COMP 3512
Assignment #1: Single-Page App *(version 1.2)*

Due Saturday February 29, 2020 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a dynamically updateable single page web application using JavaScript. You will be creating something similar to the final exercise from your JavaScript 3 lab. That lab made use of a small subset of data in a small JSON file that contained all the information you needed. This lab uses a more realistic API.

Submitting

For this assignment, you will simply submit your source files to the regular submit drive (I: drive). Name your submit folder as follows: *`username_assign1`*, where *username* is your actual user name. This folder should contain your content.

Grading

The grade for this assignment will be broken down as follows:

Visual Design	23%
Programming Design	12%
Functionality (follows requirements)	65%

Data Files

Data will be provided in an external API.

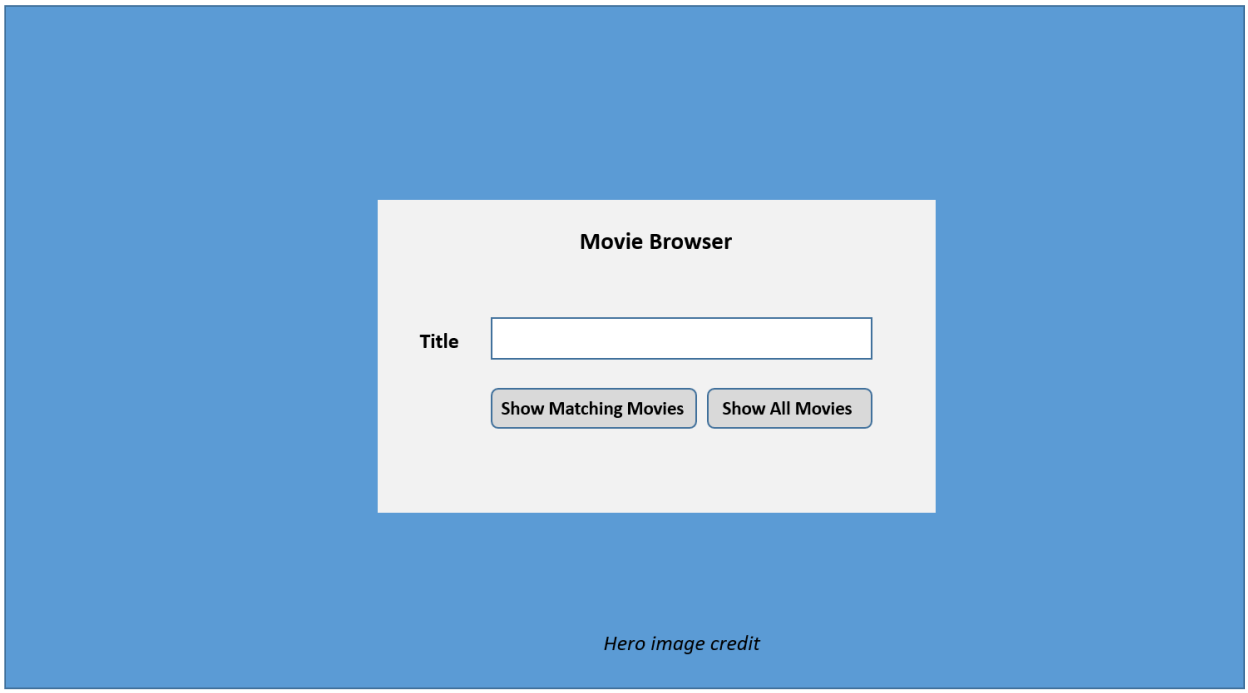
Requirements

This assignment consists of a single HTML page (hence single-page application or SPA) with the following functionality:

1. Your assignment **must** have **just** a single HTML page which **must** be named `index.html`.
2. I expect your page will have significantly more additional CSS styling compared to `lab10-test02.html`. If you make use of CSS recipes you found online, you must provide references (i.e., specify the URL where you found it) via comments within your CSS file. **Failure to properly credit other people's work in your CSS will likely result in a zero grade.**
3. Your layout should work at desktop size and at mobile size. You can test this yourself simply by resizing your browser window to a small size to simulate a mobile browser width. You can also use the Audit feature in Chrome.

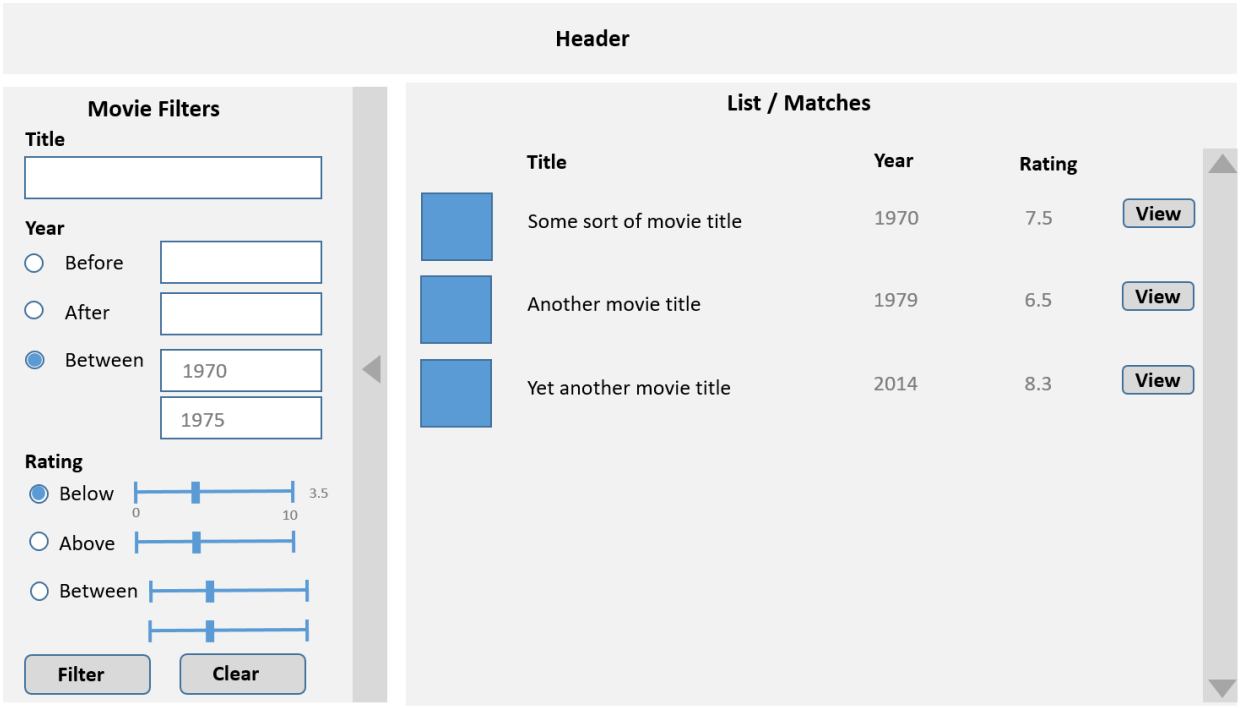
4. You must write your own JavaScript. That is, no jQuery, no React, no other third-party JavaScript libraries. You have all the knowledge you need from the three JavaScript labs to complete this assignment. If you do find yourself, however, making use of some small snippet of code you found online (say more than 4-5 lines), then you must provide references (i.e., specify the URL where you found it) via comments within your code. **Failure to properly credit other people's work in your JavaScript will likely result in a zero grade.** There is no need to credit code you found in the lab exercises.
5. Most of the functionality in the app can be found in the three sketches shown on the next few pages. Each of these is described in more details below. The sketches provided illustrate a sample layout. You can completely change the layout. The first shown below is the **Home View**.

Home View



6. When your assignment is first viewed, it should display the **Home View**. It consists of a hero image (and image that fills the entire browser width or up to a specific very wide value, say 1800 or 2200 pixels). You could use unsplash.com as a source for your image, but be sure to provide credit information somewhere on this page. In the middle of the page should be another rectangle with a place where the user can enter a title search string, as well as two buttons. Both will take the user to the Default View. If the first is clicked, then the movie list will be filtered on the movie title; if the second is clicked all the movies will be displayed.

Default View



7. **Header**. The page title should be COMP 3512 Assign1. The subtitle should be your name.
8. **Movie List / Matches**. Displays a list of movies. The URL for the API is:

   `http://www.randyconnolly.com/funwebdev/3rd/api/movie/movies-brief.php?id=ALL`

   The movie list should initially be sorted alphabetically on title. Initially, display all movies or filtered by name depending on what happened on the Home View. The Title, Year, and Rating column headings should be clickable: when clicked they sort the movies by that field. The blue boxes represent small poster images (see below for more info).

   The API will take some time to retrieve this data. Display a loading animation (there are many free animated GIF available) until the data is retrieved. Simply display the image just before the fetch and then hide it after fetch is successful.

   Clicking on the View button, the title, or the poster image will take the user to the **Movie Details** view. Be sure to set the mouse cursor to indicate they are clickable. Your click handler here for the movie list **must** use event delegation!

   To improve the performance of your assignment, you must store the content retrieved from the movies-brief API in local storage after you fetch it. Your page should thus check if movie data from this API is already saved in local storage: if it is then use it, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating an early fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in for instance Chrome via the Application tab in DevTools).

9. The movie poster images can be found in different sizes via the image server for tmdb.org (`https://image.tmdb.org/t/p/`). One of the data fields for each movie is `poster`, which contains the filename for the poster. You can then specify the image width you want by adding either `w92`, `w154`, `w185`, `w342`, `w500`, or `w780` to the URL. For instance, for the movie with id=1144, the poster field value is `"/wOBKAoUJZb5qTsWv5XXvVV2vUzz.jpg"`. Thus to get a small thumbnail image (width = 92 pixels) of the poster, then the URL would be:

   ```
   https://image.tmdb.org/t/p/w92/wOBKAoUJZb5qTsWv5XXvVV2vUzz.jpg
   ```

10. **Movie Filters**. Allow the user to easily filter the list of movies. User should be able to find movies whose title contains anywhere within it whatever was entered into the title input box. The user should be able to filter the movie list by the release date year. They should be able to specify either: movies before a year, after a year, or between two years. Similarly the user should be able to filter the movie list by the average rating by also specifying a below, above, or between value. Also provide way to remove filters (that is, return to all movies being displayed). When the user clicks the filter button, the movie list should update.

    If no matching movies are found, notify the user within the Movie List/Matches area.

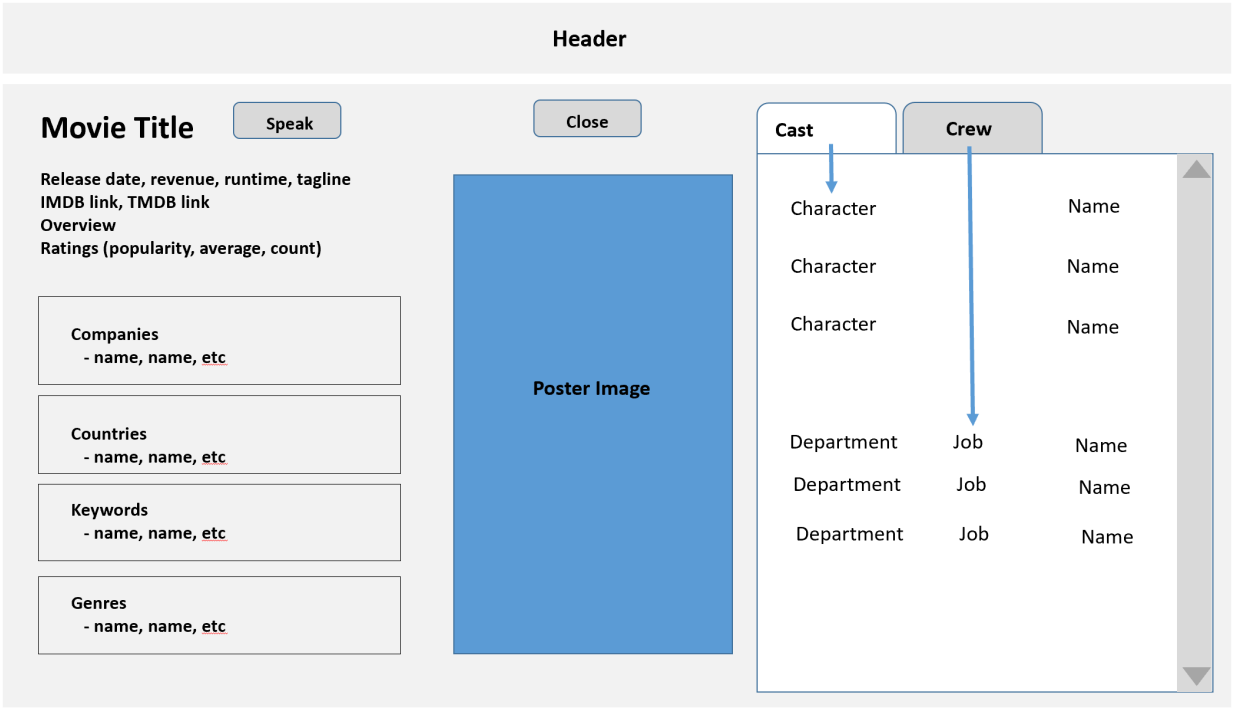
    The list of movies should scroll while the rest of the page stays. You can do this easily with the CSS `overflow` property.

    Assume these filters are mutually exclusive (only one can be active), though for extra credit you can make the filters not mutually exclusive.

    The user should be able to toggle the visibility of the filters panel (though initially should be visible). Don't be scared of using icons instead of text. There must be some type of transition effect (e.g., animation or fade), rather than just instantaneous visible/invisible.

    A lot of students will lose marks because they didn't read the instructions for steps 8 and 10 carefully … don't let that be you!

Movie Details View



11. **Movie Details**. Displays detailed information for the selected movie. The movie details can be retrieved from another API whose URL is (where `xxxx` is the id of the movie):

    `http://www.randyconnolly.com/funwebdev/3rd/api/movie/movies.php?id=XXXX`

    I would recommend running a test query in the browser with an ID of 1144 so that you can see the data and its structure. Note: do **not** store the results from this API in localstorage.

    The API will take some time to retrieve this data. Display a loading animation until the data is retrieved. Simply display the image just before the fetch and then hide it after fetch is successful.

    I expect this data to be nicely formatted and laid out sensibly. Every year students lose easy marks because they put no effort into the layout here. I have put the info here in three columns just to make it fit in Word. You can construct your layout anyway you'd like.

    Working IMDB and TMDB links can be constructed from their related ID fields (e.g. `https://www.themoviedb.org/movie/xxxx` and `https://www.imdb.com/title/yyyy`, where `xxxx` is the `tmdb_id` field and `yyyy` is the `imdb_id` field)

    The Speak button will use the browser's voice API to speak the movie title. The Close button will return the user back to the **Default View**.

12. **Cast and Crew**. In the sketch above, Cast and Crew are shown as tabs, though you could use hide / show boxes as well. The Cast tab (it should be showing by default) will show the character name and the actor's names. Sort the cast members by the order field. For the Crew tab, show the department, job, and name fields. Sort by department and then by name.
13. **Poster Image**. Display the poster using either w185 or w342 size. When the user clicks on the poster, display a larger version (w500 or w780) as a pop-up modal window. This is sometimes referred to as a lightbox effect. The easiest approach is to simply turn on the visibility of the larger image's div when smaller poster is clicked; when the larger image is clicked, then hide it. This approach makes an image request even if the user doesn't want to see it; a more efficient approach for a real site would dynamically construct the image and div only when the user clicks it. For this assignment, take the easiest approach.

Hints

1. Test the APIs out first in the browser. Simply copy and paste the URLs into the browser address bar. The JSON can be hard to understand because it doesn't have white space, so use an online JSON formatter (such as https://jsonformatter.curiousconcept.com) via copy and paste to see the JSON structure in an easier to understand format (or add a JSON formatter extension to your browser).
2. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `Id` and it won't work because the JSON field is actually `id`.
3. Your HTML will include the markup for all three views (**Home**, **Default,** and **Movie Details)**. Initially, the latter two will initially use CSS to set its `display` to `none`. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for the two views.
4. Most of your visual design mark will be determined by how much effort you took to make the two views look well designed. Simply throwing up the data with basic formatting will earn you minimal marks for this component. Look at IMDB and TMDB for inspiration: notice how they use contrast (size, weight, color) and icons.
5. Most of your programming design mark will be based on my assessment of your code using typical code review criteria. For instance, did you modularize your code using functions? Are your functions too long? Is the code documented? Do you have code duplication (you shouldn't)? Did you make use of object-oriented techniques? Are variables and functions sensibly named? Is your code inefficient (e.g., fetching the same data repeatedly)? Are you using outdated JavaScript techniques (e.g., inline coding, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, etc?

6. When constructing single-page applications in JavaScript, you may need to "insert" data into dynamic HTML elements that you modify/create in JavaScript. In HTML5, this is supported via the `data-X` attribute.

   For instance, in this assignment, you may need a way to determine the identifier of the movie that was clicked on in the list of movies. You can do this by using the `setAttribute()` method in JavaScript to set, for instance, an attribute named `data-id` whose value is the `id` field when dynamically generating the list of movies. Then, in the click event handler for each movie in the list, you can determine the unique identifier for the movie that was clicked (and thus later retrieve the movie object for that id) by using the `getAttribute()` method.

7. For the Speak buttons and the Close button, be sure to only add click event handlers for these buttons only once when the page loads (and not every time you display a movie's details).