**Module 10 – Single Page Application framework – React**

**4.ReactJS-HOL**

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

In React, components are the building blocks of the user interface. Each component goes through a process from creation to destruction. This process is called the Component Lifecycle.

**Why is it needed?**

The component lifecycle allows us to:

* Control what happens and when inside a component.
* Perform actions at specific stages (like fetching data when a component is ready).
* Improve performance by avoiding unnecessary updates.
* Handle errors gracefully in case something goes wrong.

Benefits:

* We can initialize data when the component loads.
* We can update the UI when the component receives new data.
* We can clean up resources when the component is removed.
* Helps us manage side effects, like API calls and timers.
* Makes the component more predictable and maintainable.

**Identify Various Lifecycle Hook Methods**

In class-based components, React gives us special methods called **lifecycle hooks** that allow us to tap into different stages of the component.

**Common Lifecycle Hook Methods:**

| **Lifecycle Phase** | **Hook Method** | **Purpose** |
| --- | --- | --- |
| Mounting | constructor() | Set initial state and bind methods |
|  | static getDerivedStateFromProps() | Sync props with state |
|  | render() | Renders the UI to the screen |
|  | componentDidMount() | Run code after component mounts (like API calls) |
| Updating | shouldComponentUpdate() | Decide whether to re-render |
|  | getSnapshotBeforeUpdate() | Capture state before update |
|  | componentDidUpdate() | Run code after component updates |
| Unmounting | componentWillUnmount() | Cleanup code (like timers, subscriptions) |
| Error Handling | componentDidCatch() | Handle runtime errors gracefully |

In function components, similar behavior is handled using React Hooks like useEffect.

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

When a component is rendered in React, it goes through a well-defined **sequence of steps**, especially in class-based components.

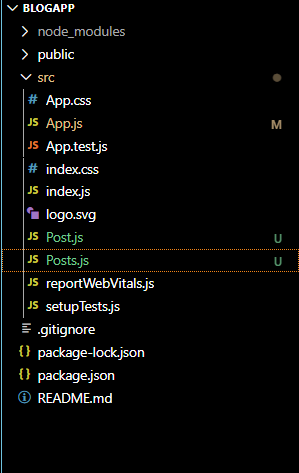
**Rendering Lifecycle Steps:**

1. **constructor(props)**
   * Called first. Used to set up initial state.
2. **getDerivedStateFromProps()** *(Optional)*
   * Syncs props with state before rendering.
3. **render()**
   * Returns the JSX to be shown on the screen.
4. **componentDidMount()**
   * Called after the component is added to the DOM. Ideal for API calls or subscriptions.

**Bonus: Updating and Unmounting Phases (For Reference)**

* When props/state change:
  + getDerivedStateFromProps()
  + shouldComponentUpdate()
  + render()
  + getSnapshotBeforeUpdate()
  + componentDidUpdate()
* When component is removed:
  + componentWillUnmount()

**Structure:**

****

**App.js:**

// 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;

**Post.js:**

// src/Post.js

import React from 'react';

const Post = ({ title, body }) => {

  return (

    <div style={{ marginBottom: "20px", border: "1px solid grey", padding: "10px" }}>

      <h3>{title}</h3>

      <p>{body}</p>

    </div>

  );

};

export default Post;

**Posts.js:**

// 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

    };

  }

  // Step 4: Method to Load Posts

  loadPosts = async () => {

    try {

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

      const data = await response.json();

      this.setState({ posts: data });

    } catch (error) {

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

      this.setState({ hasError: true });

    }

  };

  // Step 5: Lifecycle hook - componentDidMount

  componentDidMount() {

    this.loadPosts();

  }

  // Step 6: Lifecycle hook - componentDidCatch

  componentDidCatch(error, info) {

    alert("Something went wrong!");

    this.setState({ hasError: true });

  }

  // Step 7: Render method

  render() {

    if (this.state.hasError) {

      return <h2 style={{ color: 'red' }}>Failed to load posts!</h2>;

    }

    return (

      <div>

        <h1>All Blog Posts</h1>

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

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

        ))}

      </div>

    );

  }

}

export default Posts;

**Output:**

