Appointments

Episode - 11 Data is New Oil

=> Higher Order Components :- Higher Order components that takes a component and returns a new component that wraps original. It takes component and enhances ~~and~~ it and returns a new component.

Suppose we have to promote some restaurant so we will give our card to higher order component & then give it promoted label & return a new component.

Syntax :-

```
const EnhancedComponent = (component needs to be enhanced)
    => { return () => {
            return jsx of enhanced component
        }
    }

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

Appointments

So, Higher Order component takes a component as input and returns a new component (function) that returns a piece of jsx with it.

We it like this

```
const RestaurantCardPromoted = withPr
                                Label(R
// Pass props
{
  isPromoted ? (<RestaurantCardPromoted
                  resData={resData} />)
             : (<RestaurantCard
                  resData={resData} />)
}

// Receiving Props
with
export default withPromotedLabel = (RestaurantCard)
    {
        return (props) => {
            return (
                <div>
                  <label>Promoted</lab
                <RestaurantCard {...props
                </div>
            );
        };
    };
```

w-6/12 is half width of page in tailwind

25 SATURDAY
Week 43 Day (298-067)

OCTOBER

O	M	T	W	T	F	S	S	M	T	W	T	F	S	S
C			01	02	03	04	05	06	07	08	09	10	11	12
T	13	14	15	16	17	18	19	20	21	22	23	24	25	26
25	27	28	29	30	31									

Appointments

We are not changing the Restaurant card component we are returning the same component with some modification

All react applications have 2 layers

1. Data Layer
2. UI Layer

UI Layer is powered by Data Layer, means props, state, local variables and JavaScript code all include in Data Layer. So, UI Layer is contain JSX code.

The @type is a valid JS object or variable so to use this in JS we do

c.card?.card?.["@type"] = something

=) React Dev Tools

26 SUNDAY

React Developers Tools is good for react developers it can show you all the components & their hierarchy. It can also show means almost the virtualDOM This is very-very useful for debugging.

=) Now we want to build a feature that if we open an accordian then all other accordians will be closed automatically.

2025

Restaurant Menu (Parent)
↓
RestaurantCategory (child)

Appointments For this, we have to give power to show the category list to the parent of RestaurantCategory.
Because right now every category has its own state variable instead Restaurant Menu will control state of every category.

Controlled & Uncontrolled components.
Here now RestaurantMenu is controlling their children (RestaurantCategory), so it is called Restaurant controlled component and when RestaurantCategory has its own state (showList) then it was uncontrolled component.

Lifting state up → It means passing the state from lower level (child) component to a higher level (parent or commonancestor) component in the component tree.
TO achieve we will use the state variable and pass both variable & state function as children.

RestaurantMenu.js

```
Const [showIndex, setShowIndex] = useState(null);

<RestaurantCategory
    showIndex={false}
    setShowIndex={() => setShowIndex()}/>
```

Appointments

09

RestaurantCategory.js

```
Const RestaurantCategory = ({showIndex, setShow...
  =) {

<div
const handleClick=() => {
setShowIndex();
}

return <div onClick={handleClick}>
  showIndex && (
    <div>
      Item List
    </div>
  )
</div>
);
```

Here what RestaurantMenu is doing is creating a setShowIndex function for every category

For index:0
setShowIndex = () => setShowIndex(0);
For index 1
setShowIndex = () => setShowIndex(1)
and so on.

Appointments

09 Child doesn't care about index. Closure remembers the index
() => setShowIndex(index)

10 Javascript remembers Index in closure
So, when child calls, Javascript run the setShowIndex for the specific function

11 Suppose for index: 2
Local:
setShowIndex f ()

12 Closure (RestaurantMenu.js)
index: 2
setShowIndex f ()

13

14 When only one accordion opens
Because in the parent.

15 ShowList = {showIndex == index}
Only one index can open at a time.

16 Click another category to update showIndex.
So, child is controlled by parent as it cannot open itself it only open when parent says showList == true.

17 One-line :-
~~The parent already knows the index while rendering and passes a closure bound handler~~
Sometimes you want the state of two components always change together. To do it remove the state from both of them, move it to their closest common parent then pass it down to them via props. This is known as lifting state up. This is also called single source of truth.

NOVEMBER
DECEMBER
2025

30
Week 44

THURSDAY
Day (303-062)

OCTOBER

O	M	T	W	T	F	S	S	M	T	W	T	F	S	S
C			01	02	03	04	05	06	07	08	09	10	11	12
T	13	14	15	16	17	18	19	20	21	22	23	24	25	26
25	27	28	29	30	31									

Appointments

09 To make the category close by clicking again

10 we will do

~~() setShowIndex~~

11 SetShowIndex={()=> setShowIndex(ShowIndex===
index ? null : index)}

12

So, if ShowIndex has index value already
13 then make it null else give it
index.

14

15 # Props Drilling

16 React has one-way data flow
we have pass the data from parent
17 to children to their children & so on.

Restaurant Menu (it has data=dummy)
↓ dummy
Restaurant Category (Child of Restaurant Menu)
↓ dummy
Items List (Child of Restaurant Category)

But what if we had hundreds of components
and if I want to pass the data from
2025 Restaurant Menu to Item List

then I have to make it go through all those components which is not using this and it's not a good way and this concept is called props drilling

To overcome this we use React context so we can use some global component which can be accessed everywhere without using props. It is like a central store which keeps data

Let's create a userContent to store information of logged in user.

userContent.js

```
import { createContent } from "react"

const userContent = createContent ({
    loggedInUser : "Default User",
});
```

Using it (we will useContent() hook to use it)

```
const {loggedInUser} = useContent(UserContext);
```

We cannot use hooks in class based components. So, to access it we do.

import userContent fecom "./userContent.js"

Appointments

09 <UserContent.Consumer>
{(data) => console.log(data)};
10 {(loggedInUser) => <h1>{loggedInUser}</h1>}

11 <UserContent.Consumer/>

12 How to change userContent

13 Suppose there is an api which gives us
14 userInfo and give the data and change the content.

15 We will use Content Provider to update the value of the userInfo.

16

const applayout = () => {
17 const [userName, setUserName] = useState("")
useEffect(() => {

2 SUNDAY

const data = {
name: "Akshay Saini",
};
setUserName(data.name);
}, []);

return (
<UserContent.Provider value={{loggedInUser:
App
</UserContent.Provider>
);}

Action Plan

Schedule your plan .....For Better future.

November 2025

Wk	44	45	46	47	48
MON			17	24	
TUE			18	25	
WED		5	12	19	26
THU		6	13	20	27
FRI		7	14	21	28
SAT			15	22	29
SUN	2	9	16	23	30

<UserContent.Provider value={{loggedInUser: userName}}>
<App>
<UserContent.Provider value={{loggedInUser: "Elon Musk"}}>
<Header />
<UserContent.Provider />
<App />
<UserContent.Provider />

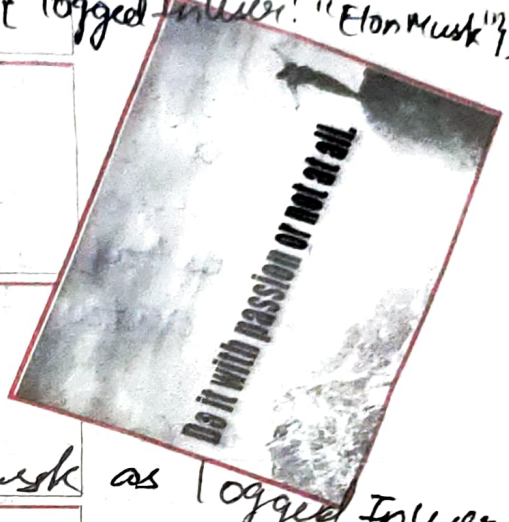
So, Header will have Elon Musk as loggedInUser and other component in App will have userName as loggedInUser

How to modify userContent value dynamically?

We have to pass the setUserName function to change it dynamically

<UserContent.Provider value={{loggedInUser: userName, setUserName}}>
</UserContent.Provider>

userName }}>



Be Passionate

Follow your Passion and success will follow you.

NOVEMBER

DECEMBER

Appointments

09 **Content Provider** :- The content provider is a component that allows its children to subscribe
10 to a content's changes. It accepts a value prop, which is the data that will be shared
11 with the components that are the descendants of the provider. The Provider component is
12 created using React.createContext() and then rendered as part of the component tree.
13 It establishes the content and provides the data to the descendants.
14
Content Consumer :- The Content Consumer is
15 a component that subscribes to the changes in the content provided by its nearest Content
16 Provider ancestor. It allows components to access the content data without the need for prop
17 drilling. The Consumer component is used within the JSX of a component to consume the content data. It takes a function as its child ; and that function receives the current content value as an argument.