React (Day-11)

Higher Order Component (HOC)

It takes a component as an input and returns a component.

↳ entrance the component and returns a modified component

→ Let's say we have "isPromoted" in our restaurant api data, and we want to display a Promoted label on all the promoted restaurants

→ So, we will create a HOC which will take <RestaurantCard /> and return another component with label (promoted).

Ex: export const withPromotedLabel = (RestaurantCard) => { ↳ (HOC) ↳ Takes a component as arg

```
return () => {
    return (
        <div>
            <label>Promoted</label>
            <RestaurantCard />
        </div>
    );
};
```

↳ returns enhanced component

Now, in our body let's import this HOC

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

const RestaurantCardPromoted = withPromotedLabel(RestaurantCard);

{restaurant.data.promoted ? (<RestaurantCardPromoted resData={restaurant} />)
    : (<RestaurantCard resData={restaurant} />)}
```

We gave resdata as prop to our promoted component, but where are we receiving it in Promoted component?

Ans: export const withPromotedLabel = (RestaurantCard) => {
return (props) => {
return (

<label>Promoted</label>
<RestaurantCard {...props} />

);
};
};

(passes all the props that are received)

(component returns just JSX)

(HOC returns just another component)

* All react apps basically have 2 layers

↳ UI Layer ← Data layer powers UI layer
↳ Data Layer

It's all about how to manage data in our app, which will make our app super fast & efficient.

UI layer — mostly JSX
Data layer — state, props, local variables, the logic part in JSX, hooks, etc.

Controlled Vs Uncontrolled components

Controlled: Component which is controlled by another component. (Basically, when a child component is controlled by parent component)

Uncontrolled: Component which controls / manages itself with its own state.

Passing data to child components

How? Props obv!

But let's say we have some data at parent component, and there's a child component that needs the data, so you have to drill the props through lot of intermediate components to reach the child component (Given the project has big tree structure)

→ Here is where we introduce React Context

So, Problem: Props drilling
Solution: Context

→ Where do you think context can be used?

Ex:1 maintaining a login state which might be useful for a lot of components which displays dynamic data based on the status of login.

Ex:2 Maintaining a theme: Dark / Bright is also needed by all the components to set their theme.

→ Since context is a global state, we won't keep our context file in components. (Not Good)

↳ we will place it in utils.

Let's say UserContext (Login / Logout)
(or) (User details)

→ So, context can be created using createContext which is provided by 'react' library.

→ createContext takes in data as argument.

Ex: import { createContext } from 'react';

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

→ Now you can access this context in any component with a useContext() hook which takes the context as argument and returns data.

Ex: import { useContext } from 'react';
import UserContext from "../utils/UserContext";

```
// Inside component
const { loggedInUser } = useContext(UserContext);
// Inside component
```

→ this data now can be used inside the component.

→ Should we be using context everywhere to pass data to components?

Not necessarily. Whenever you feel there is a need for the data to be used in multiple components, that is the case you should be using context.

→ Not for just passing down the props by one level.

→ How do we use context in class based components?

React provides 'Consumer' which is an alternative for hooks. So CBC can use Consumer to access the context since CBC can't use hooks.

Ex: import {UserContext} from "../utils/UserContext";

render() {

return (

Use wherever there is need of data

<UserContext.Consumer>

Callback function inside consumer which gives access to context data.

{({loggedInUser}) => (

<h1>{loggedInUser}</h1>

)}

</UserContext.Consumer>

);

}

→ So context can be consumed either by using

* hooks
* Consumer

→ Can we update the context data from the default value to a new value?

Yes, we can, and we can do that using 'Provider'.

→ let's say in our app, we get userName after handling authentication logic, now we have to send updated context to all child props, but how?

⇒ Just wrap all the components that need this updated context with 'Provider'.

Ex: App.js

```
return (
  <UserContext.Provider value={{ loggedInUser: userName }}>
    <div className="app">
      <Header />
      <Outlet />
    </div>
  </UserContext.Provider>
);
```

updated value

Now all the components in our app will receive the updated context, since we have wrapped our whole app inside context provider.

→ What if we do sth like this?

```
return (
  <UserContext.Provider value={{ loggedInUser: userName }}>
    <div>
      <UserContext.Provider value={{ loggedInUser: "Vijay" }}>
        <Header />
      </UserContext.Provider>
      <Outlet />
    </div>
  </UserContext.Provider>
);
```

default value → (userName)

<Header /> → ("Vijay")

→ So the components outside App, would get the default value of context.

→ Components inside App would get username as context value.

→ Header component would get "vinay" as context value.

* So it depends on which components we are wrapping our context provider with.

→ But how do we update the loggedInUser from anywhere in the components.

⇒ We have defined state in App.js.

const [username, setUsername] = useState()

So, we can send the setUsername inside the Provider as value, which can be accessed by the child components to modify the username.

Ex: <UserContext.Provider value={{loggedInUser: username, setUsername}}>

* Basically we have binded our UserContext with the state variable in App.js.