# usePopcorn description

1. Overview
   1. Calls an API at omdb api when you enter values in the Search Bar.
   2. A list of movies comes up on the left hand side
   3. Click on a movie and it opens up details about the movie on the right hand side
   4. Rate a movie with the stars and click Add to List button which will add it to the list on the right-hand side under Movies You Watched
   5. The Movies You Watched header on the right-hand side contains the total movies you have watched, the Average Rating, Number of Minues
2. Technical Overview
   1. The API key is in a const call KEY
   2. In the APP.js
      1. Uses the following useState functions

const [query, setQuery] = useState("");

const [movies, setMovies] = useState([]);

const [watched, setWatched] = useState([]);

const [isLoading, setIsLoading] = useState(false);

const [error, setError] = useState("");

const [selectedId, setSelectedId] = useState(null);

* 1. First thing that happens is that it uses a useEffect hook to fetch the Movie data.
     1. Defines and calls an asynchronous function named fetchMovies. This function fetches movie data from the OMDB API and updates the component's state with the fetched data.
     2. This useEffect function is run when the app component mounts as well as when the query state changes
     3. The fetchMovies function the setIsLoading state is set to true
     4. The setError state is set to ‘’ a blank string
     5. A constant name res for response is defined to hold the values returned from the fetch of the API.

const res = await fetch(`http://www.omdbapi.com/?s=${query}&apikey=${KEY}`);

* + 1. If the response was not successful, throw an error with a message. If the response is not ok (the response status is not in the range 200-299), then an error is thrown.

if (!res.ok) {

          throw new Error('Could not fetch movies.  Check your internet connection and try again.'); }

* + 1. If there is not an error then the constant data is assigned the response data converted to JSON.

// Parse the response as JSON

      const data = await res.json();

* + 1. Check to make sure that the response data that is converted to JSON doesn’t have an error. If no errors then set the state of the setMovies. If the API response indicates an error, throw an error with the API's error message this could also indicate that there was a 404 error meaning the movie was not found

if (data.Response === 'False') throw new Error(data.Error);

        // Update the movies state with the Search results from the API response

        setMovies(data.Search);

* + 1. Everything is in a try catch statement so anything that wasn’t caught gets caught in the catch section and the setError state is set to the error.message. If the response property of the JSON data is ‘False’, it means that the API returned an error, so an error is thrown with the Error property of the data. I f there is no error, the set Movies function is called with the Search property of the data to update the state of the movies.

catch (error) {

        // Log the error to the console

        console.error(error.message);

        setError(error.message);

* + 1. There is finally a finally statement in the try catch finally statement after everything runs it sets the setIsLoading state back to false

 finally {

        // Set the isLoading state to false to indicate that loading is complete

        setIsLoading(false);

      }

* + 1. While still inside the useEffect function the code checks to see if query state is an empty string and if so it sets the setMovies state to a blank array [], and the setError to an empty string ‘’.

    if (!query.length) {

      setMovies([])

      setError('')

      return

    }

* + 1. The first thing the App function returns is the Navbar which contains the Search and NumResults components. Also, the Navbar imports the Logo.
    2. The **Navbar** props is {children} so after the **Logo** both the **Search** and the **NumResults** are passed in via the props children.
       1. The **Search** component
          1. Takes the props query, setQuery
          2. This is a text field with the value assigned to the query state
          3. It gets assigned to the query state by the onChange event where is sets the setQuery state based on what the user enters in the Search text field
       2. **NumResults** component
          1. Takes the props movies
          2. In a p tag it appends the movies.lenth to the text
    3. The next part of the JSX that is returned is the **Main** component which wraps the **Box** Component children as <Main><Box></Box><Box></Box></Main>
       1. The **Main** component
          1. Passes {children} as its props
          2. The **Main** contains two **Box** children props
       2. The Box component has {children} as its props
          1. This component uses useState of [isOpen, setIsOpen]
          2. Inside the returned div is a button with an onClick event that sets the setIsOpen state to !open or to open depending on its current state.
          3. Also, the button label is set to – or + if isOpen then its set to – else its set to +
          4. The last part of the div is {isOpen && children} - This displays the children elements if isOpen is true else they are not displayed.
       3. The children inside this first **Box** jsx element is the **Loader**.
          1. If the state isLoading is true then the **Loader** is passed as a child to the Box component
          2. **Loader** component

Simply is a p element with Loading…. text

* + - * 1. If isLoading is false and error is false, render the **MovieList** component and pass that as part of the children to the Box.
        2. Next inside the Box we check the error. If the error is true, render the **ErrorMessage** component else if not true or there is no error then do not render the **ErrorMessage** component and do not pass it as a child to the Box component.
        3. The ErrorMessage component

The props is a message {message} the message variable is assigned the error value

* + - 1. The children inside the second Box jsx element are passed to the Box element.
         1. The first thing that is checked in this box is if the selectedId is checked or not. This is checked if a user clicked on a movie in the MovieList and it set the selectedID in the state.
         2. **MovieList** component on the lefthand column

Props are movies, onSelectMovie

Imports **Movie** component

Props are movie, onSelectMovie

Returns a list. The list element has an onClick event with a function onSelectMovie(movie.imdbID)

So if the user clicks on the Movie in the MovieList it sets the selectedId value via the function handleSelectMovie in the App. If the currently selectedID is not equal to the one clicked then the id is set to the clicked id.

* + - * 1. So if there is a selectedID then it add the selectedID to the MovieDetails component
        2. MovieDetails component

Props are selectedId, onCloseMovie, onAddWatched, watched

Its state functions are:

const [movie, setMovie] = useState({});

    const [isLoading, setIsLoading] = useState(false);

    const [userRating, setUserRating] = useState('');

isWatched variable

const isWatched = watched.map((movie) => movie.imdbID).includes(selectedId);

The watched variable is an array of movie objects, each with an imdbID property. The map function is used to create a new array that consists of the imdbID of each movie in the watched array.

The includes function is then called on this new array of imdbIDs. It checks if selectedId is present in the array. The selectedId is the imdbID of a movie that the user has selected.

The result of this check (a boolean value) is assigned to the isWatched constant. If selectedId is found within the array of imdbIDs, isWatched will be true, indicating that the selected movie has been watched. If selectedId is not found, isWatched will be false, indicating that the selected movie has not been watched.

A useEffect hook is used to fetch the details of the selected movie when the selectedId changes.

// Use the useEffect hook to perform side effects

useEffect(function () {

    // Define an asynchronous function to fetch movie details

    async function getMovieDetails() {

        // Set the isLoading state to true to indicate that the data is currently being loaded

        setIsLoading(true);

        try {

            // Make a fetch request to the OMDB API with the selectedId and the API key

            const res = await fetch(`http://www.omdbapi.com/?i=${selectedId}&apikey=${KEY}`);

            // If the response status is not ok, throw an error

            if (!res.ok) {

                throw new Error('Could not fetch movie details.  Check your internet connection and try again.');

            }

            // Parse the response as JSON

            const data = await res.json();

            // Update the movie state with the fetched data

            setMovie(data);

        } catch (error) {

            // If an error is thrown, log the error message to the console

            console.error(error.message);

        } finally {

            // Whether an error was thrown or not, set the isLoading state to false to indicate that the loading is complete

            setIsLoading(false);

        }

    }

    // Call the getMovieDetails function to start the fetch request

    getMovieDetails();

// The useEffect hook is set to run whenever the selectedId state changes

}, [selectedId]);

The MovieDetails component returns a header element