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

The component lifecycle in React refers to the sequence of phases a component goes through from creation to removal. These lifecycle phases allow developers to hook into specific moments (like mounting, updating, and unmounting) to run custom code. This is important for tasks such as data fetching, setting up subscriptions, managing timers, and cleaning up resources. Lifecycle methods provide better control over component behavior and performance, allowing for efficient updates, optimized rendering, and smooth integration with external APIs or libraries.

**Identify various life cycle hook methods**

React class components include several lifecycle hook methods categorized into different phases:

1. **Mounting Phase (Component is added to the DOM)**:
   * constructor(): Initializes state and binds methods.
   * static getDerivedStateFromProps(): Syncs state with props before rendering.
   * render(): Describes the UI structure (required method).
   * componentDidMount(): Runs after the component has rendered and been added to the DOM (commonly used for API calls or setup).
2. **Updating Phase (Component receives new props or state)**:
   * static getDerivedStateFromProps(): Called again before every render.
   * shouldComponentUpdate(): Determines whether a re-render is needed.
   * render(): Updates the UI.
   * getSnapshotBeforeUpdate(): Captures information before the DOM changes.
   * componentDidUpdate(): Called after the component re-renders (commonly used for reacting to prop/state changes).
3. **Unmounting Phase (Component is removed from the DOM)**:
   * componentWillUnmount(): Used to perform cleanup (e.g., removing listeners or clearing timers).
4. **Error Handling Phase**:
   * componentDidCatch(): Handles errors during rendering or in lifecycle methods.
   * static getDerivedStateFromError(): Updates the state so the next render shows an error fallback UI.

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

**Mounting Phase (initial render):**

1. constructor()
2. static getDerivedStateFromProps()
3. render()
4. componentDidMount()

**Updating Phase (on props/state change):**

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

**Unmounting Phase (component removal):**

1. componentWillUnmount()

This sequence allows developers to safely prepare and clean up their components at every major stage of their lifecycle.

**Create a new react application using *create-react-app* tool with the name as “blogapp”**

**Index.js :-**

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

);

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

**App.js :-**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

  return (

    <div className="App">

      <h1>Welcome to BlogApp</h1>

      <Posts />

    </div>

  );

}

export default App;

**Posts.js :-**

import React, { Component } from 'react';

class Post extends Component {

  render() {

    const { title, body } = this.props;

    return (

      <div style={{ border: '1px solid gray', margin: '10px', padding: '10px' }}>

        <h3>{title}</h3>

        <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,

      error: null

    };

  }

  // Step 6: Define loadPosts method

  loadPosts = () => {

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

      .then(res => res.json())

      .then(data => this.setState({ posts: data }))

      .catch(error => {

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

      });

  };

  // Step 7: Use componentDidMount to load posts

  componentDidMount() {

    this.loadPosts();

  }

  // Step 9: Error handling with componentDidCatch

  componentDidCatch(error, info) {

    this.setState({ hasError: true });

    alert('An error occurred while loading posts.');

    console.error('Error caught in componentDidCatch:', error, info);

  }

  // Step 8: Render posts

  render() {

    const { posts, hasError } = this.state;

    if (hasError) {

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

    }

    return (

      <div>

        <h2>Blog Posts</h2>

        {posts.map(post => (

          <Post key={post.id} title={post.title} body={post.body} />

        ))}

      </div>

    );

  }

}

export default Posts;

**App.css :-**

.App {

text-align: center;

}

.App-logo {

height: 40vmin;

pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

.App-logo {

animation: App-logo-spin infinite 20s linear;

}

}

.App-header {

background-color: #282c34;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;

}

.App-link {

color: #61dafb;

}

@keyframes App-logo-spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

Output :-



