Slim

React js tips

How useCallback Boosts Your App

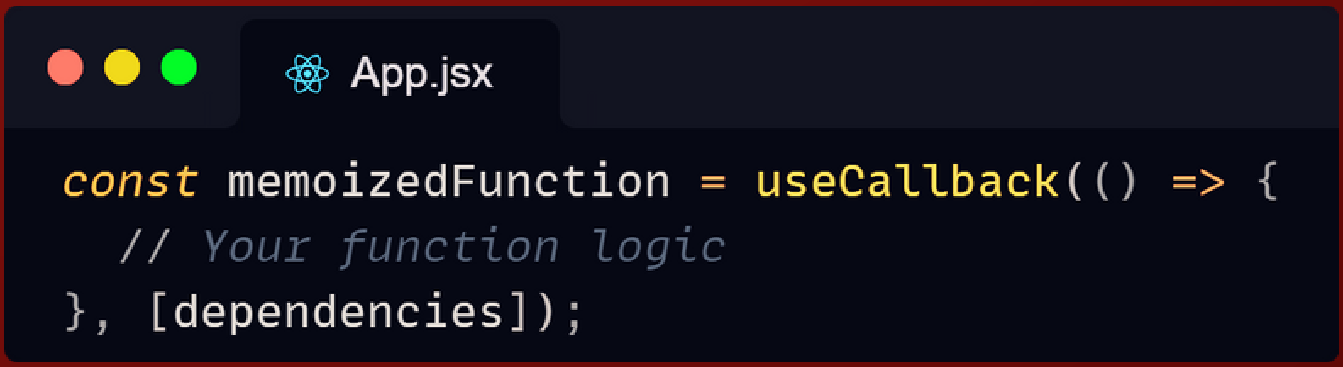
What is useCallback ?

useCallback is a React Hook that saves a function and reuses it unless its dependencies change.
Think of it like this: React remembers your function and only updates it if something it depends on changes.

Why use useCallback ?

1. **Performance Optimization:**
   - Without useCallback, React recreates functions on every render. This can cause unnecessary work and re-renders.
2. **Stable Dependencies:**
   - When a function is passed as a prop or used inside another hook, React might think it's a new function every time. useCallback ensures the function remains the same unless its dependencies change.

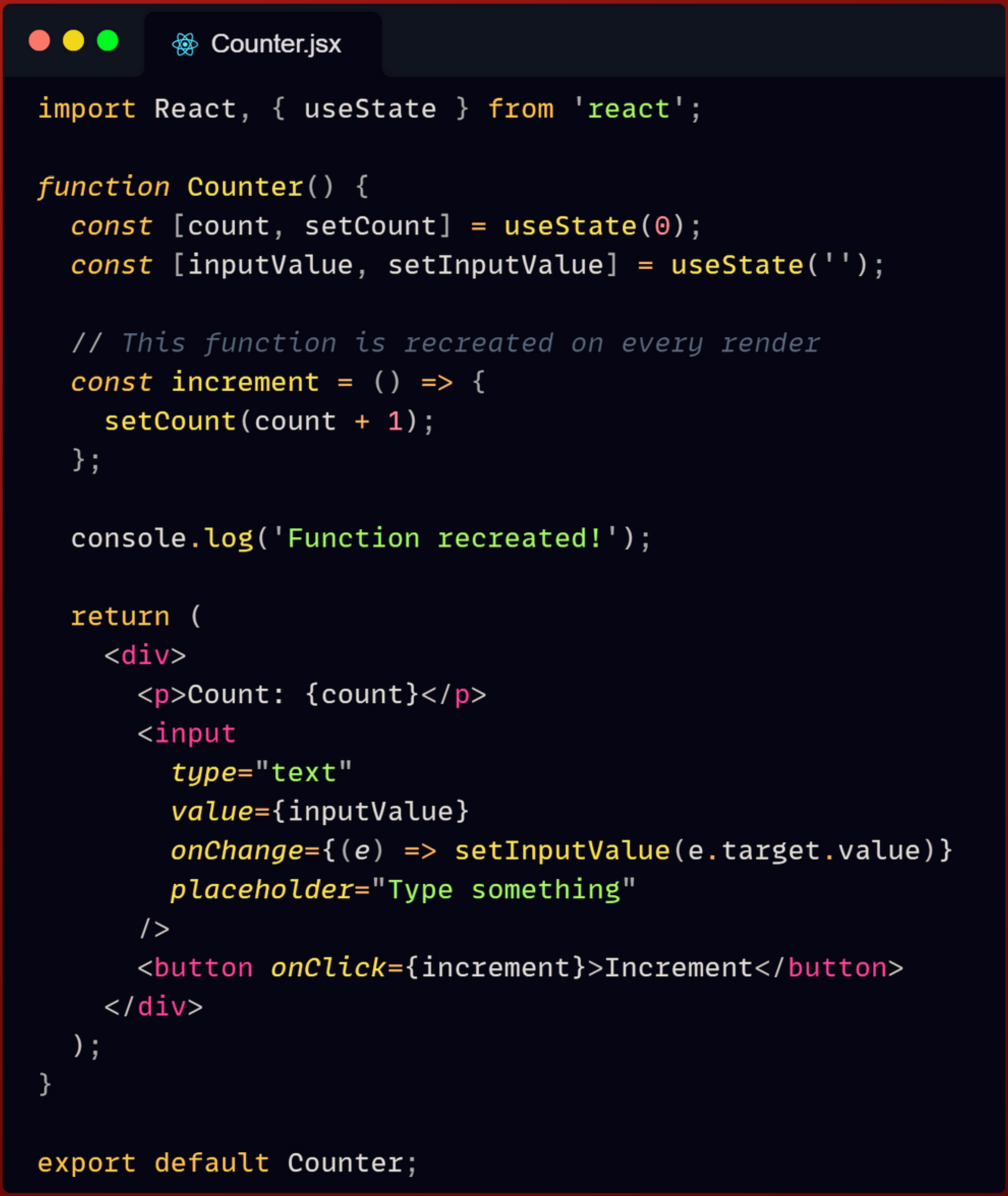
Basic syntax :


App.jsx

```
const memoizedFunction = useCallback(() => {
  // Your function logic
}, [dependencies]);
```

- **Function logic**: The code inside the function you want to memoize.
- **Dependencies**: An array of values that the function depends on. The function will be updated only when these values change.

Example Without useCallback



Counter.jsx

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  const [inputValue, setInputValue] = useState('');

  // This function is recreated on every render
  const increment = () => {
    setCount(count + 1);
  };

  console.log('Function recreated!');

  return (
    <div>
      <p>Count: {count}</p>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
        placeholder="Type something"
      />
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

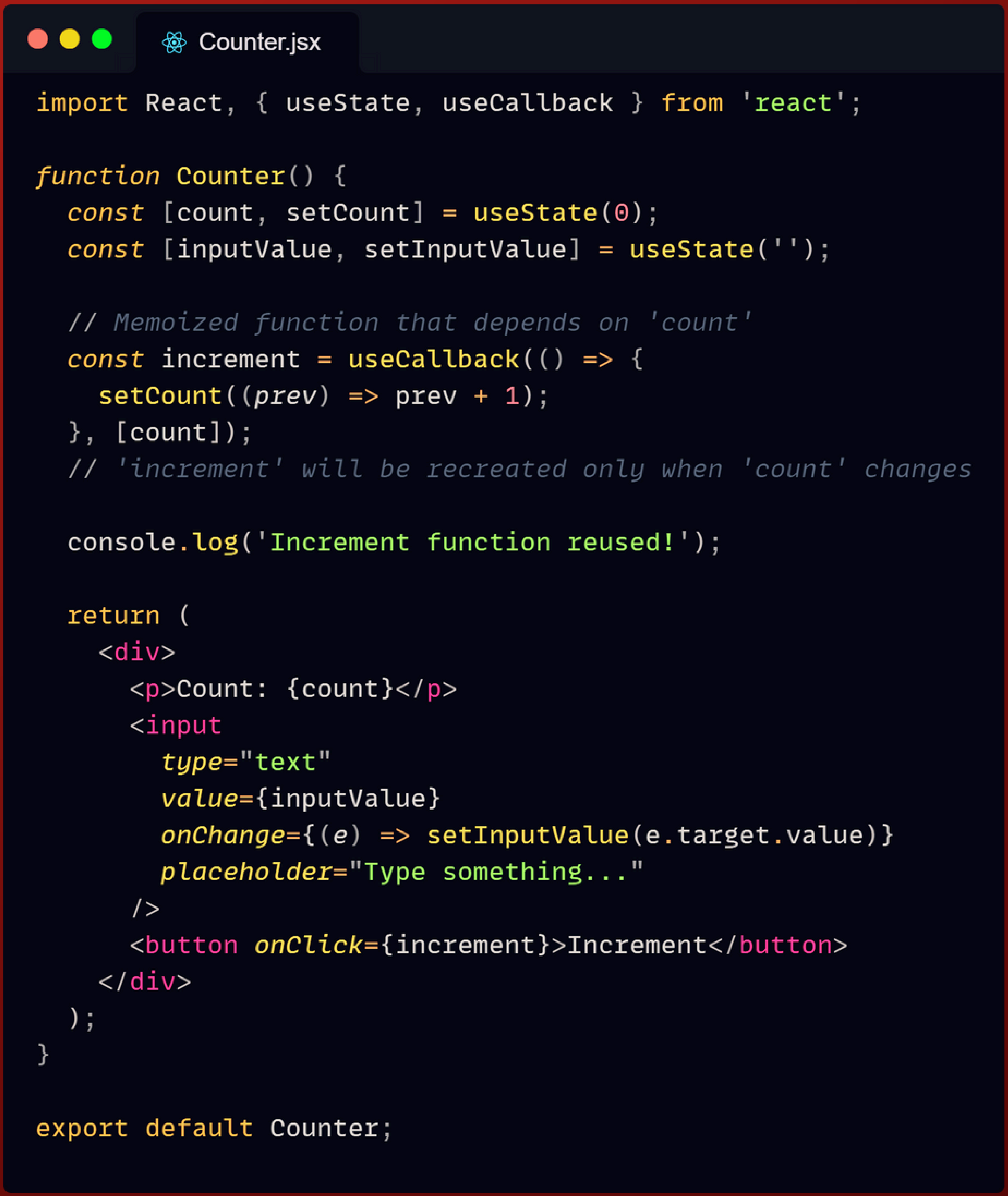
Explanation Without useCallback

- **What Happens?**
  - Every time you type in the input, the onChange updates the inputValue, causing the Counter component to re-render.
  - On each re-render, the increment function is recreated, even though it’s not related to the input field.

- **Performance Issue:**
  - Unnecessary Re-renders: The increment function is recreated on every render, wasting time and resources.
  - Increased Memory Use: Every new render creates a new function, using up memory.

- **Proof:**
  - Open the browser console. Every time you type in the input, you'll see "Function recreated!", showing that the function is recreated every time.

Example With useCallback



```jsx
import React, { useState, useCallback } from 'react';

function Counter() {
  const [count, setCount] = useState(0);
  const [inputValue, setInputValue] = useState('');

  // Memoized function that depends on 'count'
  const increment = useCallback(() => {
    setCount((prev) => prev + 1);
  }, [count]);
  // 'increment' will be recreated only when 'count' changes

  console.log('Increment function reused!');

  return (
    <div>
      <p>Count: {count}</p>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
        placeholder="Type something..."
      />
      <button onClick={increment}>Increment</button>
    </div>
  );
}

export default Counter;
```

Explanation With useCallback

- **What Happens?**
  - Every time you type in the input, the onChange handler updates the inputValue, causing the component to re-render.
  - However, with useCallback, the increment function is not recreated on every render. It only gets recreated when the count state changes.

- **Performance Benefit:**
  - Efficient Re-renders: The increment function stays the same between renders as long as count hasn't changed. This avoids recreating the function unnecessarily, improving performance.
  - Memory Optimization: The increment function is created once and reused across re-renders, saving memory and computational resources.

- **Proof:**
  - Open the browser console. Each time you type in the input, you'll see the message "Increment function reused!", showing that the function is not recreated every time the inputValue changes. It only gets recreated when count changes.

Hopefully You Found It Usefull!

“ Be sure to save this post so you can come back to it later ”

like

Comment

Share