**Objectives**

* Explain the need and Benefits of component life cycle
* In React, components go through a predictable lifecycle from creation to removal.
* React provides built-in methods (called lifecycle hooks) that let you run code at specific points during this lifecycle.

**Benefits:**

* **Manage side effects:** You can perform tasks like fetching data or subscribing to events when a component mounts.
* **Cleanup resources:** Helps release memory by cleaning up tasks (e.g., clearing timers or subscriptions) when a component unmounts.
* **Improve performance:** You can optimize rendering and avoid unnecessary updates.
* **Organized code:** Keeps logic related to different stages of a component's existence separate and readable.
* Identify various life cycle hook methods

|  |  |  |
| --- | --- | --- |
| Phase | Method | Purpose |
| Mounting | constructor() | Initializes state and binds methods |
|  | static getDerivedStateFromProps() | Sync state from props (rarely used) |
|  | render() | Renders the JSX |
|  | componentDidMount() | Runs after component is added to the DOM (used for API calls, timers) |
| Updating | static getDerivedStateFromProps() | Also runs on updates |
|  | shouldComponentUpdate() | Determines whether re-render is needed |
|  | render() | Re-renders the JSX |
|  | getSnapshotBeforeUpdate() | Captures DOM info before update (e.g., scroll position) |
|  | componentDidUpdate() | Runs after updates (used for DOM operations or more API calls) |
| Unmounting | componentWillUnmount() | Cleanup before component is removed (clear intervals, abort requests) |
| Error Handling | componentDidCatch() | Catch and handle errors in child components |

* List the sequence of steps in rendering a component

**When a component is first rendered (mounted):**

* constructor() – Initialization
* static getDerivedStateFromProps() – Optional state sync from props
* render() – JSX is returned
* DOM is updated
* componentDidMount() – Side-effects or async tasks (e.g., fetch data)

**On Re-render (update):**

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**On Unmounting:**

* componentWillUnmount()

## **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



Figure 2: Post class

1. Create a new class based component named as **Posts** inside **Posts.js** file



Figure 3: Posts Component

1. Initialize the component with a list of Post in state of the component using the constructor
2. 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>)



Figure 4: loadPosts() method

1. Implement the **componentDidMount()** hook to make calls to **loadPosts()** which will fetch the posts



Figure 5: componentDidMount() hook

1. Implement the **render()** which will display the title and post of posts in html page using heading and paragraphs respectively.



Figure 6: render() method

1. Define a **componentDidCatch()** method which will be responsible for displaying any error happing in the component as alert messages.



Figure 7: componentDidCatch() hook

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

Hands-on:



