# INTERNSHIP REPORT

# WEEK 6 DAY 3

| | | | |
|---|---|---|---|
| Submitted to: | **Ali Hyder** | Submission Date: | **29TH July 29, 2025** |
| Internship Domain: | **Front Development** | Internship Name: | **ProSensia** |
| Student Name: | **Yasal Qamar** | Roll No. | **S25031** |

# React.js Basics

# Topics: JSX, Components (Functional), Props

## Objective

To understand and implement **React component state** using the useState hook and create **interactive user interfaces** that respond to user events like button clicks.

## Topics Covered:

- Understanding the useState hook
- Managing component state
- Handling button click events
- Rendering updated state in the DOM
- Creating interactive UI (e.g., counters, toggles)

## What is React?

**React** is a **JavaScript library** used to build **user interfaces (UI)**, especially for **web applications**. It helps you create **dynamic, reusable components** that can change based on data without needing to reload the page.

## Why Use React?

- **Component-Based**: Break your UI into small, reusable pieces (like buttons, forms, headers).
- **Fast and Efficient**: Uses a virtual DOM to update only the changed parts of the UI.

- **One-Way Data Flow**: Data flows in one direction, making the app easier to debug and manage.
- **Reusable JSX Code**: Write HTML-like code inside JavaScript using JSX.

## 1. Understanding the useState Hook

- useState is a **React Hook** that lets you add **state** (data that changes over time) to a **functional component**.
- It returns an **array with two values**: the current state value, and a function to update that state.

```
const [count, setCount] = useState(0);

count is the state variable (initially 0)
```

- setCount is the function used to update the count.

## 2. Managing Component State

- State is used to store dynamic data like user input, counters, or toggles.
- When the state changes using setState (in this case setCount), React automatically **re-renders** the component to reflect the new state.

## 3. Handling Button Click Events

- Events (like clicking a button) are handled using the onClick attribute.
- You define a function (e.g., increment) and pass it to onClick.

```
<button onClick={increment}>Increment</button>
```

- When clicked, this calls the increment() function, which changes the state.

## 4. Rendering Updated State in the DOM

- When state changes, React **re-renders** the UI with the new value.
- Example: When count is updated, {count} in JSX automatically reflects the updated value on screen.

### 5. Creating Interactive UI (e.g., Counters, Toggles)

- You can build interactive components like:
- **Counters** (increment/decrement values),
- **Toggles** (show/hide content),

- **Like buttons**, **forms**, etc.
- These elements **respond to user actions** in real-time, creating a dynamic user experience.

### 6. Destructuring Props Example:

```
function Card({ title, description }) {
 return (
  <div className="card">
   <h2>{title}</h2>
   <p>{description}</p>
  </div>
 );
}
function App() {
 return (
  <div>
   <Card title="React" description="JS library for building UI" />
   <Card title="Props" description="Pass data to components" />
  </div>
 );
}
export default App;
```

## CODING:

### .app.js

```
import { useState } from "react";
import "./index.css";
```

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

  return (
    <div className="box">
      <h2>Counter App</h2>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>➕ Increment</button>
      <button onClick={() => setCount(count - 1)}>➖ Decrement</button>
    </div>
  );
}

function ToggleMessage() {
  const [show, setShow] = useState(true);

  return (
    <div className="box">
      <h2>Toggle Message</h2>
      <button onClick={() => setShow(!show)}>
        {show ? "Hide" : "Show"} Message
      </button>
      {show && <p>This is a toggleable message.</p>}
    </div>
  );
}

function ChangeColor() {
  const [color, setColor] = useState("black");

  return (
    <div className="box" style={{ color }}>
      <h2>Change Color Example</h2>
      <p>This text changes color!</p>
      <button onClick={() => setColor("red")}>🔴 Red</button>
      <button onClick={() => setColor("blue")}>🔵 Blue</button>
      <button onClick={() => setColor("green")}>🟢 Green</button>
    </div>
  );
}

export default function App() {
  return (
    <div className="container">
      <h1>useState & Event Handling Examples</h1>
```

```
      <Counter />
      <ToggleMessage />
      <ChangeColor />
    </div>
  );
}
```

**.index.css**

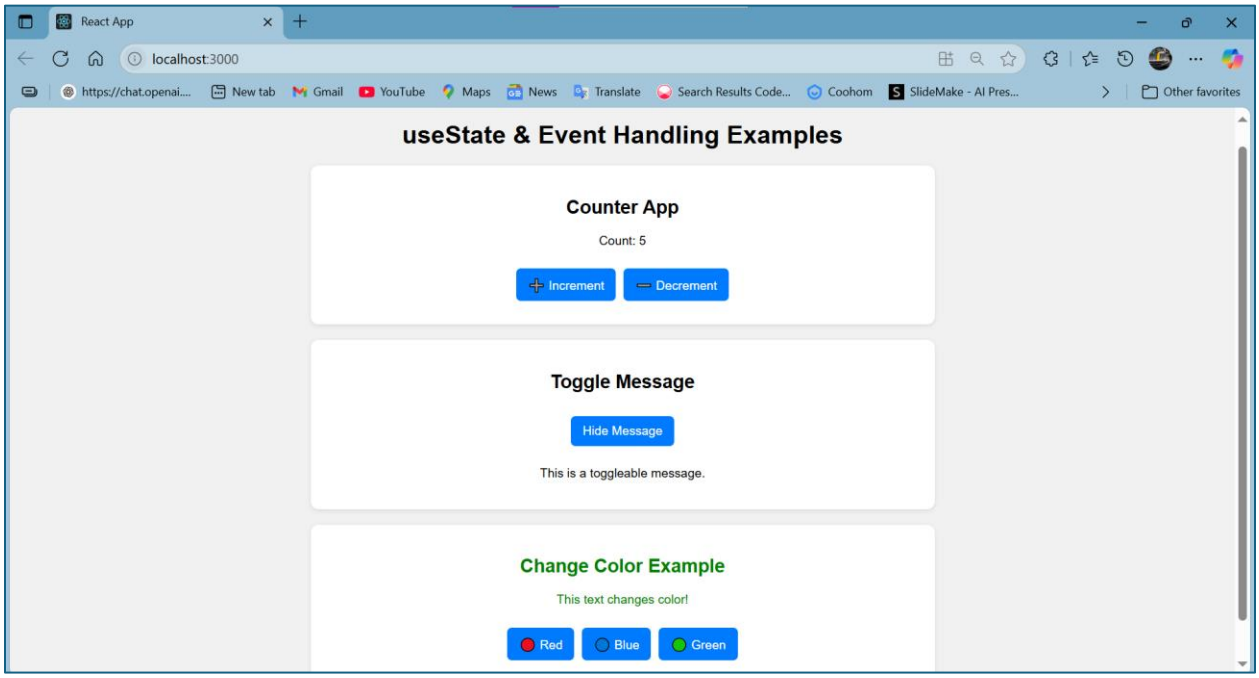
```
body {
  font-family: Arial, sans-serif;
  background-color: #f1f1f1;
  margin: 0;
  padding: 0;
}

.container {
  padding: 20px;
  max-width: 800px;
  margin: auto;
  text-align: center;
}

.box {
  background: white;
  margin: 20px 0;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 2px 6px rgba(0,0,0,0.1);
}

button {
  margin: 10px 5px;
  padding: 10px 15px;
  font-size: 16px;
  cursor: pointer;
  border-radius: 6px;
  border: none;
  background-color: #007bff;
  color: white;
  transition: 0.3s;
}
```

```
button:hover {
  background-color: #0056b3;
}
```




## CONCUSLION:

Today's session deepened my understanding of **React's interactivity features**, specifically the useState hook and event handling. I learned how to manage component state to make UIs dynamic and responsive to user actions. By building practice components like a counter, toggle message, and color changer, I experienced firsthand how state changes trigger re-rendering in React. These concepts are foundational for modern frontend development, and this practice significantly improved my confidence in creating interactive and user-friendly interfaces using functional components.