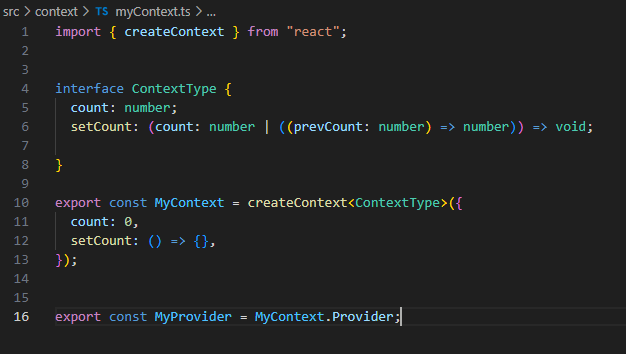
**How to use React Hooks ?**

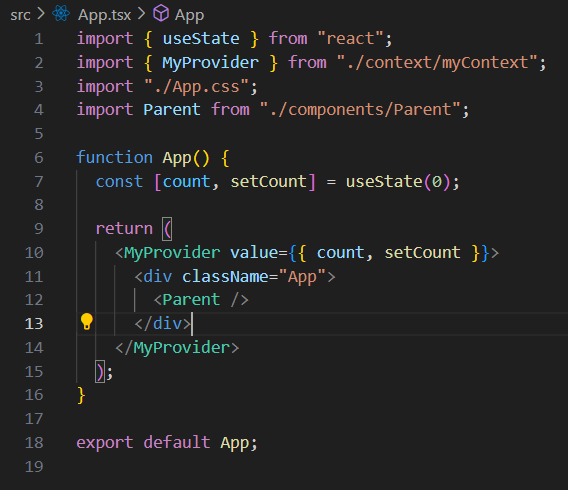
**Context API:**

To use Context API follow these steps:

1. Create a context object
   * Create a folder called context in your src directory.
   * Create a filed called myContext.ts in it.
   * Write the following code to create a context and a provider.



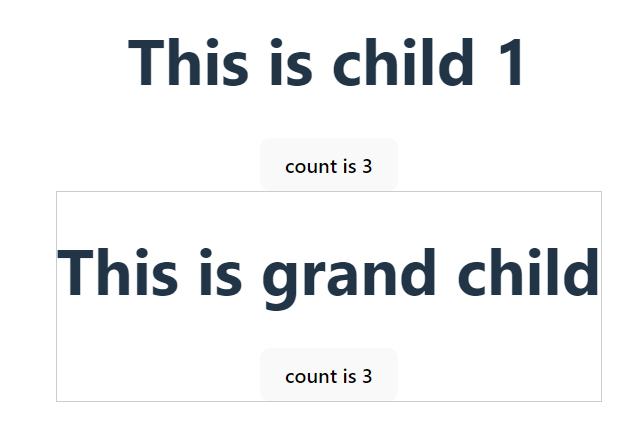
1. Wrap your app with your provider:
   * Import your provider in your App.tsx
   * Wrap your components with your provider send the default value in the props like this



1. Create a parent, child and grandchild component to see the beauty of contextAPI.
   * Create a parent component and render a child component in It.
   * In the child component render a grandchild component.
2. Using the state:
   * In your child component you can use your state like this:



1. **Use the exact same code in your grandchild and you will see that the state is same. If you change it in the child or grandchild it will be shown in both of them. This will solve the problem of prop drilling and make your code cleaner.**

****

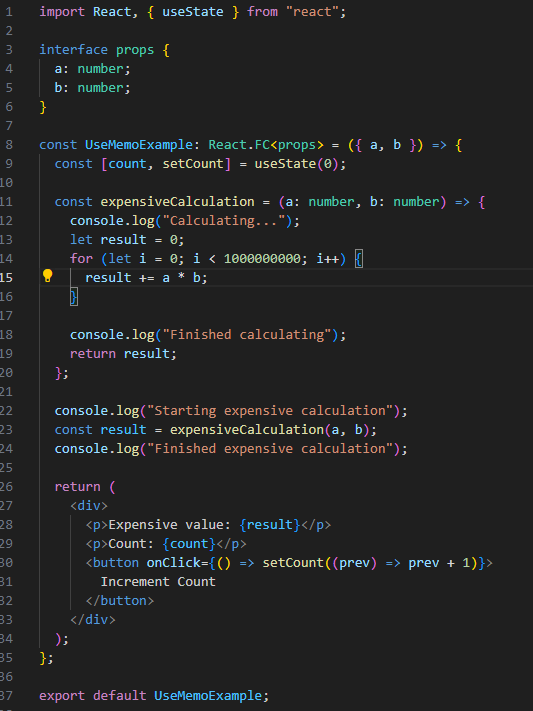
As in this example when I increment count it increments in child and grandchild as well. The state is shared.

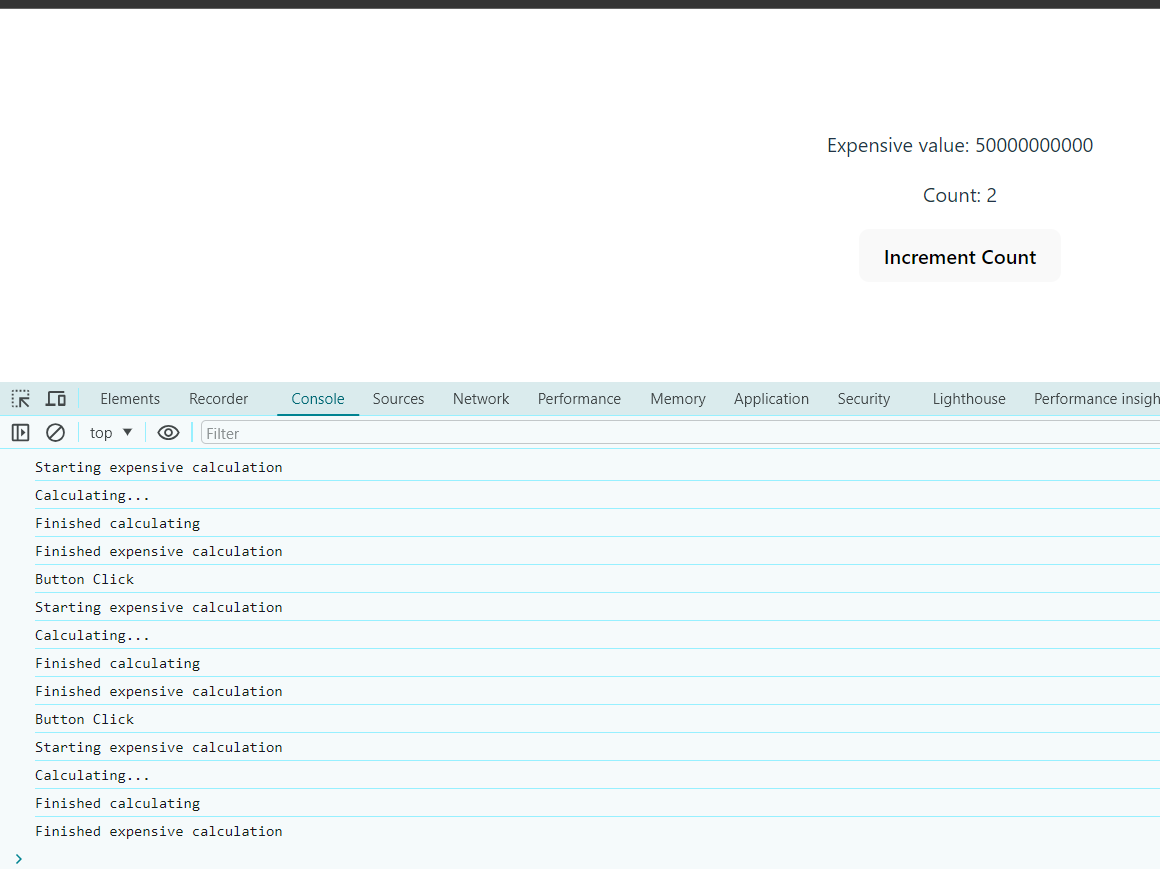
**Use Memo:**

Why do we need useMemo?

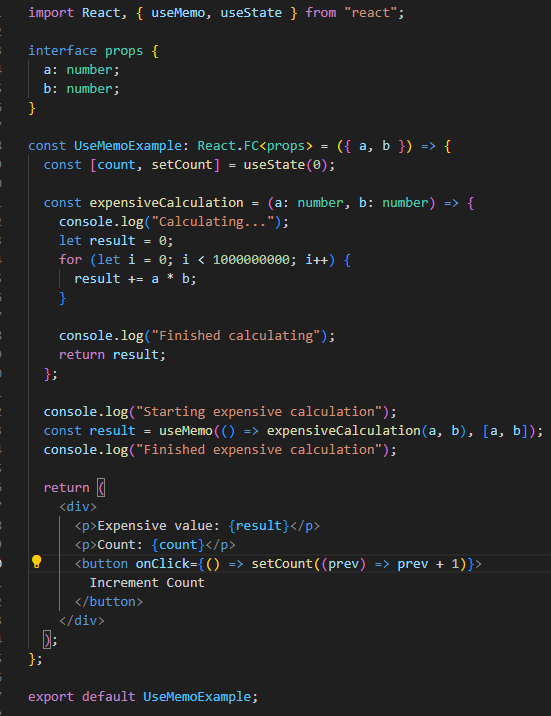
It is a hook that memoizes the result of a computation to avoid recalculating it on every render. It is useful when you have expensive calculations that you don't want to re-run unnecessarily.

For example, in the following component we have an expensive calculation which returns the same result every time. Whenever we change out state the component renders and the component lags.

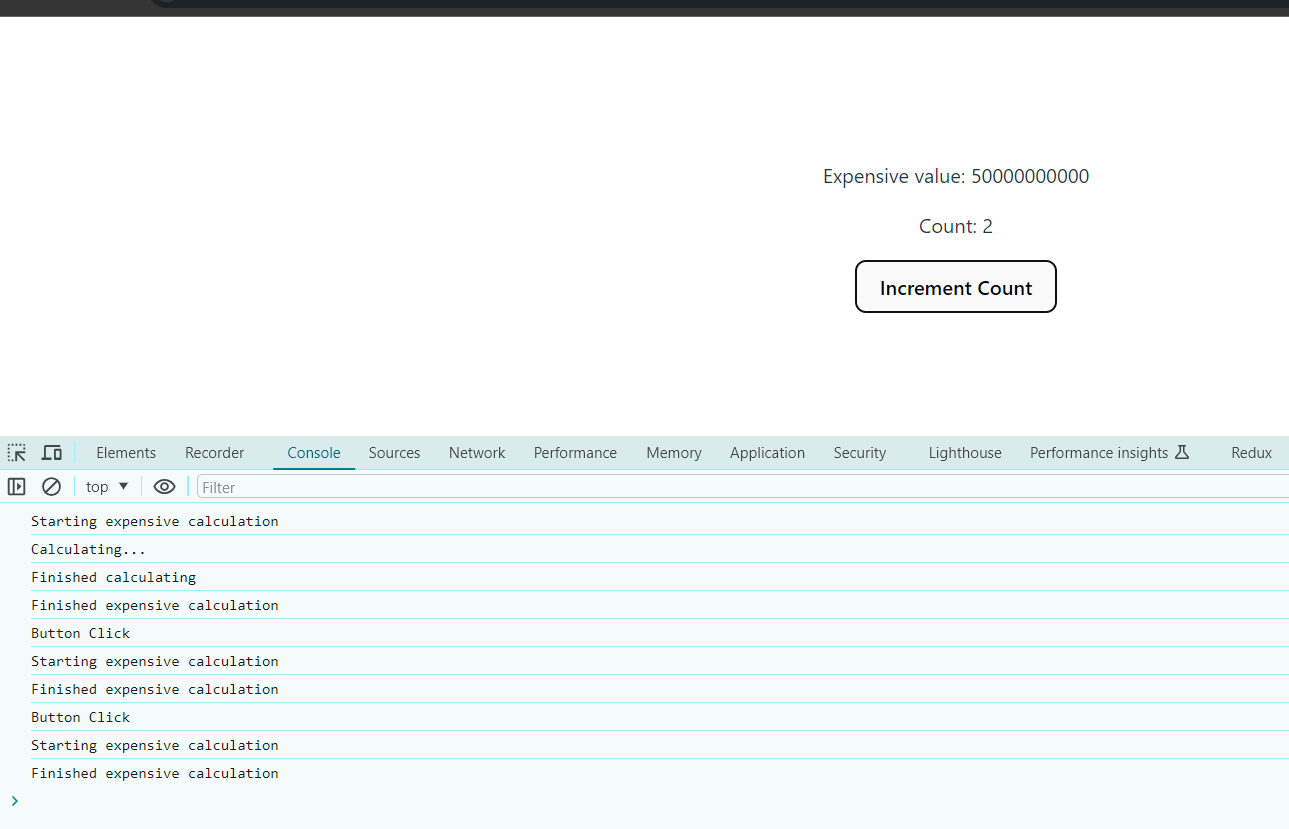
****

****

To avoid this from happening we will use useMemo Hook to memoize the expensive calculation so we don’t have to calculate it on each render.



Now it will only execute the first time the component renders or if the values in the dependency array changes.



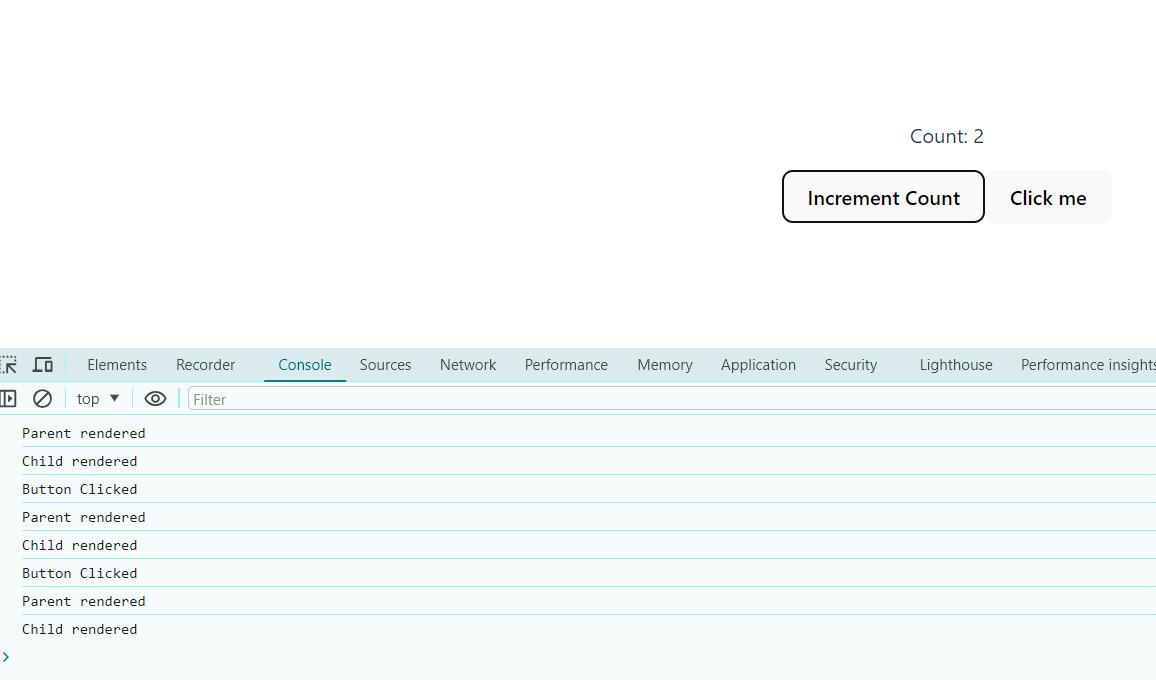
**Use Callback:**

The useCallback hook is needed to memoize functions and prevent unnecessary re-creations of those functions on every render. This is particularly important in scenarios where the function is passed as a prop to a child component. Without useCallback, a new function instance would be created on every render, potentially causing the child component to re-render even when it doesn't need to.

In the following example I will demonstrate this.

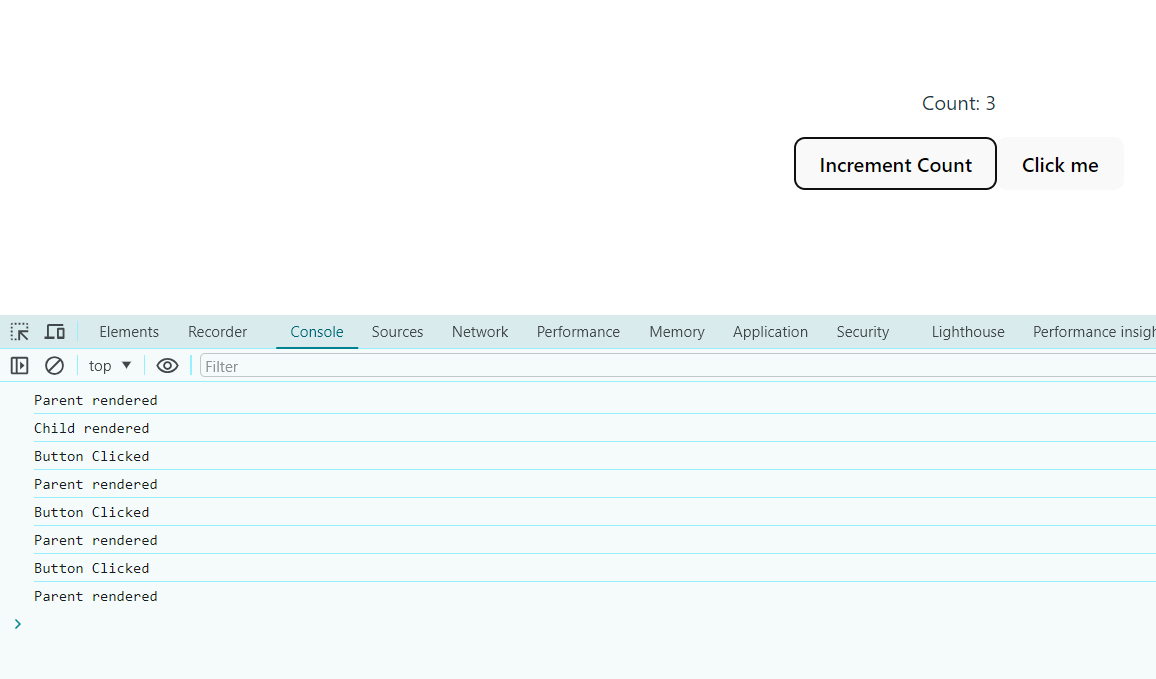


Using this code, the function handleClick is re-created every time the parent re renders causing the child component to re render as well.



To avoid the unnecessary re rendering of our child component we can use useCallback hook to memoize our function so its stays the same across all re renders and is not recreated. This will make our child component not re render unnecessarily.





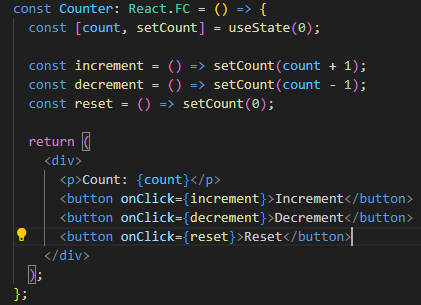
As you can see now when the parent is re rendered the child does not re render as well fixing the issue.

**Use Reducer:**

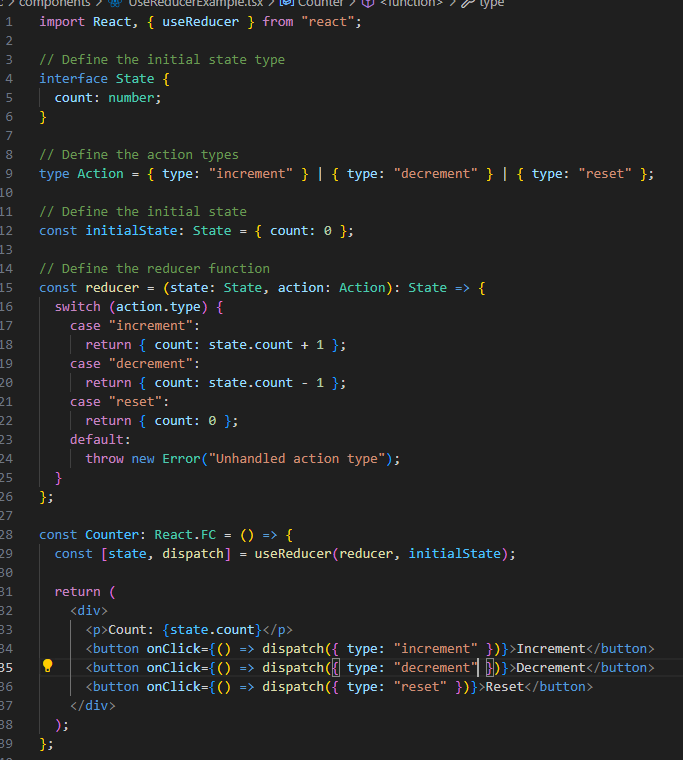
Using useReducer enables you to handle complex state transitions more effectively than useState especially when the state depends on multiple sub-values or when the next state depends on the previous one.

In the below example I will show you how it is different.

**Without use Reducer**



**With use Reducer**

****

**With this your code is much more maintainable**

**Use Ref:**

Use ref is used for several reasons:

1. **Direct DOM Access:**

useRef provides a way to access and interact with DOM elements directly, which is not possible with state or props alone.

1. **Avoiding Re-renders:**

Unlike state, updating a ref does not trigger a component re-render. This makes it ideal for storing mutable values that need to persist across renders without causing performance issues.

1. **Storing Mutable Values:**

useRef can be used to store any mutable value that you want to persist between renders. This can be useful for storing intervals, timers, or other mutable values that should not cause re-renders.

**For example:**

****

In this example useRef hook creates a variable(inputRef) that can hold a reference to an Html input element. On the button click we are focusing the input field.