DEVOPS AND FULLSTACK
Assignment – 09

Name: Thumiki Satya sree
Ht.no: 2303A51222
Batch: 04

Scenario: Counter Application (Understanding State)

Aim

To develop a simple **Counter Application** using React in order to understand the concept of **state management** using the useState() hook.

Objective

The objective of this experiment is to understand how React manages component state and re-renders the UI when the state changes. The application allows users to **increment, decrement, and reset** a counter value.

Problem Statement

Design and implement a React-based Counter Application that:

- Uses React state
- Allows increment, decrement, and reset operations
- Prevents the counter from going below zero
- Displays a message when a certain value is reached

Software & Hardware Requirements

Software Requirements

• Node.js
• npm (Node Package Manager)
• Visual Studio Code

• React (create-react-app)
• Web Browser

Hardware Requirements

• Computer / Laptop
• Minimum 4GB RAM

Description

This experiment demonstrates a basic React application that uses the useState() hook to manage the counter value. Whenever the state changes, React automatically re-renders the component and updates the UI. This helps in understanding dynamic UI updates and state-driven rendering in React.

Procedure

1. Open the terminal and create a React project using:
   npx create-react-app counter-app
2. Navigate to the project directory:
   cd counter-app
3. Start the development server:
   npm start
4. Open the project in Visual Studio Code.
5. Create a functional component named **Counter**.
6. Initialize state using the useState() hook.
7. Add buttons for Increment, Decrement, and Reset.
8. Update the counter value using state setter functions.
9. Add logic to prevent the counter from going below zero.
10. Display a message when the counter reaches a specific value.

Algorithm

1. Start the application.
2. Initialize counter value to zero.
3. On clicking Increment, increase counter by 1.
4. On clicking Decrement, decrease counter by 1 if value is greater than zero.
5. On clicking Reset, set counter value to zero.
6. Display message when counter reaches a predefined value.

7. Stop.

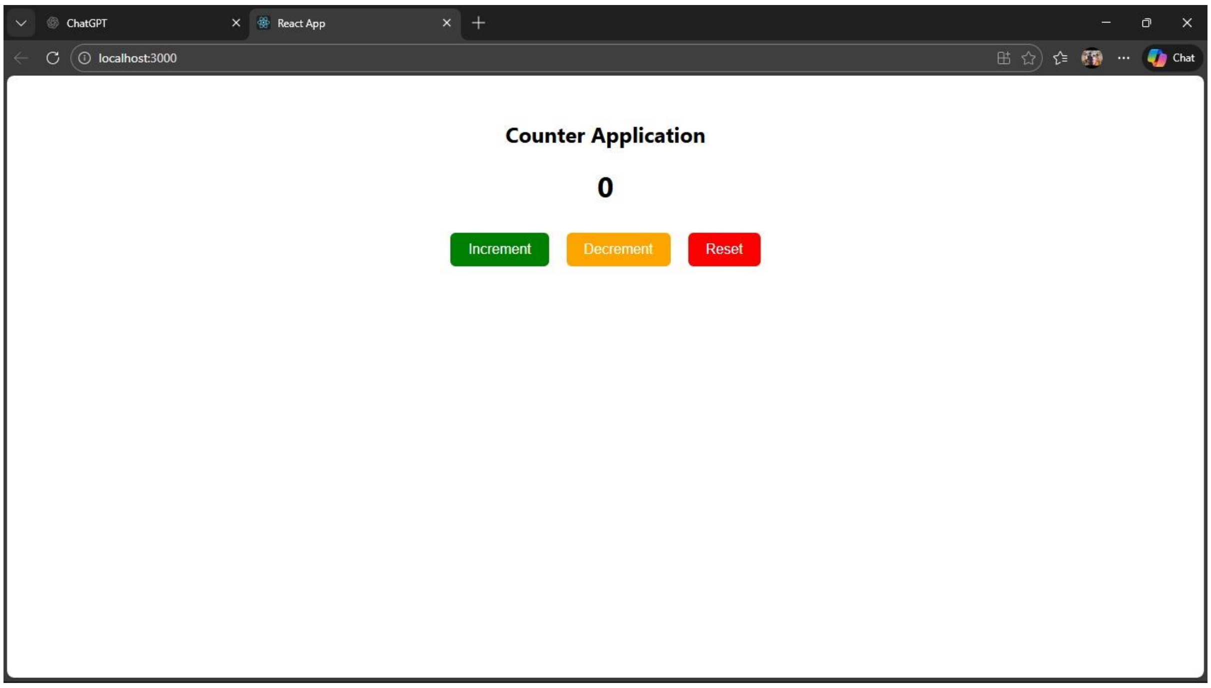
Source Code

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

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

  const increment = () => {
    setCount(count + 1);
  };

  const decrement = () => {
    if (count > 0) {
      setCount(count - 1);
    }
  };

  const reset = () => {
    setCount(0);
  };

  return (
    <div>
      <h2>Counter Value: {count}</h2>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
      {count === 10 && <p>Reached maximum value!</p>}
    </div>
  );
}
```

```
export default Counter;
```

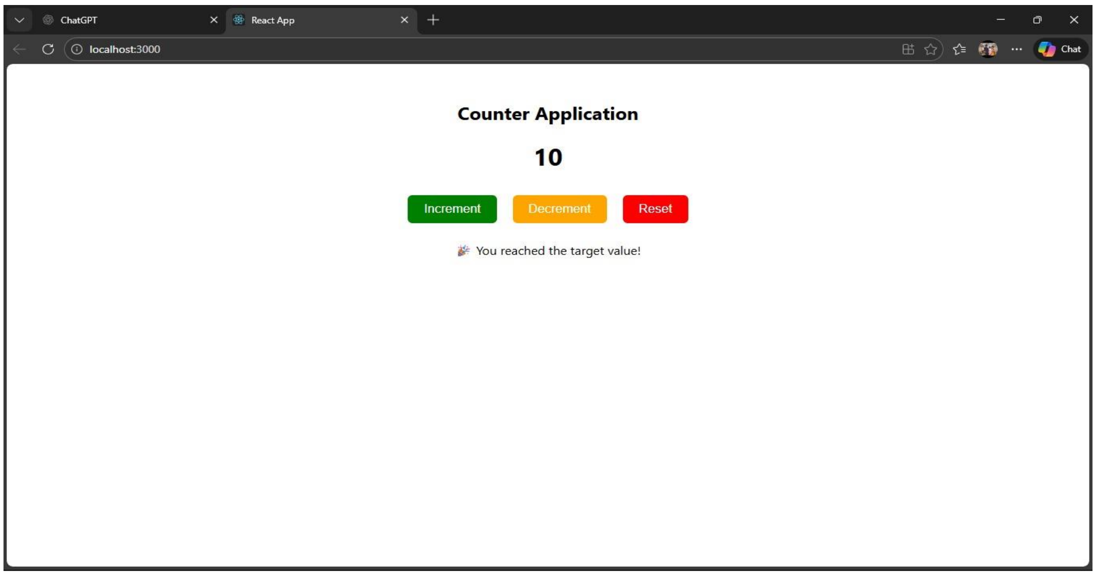
Expected Output

- Counter value updates dynamically
- State is managed using useState()
- Component re-renders on state change
- Counter does not go below zero
- Message is displayed when the counter reaches the specified value



Actual Output

The counter value increases, decreases, and resets correctly when the respective buttons are clicked. The application prevents the counter from going below zero and displays a message when the counter reaches the defined limit.



Counter Application

10

🎉 You reached the target value!

Result

Thus, the **Counter Application** was successfully developed using React. This experiment helped in understanding **state management, event handling, and re-rendering** in React using the useState() hook.

Conclusion

React state allows dynamic data handling inside components. Using useState(), UI updates automatically based on state changes, making React applications efficient and interactive.