Lab 8

New Attempt

- Due Apr 3 by 11:59pm
- Points 100
- Submitting a file upload

CS-546 Lab 8

Template Time

For this lab, you will be using HTML, CSS, and Handlebars to make your first simple templated web application! You will be building a form that allows you to search through movies in the **Open Movie Database API** (https://www.omdbapi.com/)

You will not need to use a database for this lab.

You **must** use the `async/await` keywords (not Promises). You will also be using `axios` (https://github.com/axios/axios), which is a HTTP client for Node.js; you can install it with `npm i axios`.

Open Movie Database API

You will be using the**Open Movie Database API** (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!**

Please look at the data returned so you know the schema of the data and the objects it returns **PLEASE MAKE SURE YOU READ THE DOCUMENTATION FOR THE API! IT IS SUPER IMPORTANT THAT YOU UNDERSTAND HOW AN API WORKS AND THE FORMAT OF THE DATA SO YOU CAN USE IT EFFECTIVELY! Before you ask questions LOOK at the data. You can't really use an API if you don't understand the data returned so it's SUPER important that you actually look at the data and understand the schema of the data being returned!**

You will be using two endpoints of the API for your Axios calls. The search movie endpoint where you pass the search term as a query string parameter: **http://www.omdbapi.com/?apikey=CS546&s=[search_term_here]** (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=tt0088847) (in the data, the id field is named: `imdbID`)

You will use these two endpoints to make your axios.get calls depending on which route is called.

YOU MUST use the directory and file structure in the code stub or points will be deducted. You can download the starter template here: **Lab8_Stub.zip** (https://sit.instructure.com/courses/71954/files/13215439?wrap=1) (https://sit.instructure.com/courses/71954/files/13215439/download?download_frd=1) (https://sit.instructure.com/courses/64643/files/11372587?wrap=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. DO NOT FORGET THE START COMMAND OR THE "type": "module" property

You will be making three routes/pages in your application:

- `http://localhost:3000/` the main page of this application will provide a search form to start a search of movies by title keyword.
- `http://localhost:3000/searchmovies` this page will make the axios call to the search endpoint and return up to 20 matching results that contain the provided request form param, `searchMoviesByName` (NOTE: When you search the API, only the first page of results is returned, and displays 10 results, you'll also have to get the 2nd page of results (if available) You can do this by adding a page query string param. So for example: **http://www.omdbapi.com/?apikey=CS546&s=Batman** (http://www.omdbapi.com/?apikey=CS546&s=Batman) Brings back the first 10 results, **http://www.omdbapi.com/?apikey=CS546&s=Batman&page=2** (http://www.omdbapi.com/?apikey=CS546&s=Batman&page=2) Returns the 2nd page with 10 results. You will show up to 20 (if there are less than 20 results, then just show how many there are)
- `http://localhost:3000/movie/{id}` this page will show all the details of the movie with the id matching the provided URL param, `id.` So for example: **http://localhost:3000/movie/tt0088847** (http://localhost:3000/movie/tt0088847) makes an axios call to the API endpoint: **http://www.omdbapi.com/?apikey=CS546&i=tt0088847** (http://www.omdbapi.com/?apikey=CS546&i=tt0088847)

All other URLS should return a 404

`GET http://localhost:3000/`

This page will respond with a valid HTML document. The title of the document should be "Movie *Finder*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document.

Your page should reference a CSS file, `/public/css/styles.css`; this file should have *at least 10 rulesets* you must use at least 5 of them on this page.

You should have a `main` element, and inside of the `main` element have a `p` element with a brief (2-3 sentence description) of what your website does.

Also inside the `main` element, you will have a `form`; this `form` will `POST` to `/searchmovies`. This `form` will have an `input` and a `label`; You will give the input a name attribute and id attribute of "`searchMoviesByName`". The `label` should properly reference the same `id` as the `input`. You should

also have an `input` with a **type of** `submit` that submits the form.

`POST http://localhost:3000/searchmovies`

This route will read the `searchMoviesByName` parameter and then make an axios call to the API endpoint searching for that keyword. 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 (http://www.omdbapi.com/?apikey=CS546&s=breakfast)** (gets the first 10 results). **http://www.omdbapi.com/?apikey=CS546&s=breakfast&page=2 (http://www.omdbapi.com/?apikey=CS546&s=breakfast)** (gets the second 10 results)

This route will respond with a valid HTML document with the results returned from the API. The title of the document should be "Movies *Found*". You should have the title set as the `<title>` element of the HTML document and as an `h1` in your document. In an `h2` element, you will print the supplied `searchMoviesByName`.

Your page should reference a CSS file, `/public/css/styles.css`; this file should have *at least 10 rulesets* you must use at least 5 of them on this page.

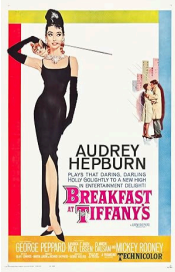
You should have a `main` element, and inside of the `main` element have a `ol` tag that has a list of up to 20 movies matching the `searchMoviesByName` found in the request body in the following format (after searching `Breakfast`). **DO NOT SHOW MORE THAN 20 Movies. Each list item will show the image of the movie, and the movie title, it should link to your /movie/:id route. One of your CSS rules, should scale the image from the data so it is just a thumbnail. Otherwise, the image will be huge! You will lose points if you make it that huge. I found a good width is 75px and a good height is 117px. DO NOT FORGET TO ADD THE ALT ATTRIBUTE TO the Image. Set it to be "[movie name] Poster" as shown below. (in the example below, I am just showing 3 of the 20 results, make sure you show up to 20 results!**

```
<ol>
    <li> <a href="/movie/tt0088847">The Breakfast Club <br> <img class="searchResultImage" alt="The Break
fast Club Poster" src="https://m.media-amazon.com/images/M/MV5BOTM5N2ZmZTMtNjlmOS00YzlkLTk3YjEtNTU1ZmQ5OT
dhODZhXkEyXkFqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg"></a>
    <li>
        <a href="/movie/tt0054698">Breakfast at Tiffany's <br> <img class="searchResultImage" alt="Breakf
ast at Tiffany's Poster" src="https://m.media-amazon.com/images/M/MV5BNGEwMTRmZTQtMDY4Ni00MTliLTk5ZmMtOWM
xYWMyMTllMDg0L2ltYWdlL2ltYWdlXkEyXkFqcGdeQXVyNjc1NTYyMjg@._V1_SX300.jpg"> </a>
    </li>
    <li>
        <a href="/movie/tt0411195">Breakfast on Pluto <br> <img class="searchResultImage" alt="Breakfast
on Pluto Poster" src="https://m.media-amazon.com/images/M/MV5BYmU5YjJlYjEtNGE5Ni00ODA5LWE4ZTAtNzZlNDhlODI
zMzE4XkEyXkFqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg"> </a>
    </li>
</ol>
```

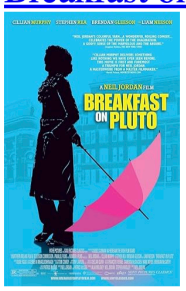
The rendered page should look something like this:

1. The Breakfast Club

2. Breakfast at Tiffany's



3. Breakfast on Pluto



You must also provide an `a` tag that links back to your `/` route with the text "`Search for Another Movie"`.

If no matches are found, you will send a response status code of 404 and render the following HTML:

```
<p class="not-found">We're sorry, but no results were found for {searchMoviesByName}.</p>
```

If the user does not input text into their form or enters just spaces into the input field, make sure to give a response status code of 400 on the page, and render an HTML page with a paragraph class called `error`; this paragraph should describe the error.

`GET http://localhost:3000/movie/{id}`

This route will query the API using the the `id` parameter in the URL (for example: **http://www.omdbapi.com/?apikey=CS546&i=tt0088847 (http://www.omdbapi.com/?apikey=CS546&i=tt0088847)**) and will respond with a valid HTML document with the movie details. The title of the document should be the `name` of the movie. You should have the title set as the `<title>` element of the HTML document.

Your page should reference a CSS file, `/public/css/styles.css`; his file should have *at least 10 rulesets* you must use at least 5 of them on this page.

You should have a `main` element, and inside of the `main` element, you will have an `article` tag that has an `h1` with the Movie `title`, an `img` which the `src` is set to the value read from the poster field in the data which is a URL to an image for the movie poster. and will have the movie data. Please see the rendered HTML example below. Not all fields from the data are used, refer to the HTML example below for the fields/elements. you must show on your page.

Matching Movie Data Returned from API:


```
{
  "Title": "The Breakfast Club",
  "Year": "1985",
  "Rated": "R",
  "Released": "15 Feb 1985",
  "Runtime": "97 min",
  "Genre": "Comedy, Drama",
  "Director": "John Hughes",
  "Writer": "John Hughes",
  "Actors": "Emilio Estevez, Judd Nelson, Molly Ringwald",
  "Plot": "Five high school students meet in Saturday detention and discover how they have a great deal m
ore in common than they thought.",
  "Language": "English",
  "Country": "United States",
  "Awards": "4 wins",
  "Poster": "https://m.media-amazon.com/images/M/MV5BOTM5N2ZmZTMtNjlmOS00YzlkLTk3YjEtNTU1ZmQ5OTdhODZhXkEy
XkFqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg (https://m.media-amazon.com/images/M/MV5BOTM5N2ZmZTMtNjlmOS00YzlkLTk3Yj
EtNTU1ZmQ5OTdhODZhXkEyXkFqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg)",
  "Ratings": [
    {
      "Source": "Internet Movie Database",
      "Value": "7.8/10"
    },
    {
      "Source": "Rotten Tomatoes",
      "Value": "89%"
    },
    {
      "Source": "Metacritic",
      "Value": "66/100"
    }
  ],
  "Metascore": "66",
  "imdbRating": "7.8",
  "imdbVotes": "433,376",
  "imdbID": "tt0088847",
  "Type": "movie",
  "DVD": "15 Jun 2012",
  "BoxOffice": "$45,875,171",
  "Production": "N/A",
  "Website": "N/A",
  "Response": "True"
}
```


HTML Format with the rendered data for the Movie. This will go into your `main` element:

```
<article>
  <h1>The Breakfast Club</h1>
  <img alt="The Breakfast Club Poster" src="https://m.media-amazon.com/images/M/MV5BOTM5N2ZmZTMtNjlmOS00Y
zlkLTk3YjEtNTU1ZmQ5OTdhODZhXkEyXkFqcGdeQXVyMTQxNzMzNDI@._V1_SX300.jpg">

  <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>
```

Your HTML structure and elements MUST match this (you can add class attributes or id attributes to target the elements with CSS, but other than that, you must follow the example above!). You must also provide an `a` tag that links back to your `/` route with the text "`Search for Another Movie"`.

If there is no movie found for the given ID, make sure to give a response status code of 404 on the page, and render an HTML page with a paragraph class called `error`; this paragraph should describe the error. This route MUST work for every valid movie ID in the API!

`http://localhost:3000/public/css/styles.css`

This file should have 10 total rulesets that apply to your various pages. Each page must use at least 5 rulesets of the 10. You can include more than 10 if you so desire.

References and Packages

Basic CSS info can easily be referenced in the [MDN CSS tutorial (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started)](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started).

Hints

You can use variables in your handlebars layout, that you pass to res.render. For example, in your layout you could have:

```
<meta name="keywords" content="{{keywords}}" />
```

And in your route:

```
res.render("someView", {keywords: "dogs coffee keto"});
```

Which will render as:

```
<meta name="keywords" content="dogs coffee keto" />
```

Or, perhaps, the title tag.

Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function
4. Check for arguments existing and of proper type.
5. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
6. You **MUST** use `async/await` for all asynchronous operations.
7. You **must remember** to update your package.json file to set `app.js` as your starting script!
8. [Your HTML must be valid (https://validator.w3.org/#validate_by_input)](https://validator.w3.org/#validate_by_input) or you will lose points on the assignment.
9. 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, then style with your CSS.
10. **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.
11. All inputs must be properly labeled!
12. All previous requirements about the `package.json` author, start task, dependenices, etc. still apply