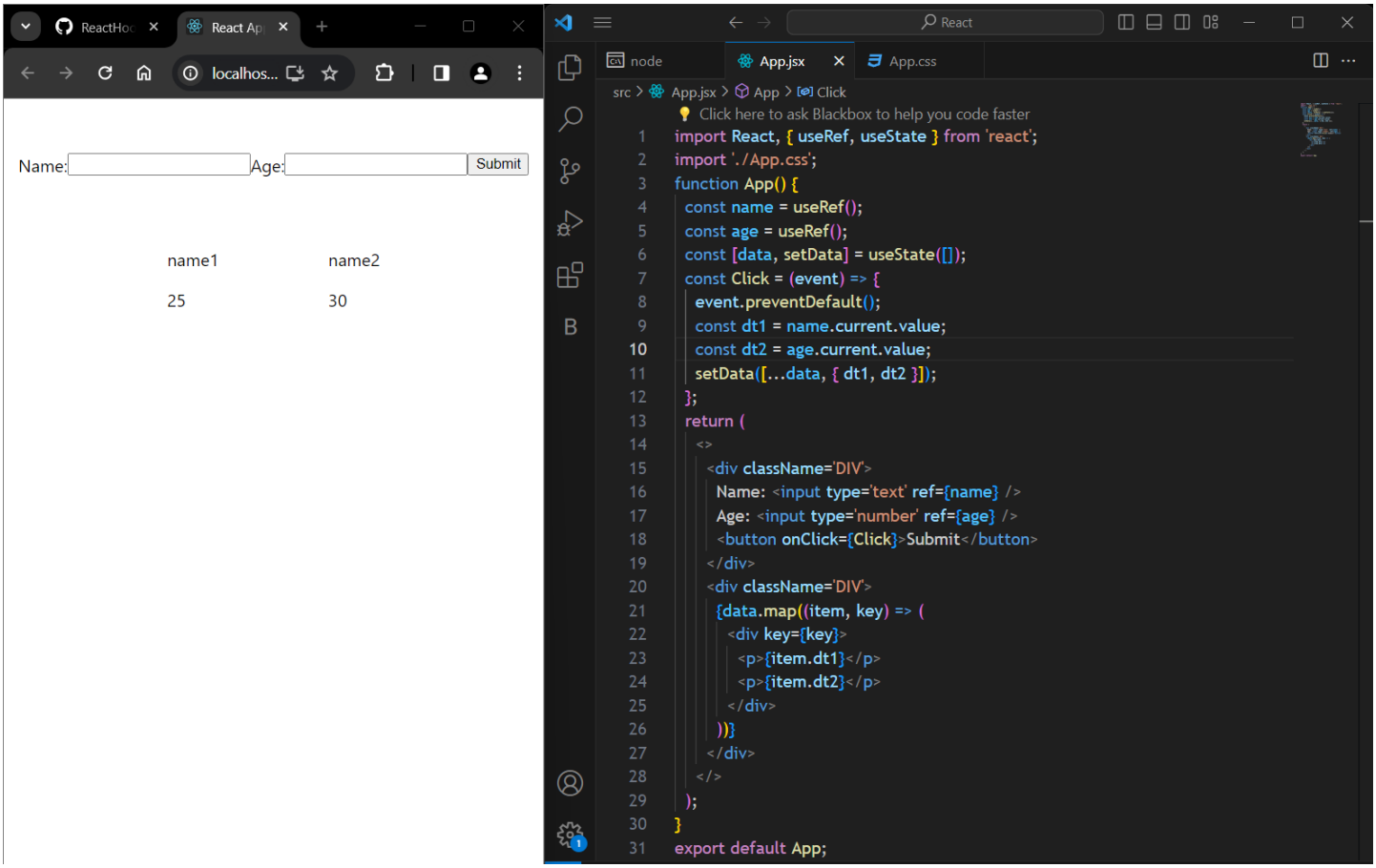
Frontend Development

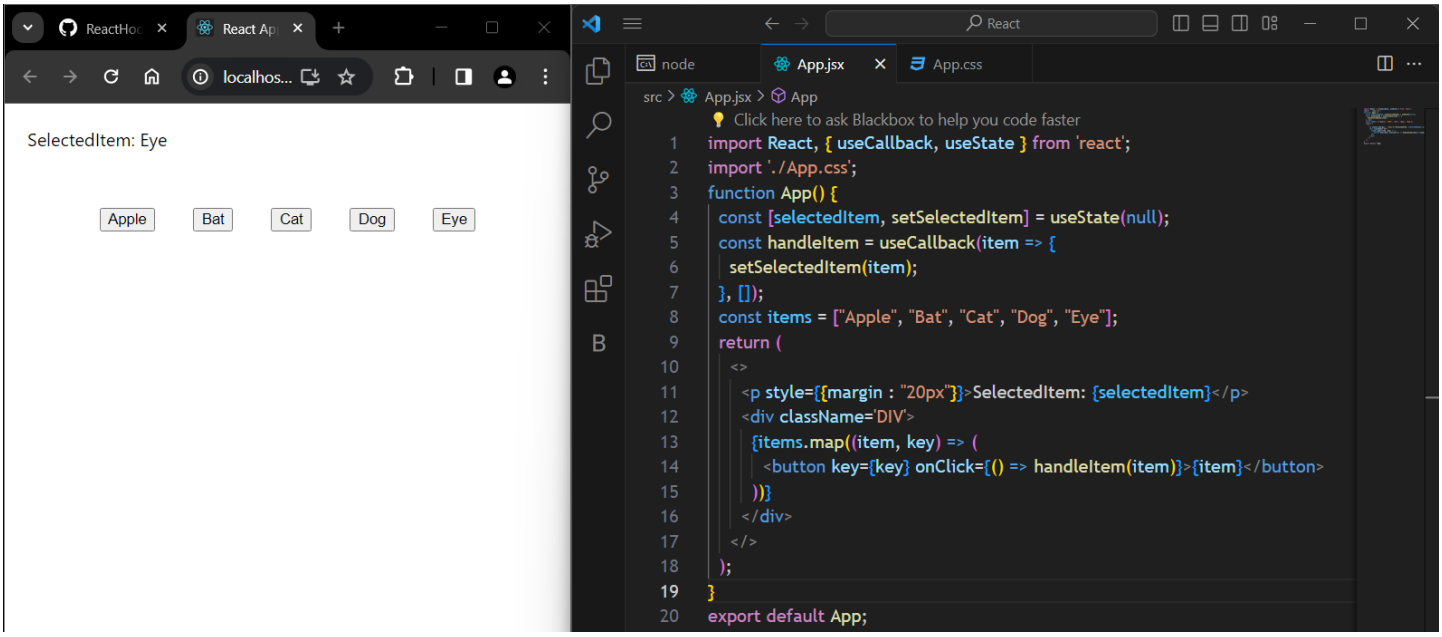
useRef Hook:

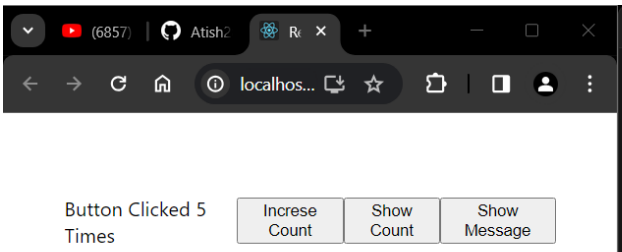
The useRef hook in React is used to create a mutable ref object whose .current property can hold a mutable value that persists across renders. It allows you to access DOM nodes or React elements directly within functional components, among other use cases.



useCallback Hook:

In React, the useCallback hook is used to memorize functions. It returns a memorized version of the callback function that only changes if one of the dependencies has changed. This is useful for optimizing performance in cases where a component re-renders frequently due to its parent re-rendering. This function will be memorized and won't change unless count changes.






```jsx
import React, { useCallback, useState } from 'react';
import './App.css';
function App() {
  const [count, SetCount] = useState(0);
  const [message, SetMessage] = useState("");

  const Dependency = useCallback(() => {
    SetMessage(`Button Clicked ${count} Times`);
  }, [count]);

  const NoDependency = useCallback (() => {
    SetMessage("Button Clicked");
  }, []);

  return (
    <>
      <div className='DIV'>
        <p>{message}</p>
        <button onClick={() => SetCount(count + 1)}>Increse Count</button>
        <button onClick={Dependency}>Show Count</button>
        <button onClick={NoDependency}>Show Message</button>
      </div>
    </>
  );
}
export default App;
```

Syntax:

```jsx
const func = useCallback (() => {
  dosomething(a, b);
}, [a, b]);
```

useCallback is a hook that returns a memorized version of the callback function that only changes if one of the dependencies has changed. It is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

In the above syntax the first argument is the callback function to memorized. The second argument is an array of dependencies. The callback will only be recalculated if one of these dependencies changes.