**WEEK6**

**1. Explain the Need and Benefits of Component Lifecycle**

**What is the Component Lifecycle?**

In React, every component goes through **a series of stages** from when it is created, updated, and finally removed (destroyed) from the DOM. These stages are called the **lifecycle of a component**.

**Why is it needed?**

React provides **lifecycle methods** (or **hooks**) so you can **run your own code** at specific stages of a component’s life. This is helpful to:

* Fetch data when the component loads
* Set timers or intervals
* Clean up before the component is removed
* Optimize performance
* Trigger animations or API calls on updates

**Benefits:**

* **Control:** Full control over what happens at each stage (mounting, updating, unmounting)
* **Efficiency:** Optimize app performance (e.g., avoid unnecessary API calls)
* **Cleaner code:** Use cleanup methods (like clearing intervals) to avoid memory leaks
* **Dynamic behavior:** Respond to prop or state changes at the right time

**2. Identify Various Lifecycle Hook Methods**

In **Class Components**, lifecycle methods are grouped into **three phases**:

**Mounting (component is created and added to DOM):**

* constructor() – initialize state and bind methods
* static getDerivedStateFromProps() – rarely used; sync state with props
* render() – returns JSX
* componentDidMount() – called once after the component is rendered

**Updating (props/state changes):**

* static getDerivedStateFromProps() – (again, if needed)
* shouldComponentUpdate() – decide whether to re-render
* render() – updates the UI
* getSnapshotBeforeUpdate() – capture info (e.g., scroll position)
* componentDidUpdate() – runs after the update is flushed to the DOM

**Unmounting (component is removed from the DOM):**

* componentWillUnmount() – cleanup (like clearing timers or event listeners)

**3. List the Sequence of Steps in Rendering a Component**

Here’s the typical **lifecycle sequence in a class component:**

**Mounting Phase:**

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

**Updating Phase (due to state/props change):**

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**Unmounting Phase:**

1. componentWillUnmount()

**POST.JS**

export class Post {

  constructor(id, title, body) {

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

**POSTS.JS**

import React, { Component } from 'react';

import { Post } from './Post';

export class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: []

    };

  }

  loadPosts = async () => {

    try {

      const response = await fetch('https://jsonplaceholder.typicode.com/posts');

      const data = await response.json();

      // convert to Post objects

      const postList = data.map(item => new Post(item.id, item.title, item.body));

      this.setState({ posts: postList });

    } catch (error) {

      alert("Error loading posts: " + error.message);

    }

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert("An error occurred in Posts component: " + error.toString());

    console.error("Error details:", info);

  }

  render() {

    return (

      <div>

        <h1>Blog Posts</h1>

        {this.state.posts.map(post => (

          <div key={post.id} style={{ border: '1px solid #ccc', marginBottom: '10px', padding: '10px' }}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

          </div>

        ))}

      </div>

    );

  }

}

**APP.JS**

import React from 'react';

import './App.css';

import { Posts } from './Posts';

function App() {

  return (

    <div className="App">

      <h1 style={{color:"red",textShadow:"0px 0px 12px grey"}}>Welcome to my Blogs</h1>

      <Posts />

    </div>

  );

}

export default App;

**output:**

