**4.ReactJs HOL**

**1.** **Explain the Need and Benefits of Component Life Cycle**

A component life cycle refers to the series of stages a React component goes through from its creation to its removal from the DOM.

**Need:**

* To perform actions at specific points in a component's life.
* To control initialization, updating, and cleanup of components.
* Helps in fetching data, managing timers, subscribing/unsubscribing, and handling errors.

**Benefits:**

* Fetch API data when the component is mounted (e.g., componentDidMount).
* Clean up timers or listeners (e.g., componentWillUnmount).
* Catch errors and show fallback UI (componentDidCatch).
* Decide when component should update (shouldComponentUpdate).

**2.** **Identify Various Life Cycle Hook Methods**

**Mounting (component is being created)**

* constructor()
* static getDerivedStateFromProps()
* render()
* componentDidMount()

**Updating (props/state change)**

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**Unmounting (component is being removed)**

* componentWillUnmount()

**Error Handling**

* componentDidCatch()
* static getDerivedStateFromError()

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

**Mounting Phase (Initial Render)**

1. constructor() → Component is initialized
2. getDerivedStateFromProps() → Updates state from props (optional)
3. render() → JSX is returned
4. componentDidMount() → Runs after the component is added to DOM (ideal for data fetching)

**Updating Phase (Props or State Change)**

1. getDerivedStateFromProps() → (again)
2. shouldComponentUpdate() → Decide whether to re-render
3. render() → JSX is re-rendered
4. getSnapshotBeforeUpdate() → Capture info before DOM update
5. componentDidUpdate() → React finishes updating DOM

**Unmounting Phase**

1. componentWillUnmount() → Cleanup before component is destroyed

**Error Handling Phase**

If there's an error in render or child component:

1. getDerivedStateFromError()
2. componentDidCatch()

**React App : blogapp**

**Posts.js**

**i**mport React, { Component } from 'react';

import Post from './Post';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [

        { id: 1, title: "React Basics", body: "Learn how components, props, and state work." },

        { id: 2, title: "Using useEffect", body: "Manage side effects like API calls in functional components." },

        { id: 3, title: "Routing with React Router", body: "Navigate between pages using React Router." },

        { id: 4, title: "Managing State with Redux", body: "Handle global state in large React apps." },

        { id: 5, title: "React Lifecycle Methods", body: "Understand how class components manage updates." }

      ],

      hasError: false

    };

  }

  componentDidCatch(error, info) {

    alert("Something went wrong in Posts component.");

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

    this.setState({ hasError: true });

  }

  render() {

    if (this.state.hasError) {

      return <h2>Something went wrong while loading posts.</h2>;

    }

    return (

      <div>

        <h2>Blog Posts</h2>

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

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

        ))}

      </div>

    );

  }

}

export default Posts;

**Post.js**

import React from 'react';

function Post({ title, body }) {

  return (

    <div style={{ border: '1px solid gray', padding: '10px', marginBottom: '10px' }}>

      <h3>{title}</h3>

      <p>{body}</p>

    </div>

  );

}

export default Post;

**App.js**

import React from 'react';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <h1>Welcome to Blog App</h1>

      <Posts />

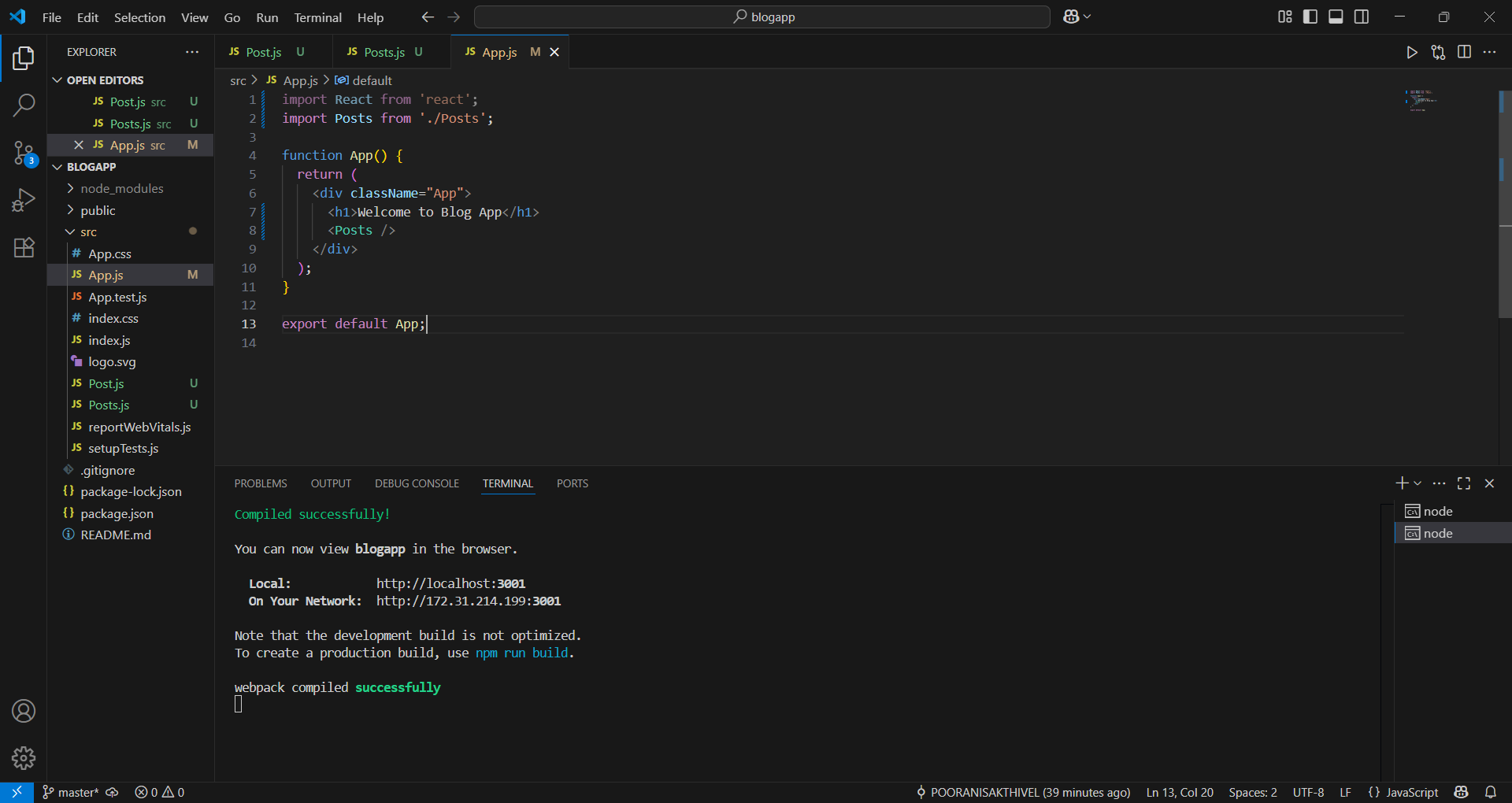
    </div>

  );

}

export default App;

**Final Output:**

****

