

Higher Order Component

Episode 114

↳ 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

```
export const withPromotedLabel = (RestaurantCard) => {  
  return () => { ← takes input  
    return (  
      <div>  
        <label> Promoted </label>  
        <RestaurantCard />  
      </div>  
    );  
  };  
};
```

It returns a component

Using it Body.js

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

~~const~~

```
const RestaurantCardPromoted = withPromotedLabel(RestaurantCard);
```

② In body.js add the logic to when to use this new RestaurantCardPromoted

→ Also pass the props

→ and also use that props in the Restaurant RestaurantCard.js

③ add background to it to make is overlap

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

* Data is dynamic UI is static {["cands"]} 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 Restaurant Menu card.

3) Building accordion Recommended ▾ [Category Accordion]
to add all dishes group wise

4) New file RestaurantCategory.js for accordion

5) again a component for ItemList.js ↖ for a particular accordion

6) Not to make it work on click

Make a usestate

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

and give onClick = {handleClick} ← to the header div

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

<div>

{showItems ? <ItemList items={data.itemCards}/>}

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

Controlled Components

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

Uncontrolled 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 control it.

③ When that handleClick() will be called the setShowIndex ^{function} will be changed.

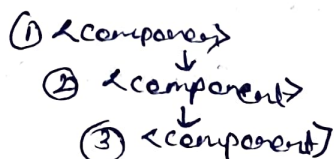
④ The logic in parent component map function is

`showItems = { index === setShowIndex ? 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



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

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

This became a problem which is called as props drilling

It's just like having a Global variable
So React gives us a Super Power create Context that can be used anywhere

→ 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 gives us a Hook

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

```
const data = useContext(UserContext);
```

* In class based components ~~we~~ 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 try

```
<UserContext.Provider value={{ loggedInUser: "Manday" }}>
```

```
  <div className="app">
```

```
    <Header/>
```

```
    <div>
```

```
      <UserContext.Provider>
```