Episode 8 - Let's Get Classy

Earlier, Class-based components were used

Children of Children in Routing

```
children: [
    {
        path: "/about",
        element: <About />,
        children: [{
            path: "profile",
            element: <Profile />,
        }]
    }
]
```

Parent Path

Relative Path

/about/profile

No need to write /profile as it will not consider it as a children of about It will become /profile not /about/profile

"/" means from the root

To render this we need to create an outlet in the parent element means inside <About/> or we can directly import Profile Component inside parent

→ Functional Component in the end is a JS function

→ Class based Component in the end is a JS class

Class Based Components

→ Class Profile extends React component {}

extends → to tell its a class Based components

Class based components can't be created without a render method. It's mandatory

render() returns some JSX which will be injected in DOM

#

```
render() {
    return <h1> Profile Class Component </h1>
}
```

we use this.props in Class Based Components

```
constructor (props) {
    super (props);
}
```

We have to use constructor when we call/use props. While creating constructor we have to write this always. If we don't do this React will complain.

Constructor is a place where initialization happens. When the class is rendered constructor is the method to call state

Instead of useState we use this.state

```
→ this.state = {
    count: 0
  }
```

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

```
→ this.state = {
    count: 0
  }

onClick {() => {
        this.setState ({
            count: 1
        })
    }}
```

```
Const [count] = useState (0);
```

```
const [count] = useState (0);
const [count2] = useState (0);
```

```
const [count, setCount] = useState (0)

onClick {() => setCount (1) }
```

We do not mutate state directly
Never do this.state = something! In class-based component, all the states can be updated at once

Functional Component has exact states to update

If there are multiple state variables React will update only the newly updated state variable, keeping the other variables same/untouched.

React Life Cycle

Constructor is called then Render is called then componentDidMount is called.

```
componentDidMount() {
    // API Call
}
```

This method is called after render like useEffect

→ If Parent & 1 Child

Parent - constructor
Parent - render
Child - constructor
Child - render
Child - componentDidMount
Parent - componentDidMount

→ If Parent & 2 Children

Parent - constructor
Parent - render
First Child - constructor
First Child - render
Second Child - constructor
Second Child - render
First Child - componentDidMount
Second Child - componentDidMount
Parent - componentDidMount

React batches up the render phases of child components as commit phase takes time as updating/manipulating DOM is expensive

DOM is updated in a single batch to prevent multiple DOM updation

React Lifecycle Methods

There are 2 phases Render Phase and Commit Phase in Reconciliation

Render Phase → constructor()
Render()

Commit Phase → componentDidMount()

Here, React is modifying the DOM

First, React tries to batch up render phase then go to commit phase
Render Phase is very fast

In parallel, rendering of 2 children / life cycle, Render Phase is completed for both then Commit Phase starts & completes for updating the DOM

Fetching in Class components

```
constructor(props) {
        super(props) {
          this.state = {
              userInfo : {
                  name : "Dummy Name",
                  location : "Dummy Location"
              }
          }
      }
  }

async componentDidMount() {
    const data = await fetch(URL);
    const json = await data.json();

    this.setState({
        userInfo : json,
    })
}

render() {

return (
   <h2> Name: {this.state.userInfo.name} </h2>
   <h2> Location : {this.state.userInfo.location} </h2>
  )
}
```

Output

Parent - constructor
Parent - render (with default value)
Child - constructor First Child
Child - render First Child
Parent - componentDidMount
{login: 'akshaymarch7', id: '28242231'.... }
Child - componentDidMount First Child
Child - render First Child

Mounting Cycle

- Constructor (dummy data)
- Render (dummy data)
  <HTML Dummy>
- ComponentDidMount
  <API call>
  <this.setState>
  └> State variable
  └> userInfo

Updating Cycle

render (API data)
<HTML (new API data)>
- ComponentDidUpdate

Parent - componentDidMount
This is called before making API call in child.
Async componentDidMount() will be called later after parent's componentDidMount() as it is async in Commit phase it will take some data to load that's why it is taking time

Child - render First Child
As async componentDidMount() is setting the state it will call the render phase again so child render is printed

Re-render cycle is known as Updating. For updating the dom componentDidUpdate is used

Difference between ComponentDidMount and ComponentDidUpdate ?

ComponentDidMount() is called after first render and componentDidUpdate is called after every render just like dependancy array

ComponentWillUnmount() will be called when we go to other page before that this method will be called to unmount rendering process.

** Never compare lifecycle methods to React Hooks

```
componentDidMount(prevProps, prevState) {
    if (this.state.count ≠ prevState.count) {
        // code
    }

    if (this.state.count2 ≠ prevState.count2) {
        // code
    }
}
```

This is how it is written in class components

```
useEffect (() => {
    // code
}, [count , count2])
```

This is how it is written in React Hooks

Unmounting in Functional Component

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

return () => {
    console.log ("useEffect Return");
    clearInterval (timer)
}, []);
```

Unmouting means Removing from the page.