

This assignment does not count toward the final grade.

Lab 11

- Due May 3 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip
- Available until May 4 at 12:04am

This assignment was locked May 4 at 12:04am.

CS-546 Lab 11

AJAX

For this lab, you will be using HTML, JQuery and Client-side JavaScript on the user's browser to make a simple application that makes AJAX requests for the data needed and then inject the elements and data onto the page. (You may use Axios or fetch as well on the client-side as a replacement to using jQuery/AJAX. For Axios, you need to view their documentation and they provide a script tag to include in your HTML, you cannot use the Axios NPM module on the client-side, you must use the script tag supplied in their documentation)

YOU MUST use the directory and file structure in the code stub or points will be deducted. You can download the starter template here: [Lab11_stub.zip](https://sit.instructure.com/courses/71954/files/13305929?wrap=1)

(<https://sit.instructure.com/courses/71954/files/13305929?wrap=1>)_ 

(https://sit.instructure.com/courses/71954/files/13305929/download?download_frd=1)

PLEASE NOTE: THE STUB DOES NOT INCLUDE THE PACKAGE.JSON FILE. YOU WILL NEED TO CREATE IT! DO NOT ADD ANY OTHER FILE OR FOLDER APART FROM PACKAGE.JSON FILE.

Open Movie API

For this lab, you will be using two endpoints of the Open Movie API for your AJAX calls.

Open Movie Database API

You will be using the [Open Movie Database API \(https://www.omdbapi.com/\)](https://www.omdbapi.com/). You will need an API key, the developer of the API gave me a key for all students to use **CS546**. **You will not be able to make requests to the API without using the API key. PLEASE DO NOT SHARE THIS API KEY WITH ANYONE AND DO NOT PUSH IT UP TO GITHUB OR ANYWHERE ONLINE PUBLICLY. THE**

DEVELOPER OF THE API WAS KIND ENOUGH TO GIVE ME AN API KEY FOR THE WHOLE CLASS TO USE FOR ASSIGNMENTS, PLEASE DO NOT SHARE IT ANYWHERE!

You will be using two endpoints of the API for your AJAX calls. The search movie endpoint where you pass the search term as a query string parameter: [http://www.omdbapi.com/?apikey=CS546&s=\[movie_search_term\]](http://www.omdbapi.com/?apikey=CS546&s=[movie_search_term]) (<http://www.omdbapi.com/?apikey=CS546&s=Breakfast>) and then you'll get an individual movie data using the endpoint

[http://www.omdbapi.com/?apikey=CS546&i=\[movie ID here\]](http://www.omdbapi.com/?apikey=CS546&i=[movie ID here]) (<http://www.omdbapi.com/?apikey=CS546&i=tt0088847>) (in the data, the id field is named: `imdbID`)

You will use these two endpoints to make your AJAX calls.

The Server

Your server this week should not do any of the processing or calculations. Your server only exists to allow someone to get to the HTML Page and download the associated assets to run the application.

The Whole Application

This route will respond with a static HTML file and all of the functionality will be done in a client-side JS file. You will make client-side AJAX requests to the API and use jQuery to target and create elements on the page.

Your page should have a few basic user interface elements:

- A header tag, with an h1 inside it, naming your site, with a title for your page (the title tag in the head element should mimic this)
- A footer with your name, student ID, and any other info about yourself you wish to include
- An empty unordered list with an id of `searchResults` that is initially hidden.
- A div with an id of `movieDetails` that is initially hidden.
- A form with an id of `searchMovieForm`.
- A text input with an id of `movie_search_term`.
- A label with a `for` attribute referencing your input
- A button to submit the form
- A link that links back to the "/" route with the text "New Search" and that has an id of `rootLink` This link will initially be hidden. It will ONLY be displayed when the `movieDetails` element is being displayed or ONLY when a search is performed and search results are being displayed. This link will simply trigger a reload of the page, which then will display the page with just the search form so the user can search for a new movie.


AJAX Requests

Remember, you will be ONLY using client-side JavaScript!


1. **Page load:** When the page loads, You will display the `searchMovieForm` form. You will have the following HTML inside the body element on page load:

```
<form id="searchMovieForm">
<label for="movie_search_term">Search for a Movie</label>
<input id="movie_search_term" type="text">
<button type="submit">Search Movies</button>
</form>
<ul id="searchResults" hidden></ul>
<div id="movieDetails" hidden></div>
<a id="rootLink" href="/" hidden>New Search</a>
```

2. **Search Form Submission:** On the client-side, you will intercept the form submission event. If there is no value for the `movie_search_term` input when the form is submitted, you should not continue and instead should inform them of an error somehow. (don't forget to take into account if they just submit the form with a bunch of spaces as the value!) If there is a value, you will first hide the `movieDetails` div, in case it is being displayed, you will then empty the list item elements in the `searchResults` element (because there may be elements from the initial `searchResults` still there from a previous search, even if it's hidden), then query the API for that `movie_search_term` using an AJAX request. Once the AJAX request returns the data, you will then create list items of links for each movie that is returned using jQuery/or the DOM API. The link text will be the name of the movie, and the `href` attribute will have a value of `'javascript:void(0)'` and you will create a data-id attribute that contains the ID for that movie (see below example). You will then append each list item to the `SearchResults` UL element and then show the `SearchResults` element (make sure you hide the `movieDetails` element). Just as you did in Lab 8, you will show up to 20 search results and you'll have to get the 2nd page of the search results, like you did in lab 8. For example, if the user typed `breakfast` in the input field, you would make the axios call to: <http://www.omdbapi.com/?apikey=CS546&s=breakfast> (gets the first 10 results). <http://www.omdbapi.com/?apikey=CS546&s=breakfast&page=2> (gets the second 10 results). You will then combine these results and display them.

Endpoint to be used: [http://www.omdbapi.com/?apikey=CS546&s=\[movie_search_term\]](http://www.omdbapi.com/?apikey=CS546&s=[movie_search_term]) 
(<http://www.omdbapi.com/?apikey=CS546&s=Breakfast>)

3. **Link Clicked:** For the link, you will need to call a function on the click event of the link and not the default link behavior of a link as a link wants to send you somewhere (triggers a load), so you are intercepting the default click event as you do with the form. (do not forget to use `preventDefault()`). When the link to a movie is clicked, it will hide the `SearchResults` element, you will then empty the `movieDetails` element (just in case there was movie data previously loaded into the `movieDetails` element). It will then make an AJAX request to the URL and fetch the data for that movie (that was the data-id property in your link). You will parse through the movie data returned from the AJAX request. You will create the following HTML elements using jQuery inside the `movieDetails` div (in the below example, say the user clicked on The Breakfast Club in the search results list, the `data-id` for that movie in the search results would be: `<a data-id="tt0088847">The Breakfast Club` You also should set the href attribute to `'javascript:void(0)'`. So the full link in the search results will look like this: `The Breakfast Club` In your click event handler for the link, you will read the `data-id` property and make a AJAX request to:

<https://www.omdbapi.com/?apikey=CS546&i=tt0088847>  <https://www.omdbapi.com/?apikey=CS546&i=tt0088847>). You will then generate the below HTML and append it to the movieDetails element and then show the movieDetails element:

```
<article>
<h1>The Breakfast Club</h1>
  

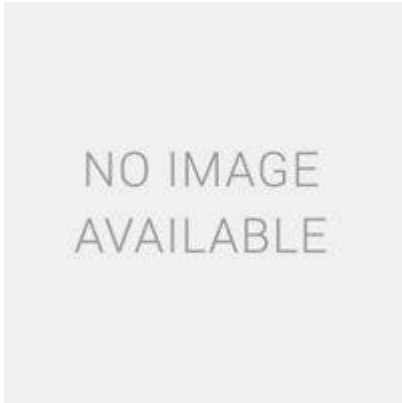
  <h2>Plot</h2>
  <p>Five high school students meet in Saturday detention and discover how they have a great deal more in common than they thought.</p>
  <section>
    <h3>Info</h3>
    <dl>
      <dt>Year Released:</dt>
      <dd>1985</dd>
      <dt>Rated:</dt>
      <dd>R</dd>
      <dt>Runtime:</dt>
      <dd>97 min</dd>
      <dt>Genre(s):</dt>
      <dd>Comedy, Drama</dd>
      <dt>Box Office Earnings:</dt>
      <dd>$45,875,171</dd>
      <dt>DVD Release Date:</dt>
      <dd>15 Jun 2012</dd>
    </dl>
  </section>

  <section>
    <h4>Cast and Crew</h4>
    <p><strong>Director:</strong> John Hughes</p>
    <p><strong>Writer:</strong> John Hughes</p>
    <p><strong>Cast:</strong> Emilio Estevez, Judd Nelson, Molly Ringwald</p>
  </section>

  <section>
    <h4>Ratings</h4>
    <table class="my_coolratings_table">
      <tr>
        <th>Source</th>
        <th>Rating</th>
      </tr>
      <tr>
        <td>Internet Movie Database</td>
        <td>7.8/10</td>
      </tr>
      <tr>
        <td>Rotten Tomatoes</td>
        <td>89%</td>
      </tr>
      <tr>
        <td>Metacritic</td>
        <td>66/100</td>
      </tr>
    </table>
  </section>
</article>
```

NOTE: Not all movies have ALL data displayed in the `movieDetails` element, which will cause your application to not work correctly when a movie link is clicked if it doesn't have all the needed data needed for the `movieDetails` element. You will be required to check each field returned from the api that you will be injecting into an element element. If there is no value for a field, you will show "N/A" instead of that field's value on that movie detail element. For the

image, if there is no image, you can load a generic "no image" image that is served from your public directory. You can save this one (right click and save the image):



Style:

`/public/css/styles.css`

In this file, you will define at least 5 CSS rule-sets to apply to your application

Requirements

1. All previous requirements still apply.
2. You **must remember** to update your package.json file to set `app.js` as your starting script!
3. **Your HTML must be valid** (https://validator.w3.org/#validate_by_input) or you will lose points on the assignment.
4. Your HTML must make semantical sense; usage of tags for the purpose of simply changing the style of elements (such as `i`, `b`, `font`, `center`, etc) will result in points being deducted; think in terms of content first.
5. **You can be as creative as you'd like to fulfill front-end requirements**; if an implementation is not explicitly stated, however you go about it is fine (provided the HTML is valid and semantical). Design is not a factor in this course.
6. **Your client side JavaScript must be in its own file and referenced from the HTML accordingly.**
7. All inputs must be properly labeled!