**WEEK 7**

**EX 9:Features of ES6 (ECMAScript 2015)**

ES6 brought significant improvements to JavaScript, including:

1. **Block-scoped declarations**: let and const
2. **Arrow functions**: Shorter function syntax
3. **Classes**: Clean OOP support
4. **Template literals**: Easy string interpolation using backticks (`)
5. **Default parameters**: Function parameters with default values
6. **Destructuring**: Unpacking values from arrays or objects
7. **Modules**: import / export statements
8. **Promises**: Built-in asynchronous handling
9. **Spread and rest operators**: ... for expanding and grouping values
10. **New data structures**: Map, Set

**let in JavaScript**

* Declares block-scoped variables (only available within {} block).
* Can be updated but **not redeclared** in the same scope.

let department = "Cardiology";

department = "Neurology";

**Differences Between var and let**

| **Feature** | **var** | **Let** |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Yes (initialized as undefined) | Yes (but not initialized) |
| Redeclaration | Allowed | Not allowed in same scope |

**const in JavaScript**

* Used to declare **block-scoped constants**.
* Cannot be reassigned after declaration.
* Does **not make object properties immutable**.

const hospitalName = "City Hospital";

// hospitalName = "New Name";

**ES6 Class Fundamentals**

Classes provide a blueprint for objects (e.g., Patients, Doctors):

class Patient {

constructor(name, age) {

this.name = name;

this.age = age;

}

getInfo() {

return `${this.name} is ${this.age} years old.`;

}

}

**ES6 Class Inheritance**

Allows one class to inherit properties and methods from another.

class User {

constructor(username) {

this.username = username;

}

}

class Doctor extends User {

constructor(username, specialization) {

super(username);

this.specialization = specialization;

}

}

**Arrow Functions in ES6**

Shorter function syntax and **lexical this binding**

function greet(name) {

return `Hello, ${name}`;

}

const greet = (name) => `Hello, ${name}`;

**Set() and Map() in ES6**

| **Feature** | **Set()** | **Map()** |
| --- | --- | --- |
| Purpose | Stores **unique values** | Stores **key-value pairs** |
| Keys | Not applicable | Keys can be any data type |

// Set

const departments = new Set();

departments.add("Cardiology");

departments.add("Neurology");

// Map

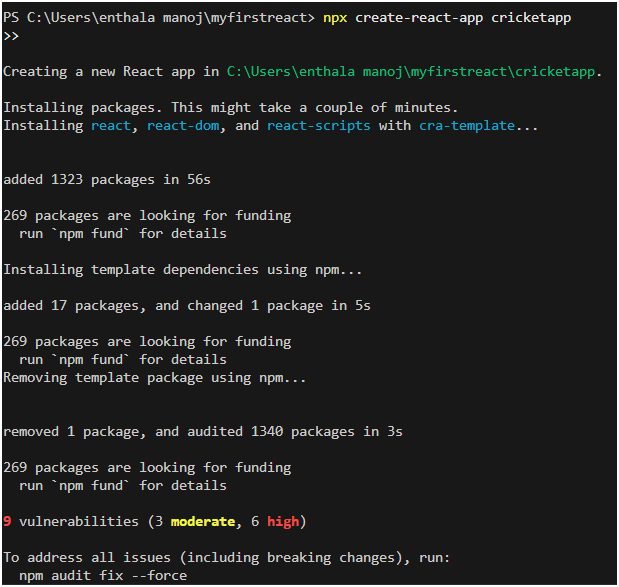
const doctorMap = new Map();

doctorMap.set("D001", "Dr. Smith");

doctorMap.set("D002", "Dr. Priya");

Ex1: Create a React Application named “cricketapp” with the following components:

Step 1:Open your terminal and run:

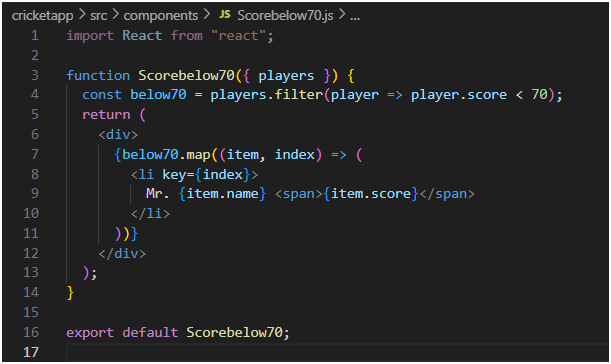


Step 2: ListofPlayers

* Declare an array with 11 players and store details of their names and scores using the map feature of ES6



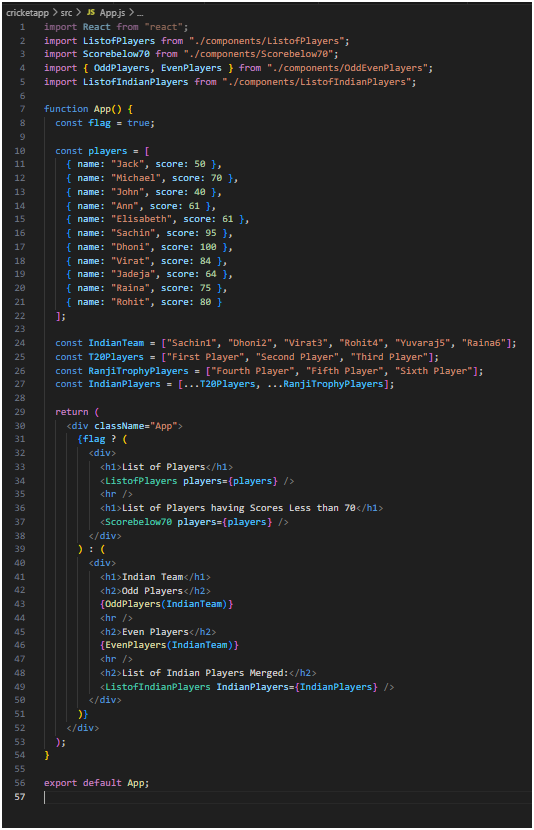
* Filter the players with scores below 70 using arrow functions of ES6.



1. IndianPlayers
   1. Display the Odd Team Player and Even Team players using the Destructuring features of ES6



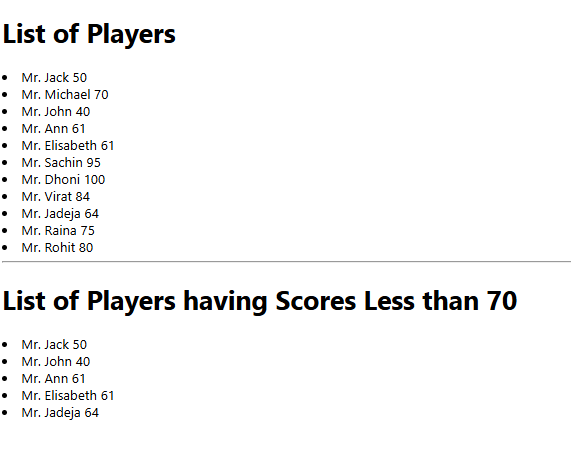
* 1. Declare two arrays T20players and RanjiTrophy players and merge the two arrays and display them using the Merge feature of ES6



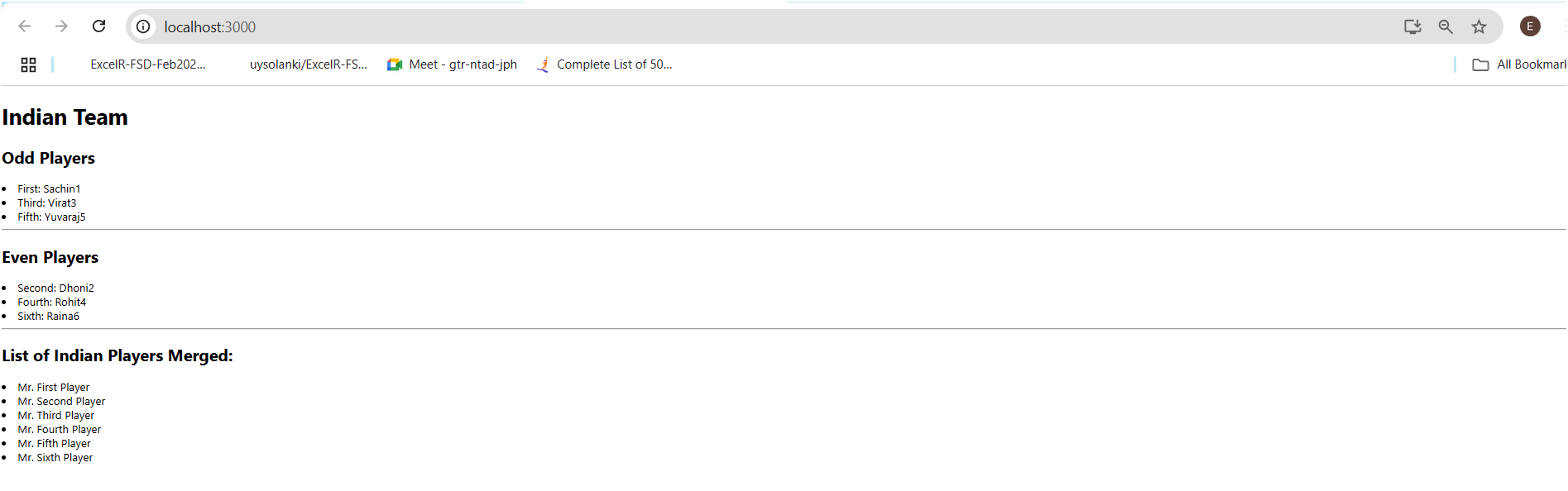
Display these two components in the same home page using a simple if else in the flag variable.

**Output:**

When Flag=true



When Flag=false



Ex10: **Define JSX**

JSX (**JavaScript XML**) is a syntax extension for JavaScript used with React to describe UI components in a way that resembles HTML.

const heading = <h1>Welcome to City Hospital</h1>;

**Note:** JSX is not HTML — it's syntactic sugar for React.createElement() calls. It requires transpilation using Babel.

**Explain about ECMAScript (ECMA Script)**

ECMAScript is the **standard** that JavaScript is based on. React development typically uses modern ECMAScript features (ES6 and beyond), such as:

* let, const
* Arrow functions
* Classes
* Modules (import, export)
* Destructuring
* Template literals

React code relies heavily on these ES features for better readability and maintainability.

**Explain React.createElement()**

React.createElement() is a method used to create **virtual DOM nodes**.

**Example:**

React.createElement('h1', { className: 'title' }, 'Welcome');

This is equivalent to:

<h1 className="title">Welcome</h1>

JSX gets compiled into React.createElement() calls during the build process.

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

You can create UI nodes using JSX syntax inside React components.

const Department = () => {

return <h2>Cardiology Department</h2>;

};

JSX nodes can include nested elements, expressions, and components.

**Define how to render JSX to the DOM**

Use ReactDOM.render() to render a JSX element to the browser DOM.

import React from 'react';

import ReactDOM from 'react-dom';

const App = () => <h1>Hospital Management System</h1>;

ReactDOM.render(<App />, document.getElementById('root'));

Ensure there’s a <div id="root"></div> in your HTML file.

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

You can embed JavaScript **expressions** (not statements) inside JSX using {}.

const name = "Dr. John";

const greeting = <h2>Welcome, {name}</h2>;

Valid expressions include variables, function calls, and math:

<h2>{10 + 5} Beds Available</h2>

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

You can apply inline styles using the style attribute with a JavaScript object.

const titleStyle = {

color: 'blue',

fontSize: '20px'

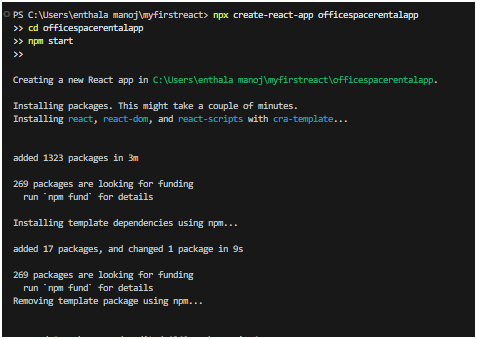
};

<h1 style={titleStyle}>City Hospital</h1>

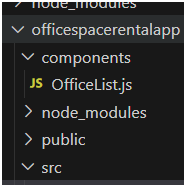
Property names must use **camelCase** (backgroundColor, fontSize), not kebab-case.

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

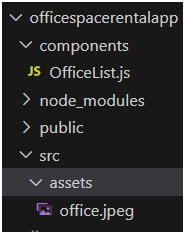
**Step 1: Create React App**



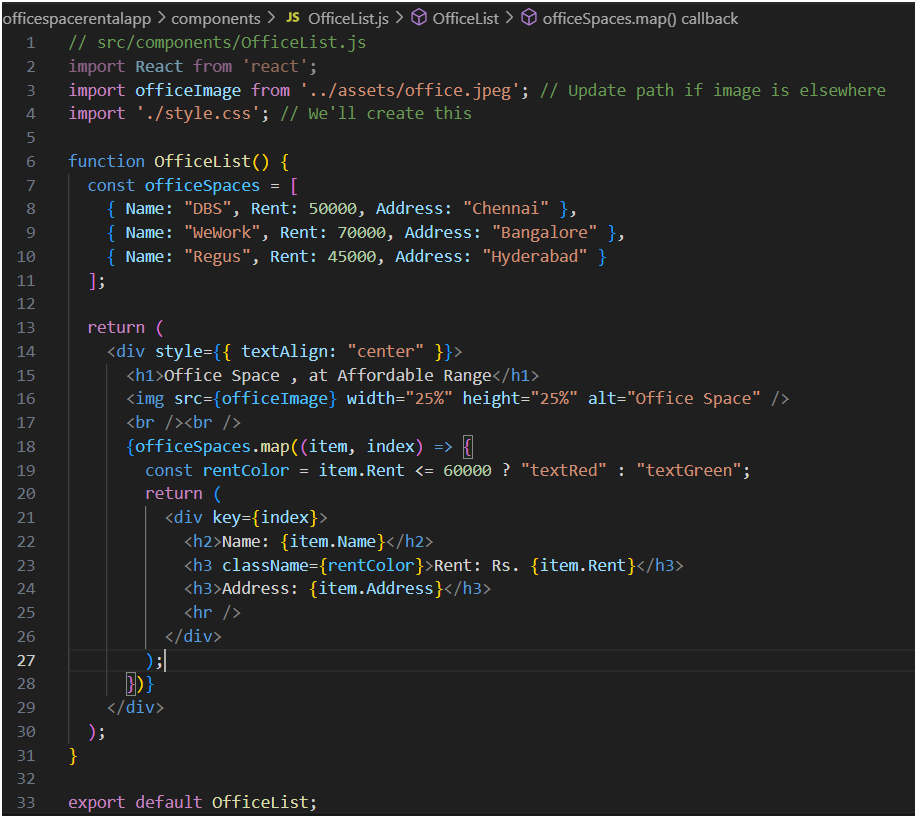
**Step 2: Add Project Structure**

****

**Step 3: Add Office Image**

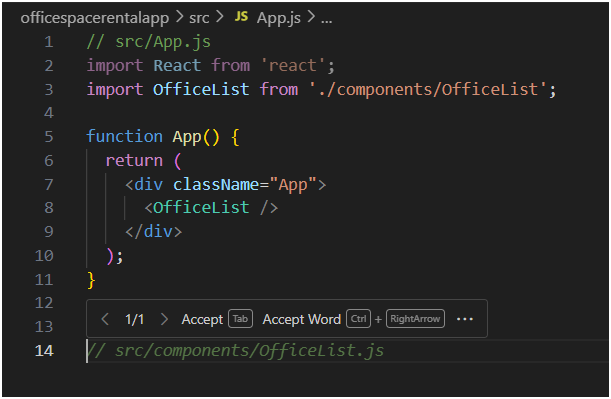


**Step 4: Write Component Code**

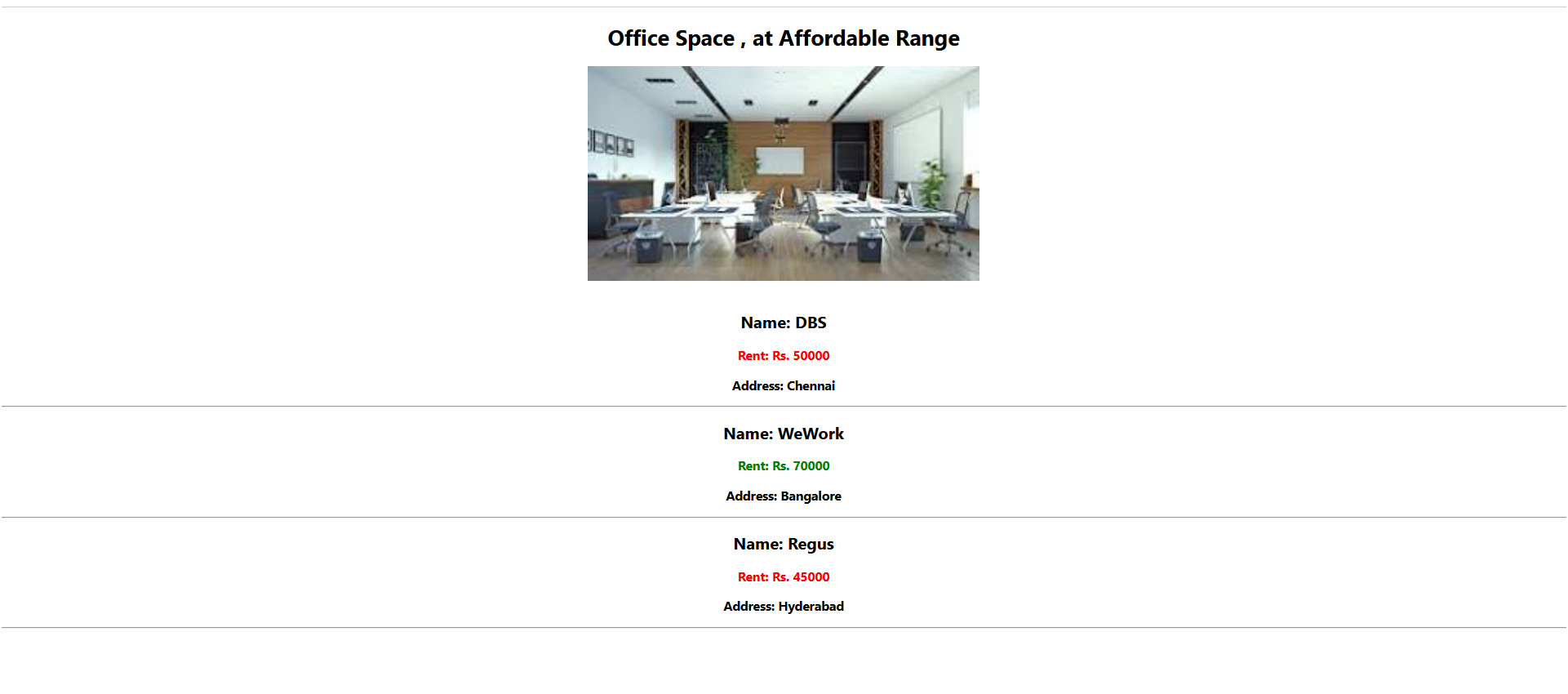
**Step 5: Create CSS File**

****

**Step 6: Modify App.js**

****

**Output**

****

**Ex:11: Explain React Events**

**React events are wrapper objects built on top of native browser events. They allow React components to respond to user interactions such as clicks, form submissions, key presses, etc.**

**Example:**

**<button onClick={handleClick}>Discharge Patient</button>**

**In this case, onClick is the React event attached to the button.**

**Explain About Event Handlers**

**Event handlers are functions that get called when an event occurs. They are typically passed as props to JSX elements.**

**Example:**

**function handleClick() {**

**alert('Patient record updated!');**

**}**

**<button onClick={handleClick}>Update</button>**

**React will call handleClick when the button is clicked.**

**Event handlers can also be defined as arrow functions:**

**const handleDelete = () => {**

**console.log("Patient deleted.");**

**};**

**Define Synthetic Event**

**A SyntheticEvent is a cross-browser wrapper around the native browser event system in React. It ensures events behave consistently across all browsers.**

**Example usage:**

**function handleInputChange(e) {**

**console.log(e.target.value); // SyntheticEvent wraps the native input event**

**}**

**SyntheticEvent wraps methods like stopPropagation() and preventDefault() and unifies them across all platforms.**

**Identify React Event Naming Convention**

| **Attribute** | **React Convention** |
| --- | --- |
| **Event Name** | **camelCase (onClick)** |
| **Handler Value** | **Pass a function** |

**Example Comparison:**

**HTML:**

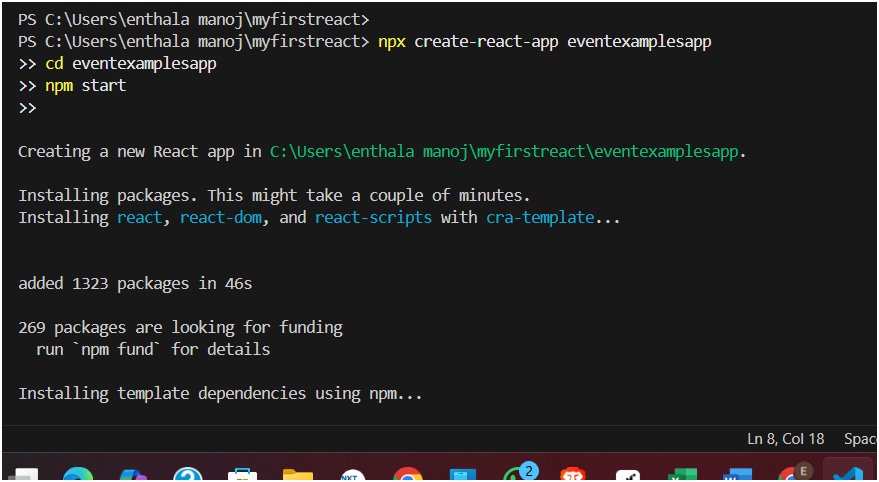
**<button onclick="doSomething()">Click Me</button>**

**React:**

**<button onClick={doSomething}>Click Me</button>**

**Always use camelCase and pass function references, not strings.**

**Ex 11:Create a React Application “eventexamplesapp” to handle various events of the form elements in HTML.**

**Step1:** ****

**Step 2:Create CurrencyConvertor.js Component**

**src/components/CurrencyConvertor.js**

**Code:**

**step 3:Modify App.js**

**Replace your App.js content with this:**

****

**How This Implements Your Objectives**

**React Events**

* **onClick, onChange, onSubmit**

**Event Handlers**

* **increment, decrement, sayHello, sayWelcome, handleClick**

**Synthetic Events**

* **handleClick uses e.preventDefault() (Synthetic Event)**

**React Event Naming**

* **All handlers are camelCase (onClick, onSubmit)**

**Use of this**

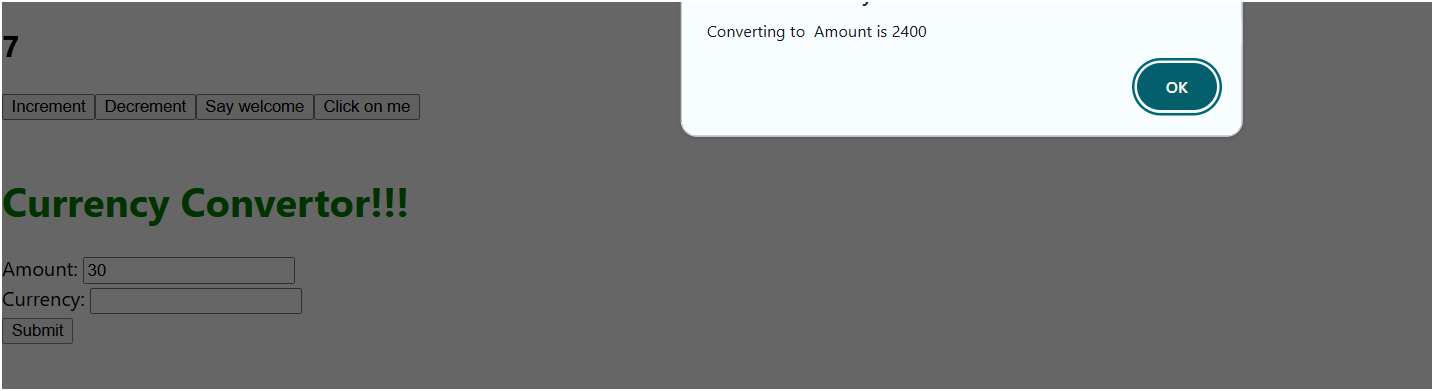
* **this.increment**
* **this.sayHello**

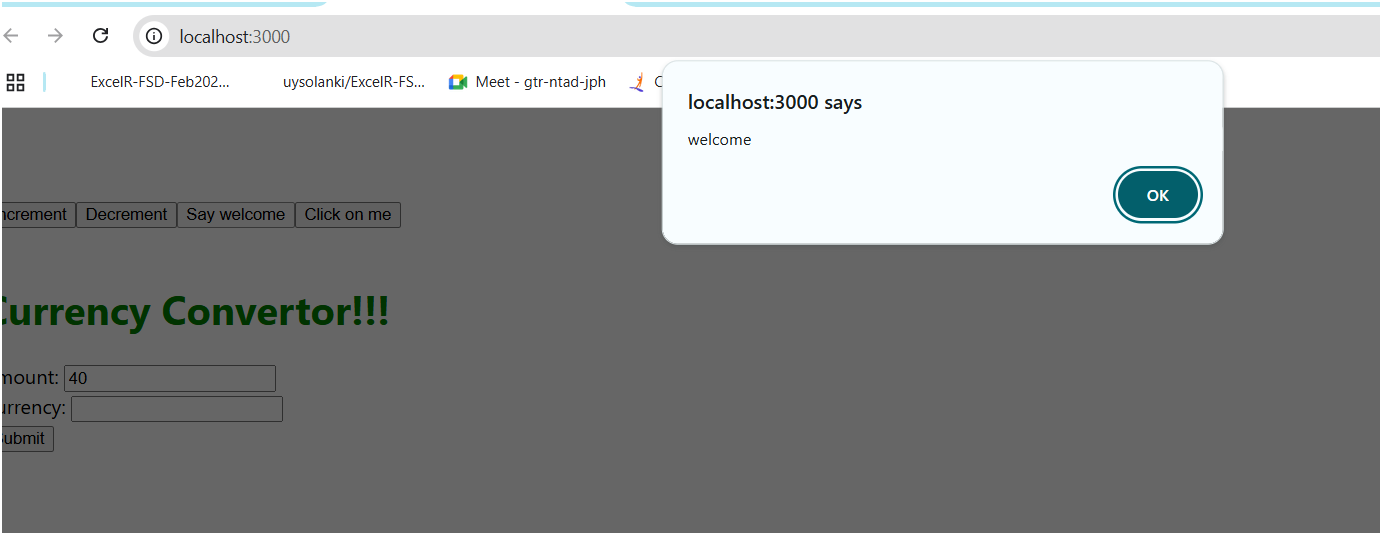
**Run the App**

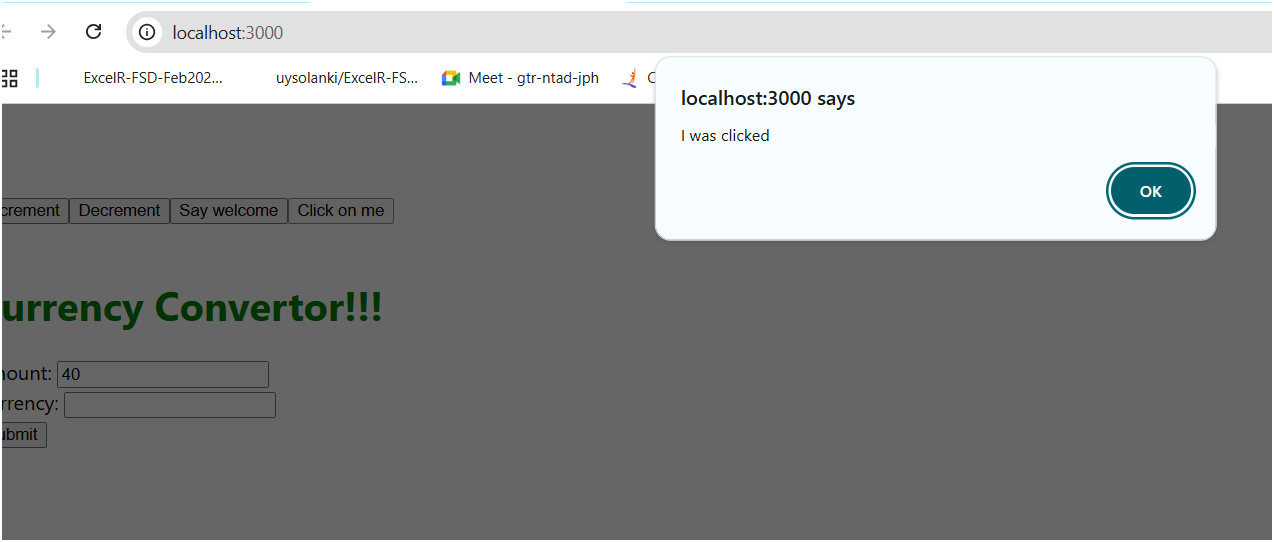
**npm start**

**output:**

****

****

****

****

**EX 12: Conditional Rendering in React**

**Conditional rendering in React lets you display different content based on conditions, similar to how if statements work in JavaScript.**

**1. Using if/else Conditions**

**You can define conditions before returning JSX**

**function Greeting({ isLoggedIn }) {**

**if (isLoggedIn) {**

**return <h1>Welcome back!</h1>;**

**} else {**

**return <h1>Please sign in</h1>;**

**}**

**}**

**2. Using Element Variables**

**You can assign JSX to a variable and use logic to update it before rendering.**

**function Message({ unread }) {**

**let content;**

**if (unread > 0) {**

**content = <p>You have {unread} unread messages</p>;**

**} else {**

**content = <p>You're all caught up!</p>;**

**}**

**return <div>{content}</div>;**

**}**

**3. Using Ternary Operator**

**Inline conditional rendering with the ternary operator:**

**const UserStatus = ({ isOnline }) => (**

**<span>{isOnline ? 'Online' : 'Offline'}</span>**

**);**

**4. Using Logical AND (&&)**

**Use && to render content only if the condition is true.**

**function Warning({ show }) {**

**return (**

**<div>**

**{show && <p className="warning">This is a warning message!</p>}**

**</div>**

**);**

**}**

**5. Preventing Component Rendering**

**Return null to render nothing.**

**function HiddenPanel({ visible }) {**

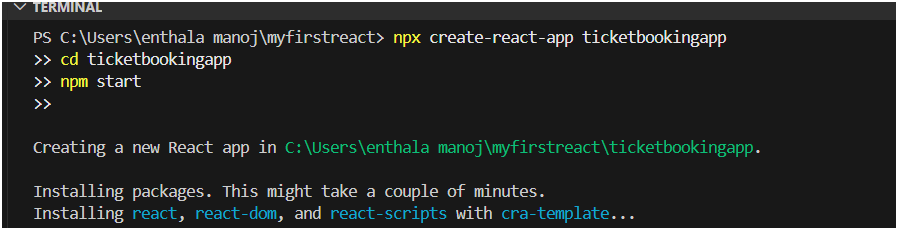
**if (!visible) return null;**

**return <div>This panel is visible</div>;**

**}**

**This approach is useful when you want to completely skip rendering a component, not just hide it with CSS.**

**Create the React App**

** Modify App.js**

**Replace all code in App.js with:**

****

**Run the App**

**npm start**

**When Logged Out:**

****

**When Logged In:**

** Clicking Login and Logout switches between views.**

**Ex 13:**

**1. Various Ways of Conditional Rendering in React**

**React lets you render elements or components based on conditions. Here are several ways to do it:**

**a. Using if/else Statement**

**function Greeting({ isLoggedIn }) {**

**if (isLoggedIn) {**

**return <h1>Welcome!</h1>;**

**} else {**

**return <h1>Please log in.</h1>;**

**}**

**}**

**b. Using Element Variables**

**function Status({ online }) {**

**let statusMessage;**

**if (online) {**

**statusMessage = <span>Online</span>;**

**} else {**

**statusMessage = <span>Offline</span>;**

**}**

**return <div>{statusMessage}</div>;**

**}**

**c. Using Ternary Operator**

**function Access({ isAdmin }) {**

**return <p>{isAdmin ? 'Access Granted' : 'Access Denied'}</p>;**

**}**

**d. Using Logical AND (&&)**

**function Notification({ hasMessage }) {**

**return <div>{hasMessage && <p>You have new messages</p>}</div>;**

**}**

**e. Returning null**

**function Hint({ visible }) {**

**if (!visible) return null;**

**return <p>Helpful hint displayed</p>;**

**}**

**2. How to Render Multiple Components**

**You can render multiple components by wrapping them:**

**a. Using a Parent Element**

**function App() {**

**return (**

**<div>**

**<Header />**

**<Main />**

**<Footer />**

**</div>**

**);**

**}**

**b. Using Fragments**

**function App() {**

**return (**

**<>**

**<Header />**

**<Main />**

**</>**

**);**

**}**

**3. Define a List Component**

**A List Component renders a set of similar elements from an array.**

**function ItemList({ items }) {**

**return (**

**<ul>**

**{items.map(item => <li key={item.id}>{item.name}</li>)}**

**</ul>**

**);**

**}**

**4. About Keys in React**

**Keys help React identify which items have changed, been added, or removed.**

* **Must be unique and stable**
* **Prefer using a unique id instead of the index**

**const items = ['A', 'B', 'C'];**

**const list = items.map((val, idx) => <li key={idx}>{val}</li>);**

**5. Extract Components with Keys**

**You can extract a list item into its own component:**

**function ListItem({ value }) {**

**return <li>{value}</li>;**

**}**

**function ListContainer({ values }) {**

**return (**

**<ul>**

**{values.map((v, index) => (**

**<ListItem key={index} value={v} />**

**))}**

**</ul>**

**);**

**}**

**Key must be passed at the point of mapping, not inside the ListItem itself.**

**6. React .map() Function**

**.map() is used to loop over arrays and return JSX:**

**const numbers = [1, 2, 3, 4];**

**const listItems = numbers.map(num => (**

**<li key={num}>{num}</li>**

**));**

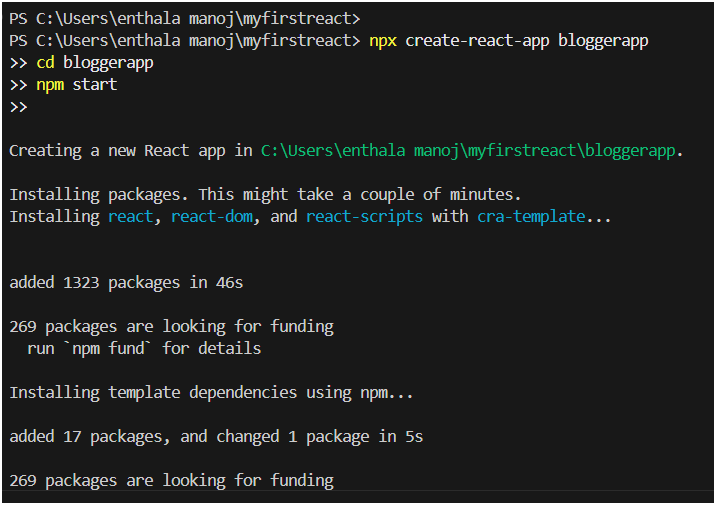
* **Always use a key inside .map()**
* **The return value must be valid JSX**

**EX: 13 Create the React App**

**npx create-react-app bloggerapp**

**cd bloggerapp**

**npm start**

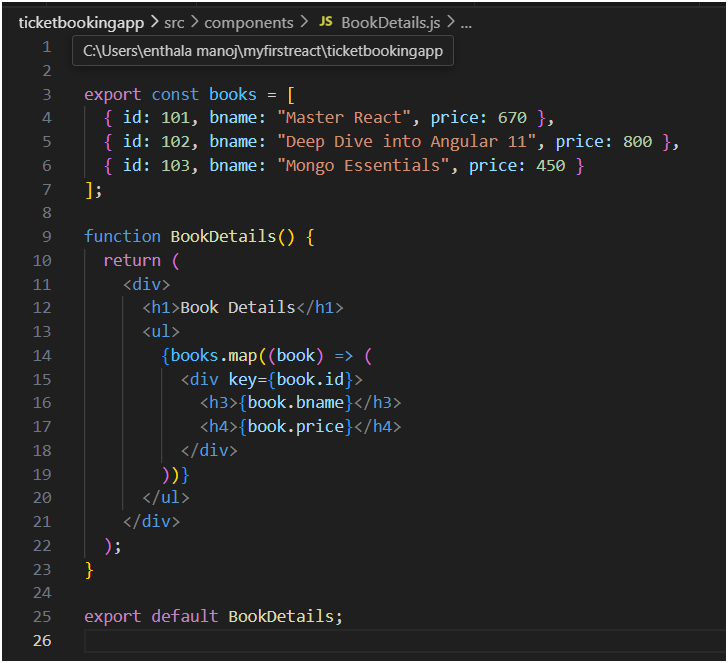
****

**Create BlogDetails.js**

**File: src/components/BlogDetails.js**

**Create BookDetails.js**

**File: src/components/BookDetails.js**

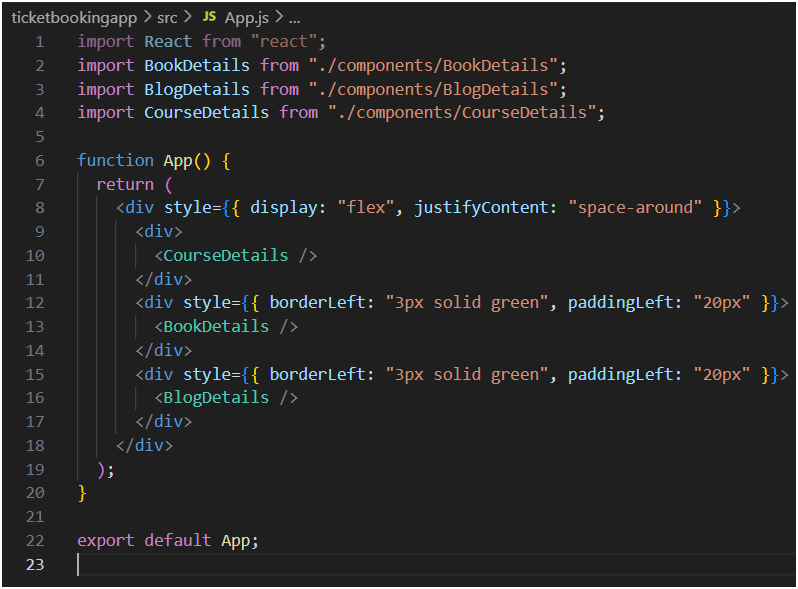
**Create CourseDetails.js**

**File: src/components/CourseDetails.js**

****

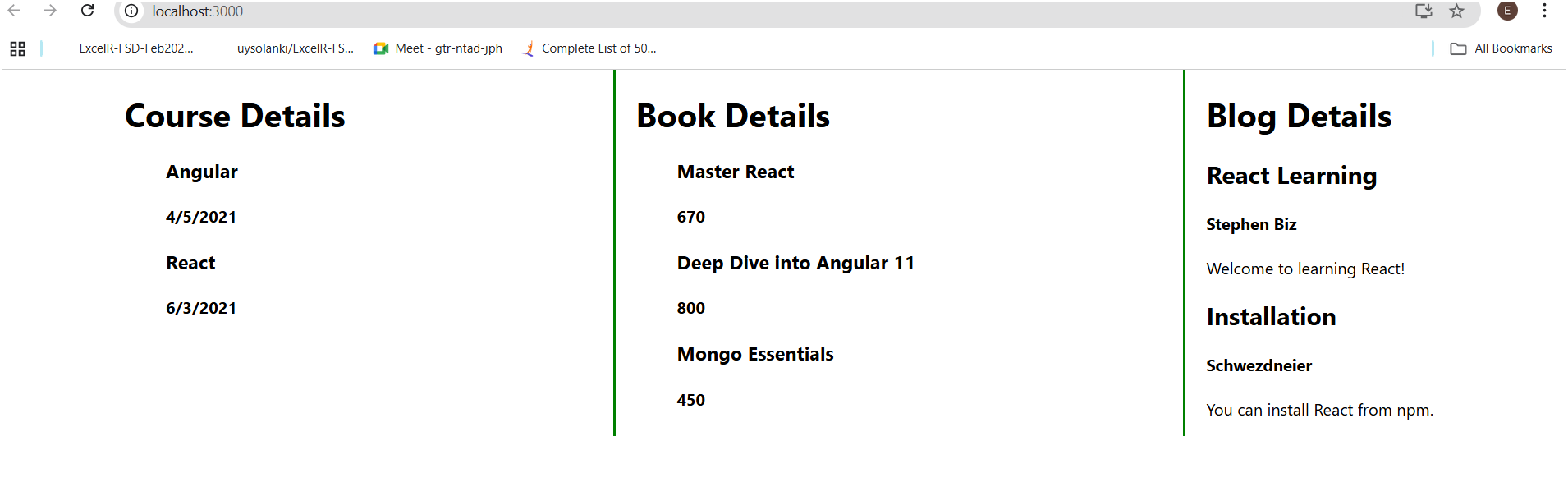
**Modify App.js to Combine All**

**Replace all content in src/App.js with:**

**Run the App**

**npm start**

**Output:**

****