COMP 4513 Assignment #1: React

Due Monday February 24th at midnight
Version 1.0 (Feb 3, 2020)

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 or three on this assignment. Please don't ask for a group of 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. Heroku, 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 15% of the course grade. The grade for this assignment will be broken down as follows:

Visual Design, Styling, and Usability	23%
Programming Design + Documentation	12%
Functionality (follows requirements)	65%

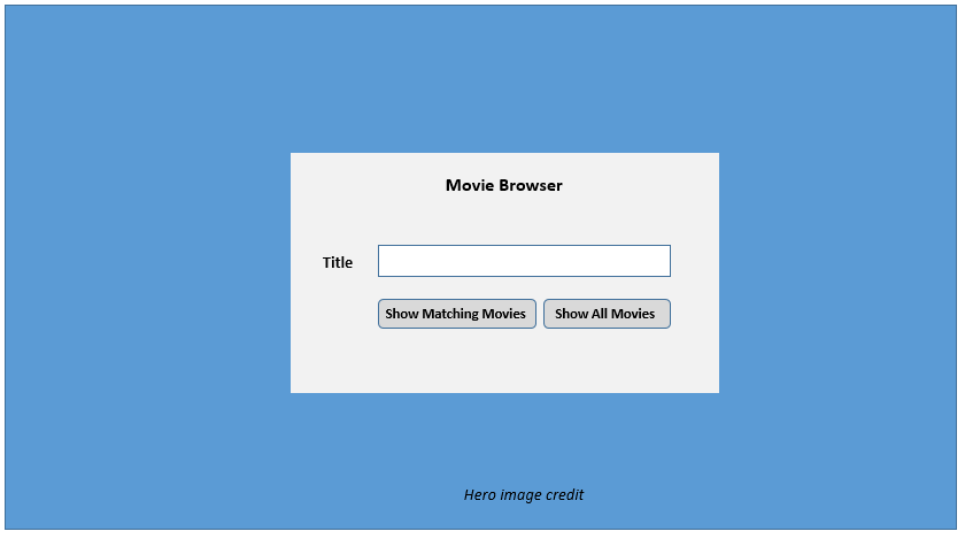
Requirements

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 23% design mark will be based on my assessment of how much effort you made on the styling and design.

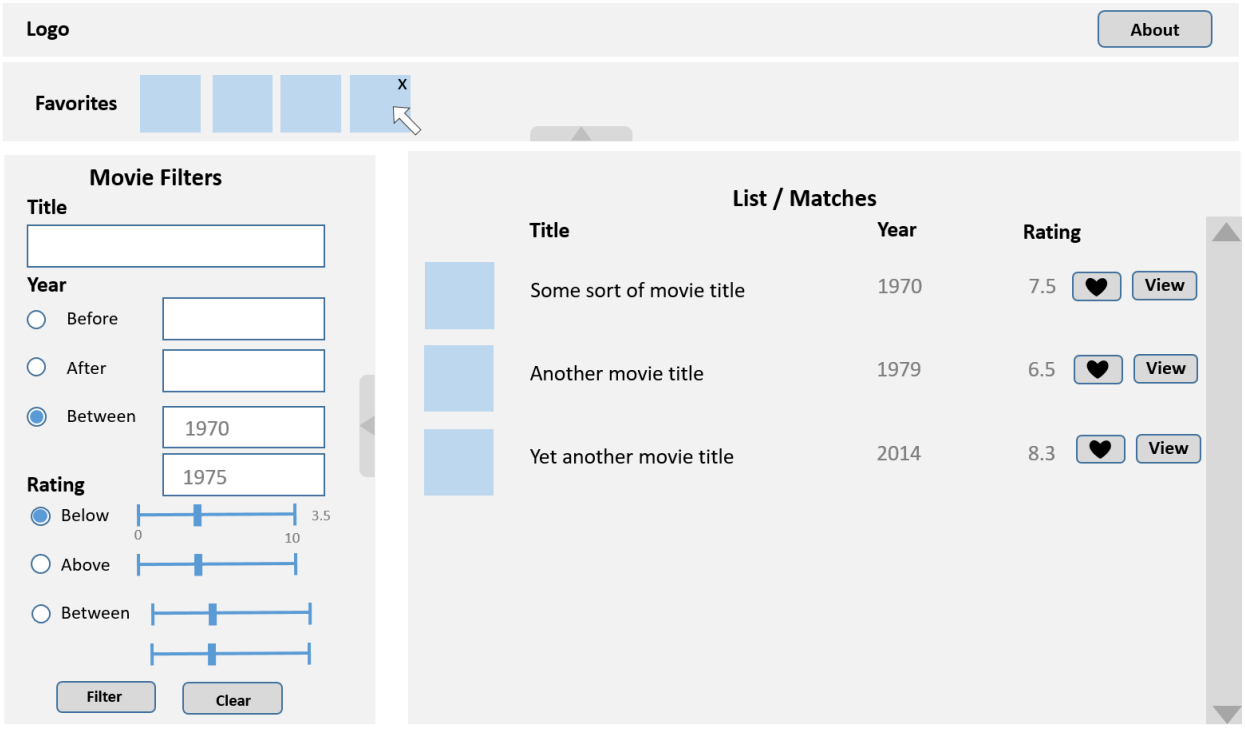
   You can change the layout if you wish. If you want your movie list to be horizontal on the bottom and the favorites to be vertical on the right, you can do so if you think that would be best … though I may not necessarily agree about the usability of your choices. I'd rather see you try something different 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 or CSS that you've found online, I expect you to indicate that in your documentation and in the About view. Be sure to 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!

Home View



4. 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.
5. You must include some type of CSS transition and animation effect on this page. The more interesting it is, the more marks to be gained. I would also like you to use react-transition-group for this effect. See `https://reactjs.org/docs/animation.html` for more information.

Default View



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.

6. **Header**. The header should have a logo and a link/button to an About dialog. The logo should return to the Home View.

7. **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.

   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).

8. 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.

9. 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).

10. 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
    ```

11. **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.

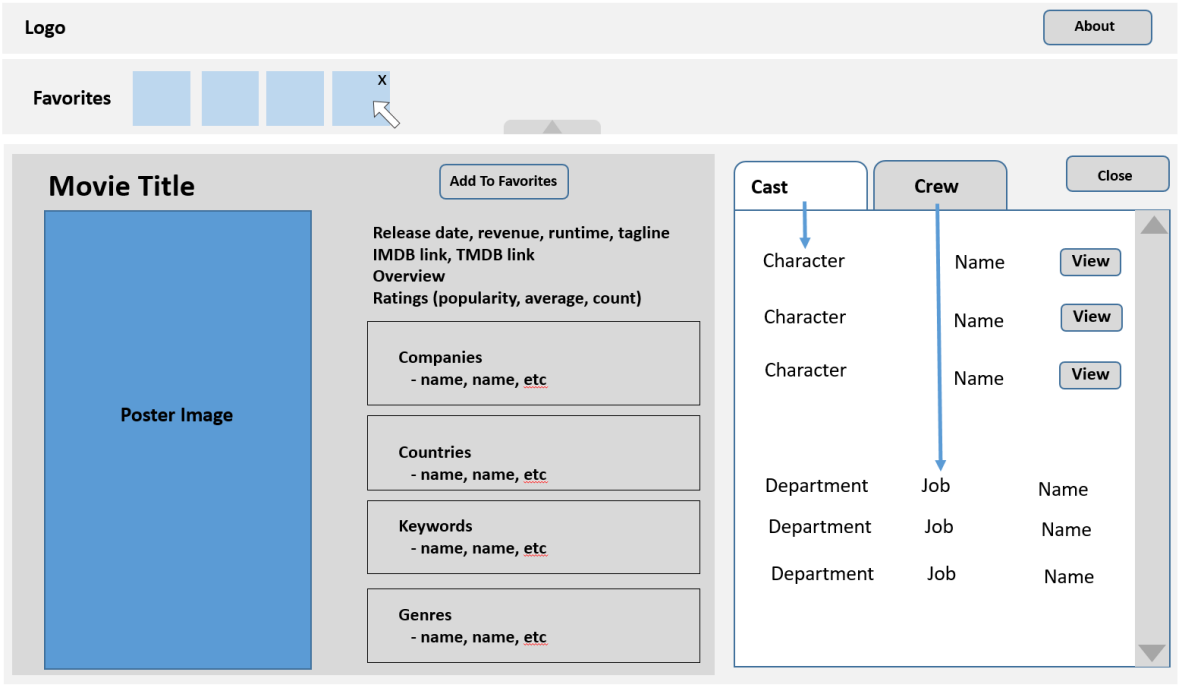
    If no matching movies are found, notify the user within the Movie List/Matches area.

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

    These filters are not mutually exclusive. For instance, a user should be able to search for movies with the word "car" in it with a release year between 1990 and 2000 and a rating above 4.

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



12. **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. 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 Close button will return user to the Default View. The Add to Favorites will add movie to favorites list.

    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).

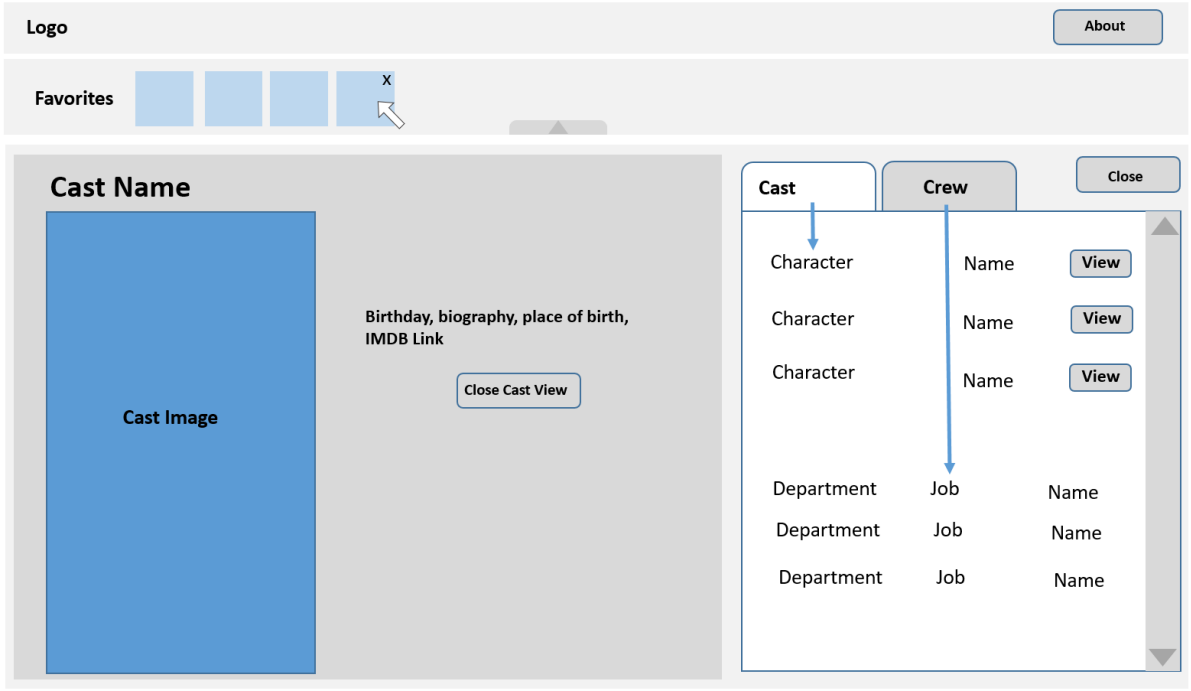
13. **Cast and Crew**. In the sketch above, Cast and Crew are shown as tabs, though you could use hide / show boxes as well. The Cast tab (it should be showing by default) will show the character name (in the movie) and the actor's name (in real-life).

    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.

    For the cast members, also display a View button. Clicking on it will take the user to **Cast View**.

14. **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.

Cast View



15. Cast View replaces the movie specific information with more information about the selected cast member. Clicking Close Cast View will hide this new cast information (that is, return to Movie Details View).

    This information will be retrieved from the TMDB API, which will require registering at https://www.themoviedb.org/. You either immediately get an API key when you register or you have to request one (I can't remember, it's been a long time since I had mine). The API key is under your profile settings for the site.

When you retrieve information about a movie from `www.randyconnolly.com/funwebdev/3rd/api/movie/movies.php`, each object in the `cast` array has the following structure:

```
{
"cast_id": 4,
"character": "Chili Palmer",
"credit_id": "52fe43cbc3a36847f8070361",
"gender": 2,
"id": 8891,
"name": "John Travolta",
"order": 0
}
```

The `id` property will be used to make a person request from the tmdb API. For instance, a person request for John Travolta will look like:

`https://api.themoviedb.org/3/person/8891?api_key=[your API key]`

This will return a JSON object containing various information fields. I would like you to display birthday, biography, place of birth, and their IMDB link (using `https://www.imdb.com/name/yyyy`, where yyyy is the `imdb_id` field). For some cast members these fields will be empty, so handle that gracefully.

This JSON object also contains a `profile_path` property, which is a image of the cast member. You construct the request the same as with the movie posters, except the only size is w185.