# Internship Report – Frontend Dev

# Week 6:  React.js Basics

**Name: Zainab**
**Father Name: Assad Qayyum**
**Date: 29th July, 2025**
**Internship Domain: Front-end Intern**
**Task:  State with useState, Event Handling, Conditional Rendering**

## Task Overview: (Day2)

On Week 6 Day 2 of my Front-end Web Development Internship, I learned three fundamental concepts in React.js: **State with useState, Event Handling,** and **Conditional Rendering**. These concepts are essential for making React applications interactive and dynamic.

## Content Covered:

The content covered today is:

- State with useState
- Event Handling
- Conditional Rendering

# 1. State with useState

State is a built-in way to **store and manage changing data** in a React component. For example: A counter that increases every time you click a button → the number is stored in the state.

When the state changes, react automatically re-renders the component to show the updated value.

### Why do we need useState?

In normal JavaScript, changing a variable doesn't automatically update the UI. React's useState hook solves this by:

- Storing data inside a component.
- Updating the UI automatically when that data changes.

### How to declare state variables using useState:

Syntax:

```
const [variable, setVariable] = useState(initialValue);
```

- variable → stores the current state value.
- setVariable → a function to update that value.
- initialValue → starting value of the state.

### Example:

Use the setter function like this, this tells React: "Increase the count and re-render the UI."

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

setCount(count + 1);
```

### Updating state triggers a re-render:

Every time you call setCount, React refreshes the component only where it's needed, showing the new value.

```
<button onClick={() => setCount(count + 1)}>Increase</button>
```

When clicked, count changes; page updates automatically without reloading.

## 2. Event Handling in React

Event handling allows React components to respond to user actions like clicks, typing, form submissions, etc. Just like in normal JavaScript (e.g., onclick), React lets you handle user actions like clicks, typing, mouse moves, etc.

### Common Event Types

React has many event handlers that look like HTML but use camelCase:

- onClick → For clicks.
- onChange → For inputs changing.
- onSubmit → For form submission.
- onMouseOver, onKeyDown, etc.

### How to attach event listeners to JSX elements:

In JSX, you pass a function to the event property.

```
<button onClick={handleClick}>Click Me</button>
```

### Event Handler Functions

Functions that handle what happens when an event occurs.

```
function handleClick() {
  alert("Button clicked!");
}
```

### Accessing Event Objects

Event handlers automatically receive an event object, which has info about the action.

```
function handleChange(e) {
  console.log(e.target.value); // Gets the input's value
}

<input type="text" onChange={handleChange} />
```

## 3. Conditional Rendering in React

React lets you **show or hide elements** or change the UI based on conditions. Conditional rendering is used to **display elements** or components only when certain **conditions are met**.

### a) Using if statement (outside JSX)

You can decide what to show before returning JSX.

```
function Greeting({ isLoggedIn }) {
  let message;
  if (isLoggedIn) {
    message = "Welcome Back!";
  } else {
    message = "Please Log In";
  }
  return <p>{message}</p>;
}
```

### b) Using ternary operator (condition ? true : false)

Shorter way to do inline conditionals.

```
<p>{isLoggedIn ? "Welcome Back!" : "Please Log In"}</p>
```

### c) Using logical AND (condition && element)

Renders something only if the condition is true.

```
{isLoggedIn && <p>You are logged in!</p>}
```

If isLoggedIn is false → nothing is shown.

## Practice Code:

App.js

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

function LoginApp() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [username, setUsername] = useState("");

  function handleLoginLogout() {
    setIsLoggedIn(!isLoggedIn);
  }

  function handleInputChange(e) {
    setUsername(e.target.value);
  }

  const containerStyle = {
    textAlign: "center",
    marginTop: "50px",
    padding: "20px",
    width: "300px",
    margin: "50px auto",
    border: "1px solid #ccc",
    borderRadius: "10px",
    backgroundColor: "#f9f9f9",
    boxShadow: "0 2px 6px rgba(0, 0, 0, 0.1)",
  };

  return (
    <div style={containerStyle}>
      <h1>React Login App</h1>
```

App.js

```
function LoginApp() {
  return (
    <div style={containerStyle}>
      <h1>React Login App</h1>

      <h2>
        {isLoggedIn ? `Welcome, ${username.trim() || "User"}!` : "Please Log In"}
      </h2>

      {!isLoggedIn && (
        <input
          type="text"
          placeholder="Enter your name"
          value={username}
          onChange={handleInputChange}
          style={{
            padding: "8px",
            marginBottom: "12px",
            width: "90%",
            borderRadius: "5px",
            border: "1px solid #ccc",
          }}
        />
      )}
      <br />

      <button
        onClick={handleLoginLogout}
        disabled={!isLoggedIn && username.trim() === ""}
        style={{
```

App.js

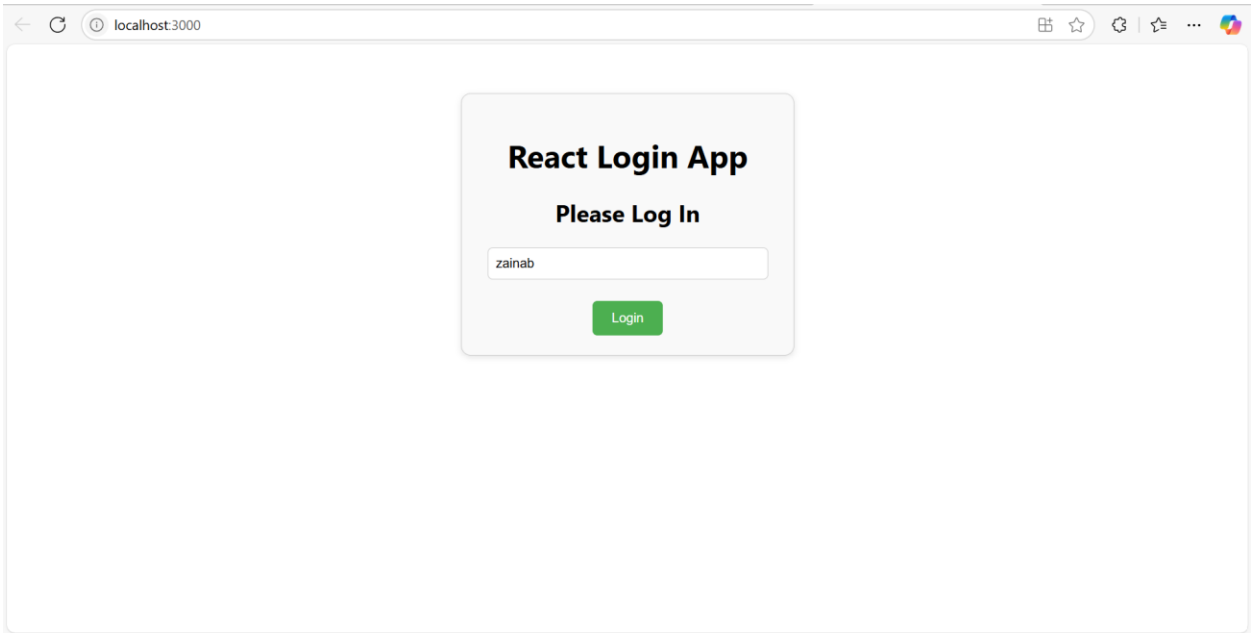
src > App.js > LoginApp

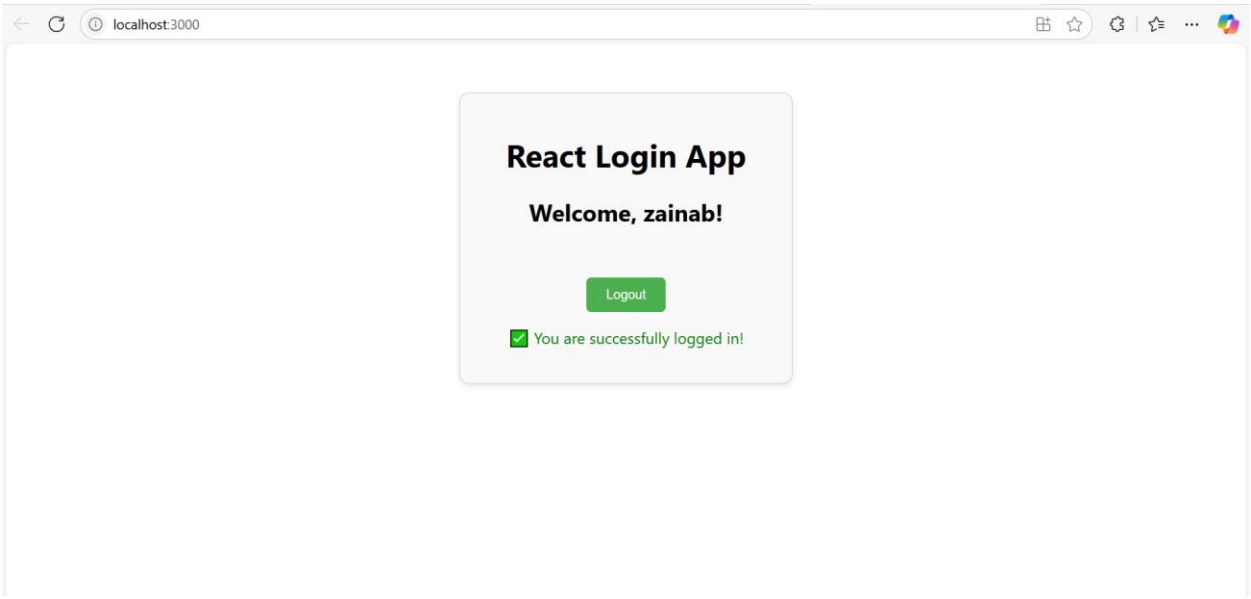
```
function LoginApp() {
        <button
          onClick={handleLoginLogout}
          disabled={!isLoggedIn && username.trim() === ""}
          style={{
            padding: "10px 20px",
            marginTop: "10px",
            backgroundColor: "#4CAF50",
            color: "white",
            border: "none",
            borderRadius: "5px",
            cursor: "pointer",
          }}
        >
          {isLoggedIn ? "Logout" : "Login"}
        </button>

        {isLoggedIn && (
          <p style={{ color: "green", marginTop: "15px" }}>
            ✅ You are successfully logged in!
          </p>
        )}
      </div>
    );
}

export default LoginApp;
```

Ln 64, Col 8 Spaces: 2 UTF-8 LF {} JavaScript Port : 5500






**Conclusion:**

In today's task, I learned how to use useState to manage component data, handle user events like clicks and input changes, and conditionally render elements based on state. These three concepts form the foundation for building **interactive and dynamic** React applications.