14. Optimizing React Performance

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

📘 Why Optimizing React Performance Is Important

Optimizing React performance ensures your app runs smoothly and efficiently, especially when scaling. React apps can experience performance bottlenecks due to re-renders, large data sets, and unnecessary component updates. This topic covers techniques to improve performance in React applications.

🧠 Key Concepts

✅ Reconciliation and Virtual DOM

React uses a **Virtual DOM** to efficiently update the real DOM. It compares the Virtual DOM with the previous version and updates only the changed elements.

✅ Memoization (React.memo, useMemo, useCallback)

- `React.memo` : Prevents unnecessary re-renders by memoizing the component. It only re-renders if the props change.

  const MyComponent = React.memo(function MyComponent({ name }) {
  return

  {name}
  ;
  });

- `useMemo` : Memoizes expensive calculations to prevent them from being recalculated on every render.

  const expensiveValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

- `useCallback` : Memoizes functions to prevent them from being recreated on every render.

  const handleClick = useCallback(() => {
  console.log('Button clicked');
  }, []); // Only created once

✅ Lazy Loading and Code Splitting

React's **Lazy Loading** allows you to load components only when needed, which can improve initial load performance.

```
import React, { Suspense, lazy } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <LazyComponent />
    </Suspense>
  );
}
```

This enables you to split your bundle and load components on-demand.

✅ React Profiler

The **Profiler** API helps in identifying performance bottlenecks by measuring the render time of components.

```
import { Profiler } from 'react';

<Profiler id="App" onRender={(id, phase, actualDuration, baseDuration) => {
  console.log({ id, phase, actualDuration, baseDuration });
}}>
  <App />
</Profiler>
```

✅ Avoiding Unnecessary Re-renders

- Use `shouldComponentUpdate` (for class components) or `React.memo` (for function components) to avoid unnecessary renders.
- Optimize state updates by avoiding frequent re-renders of large lists or complex components.

📊 Visual Overview

```
flowchart TD
  A[App Component] --> B[React.memo (prevents re-renders)]
  A --> C[useMemo (memoizes expensive calculations)]
  A --> D[useCallback (memoizes functions)]
  A --> E[Lazy Loading (loads components only when needed)]
  A --> F[React Profiler (measures performance)]
  A --> G[Avoid Unnecessary Re-renders]
```

💡 Guidelines

- Use **React.memo** for components that do not change often.
- Memoize expensive calculations using `useMemo` to avoid recalculating them on every render.
- Break down large components using **lazy loading** and **code splitting** to improve initial load time.
- Profile your app with the **React Profiler** to identify performance bottlenecks.

📝 Practice Exercises

1. Memoize a component with `React.memo` and observe the performance improvement.
2. Implement **lazy loading** for a component in your app.
3. Use `useMemo` and `useCallback` in your app to optimize expensive calculations and functions.
4. Profile your app using **React Profiler** and identify slow components.

❓ Quiz Questions

1. What is the purpose of `React.memo`?
a) To prevent re-renders of the app
✅ **b) To prevent re-renders of a component when props don't change**
c) To update the component's state
d) To make the component asynchronous

---

2. How does lazy loading improve React app performance?
a) By reducing memory usage
b) By loading components only when needed
✅ **c) By splitting the code into smaller bundles that are loaded on demand**
d) By reusing components

---

3. What is useMemo used for in React?
a) To prevent functions from being recreated
b) To optimize rendering by memoizing expensive calculations
✅ **c) To prevent unnecessary recalculations during re-renders**
d) To fetch data from an API
