24/03/24 **Netflix- GPT** Episode - 14

→ npx create-react-app netflix-Gpt

→ Setup Tailwind css.

★ Features :-

- → Login → Sign In/ signup Form → Redirect to Browse page
- → Browse Page (After Authentication)
  - → Header
  - → Main Movie
    - → Trailer in background
    - → Movie Title / Description
    - → Movie Suggestions
      - → Movie List x n

{Also create Home page In home page give a button to login/signup} button is then redirect to login page.

→ Netflix-GPT
- → Search Bar
- → Movie Suggestions

shortcut for making functional component:

→ rafce → enter (React arrow function component export)

25/03/24 → HAPPY HOLI

28/03/24 ★ use Ref.

★ useRef is an inbuit Hook given by react.

★ His HOOK is used to create a reference of a value that does not need for rendering.

→ How to use

```
import {useRef} from "react";

const name = useRef(null);
```

$$\underbrace{\texttt{null}}_{\text{initial value.}}$$

→ How to use with input tags to create a reference of it

```
<input
  ref= {name},
  Placeholder = "name"
/>
```

→ useRef returns an object named as current, inside current we get value in value key.

```
console.log (name.current.value);
```

27/05/24

Back after a 2 month long break because of some oncampus and off campus hirings, farewell & my end semesters.

→ Today i revised the code of Netflix GPT & also done some changes in styling.

28/05/24 → Regex stands for regular expression.

→ We have used regex for email & password validation

→ It have a .test() function & returns True or false

```
const pass = "{ /        /}".test(Password);
```

↑ Password entered by user or created by user

It will be true if all the requirements matched like no. of alphabets, upperscase, lower case, number, special character etc.

It will be false if any of the given conditions does not matched.

29/05/24 done language translation using react-i18-Next

30/05/24 ★ Starting Authentication

→ After setting up your account & your project on firebase. Follow the commands shown there
- → Install firebase to your project npm install firebase
- → Make a firebase.js file & copy paste given code.
- → Install firebase CLI. npm install -g firebase-tools.
- → firebase login
- → firebase init
- → firebase deploy

★ useNavigate() :- A HOOK given to us by react-router-dom

import { useNavigate } from react-router-dom

It is basically used to Navigate user from one page to another.

const navigate = use-Navigate();

navigate("/browse");

→ navigate can be used only inside Router Provider. ★★

01/06/23 ★★★★★★

→ We cannot call any custom hook or even normal ~~fun~~ hook (inbuild hook) inside a normal function.

~~or~~ "

→ It can only be called inside react functional component or react custom hook.

15/06/24

★ Some more HOOKS

① useMemo( ); → It is basically used to optamize our react application. It returns a memorized value.

Syntax: Same as useEffect

```
const val = useMemo( () => {
        return solvefunc(myNum)
        }, [dependency]);
```

} whenever anyvariable in dependency array changes this callback fun will be called

→ We can not send any argument to our ~~a~~ callback function of useMemo HOOK.

~~★ Whenever we want to memorize any value or any complex computation & store into variable we use useMemo hook, it reduces unwanted re-renders.~~

② usecallback ( ): It is also used to improve the performance of our react application & it returns a memorized function.

→ In usecallback's call back function we can send any no of arguments.

```
~~a~~ const todo = useCallback( () => {
        do something}, [todo]);
```

When to use useMemo ?

→ Suppose in our component we have a complex computation & some state variables, whenever any variable is getting changed then that complex computation is getting re computed, due to which is it slowing down the overall performance of our app.

→ To resolve this we can use useMemo Hook & we can put the computation inside the callback function of the useMemo & a dependency array, whenever the value present into dependency changes then only that computation will be computed.

→ Similarly when we want to call ~~a~~ any function on the changes of any specific variable then we use useCallback.

③ useReducer ()

<u>**Syntax:**</u> const [state, dispatch] = useReducer (reducer, ~~[illegible]~~ initialState)

↓ current state.
↓ dispatching an action
↓ reducer function
↓ Inital value

→ we need to declare reducer function & initialState.

const initialState = 0;

```
const reducer = (state, action) => {

    if (action.type = "Increase") { return state+1 }
    if (action.type = "decrease") { return state-1 };



    return state;                    } reducer func always
                                     } return something.
}
```

```
<button onclick={() => dispatch ({type : "Increase"})}>

<button onclick = {() => dispatch ({type : "decrease" })}>
```

use case: ~~Instead of using alot of different different state variables~~ when we have complex state logic give example of increase & decrease.

→ watch SSR vs CSR in yt

Difference between import ReactDOM from "react-dom/client"; and import ReactDOM from "react-dom";

While both import statements import modules from the React DOM library, the first one specifically targets the client-side rendering APIs, whereas the second one imports the entire ReactDOM package, which encompasses both server-side and client-side rendering functionalities. The choice between them depends on your specific requirements and the features you need to utilize in your application.

Server-side rendering (SSR) and client-side rendering (CSR) are two different approaches to how web pages are generated and displayed to users:

1. Server-side Rendering (SSR): In SSR, the HTML content of a web page is generated on the server in response to a user request. The server processes the request, executes any necessary code (such as fetching data from a database or running business logic), and generates a complete HTML page. This HTML page is then sent to the client (web browser), where it is rendered and displayed to the user. The user receives a fully-formed page ready for display without any additional processing required on the client side.

2. Client-side Rendering (CSR): In CSR, the HTML content of a web page is generated dynamically on the client side, usually by JavaScript code running in the user's web browser. When a user requests a page, the server sends a minimal HTML document along with JavaScript code. The browser then downloads and executes this JavaScript code, which fetches data from APIs and dynamically updates the HTML structure of the page based on the data received. This process is often referred to as "hydration." The user sees a blank page initially, which is then populated with content as the JavaScript executes and updates the DOM (Document Object Model) in the browser.

Key Differences:

- Initial Load Time:

  - SSR typically results in faster initial load times because the server sends a pre-rendered HTML page to the client, which can be displayed immediately.

  - CSR may have slower initial load times because the browser needs to download and execute JavaScript code before rendering the page, especially if the page relies heavily on client-side rendering for content generation.

- SEO (Search Engine Optimization):

  - SSR is generally better for SEO because search engine crawlers can easily parse the fully-formed HTML content sent by the server.

- CSR can pose challenges for SEO because search engine crawlers may have difficulty parsing dynamically generated content rendered client-side.

- Performance:

- SSR may offer better perceived performance for users with slower internet connections or less powerful devices since they receive a fully-rendered page from the server.

- CSR can provide smoother interactions and faster subsequent navigations within the application once the initial JavaScript is loaded, as it can fetch and update content dynamically without requiring full page reloads.

Both SSR and CSR have their advantages and are suitable for different use cases depending on factors such as performance requirements, SEO considerations, and the nature of the application being developed.

::after pseudo selector

The ::after pseudo-element should work on an image element (<img>), but there are certain considerations to keep in mind:

1. Content requirement: The ::after pseudo-element is used to insert content after the content of the selected element. Since an <img> element doesn't have any inherent content (like text), you need to ensure there's a parent element wrapping the <img> to which you can apply the ::after pseudo-element.

2. Positioning context: Ensure that the parent element of the <img> has a non-static positioning context (i.e., relative, absolute, or fixed). This is necessary for the ::after pseudo-element to be positioned correctly relative to its parent.

How to apply css property with setState() show and hide

```
<div
     className={`mt-2 text-sm transition-opacity duration-300 ease-in-out ${
            show ? "opacity-100" : "opacity-0"
          }`}
```

- ➔ We are writing JSX that’s why we are using {} in className
- ➔ We are using template literals (``) because we are using a state variable to add in a string on a condition. Whenever we want to alter/change our string depending upon any variable we use template literals