Welcome Geeks

Performance Optimisations

React re-renders components when their parent re-renders, which can impact performance, especially in complex apps. React.memo and useCallback are tools to avoid unnecessary re-renders and function re-creations.

React.memo for Memoizing Components

- It memoizes the result of a functional component’s render.
- Re-renders the component only if its props change.

```jsx
import React from 'react';

// Simple component that displays a name
const ChildComponent = ({ name }) => {
  console.log('ChildComponent rendered');
  return <div>Hello, {name}!</div>;
};

// Memoize the ChildComponent to prevent unnecessary re-renders
export default React.memo(ChildComponent);
```

```jsx
import React, { useState } from 'react';
import ChildComponent from './ChildComponent'; // Memoized child component

const ParentComponent = () => {
  const [count, setCount] = useState(0);

  console.log('ParentComponent rendered');

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment Count</button>

      {/* Memoized ChildComponent only re-renders when 'name' changes */}
      <ChildComponent name="John" />
    </div>
  );
};

export default ParentComponent;
```

Without React.memo:
ChildComponent would re-render every time the parent (ParentComponent) re-renders, even if the name prop hasn't changed.

With React.memo:
ChildComponent only re-renders when the name prop changes.
In this case, the name is always "John", so ChildComponent won't re-render even when the parent re-renders due to the count update.

```
function ParentComponent() {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount(count + 1);
  }, [count]); // Only recreate handleClick when count changes

  return (
    <div>
      <p>Count: {count}</p>
      <ChildComponent onClick={handleClick} />
    </div>
  );
}

function ChildComponent({ onClick }) {
  return (
    <button onClick={onClick}>Increment</button>
  );
}
```