**1. Need & Benefits of Component Lifecycle**

Lifecycle methods allow developers to **hook into different stages** of a component’s life (creation, update, destruction).

**Benefits:**

* Fetch data when the component mounts (componentDidMount)
* Catch and handle errors gracefully (componentDidCatch)
* Optimize performance by controlling rendering behavior

**2. Various Lifecycle Hook Methods (Class Components)**

| **Phase** | **Lifecycle Method** | **Description** |
| --- | --- | --- |
| Mounting | constructor() | Initializes state and binds methods |
|  | componentDidMount() | Executes after the component is mounted |
| Updating | shouldComponentUpdate() | Determines if re-render is needed |
|  | componentDidUpdate() | Executes after update |
| Unmounting | componentWillUnmount() | Cleanup before component is removed |
| Error Handling | componentDidCatch() | Catches errors in child components |

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

1. Constructor →
2. render() →
3. componentDidMount() →
4. (if error) componentDidCatch() →
5. Component updates (if state/props change)

**Hands-On Lab Steps**

**1. Create a React App**

npx create-react-app blogapp

**2. Open in VS Code**

cd blogapp

code .

**3. Create Post.js (Component to display single post)**

**Path:** src/Post.js

import React from 'react';

function Post({ title, body }) {

return (

<div style={{ border: '1px solid #ccc', padding: '15px', margin: '10px 0' }}>

<h3>{title}</h3>

<p>{body}</p>

</div>

);

}

export default Post;

**4. Create Posts.js (Class-based Component to fetch and display posts)**

**Path:** src/Posts.js

import React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

constructor(props) {

super(props);

this.state = {

posts: [],

hasError: false

};

}

// 6. Load posts from API

loadPosts = () => {

fetch('https://jsonplaceholder.typicode.com/posts')

.then(response => {

if (!response.ok) throw new Error("Network response was not ok");

return response.json();

})

.then(data => {

this.setState({ posts: data });

})

.catch(error => {

console.error("Fetch error:", error);

this.setState({ hasError: true });

});

};

// 7. Lifecycle hook

componentDidMount() {

this.loadPosts();

}

// 9. Error boundary

componentDidCatch(error, info) {

alert("An error occurred in Posts component!");

console.log("Error:", error);

console.log("Info:", info);

}

// 8. Render UI

render() {

const { posts, hasError } = this.state;

if (hasError) {

return <p style={{ color: 'red' }}>Something went wrong while loading posts.</p>;

}

return (

<div style={{ padding: '20px' }}>

<h2>Blog Posts</h2>

{posts.map(post => (

<Post key={post.id} title={post.title} body={post.body} />

))}

</div>

);

}

}

export default Posts;

**10. Edit App.js to Use Posts Component**

**Path:** src/App.js

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<Posts />

</div>

);

}

export default App;

**11. Run the App**

npm start

