React-7 : watch list and localstorage

Agenda
Creating the watchlist
Features of Watchlist
Local Storage for keeping data persistent
sorting
searching
filtering on the basis of gener of movie

title: storing watchList in local storage

let's go to page number , add some movies to watch list and reload our page and check whether our application still works or not .

Observation : We see our application is still working as we are able to see list of movies but page number is reset to 1 and watch List is removed .

Question : Why is this happening. Why are we facing the above issue ??

Answer : all the states are reinitialised

local storage (Review)

Local storage is a feature in web browsers that allows web applications to store data locally within the user's browser.

Features:

1. Persistent Storage: Data stored in local storage persists even after the browser is closed and reopened.
2. Large Storage Capacity: Typically allows storing more data (usually up to 5-10MB) compared to cookies.
3. Accessible Across Pages: Data stored in local storage can be accessed by any page from the same origin (domain).
4. No Expiration: Data remains stored indefinitely until explicitly removed by the web application or cleared by the user.

```
// To set an item in local storage
localStorage.setItem('key', 'value');

// To get an item from local storage
const value = localStorage.getItem('key');

// To remove an item from local storage
localStorage.removeItem('key');

// To clear all items from local storage
localStorage.clear();
```

Let's use local storage persist the watchList

```
const addToWatchList = (movieObj) => {
  const updatedWatchlist = [...watchList, movieObj];
  setWatchList(updatedWatchlist);
  localStorage.setItem('movies', JSON.stringify(updatedWatchlist));
};
```

```
const removeFromWatchList = (movieObj) => {
  let filtredMovies = watchList.filter((movie) => {
    return movie.id != movieObj.id;
  });
  setWatchList(filtredMovies);
  localStorage.setItem('movies', JSON.stringify(filtredMovies))
};
```

retrieving the local storage: We will be using useffect that will run only once after first render to check for any movies in the watchList

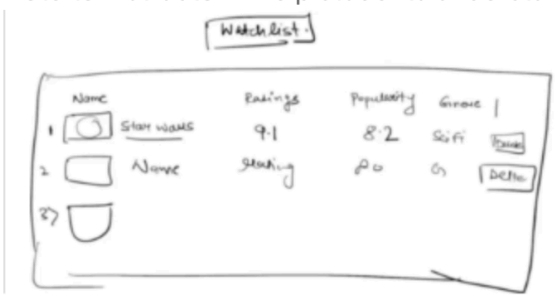
```
useEffect(() => {
  const moviesFromLocalStorage = localStorage.getItem("movies");
  if (moviesFromLocalStorage) {
    setWatchList(JSON.parse(moviesFromLocalStorage));
  }
}, []);
```

If we reload it again it will watch list will still be intact

title: Watch List component

Wireframe

Note to instructor - Help students understand and brainstorm this.



our watchList look like this

Name	Ratings	Popularity	Genre
 Godzilla x Kong: The New Empire	7.911	1758.469	Action
 Imaginary	5.888	286.724	Action
 Dune: Part Two	8.39	702.285	Action

let's create a static version of watch List

```
function WatchList() {
 return (
   <div className="overflow-hidden rounded-lg border border-gray-200 shadow-md m-5">
     <table className="w-full border-collapse bg-white text-left text-sm
text-gray-500">
       <thead>
         <tr className="bg-gray-50">
           <th className="px-6 py-4 font-medium text-gray-900">Name</th>
           <th>
             <div className="flex">
               <div>Ratings</div>
             </div>
           </th>
           <th>
```

```
            <div className="flex">
              <div>Popularity</div>
            </div>
          </th>
          <th>
            <div className="flex">
              <div>Genre</div>
            </div>
          </th>
        </tr>
      </thead>
      <tbody className="divide-y divide-gray-100 border-t border-gray-100">
        <tr className="hover:bg-gray-50">
          <td className="flex items-center px-6 py-4 font-normal text-gray-900">
            <img className="h-[6rem] w-[10rem] object-fit" src="" alt="" />
            <div className="font-medium text-gray-700 text-sm">Star Wars</div>
          </td>
          <td className="pl-6 py-4">7.8</td>
          <td className="pl-6 py-4">7.8</td>
          <td className="pl-2 py-4">Action</td>
        </tr>
      </tbody>
    </table>
  </div>
 );
}
export default WatchList;
```

Now, we'd like to incorporate this data. Inside the table body, We will utilize the movies array:
We will be using the data in local storage for that

1. Define a watchList state variable -> it will be empty
2. use useEffect to get the list of movies
3. update the state variable
4. and replace the static values with state data

```
import React, { useEffect, useState } from "react";

function WatchList() {
 const [watchList,setWatchList]=useState([]);
 useEffect(()=>{
   let moviesFromLocalStorage = localStorage.getItem('movies')
   if(!moviesFromLocalStorage){
     return
   }
   setWatchList(JSON.parse(moviesFromLocalStorage))
} , [])
 return (
   <div className="overflow-hidden rounded-lg border border-gray-200 shadow-md m-5">
     <table className="w-full border-collapse bg-white text-left text-sm
text-gray-500">
       <thead>
         <tr className="bg-gray-50">
           <th className="px-6 py-4 font-medium text-gray-900">Name</th>
           <th>
             <div className="flex">
               <div>Ratings</div>
             </div>
           </th>
           <th>
             <div className="flex">
               <div>Popularity</div>
             </div>
           </th>
           <th>
             <div className="flex">
               <div>Genre</div>
             </div>
           </th>
         </tr>
       </thead>
       <tbody className="divide-y divide-gray-100 border-t border-gray-100">
       {watchList.map((movie) => (
         <tr className="hover:bg-gray-50" key={movie.id}>
           <td className="flex items-center px-6 py-4 font-normal text-gray-900
gap-4">
```

```
            <img className="h-[6rem] w-[10rem] object-fit object-cover rounded-md"
src={`https://image.tmdb.org/t/p/original/${movie.poster_path}`} alt="" />
            <div className="font-medium text-gray-700 text-sm">{movie.title}</div>
          </td>
          <td className="pl-6 py-4">{movie.vote_average}</td>
          <td className="pl-6 py-4">{movie.popularity}</td>
          <td className="pl-2 py-4">Action</td>
        </tr>
      ))}
      </tbody>
    </table>
  </div>
 );
}
export default WatchList;
```

rendering correct genre

We are getting only the genre ids from the api.

I was given a mapping of these ids to the actual genre.

1. Create a constants folder under src and file called index.js

```
const genreids = {
   28: "Action",
   12: "Adventure",
   16: "Animation",
   35: "Comedy",
   80: "Crime",
   99: "Documentary",
   18: "Drama",
   10751: "Family",
   14: "Fantasy",
   36: "History",
   27: "Horror",
   10402: "Music",
   9648: "Mystery",
   10749: "Romance",
```

```
  878: "Sci-Fi",
  10770: "TV",
  53: "Thriller",
  10752: "War",
  37: "Western",
};
export default genreids;
```

2. You can move your URLs and other application constants here
3. Import genereids in WatchList

```
import genreids from "../constants";
```

4. Create a function to get the genres and join them

```
const genres = (genre_id) => {
  return genreids[genre_id];
};
```

5. Call this function

```
<td className="pl-6 py-4">{movie.popularity}</td>
<td className="pl-2 py-4">{genres(movie.genre_ids[0])}</td>
```

Styling Explaination of above component

1. overflow-hidden: This class hides any content that overflows the container's boundary.
2. rounded-lg: This class applies rounded corners to the container, making it visually softer.
3. border: This class applies a border to the container.

4. border-gray-200: This class sets the color of the border to a shade of gray.
5. shadow-md: This class applies a medium shadow to the container, giving it depth.
6. m-5: This class adds margin spacing of size 5 on all sides of the container.

<table class="w-full border-collapse bg-white text-left text-sm text-gray-500">

1. w-full: This class makes the table take up the full width of its container.
2. border-collapse: This class collapses the borders of adjacent cells into a single border.
3. bg-white: This class sets the background color of the table to white.
4. text-left: This class aligns the text in the table cells to the left.
5. text-sm: This class sets the font size of the text in the table cells to small.
6. text-gray-500: This class sets the text color to a shade of gray.

<tr class="bg-gray-50">

1. bg-gray-50: This class sets the background color of the table row to a light gray shade.

<th class="px-6 py-4 font-medium text-gray-900">Name</th>

1. px-6: This class adds horizontal padding of size 6 to the table header cell.

2. py-4: This class adds vertical padding of size 4 to the table header cell.
3. font-medium: This class applies a medium font weight to the text in the table header cell.
4. text-gray-900: This class sets the text color to a dark gray shade.

<td class="flex items-center px-6 py-4 font-normal text-gray-900">

1. flex: This class makes the table data cell a flex container.
2. items-center: This class vertically centers the content inside the flex container.
3. px-6: This class adds horizontal padding of size 6 to the table data cell.
4. py-4: This class adds vertical padding of size 4 to the table data cell.
5. font-normal: This class applies a normal font weight to the text in the table data cell.
6. text-gray-900: This class sets the text color to a dark gray shade.

1. h-[6rem]: This class sets the height of the image to 6rem (using a responsive length unit).
2. w-[10rem]: This class sets the width of the image to 10rem (using a responsive length unit).

3. object-fit: This class ensures that the image maintains its aspect ratio while fitting within its container.

Search, Sort, Filter

1. We will implement Search, sort, filter in the watchlist page
2. We will put some arrow icons to show ascending / descending
3. We will create a search input box that searches as the user types in the box
4. Lastly we will create labels for different genres and clicking on that buttons with those labels, we will filter or you can use dropdown as well

title: Sorting Implementation

1. Like always we will need to follow these steps
2. Add the UI elements
3. add event listener
4. apply the state update logic for sorting
5. Wrap Rating with one arrow up and one arrow down .
6. Add the below classes . These are coming from font-aweomse library that we used earlier in pagination.
7. Wrap ratings text with icons

```
<div>
                <i className="fa-solid fa-arrow-up hover:cursor-pointer
mx-1"></i> Ratings{" "}
```

```
                <i className="fa-solid fa-arrow-down hover:cursor-pointer
mx-1"></i>
              </div>
```

8. Add the handlers

```
const handleAscendingRatings = () => {
  console.log("arranging movies by ascending order");
};

const handleDescendingRatings = () => {
  console.log("arranging movies by descending order");
};
```

9. Call the handlers on the icons

```
<i
                  onClick={handleAscendingRatings}
                  className="fa-solid fa-arrow-up hover:cursor-pointer mx-1"
                ></i>{" "}
                Ratings{" "}
                <i
                  onClick={handleDescendingRatings}
                  className="fa-solid fa-arrow-down hover:cursor-pointer mx-1"
                ></i>
```

Extra Notes about Sorting

1. **Sorting Without a Comparator Function:**
    a. When you sort an array of objects in JavaScript without a comparator function, JavaScript's built-in sorting algorithm will attempt to convert each element into a string and then compare them based on their Unicode code points. This means that the sorting is done alphabetically or

numerically, depending on the data type of the property you're sorting by.

b. Let's say we have an array of objects representing people with their ages:

```
const people = [
  { name: "Alice", age: 30 },
  { name: "Bob", age: 25 },
  { name: "Charlie", age: 35 }
];
```

c. If we simply call people.sort() without any additional arguments, JavaScript will sort this array alphabetically based on the string representation of the objects:

d. people.sort();

e. Result:

```
// [
//   { name: "Alice", age: 30 },
//   { name: "Bob", age: 25 },
//   { name: "Charlie", age: 35 }
// ]
```

2. Sorting With a Comparator Function:

   a. When you need to sort an array of objects based on a specific property, or based on custom sorting criteria, you can provide a comparator function to the sort() method.

   b. The comparator function should return a negative value if the first argument should come before the second, a

positive value if the first argument should come after the second, and zero if they are equal.

3. Let's sort the people array by age:
   people.sort((person1, person2) => person1.age - person2.age);
4. // Result:
5. // [
6. // { name: "Bob", age: 25 },
7. // { name: "Alice", age: 30 },
8. // { name: "Charlie", age: 35 }
9. //]
10. Here, the comparator function person1.age - person2.age compares the ages of two people. If person1's age is less than person2's age, it returns a negative value, indicating that person1 should come before person2. If person1's age is greater, it returns a positive value, and if they're equal, it returns zero.
11. By providing a comparator function, we can sort the array based on any property or custom criteria we need.

Add the sorting state variable and add update logic to it

```
const handleAscendingRatings = () => {
  let sortedAscending = watchList.sort((movieObjA, movieObjB) => {
    return movieObjA.vote_average - movieObjB.vote_average;
  });

  setWatchList([...sortedAscending]);
};

const handleDescendingRatings = () => {
  let sortedDescending = watchList.sort((movieObjA, movieObjB) => {
```

```
    return movieObjB.vote_average - movieObjA.vote_average;
  });

  setWatchList([...sortedDescending]);
};
```

title: Searching

1. Add the state variable and handler for input change

```
function WatchList() {
 const [watchList, setWatchList] = useState([]);
 const [search, setSearch] = useState("");

 const handleSearch = (e) => {
   setSearch(e.target.value);
 };
```

2. Add this div to the top of the watchlist component

```
<>

    <div className="flex justify-center my-10">
      <input
        placeholder="Search Movies"
        className="h-[3rem] w-[18rem] bg-gray-200 px-4 outline-none border
border-gray-300"
        type="text"
        onChange={handleSearch}
        value={search}
      />
    </div>

    <div className="overflow-hidden rounded-lg border border-gray-200 shadow-md m-5">
```

3. Filter the movies before they are rendered

```
{watchList
              .filter((movie) => {
                return movie.title.toLowerCase().includes(search.toLowerCase());
              })
              .map((movie) => (
```

Filtering

filter by genre is a two part process

Listing out valid genre options -> because we are showing only the genres that have at least one movie in the list

when you click on a particular genre then only showing the movies that belong to that genre
Let's tackle the first Part

listing out all the valid Genres

1. UI for Genres buttons with dummy generes data
2. modifying the code to only generate the valid genres
3. Interactivity: Adding Event listener to get to know about the which genre is clicked

<u>UI of Generes:</u>

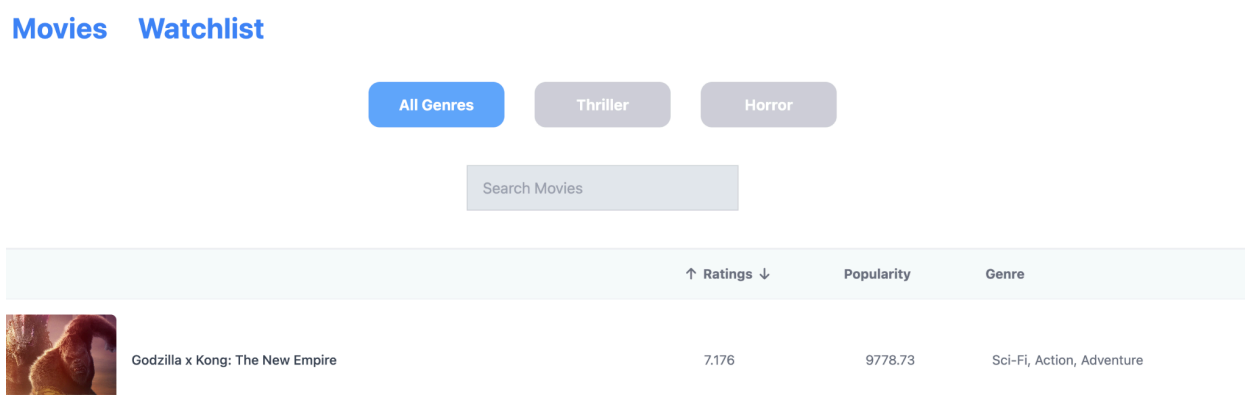
listing out all the valid Genres

1. Add a row of above search for genres
2. add two state variable in WatchList
    a. genreList->that list out all the genres
    b. currGenre -> that will indicate currently selected genere . By default it should be all generes and should be blue in color

```
const [genreList, setGenreList] = useState(["All Genres", "Thriller","Horror"]);
const [currGenre, setCurrGenre] = useState("All Genres");
```

3. Add the component to the top of the WatchList

```
<>
     {/* Genre */}
     <div className="flex justify-center m-4">
       {genreList.map((genre) => {
         const isActive = currGenre === genre;
         const baseStyles =
           "flex justify-center items-center h-[3rem] w-[9rem] rounded-xl
text-white font-bold mx-4";
         const bgColor = isActive ? "bg-blue-400" : "bg-gray-400/50";
         return <div className={`${baseStyles} ${bgColor}`}>{genre}</div>;
       })}
     </div>
     {/* Search Field */}
```



code to only generate the valid genres

Logic is simple ->map through all the movies in watchList and create a list of unique generes and use it to list out the options . We need thing logic to not run alwyas it should only run when watchList movie changes so we will wrap the above logic in an useEffect

```
function WatchList() {
 const [watchList, setWatchList] = useState([]);
 const [search, setSearch] = useState("");
 const [genreList, setGenreList] = useState([
   "All Genres",
   "Thriller",
   "Horror",
 ]);
 const [currGenre, setCurrGenre] = useState("All Genres");
 //     this useEffect only runs when watchList updates
 //     this useEffect only runs when watchList updates
 useEffect(() => {
   let temp = watchList.map((movie) => {
     return genreids[movie.genre_ids[0]];
   });
   // set stores only the unique entries
   temp = new Set(temp);

   console.log(temp);
```

```
  setGenreList(["All Genres", ...temp]);

  //  console.log([...temp])
}, [watchList]);
```

This useEffect althoug not making any side effect is a valid use as we are trying to set a derived state using watchList and only once.

Adding Event listener to get to know about the which genre is clicked

Add the handler which sets the state

```
const handleFilter = (genre) => {

  setCurrGenre(genre);

};
```

Add onClick handler

```
return <div onClick={() => handleFilter(genre)} className={`${baseStyles}
${bgColor}`}>{genre}</div>;
```

Showing only the movies that belong to current selection of Genres

Like in filtering here also just before we apply the filter we will check if there is any genre is selected or not if yes then we will filter

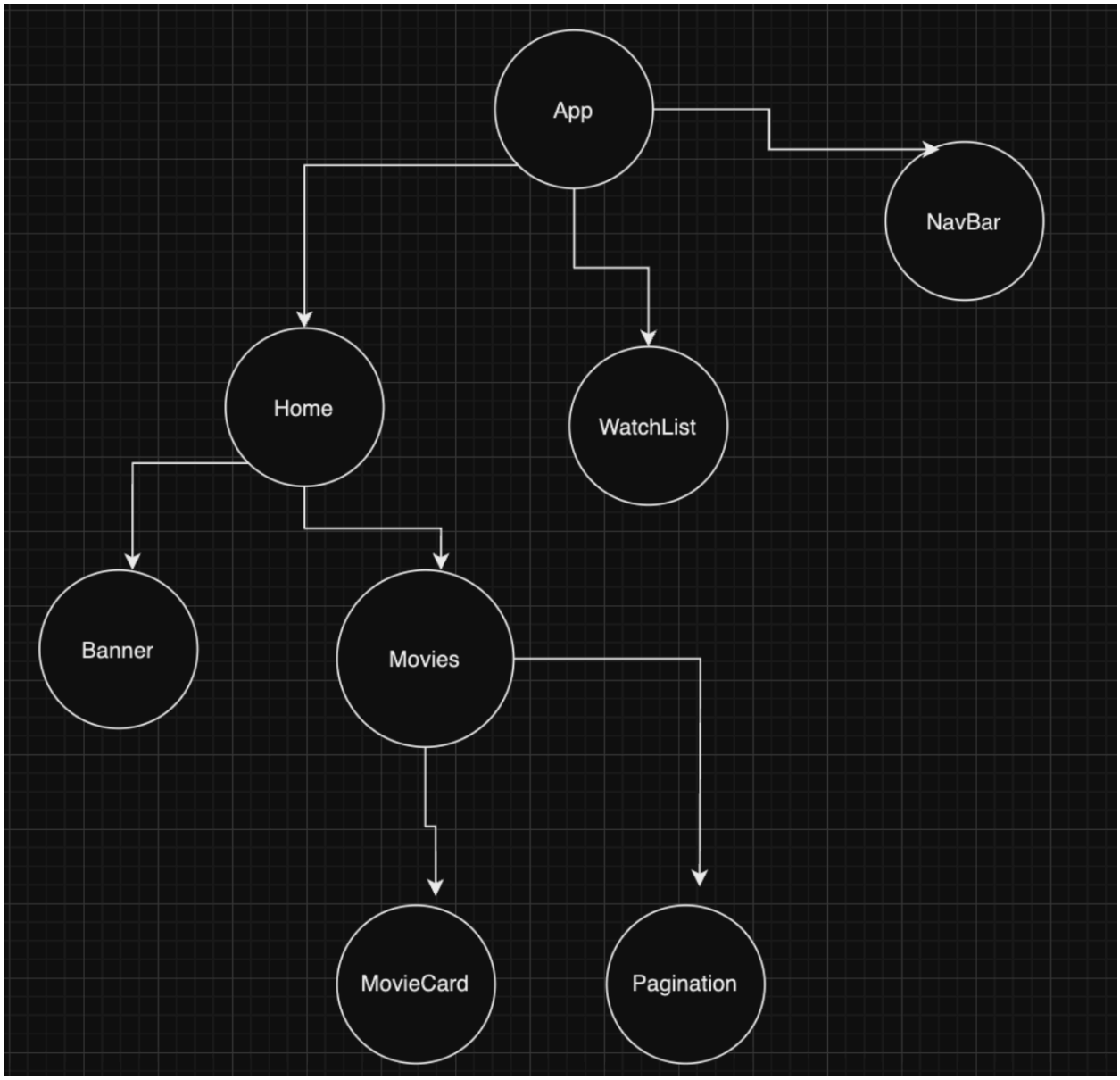
```
{watchList
              .filter((movie) => {
                if (currGenre == "All Genres") {
                  return true;
                } else {
                  return genreids[movie.genre_ids[0]] == currGenre; //
Drama;
                }
              })
            .filter((movie) => {
              return movie.title.toLowerCase().includes(search.toLowerCase());
            })
            .map((movie) => (
```

title: Problem of prop drilling

On UI it should looks like this

	Name	↑ Ratings ↓	Popularity	Genre	Delete Movies
	Rest in Peace	0	18.945	Thriller	Delete

1. The thing is if we try to delete from WatchList, we will need to add the handlers and update state.
2. Two source of truth for same data - is this correct
3. But we need watchlist in the Movies page so that we can update it when user clicks on icons
    4. Now there are two different component in our hierarchy that needs the same state

5. 

6. Solution one : Lifting the state up
    a. Problem - props drilling
    b. we will need to pass the same props from App to Home and then watch List
    c. Prop-drilling : When the descendant needs a prop and the height of that component tree is huge then we have to pass props to the whole chain.

7. Solution two : context API
    a. context API is a feature that is built into react and it is used to fight this Problem . In the next section we will learn about in detail