**React Component Lifecycle**

# 1. Explain the Need and Benefits of Component Lifecycle

Component lifecycle in React refers to the series of phases a component goes through from its creation to its removal from the DOM. React provides lifecycle methods (in class components) and hooks (in function components) to manage behavior at different stages.

Need for Component Lifecycle:

* - Control behavior at different stages (e.g., fetch data before UI shows)  
  - Optimize performance by avoiding unnecessary renders  
  - Manage side effects such as subscriptions or API calls  
  - Clean up resources like timers or event listeners  
  - Debug more easily and understand when something changed

Benefits:

|  |  |
| --- | --- |
| Benefit | Description |
| Efficient Resource Management | Lifecycle methods allow developers to clean up memory and remove subscriptions. |
| Control Over Rendering | Developers can control what and when something renders or updates. |
| Data Fetching & Setup | Allows fetching data or performing operations right before or after rendering. |
| Debugging Made Easier | Easier to trace bugs by knowing the component’s state. |
| Customization | Execute logic at specific points, like animations or condition-based actions. |

# 2. Identify Various Lifecycle Hook Methods

React lifecycle methods differ between class components and function components.

Class Component Lifecycle Methods:

|  |  |  |
| --- | --- | --- |
| Phase | Method | Description |
| Mounting | constructor() | Initializes state and binds methods. |
| Mounting | static getDerivedStateFromProps() | Updates state based on props before rendering. |
| Mounting | render() | Renders the component’s UI. |
| Mounting | componentDidMount() | Executed after component mounts, suitable for API calls. |
| Updating | shouldComponentUpdate() | Determines whether the component should re-render. |
| Updating | getSnapshotBeforeUpdate() | Captures info from DOM before update. |
| Updating | componentDidUpdate() | Runs after updates, suitable for reacting to prop/state changes. |
| Unmounting | componentWillUnmount() | Used for cleanup like stopping timers or removing listeners. |

Function Component Lifecycle Using Hooks:

|  |  |
| --- | --- |
| Purpose | useEffect Equivalent |
| On Mount | useEffect(() => { ... }, []) |
| On Update | useEffect(() => { ... }, [dependency]) |
| On Unmount | useEffect(() => { return () => { ... } }, []) |

Other useful hooks: useState, useRef, useMemo, useCallback.

# 3. List the Sequence of Steps in Rendering a Component

Class Component Rendering Sequence:

Mounting Phase:  
1. constructor()  
2. static getDerivedStateFromProps()  
3. render()  
4. componentDidMount()  
  
Updating Phase:  
1. static getDerivedStateFromProps()  
2. shouldComponentUpdate()  
3. render()  
4. getSnapshotBeforeUpdate()  
5. componentDidUpdate()  
  
Unmounting Phase:  
1. componentWillUnmount()

Function Component Rendering Sequence:

- On mount: useEffect(() => { ... }, [])  
- On update: useEffect(() => { ... }, [dependencies])  
- On unmount: useEffect(() => { return () => { ... } }, [])

Summary Table:

|  |  |  |
| --- | --- | --- |
| Phase | Class Component | Function Component |
| Mount | constructor, render, componentDidMount | useEffect(() => {...}, []) |
| Update | shouldComponentUpdate, render, componentDidUpdate | useEffect(() => {...}, [deps]) |
| Unmount | componentWillUnmount | useEffect(() => { return () => {...} }, []) |

**App.js**

import React from 'react';

import './App.css';

import Posts from './Posts';

function App() {

return (

<div className="App">

<h1>Blog App</h1>

<Posts />

</div>

);

}

export default App;

**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: []

};

}

// Fetch posts from API and map them to Post model

loadPosts = () => {

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

.then(response => response.json())

.then(data => {

const postList = data.map(

post => new Post(post.userId, post.id, post.title, post.body)

);

this.setState({ posts: postList });

})

.catch(error => {

throw error;

});

};

// Load posts after component mounts

componentDidMount() {

this.loadPosts();

}

// Handle unexpected rendering errors

componentDidCatch(error, info) {

alert("An error occurred in Posts component:\n" + error);

}

render() {

return (

<div>

<h2>Posts</h2>

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

<div key={post.id}>

<h3>{post.title}</h3>

<p>{post.body}</p>

</div>

))}

</div>

);

}

}

export default Posts;

**Output:**



