MOFLIX

Movie App

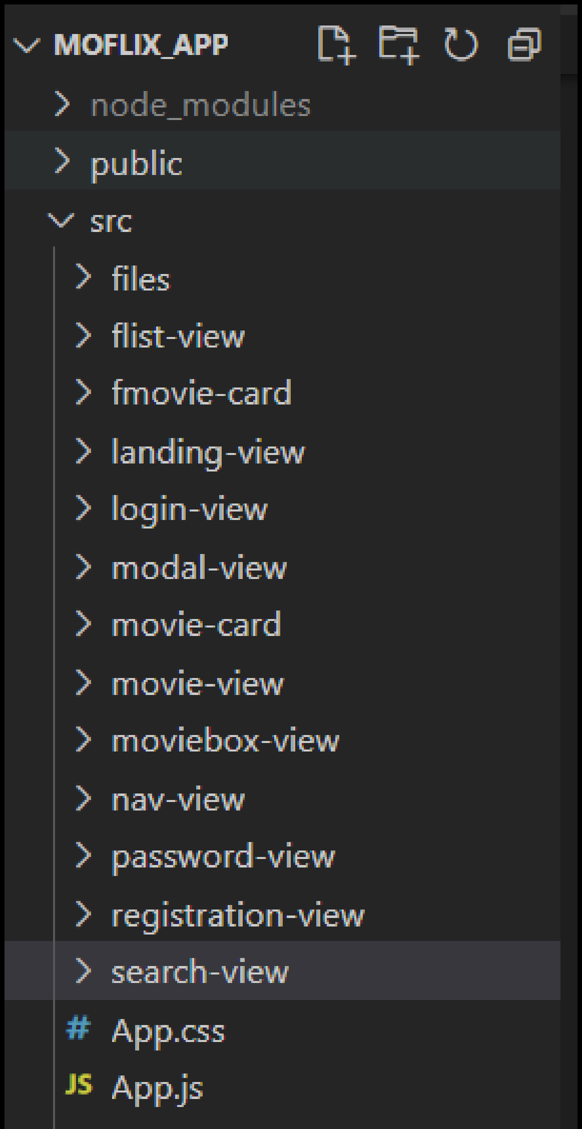
The Challenge

Building Client side code (Front End code) of a movie app. This was a 2 part project in which the REST API and database was first built, then, React was used to create the interface to display the Movie API in a well designed fashioned. This case study will only focus on the front end code and User Interface.

Project Setup

Determining what components I need to build is based off 2 things; what functionalities a movie app requires and what custom functionalities I built in my Rest API and Database.

- Registering a user
- Logging in a user
- Password changer
- Navbar for navigating links
- Home Page
- Modal Pop up Youtube video for featured Movie
- A list of different Movies by category
- MovieCard ; Details / Add movie to FavoriteList
- Search input and search movie results
- List of Favorite Movies
- Favorite MovieCard ; Movie Details / Delete movie from FavoriteList

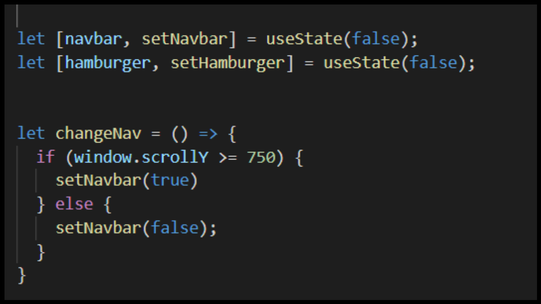


```
MOFLIX_APP
  > node_modules
  > public
  v src
    > files
    > flist-view
    > fmovie-card
    > landing-view
    > login-view
    > modal-view
    > movie-card
    > movie-view
    > moviebox-view
    > nav-view
    > password-view
    > registration-view
    > search-view
    # App.css
    JS App.js
```

Navigation Bar

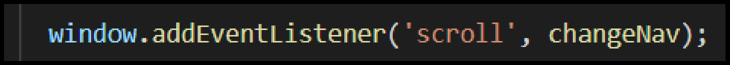
The navigation bar was set up as one component but has 3 changing parts based on useState implementation and Responsive Design

The state 'navbar' is set to false to show the default navigation bar at the top.



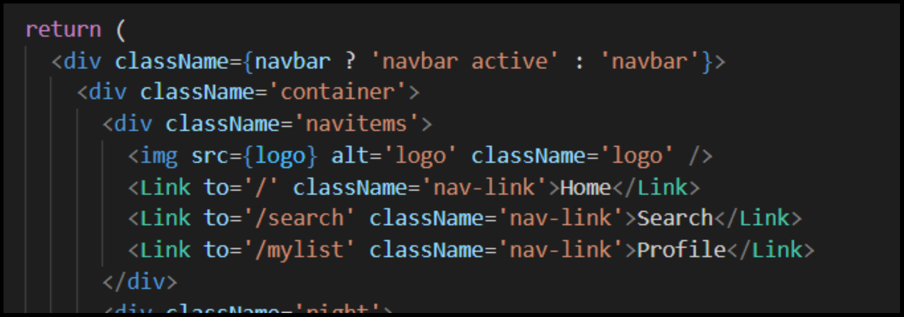
```
let [navbar, setNavbar] = useState(false);
let [hamburger, setHamburger] = useState(false);


let changeNav = () => {
  if (window.scrollY >= 750) {
    setNavbar(true)
  } else {
    setNavbar(false);
  }
}
```

The changeNav() is used to initiate a fixed black Navigation bar when the user scrolls down at 750px Y, setting the state to true.

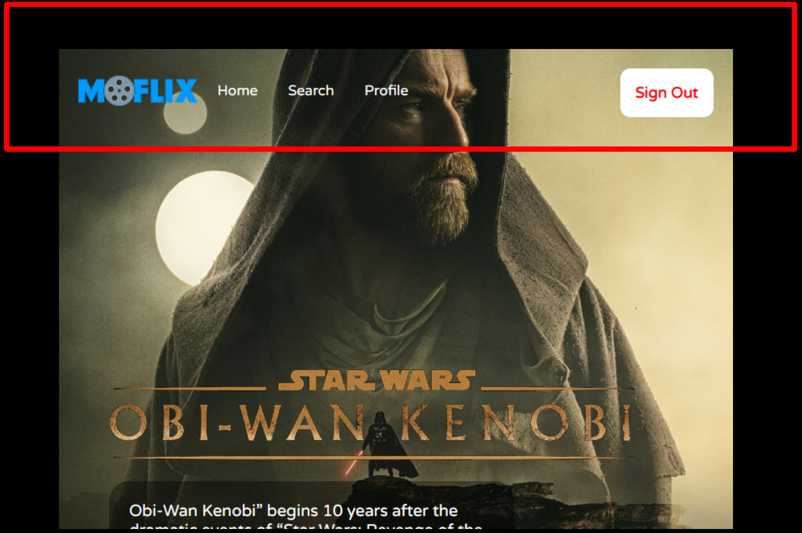


```
window.addEventListener('scroll', changeNav);
```

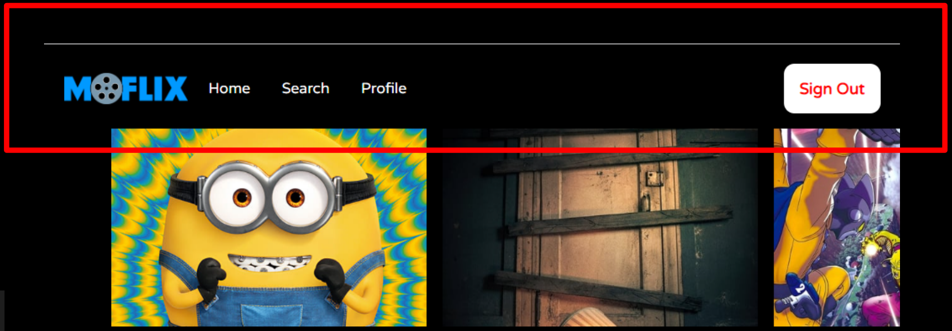
When false, the default navbar is set to class name '.navbar', when true, its set to class name by '.navbar active'



```
return (
  <div className={navbar ? 'navbar active' : 'navbar'}>
    <div className='container'>
      <div className='navitems'>
        <img src={logo} alt='logo' className='logo' />
        <Link to='/' className='nav-link'>Home</Link>
        <Link to='/search' className='nav-link'>Search</Link>
        <Link to='/mylist' className='nav-link'>Profile</Link>
      </div>
      <div className='right'>
```



Obi-Wan Kenobi” begins 10 years after the

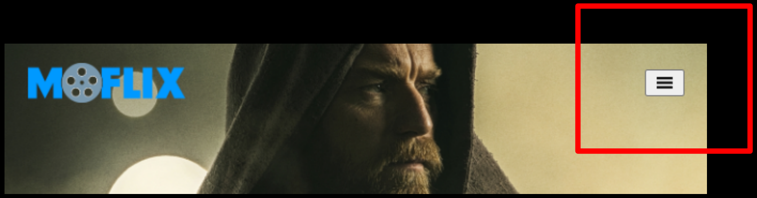


Hamburger Menu

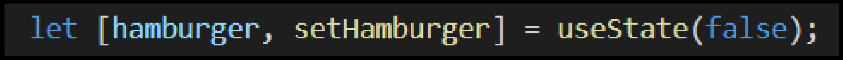
The hamburger menu was set up as one component butis divided in 2 groups.; the button and the actual menu. Based on useState implementation and Responsive Design

-The Button

At 600px X, the , '.hamburger' is changed to display flex. And displays an icon from a package library

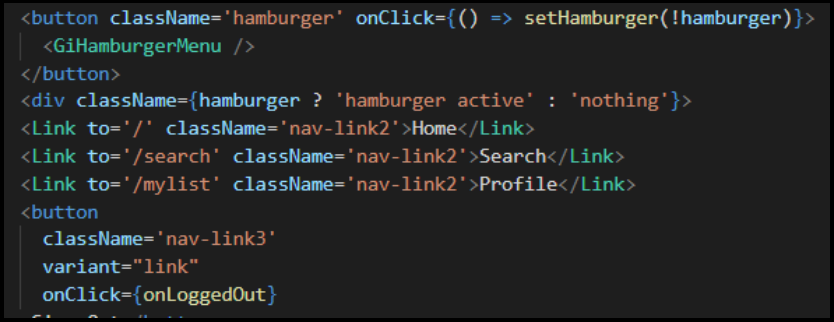


- The Actual Hamburger Menu

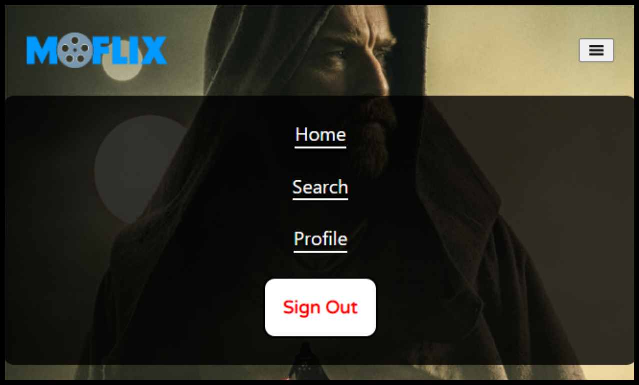


```
let [hamburger, setHamburger] = useState(false);
```

Controlled by the state 'hamburger' in useState. It is triggered on and off (true or false) by an onClick event.



```
<button className='hamburger' onClick={() => setHamburger(!hamburger)}>
  <GiHamburgerMenu />
</button>
<div className={hamburger ? 'hamburger active' : 'nothing'}>
<Link to='/' className='nav-link2'>Home</Link>
<Link to='/search' className='nav-link2'>Search</Link>
<Link to='/mylist' className='nav-link2'>Profile</Link>
<button
  className='nav-link3'
  variant="link"
  onClick={onLoggedOut}
```

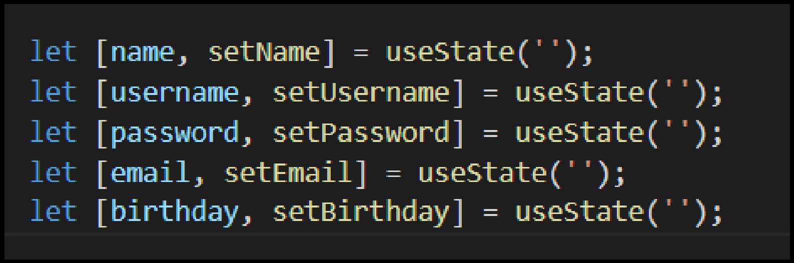


When false, the hamburger menu is set to className '.nothing'that displays none. When true, its set to '.hamburger active'

Registration & Login

The login and registration components were written in a similar way. A state , an axios post request and an input.

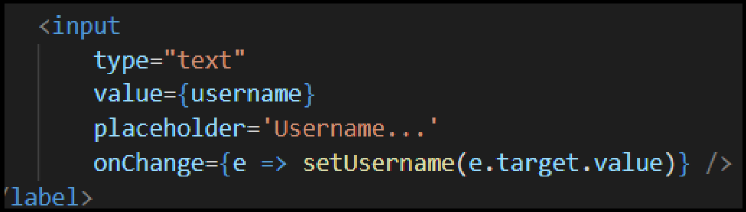
Each input value is set to a state



```
let [name, setName] = useState('');
let [username, setUsername] = useState('');
let [password, setPassword] = useState('');
let [email, setEmail] = useState('');
let [birthday, setBirthday] = useState('');
```

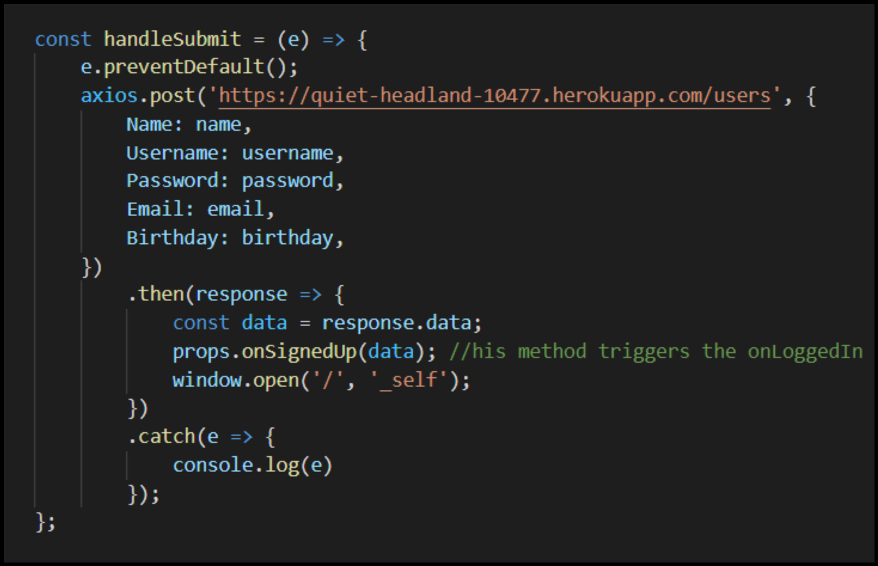
Each state is changed by on onChange

event in the rendering input.

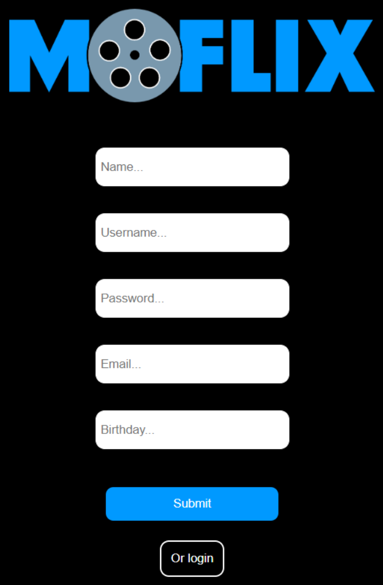


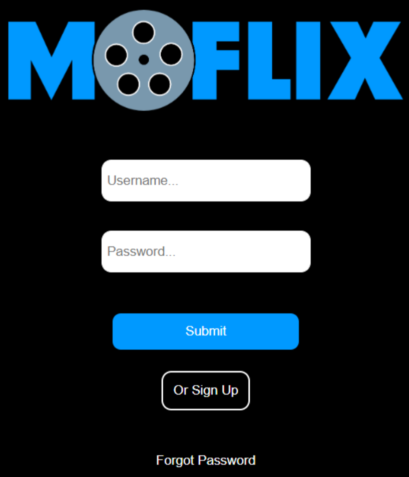
```
  <input
      type="text"
      value={username}
      placeholder='Username...'
      onChange={e => setUsername(e.target.value)} />
label>
```

The handlesubmit function initiates upon a click event, an axios post request is initiated to post the state values to the database to create a new user



```
const handleSubmit = (e) => {
    e.preventDefault();
    axios.post('https://quiet-headland-10477.herokuapp.com/users', {
        Name: name,
        Username: username,
        Password: password,
        Email: email,
        Birthday: birthday,
    })
        .then(response => {
            const data = response.data;
            props.onSignedUp(data); //his method triggers the onLoggedIn
            window.open('/', '_self');
        })
        .catch(e => {
            console.log(e)
        });
};
```

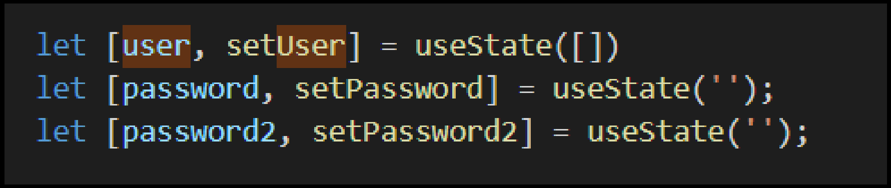




Forgot Password?

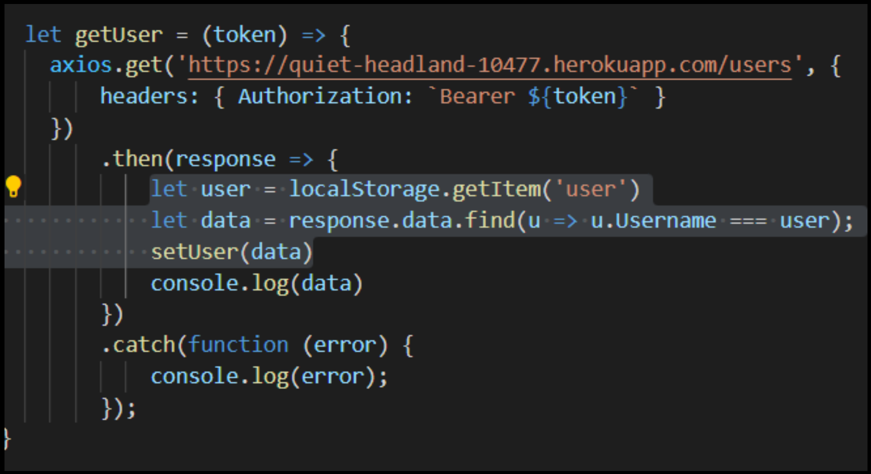
Similar to the login and registration components, the password component was written in a similar way. A state , an axios update request and an input. However, a request to get the user information is essential.

Each input value is set to a state



```
let [user, setUser] = useState([])
let [password, setPassword] = useState('');
let [password2, setPassword2] = useState('');
```

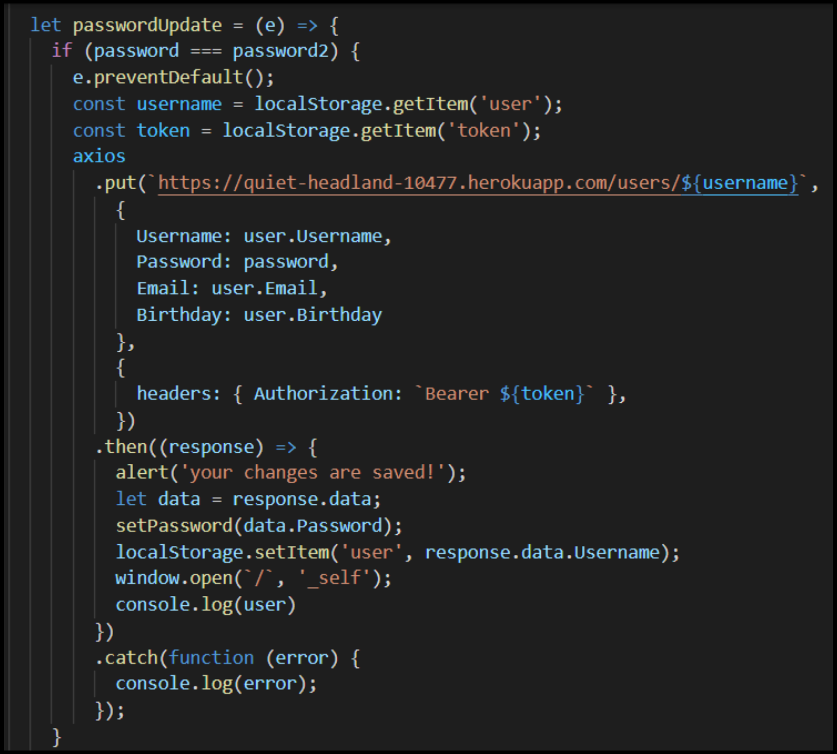
getUser() identifies the specific user that matches the username in the localstorage and returns the full data of the user from the user's collection from MongoDB. Fills empty array with user object.



```
let getUser = (token) => {
  axios.get('https://quiet-headland-10477.herokuapp.com/users', {
      headers: { Authorization: `Bearer ${token}` }
  })
      .then(response => {
          let user = localStorage.getItem('user')
          let data = response.data.find(u => u.Username === user);
          setUser(data)
          console.log(data)
      })
      .catch(function (error) {
          console.log(error);
      });
}
```

Getting all of the user's data is essential so that when an update request for the password is sent, the other data is unchanged.

Using the named saved as 'user' in localstorage as a template literal, a put axios request is sent using the state of 'password with the user data obtained from getUser() to update the user details



```
let passwordUpdate = (e) => {
  if (password === password2) {
    e.preventDefault();
    const username = localStorage.getItem('user');
    const token = localStorage.getItem('token');
    axios
      .put(`https://quiet-headland-10477.herokuapp.com/users/${username}`,
        {
          Username: user.Username,
          Password: password,
          Email: user.Email,
          Birthday: user.Birthday
        },
        {
          headers: { Authorization: `Bearer ${token}` },
        })
      .then((response) => {
        alert('your changes are saved!');
        let data = response.data;
        setPassword(data.Password);
        localStorage.setItem('user', response.data.Username);
        window.open(`/`, '_self');
        console.log(user)
      })
      .catch(function (error) {
        console.log(error);
      });
  }
```

Youtube Video Modal

The Modal component is imported into the Landing Page. And is controlled by a state

```
let [modal, setModal] = useState(false)
```

The state 'modal' is used control the React Video Player from the actual Modal Component. The modal state is carried to the modal component as a prop. When false, display none, when true display flex.

```
<button
 className='play'
 onClick={() => setModal(true)}>Play Trailer</button>
<Modal isOpen={modal} isClosed={() => setModal(false)}/>
```

isOpen is carried to the Modal component as a representation of the state of 'modal'. If the state of modal changes, so does isOpen

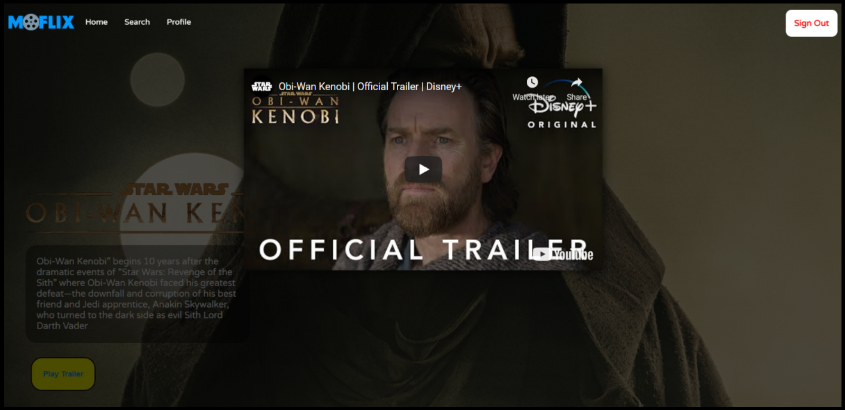
isClosed is carried to the Modal component as a function, when invoked on the otherside by an onClick event, it changes the state of modal on this component, to false

Modal unsets when user scrolls 550pxY, and is control by an event listener.

```
let unsetModal = () => {
  if(window.scrollY >= 550) {
    setModal(false)
  }
}

window.addEventListener('scroll', unsetModal);
```

if state of 'modal' in landing-view component is false return null, else return the React Player, youtube video.

```
let Modal = ({isOpen, isClosed}) => {

    if(!isOpen){ //if state of 'modal' in landing-view component is false
        return null
    } else {
  return (
    <div  className= 'overlay' onClick={isClosed}>
        <div className='modalContainer'>
       <ReactPlayer url='https://www.youtube.com/watch?v=3Yh_6_zItPU' className
        </div>
    </div>
  )
    }
}
```

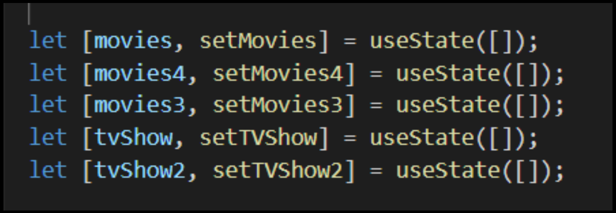


Obi-Wan Kenobi" begins 10 years after the dramatic events of "Star Wars: Revenge of the Sith" where Obi-Wan Kenobi faced his greatest defeat—the downfall and corruption of his best friend and Jedi apprentice, Anakin Skywalker, who turned to the dark side as evil Sith Lord Darth Vader

Movie List by Category

Every line of movie category is fetched by a different api link.

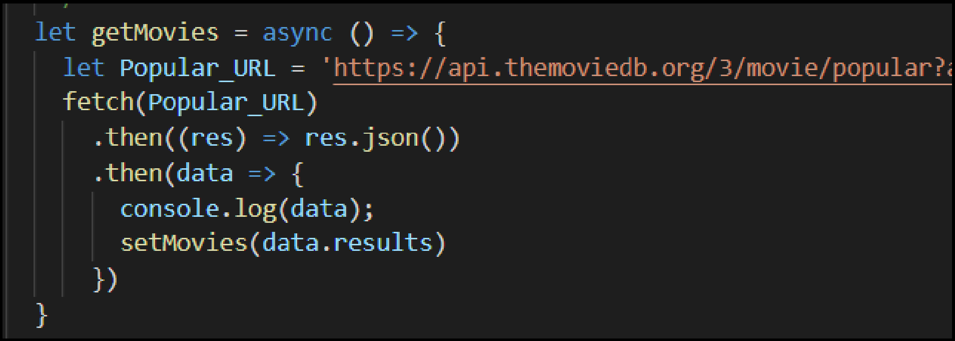
Every line of movie category is saved in a state



```
let [movies, setMovies] = useState([]);
let [movies4, setMovies4] = useState([]);
let [movies3, setMovies3] = useState([]);
let [tvShow, setTVShow] = useState([]);
let [tvShow2, setTVShow2] = useState([]);
```

getMovies() (has to be async function)

* Its a multi step process in which it needs to orderly execute
1. Fetches the api link
2. Saves the response in JSON format in a passing variable
3. pass the results to the setMovies function, fills the empty state array with data results
4. useEffect() is used similarily like componentDidMount()

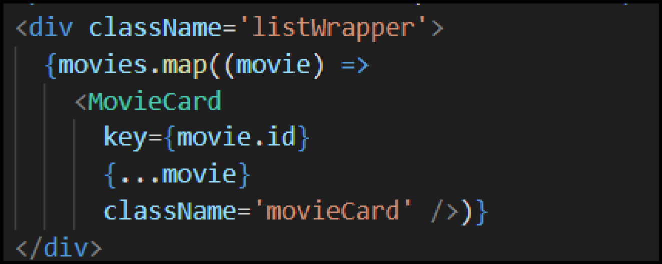


```
let getMovies = async () => {
  let Popular_URL = 'https://api.themoviedb.org/3/movie/popular?a
  fetch(Popular_URL)
    .then((res) => res.json())
    .then(data => {
      console.log(data);
      setMovies(data.results)
    })
}
```

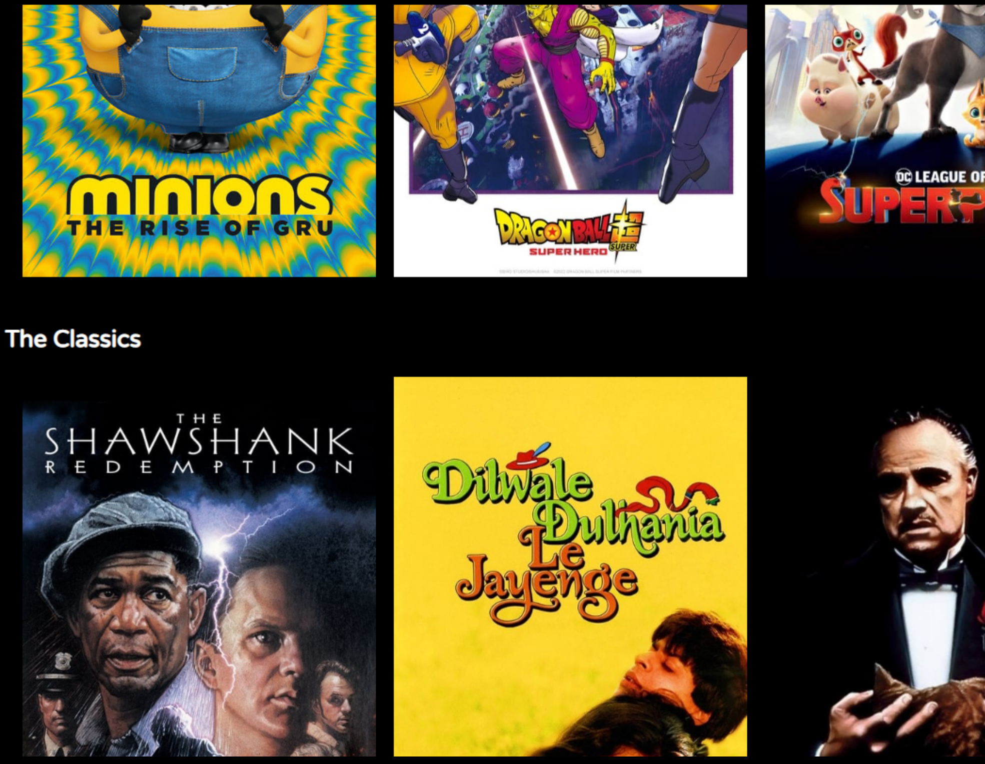
Every movie category is mapped out 1 by 1

* Each movie has a key(id)

* Sends each individual movie as an array object to the MovieCard component

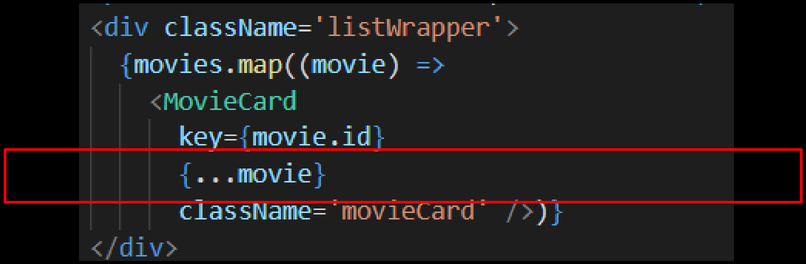


```
<div className='listWrapper'>
  {movies.map((movie) =>
    <MovieCard
      key={movie.id}
      {...movie}
      className='movieCard' />)}
</div>
```



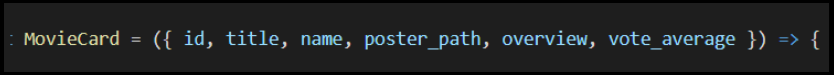
MovieCard

The data sent to this component is through the MovieView component.

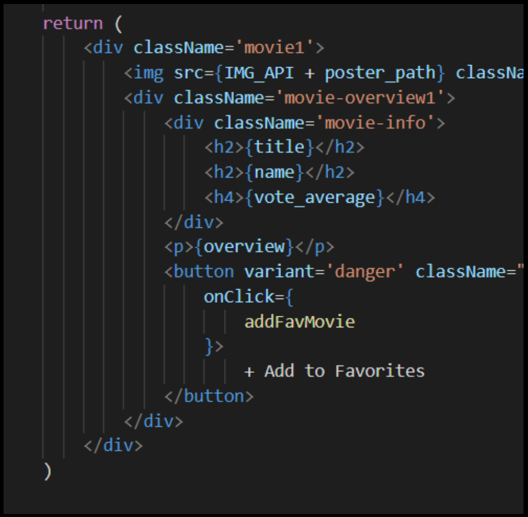


```
<div className='listWrapper'>
  {movies.map((movie) =>
    <MovieCard
      key={movie.id}
      {...movie}
      className='movieCard' />)}
</div>
```

The movie object array is transferred into this component as parameters



```
: MovieCard = ({ id, title, name, poster_path, overview, vote_average }) => {
```



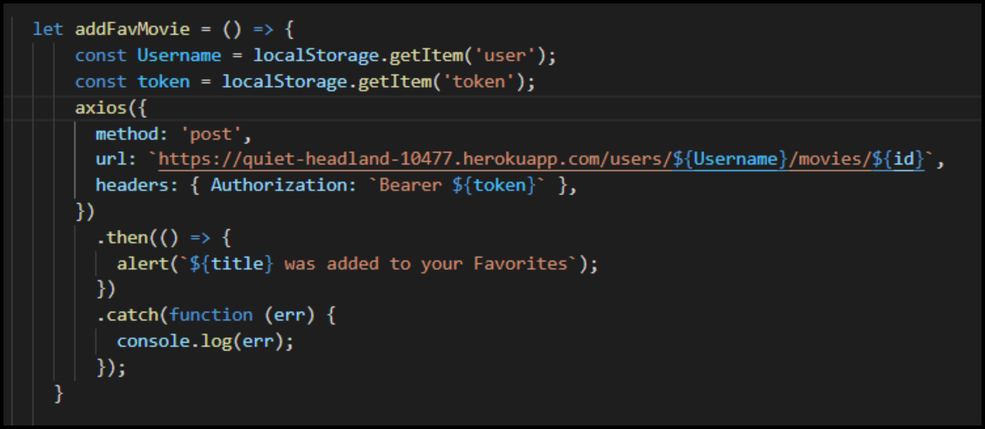
```
return (
    <div className='movie1'>
        <img src={IMG_API + poster_path} classNa
        <div className='movie-overview1'>
            <div className='movie-info'>
                <h2>{title}</h2>
                <h2>{name}</h2>
                <h4>{vote_average}</h4>
            </div>
            <p>{overview}</p>
            <button variant='danger' className="
                onClick={
                    addFavMovie
                }>
                    + Add to Favorites
            </button>
        </div>
    </div>
)
```

The addFavMovie() function adds a favorite movie to the favoriteMovies[] in the user object

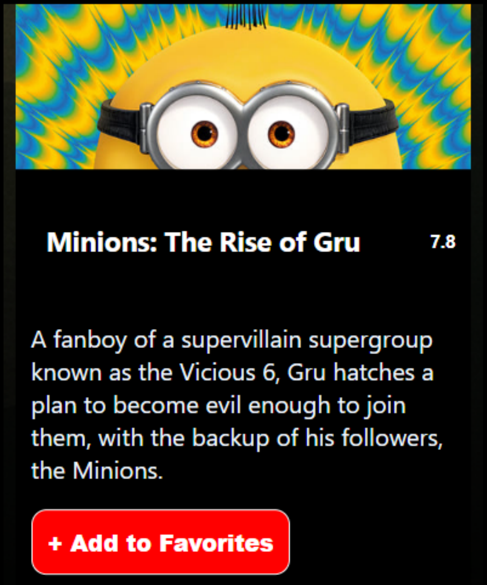
*The function takes the id of the movie from the parameter as a template literal and sending a post method via axios

*The function is invoked on an onClick event.

*The logic is the same for deleting a movie from the FavoriteMovies[] with instead a delete axios request



```
let addFavMovie = () => {
    const Username = localStorage.getItem('user');
    const token = localStorage.getItem('token');
    axios({
      method: 'post',
      url: `https://quiet-headland-10477.herokuapp.com/users/${Username}/movies/${id}`,
      headers: { Authorization: `Bearer ${token}` },
    })
      .then(() => {
        alert(`${title} was added to your Favorites`);
      })
      .catch(function (err) {
        console.log(err);
      });
  }
```



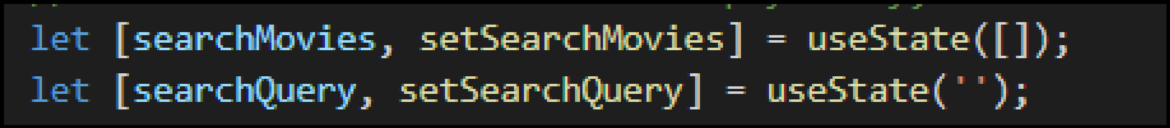
Minions: The Rise of Gru 7.8

A fanboy of a supervillain supergroup known as the Vicious 6, Gru hatches a plan to become evil enough to join them, with the backup of his followers, the Minions.

Search View

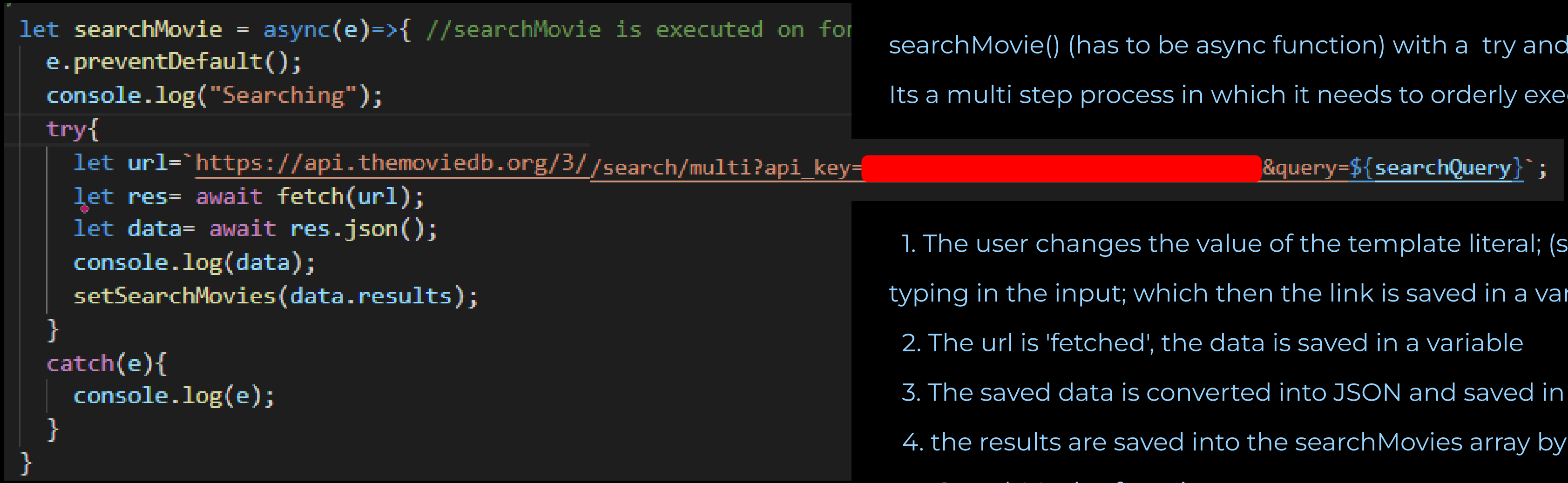
The search view is broken up into 2 parts; the search input and the search results. Each has their own state and logic.

* searchMovies is started as an empty array, which will later be replaced by movie data(setSearchMovies)'



```
let [searchMovies, setSearchMovies] = useState([]);
let [searchQuery, setSearchQuery] = useState('');
```

*searchQuery is representing the value of the input, which starts as an empty string



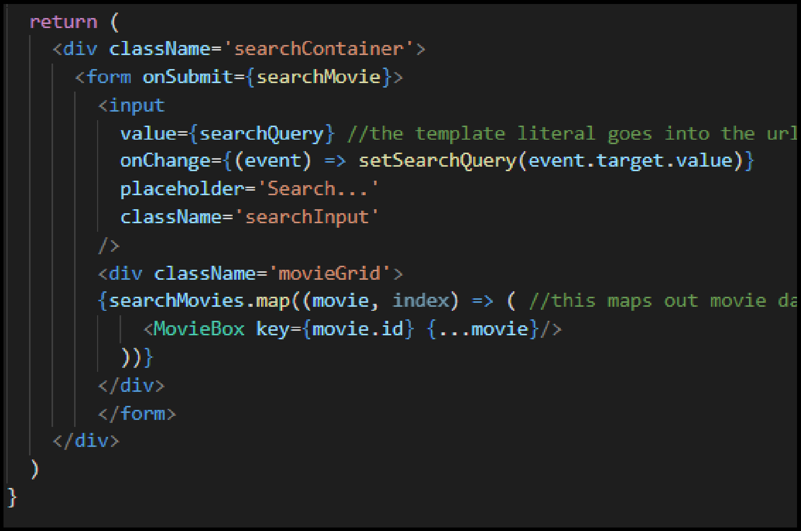
```
let searchMovie = async(e)=>{ //searchMovie is executed on for
  e.preventDefault();
  console.log("Searching");
  try{
    let url=`https://api.themoviedb.org/3//search/multi?api_key=[redacted]&query=${searchQuery}`;
    let res= await fetch(url);
    let data= await res.json();
    console.log(data);
    setSearchMovies(data.results);
  }
  catch(e){
    console.log(e);
  }
}
```

searchMovie() (has to be async function) with a try and catch. Its a multi step process in which it needs to orderly execute

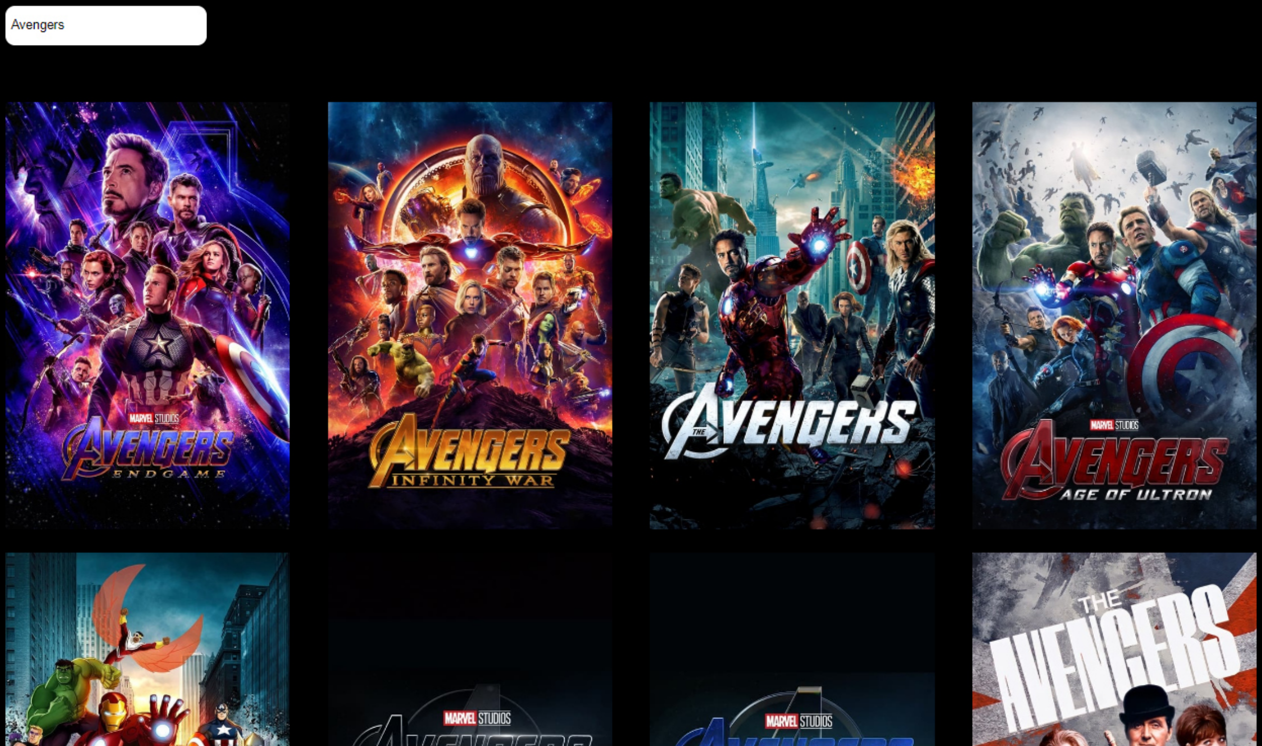
1. The user changes the value of the template literal; (searchQuery) by typing in the input; which then the link is saved in a variable
2. The url is 'fetched', the data is saved in a variable
3. The saved data is converted into JSON and saved in a variable
4. the results are saved into the searchMovies array by the sertSearchMovies function

Search View .... continued

In the rendering code, searchMovie() is executed onSubmit. the data from the searchMovies array is mapped out to display one by one.



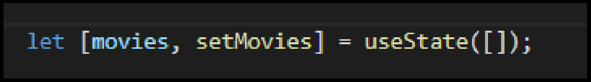
```
  return (
    <div className='searchContainer'>
      <form onSubmit={searchMovie}>
        <input
          value={searchQuery} //the template literal goes into the url
          onChange={(event) => setSearchQuery(event.target.value)}
          placeholder='Search...'
          className='searchInput'
        />
        <div className='movieGrid'>
        {searchMovies.map((movie, index) => ( //this maps out movie da
            <MovieBox key={movie.id} {...movie}/>
          ))}
        </div>
        </form>
    </div>
  )
}
```



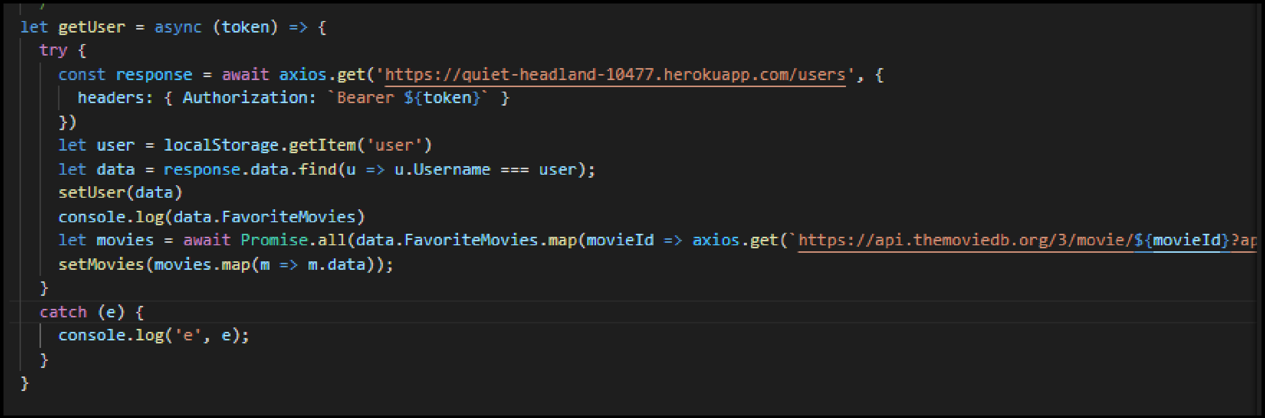
Avengers

Profile... Favorite Movie List

This component is intended to display the Favorite Movies list and delete the User. This first slide will explain how the Favorite Movies are rendered.



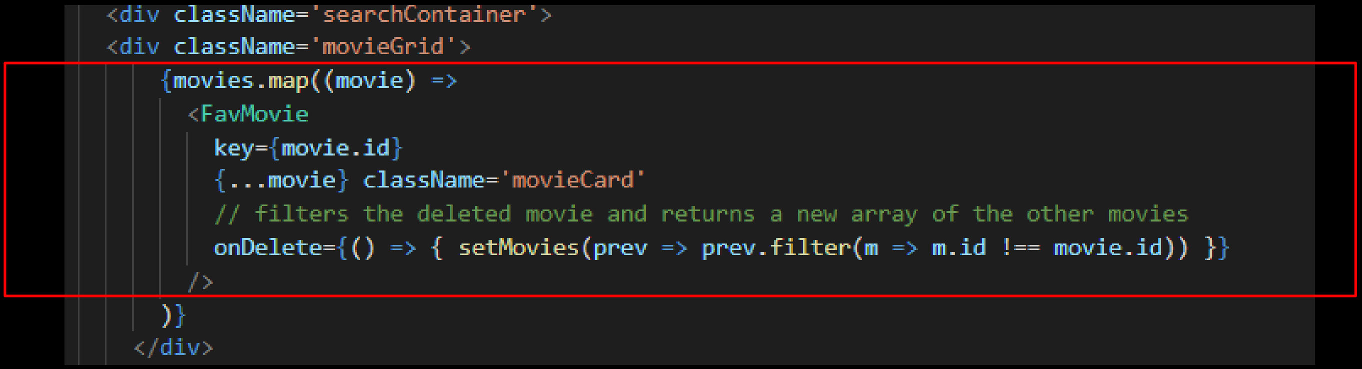
```
let [movies, setMovies] = useState([]);
```



```
let getUser = async (token) => {
  try {
    const response = await axios.get('https://quiet-headland-10477.herokuapp.com/users', {
      headers: { Authorization: `Bearer ${token}` }
    })
    let user = localStorage.getItem('user')
    let data = response.data.find(u => u.Username === user);
    setUser(data)
    console.log(data.FavoriteMovies)
    let movies = await Promise.all(data.FavoriteMovies.map(movieId => axios.get(`https://api.themoviedb.org/3/movie/${movieId}?ap
    setMovies(movies.map(m => m.data));
  }
  catch (e) {
    console.log('e', e);
  }
}
```

getUser() returns the user that is logged in, saves the users favoriteMovies into the state of 'movies'

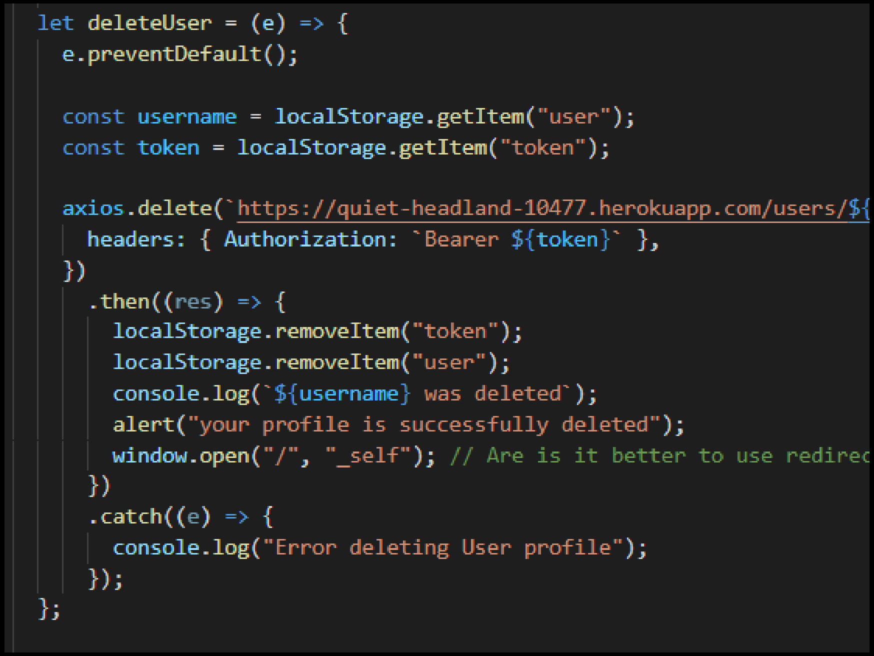
1. Maps out the FavoriteMovies array to return each movie one by one and save it in a variable
2. The movies variable is mapped out into an array and saved in the setMovies function



```
<div className='searchContainer'>
<div className='movieGrid'>
    {movies.map((movie) =>
      <FavMovie
        key={movie.id}
        {...movie} className='movieCard'
        // filters the deleted movie and returns a new array of the other movies
        onDelete={() => { setMovies(prev => prev.filter(m => m.id !== movie.id)) }}
      />
    )}
  </div>
```

Profile… Delete a User

This component is intended to display the Favorite Movies list and delete the User. This second slide will explain how logic for deleting the user is written and executed.



```
let deleteUser = (e) => {
  e.preventDefault();

  const username = localStorage.getItem("user");
  const token = localStorage.getItem("token");

  axios.delete(`https://quiet-headland-10477.herokuapp.com/users/${
    headers: { Authorization: `Bearer ${token}` },
  })
    .then((res) => {
      localStorage.removeItem("token");
      localStorage.removeItem("user");
      console.log(`${username} was deleted`);
      alert("your profile is successfully deleted");
      window.open("/", "_self"); // Are is it better to use redirec
    })
    .catch((e) => {
      console.log("Error deleting User profile");
    });
};
```

deleteUser() deletes a user

* sends a delete request via axios to delete the user

* identifies the username as a template literal by retrieving the username from localstorage

* .then() removes the user from localstorage

* initiates on an onClick event

MOFLIX

Movie App

In Conclusion

Building this client side app taught me a few things

- Axios requests
- How to fetch API data and display it as a user interface by mapping out the data
- Working with a UseState()
- Event listeners
- React Router
- Modal