COMP 4513 Assignment #1: React

Due Monday February 27th at midnight
Version 1.1 (Feb 7, 2023), changes in yellow

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a single-page application using React. **You can work alone or in groups of two on this assignment. Please don't ask for a group of three or four.** Let me know if you need a private github repo for your group.

Beginning

Use the same techniques covered in the second React lab for this assignment.

Submit

You must host your site on a working server platform that I can access. Since your React application is a static site, any host that supports static files will work. GitHub Pages, FireBase Hosting, Netlify, Render, and Surge, all provide relatively trouble-free solutions for hosting React apps. If DevOps is your thing, you could use Google Cloud Platform or AWS to create a virtual servers. Or you could use any third-party server that has ftp access.

You must upload the deploy version of create-react-app. The `npm run build` command will create a production build of your application in a folder named `build`. You can find instructions for deploying to numerous environments at https://create-react-app.dev/docs/deployment/.

I will also need access to the source code, so you will have to make it available to me on GitHub. **The development version, not the production version, must be uploaded to GitHub**! If on a private repo, be sure to invite me.

So, to submit your assignment you must send me an email with three bits of information: 1) the URL of your React app, 2) the github repo URL, 3) the names of the people in your group.

Grading

This assignment is worth 17% of the course grade. The grade for this assignment will be broken down as follows:

Visual Design, Styling, and Usability	20%
Programming Design + Documentation	10%
Functionality (follows requirements)	70%

Requirements

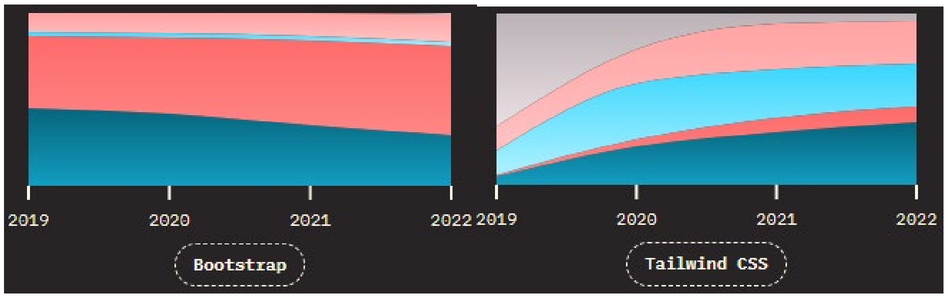
1. You must make use of the `create-react-app` starting files and structure. You must deploy a production build of the application to your live server; the development version must be on GitHub. The filename for your starting file should be `index.html` (`create-react-app` does this automatically).
2. This assignment consists of three main views (remember this is a single-page application, so it is best to think of the assignment consisting of different views). In the remainder of this assignment, I have provided a sketch of the basic layout of each view (some views have multiple subviews).

   These sketches do not show the precise styling (or even the required layout); rather they show functionality and content. I will be expecting your pages to look significantly more polished and attractive than these sketches! A lot of the 20% design mark will be based on my assessment of how much effort you made on the styling and design.

   You can change the layout if you wish. If you want your movie list to be horizontal on the bottom and the favorites to be vertical on the right, you can do so if you think that would be best … though I may not necessarily agree about the usability of your choices. I'd rather see you try something different in the layout/design than simply slap together something that is just a slightly improved version of the layout in my sketches.
3. If you've used JS that you've found online, I expect you to indicate that in your documentation. Provide a URL of where you found it in the documentation. Failure to do so might result in a zero mark, so give credit for other people's work!

4. You **must** use the Tailwind CSS library (but **not** one of the Tailwind React component libraries). Tailwind is a utility-first CSS framework that is especially well-suited to React. Rather than providing pre-built semantic components such as navigation bars and cards, with a utility-first framework you build up your page design by adding numerous "lower-level" utility classes to your markup. You can find an endless supply of Tailwind recipes online, and due to the nature of Tailwind, you can easily use them in your React components. Check out https://tailwindcomponents.com/ or the "CSS To Tailwind Chrome" extension. You might want to read the online documentation or watch a few youtube videos to give you a sense of how it works.

   There is a learning curve to Tailwind for sure, but if you are interested in web development, Tailwind CSS has absolutely become the CSS Framework of choice, esp. for React developers. These two charts illustrate developer opinions on the https://2022.stateofcss.com/ website. The blue bands indicate "Would Use again" and "Interested" while the orange bands indicate "Would not use again" or "Not Interested". Of the twelve CSS frameworks in the survey, all are flat or declining except for Tailwind, which is continually growing. I find it especially interesting that those who have used Tailwind, are the most keen to use it again. That's a pretty strong endorsement.



5. The assignment will be working with movie data provided by my API. To retrieve all the movies, you can use the URL:

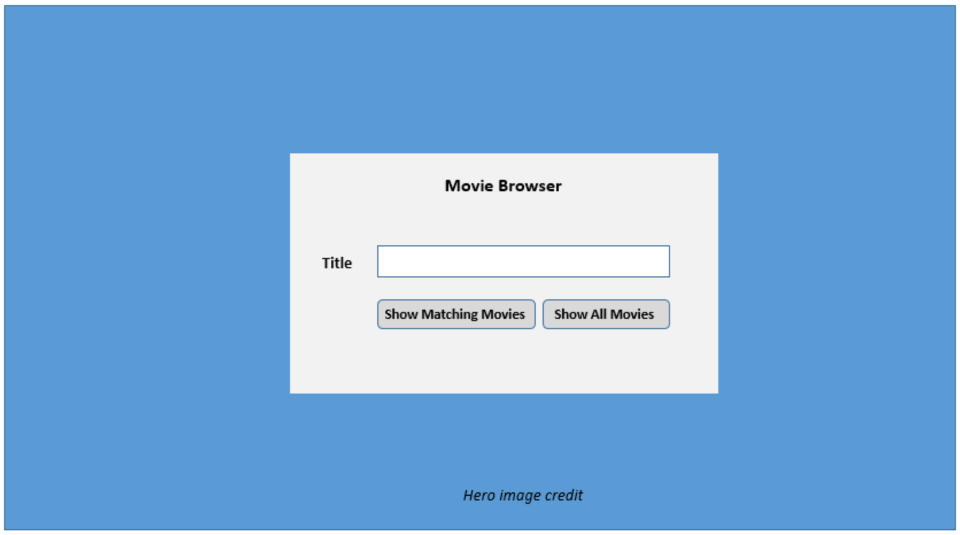
```
https://www.randyconnolly.com/funwebdev/3rd/api/movie/movies-brief.php?limit=200
```

   There are over 600 movies in the database, but only the first 200 will have the correct poster paths (in fact right now only the first 100 have the correct paths). If you want to improve the performance of your project while testing and debugging, try a smaller limit value (say 10-20). Just remember to change it back to 200 when submitting.

6. **Make sure you are only requesting/fetching the movie data once!** To improve the performance of your assignment (and reduce the number of requests on my server), you must store the movie data in `localstorage` after you fetch it from the API (see Exercise 10.11 in Lab 10). Notice that you have to use `JSON.stringify` to create a JSON string version of the array and then saving that in `localstorage`; similarly, after you retrieve it from `localstorage`, you will need to uses `JSON.parse()` to turn it into an array. Your page should thus check if this movie data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first 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 Chrome via the Application tab in DevTools). **Please do this task early on so that you reduce the load on my server!** Failure to implement this functionality will result in a large loss of marks so don't neglect it. **Last year, a student's fetching code in React got into a recursive loop (it's easy to do alas in React if you mess up the logic), which meant my server's monthly bandwidth allotment was quickly consumed and I got charged $100s of over-use charges. Please don't do this to me, so be sure to switch to localStorage ASAP!**

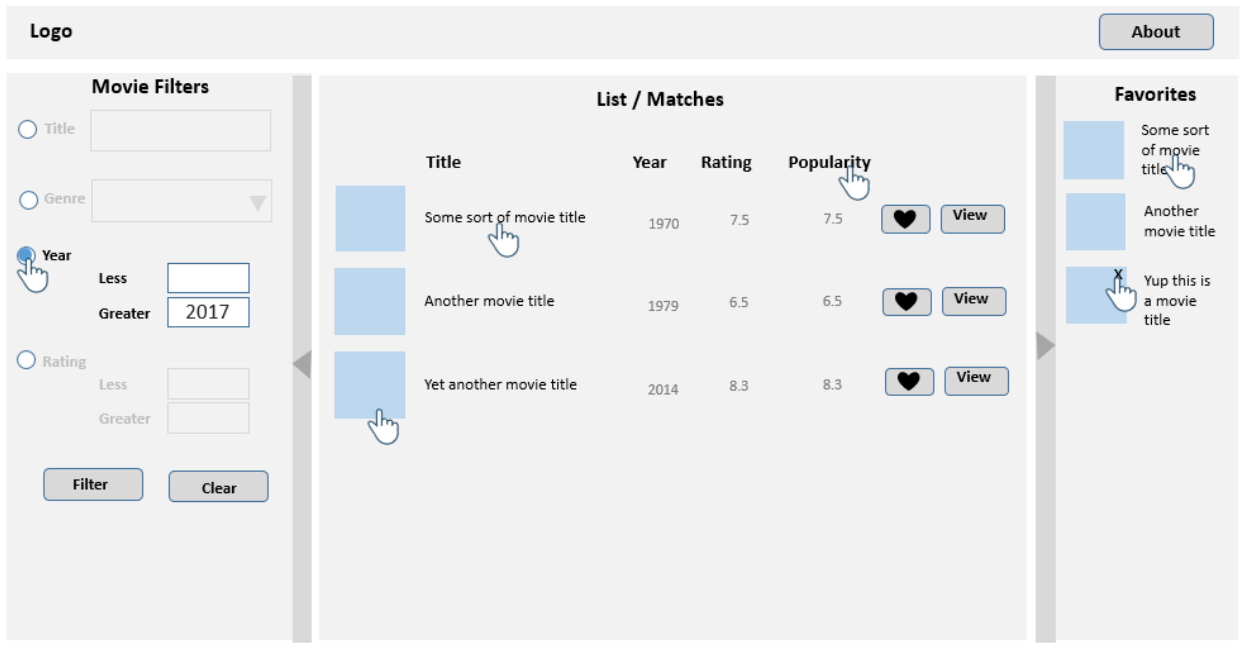
7. The API will take some time to retrieve this data the first time. Display a loading animation (there are many free animated GIF available) until the data is retrieved. Simply display the image just before the fetch and then hide it after fetch is successful.

Home View



8. 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



[View has been revised].This view should be broken down into a lot of different hierarchical components. Remember: the reason we are using React is that it allows us to break a complex application down into smaller components.

9. **Header**. The header should have a logo and a link/button to an About dialog. The logo should return to the Home View. Be sure to include a working link to the assignment's source code github repo.

10. **Movie List / Matches**. Displays a list of movies.

    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, Rating, and Popularity 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).

    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.

11. Clicking on the Heart button will add the movie to the Favorites list. When a movie is added to the favorites list, a small poster image for it should be added to the list (be sure to set both the title and alt attributes to the movie title). The Favorites list should be hide-able. If the user clicks on the small poster image in the favorites list, it should switch to **Movie Details** view for that movie. When hovering over the poster image in the favorites list, it should display some type of close/delete icon.

    When the user clicks on the Close icon, remove the photo from the list (and from state). The best way to implement this is to implement the Close icon as an image (or use just CSS transforms) and position in upper right. Use CSS :hover selector to change its opacity or visibility. If you want to always display the close button, that's fine but there should be some type of state change on hover (e.g., change button opacity, or color, or size).

    Clicking on the close button must remove the photo from favorites in state. Ideally, there will be some type of animation/transition on the Photo Thumb box to provide visual feedback that the photo is being deleted. Also, a movie can only be added once to favorites.

12. When the user selects the About link/button, it should display some type of modal dialog/window with information about assignment. Include group members, github link, technology used, any third-party source code, etc. Sometimes it makes sense to make use of existing components rather than re-invent the wheel. It is important to know how to integrate existing components. Thus here you **must** make use of an existing React modal-dialog component (e.g., react-modal or react-modal-dialog).

13. 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 tmdb_id=13, the poster field value is `"/wby9315QzVKdW9BonAefg8jGTTb.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/wby9315QzVKdW9BonAefg8jGTTb.jpg
    ```

    Unfortunately, tmdb changes the poster image filenames relatively frequently, so it is possible that some (or even many or all) of the poster filenames do not work. If you find that this is happening, you might want to instead use an image placeholder service, e.g.,

    ```
    https://via.placeholder.com/92
    ```

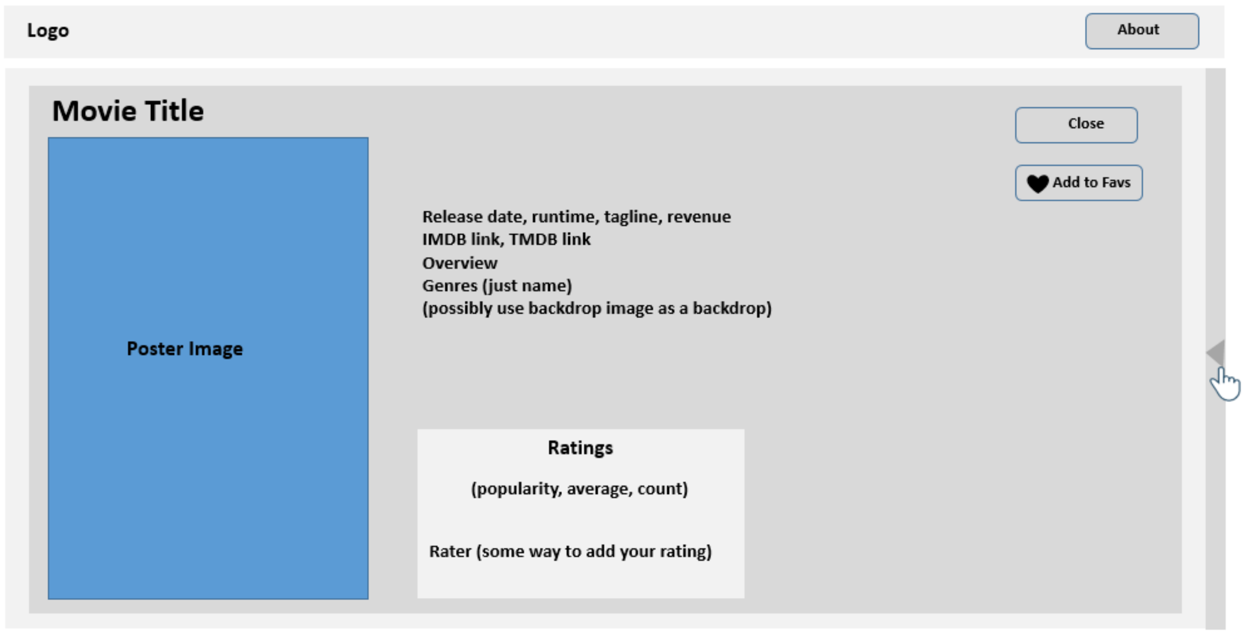
14. **Movie Filters**. Allow the user to easily filter the list of movies. User should be able to find movies whose title contains anywhere within it whatever was entered into the title input box. The user should be able to filter the movie list by the release date year, rating, and genre. Also provide way to remove filters (that is, return to all movies being displayed). When the user clicks the filter button, the movie list should update. These filters are mutually exclusive.

    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.

    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.

Movie Details View


Logo
About
Movie Title
Close
Add to Favs
Poster Image
Release date, runtime, tagline, revenue
IMDB link, TMDB link
Overview
Genres (just name)
(possibly use backdrop image as a backdrop)
Ratings
(popularity, average, count)
Rater (some way to add your rating)

15. **Movie Details**. [View has been revised]. Displays detailed information for the selected movie. I expect this data to be nicely formatted and laid out sensibly. I have put the info here in two columns 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 Close button will return user to the Default View. The Add to Favorites will add movie to favorites list. The favorites list will still be available.

    The average rating is a value between 0 and 10. Create a component that displays the appropriate number of filled, empty, or half-star icons, using free font-awesome icons. Round the average to nearest half number (e.g., 5.2 round to 5, 5.4 round to 5.5, 5.8 round to 6). Create another components that allows the user to add a rating. After adding a rating to a movie, the user should no longer be able to add a rating.

16. **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. Use the same modal component that you used for the About information.