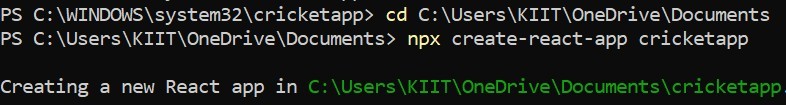
WEEK 7 MANDATORY HANDS-ON(REACT)

1. ES6 introduced several modern features that enhanced JavaScript's readability, structure, and performance. Key features include let and const for block-scoped variable declarations, arrow functions for concise function expressions, template literals for easier string handling, default parameters, destructuring for arrays and objects, spread/rest operators, Promises for handling asynchronous operations, modules (import/export), and new data structures like Map and Set.
2. The let keyword in JavaScript is used to declare variables that are **block-scoped**, meaning they are only accessible within the nearest set of curly braces {} (like inside loops or conditionals).
3. The difference between var and let lies in their scope and behaviour. var is **function-scoped** and allows redeclaration within the same scope, whereas let is **block-scoped** and throws an error if redeclared in the same scope. var variables are hoisted and initialized with undefined, while let variables are hoisted but remain uninitialized (in a "temporal dead zone") until the declaration is evaluated.
4. The const keyword also declares block-scoped variables, but these are **read-only** after their initial assignment, meaning reassignment is not allowed. However, if a const holds an object or array, the contents of that object or array can still be modified — only the reference cannot be changed.
5. ES6 also introduced classes to provide a cleaner and more intuitive syntax for creating objects and handling inheritance. A class in ES6 is defined using the class keyword and typically includes a constructor() method for initializing new instances and other methods for behaviour. This is syntactic sugar over JavaScript's prototype-based inheritance.
6. ES6 class inheritance uses the extends keyword to create a subclass that inherits from a parent class. Within the subclass constructor, the super() function must be called to invoke the parent class’s constructor and access its properties or methods. This allows for a more structured and hierarchical object model.
7. Arrow functions (=>) in ES6 provide a shorter syntax for writing functions and do not have their own this, arguments, or super binding, making them ideal for non-method functions like callbacks or array transformations. They are more concise and avoid common pitfalls with this in traditional functions.
8. Lastly, Set and Map are two new built-in data structures. A Set is a collection of **unique** values of any type, useful for ensuring no duplicates in a list. A Map is a collection of **key-value** pairs where keys can be of any type, offering more flexible and performant alternatives to plain objects for dynamic key-based storage.

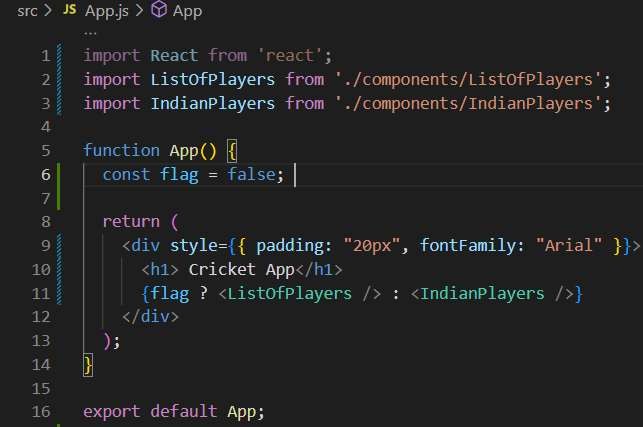
# Creating a React Application named “cricketapp” with the following components

****





App.js

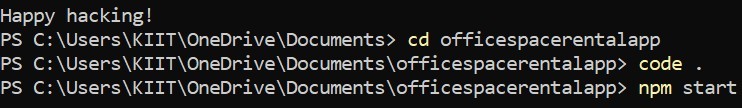
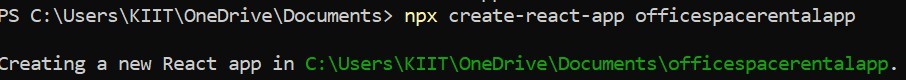


## OUTPUT

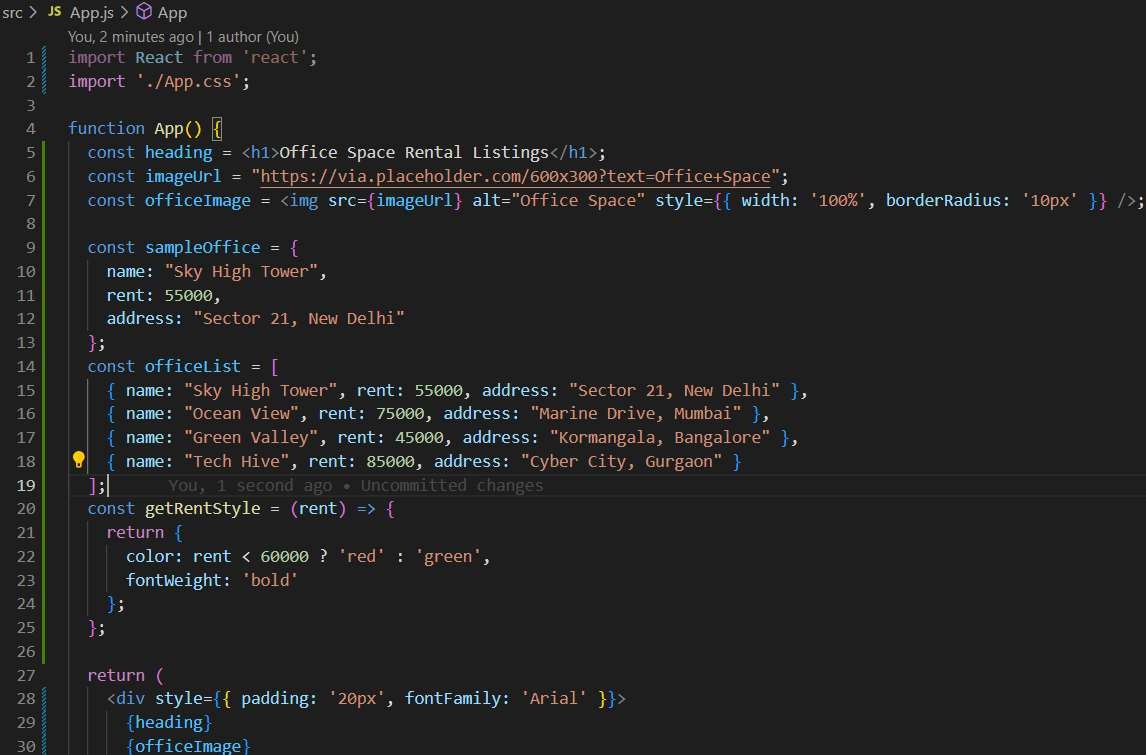


1. JSX is a syntax extension for JavaScript used in React. It allows developers to write HTML-like code directly within JavaScript, making the code more readable and easier to visualize UI structure.
2. ECMA is the standardized specification for JavaScript, maintained by ECMA International. Versions like ES5, ES6 (also known as ES2015), and beyond introduced modern features such as let, const, arrow functions, classes, promises, and modules, which make JavaScript more powerful and developer-friendly.
3. **React.createElement()** is a core React method that creates a virtual DOM element. It takes three arguments: the HTML tag or component, a props object, and children elements. JSX gets compiled into React.createElement() calls, which React uses to build the UI efficiently.
4. **Creating React nodes with JSX** involves writing elements like <h1>Hello</h1> inside a React component’s return statement. These elements are React nodes and represent the UI structure. You can nest elements and even use JavaScript logic inside them.
5. **Rendering JSX to the DOM** is done using ReactDOM.render() in vanilla React. You pass the JSX element and the target DOM node like this: ReactDOM.render(<App />, document.getElementById('root'));. This connects your virtual DOM to the actual DOM.
6. **JavaScript expressions in JSX** are embedded inside curly braces {}. You can use variables, functions, expressions, and even ternary operators.
7. **Inline CSS in JSX** is written as an object using camelCase property names. The style attribute accepts a JavaScript object instead of a regular string.

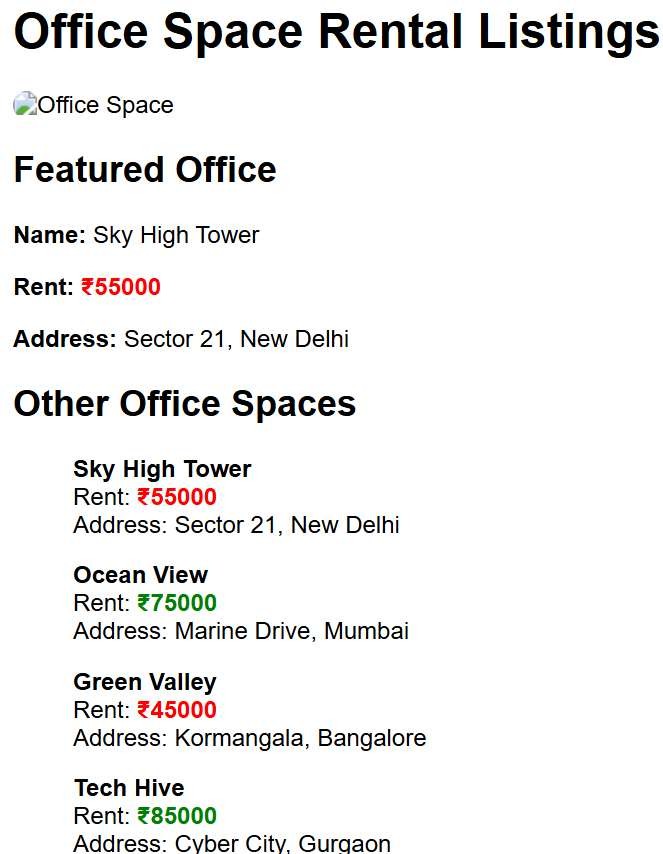
# Creating a React Application named “officespacerentalapp” which uses React JSX to create elements, attributes and renders DOM to display the page.



App.js

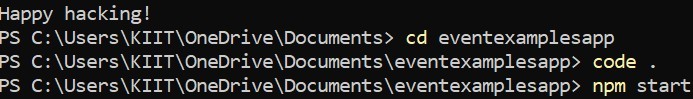
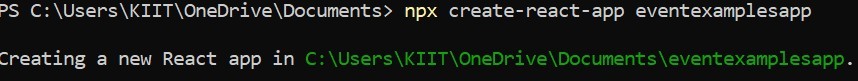


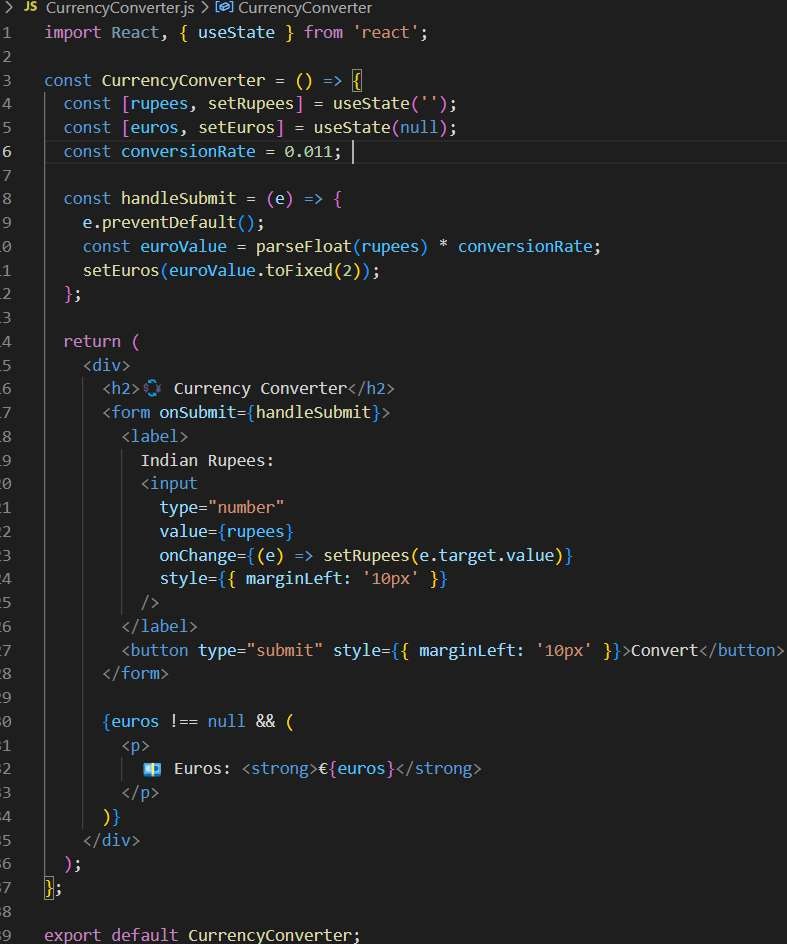
## OUTPUT



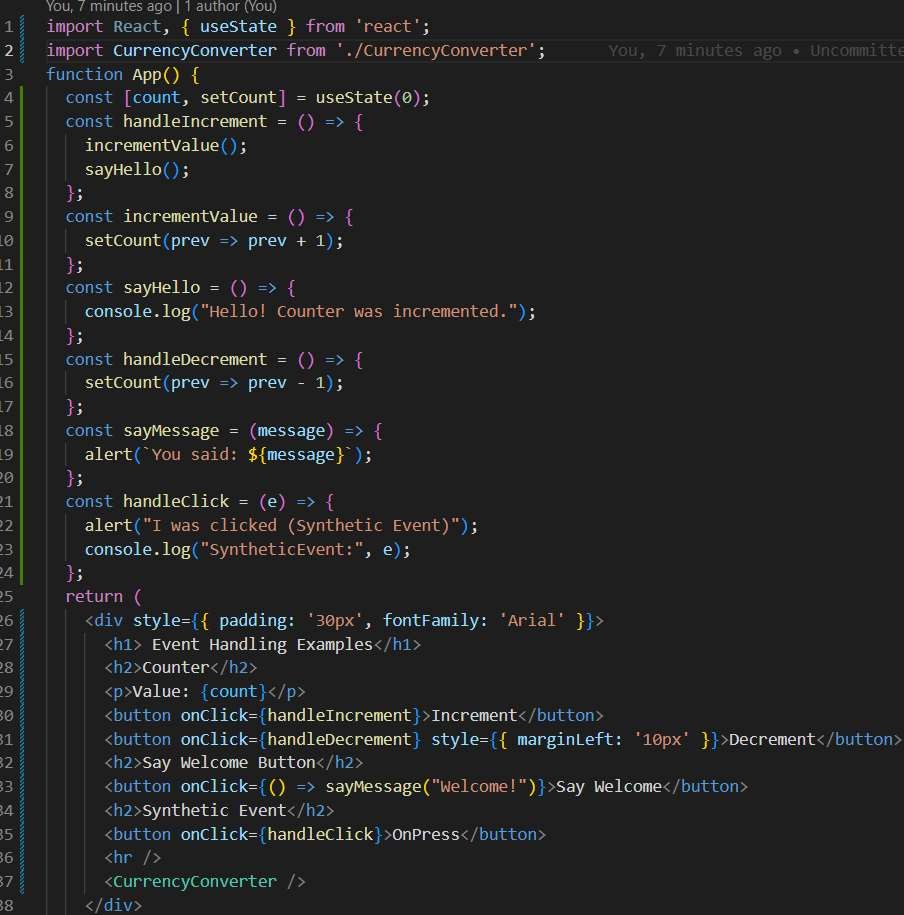
1. **React events** are similar to DOM events in regular HTML but are handled in a more consistent and cross-browser way using React’s synthetic event system. They include common event types like onClick, onChange, onSubmit, onMouseOver, and many others that can be attached to elements in JSX.
2. **Event handlers** in React are functions that are triggered in response to user actions such as clicks, key presses, or form submissions. You typically define an event handler function in your component and pass it as a prop to a JSX element. For example, <button onClick={handleClick}> Click Me</button> runs the handleClick function when the button is clicked.
3. A **synthetic event** is React’s wrapper around the browser’s native event system. It provides a consistent API for different browsers and mimics the behavior of the standard DOM events. This synthetic event is part of React’s event delegation system, which improves performance by attaching a single event listener to the root of the DOM.
4. React uses a specific **event naming convention** where event names are written in **camelCase** instead of lowercase. For instance, onclick in HTML becomes onClick in React, and onchange becomes onChange. This ensures consistency with JavaScript function naming and helps React identify and bind events correctly.

# Creating a React Application “eventexamplesapp” to handle various events of the form elements in HTML.

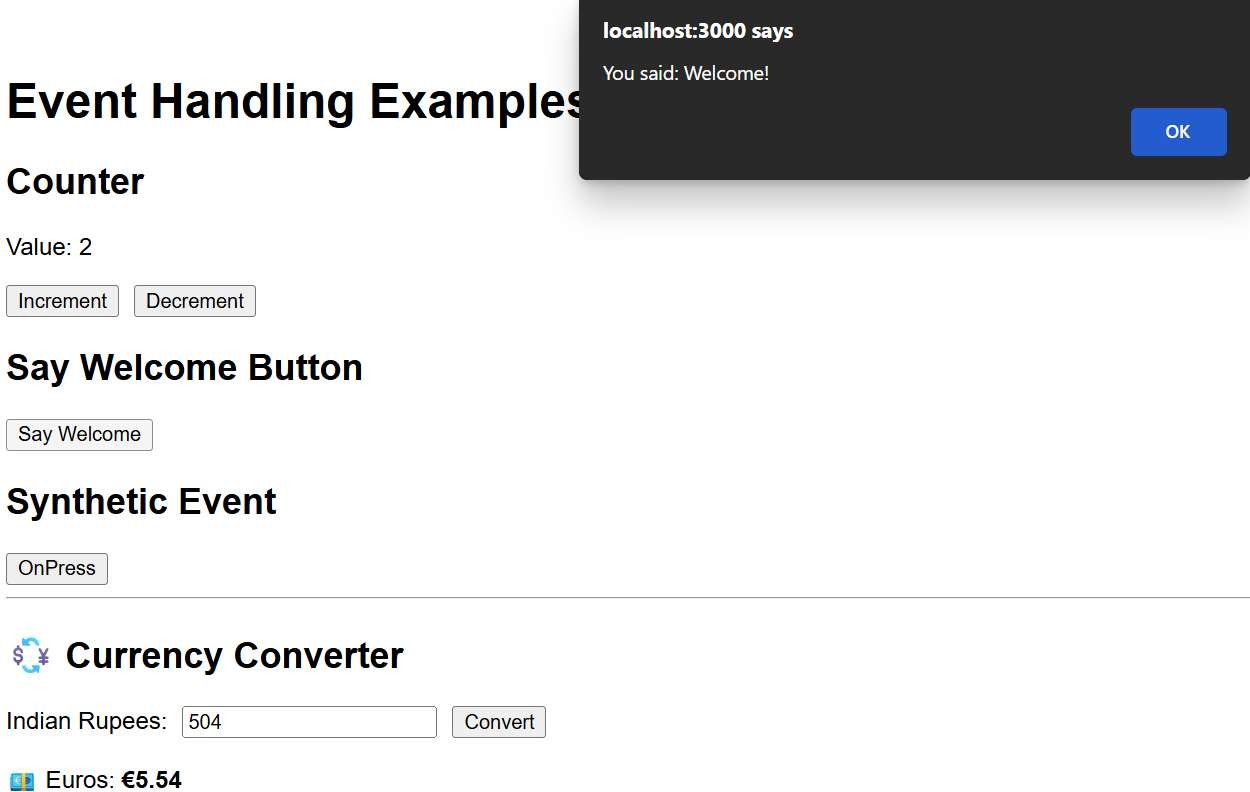




App.js

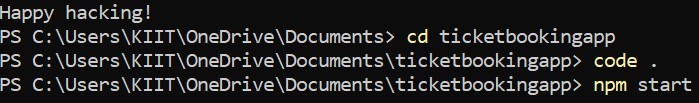
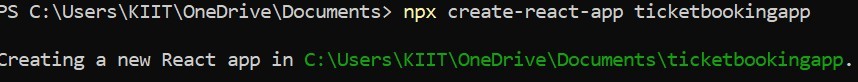


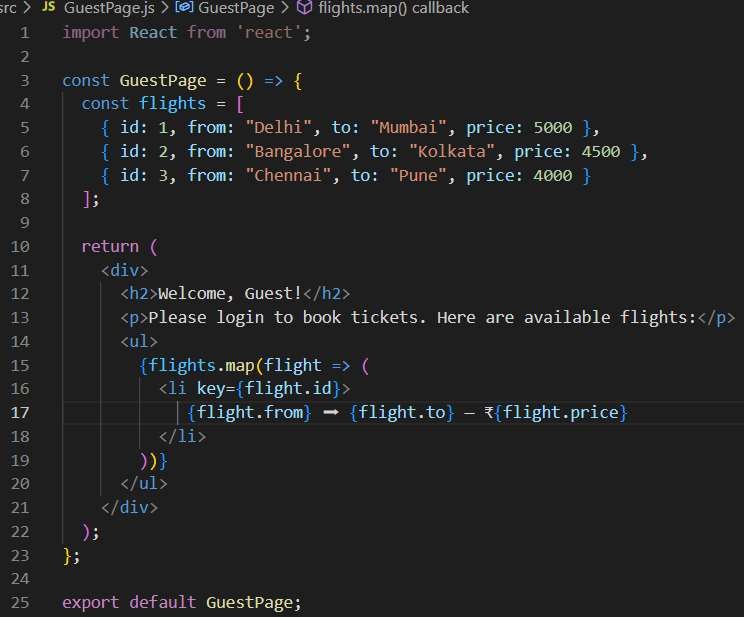
## OUTPUT

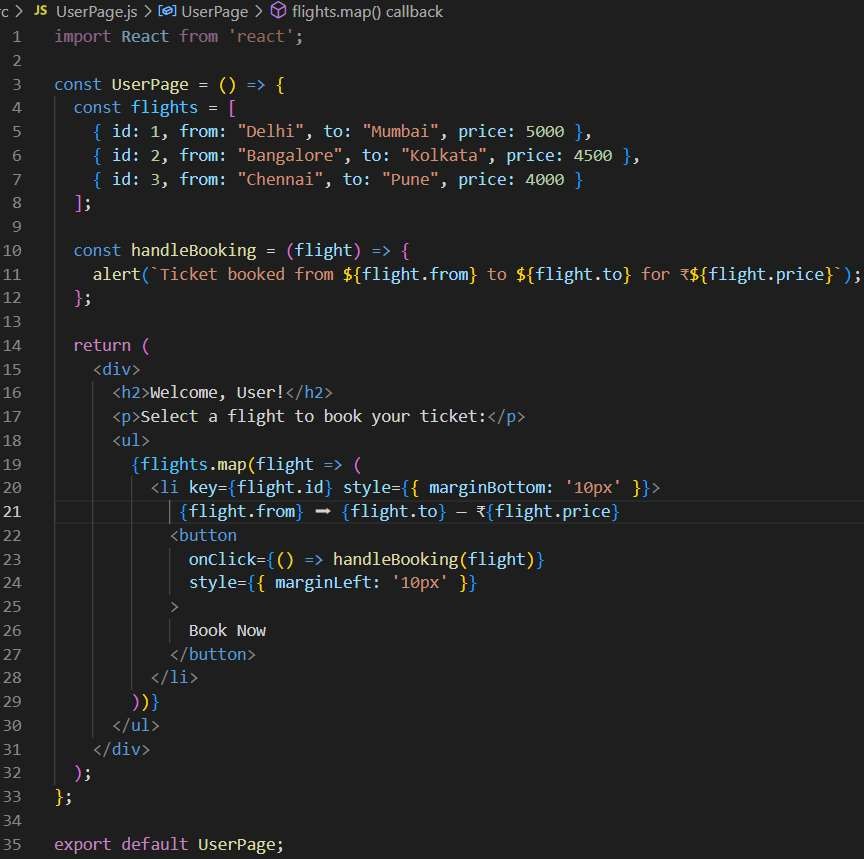


1. **Conditional rendering in React** refers to the technique of displaying different UI elements based on certain conditions. It works just like regular JavaScript conditions using if-else, ternary operators, logical AND, or switch statements inside JSX. This allows React to show, hide, or switch components dynamically based on application state or props.
2. **Element variables** in React are used to store JSX elements inside a variable, which can then be returned conditionally from the render() function or inside a functional component. For example, you might define let button; and assign it different JSX based on a condition before rendering it inside the return statement.
3. To **prevent components from rendering**, you can return null instead of JSX from a component. React will skip rendering that component entirely. This is useful for hiding components without unmounting them or for early exits based on props or state conditions.

# Creating a React Application named “ticketbookingapp” where the guest user can browse the page where the flight details are displayed whereas the logged in user only can book tickets.



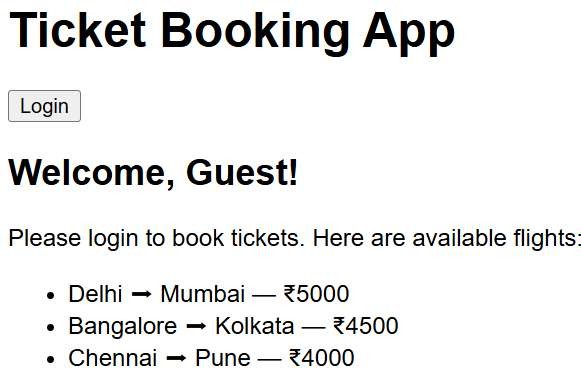


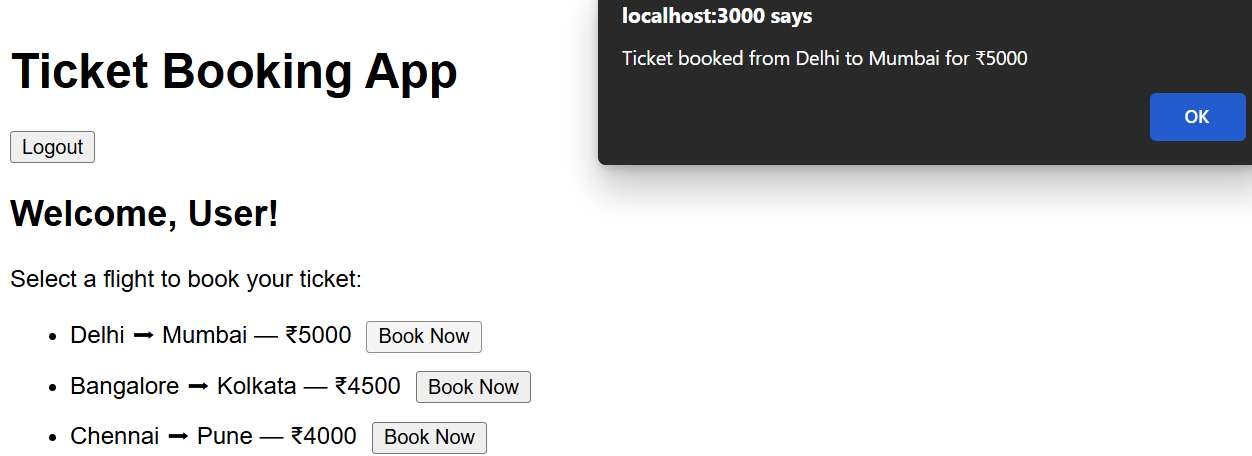
****

App.js



## OUTPUT





1. **Various ways of conditional rendering** in React include using JavaScript expressions like if-else, the **ternary operator** (condition ? trueCase : falseCase), **logical AND** (condition && <Component

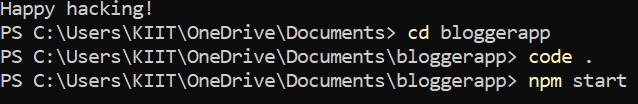
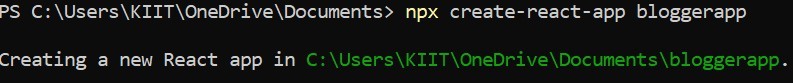
/>), and conditional **element variables**. These approaches allow components to show different UIs based on state or props.

1. **To render multiple components** in React, you can include them side by side inside a parent element such as a <div>, <React.Fragment>, or <> </>. This is useful when you want to display several parts of a UI together, such as a header, content area, and footer, each as a separate component.
2. A **list component** in React is used to render a collection of items dynamically, typically by using the .map() function on an array. It helps display repetitive UI structures like a list of users, products, or blog posts, where each item is rendered as a separate component or HTML element.
3. **Keys** in React are unique identifiers used when rendering lists of components. They help React identify which items have changed, been added, or removed, improving performance and preventing rendering issues. A key is usually a unique value like an id or index.
4. **To extract components with keys**, you define a separate child component (like ListItem) and

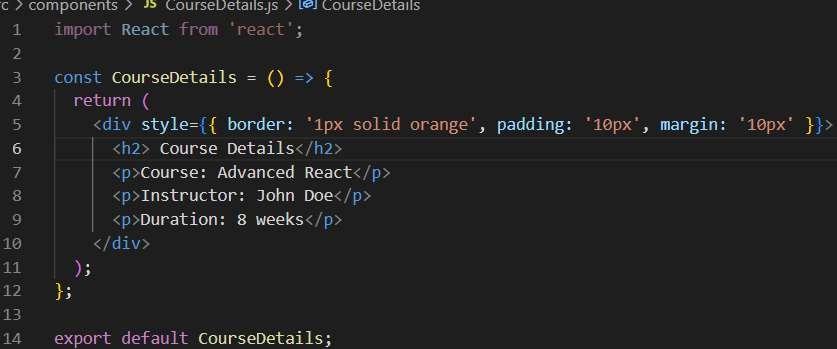
use .map() to render it for each item in an array. You pass a key prop to each child component to help React track them efficiently, especially when the list changes.

# Create a React App named “bloggerapp” in with 3 components. Book Details

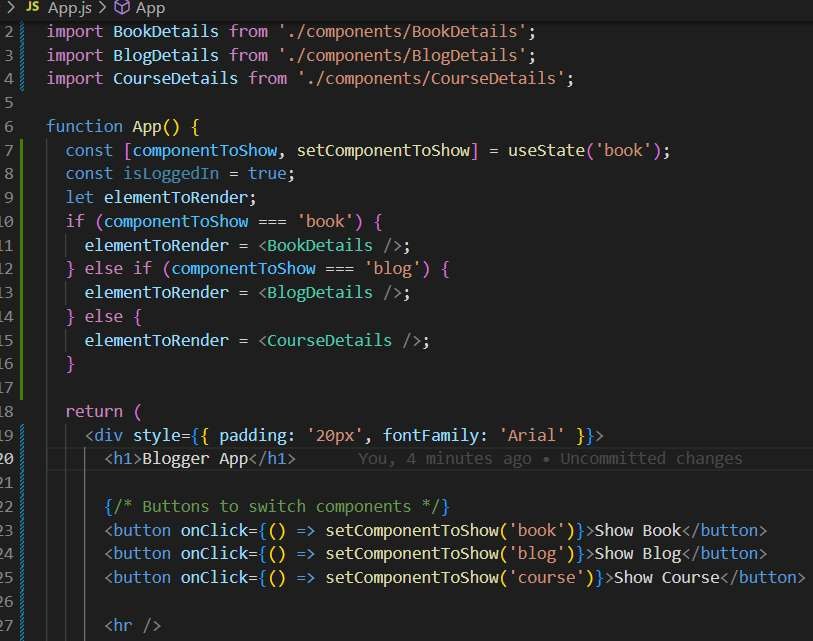
**Blog Details Course Details**

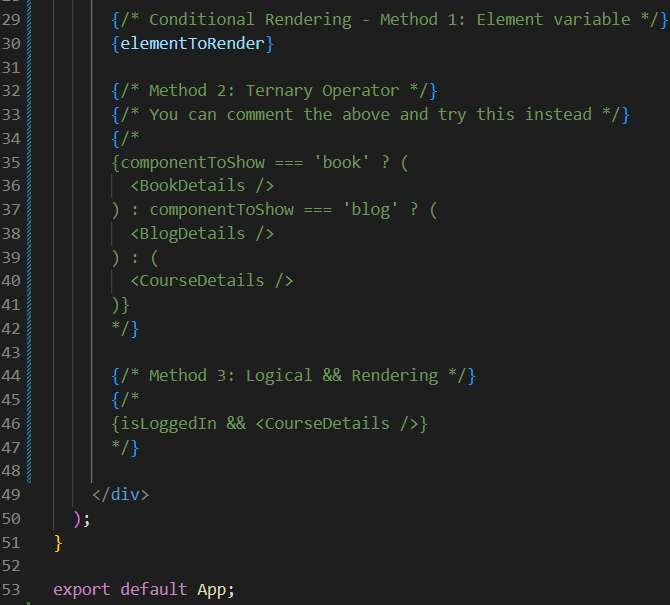




****

App.js





OUTPUT



