WEB422 Assignment 2

Submission Deadline:

Friday, February 3rd @ 11:59pm

Assessment Weight:

9% of your final course Grade

Objective:

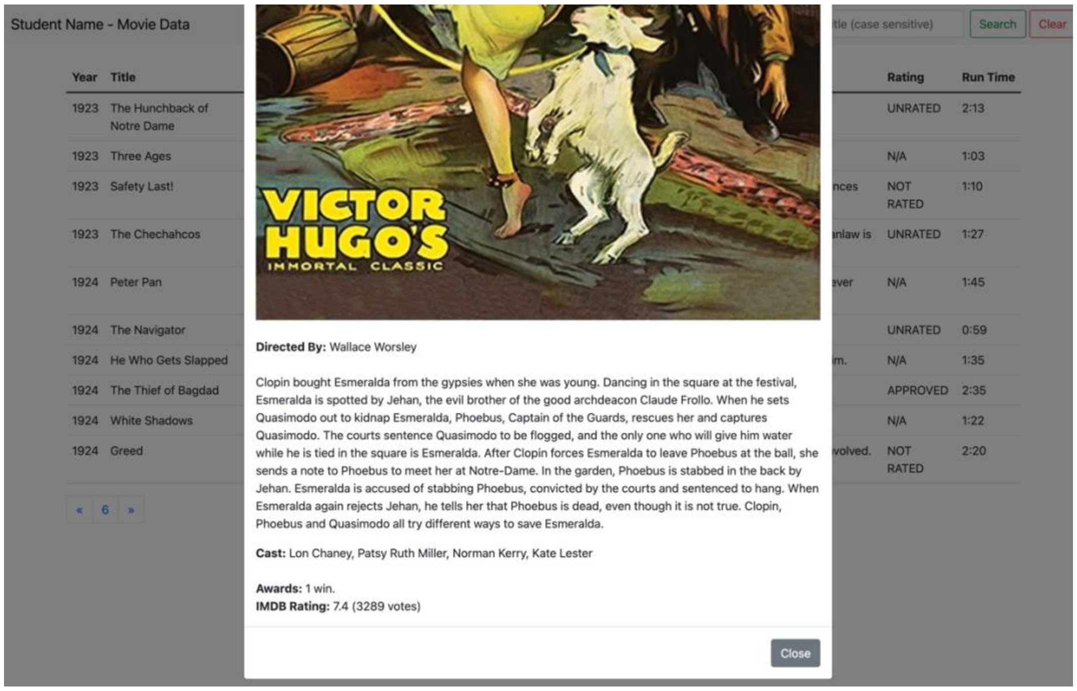
To work with our "Movies" API (from Assignment 1) on the client-side to produce a rich user interface for accessing data.

Sample Solution:

You can see a video of the solution running at the following location:
https://pat-crawford-sdds.netlify.app/shared/winter-2023/web422/A2/A2.mp4

Specification:

For this assignment, we will create a single, searchable table that shows a subset of the movie data (ie: columns: **Year, Title, Plot, Rating and Run Time**). When the user clicks on a specific movie (row) in the table, they will be shown a modal window that shows the movie “poster” (if available) as well as additional information such as the director, full plot, cast, awards and IMDB rating.



Student Name - Movie Data

Year	Title		Rating	Run Time
1923	The Hunchback of Notre Dame		UNRATED	2:13
1923	Three Ages		N/A	1:03
1923	Safety Last!	nces	NOT RATED	1:10
1923	The Chechahcos	nlaw is	UNRATED	1:27
1924	Peter Pan	ever	N/A	1:45
1924	The Navigator		UNRATED	0:59
1924	He Who Gets Slapped	im.	N/A	1:35
1924	The Thief of Bagdad		APPROVED	2:35
1924	White Shadows		N/A	1:22
1924	Greed	volved.	NOT RATED	2:20

Directed By: Wallace Worsley

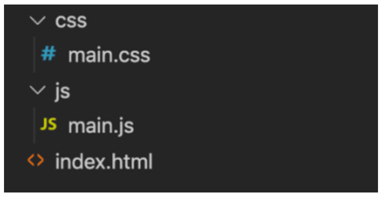
Clopin bought Esmeralda from the gypsies when she was young. Dancing in the square at the festival, Esmeralda is spotted by Jehan, the evil brother of the good archdeacon Claude Frollo. When he sets Quasimodo out to kidnap Esmeralda, Phoebus, Captain of the Guards, rescues her and captures Quasimodo. The courts sentence Quasimodo to be flogged, and the only one who will give him water while he is tied in the square is Esmeralda. After Clopin forces Esmeralda to leave Phoebus at the ball, she sends a note to Phoebus to meet her at Notre-Dame. In the garden, Phoebus is stabbed in the back by Jehan. Esmeralda is accused of stabbing Phoebus, convicted by the courts and sentenced to hang. When Esmeralda again rejects Jehan, he tells her that Phoebus is dead, even though it is not true. Clopin, Phoebus and Quasimodo all try different ways to save Esmeralda.

Cast: Lon Chaney, Patsy Ruth Miller, Norman Kerry, Kate Lester

Awards: 1 win.
IMDB Rating: 7.4 (3289 votes)

The Solution Directory

The first step is to create a new directory for your solution, open it in Visual Studio Code and add following folders / files:



```
css
  main.css
js
  main.js
index.html
```

We will not be including any of the JavaScript / CSS libraries locally. Instead, we will be leveraging their CDN locations (See the Bootstrap Introduction notes for the <script> and <link> elements necessary).

NOTE: While the below instructions provide guidelines on how to use Bootstrap, you are not forced to use it if you prefer to use another UI / CSS framework. As long as your solution functions according to the below specification, it will still be accepted.

Creating the Static HTML:

Next, we must create some Static HTML as a framework for the dynamic content.

Open your index.html file and add the minimum code required for an HTML5 page (HINT: type **!** and then immediately type the **tab** key to get an HTML 5 skeleton). Once this is complete, include links for:

- The Bootstrap Minified CSS File (Using the CDN)
- Your **main.css** file (**NOTE**: This file will only consist of one selector (for now) to ensure that your "moviesTable" (or whatever you wish to call it) causes the cursor to change to a "pointer" whenever a user moves their mouse over a row, ie:

  ```
  #moviesTable tr:hover{ cursor:pointer; }
  ```

- The Bootstrap Minified JS File (Using the CDN)
- Your **main.js** file

With all of our libraries and files in place, we can concentrate on placing the static HTML content on the page. This includes the following:

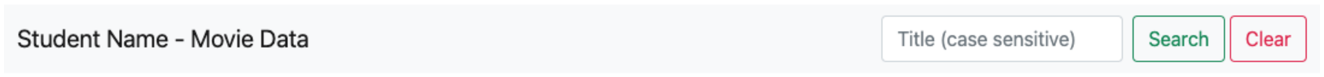
Navbar

Assignment 2 will use an extremely simplified navbar. Begin by copying the full **Default Navbar** example HTML code from the official documentation: https://getbootstrap.com/docs/5.1/components/navbar and pasting it as the first element within the <body> of your file.

- Next, proceed to remove the <ul>…</ul> element from the <div> element with id: “navbarSupportedContent”

- This should leave you with just the search form. To match the sample, make the following changes:
  - Add the class "justify-content-end" to the <div> element with id: "navbarSupportedContent"
  - Give the form an "id" value, ie: "searchForm"
  - Update the placeholder in the search field to read: "Title (case sensitive)" and give it an "id" value, ie: "title"
  - Add a 2nd button (type: "button") beside "Search" with the text "Clear" and give it an "id" value, ie: "clearForm"
- Finally, change the "navbar-brand" to include your name

When completed, your navbar should look like the following



Bootstrap Grid System (1 Column)

Since we are leveraging Bootstrap for this assignment, we should make use of their excellent responsive grid system. Beneath the navbar, add the following HTML

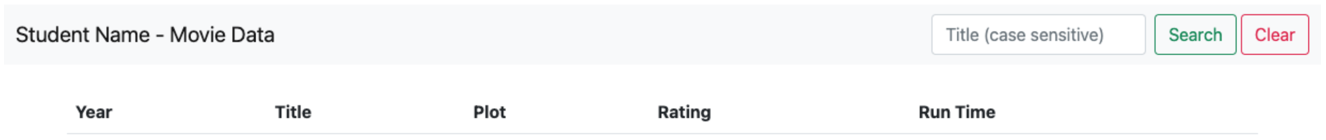
- Include a <div> element with the class "container" (so that our content is centered)
- Within the "container", create a <div> with class "row"
- Within the "row", create a <div> with class "col" (we will only have one column to show our data)

Main Table Skeleton

The main interface that users will interact with to view data in our application is a HTML table consisting of **5 columns: Year, Title, Plot, Rating** and **Run Time**. Create this table within your "col" div according to the following specification:

- The <table> element should (at a minimum) have the class "table" and a unique id, ie: "moviesTable", since we will be accessing it programmatically from JavaScript
- The <thead> element should contain one row
- The single header row should have 5 table heading elements with the text:
  - Year
  - Title
  - Plot
  - Rating
  - Run Time
- The <tbody> element should be empty

Once your table is in place, your app should look like the following:



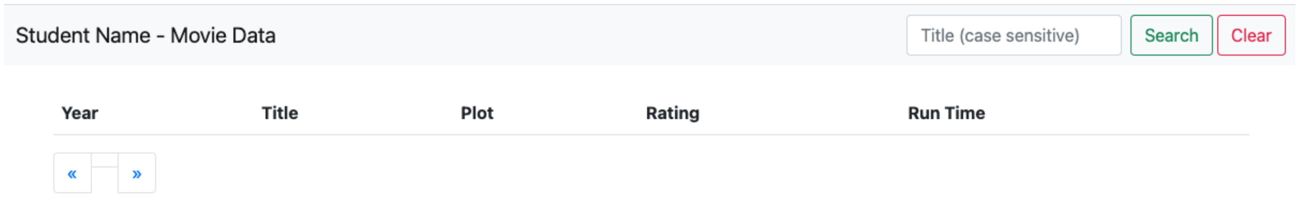
Paging Control

Since our "movies" collection contains 23530 documents, we will leverage our web API's pagination feature when pulling movies from the database (ie: **/api/movies?page=1&perPage=10**, etc). To give the user some control over which page they wish to see, we must include a primitive pagination control (for this assignment, we will not let them "jump" to a specific page, but instead we will let them go back and forth between the pages in sequence).

To accomplish this, we must place the pagination buttons on our page before wiring up their functionality:

- Begin by copying the full **Pagination** HTML code from the official documentation: https://getbootstrap.com/docs/5.1/components/pagination and pasting it somewhere underneath your newly created "moviesTable".
- Next, delete the list items that contain the numbers **2**, and **3** (leaving just **1**)
- Give each of the 3 remaining <a> elements (nested within the <li> elements) unique id values such as "previous-page", "current-page" and "next-page" (we will use these id values to add functionality to the links and display the current page)
- Finally, remove the text **1** from the middle link (it will be added dynamically later).

Once your pagination control is in place, your app skeleton should look like the following:



"Generic" Modal Window Container

We will be showing the poster & details for a specific movie in a Bootstrap modal window. Since every time we show the modal window, it will have different content (Specific to the movie that was clicked), we must add an empty, **generic** modal window to the bottom of our page.

To get the correct HTML to use for your Bootstrap modal window, use the following example from the notes as a starting point.

Once you have copied and pasted the "modal" HTML into the bottom of your <body> element (ie, below the other content) , make the following changes:

- Give your <div> with the class "**modal fade**" a unique **id**, ie: "**detailsModal**". We will need to reference this element every time we wish to show / work with the **modal window.**

- Remove the "Modal title" text from the <h5> element with class "**modal-title**". We will be using JavaScript to populate this with the selected movie title.

- Remove the <p> element with the text "Modal body text goes here." from the <div> element with class "**modal-body**". Again, we will be using JavaScript to populate this element

- Finally, remove the button element with the text "Save changes". This modal is used to display information only, so a "save" button is not required.

NOTE: If you wish to use a larger modal window, the class "modal-lg" can be added beside the "modal-dialog" class.

JavaScript File (main.js):

Now that we have all of our static HTML / CSS in place, we can start dynamically adding content and responding both user and bootstrap events using JavaScript. In your **main.js** file add the following variables & functions at the top of the file:

- **page** (number)

  This will keep track of the current page that the user is viewing. Set it to **1** as the default value

- **perPage** (number)

  This will be a constant value that we will use to reference how many movie items we wish to view on each page of our application. For this assignment, we will set it to **10**.

- **loadMovieData(title = null)** (function)

  This is arguably the most complex function to be created for this assignment. Its job is to pull movie data from your published movies API (created in A1), correctly format the data / add it to the DOM, add "click" events to each row item and populate / show a modal window with the correct data when the event fires.

  NOTE: It is not mandatory to do so, but you may wish to split the following logic up into multiple functions.

Loading The Data

Depending on whether the value of "title" is null or not, this function must make a "**fetch**" request to your published API (created in A1) using the **page** and **perPage** values defined (above) to obtain the data.

For example, if the **title** parameter is *not* null, then you must make the following request for data:

/api/movies?page=**page**&**perPage**=perPage&title=**title**

Additionally (since the title parameter is not null), you must set the **page** value to **1** and *hide* the **pagination** control – this can be done by adding the class "d-none" to the element with class "pagination". **HINT:** Element.classlist can be used here: https://developer.mozilla.org/en-US/docs/Web/API/Element/classList

If the **title** parameter is null, then you must make a similar request for data, only omitting the "title" query parameter. Also, you must *show* the **pagination** control by removing the class "d-none" from the element with class "pagination"

Creating the <tr> Elements

Once we have obtained the data from the API, we must transform it into something that we can add to the DOM (specifically the <body> element of our moviesTable). This can be done by utilizing the Array "map" method within a Template literal (**HINT:** Review "Generating HTML" section of the notes for more information).

For guidance on how the data must be transformed, consider the following film: "The Matrix" (ie: /api/movies/573a139bf29313caabcf3d23). Reference the JSON output from your API at this path and note the generated HTML below: (**NOTE:** each element in the array of returned results must use the this format for rows in the table)

```
<tr data-id="573a139bf29313caabcf3d23">
        <td>1999</td>
        <td>The Matrix</td>
        <td>A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.</td>
        <td>R</td>
        <td>2:16</td>
</tr>
```

The HTML shown above can be obtained using the following properties from the returned JSON data:

- **573a139bf29313caabcf3d23** – obtained from the **_id** property
- **1999** – obtained from the **year** property
- **The Matrix** – obtained from the **title** property

- **A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.** – obtained from the **plot** property (**NOTE:** There are some movies in the dataset without a plot. If this is the case, show **N/A** instead)

- **R** – obtained using **rated** property (**NOTE:** As above, there are some movies in the dataset without a rated value. If this I the case, show **N/A** instead)

- **2:16** – obtained using the **runtime** property, where the number of hours (**2**) was calculated using the formula: **Math.floor(runtime / 60)** and the number of minutes (**16**) was calculated using the formula: **(runtime % 60).toString().padStart(2, '0')**

__Adding <tr> Elements to the Table__

Using **document.querySelector**, obtain the <tbody> element of your moviesTable and update it to show your newly created <tr> elements.

__Updating the "Current Page"__

Using **document.querySelector**, obtain the element with id "current-page" and update it to show the value of the global **page** value (from above)

__Adding Click Events & Loading / Displaying Movie Data__

With the rows in place, we can once again use **document.querySelector** to obtain all of newly created <tr> elements added to the table (above). Once you have selected them, loop through the array and add a "click" event to each element, in order to execute the following logic:

- Obtain the value of the "data-id" attribute of the clicked element

- Use this value to make a request for data using the path: /api/movies/**data-id**

- Set the "modal-title" of your "detailsModal" to show the value of the **title** property from the returned data

- Set the "modal-body" of your "detailsModal" to show the data using the following format. For example, once again using the data from "The Matrix" (ie: /api/movies/573a139bf29313caabcf3d23), the following HTML must be generated

  <img class="img-fluid w-100" src="**https://m.media-amazon.com/images/M/MV5BNzQzOTk3OTAtNDQ0Zi00ZTVkLWI0MTEtMDllZjNkYzNjNTc4L2ltYWdlXkEyXkFqcGdeQXVyNjU0OTQ0OTY@._V1_SY1000_SX677_AL_.jpg**"><br><br>
  <strong>Directed By:</strong> **Andy Wachowski, Lana Wachowski**<br><br>
  <p>**Thomas A. Anderson is a man living two lives.**</p>
  <strong>Cast:</strong> **Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo Weaving**<br><br>
  <strong>Awards:</strong> **Won 4 Oscars. Another 33 wins &amp; 40 nominations.**<br>
  <strong>IMDB Rating:</strong> **8.7** (**1080566** votes)

The HTML shown above can be obtained using the following properties from the returned JSON data:

- **https://m.media-amazon.com/images/M/MV5BNzQzOTk3OTAtNDQ0Zi00ZTVkLWI0MTEtMDllZjNkYzNjNTc4L2ltYWdlXkEyXkFqcGdeQXVyNjU0OTQ0OTY@._V1_SY1000_SX677_AL_.jpg** – obtained from the **poster** property

- **Andy Wachowski, Lana Wachowski** – obtained from the **directors** property (Array displayed using the "join" method with a value of ', ')

- **Thomas A. Anderson is a man living two lives.** – (**NOTE:** this is a shortened version of the full value from the API, shown here to save space) obtained from the **fullplot** property

- **Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo Weaving** – obtained from the **cast** property (Array displayed using the "join" method with a value of ', '). (**NOTE:** There are some movies in the dataset without a cast. If this is the case, show **N/A** instead)

- **Won 4 Oscars. Another 33 wins &amp; 40 nominations.** – obtained from the **awards.text** property

- **8.7** – obtained from the **imdb.rating** property

- **1080566** – obtained from the **imdb.votes** property

▪ Using **document.querySelector**, obtain the <div> element of your "detailsModal" with class "modal-body" and update it to show your newly created elements (from above).

▪ Once this is complete, show the modal (**HINT:** Refer to the code outlined in the notes under "Modal Windows" for more information).

The remainder of the code within main.js **must** be executed when the "DOM is ready", ie: once the "DOMContentLoaded" event has fired for the "document" object:

- **Click** event for the "previous page" pagination button:

  When this event is triggered we simply need to check if the current value of **page** (declared at the top of the file) is greater than **1**. If it is, then we decrease the value of **page** by 1 and invoke the **loadMovieData** function (without the "title" parameter) to refresh the rows in the table with the new page value.

- **Click** event for the "next page" pagination button:

  This event behaves almost exactly like the click event for the "previous page", except that instead of *decreasing* the value of page, we **increase** the value of **page** by 1 and invoke the **loadMovieData** function (without the "title" parameter) to refresh the rows in the table with the new page value.

- **Submit** event for the "searchForm" form:

  This event simply prevents the default submit from occurring for the form and invokes the loadMovieData function with the value of the "title" field (from the search form).

- **Click** event for the "clearForm" button:

  This event resets the value of the "title" field (from the search form) to an empty string ("") and invokes the loadMovieData function without any parameters

Assignment Submission:

- Add the following declaration at the top of your main.js file

```
/*********************************************************************************
*  WEB422 – Assignment 2
*  I declare that this assignment is my own work in accordance with Seneca Academic Policy.
*  No part of this assignment has been copied manually or electronically from any other source
*  (including web sites) or distributed to other students.
*
*  Name: ______________________ Student ID: __________________ Date: ____________________
*
********************************************************************************/
```

- Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code).

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.