**WEEK-6**

**React JS Handson-4**

* **Explain the need and Benefits of component life cycle**

The component life cycle helps us understand how a React component behaves from creation to removal. It allows us to control what happens during mounting, updating, and unmounting—like fetching data, optimizing performance, or cleaning up resources.

* **Identify various life cycle hook methods**

1. constructor()
2. componentDidMount()
3. shouldComponentUpdate()
4. componentDidUpdate()
5. componentWillUnmount()

**• List the sequence of steps in rendering a component**

1. Constructor – Initializes state and props
2. Render – Returns the JSX
3. componentDidMount – Runs after the component appears on screen
4. componentDidUpdate – Runs after an update
5. componentWillUnmount – Runs before removal from DOM
6. Create BlogApp Project  
   I created a new React app named blogapp using the create-react-app tool.
7. Open Project in VS Code  
   I opened the newly created blogapp project in Visual Studio Code for editing the files.
8. Create Post.js File  
   Inside the src folder, I created a file called Post.js. This file contains a class named Post which has id, title, and body as its properties.
9. Create Posts Component  
   I created another file named Posts.js in the src folder. In this file, I created a class-based component named Posts using React.Component.
10. Add State with Constructor  
    In the constructor of Posts, I initialized the state with an empty array called posts which will later hold all post data fetched from the API.
11. Create loadPosts() Method  
    I wrote a method called loadPosts() inside Posts which uses Fetch API to call https://jsonplaceholder.typicode.com/posts. Once data is fetched, it updates the state with the received posts.
12. Use componentDidMount() Hook  
    I used the componentDidMount() lifecycle method to call loadPosts() as soon as the component is mounted. This ensures the data is loaded once the page loads.
13. Display Posts using render()  
    Inside the render() method, I mapped over the posts array and displayed each post's title as a heading and body as a paragraph in the browser.
14. Handle Errors with componentDidCatch()  
    I added a componentDidCatch() method to catch any runtime errors in the component. If an error occurs, an alert message is shown to the user.
15. Run the App and Show Output  
    Finally, I imported and used the Posts component in App.js, then started the app using npm start and viewed it in the browser.

**Post.js**

class Post {

  constructor(userId, id, title, body) {

    this.userId = userId;

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

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(res => res.json())

      .then(data => {

        const postList = data.map(post => new Post(post.userId, post.id, post.title, post.body));

        this.setState({ posts: postList });

      })

      .catch(error => {

        this.setState({ hasError: true });

        console.error('Error fetching posts:', error);

      });

  };

  componentDidMount() {

    this.loadPosts();

  }

  componentDidCatch(error, info) {

    alert('An error occurred: ' + error);

  }

  render() {

    if (this.state.hasError) {

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

    }

    return (

      <div>

        <h1>Blog Posts</h1>

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

          <div key={post.id} style={{ border: '1px solid #ddd', padding: '15px', margin: '10px 0' }}>

            <h2>{post.title}</h2>

            <p>{post.body}</p>

          </div>

        ))}

      </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;



