**ReactJS-HOL**

**1.**

**List the features of ES6**

**Solution:-**

* let and const for variable declaration
* Arrow functions (=>)
* Classes and inheritance
* Template literals
* Destructuring
* Default parameters
* Spread (...) and rest operators
* Map and Set
* Promises
* Modules (import/export)

**Explain JavaScript let**

**Solution:-**

* let is used to declare **block-scoped** variables.
* Unlike var, it **doesn't hoist** the same way.
* It prevents redeclaring the same variable in the same scope.

**Identify the differences between var and let**

**Solution:-**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped {} |
| Redeclaration | Allowed | Not allowed |
| Hoisting | Yes (initialized as undefined) | Yes, but not accessible before declaration |
| Usage | Older JavaScript | ES6 (modern JS) |

**Explain JavaScript const**

**Solution:-**

const is used to declare **constants** (cannot be reassigned).

It's also **block-scoped**.

However, if the constant is an **object or array**, its **contents** can still be modified.

**Explain ES6 class fundamentals**

**Solution:-** ES6 introduced a cleaner syntax for **creating classes**.

js

CopyEdit

class Person {

constructor(name) {

this.name = name;

}

greet() {

return `Hello, ${this.name}`;

}

}

const p = new Person("Shalagya");console.log(p.greet());

**Explain ES6 class inheritance**

**Solution:-** create a subclass using extends and super().

js

CopyEdit

class Student extends Person {

constructor(name, roll) {

super(name); // call parent constructor

this.roll = roll;

}

details() {

return `${this.name} has roll number ${this.roll}`;

}

}

**Define ES6 arrow functions**

**Solution:-** Arrow functions provide a shorter syntax and bind this lexically.

**Identify set(), map()**

#### **Solution:- set()**

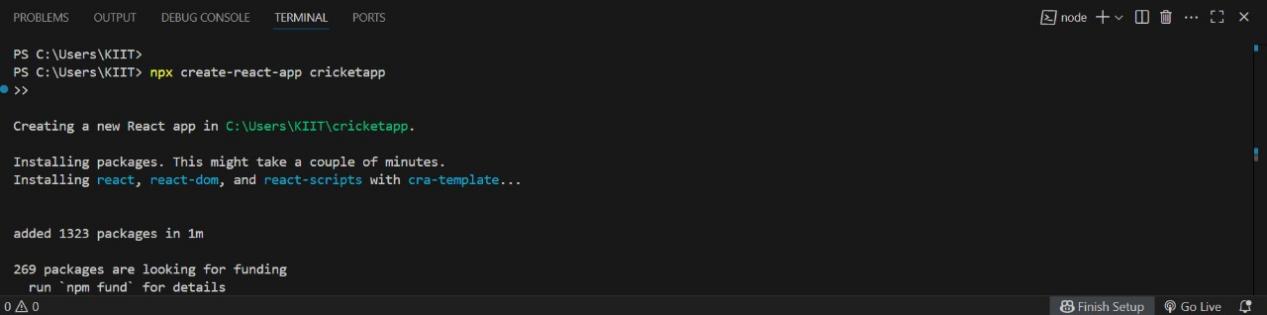
#### Stores ****unique values****

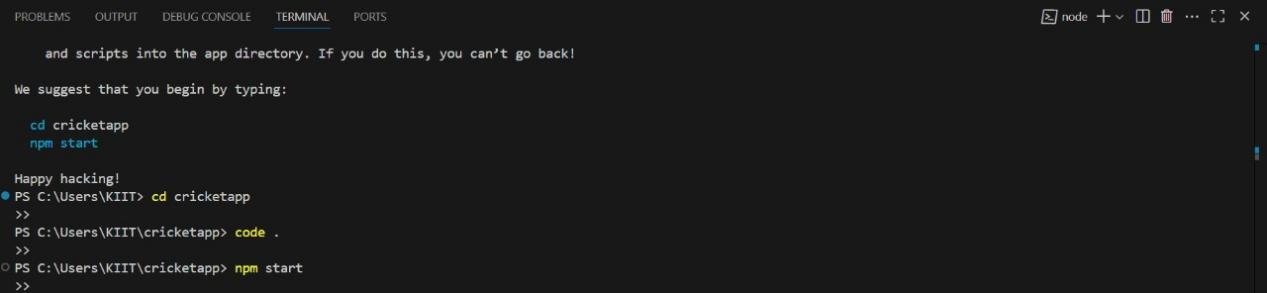
No duplicates allowed

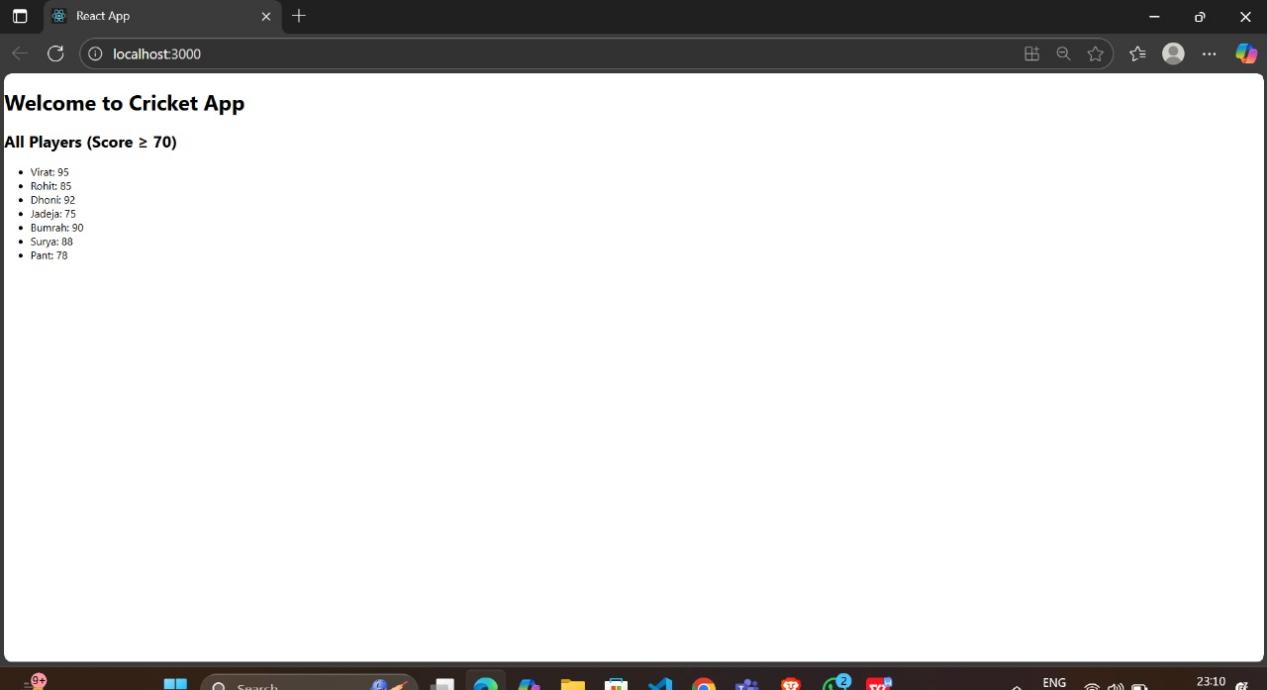
#### Map()

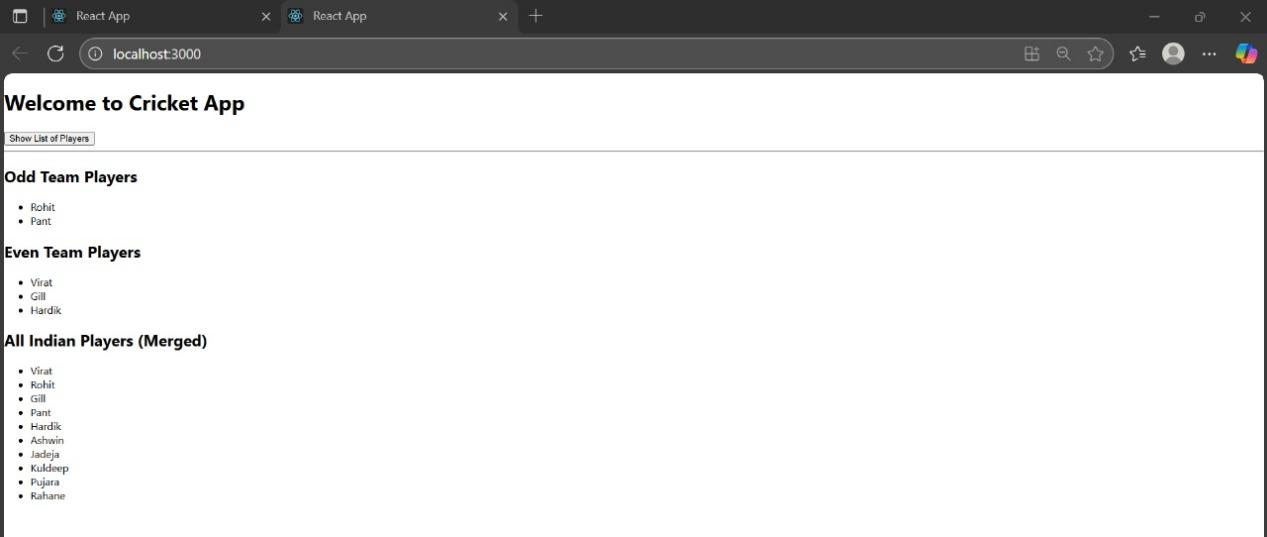
Stores **key-value pairs**

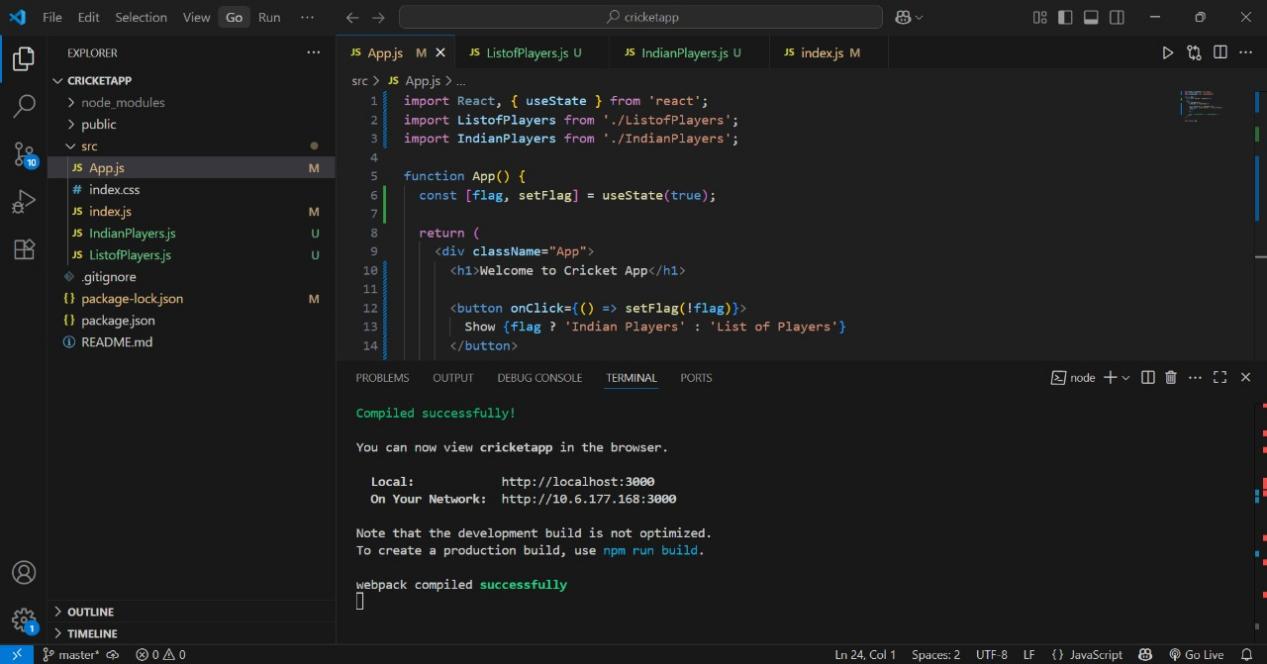
Keys can be of **any type**

****

****

****

****

****

**2.**

**Define JSX**

**Solution:-**

**JSX (JavaScript XML)** is a syntax extension for JavaScript used in React. It allows writing HTML-like code directly within JavaScript.

JSX makes it easier to:

Create React elements visually similar to HTML

Combine layout and logic in the same place

**Explain about ECMA Script**

**Solution:-**

**ECMAScript** is the official scripting language specification that JavaScript follows.

**ES6** (ECMAScript 2015) introduced major features like:

* let, const
* Arrow functions
* Classes and inheritance
* Template literals
* Destructuring
* Modules (import/export)
* Set, Map

React applications rely heavily on ES6+ syntax.

**Explain React.createElement()**

**Solution:-** This is the **core function** React uses to create **React node.**

**Explain how to create React nodes with JSX**

**Solution:-** In React, **JSX is used to create UI elements (nodes)** like:

jsx

CopyEdit

const heading = <h1>Welcome</h1>;const img = <img src="logo.png" />;const list = <ul><li>Item</li></ul>;

These nodes behave like regular JavaScript objects (called **React elements**) and describe what the UI should look like.

**Define how to render JSX to DOM**

**Solution:-** React uses ReactDOM.render() (or createRoot().render() in React 18+) to **render JSX** to the actual browser DOM.

**Explain how to use JavaScript expressions in JSX**

**Solution:-** You can embed JavaScript inside JSX using **curly braces** {}.

Example:

const name = "Shalagya";<h1>Hello, {name}!</h1>

You can use:

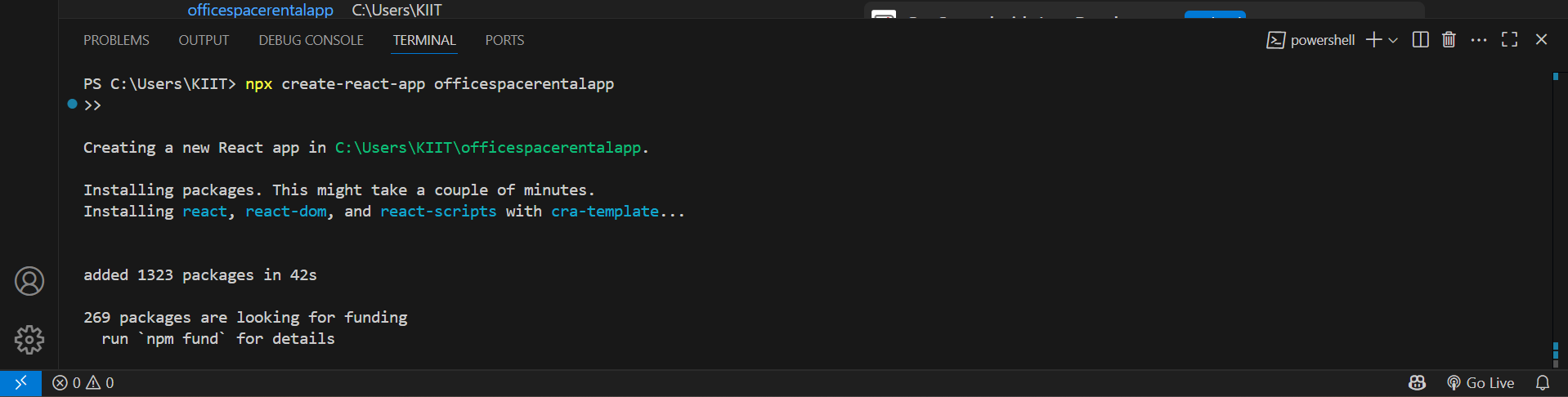
* Variables
* Functions
* Arithmetic
* Conditional expressions

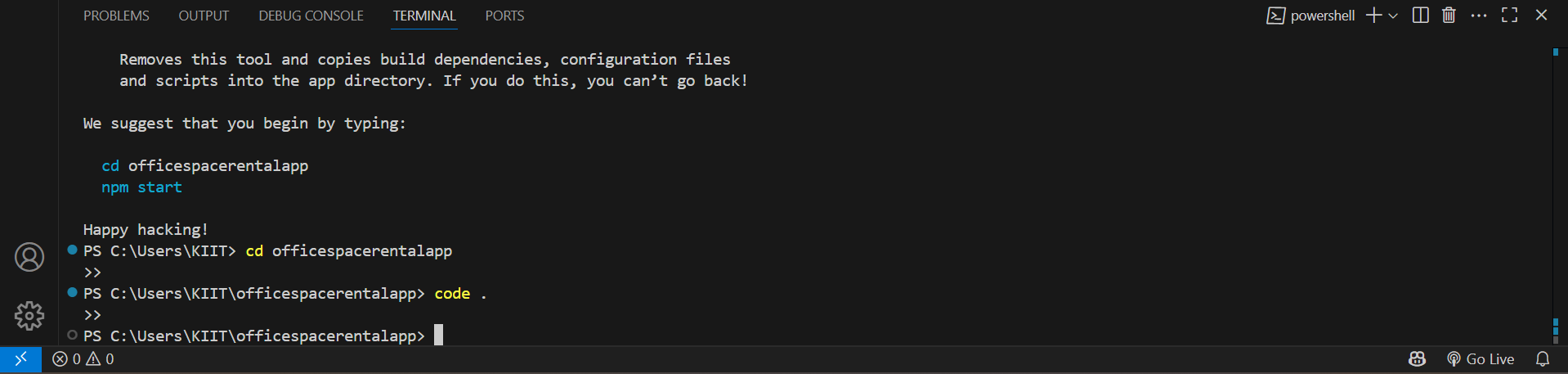
Not allowed: if statements, for loops (use ternary or .map() instead)

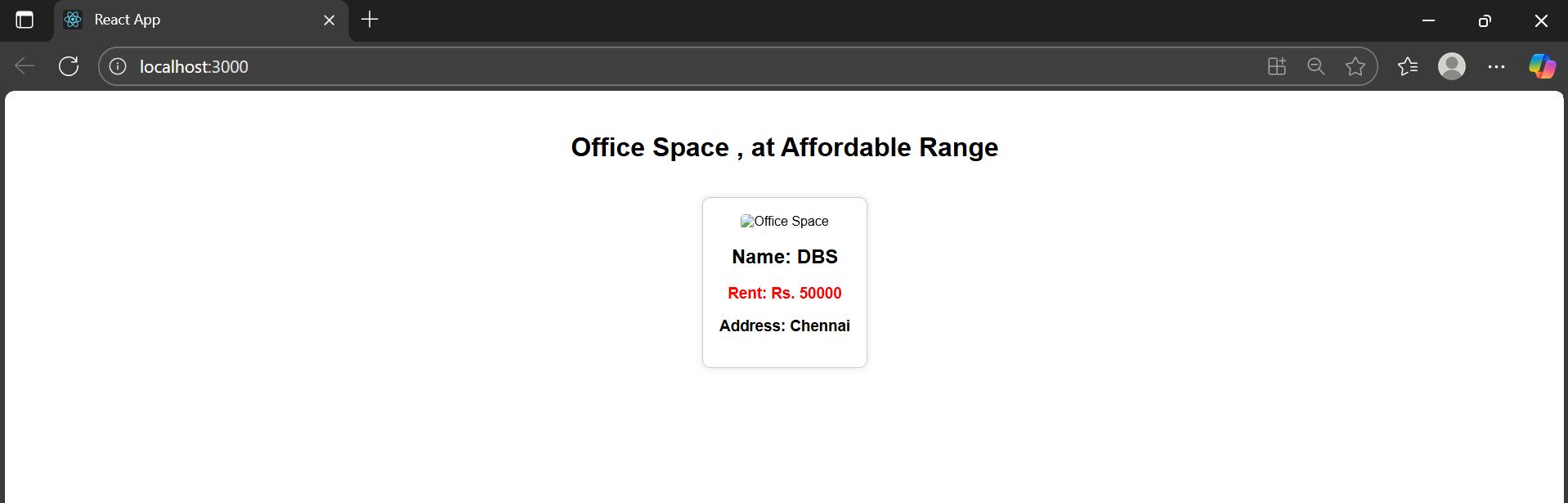
**Explain how to use inline CSS in JSX**

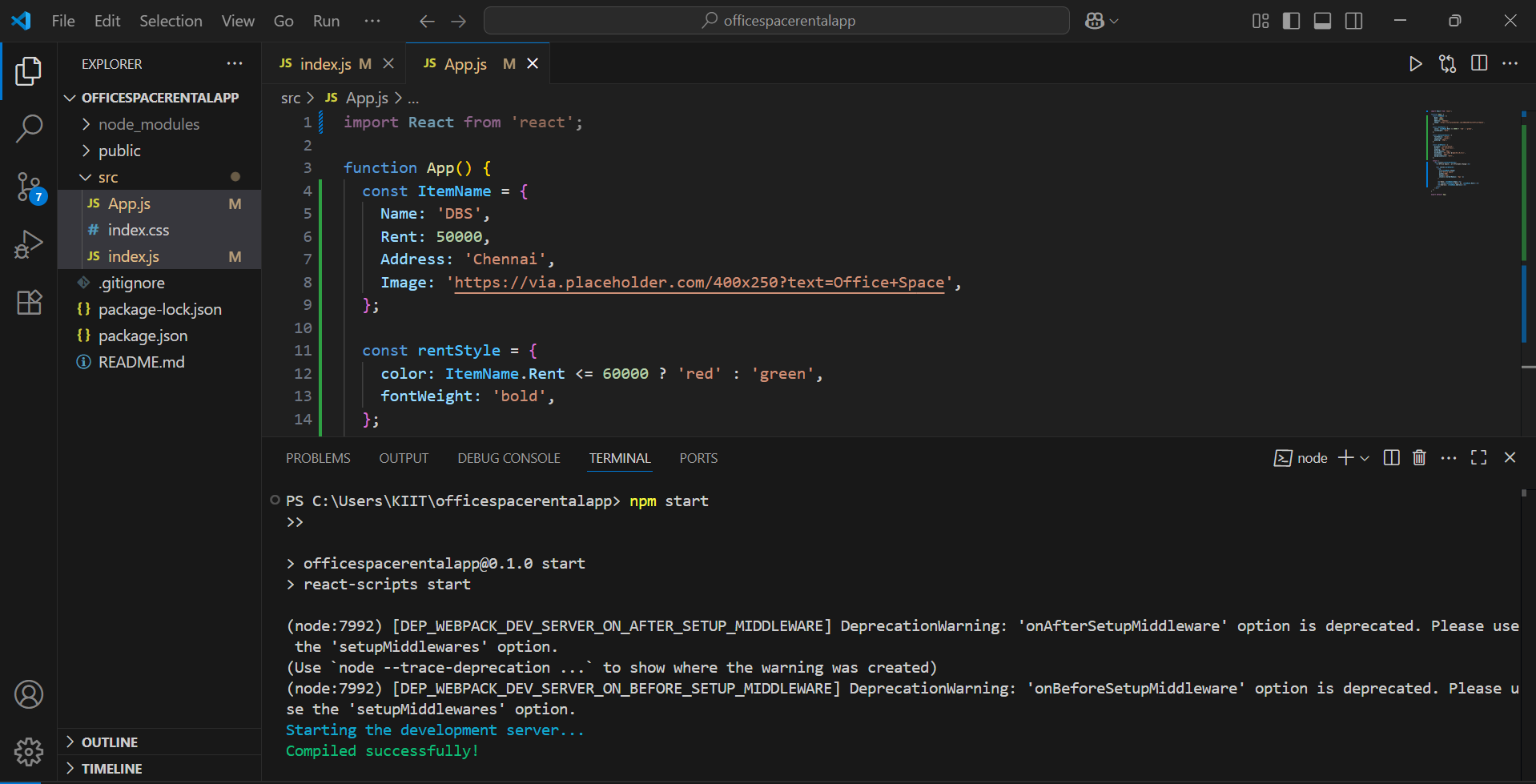
**Solution:-** You can apply styles directly inside JSX using the style attribute. It takes a **JavaScript object**, not a string.

* Property names use **camelCase** (e.g. fontSize, backgroundColor)
* Values are written as strings or expressions









**3.**

**Explain React events**

**Solution:-** eact **events** are actions or occurrences (like clicks, typing, form submission, etc.) that happen in the browser and can be handled by JavaScript.

React uses a **wrapper around the browser's native events** to make them consistent across different browsers.

**Explain about event handlers**

**Solution:-** An **event handler** is a function that’s called in response to an event (like onClick, onChange, etc.).

In React:

You pass the **function reference** or an **arrow function** inside JSX.

React will call the handler when the event occurs.

**Define Synthetic event**

**Solution:-** A **SyntheticEvent** is a React wrapper around the browser's native event.

It ensures **cross-browser compatibility**

It has the same interface as native events (e.g., event.target, event.preventDefault())

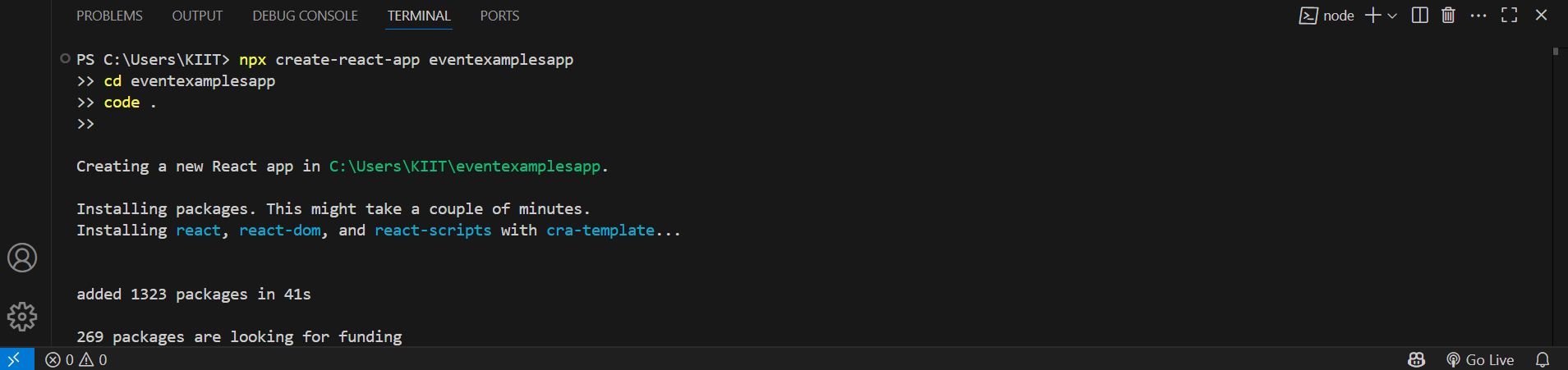
**Identify React event naming convention**

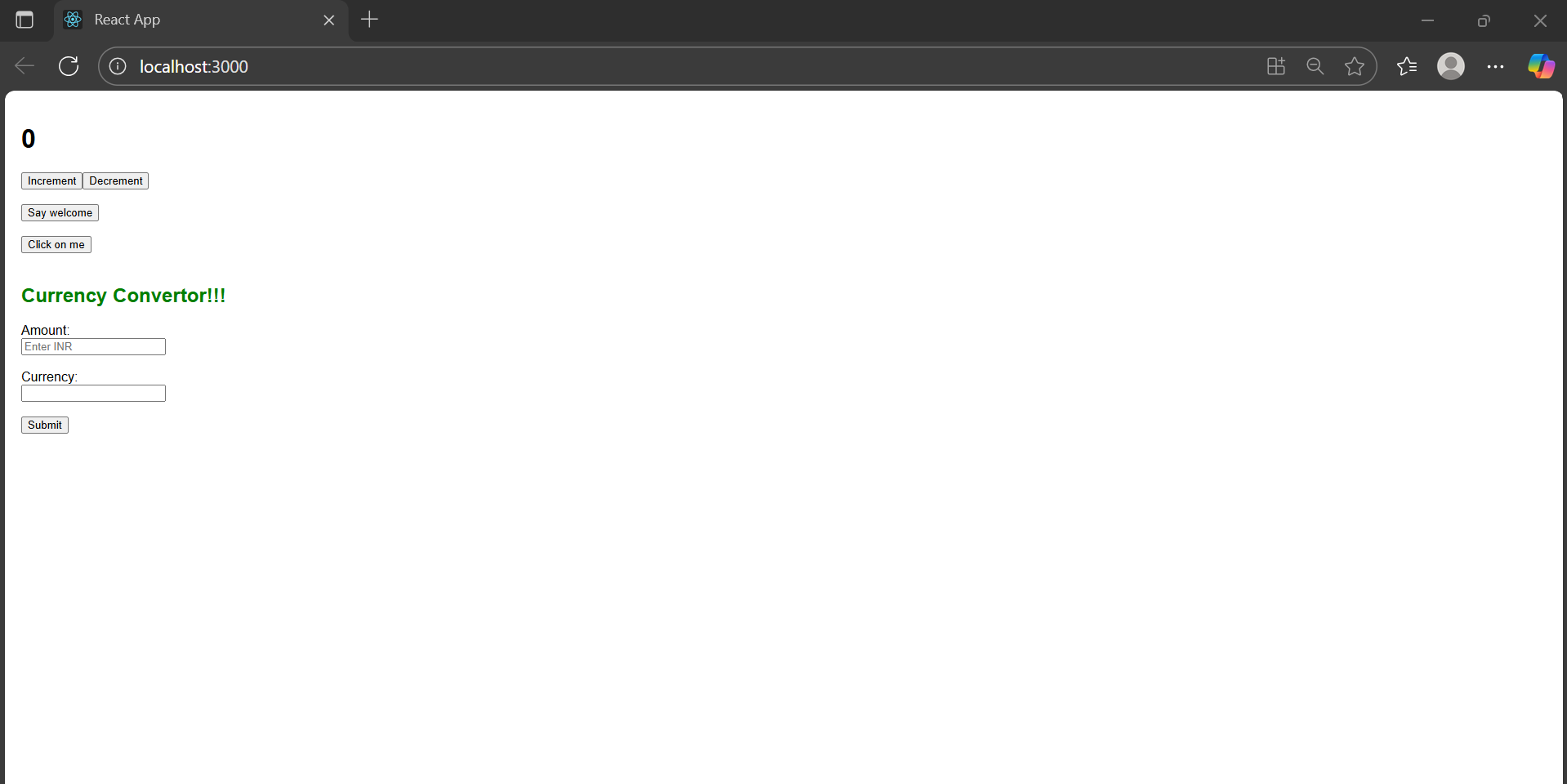
### **Solution:-**

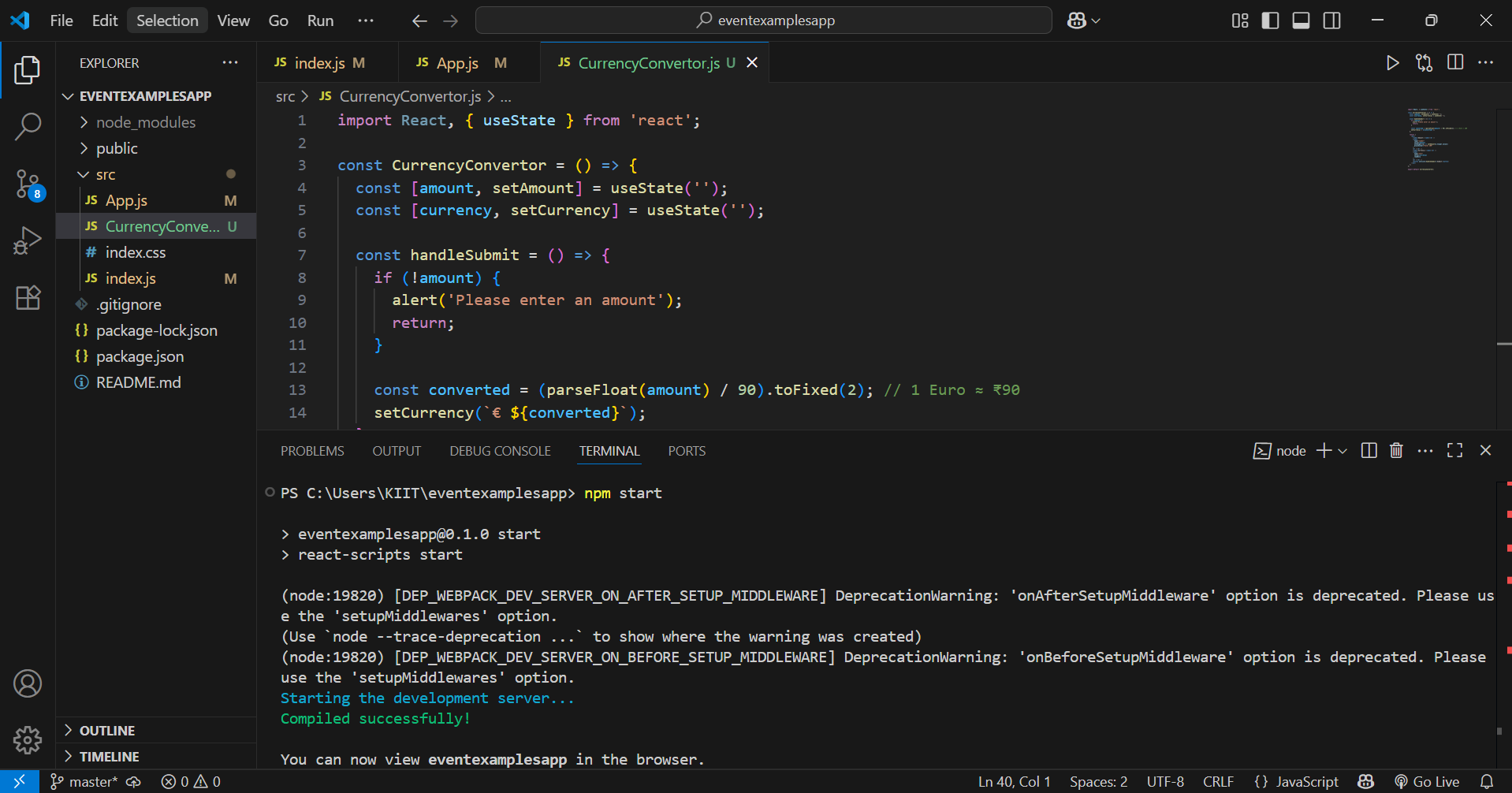
React follows a specific naming convention for event handling:

| **Native HTML** | **React JSX** |
| --- | --- |
| onclick | onClick |
| onchange | onChange |
| onmouseover | onMouseOver |
| onsubmit | onSubmit |

* React uses **camelCase** for event names
* The value passed must be **a function**, not a string







**4.**

**Explain about conditional rendering in React**

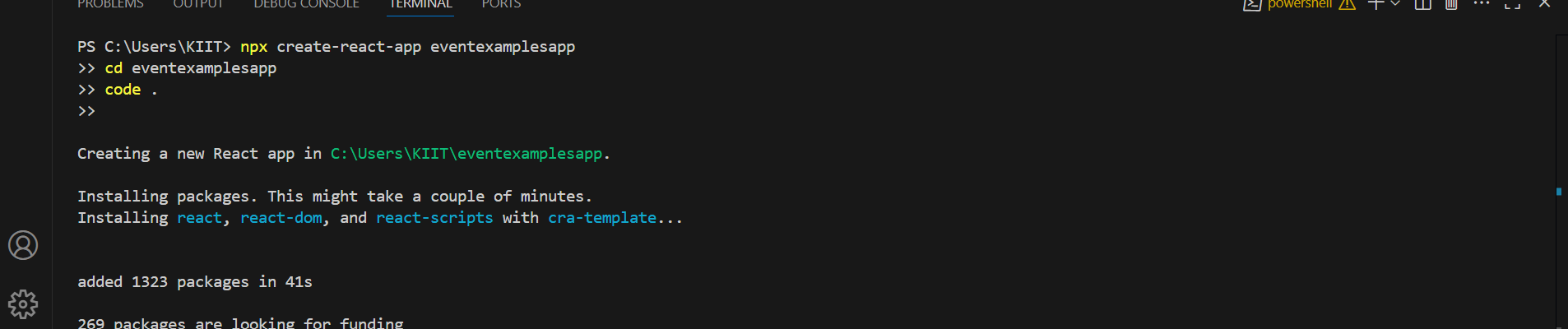
**Solution:- Conditional rendering** means showing **different UI elements** or components based on a **condition** (like if, else, ?, or &&).

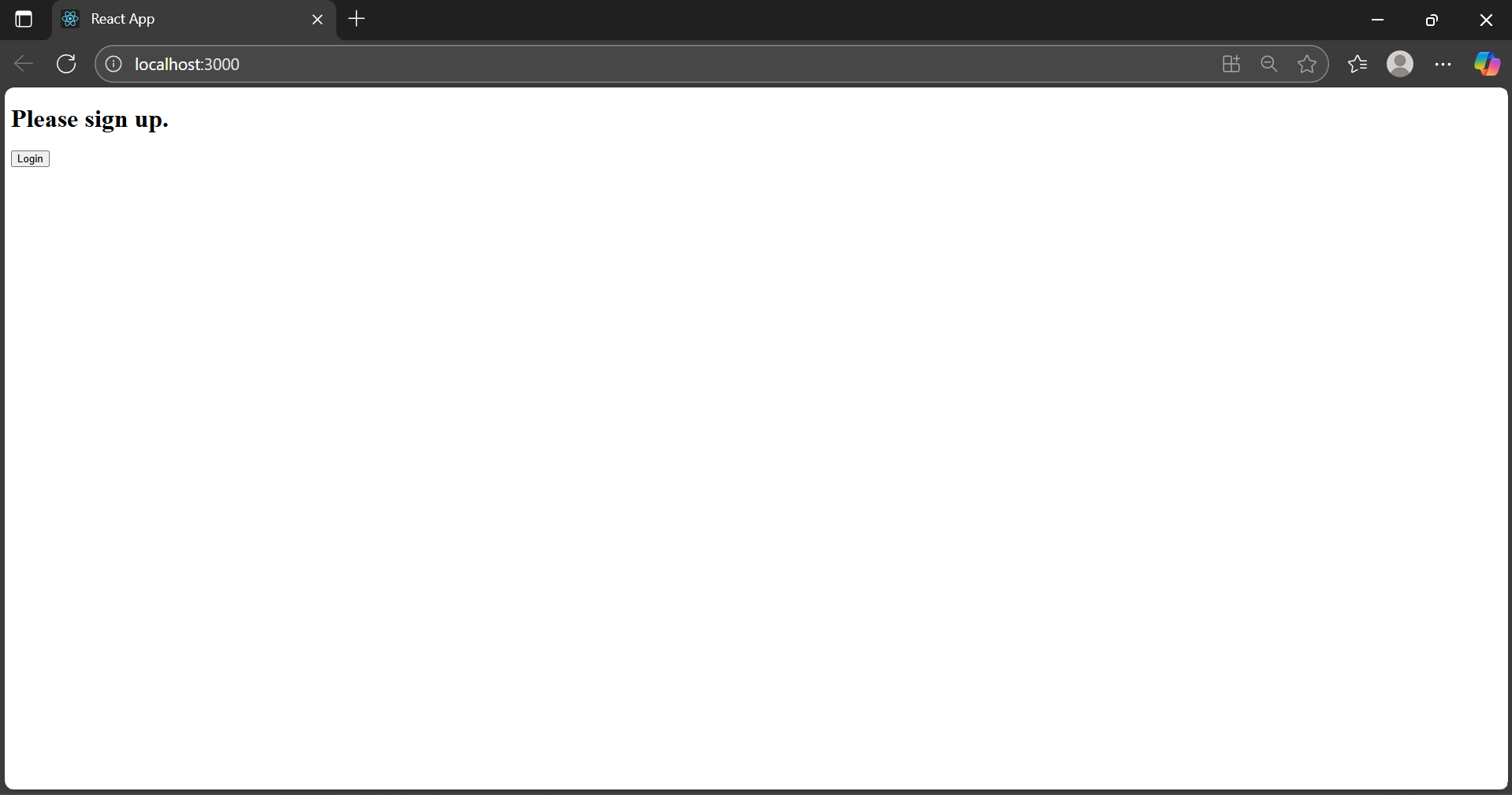
**Define element variables**

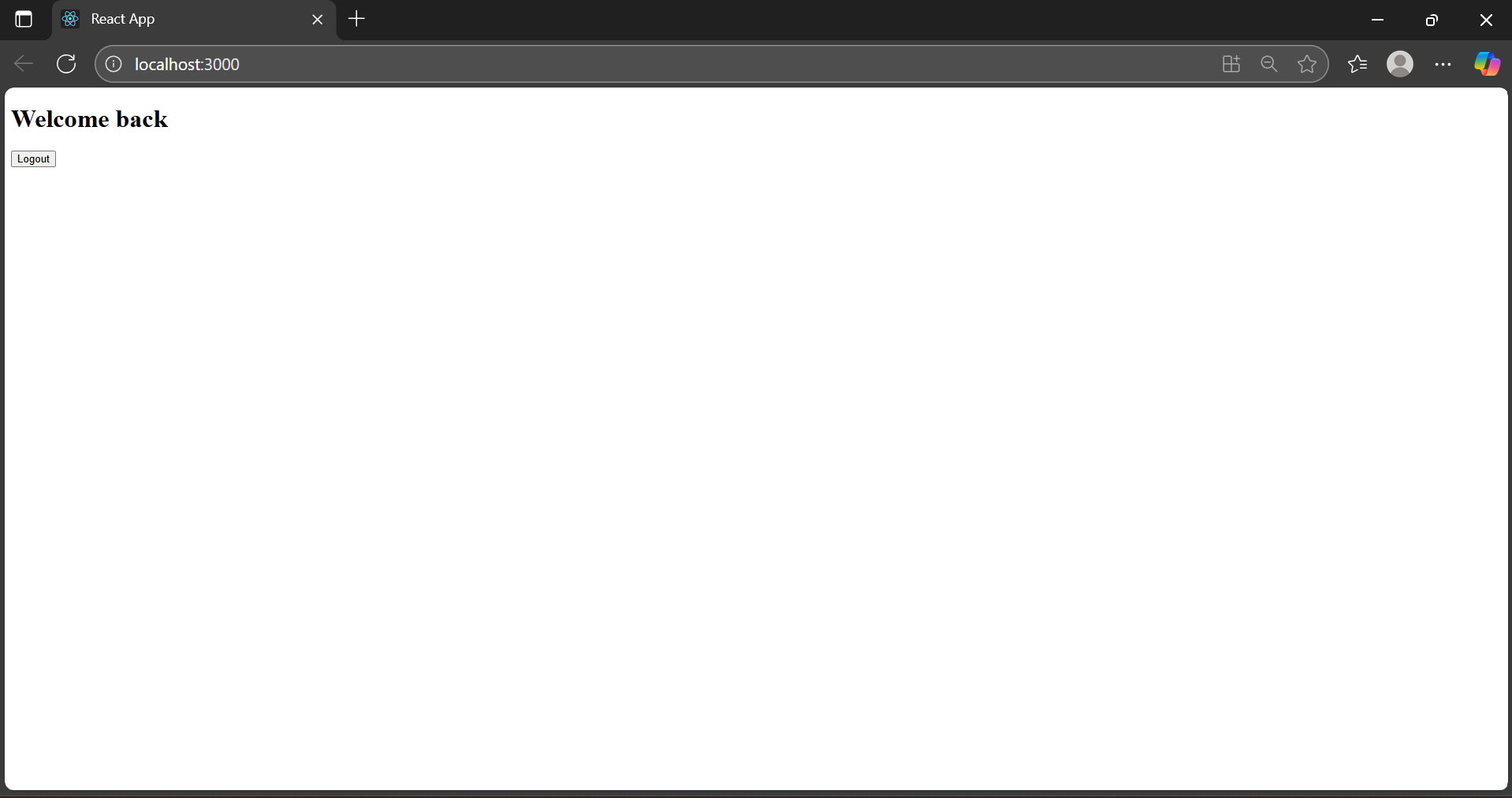
**Solution:-** Element variables let you **store JSX** in a variable so that it can be conditionally rendered or reused.It help simplify logic and improve readability when you need to show different JSX blocks.

**Explain how to prevent components from rendering**

**Solution:-** You can **prevent a component from rendering** by returning null inside it. Returning null means **nothing will be displayed**.







**5.**

**Explain various ways of conditional rendering**

**Solution:-** Conditional rendering in React means **showing different UI elements** based on some condition (like a user is logged in or not).

Here are the common ways:

#### a. ****Using if-else:****

if (isLoggedIn) {

return <h1>Welcome</h1>;

} else {

return <h1>Please Login</h1>;

}

#### b. ****Using Ternary Operator:****

#### <h1>{isLoggedIn ? "Welcome" : "Please Login"}</h1>

#### c. ****Using Logical AND (****&&****):****

{isLoggedIn && <h1>Welcome</h1>}

**Explain how to render multiple components**

**Solution:-** can render multiple components by:

#### a. Inside a parent <div> or <> fragment:

function App() {

return (

<div>

<Header />

<MainContent />

<Footer />

</div>

);

}

Or:

<>

<Header />

<MainContent />

<Footer />

</>

**Define list component**

**Solution:-** A **List Component** in React means a component that takes an array and displays a list from it.

**Explain about keys in React applications**

**Solution:- Keys** help React identify which items have changed, are added, or removed.

**Rule**: Always give a unique key prop when rendering a list.

{players.map(player => <li key={player.id}>{player.name}</li>)}

**Explain how to extract components with keys**

**Solution:-** Suppose you're displaying many users. You can extract a **single item component** like this:

function PlayerItem({ player }) {

return <li>{player.name}</li>;

}

function PlayerList({ players }) {

return (

<ul>

{players.map((p) => (

<PlayerItem key={p.id} player={p} />

))}

</ul>

);

}

This makes the code **cleaner** and **reusable**.

**Explain React Map, map() function**

**Solution:-** React uses JavaScript's native .map() to loop through arrays and return components.

