Name : Kunal Rajesh Kumbhare
Prn : 120A3024
Branch : IT
3rd Year (5th semester)

Experiment No: 9

Aim: - WAP to Create counter using class and Hook.

Theory :-

Hooks are the new feature introduced in the React 16.8 version. It allows you to use state and other React features without writing a class. Hooks are the functions which "hook into" React state and lifecycle features from function components. It does not work inside classes. Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.
There are multiple reasons responsible for the introduction of the Hooks which may vary depending upon the experience of developers in developing React product. Some of them are as follows:

- **Use of *'this'* keyword:** The first reason has to do more with javascript than with React itself. To work with classes one needs to understand how 'this' keyword works in javascript which is very different from how it works in other languages. It is easier to understand the concept of props, state, and uni-directional data flow but using 'this' keyword might lead to struggle while implementing class components. One also needs to bind event handlers to the class components. It is also observed by the React developers team also observed that classes don't concise efficiently which leads to hot reloading being unreliable which can be solved using Hooks.

- **Reusable stateful logics:** This reason touches advance topics in React such as Higher-order components(HOC) and the render props pattern. There is no particular way to reuse stateful component logic to React. Though this problem can be solved by the use of HOC and render props patterns it results in making the code base inefficient which becomes hard to follow as one ends up wrapping components in several other components to share the functionality. Hooks let us share stateful logic in a much better and cleaner way without changing the component hierarchy.

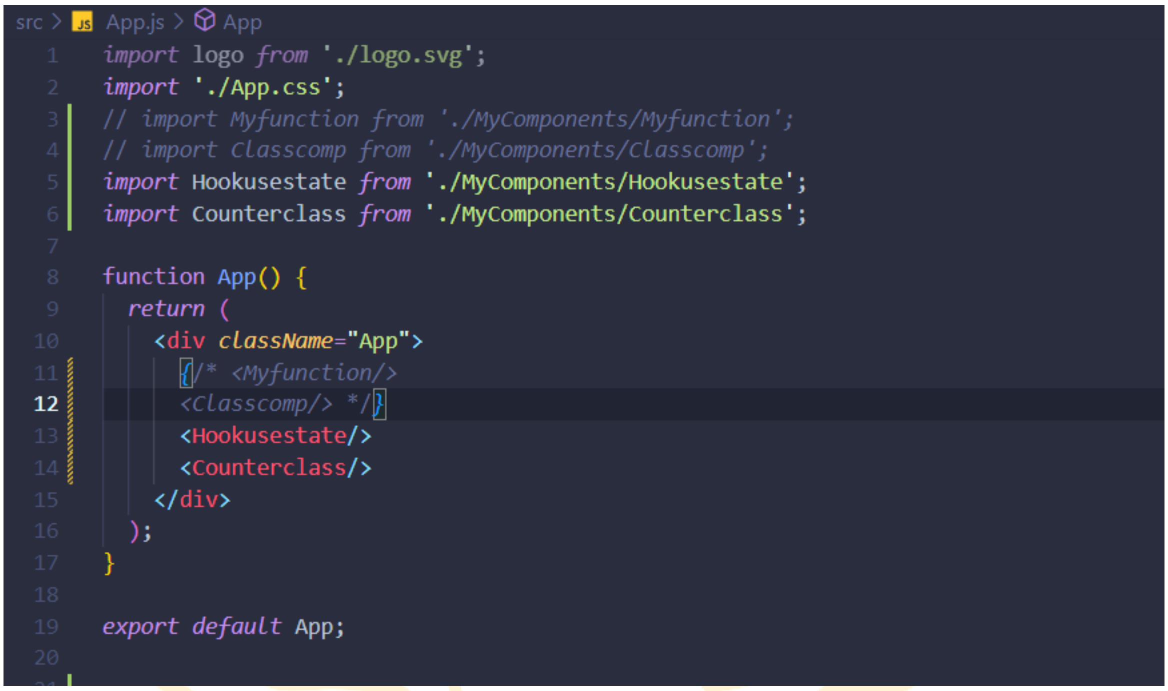
- **Simplifying complex scenarios:** While creating components for complex scenarios such as data fetching and subscribing to events it is likely that all related code is not organized in one place are scattered among different life cycle methods. For example, actions like data, fetching are usually done in componentDidMount or componentDidUpdate, similarly, in case of event listeners, it is done in componentDidMount or componentWillUnmount. These develop a scenario where completely different codes like data fetching and event listeners end up in the same code-block. This also makes impossible to brake components to smaller components because of stateful logic. Hooks solve these problems by rather than forcing a split based on life-cycle method Hooks to let you split one component into smaller functions based on what pieces are related.

The two most used hooks are the useState() hook, which allows functional components to have a dedicated state of their own, and the useEffect() hook, which allows functional components to manipulate DOM elements before each render (almost like one gets to do it in lifecycle functions).
useState() hook allows one to declare a state variable inside a function. It should be noted that one use of useState() can only be used to declare one state variable.

Code:

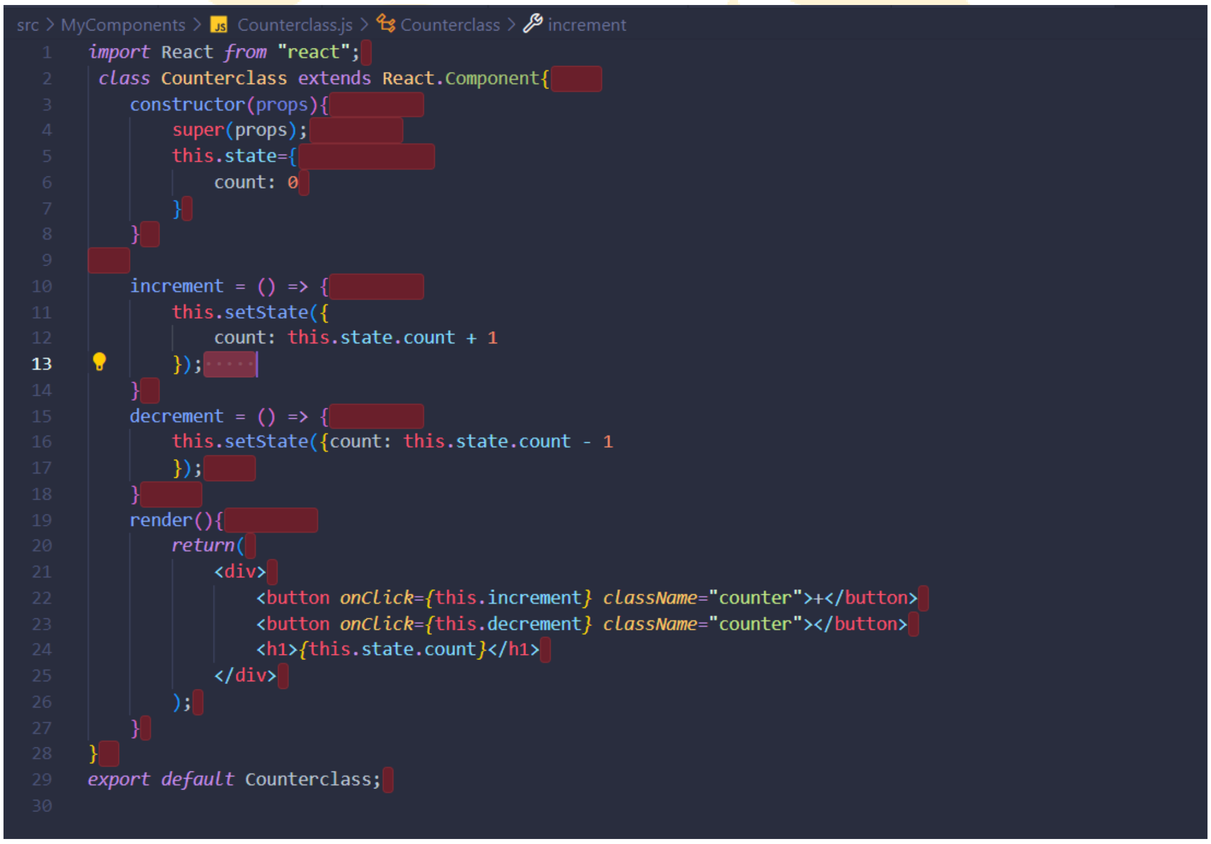
Output :

App.js



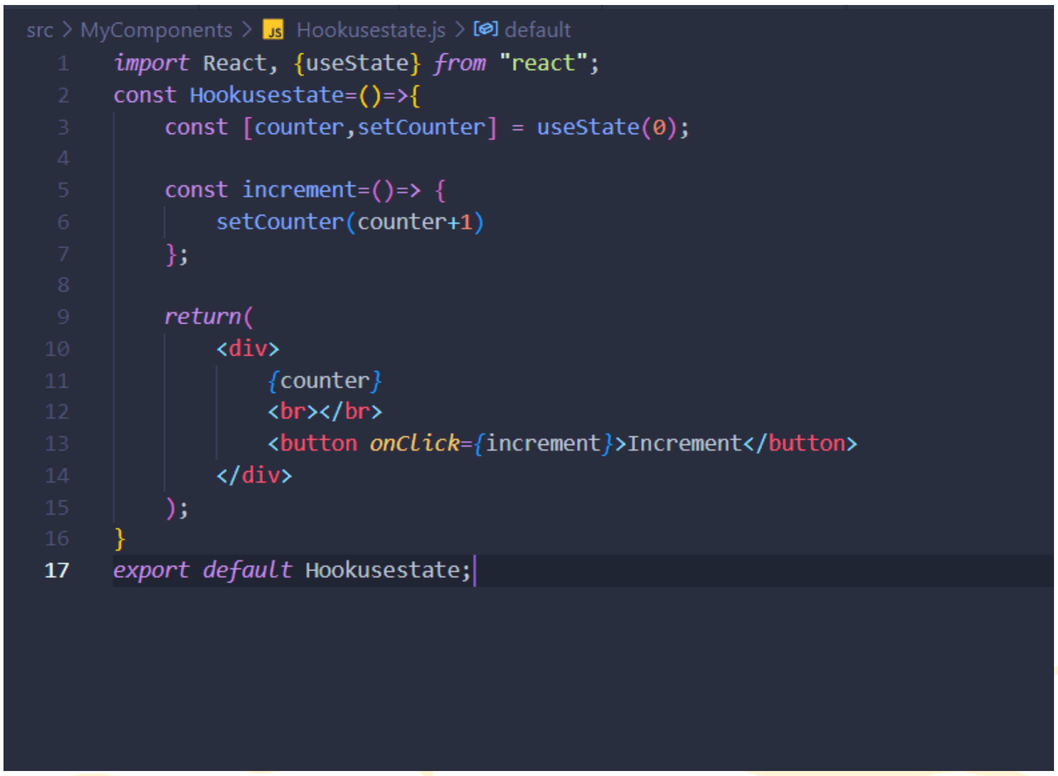
```
import logo from './logo.svg';
import './App.css';
// import Myfunction from './MyComponents/Myfunction';
// import Classcomp from './MyComponents/Classcomp';
import Hookusestate from './MyComponents/Hookusestate';
import Counterclass from './MyComponents/Counterclass';

function App() {
  return (
    <div className="App">
      {/* <Myfunction/>
      <Classcomp/> */}
      <Hookusestate/>
      <Counterclass/>
    </div>
  );
}

export default App;

```

Counterclass.js


src > MyComponents > Counterclass.js > Counterclass > increment

```
import React from "react";
 class Counterclass extends React.Component{
    constructor(props){
        super(props);
        this.state={
            count: 0
        }
    }

    increment = () => {
        this.setState({
            count: this.state.count + 1
        });
    }
    decrement = () => {
        this.setState({count: this.state.count - 1
        });
    }
    render(){
        return(
            <div>
                <button onClick={this.increment} className="counter">+</button>
                <button onClick={this.decrement} className="counter"></button>
                <h1>{this.state.count}</h1>
            </div>
        );
    }
}
export default Counterclass;

```

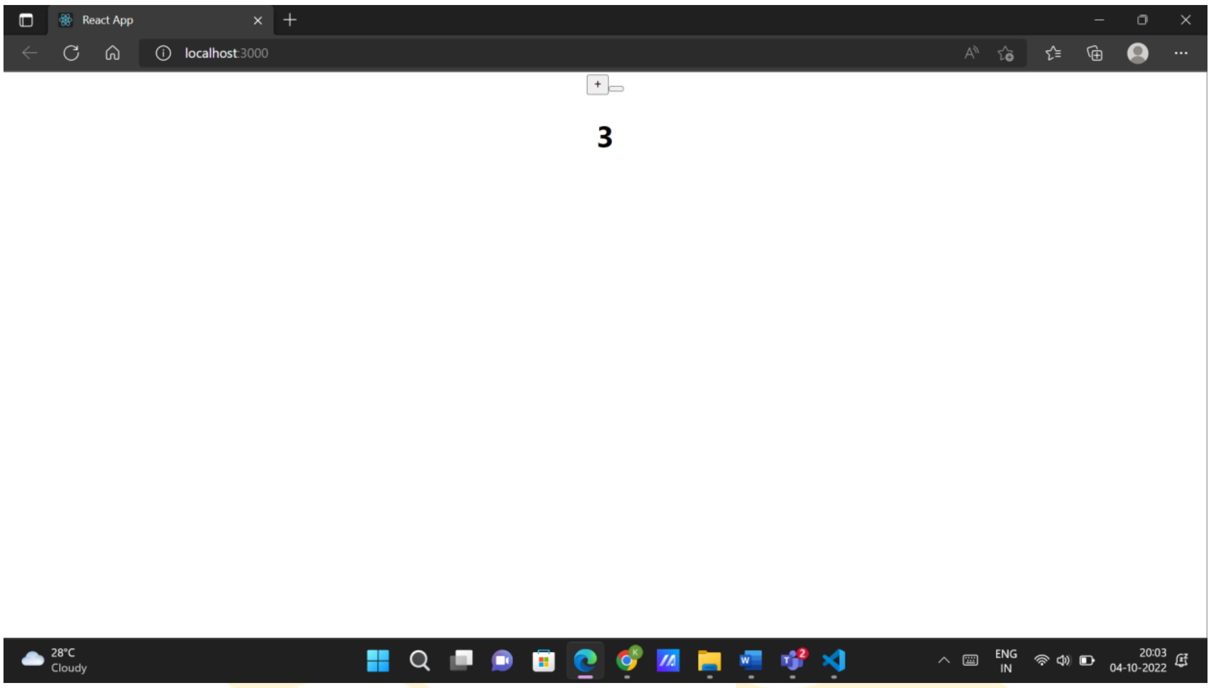
Hookusestate.js



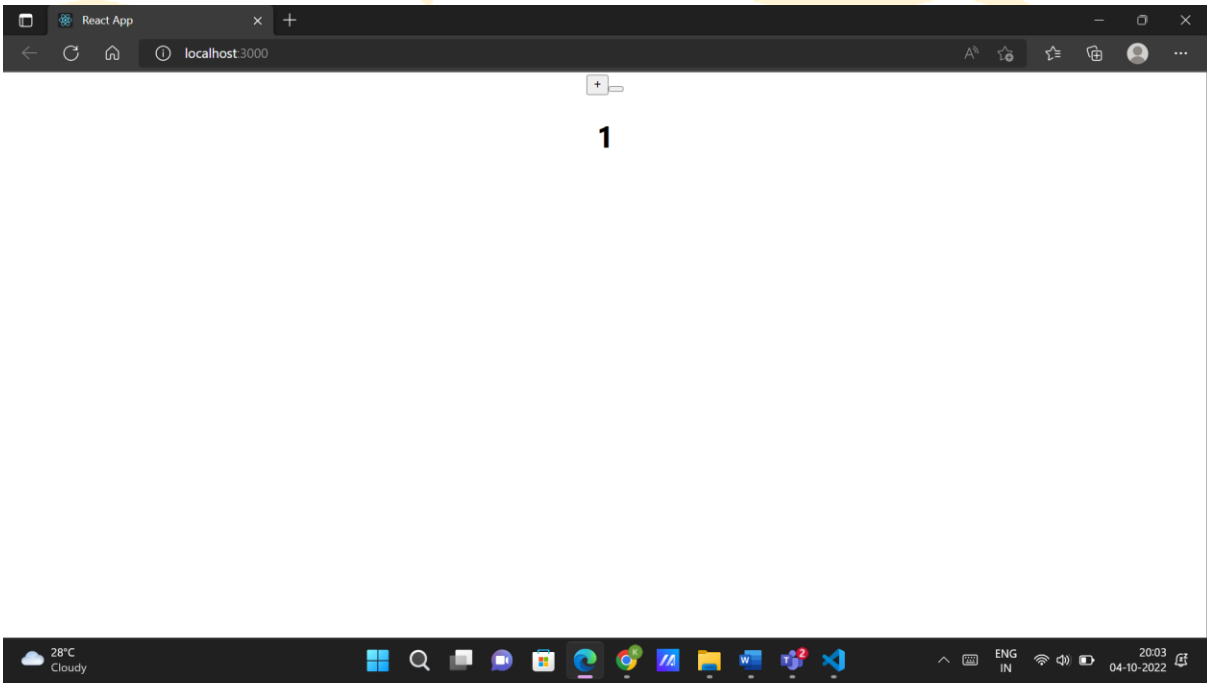
src > MyComponents > Hookusestate.js > default

```
import React, {useState} from "react";
const Hookusestate=()=>{
    const [counter,setCounter] = useState(0);

    const increment=()=> {
        setCounter(counter+1)
    };

    return(
        <div>
            {counter}
            <br></br>
            <button onClick={increment}>Increment</button>
        </div>
    );
}
export default Hookusestate;
```

Class Output:

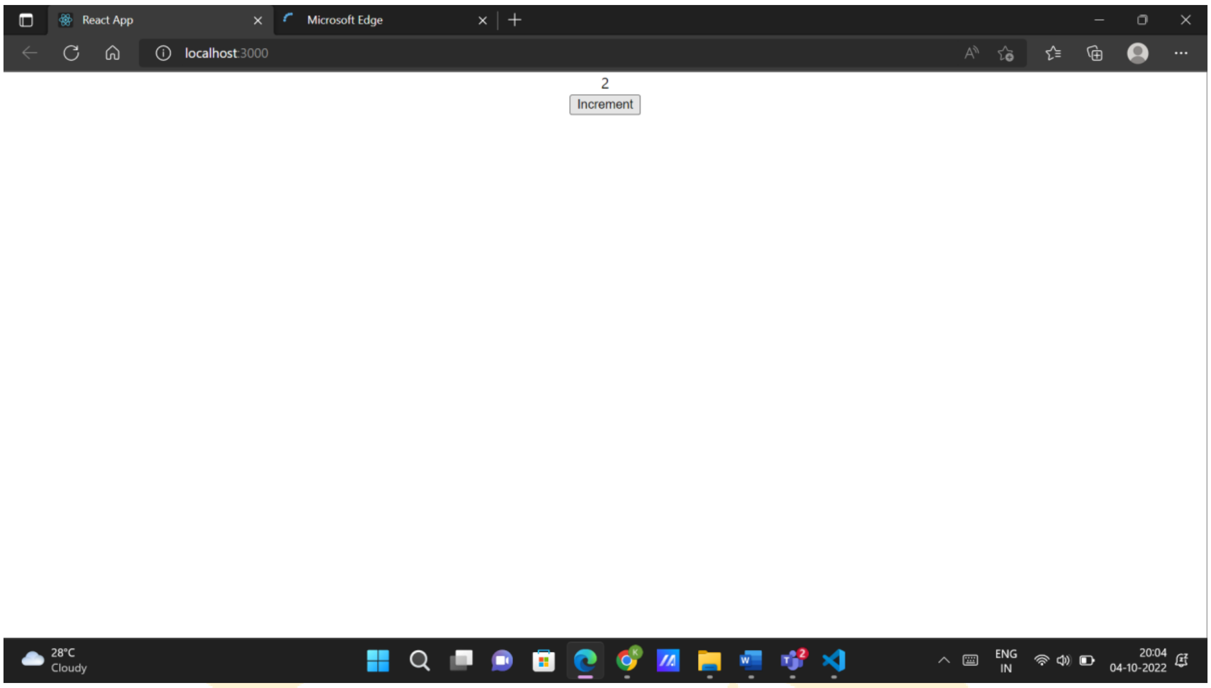


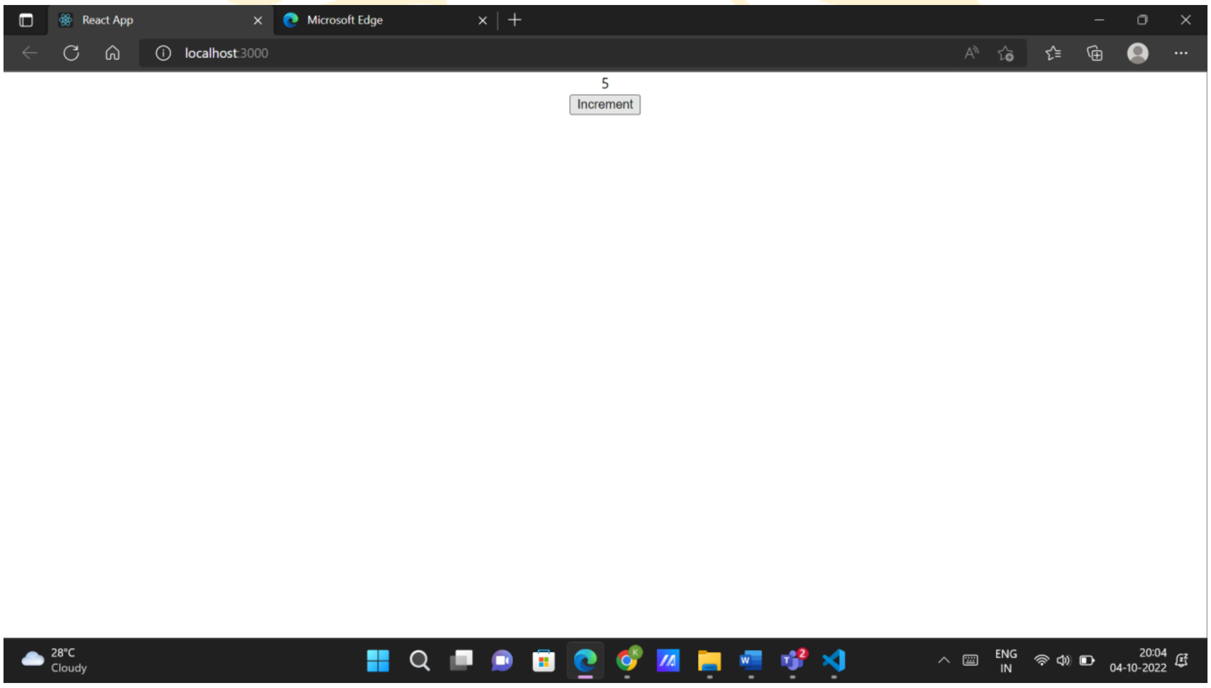
```
3
```



```
1
```

Hook Output:





Conclusion:

Thus, we have successfully able to create counter using class and Hook.