React-4

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

The **React component lifecycle** refers to the sequence of methods invoked during the existence of a component—from creation through updating to unmounting. Understanding lifecycle methods is crucial for executing code at specific points, such as data fetching, DOM manipulation, or cleanup tasks.

**Benefits:**

* Enables efficient data loading using hooks like componentDidMount().
* Helps in managing resources, such as setting or clearing timers.
* Allows error handling using methods like componentDidCatch().
* Optimizes performance by managing updates and rendering intelligently.

**2. Identify Various Lifecycle Hook Methods**

React class components include several lifecycle hook methods, broadly categorized into:

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

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

The sequence of steps in rendering a React class component is as follows:

1. constructor() – Initialization and state setup.
2. static getDerivedStateFromProps() – Sync state with props.
3. render() – Return JSX elements.
4. componentDidMount() – Called after the component is mounted.

For updates:

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

**Hands-on Lab Instructions Summary:**

To implement component lifecycle features in a React application:

1. **Create a React app named blogapp:**
2. npx create-react-app blogapp
3. **Open the project in Visual Studio Code.**
4. **Create a file named Post.js under src:**
5. import React from 'react';
6. function Post(props) {
7. return (
8. <div>
9. <h2>{props.title}</h2>
10. <p>{props.body}</p>
11. </div>
12. );
13. }
14. export default Post;
15. **Create a file named Posts.js and define a class component:**
16. import React, { Component } from 'react';
17. import Post from './Post';
18. class Posts extends Component {
19. constructor(props) {
20. super(props);
21. this.state = {
22. posts: [],
23. hasError: false
24. };
25. }
26. loadPosts = () => {
27. fetch('https://jsonplaceholder.typicode.com/posts')
28. .then(response => response.json())
29. .then(data => this.setState({ posts: data }))
30. .catch(error => {
31. console.error("Error loading posts", error);
32. this.setState({ hasError: true });
33. });
34. };
35. componentDidMount() {
36. this.loadPosts();
37. }
38. componentDidCatch(error, info) {
39. alert('An error occurred: ' + error.message);
40. console.error("Caught error:", info);
41. }
42. render() {
43. const { posts, hasError } = this.state;
44. if (hasError) {
45. return <p>Something went wrong while loading posts.</p>;
46. }
47. return (
48. <div>
49. {posts.map(post => (
50. <Post key={post.id} title={post.title} body={post.body} />
51. ))}
52. </div>
53. );
54. }
55. }
56. export default Posts;
57. **Edit App.js to include Posts component:**
58. import React from 'react';
59. import Posts from './Posts';
60. function App() {
61. return (
62. <div className="App">
63. <h1>Blog Posts</h1>
64. <Posts />
65. </div>
66. );
67. }
68. export default App;
69. **Run the application:**
70. npm start
71. **Visit** http://localhost:3000 **in your browser.**