I've implemented in this project.

In short, I've learned and really come to appreciate the things we've learned in this course that we can do with JavaScript/jQuery as well as how jQuery can be used to condense certain bits of JavaScript drastically. I'm also glad I got a taste of AJAX, even if it didn't require using a server-side language. From the example I implemented, I read plenty of information about it and that in itself was useful for me.