**ReactJS Hands-On Lab Report - Blog App with Lifecycle Hooks**

**Objectives**

* Explain the need and Benefits of component life cycle
* Identify various life cycle hook methods
* List the sequence of steps in rendering a component

**Concepts and Explanations**

**Why Component Life Cycle Matters**

React component life cycle methods allow developers to manage side effects and control the behavior of a component at various points in its life—such as mounting, updating, and unmounting. They're essential for tasks like fetching data, setting up subscriptions, or handling errors.

**Lifecycle Hook Methods**

* constructor()
* render()
* componentDidMount()
* componentDidUpdate()
* componentWillUnmount()
* componentDidCatch()

**Sequence of Component Rendering**

1. constructor()
2. render()
3. componentDidMount()
4. (If state/props update) componentDidUpdate()
5. (If error occurs) componentDidCatch()
6. componentWillUnmount() (on component removal)

**Hands-On Lab: Blog App using Lifecycle Hooks**

You will build a React application named **blogapp** that fetches and displays blog posts using componentDidMount() and handles errors using componentDidCatch().

**Step-by-Step Instructions**

1. Open **Command Prompt** and run:
2. npx create-react-app blogapp
3. Navigate to the project folder:
4. cd blogapp
5. Open the project in Visual Studio Code:
6. code .
7. Inside the **src** folder, create a new file named Post.js.
8. Add the following code to Post.js:
9. import React from 'react';
10. class Post extends React.Component {
11. render() {
12. const { title, body } = this.props;
13. return (
14. <div style={{border: '1px solid #ccc', margin: '10px', padding: '10px'}}>
15. <h3>{title}</h3>
16. <p>{body}</p>
17. </div>
18. );
19. }
20. }
21. export default Post;
22. Create another file named Posts.js inside the **src** folder.
23. Add the following code to Posts.js:
24. import React from 'react';
25. import Post from './Post';
26. class Posts extends React.Component {
27. constructor(props) {
28. super(props);
29. this.state = {
30. posts: [],
31. hasError: false
32. };
33. }
34. loadPosts() {
35. fetch('https://jsonplaceholder.typicode.com/posts')
36. .then(res => res.json())
37. .then(data => this.setState({ posts: data }))
38. .catch(error => {
39. console.error("Error fetching posts:", error);
40. this.setState({ hasError: true });
41. });
42. }
43. componentDidMount() {
44. this.loadPosts();
45. }
46. componentDidCatch(error, info) {
47. alert('Something went wrong: ' + error);
48. this.setState({ hasError: true });
49. }
50. render() {
51. if (this.state.hasError) {
52. return <h2>Unable to display posts.</h2>;
53. }
54. return (
55. <div>
56. <h2>Blog Posts</h2>
57. {this.state.posts.slice(0, 10).map(post => (
58. <Post key={post.id} title={post.title} body={post.body} />
59. ))}
60. </div>
61. );
62. }
63. }
64. export default Posts;
65. Now open App.js (inside src) and replace its contents with:
66. import React from 'react';
67. import Posts from './Posts';
68. function App() {
69. return (
70. <div className="App">
71. <h1>My Blog App</h1>
72. <Posts />
73. </div>
74. );
75. }
76. export default App;
77. Save all files.
78. Back in the terminal, run the app:

npm start

1. Open your browser and go to:  
   [http://localhost:3000](http://localhost:3000/)





