Higher Order Component

Episode 11th

↳ It is JS function that takes a component as an input enhances it and returns a component.

(1) Make higher order component to add promoted tag on top of the card

ex) in RestaurantCard.js — takes input

```
export const withPromotedLabel = (RestaurantCard) => {
    return () => { <-- It returns a component
        return (
            <div>
                <label> Promoted </label>
                < RestaurantCard />
            </div>
        );
    };
};
```

Using it — Body.js

```
import RestaurantCard, {withPromotedLabel} from "./RestaurantCard";

const RestaurantCardPromoted = withPromtedLabel(RestaurantCard);
```

(2) In body.js add the logic so when to use this new RestaurantCard Promted

→ Also pass the props

→ and also use that props in the ~~Restaurant~~ RestroCards.js

(3) add tailwind to it to make is overlap

className → "absolute bg-black text-white m-2, p-2 "

Data is dynamic UI is static

{["cards"] same as cards} in JS

→ Building all the dishes menu category wise
ex- Recommended, Starters etc.

So we are filtering out the data of a particular type

(1) filtering all the data of @type : . ItemType
(2) add tailwind to the Restraunt Menue card
(3) Building accordium [Recommended ▼] [Category Accordian]
to add all dishesh group wise
(4) New file RestaurantCategory.js for accordian
(5) again a component for ItemList.js ← for a particular accordian
(6) Not to make it work on click
Make a usestate

```
const [showItems, setShowItems] = useState(false);
```

and give `onClick = {handleClick}` ← to the header dive

```
const handleClick = () => {
    setShowItems(!showItems); // for toggle
}
```

```
<div>
    {showItems && <ItemList items={data.itemCards}/>}
```

[Now we are adding a feature such that if we open one Accordian other all should close]

Controled Components

↳ So these are the components those who have their own control in their function.

Vs Controlled Components

↳ These are those whose control is in their's parents component

(1) in Restaurant Category.js we are creating a use state

const [showIndex, setShowIndex] = useState()

(2) And to modify in the parent through the child we pass the setShowIndex to the child so it can controll it.

(3) when that handleClick() function will be called the setShowIndex will be changed

(4) The logic in Parent component map function is

showItems = { index === showIndex ? true : false }

↑ this can be got from map function

This feature is called as Lifting State UP

[this is not coded in code]

Props Drilling

① <component>
② <component>
③ <component>

So if component 1 has some data that is to needed by component ③ or ④ and more level down.

So we need to pass props to each component and nither the middle component needs the data but they are just passing them.

This become a problem which is called as props drilling

It just like having a Global variable
So React gives us a Super Power createContext that can be used any where

→ So we will create a context in Utils/UserContext.js

→ So we are creating a feature that every page needs
→ that is: Is user is logged in or not

UserContext.js

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

const UserContext = createContext({
        loggedInUser : "Default User",
});

export default UserContext;
```

Now it can be accessed from anywhere in the app.

To access it react give us a Hook

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

const data = useContext(UserContext);
```

* In class Based components Hooks cannot be used

To use context in class component we need to write

```
<div> <UserContext.Consumer>
        {(data) => console.log(data)}
      </UserContext.Consumer>
</div>
```

↑ this will return a object in which all the context data will be stored.

* To update the UserContext
→ ① import UserContext from "./utils/UserContext"
② Wrap the whole root app function in a tag

```
<UserContext.Provider value={{loggedInUser : "Mandav"}}>
    <div className = "app">
        <Header/>
        <Outlet/>
    </div>
</UserContext.Provider>
```