

COMP 3512

Assignment #1: Single-Page App

Due Wednesday, March 2, 2022, at 9:00 PM

NEW: Comment from JP

I realize it's a bit weird – and I daresay confusing – to have both THIS document AND the milestone documents. The reason I have kept these original instructions (with some modifications) is because they give YOU and ME a big-picture view of the task at hand. That's important.

But the milestone instructions are also important – they break up the big picture into smaller, explicit requirements...things you can check off and say “yep, did that”. That's important, too.

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a dynamically updateable single-page web application using JavaScript.

NEW: Modifications to the Application

I've decided to tweak the behaviour of the application a bit. Here is a summary of what has changed from the original version:

1. Movie information will no longer be taken from the original assignment's API (<http://www.randyconnolly.com/funwebdev/3rd/api/movie>). Instead, it will be taken directly from the TMDB API.
2. The title search on the Home View has an autocomplete feature.
3. Users now have the ability to favourite a movie.
4. The Home View's “show all movies” feature has been replaced with “show favourite movies”.
5. Instead of having a subset of 666 movies (accessed through the original API) used all the time and stored off in local storage, all movies available through the TMDB are available and only *favourited* movies will be stored in local storage.
6. User's can now *favourite* and *unfavourite* a movie in the Default View.
7. Results of the title searches on the Home View are stored in session storage.
8. There are opportunities to go from Excellent (an A) to Excellent+ (an A+) in the 3 broad marking categories in the final milestone. See the last page for further details.

For the remainder of this document, **text in orange** denotes modification that have been made since this document was last provided for milestone 3 and should be read carefully.

Requirements

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

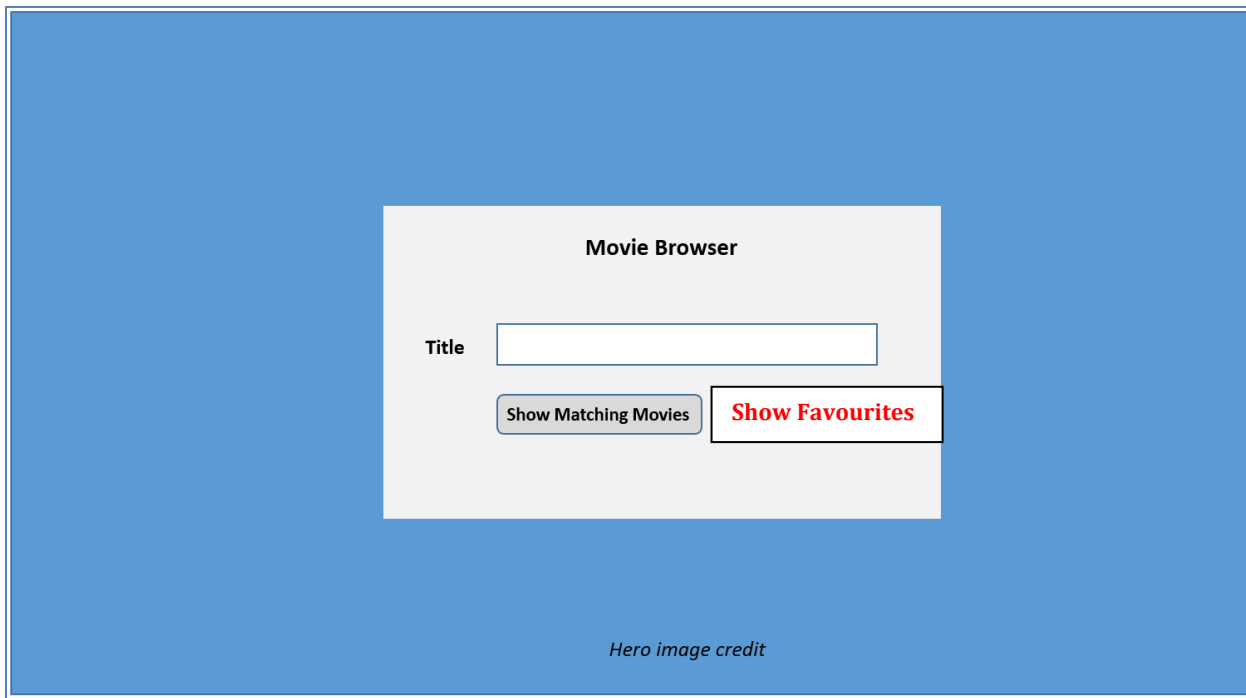
1. Your assignment **must** have **just** a single HTML page named `index.html`.
2. You can either use a non-Bootstrap CSS framework (like Tailwind CSS) or your own CSS. If you make use of CSS recipes you found online, you must provide references (i.e., specify the URL where you found it) via comments within your CSS file. **Failure to properly credit other people's work in your CSS will likely result in a zero grade.** In any case, I expect your final site to look portfolio-worthy.
3. Your layout should work in Chrome's device toolbar at desktop L size and at mobile L size.
4. You must write your own JavaScript. That is, no jQuery, no React, no other third-party JavaScript libraries – and this also means that if you're using a CSS framework, you cannot use any JS features found there. You have all the knowledge you need from the course content 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 detail below. The sketches provided illustrate **sample** layouts. You can – **AND OFTEN SHOULD** – change various parts of these layouts if you wish!

The first sketch shown below is for 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. *The credit should be a working link – but don't put in a raw URL, 'cause that looks pretty crappy. The credit shouldn't be in your face, because it's not meant to be the primary focus of the page - but it should be somewhere visible, because you want to do the Right Thing.*

In the middle of the page should be another rectangle with a place where the user can enter a title search string.

7. **Autocomplete.** When the user types in the title search box, from the third letter onward, autocomplete suggestions will be made. These suggestions will come from results coming back from the TMDB API.
8. There should be an obvious way (not necessarily a button) to initiate a search for a title that's been entered in the search box. When this happens, the user is shown the Default View, where a list of movies whose title contains what is in the search box (case-insensitive) are present. This means that the search term from the Home View's search box should be in the title search box of the Default View when it appears.

There should also be an obvious way (not necessarily a button) to view all the movies the user has favoured. When this happens, the user is also shown the Default View, where a list of favoured movies are present.

9. **Session Storage.** If a title search occurs, all matching movie objects will need to be created and saved in session storage, as they will be needed for both the Default and Detail Views.




Default View

The screenshot shows a web interface with a header bar. Below the header, there are two main sections: 'Movie Filters' on the left and 'List / Matches' on the right.

Movie Filters:

- Title:** A text input field.
- Year:** Radio buttons for 'Before', 'After', and 'Between'. The 'Between' option is selected, with input fields for '1970' and '1975'.
- Rating:** Radio buttons for 'Below', 'Above', and 'Between'. The 'Below' option is selected, with a slider ranging from 0 to 10, currently set at 3.5.
- Buttons:** 'Filter' and 'Clear' buttons at the bottom.

List / Matches:

Title	Year	Rating	
 Some sort of movie title	1970	7.5	<button>View</button>
 Another movie title	1979	6.5	<button>Fave</button>
 Yet another movie title	2014	8.3	<button>Fave</button>

10. **Header.** The page title should be COMP 3512 Assignment1. Please do not put any other identifying marks here, as I might want to show examples of student work in lecture and wish to provide the site creators some privacy.

11. **Movie List / Matches.** Displays a list of movies based on what happened happened on the Home View.

The movie list should initially be sorted alphabetically on title. The Title, Year, and Rating column headings should clearly indicate that they will initiate a sort of that field and that both ascending and descending sorting is available.

The blue boxes represent small poster images (see below for more info).

If this page has been shown because of a title search, accessing the matching movies via the API will take some time. 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. This loading animation will need to be done for milestone 6.

Clicking on 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! Note that the View button has been changed to a Favourite feature. See the Favourites section below.

12. 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_path, 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 "/qzm3nKhyc9DEiJJ2EBccIHpYM3W.jpg". Thus to get a small thumbnail image (width = 92 pixels) of the poster, the URL would be:

<https://image.tmdb.org/t/p/w92/qzm3nKhyc9DEiJJ2EBccIHpYM3W.jpg>

13. **Movie Filters.** Allow the user to easily filter the list of movies. Users should be able to find movies whose title contains any part of 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 a way to remove filters (*that is, reset them to what they were when the Default View was shown*). 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. *You may also make this an “infinite scroll” for an Excellent+ Functionality mark. See the end of this document for further information on this opportunity.*

Assume these filters are mutually exclusive (only one can be active). *For an Excellent+ result in your Functionality mark, you can make the filters not mutually exclusive. See the end of this document for further information on this opportunity.*

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!

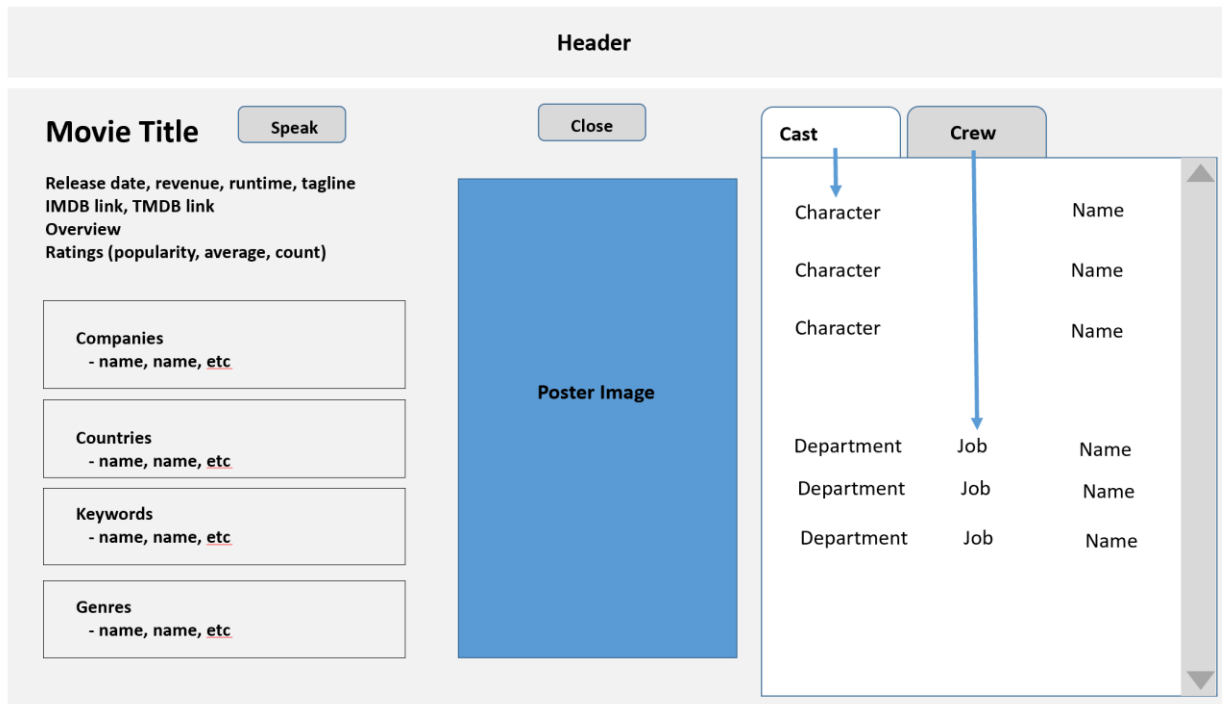
14. **Favourites.** The user should be able to easily favourite a movie. When a movie is favourited, there must be some visual cue that the movie *has been* favourited.

Favourited movies must be stored in local storage.

A user should naturally also be able to easily unfavourite a movie. When a movie is unfavourited, there must be some visual cue that the movie is no longer favourited.

Unfavourited movies must be removed from local storage.

Movie Details View



15. **Movie Details.** Displays detailed information for the selected movie. The movie information will need to be retrieved from session storage.

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 Web Speech API to speak the movie title. The Close button will return the user back to the **Default View**.

16. **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.
17. **Poster Image.** Display the poster using either w185 or w342 size. At Laptop L 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. 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.

6. 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).

Excellent+ Opportunities

If your team has reached the Excellent level in any of the 3 broad marking categories (functionality, visual design, and code craftsmanship), you may try to bump up your level in that category to an Excellent+.

If you're going to try to for an Excellent+, you need to let me know before submission of your assignment, so that I know to keep an eye out for it when I mark your work!

Here are your options:

Functionality

You can choose either (or both, though it won't get you any extra marks) of these functionalities:

1. **Infinite Scroll.** Instead of having a standard scroll view when many movies are being displayed in Default View, you can implement an infinite scroll instead. This must be done with vanilla JS that you create yourself – no libraries or plugins can be used. Of course, you can find code to help you with this on the web – and there's similar code in the labs! – so you must quote your sources as usual.
2. **Non-mutally-exclusive filters.** Allow users to combine filters so that all movies fulfilling all filters present are displayed.

Visual Design

Tablet layout. Your site looks professional at Mobile L, Laptop L, AND at Mobile size in Chrome.

Code Craftsmanship

ES6 Classes. Convert your Movie constructor function into an ES6-style class. You can find further information in section 10.2.2 in the textbook, or in MDN, etc.