**Objectives**

* Explain the need and Benefits of component life cycle

The **component lifecycle** in React is the series of methods that are called at different stages of a component's existence—from creation (mounting), updating (due to state/props changes), to removal (unmounting).

**Need and Benefits:**

1. It allows developers to run code at specific points, such as fetching data once a component is ready (after mounting) or cleaning up resources (before unmounting).
2. It helps in managing side effects like API calls, timers, or subscriptions systematically.
3. Lifecycle methods provide fine control over rendering and updating flows, enhancing app performance and behaviour.

* Identify various life cycle hook methods
  + 1. React class components have these main lifecycle methods/hooks:

1. **Mounting** (when the component is inserted into the DOM):
   1. constructor() – initializes state and binds methods.
   2. static getDerivedStateFromProps() – updates state based on props.
   3. render() – returns the JSX to display.
   4. componentDidMount() – runs after component is mounted; ideal for fetching data.
2. **Updating** (when component’s state or props change):
   1. static getDerivedStateFromProps()
   2. shouldComponentUpdate() – decides if re-render is needed.
   3. render()
   4. getSnapshotBeforeUpdate() – captures information before DOM changes.
   5. componentDidUpdate() – runs after update.
3. **Unmounting** (when component is removed from DOM):
   1. componentWillUnmount() – cleanup tasks (e.g., cancel timers).

* List the sequence of steps in rendering a component

For **initial render (mounting)**, the sequence is:

1. constructor() – Initialize state and props.
2. static getDerivedStateFromProps() (rarely used) – Update state from props.
3. render() – Returns JSX, telling React what to draw.
4. React updates the actual DOM.
5. componentDidMount() – Runs after DOM has been updated; great place for side effects like fetching data.

For **updates**, the sequence includes getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate, and componentDidUpdate.

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

## **Notes**

Estimated time to complete this lab: **60 minutes.**

1. Create a new react application using *create-react-app* tool with the name as “blogapp”
2. Open the application using VS Code
3. Create a new file named as **Post.js** in **src folder** with following properties

// src/Post.js

class Post {

  constructor(userId, id, title, body) {

    this.userId = userId;

    this.id = id;

    this.title = title;

    this.body = body;

  }

}

export default Post;

Figure 2: Post class

1. Create a new class based component named as **Posts** inside **Posts.js** file
2. Initialize the component with a list of Post in state of the component using the constructor
3. Create a new method in component with the name as **loadPosts()** which will be responsible for using Fetch API and assign it to the component state created earlier. To get the posts use the url (<https://jsonplaceholder.typicode.com/posts>)
4. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts
5. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.
6. Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.

// src/Posts.js

import React, { Component } from 'react';

class Posts extends Component {

  constructor(props) {

    super(props);

    this.state = {

      posts: [], // array of post objects

      hasError: false,

      errorMessage: ''

    };

  }

  // Method to fetch posts data and assign it to state

  loadPosts = () => {

    fetch('https://jsonplaceholder.typicode.com/posts')

      .then(response => {

        if (!response.ok) {

          throw new Error(`HTTP error! status: ${response.status}`);

        }

        return response.json();

      })

      .then(data => {

        this.setState({ posts: data });

      })

      .catch(error => {

        this.setState({ hasError: true, errorMessage: error.message });

      });

  }

  // Lifecycle method to trigger data fetch after component mounts

  componentDidMount() {

    this.loadPosts();

  }

  // Error boundary method to catch rendering errors in this component or its children

  componentDidCatch(error, info) {

    alert('An error occurred in Posts component: ' + error.toString());

    // Optionally, could update state or log error here

  }

  render() {

    const { posts, hasError, errorMessage } = this.state;

    if (hasError) {

      return <div style={{ color: 'red' }}>Error: {errorMessage}</div>;

    }

    return (

      <div style={{ padding: '20px' }}>

        <h2>Blog Posts</h2>

        {posts.length === 0 ? (

          <p>Loading posts...</p>

        ) : (

          posts.map(post => (

            <div key={post.id} style={{ marginBottom: '20px', borderBottom: '1px solid #ccc', paddingBottom: '10px' }}>

              <h3>{post.title}</h3>

              <p>{post.body}</p>

            </div>

          ))

        )}

      </div>

    );

  }

}

export default Posts;

1. Add the Posts component to App component.
2. Build and Run the application using *npm start* command.

