[lecture-8]

[Let's get classy] Date.....

- Class Based Components →

- In our about us page we will show information of the team members this the feature we will add using class based component.

- First we will write functional component then we will modify to class Based Component

User.js → const User = () => {

```
return (
<div className="user-card">
<h2> Name: Rajesh </h2>
<h3> location: Faridabad </h3>
<h4> Contact: @Rajesh_fb </h4>
</div>
);
};
export default User;
```

UserClass.js ⟶ Class Based Component

- It's a normal JS class.

- Render method returns some piece of JSX

Scanned with CamScanner

"extends React.Component" will make react know that this is a class Based component, now React will start tracking it.

Date.....

```
class UserClass extends React.Component {

  render() {
    return (
      <div className="user-card">
        <h2> Name: Rajesh </h2>
        <h3> location: Faridabad </h3>
        <h4> contact: @Rajeshjha </h4>
      </div>
    );
  }
}

export default UserClass;
```

- Render method will return some piece of JSX which will be displayed onto the UI.
- similarly if we compare it with functional component (a function that returns some piece of JSX)
- Class Based Component is a class which has a render method which returns somepiece of JSX. This is the JSX which will be converted into HTML and rendered onto the webpage.
- React.Component is basically a class which is given to us by React and UserClass is inheriting some properties from it.

Spiral

Scanned with CamScanner

Date.....

- We will have to import it from somewhere

```
import React from "react";
```

About.js → import UserClass from "./UserClass";

```
const About = () => {
  return (
    <div>
      <h1> About </h1>
      <h2> Namaste </h2>
      <UserClass />
    </div>
  );
};

export default About;
```

Props →

```
<UserClass name={"Rajesh (class)"} />
```

UserClass.js →

```
class UserClass extends React.Component {
  constructor (props) {
    super(props);
  }
```

Spiral

Scanned with CamScanner

→ Why do we write super(props) ?

Date.....

```
render() {
return (
  <div>
  <h2> Name: {this.props.name} </h2>
  <h4> location: {this.props.location} </h4>
```

destructuring ⟹

```
render() {
const {name, location} = this.props;
return (
  <div>
  <h2> Name: {name} </h2>
  <h4> location: {location} </h4>
```

Q) How we can create State/local variable inside our class Based Component ?

A) Functional component & Hooks are very new concept inside React, Earlier there used to be no hooks & functional component, so there was an old way of creating State.

- When we say we are creating instance of class Based component → we are loading a class Based component on our Web page.
- Whenever we create instance of a class this Constructor is called and this is the best

Spiral

Scanned with CamScanner

destructuring → const { count } = this.state;

Date.....

place to receive prop and this the best place to create State Variable.

- this.state, it is basically a big whole Object which contains state variables.

```
class UserClass extends React.Component {
    constructor (props) {
    super (props);

    this.state = {
    count : 0,
    };
}

render () {

const { name, location } = this.props;

return (
<div>
<h1> Count : {this.state.count} </h1>
<h2> Name : {name} </h2>
<h3> Location : {location} </h3>
```

Scanned with CamScanner

Date.....

→ this.state = {
count : 0,
count2 : 2,
};

render() {
const {count, count2} = this.state;

return (

<h2> count : {count} </h2>
<h2> count2 : {count2} </h2>

Q.) How we can update the state variables
A.) Never update state variable directly.

this.state.count = this.state.count + 1

wrong method, it create inconsistency

→ <h2> count : {count} </h2>
<button
onClick = {() => {
this.setState ({
count : this.state.count + 1
});
}} >
count increase </button>

Spiral

Scanned with CamScanner

Date.....

- React will re-render the component, and will change this portion of HTML.
- If we are sending just "count" for increment purpose, it will only update "count" in state variable, it will not touch "count2".

÷ life cycle Of React Class Based component ÷

Q. How this component is mounted on a web Page?

Ans: About Us component is the parent component, And user class is inside this About Us component so when we load/render About Us component in the web page, when it goes line by line and sees the class Based component over there so it starts loading the class Based component, it goes the Class Based component and now a new instance of this class is created and what happen when this class is called or this class is instantiated so what happens is the constructor is called. Once the constructor is called then render is called

Scanned with CamScanner

Date.....

- import React from "react";

class UserClass extends React.Component {

or

- import {Component} from "react";
class UserClass extends Component {

- Parent constructor → Parent Render → Child constructor → Child Render

→ This is how the life cycle works.

→ Another method in class Based Component is ComponentDidMount()

class UserClass extends React.component {

→ constructor (props) {----

}

→ componentDidMount () {---

}

→ render () {---

}

Scanned with CamScanner

Date.....

• When this component is loaded first of all is constructor is called then the render method is called and once this class Based component is mounted onto the DOM then the componentDidMount is called.

→ Suppose, if we have this → componentDidMount inside our parent component → About Us. Js also In what order will these functions be called

Parent constructor → Parent render → Child constructor → Child render → Child componentDidMount → then → parent componentDidMount

• So this componentDidMount is called once the component has been completely mounted onto the webpage

• This is the life cycle of the parent child relationship.

• Why componentDidMount is used ?

→ So there are some things that we do once the component is mounted successfully.

→ componentDidMount is used to make API calls

But why we do API calls in componentDidMount ?

Scanned with CamScanner

Date.....

• First of all we load our component, once the component is loaded with basic details then we make an API call and fill the details, so that react component loads very fast. React doesn't wait for API ~~to~~ call to return the result then load it.

→ In react component if we make an API call ~~we don't want to~~, we want to render the component once make an API call and then fill the data inside the component, we don't want ~~API~~ to wait for API to return the data to render the component, otherwise the component will not render and it will keep on waiting for the data to come from the API, react wants to quickly render it then make an API call and then fill the data.

Q. Parent have multiple Children (same → UserClass.js)

~~About~~.js →

```
render() {
return (
<UserClass name={"First"}
<UserClass name={"Second"}
```

Scanned with CamScanner

Date.....

Parent constructor → Parent Render → First child constructor → First child Render → Second child constructor → Second child render → First child Component Did Mount → Second child Component Did Mount → Parent Component Did Mount

But why did the life cycle method work like this?

React lifecycle methods Diagram

→ React has 2 phases –
(1) Render Phase (2) Commit Phase

- First of all constructor is called, then the render method is called then actually react updates the DOM, once the DOM is updated then component Did Mount is called.

- Render phase → (1) constructor
(2) Render

Commit Phase → (1) Updates the DOM
(2) Component Did Mount is called

- Because there are 2 children, react optimise this, react will not call the Component Did Mount of the first child first but it will do is it will just batch the Render phase for these 2 child.

Scanned with CamScanner

1:32:00

Date .....

- These 2 child Render Phase will happen then commit phase will happen together.

Render phase is batched → then commit Phase is batched & completed

But why it is doing that like this ?

Ans React is batching up the Render phase for multiple childrens because once the commit phase starts react tries to update the DOM and DOM manipulation is the most expensive thing when we are updating a component, it takes a lot of time. So React want to batch the render (react calls the constructor, when the react is rendering in the Render phase it basically triggering the reconciliation and in this reconciliation ~~the object~~ it finds out the diff between these virtual DOM, Every thing in Render Phase is happening inside the virtual DOM), every thing in Render Phase is happening inside the virtual DOM, in render phase react is finding the diff then the commit happens because it takes the time, then DOM manipulation happens

Spiral

Scanned with CamScanner

Date.....

• API call →

```
async componentDidMount() {
  const data = await fetch("https://api.github.com/
                users/rajeshjha2000");
  const json = await data.json();
}
```

→ we will create a local state variable on top of it and will put that data into that

```
this.state = {
  userInfo: {
    name: "ABC",
    location: "XYZ",
  },
};
```

Default value

→ we will now update a state variable after we have got the data inside our api call.

```
async componentDidMount() {
  ------
  ------
  this.setState({
    userInfo: json,
  })
```

Spiral

Scanned with CamScanner

1:02:00

Date.....

```
render () {
const {name, location} = this.state.userInfo;
return (
<div>
<h2> Name : {name} </h2>
<h3> location : {location} </h3>
```

Q.> How the whole life cycle of this works ?

A.> First of all, as soon as the UserClass was loaded the constructor was called and when the constructor was called (this.state had default value) then Render happens, state variable had default value, so the render happens with the default value (our component renders with the default value of state variable) ex → name : "ABC", location : "xyz" on to the webpage.

- That means React will update DOM with dummy (default) data.
- Now our Component Did Mount was called, now API call was made it called this.setState.

The Mounting Cycle happened, now we will see the Updating Cycle.

Spiral

Scanned with CamScanner

Date.....

- When we call this.setState our Updated Cycle begins
- Mounting cycle finished when the component was rendered once. (with dummy data) we didn't wait for API call to finish, Component rendered quickly with dummy data, so that users see something, instead we can show some Shimmer UI.
- Updating happens when setState is called, setState updates the State variable, when the State variable is updated React triggers the Render function once again, Constructor is called first once.
- Now the Render happens but this time the state variable has been change it with updated value.
- In the Update cycle react will now Update the DOM.
- React will calculate the diffs and it will update the DOM with the new value now.
- Now the Update Cycle says now it will call the componentDidUpdate.

Spiral

Scanned with CamScanner

Mounting → Updating → Unmounting

1 2:00:00

Date.....

- Just like we had componentDidMount we had componentDidUpdate also.

- componentWillUnmount → This function is called just before our component is unmounting. unmount means when the component will disappear/disabling from the UI.

- componentWillUnmount will be called when the component is gone from the page, we we will go to the next page the earlier component will unmount.

- Never Compare React's life cycle method to Functional component.

- Don't compare componentDidMount with useEffect

- Like when we don't put Dependency Array in useEffect, then it is called after every render, now things are different in Class Based Component after first render componentDidMount is called and after every subsequent render it is updated, it is not mounted.

→ There is a difference bet^n Mount, Update & Unmount

- useEffect(() => {--

}, [count])

- componentDidUpdate (prevProps, prevState) {

→ if (this.state.count !== prevState.count) { - - -

}

Spiral

Scanned with CamScanner

Date.....

• useEffect(() => {...
...}, [count, count2]);
	• componentDidUpdate(prevProps, prevState) {
	if(this.state.count !== prevState.count
	|| this.state.count2 !== prevState.count2)
	{ - - - - - - - - - - }
	}

→ So the array present in useEffect because earlier people used to write multiple state variable condition in if-else condition.

So there can be multiple state variable that's why array is present over there.

→ If we want to do ~~something~~ ~~which we want~~ 2 different things after change in 2 state variable count & count2

```
useEffect(() => {
    abc
}, [count]);

useEffect(() => {
    xyz
}, [count2]);
```

Scanned with CamScanner

Date.....

And this same thing if we want to do in Class Based component →

```
componentDidUpdate (prevProps, prevState) {

  if (
    this.state.count !== prevState.count
  ) {
    abc
  }

  if (
    this.state.count2 !== prevState.count2
  ) {
    xyz
  }
}
```

→ these if-else statement used to go so long, like 30-40 lines.

Q.7 What is the main use of componentWillUnmount

A.7 CleanUp.

→ what type of cleanUp when we move from one component to other component

→ let's take an example —

Spiral

Scanned with CamScanner

Date.....

→ componentDidMount () {

setInterval (() ⇒ {

console.log ("Namaste");

}, 1000)

}

- It started calling "Namaste" after each second and if we move to another component, it is still calling.
This is the cons of SPA.

- If we move back to that component where we have written this code then "Namaste" is calling twice after each second, it started two intervals

→ This is the problem of SPA because it is not reloading our page, it is just changing the component, everytime we come to this component it sets new interval 3, 4, 5 ...
It is a performance loss.

→ That's why we want to unmount things

→ we will use clearInterval in componentWillUnmount

Date.....

```
componentDidMount () {
  this.timer = setInterval (() => {
    console.log ("Namaste") ;
  }, 1000);
}

componentWillUnmount () {
  clearInterval (this.timer);
}
```

```
useEffect (() => {
  setInterval ( () => {
    console.log ("Namaste");
  }, 1000 );
```

→ It will also behave like earlier one, we also have to clear it (unmounting)

```
useEffect (() => {
  const timer = setInterval (() => {
    console.log ("Namaste");
  }, 1000);

  return () => {
    clearInterval (timer);
  }
}, [] );
```

Spiral

Scanned with CamScanner

Date.....

- Unmounting/cleaning will happen if we switch to another component.

Q. Why do we write constructor(props) & super(props)

Q. Why → asyn componentDidMount
Why we can't use async in useEffect callback

```
useEffect(async() => {

});
```

Scanned with CamScanner