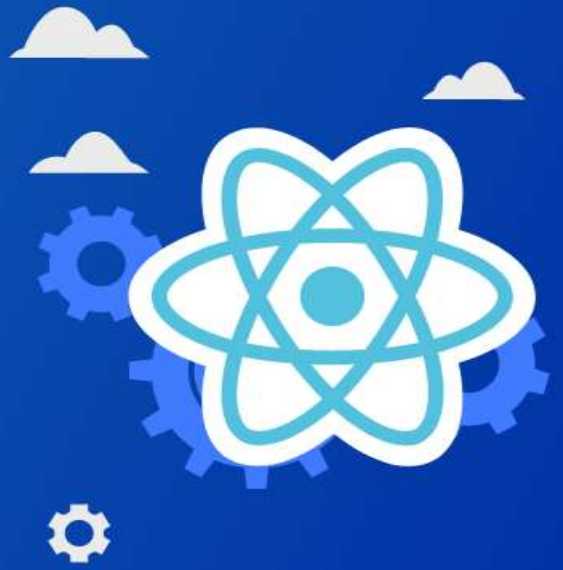
# 5 Go-To Performance Optimization Techniques For a React App

https://fibonalabs.com/

Fibonalabs

# 5 GO-TO PERFORMANCE OPTIMIZATION TECHNIQUES FOR A REACT APP



Ensuring fast and seamless user experience in a web app is a critical aspect of attracting and subsequently retaining users. Walmart reported increased conversion by 2 percent per the second improvement in load time and Amazon recorded a 1 percent loss in revenue per 100ms site load delay. Given how big these giants are, the numbers translate to millions of dollars gained and lost based on a fraction of a second. Having established the importance of performance, the key to increase performance is to utilize various optimization techniques which are behind the curtain. In this article, I'll list out a few techniques in React.

**React.memo and React Pure Component**

Consider this react component where the parent component "componentParent" has two child components, "componentOne" and "ComponentTwo".

ComponentOne has a button that increases the count state set in the parent component. So, ideally when countOne changes on the click of the button, only componentOne should re-render. That is not the case with componentTwo re-rendering as well despite its props remaining unchanged. This is due to the parent component re-rendering and can subsequently cause a performance bottleneck if there are a lot of child components under the parent. It is the same scenario with the render method in class components as well.

In order to optimize the re-rendering process, react authors came up with React.memo and React pure component for functional and class components respectively, both provide a way to compare old and new props and re-rendering only if there is a change.

Wrap componentTwo in memo( ) function and it will only render if the value of propsTwo changes.

React Pure component does the same job but for class components.

It seems like a no-brainer to implement the above optimization in every complex component that you write. But the fact of the matter is that these come with costs of their own while comparing complex props or when components are passed in as props. Careful considerations are necessary while implementing these and it is the reason that they don't come baked in with React under the hood.

## Virtualize long lists

Long lists or tables with many rows in React can have a substantial impact on performance since it involves rendering and re-rendering a lot of DOM nodes. React documentation recommends a method known as "windowing". This technique renders only a small subset of the data within the viewport, upon scrolling more data is rendered, hence at any given time only the data visible in the viewport is rendered, increasing the performance. Tools that implement windowing need not be built from scratch since there are exciting libraries available. "react-window" and "react- virtualized" are two of the popular ones.

## Code splitting and Suspense

Lazy Loading is made possible in React with React.Lazy. This feature prevents unnecessary loading of components unless it is absolutely required. As a general rule of thumb, the lesser you need to load, the better the performance. React.lazy can be combined with react suspense to provide fallback content. For an in-depth explanation about the topic, you can checkout our lazy loading and code splitting blogpost.

## Avoiding Anonymous Functions and Object Literals

Using anonymous functions for callbacks and event handlers inside a component can affect the performance in a deleterious way.

Anonymous functions aren't persistent when a component gets re-rendered due to the fact that it is not assigned an identifier via const/let/var. This causes JavaScript to allocate new memory for the function each time a component is rendered and hence affects the performance in a negative way. The fix for this is fairly simple and all you have to do is add a named function.

In functional components, inline anonymous functions are a healthy compromise given that the solution involves using the useCallback hook which comes with its overheads and might cause performance drops if overused.

Using Object literals is a similar issue. If you pass in an object defined via an object literal as a prop, the object will be recreated per each render.

Defining the object outside the component saves it once in memory during the initial render and hence fixes the issue.

**Avoid Frequent Mounting/Unmounting**

Conditionals or ternary operators are commonly used methods to make a component disappear. But it is a costly operation since it can cause the browser to repaint and reflow. Positions of DOM elements will have to be recalculated in that case, which is expensive. Instead of completely un-mounting components, we can set CSS opacity or visibility values to zero in order to hide the component. This way, the component will still be in the DOM but invisible, without any performance cost.

## Conclusion

Implementing some of the techniques listed above will help you to improve the performance of your React app. Having said that, performance optimization techniques need to be validated with benchmarks analysis in order to measure the overall effect they have on app load times and the in-app experience.

# THANK YOU