[Lecture-11]
[Data is the new oil] Date.....................

→ Higher Order Component – It is a function that takes a component and returns a component. It just a normal JS function.

→ HOC basically takes a component as an input and then it enhances that component, it adds some extra features to that component and returns it back.
– It takes an existing component and acts as an enhancer.

→ {If the restaurant is promoted then add a promoted label to it}:-

// Higher order component

// input – RestaurantCard ⇒ RestaurantCard Promoted

RestaurantCard.js →

. - - -

export const withPromotedLabel = (RestaurantCard) ⇒ {
return () ⇒ {
return (
<div>
<label> Promoted </label>

Date....................

```
<RestaurantCard />

  </div>
 );

};

};
```

Body.js →

```
• import RestaurantCard, {withPromotedlabel} from
"./RestaurantCard";

const RestaurantCardPromoted = withPromotedlabel
                                (RestaurantCard)

<link
  key = {restaurant.data.id}
  to = {"/restaurants/" + restaurant.data.id }>

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

Spiral

Date.....................

• We are passing props to promoted label component so where we will receive it ?

RestaurantCard.js →

```
export const withPromotedLabel = (RestaurantCard) =>
{
return (props) => {
return (
<div>
<label> Promoted </label>
<RestaurantCard {...props} />
                    ↑
        all the props we are receiving
</div>
);
};
};
```

• These HOC are pure functions, it means it will not change/modify the code of RestaurantCard (which we are passing), we are just taking it as an input and we are using it exactly how we use RestaurantCard.

accordion → collapsable body

22 : 45
28 : 25

W-6/12 → hag of the page

Date.........

• We are not modifying the RestaurantCard feature, we are just adding on the top of it, we are just enhancing it.

→ An Important Part Of React Application is to manage it's Data.

→ UI is Very Static, UI doesn't have it's logic own, we give some JSX it just be on that page. UI & data is different part in React.

→ All the React application have 2 layers, One is UI & another One is data layer. And UI layer is powered by Data layer

• The Data layer consist of state, Props, local Variable, curly braces inside JSX (JS code).

• UI layer is the JSX (the code).

1:02:17

```
const handleClick = () => {
    setShowItems(!showItems)
}
```

↓

toggle feature

Spiral

React Dev tools → extension   12:37:00

1:47 ↳ list of components
shows

Date.....................

↳ it shows all component tree

• controlled & uncontrolled component

→ The parent RestaurantMenu is controlling the RestaurantCategory, so it is controlled Component.

→ When it had it's own state, it was an uncontrolled component, ~~as it~~ because RestaurantMenu does not have it's control if it wants to show & hide something it can do itself.

Parent doesn't control over its children

<RestaurantCategory
    key -- -
    data - - - -
    showItems = {index ~~[illegible]~~ === showIndex ? true : false}
    setShowIndex = {() → setShowIndex (index)}
>

lifting the state up → Sometimes we have to lift the state up if we have to control our children.

Date.....................

- Prop Drilling → passing the data betw the intermediate Parents

- passing data in nested components.

- props are drilling in ~~to~~ down and bringing it up at the bottom.

- Managing the data is the crucial part of our application, we don't want to pass random props

- React context → Central Store

→ we use Context which is kind of Global place where our Data is kept and anybody can access it. This is known as React Context.

utils → UserContext.js

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

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

export default UserContext
```

• Don't use "Global Object" term.

Date.....................

Header.js →

∘ import {useContext} from "react";   (→ hook)

∘ import UserContext from "../utils/UserContext";

const data = useContext(UserContext);

→ In our whole react application there is not just one context we can create, we can have as many as context we want to

→ whatever context we want to, pass it to the useContext.

→ Should we keep all the data inside context? So we don't need to use props anywhere?

No, Only the data which we are using multiple places or we feel that it can be use at multiple places that is where we will use context.

we Don't try to put all data inside context.

→ About.js — It is made using class based component and class based component don't have hooks.

import UserContext from "../utils/UserContext";

return (
<div> LoggedIn User ↗ and it as component
<UserContext.Consumer> → {(data) => console.log(it)}
</UserContext.Consumer>
<div>

App.js →

Date..................

→ import UserContext from "./utils/UserContext"

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

→ userprovider wrapping our whole app that's why everywhere we are getting the value of loggedInUser.

Scanned with CamScanner