

## Day 4 – Introduction to ReactJS (JSX, Components, Props, and State) Date: 26-10-2025

**Objective:** Understand the fundamentals of ReactJS, how it differs from traditional JavaScript, and how to build reusable UI components using JSX syntax, props, and state management.

### 1. WHAT IS ReactJS

ReactJS is an **open-source JavaScript library** developed by Facebook (now Meta) for building dynamic and responsive user interfaces, especially single-page applications (SPAs).

#### Key Features:

- **Component-Based:** UI is built using independent reusable pieces called components.
- **Declarative:** Focus on “what to render” rather than “how”.
- **Virtual DOM:** React updates only the changed parts of the UI for better performance.
- **One-Way Data Binding:** Data flows from parent to child components.
- **JSX:** A syntax extension that allows writing HTML-like code inside JavaScript.

### 2. SETTING UP REACT

Before working with React, ensure you have **Node.js** and **npm** installed.

#### Create a new React app

```
npx create-react-app myapp
```

```
cd myapp
```

```
npm start
```

This will start a local development server and open your app at <http://localhost:3000/>.

### 3. UNDERSTANDING JSX (JavaScript XML)

JSX allows you to write HTML elements inside JavaScript.

React then converts this JSX into JavaScript using a compiler like Babel.

#### Example (JSX vs JS):

Without JSX:

```
const element = React.createElement('h1', null, 'Hello React');
```

With JSX:

```
const element = <h1>Hello React</h1>;
```

## JSX Rules:

- You can only return **one root element**.
- Use **className** instead of class.
- Expressions can be embedded using {}.

Example:

```
const name = "Udaya";  
const element = <h2>Hello, {name}! Welcome to React.</h2>;
```

## 4. Components in React

A **component** is a reusable block of code that controls a part of the UI.

### Types of Components

1. **Functional Components** – simple functions returning JSX.
2. **Class Components** – ES6 classes extending React.Component.

### Functional Component Example

```
function Welcome() {  
  return <h2>Welcome to Dhruv Compusoft Training</h2>;  
}  
export default Welcome;
```

### Rendering the Component

In App.js:

```
import Welcome from './Welcome';
```

```
function App() {  
  return (  
    <div>  
      <Welcome />  
    </div>  
  );  
}
```

```
}
```

**Output:**

Welcome to Dhruv Compusoft Training

**Class Component Example**

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

class WelcomeClass extends Component {

  render() {

    return <h2>Hello from Class Component</h2>;

  }

}

export default WelcomeClass;
```

**5. Props (Passing Data Between Components)**

**Props** (short for *properties*) are used to pass data from parent to child components.

**Parent Component:**

```
function App() {

  return <Student name="Udaya" course=".NET" />;

}
```

**Child Component (Student.js):**

```
function Student(props) {

  return (

    <div>

      <h3>Student Name: {props.name}</h3>

      <p>Course: {props.course}</p>

    </div>

  );

}

export default Student;
```

**Output:**

Student Name: Udaya

Course: .NET

**Props are immutable**, meaning you cannot modify them inside the child component.

**6. State (Component Data Management)**

**State** is used to store dynamic, changeable data in a component.

When state changes, the component re-renders automatically.

**Using useState Hook (in Functional Components)**

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

```
function Counter() {  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
      <h3>Count: {count}</h3>  
      <button onClick={() => setCount(count + 1)}>Increase</button>  
      <button onClick={() => setCount(count - 1)}>Decrease</button>  
    </div>  
  );  
}  
  
export default Counter;
```

**Explanation:**

- `useState(0)` initializes state variable `count` with value 0.
- `setCount` is a function to update `count`.
- Clicking buttons updates state and re-renders UI.

## 7. COMBINING MULTIPLE COMPONENTS

```
function App() {  
  return (  
    <div>  
      <Welcome />  
      <Student name="Udaya" course=".NET Full Stack" />  
      <Counter />  
    </div>  
  );  
}
```

Each component handles a small part of the app, making code modular and easier to maintain.

## 8. COMPONENT HIERARCHY

Example:

App.js

└─ Header.js

└─ Student.js

└─ Counter.js

This structure ensures a **clean, scalable** React application.

## 9. BEST PRACTICES

- Always capitalize component names (e.g., Student, not student).
- Keep each component in its own file.
- Use props for data passing and useState for internal state.
- Keep components small and focused on one task.
- Use meaningful variable and function names.

### Mini Practice Task (Day 4)

**Objective:** Create a simple React app with three components:

1. Header.js – Displays page title.
2. Student.js – Displays student info using props.
3. Counter.js – Demonstrates use of useState.

**App Structure:**

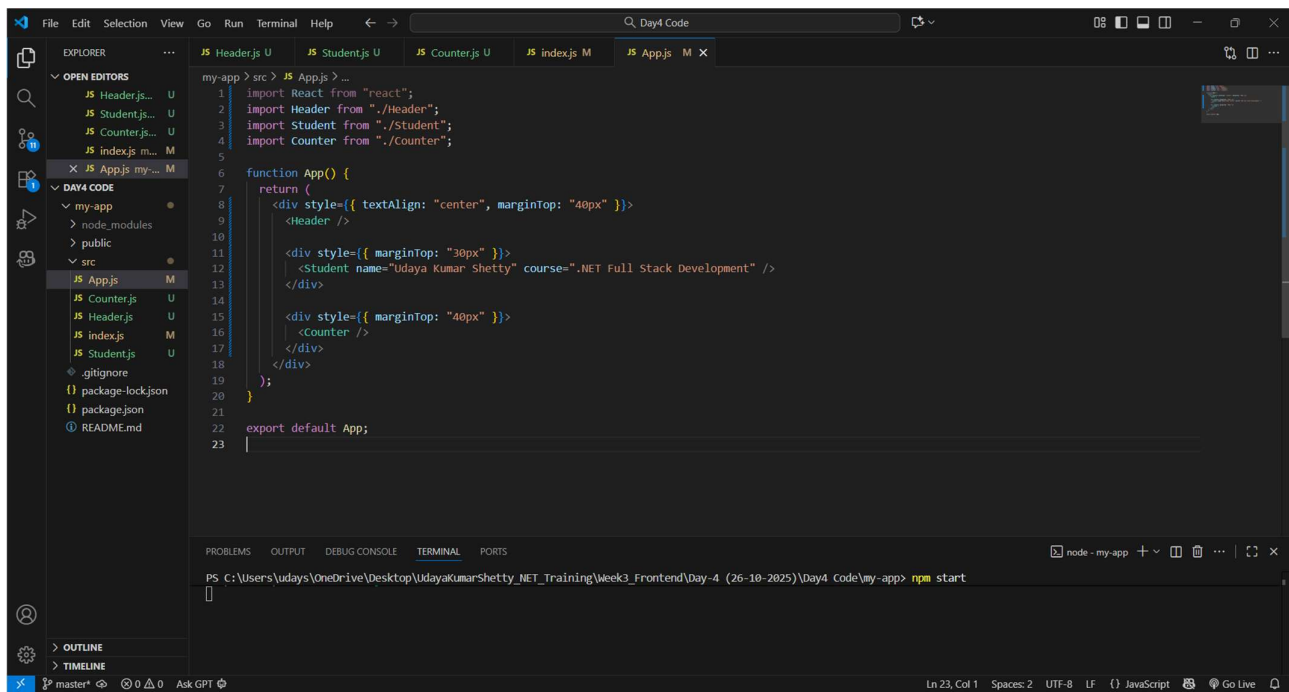
/src

```
|— App.js
|— Header.js
|— Student.js
└— Counter.js
```

**Task:**

- Display “Dhruv Compusoft Training Portal” as heading.
- Pass props from App to Student (name, course).
- Create a counter with increment and decrement buttons.

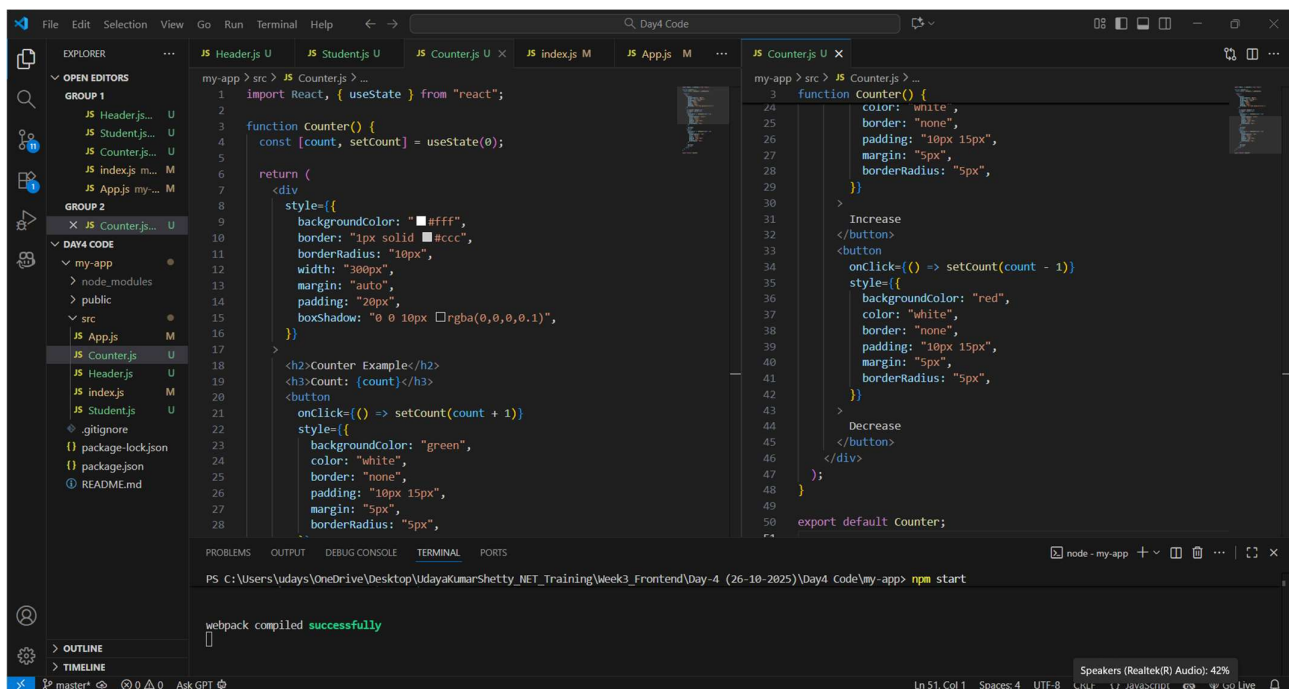
## Snapshots :



```
my-app > src > JS Appjs > ...
1 import React from "react";
2 import Header from "../Header";
3 import Student from "../Student";
4 import Counter from "../Counter";
5
6 function App() {
7   return (
8     <div style={{ textAlign: "center", marginTop: "40px" }}>
9       <Header />
10
11       <div style={{ marginTop: "30px" }}>
12         <Student name="Udaya Kumar Shetty" course=".NET Full Stack Development" />
13       </div>
14
15       <div style={{ marginTop: "40px" }}>
16         <Counter />
17       </div>
18     </div>
19   );
20 }
21
22 export default App;
```

PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty\_NET\_Training\Week3\_Frontend\Day-4 (26-10-2025)\Day4 Code\my-app> npm start

## Code: App.js

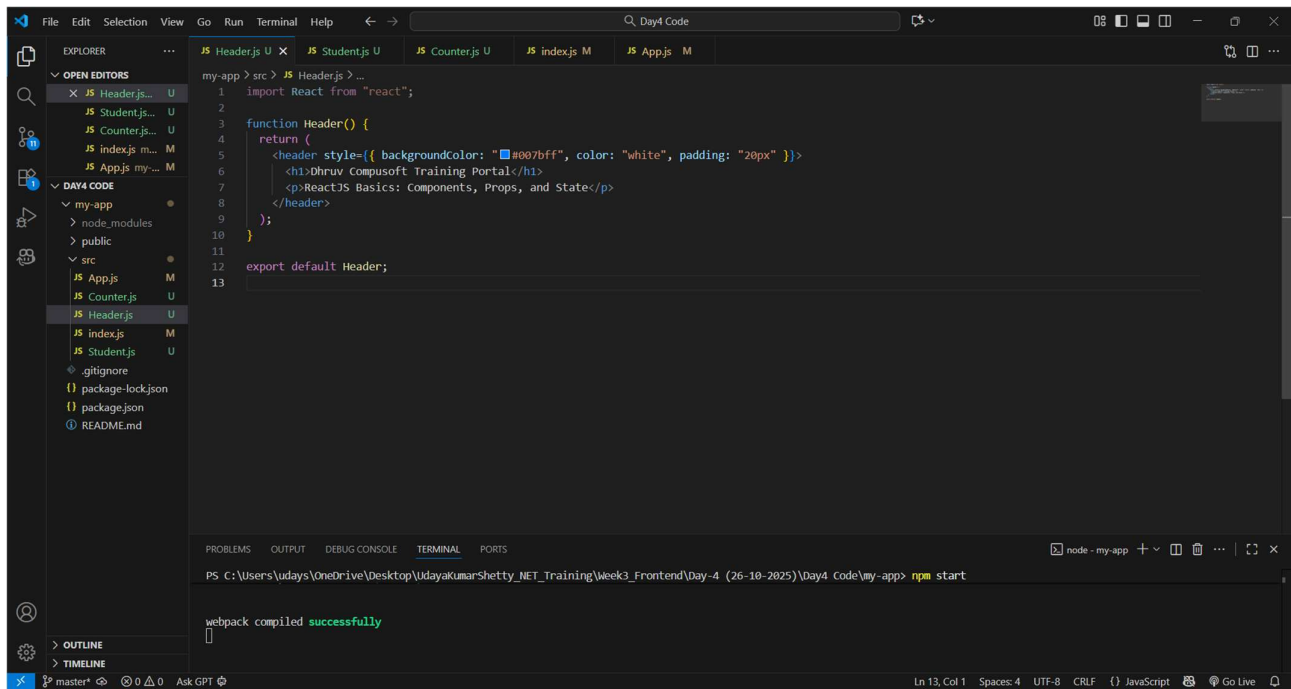


```
my-app > src > JS Counterjs > ...
1 import React, { useState } from "react";
2
3 function Counter() {
4   const [count, setCount] = useState(0);
5
6   return (
7     <div
8       style={{
9         backgroundColor: "#ffff",
10         border: "1px solid #ccc",
11         borderRadius: "10px",
12         width: "300px",
13         margin: "auto",
14         padding: "20px",
15         boxShadow: "0 0 10px rgba(0,0,0,0.1)",
16       }}
17     >
18       <h2>Counter Example</h2>
19       <h3>Count: {count}</h3>
20       <button
21         onClick={() => setCount(count + 1)}
22         style={{
23           backgroundColor: "green",
24           color: "white",
25           border: "none",
26           padding: "10px 15px",
27           margin: "5px",
28           borderRadius: "5px",
29         }}
30       >
31         Increase
32       </button>
33       <button
34         onClick={() => setCount(count - 1)}
35         style={{
36           backgroundColor: "red",
37           color: "white",
38           border: "none",
39           padding: "10px 15px",
40           margin: "5px",
41           borderRadius: "5px",
42         }}
43       >
44         Decrease
45       </button>
46     </div>
47   );
48 }
49
50 export default Counter;
```

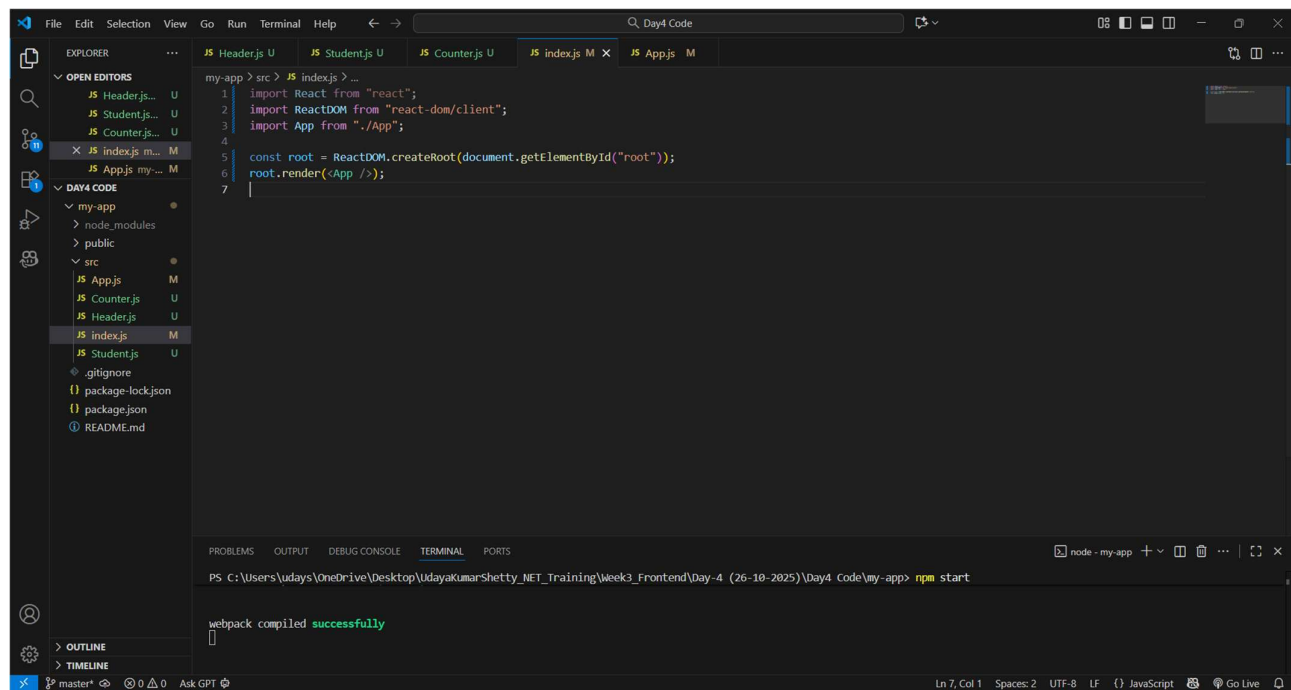
PS C:\Users\udays\OneDrive\Desktop\UdayakumarShetty\_NET\_Training\Week3\_Frontend\Day-4 (26-10-2025)\Day4 Code\my-app> npm start

webpack compiled successfully

## Code: Counter.js

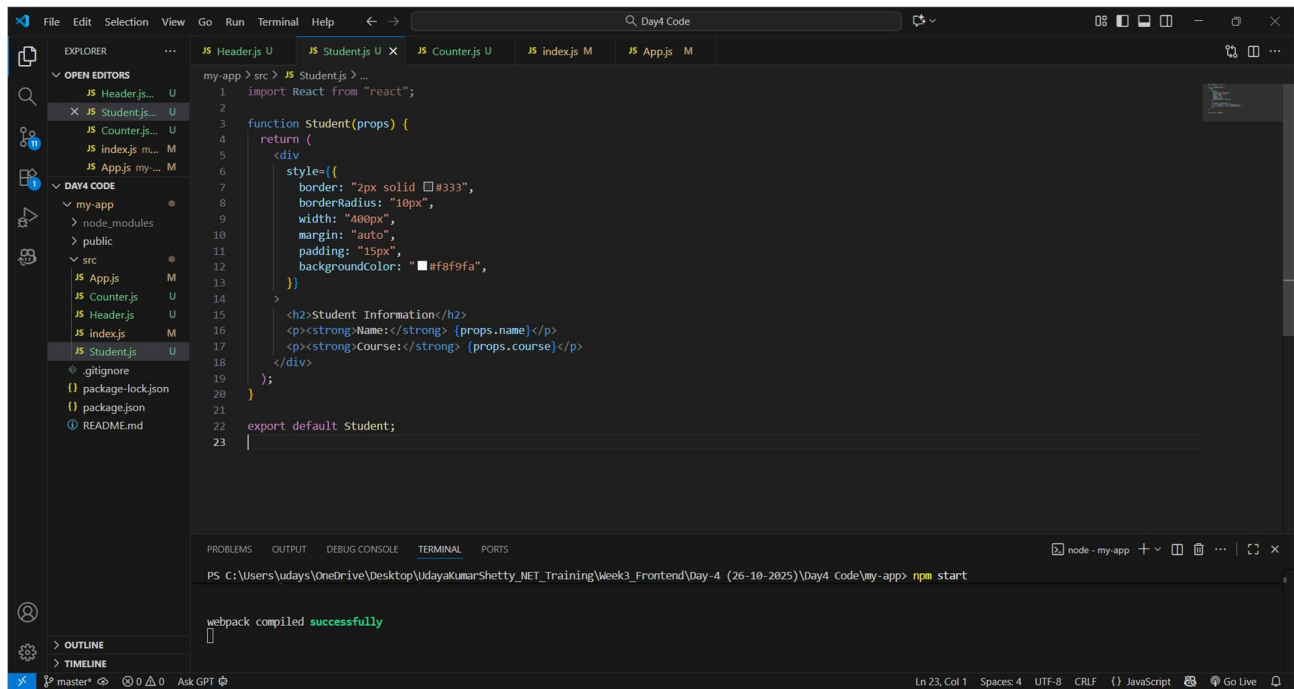


Code: Header.js

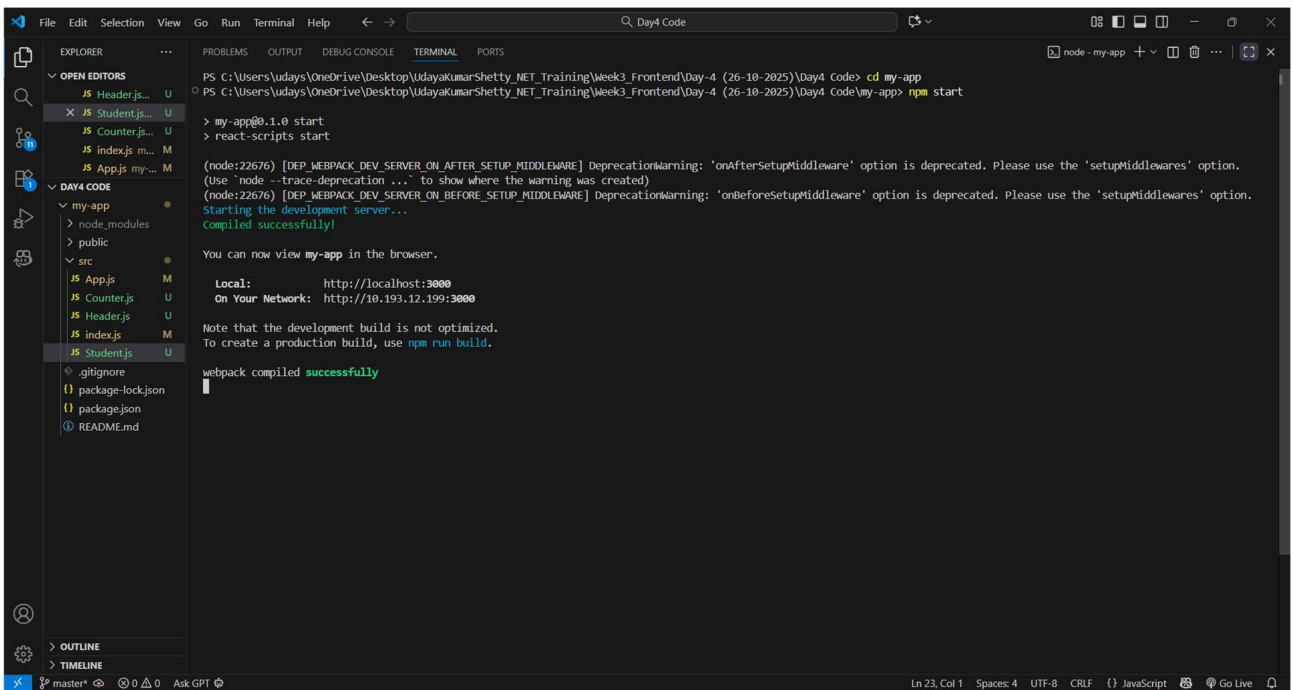


Code: index.js

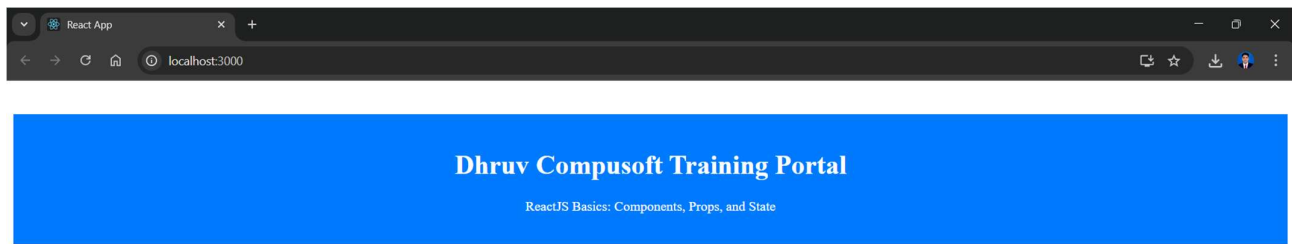




Code: Student.js



Terminal Commands



**Student Information**

**Name:** Udaya Kumar Shetty

**Course:** .NET Full Stack Development

**Counter Example**

**Count:** 4

Increase Decrease

Output: Counter Increment



**Student Information**

**Name:** Udaya Kumar Shetty

**Course:** .NET Full Stack Development

**Counter Example**

**Count:** 1

Increase Decrease

Output: Counter Decrement