4.ReactJS-HOL

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

In this hands-on lab, you will learn how to:

- Implement componentDidMount() hook
- Implementing componentDidCatch() life cycle hook.

Prerequisites

The following is required to complete this hands-on lab:

- Node.js
- NPM
- Visual Studio Code

Code:

Post.js

```
import React, { Component } from 'react';

class Post extends Component {
    render() {
        const { title, body } = this.props;
        return (
            <div style={{ border: "1px solid #ccc", padding: "10px", margin:
"10px" }}>
                <h2>{title}</h2>
                <p>{body}</p>
            </div>
        );
    }
}

export default Post;
```

Posts.js

```
import React, { Component } from 'react';
import Post from './Post';

class Posts extends Component {
    constructor(props) {
        super(props);
        this.state = {
            posts: [],
            hasError: false
        };
    }

    loadPosts = () => {
        fetch("https://jsonplaceholder.typicode.com/posts")
            .then(response => response.json())
            .then(data => this.setState({ posts: data }))
            .catch(error => {
                console.error("Fetch error:", error);
                this.setState({ hasError: true });
            });
    }

    componentDidMount() {
        this.loadPosts();
    }

    componentDidCatch(error, info) {
        alert("An error occurred in the Posts component.");
        console.error("Error caught:", error, info);
    }

    render() {
        const { posts, hasError } = this.state;

        if (hasError) {
            return <h2>Something went wrong while fetching posts.</h2>;
        }

        return (
            <div>
                <h1>Blog Posts</h1>
                {posts.slice(0, 10).map(post => (
                    <Post key={post.id} title={post.title} body={post.body} />
                ))}
            </div>
        );
    }
```

```
}

export default Posts;
```

App.js

```
import React from 'react';
import Posts from './Posts';

function App() {
  return (
    <div className="App">
      <Posts />
    </div>
  );
}

export default App;
```

Terminal Output:

```
PS E:\Cognizant-Java FSE\Week 6\Code\4.ReactJS-HOL> npx create-react-app blogapp

Creating a new React app in E:\Cognizant-Java FSE\Week 6\Code\4.ReactJS-HOL\blogapp.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...


added 1323 packages in 57s

269 packages are looking for funding
  run `npm fund` for details

Installing template dependencies using npm...

added 18 packages, and changed 1 package in 6s

269 packages are looking for funding
  run `npm fund` for details
Removing template package using npm...


removed 1 package, and audited 1341 packages in 5s

269 packages are looking for funding
  run `npm fund` for details
```

```
PS E:\Cognizant-Java FSE\Week 6\Code\4.ReactJS-HOL> cd blogapp
PS E:\Cognizant-Java FSE\Week 6\Code\4.ReactJS-HOL\blogapp> npm start

> blogapp@0.1.0 start
> react-scripts start

(node:9228) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the
pMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:9228) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use
tupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view blogapp in the browser.


Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Browser Output: