**WEEK-6\_REACTJS\_HOL**

**SUPERSET ID: 6393676**

**Objectives**

**1.Explain the need and Benefits of component life cycle**

Lifecycle methods allow React components to run code at specific points (mounting, updating, unmounting).  
**Benefits:**

* Handle side effects (e.g., API calls)
* Optimize performance
* Manage cleanup (e.g., timers, subscriptions)

**2. Identify various life cycle hook methods**

**Class Component Lifecycle Methods:**

* **Mounting:** constructor(), getDerivedStateFromProps(), render(), componentDidMount()
* **Updating:** getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate(), componentDidUpdate()
* **Unmounting:** componentWillUnmount()
* **Error Handling:** getDerivedStateFromError(), componentDidCatch()

**3.List the sequence of steps in rendering a component**

**Mounting Sequence:**  
constructor() → getDerivedStateFromProps() → render() → componentDidMount()

**Updating Sequence:**  
getDerivedStateFromProps() → shouldComponentUpdate() → render() → getSnapshotBeforeUpdate() → componentDidUpdate()

**Unmounting:**  
componentWillUnmount()

**Create a new react application using *create-react-app* tool with the name as “blogapp”**

**App.js**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**Posts.js**

import React, { Component } from 'react';

import './App.css';

class Posts extends Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false,

};

}

componentDidMount() {

this.loadPosts();

}

loadPosts() {

// Replace fetched data with meaningful English posts

const data = [

{ id: 1, title: "React Lifecycle Overview", body: "Learn the stages of component lifecycle: mounting, updating, and unmounting." },

{ id: 2, title: "componentDidMount Usage", body: "Fetch data or initiate API calls once the component is mounted." },

{ id: 3, title: "Error Handling in React", body: "Use componentDidCatch to gracefully handle runtime errors in components." },

{ id: 4, title: "Props vs State", body: "Props let instances communicate; state lets components control their own data." },

{ id: 5, title: "Building a Blog Layout", body: "Design reusable Post and Posts components to render blog entries." },

{ id: 6, title: "Implementing Responsive CSS", body: "Use CSS grid, flexbox, and media queries for mobile-friendly design." },

{ id: 7, title: "Adding Hover Effects", body: "Enhance user experience by adding smooth hover transitions." },

{ id: 8, title: "Using Custom Components", body: "Break UI into smaller components for better maintainability." },

{ id: 9, title: "Styling in React", body: "Use CSS files, inline styles, or styled-components to theme your app." },

{ id: 10, title: "Final Thoughts", body: "React class components with lifecycle hooks are powerful and clear to manage." }

];

this.setState({ posts: data });

}

componentDidCatch(error, info) {

this.setState({ hasError: true });

alert("An error occurred: " + error.message);

}

render() {

const { posts, hasError } = this.state;

if (hasError) {

return <h2 style={{ color: 'red', textAlign: 'center' }}>Something went wrong.</h2>;

}

return (

<div>

<h1>Blog Posts</h1>

<div className="posts-container">

{posts.map(post => (

<div key={post.id} className="post-card">

<div className="post-title">{post.title}</div>

<div className="post-body">{post.body}</div>

</div>

))}

</div>

</div>

);

}

}

export default Posts;

**OUTPUT:**

