Training Day 2 Report

Date: 26 June 2025

Topic Covered: State, Events, and Lifecycle in React

On Day 2 of MERN Stack training, I explored key React concepts such as **state management**, **event handling**, and a basic introduction to the **React lifecycle**.

These concepts are essential for creating interactive components and managing dynamic data within a React application.

React State

React components can hold local state using the `useState()` hook.

State allows components to reactively re-render when data changes.

Example:

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

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click Me</button>
    </div>
  );
}
```

Event Handling in React

Event handling in React uses camelCase syntax and functions are passed as values instead of using inline HTML strings.

React automatically passes an event object.

Example:

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

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

State vs Props

- **Props** are passed from parent to child and are read-only.
- **State** is managed within the component and can be updated.
- Changing state triggers re-rendering, while props reflect external data.

Component Lifecycle (Basics)

Though functional components don't use traditional lifecycle methods, we use the `useEffect()` hook to simulate behavior like `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.

Example:

```
useEffect(() => {
  console.log("Component mounted");
  return () => {
    console.log("Component unmounted");
  };
}, []);
```

Outcome of the Day

- Learned to manage internal component state using `useState()`.
- Implemented click event handlers and understood React's event system.
- Understood how `useEffect()` replaces lifecycle methods in functional components.
- Practiced dynamic UI updates and rendering based on state changes.