AJAX, fetch, and Axios

Asynchronous JavaScript?

HTTP Requests in the Browser

- URL bar
- Links
- JavaScript
  - window.location.href = ‘http://www.google.com'
- Submitting forms (GET/POST)

All of the above make the browser navigate and retrieve new documents

HTTP Requests in the Browser

- Often times for each of the above actions, views are stored on the server and served up as HTML pages

- When a user goes to a new page, the browser navigates in totality, refreshing and retrieving a brand new HTML document.

- Each page, since it's a new page, retrieves stylesheets, scripts, files, etc.

What is AJAX?

- “Asynchronous JavaScript And XML”
- Making background HTTP requests using JavaScript
- Handling the response of those HTTP requests with JavaScript
- No page refresh necessary
- `window.fetch()`

Asynchronous JS

- Basically, we are referring to JavaScripts ability to act in a non-blocking manner.

- Imagine if every network request that took time to give us a response blocked any other operations from executing? The entire internet would be at a stand-still

Asynchronous JS

- The initial method developed to deal with asynchronous code was to use callbacks (hello, familiar!) to provide a function to run once a request has been resolved.
- The following code snippet is an example of a callback being used to deal with the result of the async downloadPhoto function

```
downloadPhoto('http://coolcats.com/cat.gif', handlePhoto)

function handlePhoto (error, photo) {
  if (error) console.error('Download error!', error)
  else console.log('Download finished', photo)
}

console.log('Download started')
```

Asynchronous JS

- While callbacks are important to understand, they can lead to something referred to as callback hell:

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Async action # 1

Action on #1 Result

Async action # 2 using async #1 Result

Action on # 2 result

Asynchronous JS

- To better understand proper asynchronous callback usage, there is a great website called callbackhell.com that does a good job of getting into best practices for composing async callback functions and avoiding the dreaded ‘callback hell’.

- We will explore a better option later.

Why AJAX?

- AJAX allows us to build Single Page Applications (SPAs). Via wikipedia:
    - “An SPA is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server”
- SPAs mean no reload or “refresh” within the user interface
    - JS manipulates the DOM as the user interacts
- User experience similar to a native / mobile application

Wait, what is `fetch()`?

- The Fetch API provides an interface for fetching resources (including across the network).

- Provides a generic definition of Request and Response objects, as well as other things involved with network requests

- The `fetch()` method takes one mandatory argument, the path to the resource you want to fetch, and returns a promise that resolves to the response to that request (successful or not).

- You can optionally pass an init options object as second argument (used to configure req headers for other types of HTTP requests such as PUT, POST, DELETE)

Using `fetch()`

```
fetch('http://example.com/movies.json')
  .then(function(response) {
    return response.json();
  })
  .then(function(myJson) {
    console.log(JSON.stringify(myJson));
});
```

The simplest use of fetch takes one argument - the path to the resource you want to fetch - and returns a promise containing the response body.

Above, we are fetching a JSON file across the network to print to the console.

What is Axios?

- Axios is a promise-based HTTP client for JavaScript. It allows you to:
  - Make XMLHttpRequests from the browser
  - Make http requests from node.js
  - Supports the Promise API
  - Automatic transforms for JSON data

Using Axios

```
axios.get('http://jsonplaceholder.typicode.com/todos')
  .then(function (response) {
    resultElement.innerHTML = generateSuccessHTMLOutput(response);
  })
  .catch(function (error) {
    resultElement.innerHTML = generateErrorHTMLOutput(error);
  });
}
```

- Above, we are using the `axios.get(<uri>)` function to send an HTTP GET request to the endpoint that we want to get information from

Using Axios

- Axios provides more functions to make other network requests as well, matching the HTTP verbs that you wish to execute, such as:
  - `axios.post(<uri>, <payload>)`
  - `axios.put(<uri>, <payload>)`
  - `axios.delete(<uri>, <payload>)`
- You can also pass a config object instead:

```
axios({
  method: ‘get’,
  url: ‘http://dummy.data'
  responseType: ‘<insert response type, e.g. stream>’
})
```

Using Axios

- In order to use Axios, you can simply `npm install axios` in your project and either import or require it to use.

Fetch vs Axios

- Fetch API is built into the window object, and therefore doesn't need to be installed as a dependency or imported in client-side code.

- Axios needs to be installed as a dependency. However, it automatically transforms JSON data for you, thereby avoiding the two-step process of making a `.fetch()` request and then a second call to the `.json()` method on the response.

- There is a good medium article outlining some more differences here: https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5