Movie Browser App using React

# Description of the application

The Movie Browser App is a single-page-application that acts as a movie browser simulation where users can interact with numerous movie objects. The application is created using React, and mostly designed using Tailwind CSS. It allows users to search a movie, and results will be displayed on the default page where filters and the user’s favourites list are being displayed as well. Filters such as title, year, rating, and genres are included and are available for the users to filter their movies. If a user enters a search query using the title, the app will filter out the right set of movies. Same logic applies when applying the other filters. Users can interactively view each movie by clicking the view button which leads to the movie details page. This page consists of all the information that was gathered from the Movie API. On this page, users can click on the poster to display a modal of a bigger version of it, and submit their rating of the movie. However, the user rating submissions are only for show and will not affect the data being used.

# Application Requirements

* Used create-react-app
* Application deployment on a hosting site
* Redesigned layouts
* Referenced JS code
* Tailwind CSS
* Limit API data in order to reduce bandwidth usage (Fetching movie data once)
* Loading GIF animation while data is being gathered and stored completely (GIF Animation is done on Home, but all the data loading and gathering are being done on the App.js when the application starts up.)
* Multiple React component views such as Home, Default, Movie Details, and more.
* Header must consist of a logo and links to other views of the application
* Filters must consist of title, year, rating and genre, and sorting must consist of title, year and rating (Filters are mutually exclusive)
* Modal usage for About Us and the Movie Details Poster
* User rating submission (this is for show) and generate stars depending on movie’s ratings

Although there are a few more requirements listed on the assignment document, this list summarizes what is essentially needed for functionality.

# Data Structures

We have two data structures that are saved in local storage. The first is a JSON object of the Movie Data from the API. This data is unchanged from the API. The second is a similar list of movies for the favourites. WHen a favourite is added, it is added to this local storage item. This is so that the favourites are viewable on multiple different routes as state is not preserved when a route changes.

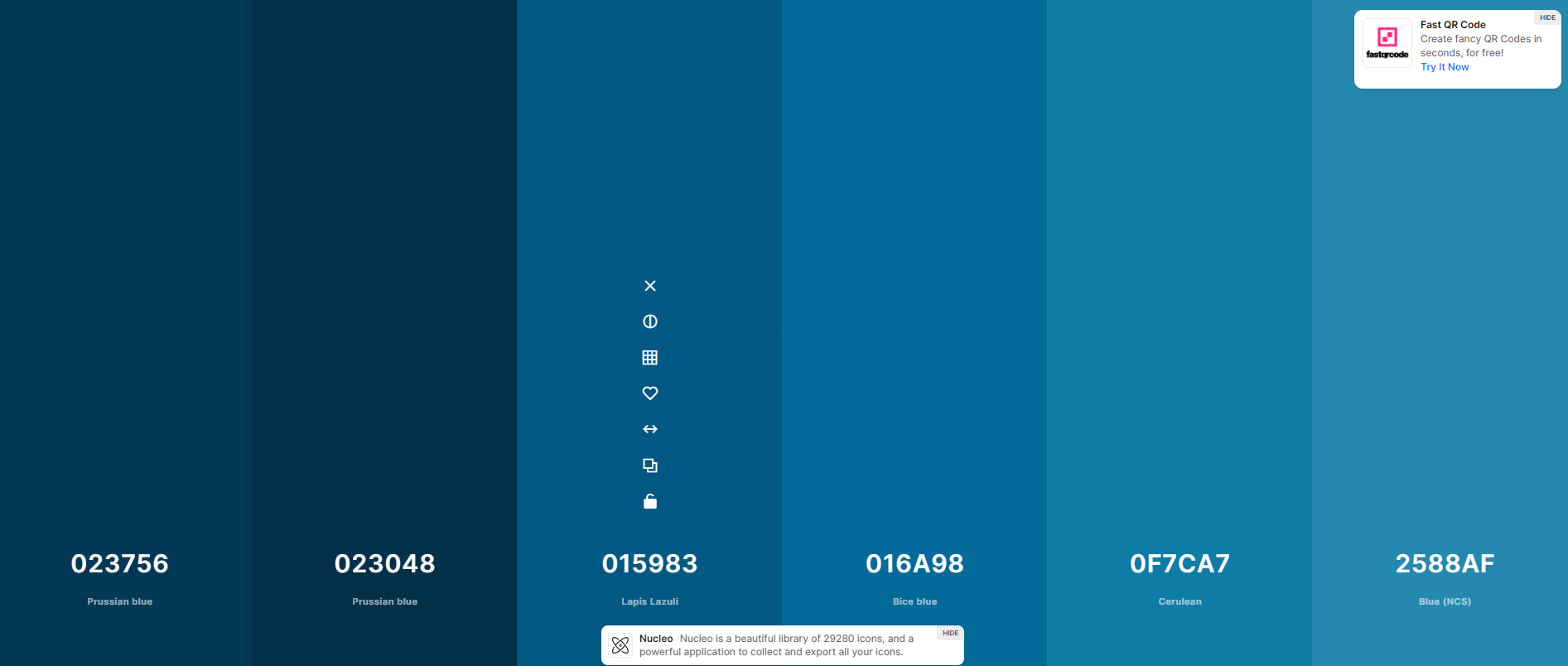
# Visual Design/Aesthetics

## App Logo

# 

Reference: <https://looka.com/editor/114029708>

## Color Palette



# Programming/Code design

## Functions

### Data access

To load data into our application, we’ve included a state hook to achieve this:

// state hook for movie data

const [movieData, setMovieData] = useState([]);

useEffect(() => {

if (!movieData.length > 0) {

// check if the movie data is already in local storage

if (localStorage.getItem('movieData')) {

// get the movie data from local storage

const data = JSON.parse(localStorage.getItem('movieData'));

// set the movie data

setMovieData(data);

return;

} else {

// fetch data from the API

fetch('https://www.randyconnolly.com/funwebdev/3rd/api/movie/movies-brief.php?limit=300')

.then(response => response.json())

.then(data => {

// set the movie data

setMovieData(data);

// save the movie data to local storage

localStorage.setItem('movieData', JSON.stringify(data));

localStorage.setItem('favourites', JSON.stringify([]));

})

.catch(error => console.log(error));

}

}

}, [movieData]);

### Route Usage

We implemented the Routes and Route components from react-router-dom to be able to navigate through the application:

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

<Route path="/moviedetails" element={<MovieDetails movieData={movieData} />} />

<Route path="/default" element={<Default movieData={movieData} />} />

<Route path="/stars" element={<StarsRating />} />

</Routes>

### Filtering

We implemented the following block of code to be able to filter the movies:

const allGenres = [];

props.movies.forEach((movie) => {

if (movie.details.genres) {

movie.details.genres.forEach((genre) => {

if (!allGenres.some((g) => g.id === genre.id)) {

allGenres.push(genre);

}

});

}

});

setGenres(allGenres);

}, [props.movies]);

useEffect(() => {

if (searchParms.get("title") && initialTitle !== null) {

console.log("setting title");

setTitle(searchParms.get("title"));

props.onFilterChange({ title: searchParms.get("title") });

setTimeout(() => {

setInitialTitle(null);

}, 1);

}

}, [props, searchParms, initialTitle]);

const handleTitleChange = (event) => {

setTitle(event.target.value);

props.onFilterChange({ title: event.target.value });

// setSearchParms({ title: event.target.value });

};

const handleGenreChange = (event) => {

setGenre(event.target.value);

props.onFilterChange({ genre: event.target.value });

};

const handleYearChange = (event) => {

if (event.target.id === "year-filter-min") {

setMinYear(event.target.value);

props.onFilterChange({ minYear: event.target.value, maxYear: maxYear });

} else if (event.target.id === "year-filter-max") {

setMaxYear(event.target.value);

props.onFilterChange({ minYear: minYear, maxYear: event.target.value });

}

};

const handleRatingChange = (event) => {

if (event.target.id === "rating-filter-min") {

setMinRating(event.target.value);

props.onFilterChange({

minRating: event.target.value,

maxRating: maxRating,

});

} else if (event.target.id === "rating-filter-max") {

setMaxRating(event.target.value);

props.onFilterChange({

minRating: minRating,

maxRating: event.target.value,

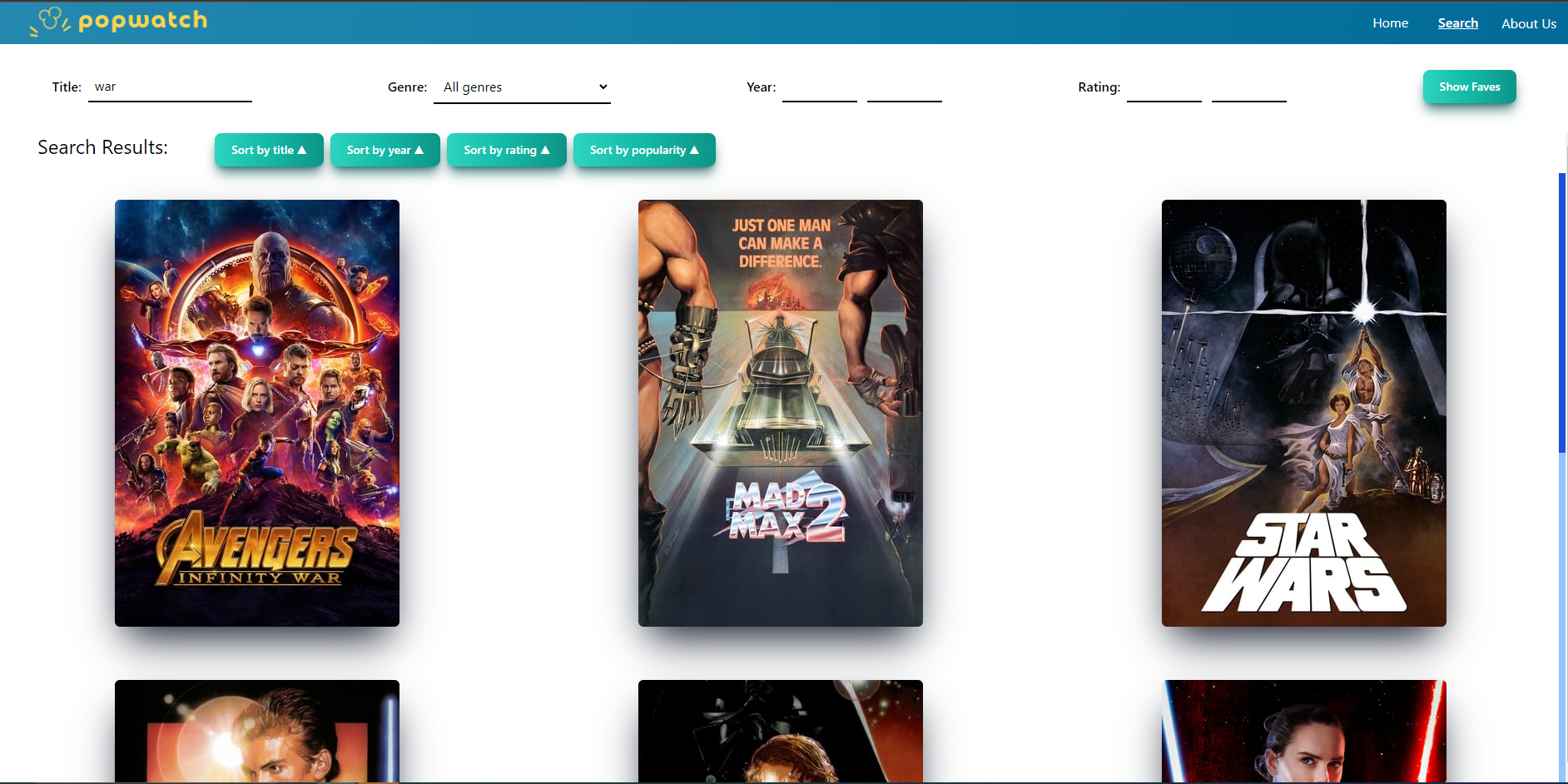
});

}

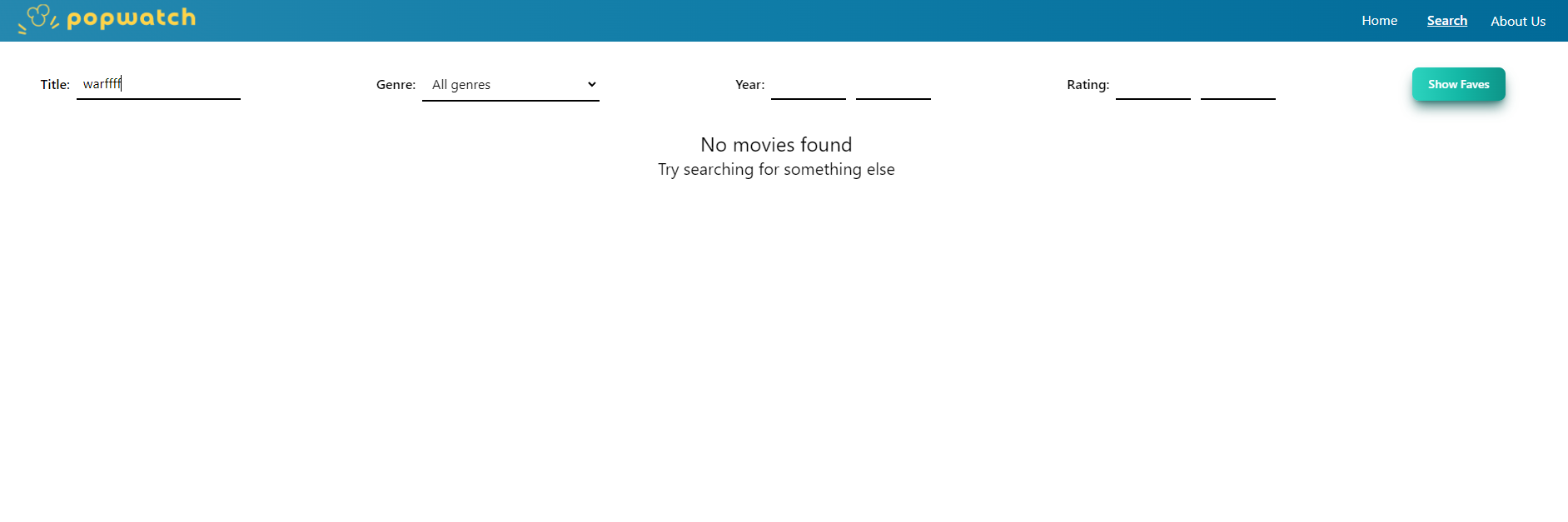
};

# Sample Run

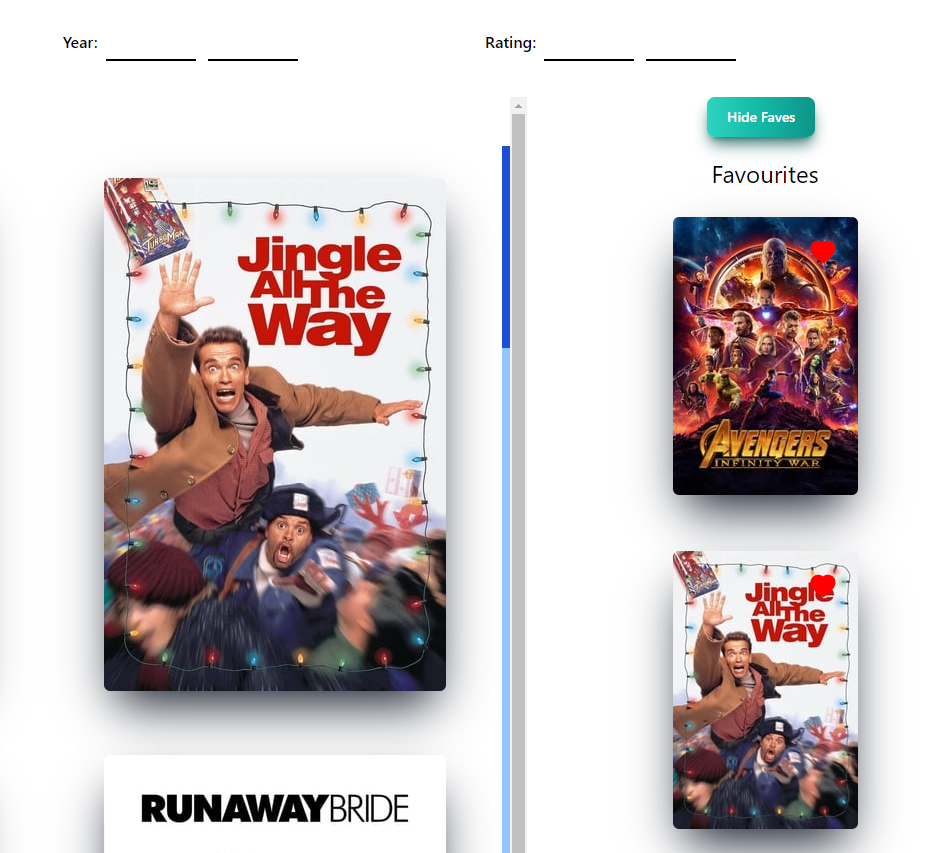
1. Title Query -
   1. User enters “war” from Home Page, Movie List will display the filtered movies containing the query.



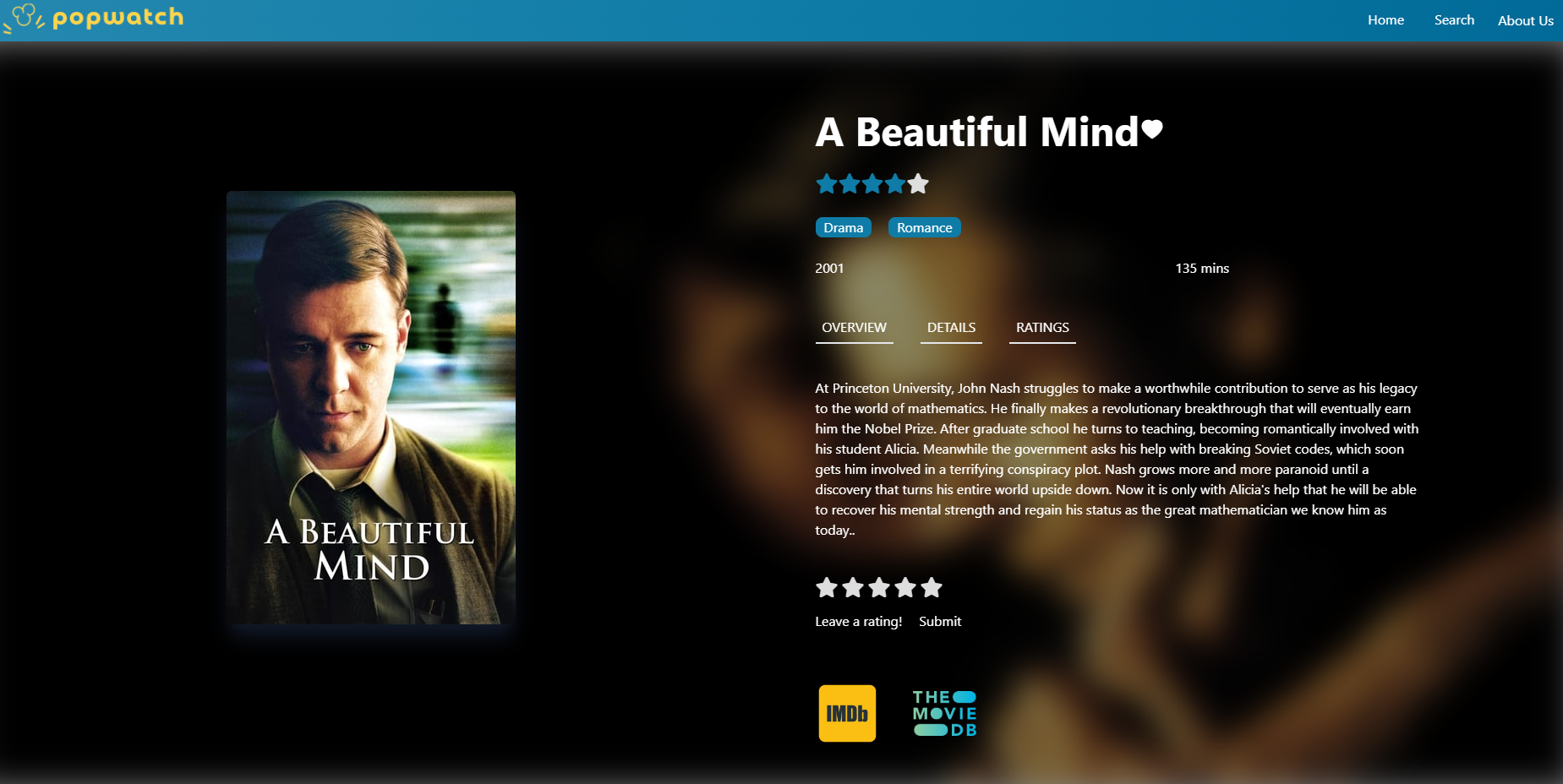
* 1. User enters a query that does not display any movies.



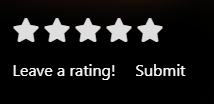
1. Favourite List - Contains something



1. Movie Details
   1. Layout



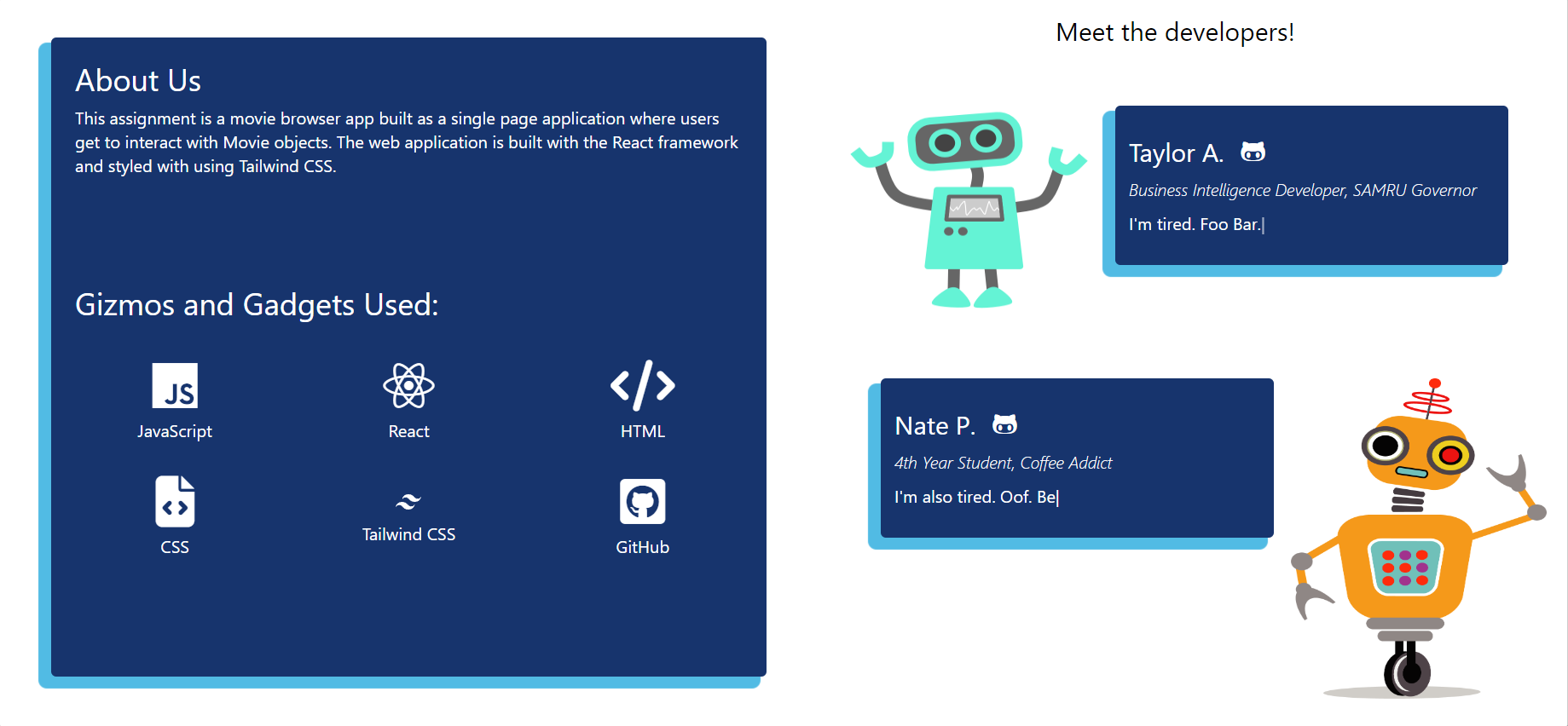
* 1. User Stars Submission
     1. Before



* + 1. After



3. About Us



# References in Code:

**MovieList.js - Line 111**

Scrollbar css reference => https://tailwind-scrollbar-example.adoxography.repl.co/ && <https://www.npmjs.com/package/tailwind-scrollbar>

**Button CSS** - MovieDetails, Home, Default

Button CSS => <https://flowbite.com/docs/components/buttons/>

**React Modal** - MovieDetails Poster popup, About Us pop

React Modal => <https://reactcommunity.org/react-modal/>

**Input Tailwind CSS** - Home, Default

Input CSS => <https://tailwindcomponents.com/component/search-input-1>

**Hero Image** - Home

Hero BG -> Joanna Kosinka, <https://unsplash.com/photos/mjC9apK53a8>

**Nate Robot and Taylor Robot** - About Us

Nate => https://www.pngwing.com/en/free-png-npqzk

Taylor => <https://www.pngwing.com/en/free-png-bywtx>

**Typed** **Component** - Home, About Us (NOTE: THIS COMPONENT IS ONLY FOR AESTHETICS PURPOSES!!)

<https://www.npmjs.com/package/react-typed>

**Stars Rating Component** - MovieDetails, StarsRating.js - (YouTube video)

<https://www.youtube.com/watch?v=eDw46GYAIDQ>

**Stars for Rating Components (Font Awesome) -** MovieDetails, StarsRating.js, MovieDetailsStar.js

<https://fontawesome.com/icons/star?s=solid&f=classic>

**Currency Formatter** - MovieDetails, MovieDetailsExtra.js

https://www.freecodecamp.org/news/how-to-format-number-as-currency-in-javascript-one-line-of-code/

**TMDB and IMDB Logos**

IMDB - <https://brand.imdb.com/imdb>

TMDB - <https://www.themoviedb.org/about/logos-attribution?language=en-CA>