11. useEffect and Side Effects

Last updated by | Subramanya Dixit | May 5, 2025 at 7:59 PM GMT+5:30

📘 Why useEffect is Important

`useEffect` is a powerful hook that allows React developers to handle side effects such as data fetching, subscriptions, and manual DOM manipulation without directly interacting with class-based lifecycle methods. It provides a cleaner and more declarative way to handle side effects in functional components.

🧠 Key Concepts

✅ Side Effects in React

Side effects are operations that interact with the outside world (e.g., API calls, timers, DOM manipulations) or require cleanup (e.g., subscriptions, event listeners). React handles side effects in components with the `useEffect` hook.

✅ Basic Usage of useEffect

The `useEffect` hook is used to perform side effects in function components.

```
import { useState, useEffect } from 'react';

function Timer() {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => setSeconds(s => s + 1), 1000);
    return () => clearInterval(timer); // Cleanup the timer on unmount
  }, []); // Runs only once after the first render

  return <h1>Time: {seconds} seconds</h1>;
}
```

✅ Dependencies Array

The second argument to `useEffect` is the **dependency array**, which determines when the effect runs.

- **Empty array (`[]`)**: Runs only once, after the component mounts (similar to `componentDidMount`).
- **List of dependencies**: Runs the effect whenever the listed values change.

  useEffect(() => {
  // Effect runs only when 'count' changes
  console.log('Count changed:', count);
  }, [count]);

✅ Cleanup in useEffect

The function passed to `useEffect` can return another function that will be run when the component unmounts or before the effect runs again (useful for cleanup).

```
useEffect(() => {
  const interval = setInterval(() => console.log("Running..."), 1000);
  return () => clearInterval(interval); // Cleanup on unmount
}, []);
```

📊 Visual Overview

```
graph LR
  A[Component Render] --> B[Effect Runs]
  B --> C[Update State]
  C --> B[Effect Runs Again (if dependencies change)]
  B --> D[Effect Cleanup (if specified)]
```

💡 Guidelines

- Use `useEffect` for side effects that require interaction with external systems (APIs, DOM, etc.).
- Remember to clean up resources using the return function in `useEffect` (especially for timers, subscriptions).
- Use the dependency array wisely to control when the effect runs.

📝 Practice Exercises

1. Set up a timer that increments a counter every second.
2. Fetch data from an API when the component mounts and display the result.
3. Set up an event listener in `useEffect` and clean it up when the component unmounts.

❓ Quiz Questions

1. What is the purpose of the dependency array in `useEffect`?
a) To control how often the effect runs
✅ **b) To specify when the effect should be re-executed**
c) To define cleanup tasks
d) To create a side effect

---

2. When is the cleanup function in `useEffect` called?
a) When the component renders
b) When the dependencies change
✅ **c) When the component unmounts or before the effect re-runs**
d) After the effect runs the first time

---

3. What is the main purpose of `useEffect` in React?
a) To update state
b) To directly modify the DOM
✅ **c) To handle side effects such as fetching data or setting up event listeners**
d) To render JSX
