


Optimizing Performance in React Applications

follow



chetan khulage

for more content like this

 Like  Comment  Repost  Share

1. Memoization with React.memo()

Memoization is a technique to optimize rendering performance by memoizing the result of a component's render. `React.memo()` is a higher-order component that memoizes the rendered output of a component, preventing unnecessary re-renders.



```
1  const MemoizedComponent = React.memo((props) => {  
2    // Component Logic  
3  });
```

2. Lazy Loading with React.lazy()

Lazy loading is a technique to defer the loading of non-essential parts of your application until they are actually needed. `React.lazy()` allows you to load components lazily, improving initial load times.



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

3. Code Splitting with React Suspense

Code splitting involves breaking your code into smaller chunks that are loaded on-demand. React Suspense, in combination with `React.lazy()`, enables you to easily implement code splitting in your React application.



```
1  const LazyComponent = React.lazy(() => import('./LazyComponent'));
2
3  function App() {
4    return (
5      <React.Suspense fallback={<div>Loading...</div>}>
6        <LazyComponent />
7      </React.Suspense>
8    );
9  }
10
```

4. Virtualized Lists for Large Data Sets

When dealing with large data sets, rendering all items at once can lead to performance issues. Virtualized lists, such as React Window or React Virtualized, only render the items visible in the viewport, significantly improving performance.



```
1 import { FixedSizeList } from 'react-window';
2
3 function VirtualizedList() {
4   return (
5     <FixedSizeList height={400} width={300} itemSize={50} itemCount={1000}>
6       {( { index, style } ) => <div style={style}>Item {index}</div>}
7     </FixedSizeList>
8   );
9 }
```

5. Web Workers for Background Processing

Offloading heavy computations or tasks to Web Workers can prevent blocking the main thread and enhance the overall responsiveness of your React application. Libraries like comlink make it easier to use Web Workers with React.



```
1 // In a separate file (worker.js)
2 const workerCode = () => {
3     // Worker logic
4 };
5
6 const worker = new Worker(URL.createObjectURL(new Blob([`${workerCode}()`])));
7
8 // In your React component
9 useEffect(() => {
10     worker.postMessage({ /* message */ });
11 }, []);
```

follow



chetan khulage
for more content like this

 Like

 Comment

 Repost

 Share