Ch. 13 AJAX

- Cross Origin Resource Sharing (CORS) is a method through which resources can be shared across domains.
- AJAX is actually an acronym: Asynchronous JavaScript and XML.
  - Asynchronous – the software doesn't stop and wait after sending a request.
  - XML – has been more or less replaced by JSON now, either way, allows for storing the characteristics of lists of items.
- The fetch API is used to grab data from websites, such as a JSON file containing the latest weather data. An example of how to use the weather API follows:

  -
    ```
    const url = 'https:example.com/data';
    fetch(url)
    .then((response) => {
      if(response.ok) {
        return response;
      }
      throw Error(response.statusText);
    })
    .then( response => // do something with response )
    .catch( error => console.log('There was an error!') )
    ```
- Turning JSON into an object:

  -
    ```
    fetch(url)
    .then( response => response.json() ); // transforms the JSON data
    .then( data => console.log(Object.entries(data)) )
    .catch( error => console.log('There was an error: ', error))
    ```
- AJAX Load is worth looking into for creating things such as loading spinners.
- This was an interesting application to me – overriding the default behavior of a button:

  -
    ```
    const form = document.forms['todo'];
    form.addEventListener('submit', addTask, false);

    function addTask(event) {
      event.preventDefault();
      const number = form.task.value;
      const task = {
        userId: 1,
        title: form.task.value,
        completed: false
      }
      const data = JSON.stringify(task);
      const url = 'https://jsonplaceholder.typicode.com/todos';
    ```

```
        const headers = new Headers({
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        });
        const request = new Request(url,
        {
            method: 'POST',
            header: headers,
            body: data
        }
        )

        fetch(request)
        .then( response => response.json() )
        .then( task => console.log(`Task saved with an id of ${task.id}`) )
        .catch( error => console.log('There was an error:', error))

    }
```

Ch. 8 Forms

- Traditionally were processed in the backend, though processing forms using JavaScript in the frontend is becoming more common.
- Two possible methods of identifying forms in JavaScript are the following:
    - const form = document.forms['search'];
    - const form = document.forms[0];
        - Note that forms will be listed within the collection in the order that they appear on the document.
- Note that while reset buttons are an option, they typically are not recommended for use.
- When the form is submitted, it can be intercepted using the following line of code:
    - const form = document.forms['search'];
    - form.addEventListener ('submit', search, false);
    -
    - function search() {
    -     alert(' Form Submitted');
    -     event.preventDefault();
    - }
- This will be a good chapter to reference to identify different form input types, such as the text box or radio buttons.
- Form validation should occur on both the front and backends, in order to enhance both security and usability.

- Note that most users prefer instant feedback when filling out a form. Rather than letting them hit enter just to fail, tell them as soon as they've done something wrong.
- It may be worth disabling the submit button altogether if there are field errors.

Using FormData Object Effectively

- Make sure inputs have **both** a name and an ID.
- There are definitely ways to get data out of a form that are more efficient than referencing fields one at a time.

Client Side Form Validation

- As mentioned before, validation should occur on both the front and backend.
- Some of the things that forms can look for are:
    - Required field, which must always be indicated
    - minlength and maxlength (text)
    - min and max (numerical)
    - type
    - pattern (regex)
- It is typically good practice to override the default error messages with custom JavaScript, in order increase the user accessibility of the website.