**WEEK-7 Handson**

**Module 8 – Single Page Application framework - React**

**Ex9: ReactJS-HOL**

**Key Features of ES6**

ES6, also known as ECMAScript 2015, introduced significant enhancements to the JavaScript language. Some of the most notable features include:

* New variable declarations: let and const provide more predictable, block-scoped variable handling.
* Arrow functions: A more concise syntax for writing function expressions.
* Classes: A simpler syntax for creating objects and implementing inheritance.
* Template literals: Enhanced string interpolation and multi-line string support.
* Destructuring assignment: Easily extract values from arrays and objects.
* Default parameters: Specify default values for function arguments.
* Rest and spread operators: Versatile syntax for working with arrays and objects.
* Modules: Native support for importing and exporting code between files.
* Promises: A more streamlined way to handle asynchronous operations.
* New data structures: Map and Set for more flexible data management.

**JavaScript let Explained**

The let keyword, introduced in ES6, allows for the declaration of block-scoped local variables, which can be optionally initialized to a value. This means a variable declared with let is only accessible within the block, statement, or expression where it is defined. This is a significant improvement over the function-scoped var keyword, leading to more predictable and manageable code, especially within loops and conditional statements. While let variables are hoisted to the top of their scope, they are not initialized, which prevents them from being accessed before their declaration.

**JavaScript const Explained**

The const keyword is used to declare block-scoped local variables that must be initialized at the time of declaration and cannot be reassigned a new value. This creates an immutable binding to a value. It's important to note that for objects and arrays declared with const, the contents (properties or elements) can still be modified; however, the variable itself cannot be reassigned to a new object or array.

**Differences Between var and let**

The primary distinction between var and let lies in their scoping rules.

|  |  |  |
| --- | --- | --- |
| **Feature** | **var** | **let** |
| **Scope** | Function-scoped. | Block-scoped ({}). |
| **Hoisting** | Hoisted and initialized with undefined. | Hoisted but not initialized (in a "temporal dead zone"). |
| **Redeclaration** | Can be redeclared within the same scope. | Cannot be redeclared within the same scope. |
| **Global Object** | When declared globally, it becomes a property of the global object. | Does not add properties to the global object. |

**ES6 Class Fundamentals**

ES6 classes provide a more straightforward and cleaner syntax for creating objects and dealing with inheritance, serving as syntactic sugar over JavaScript's existing prototype-based inheritance.A class is defined using the class keyword and can include a constructor method for initializing an object's properties. You can also define methods within the class. An object is created from a class using the new keyword.

**ES6 Class Inheritance**

To create a class that inherits from a parent class, the extends keyword is used. The child class gains access to all the methods of the parent class. The super keyword can be used within a child class's constructor to call the parent class's constructor and access its properties and methods. This simplifies the process of creating subclasses and reusing code.

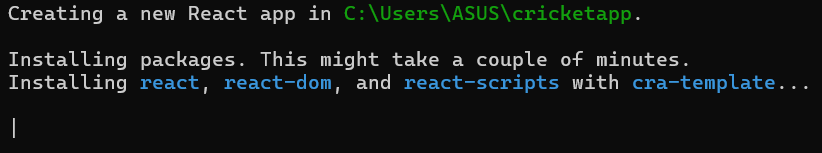
**ES6 Arrow Functions**

Arrow functions offer a more concise syntax for writing function expressions. They are defined using the => (fat arrow) notation and are always anonymous. A key characteristic of arrow functions is that they do not have their own this context; instead, they inherit this from the surrounding lexical scope. For single-expression functions, the return keyword is implicit.

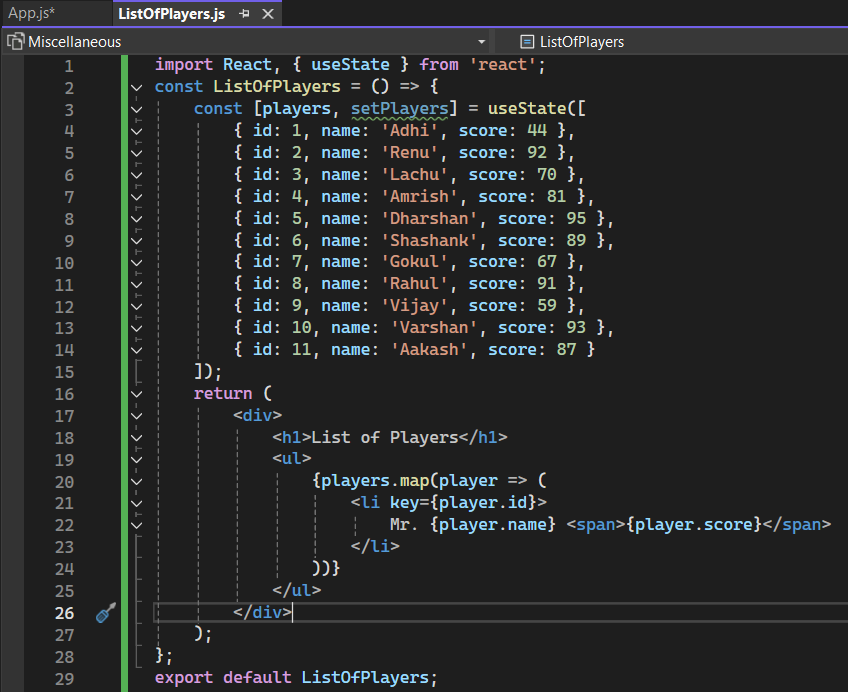
**Set and Map** are two new data structures introduced in ES6 for storing collections of data.

* Set(): A Set object allows you to store a collection of unique values of any type. Duplicate values are automatically ignored. You can add values to a set using the add() method.
* Map(): A Map object holds key-value pairs and remembers the original insertion order of the keys. Unlike regular objects, the keys in a Map can be any datatype.

Step1: create “cricketapp” using “npx create-react-app cricketapp” in cmd

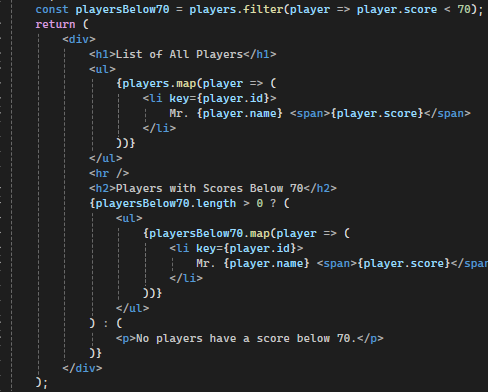


Step2: create “components” folder inside src and add “ListOfPlayers.js” inside it with below code to display List of Players



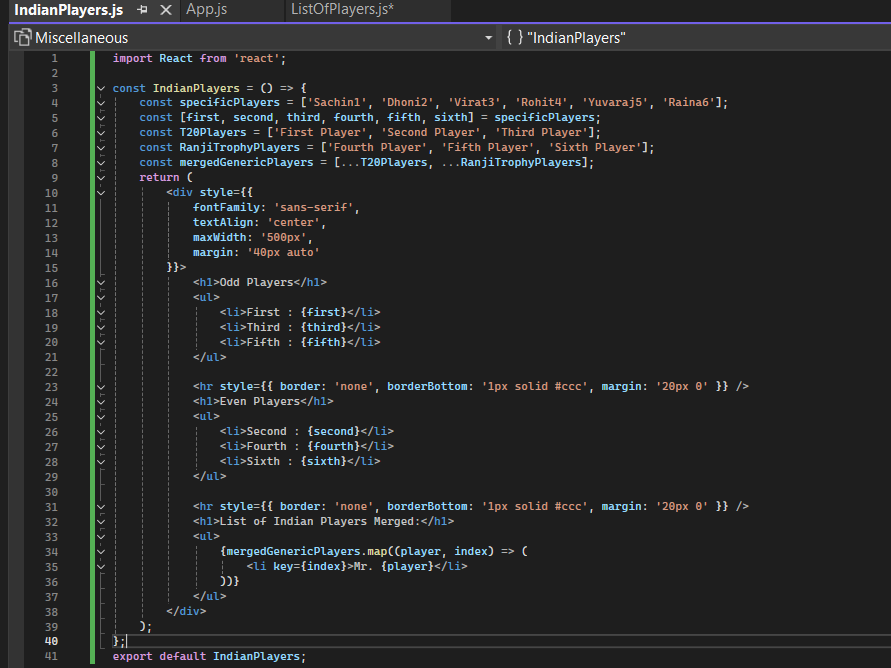


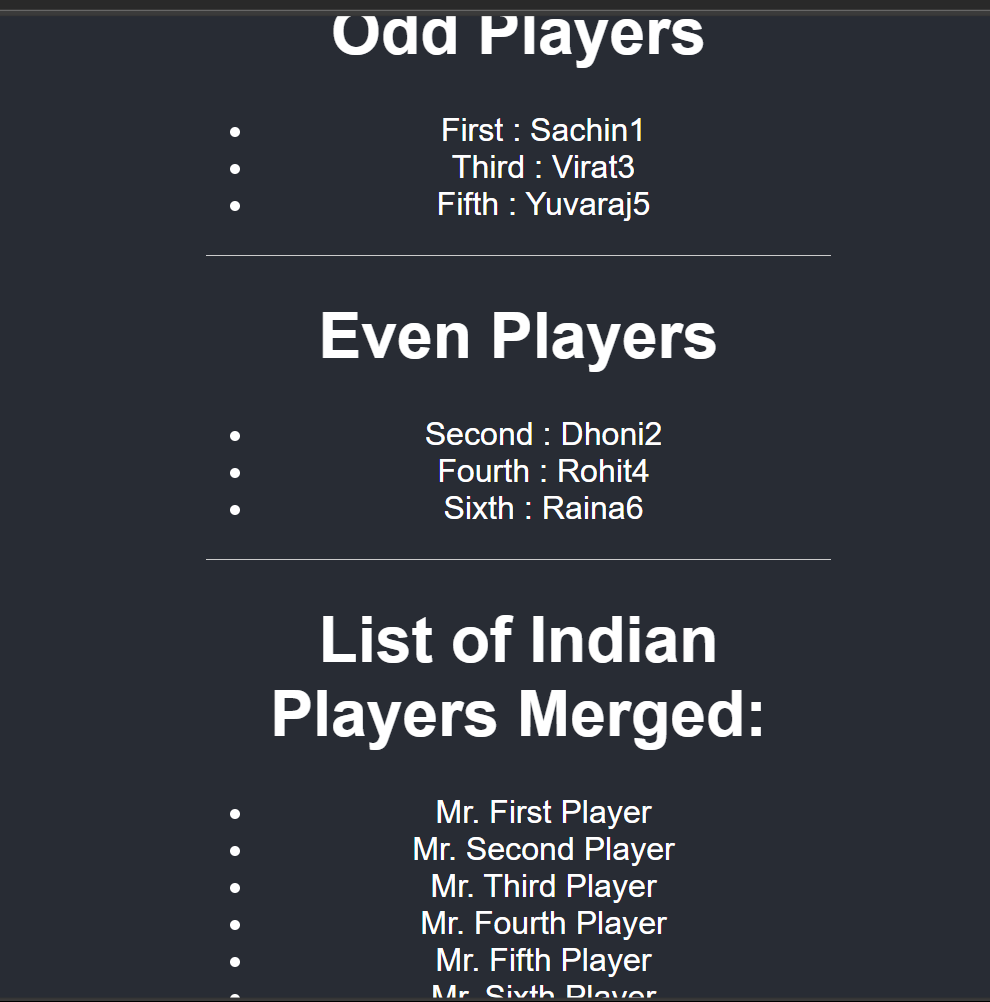
Step3: Add filters for fetching data in “ListOfPlayers.js”

****

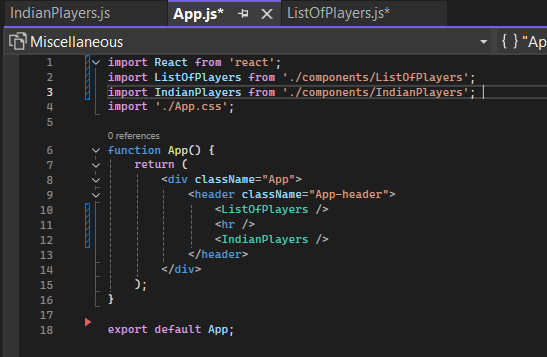
****

Step4: create “IndianPlayers.js” inside components with below code





Step5: Update “App.js” to run those codes



**Ex10: ReactJS-HOL**

**JSX**

JSX stands for JavaScript XML. It is a syntax extension for JavaScript that allows you to write HTML-like code directly within a JavaScript file. While it looks like HTML, it is not native to browsers and must be transpiled (converted) into regular JavaScript function calls before it can be executed. JSX makes writing React components more intuitive and readable by combining markup and logic in one place.

**ECMA Script**

ECMAScript is a scripting language specification standardized by Ecma International. It serves as the official standard on which JavaScript is based. Each new version of ECMAScript (like ES6, ES2022) introduces new features and syntax to the language. This standardization ensures that JavaScript code behaves consistently across different web browsers and environments.

**React.createElement()**

React.createElement() is the core function that React uses to create React elements. When you write JSX, a transpiler like Babel converts it into React.createElement() calls. The function typically takes three arguments: the element *type* (e.g., 'div', or a component), *props* (an object of attributes), and *children* (the content inside the element). These function calls produce JavaScript objects that describe your UI, which React then uses to build the DOM.

**Create React nodes with JSX**

You create React nodes with JSX by writing HTML-like syntax directly in your JavaScript files. A React node can be a simple HTML tag or a custom React component. For a component to be valid, it must return a single root element; if you need to return multiple elements, you must wrap them in a containing tag like a <div> or a React Fragment.

**JSX to DOM**

To render JSX to the browser's DOM, you use the react-dom library. In modern React (version 18+), you first create a "root" attached to a specific DOM element (like a div with an id of 'root'). Then, you call the render() method on that root, passing your JSX element or component as the argument.

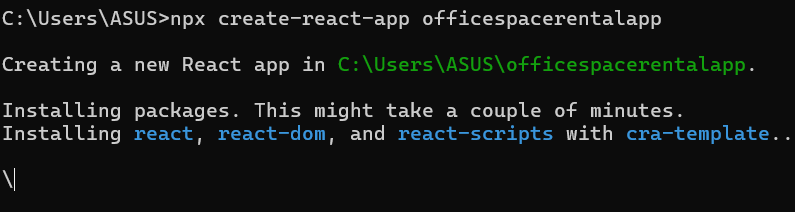
**JavaScript expressions in JSX**

You can embed any valid JavaScript expression inside JSX by wrapping it in curly braces {}. This allows you to insert dynamic content, such as variables, function calls, or arithmetic operations, directly into your markup.

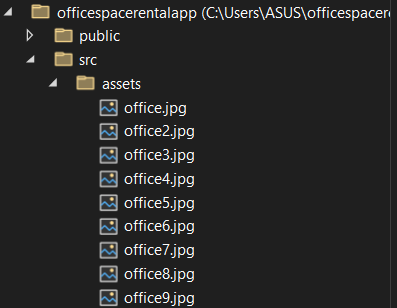
**Inline CSS in JSX**

Inline CSS is applied to JSX elements using the style attribute. Instead of a string, you pass a JavaScript object. The properties of this object must be camelCased (e.g., fontSize instead of font-size), and the values are typically strings.

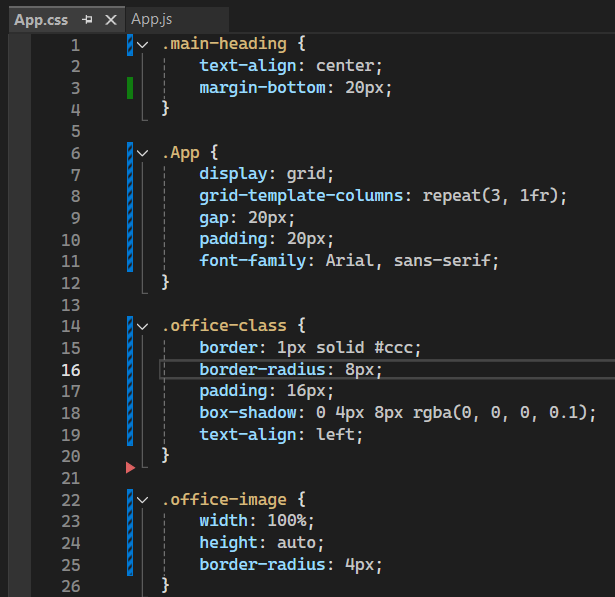
Step1: Create a React Application named “officespacerentalapp” which uses React JSX to create elements, attributes and renders DOM to display the page.



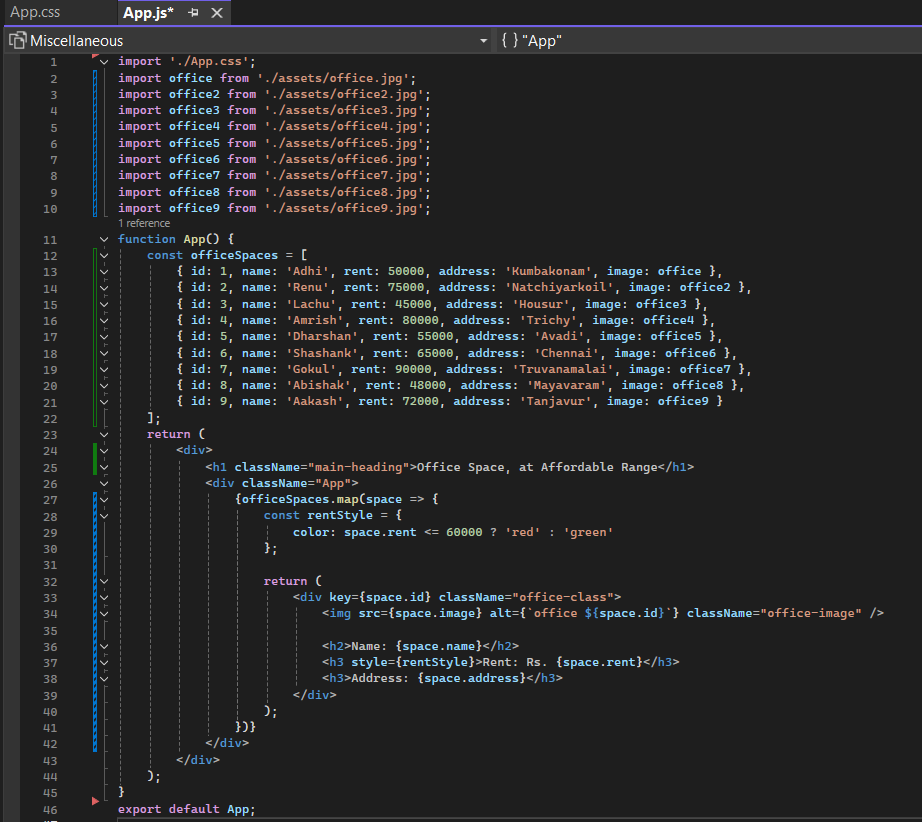
Step2: Create “assets” folder inside src and put a image inside it



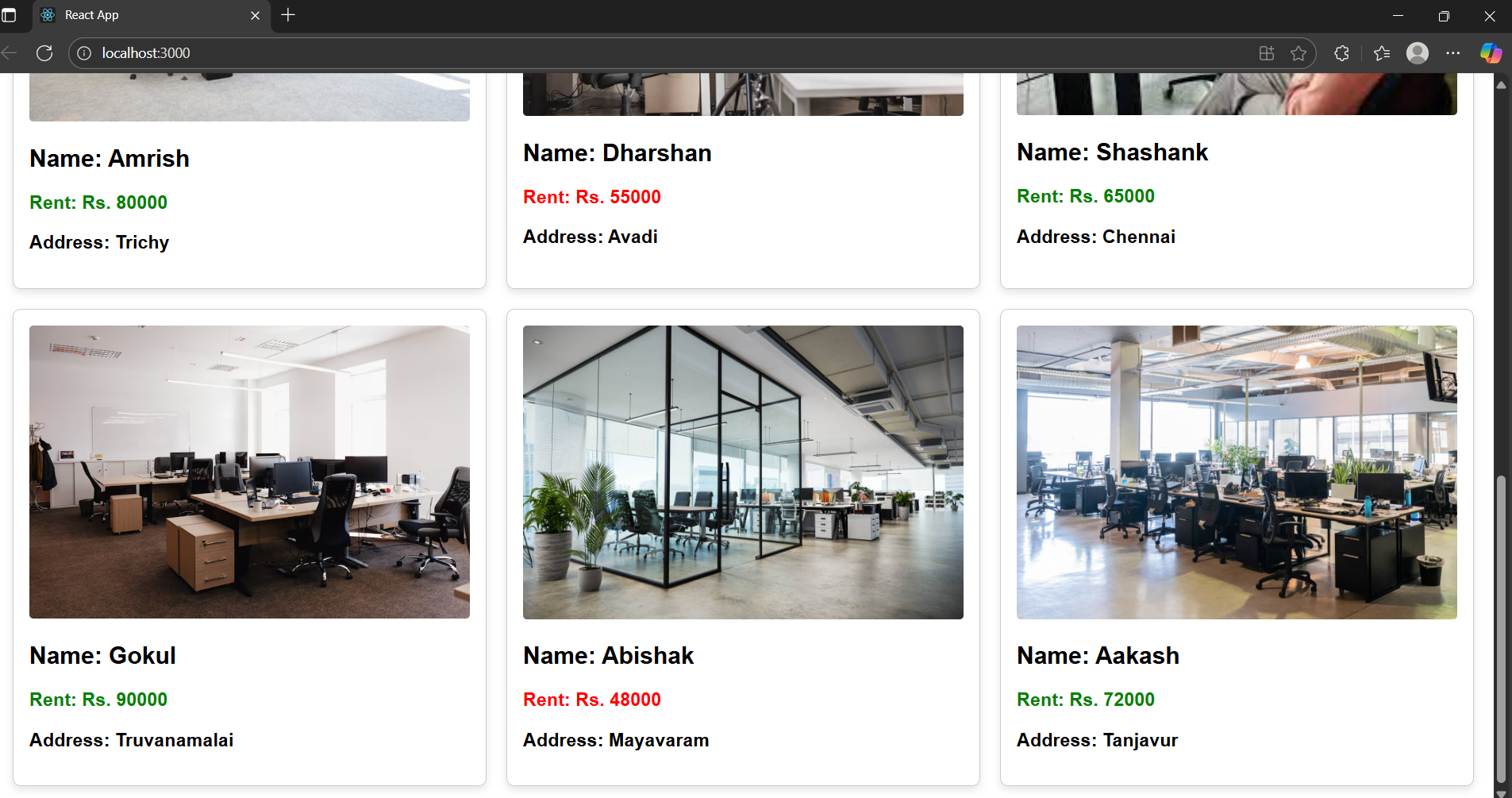
Step3: Update your App.css to add style clsses to image



Step4:Update App.js to import all images and displaying with the color of the Rent in Red if it’s below 60000 and in Green if it’s above 60000.



Step5: run “nmp start” in terminal



**Ex11: ReactJS-HOL**

**React events**

React events are actions that a user performs on a web page, such as a click, a key press, or a mouse hover. React has its own event handling system which is a wrapper around the browser's native event system. This wrapper is designed to make event handling consistent and reliable across all browsers, normalizing their differences.

**Event handlers**

An event handler is a function that is written to respond to a specific event. In React, you provide an event handler as a prop to a JSX element. When the user triggers the event (e.g., clicks a button), React calls the corresponding event handler function, allowing you to execute code in response to the user's action.

**Synthetic event**

A SyntheticEvent is a cross-browser wrapper created by React around the browser's native event object. It provides a consistent API, meaning that event properties like e.target.value will work the same way in all major browsers without you having to write special code. This abstraction layer smooths out browser inconsistencies and improves performance by using event pooling.

**React event naming convention**

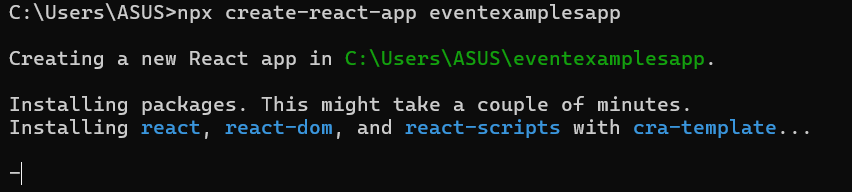
React follows a specific naming convention for event handler props that is both consistent and easy to remember:

* **camelCase:** The event name is written in camelCase.
* **"on" Prefix:** The event prop always starts with the prefix on.

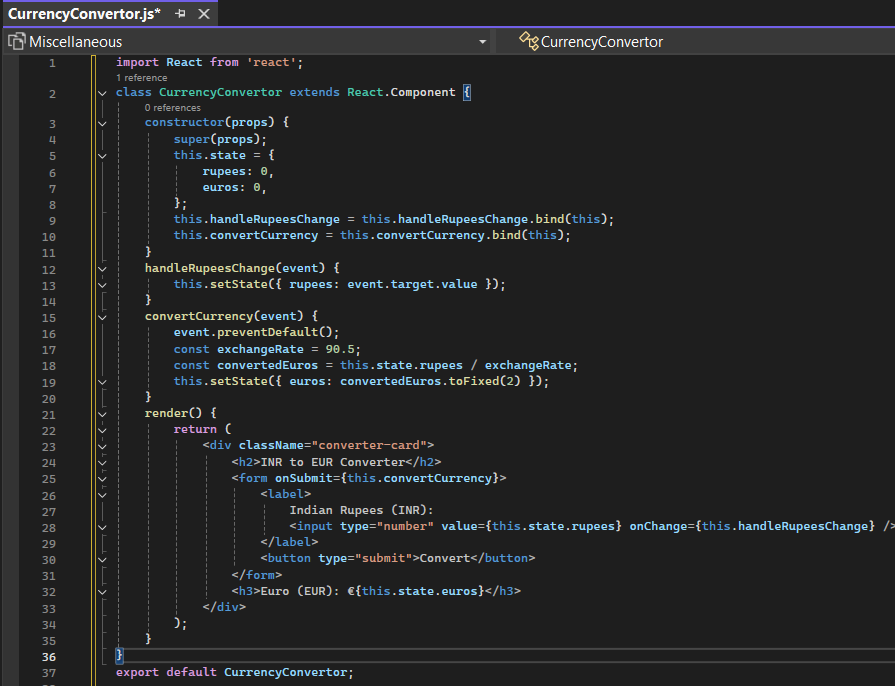
For example:

* A click event is handled by the onClick prop.
* A change event (like in an input field) is handled by the onChange prop.
* A form submission is handled by the onSubmit prop.
* A mouse-over event is handled by the onMouseOver prop.

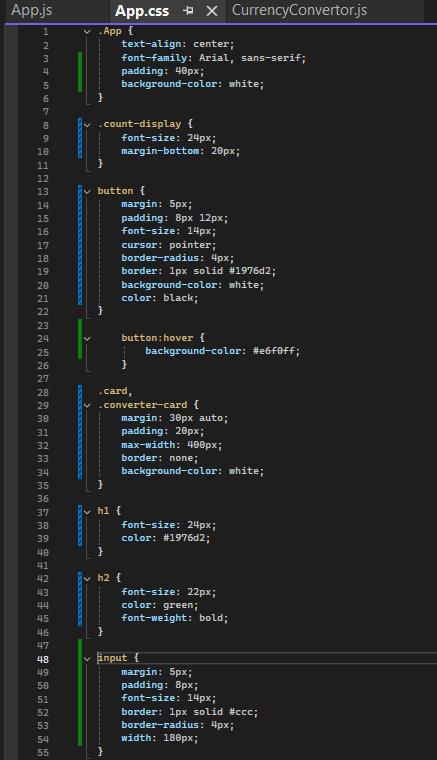
Step1: create “eventexxamplesapp” using below commend in cmd



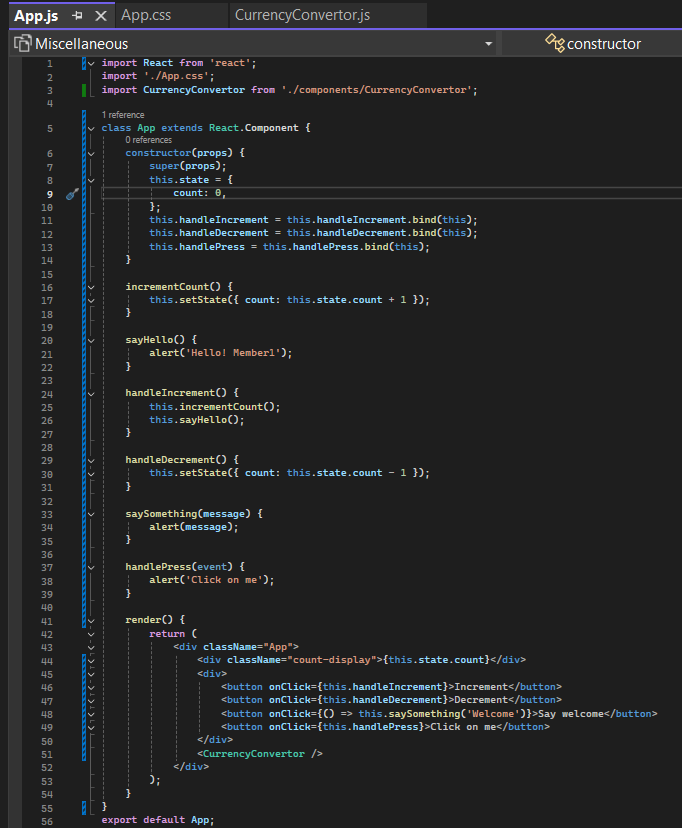
Step2: Create new folder “components” inside src and add “CurrencyConvertor.js” with below code to convert the Indian Rupees to Euro when the Convert button is clicked.



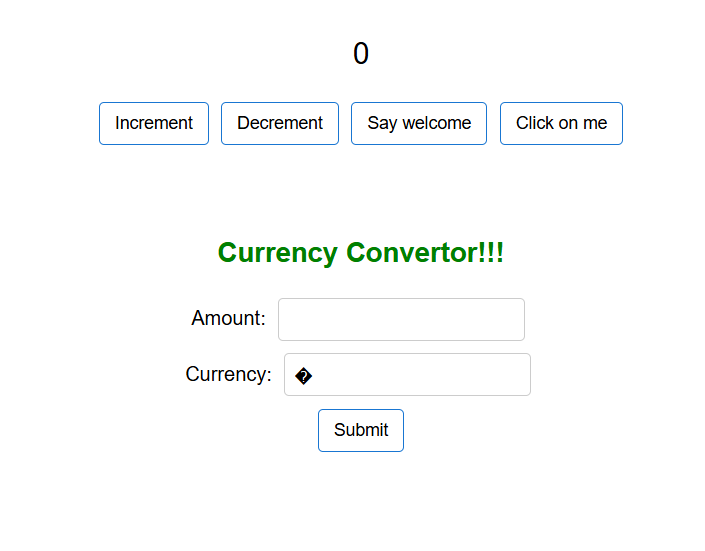
Step3: update App.cs with below code to add style to buttons and inputs



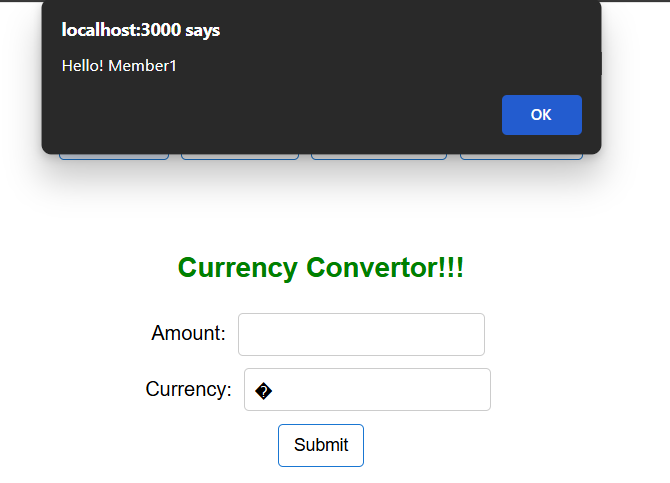
Step4: Update App.js by below



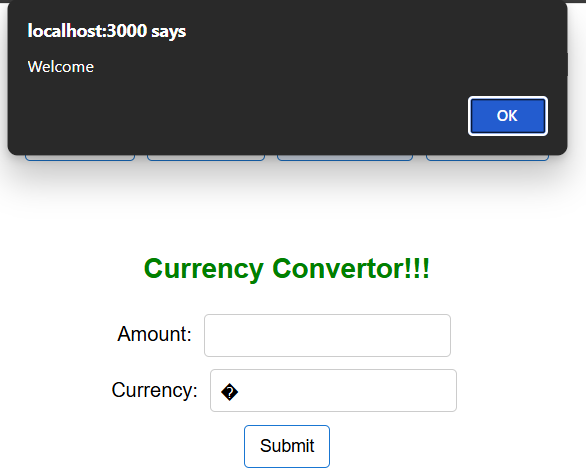
Step5: run “npm start” in terminal to start application



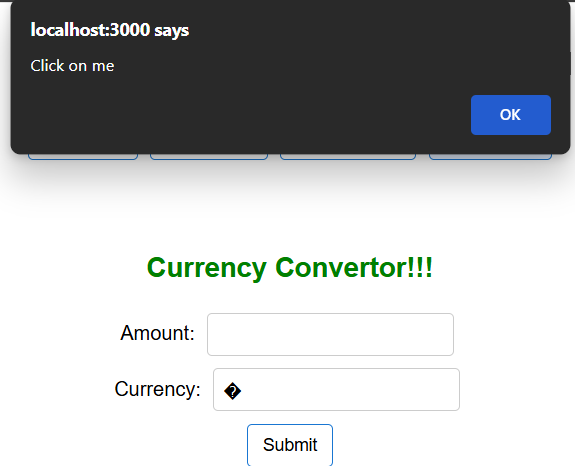
**When increment pressed**



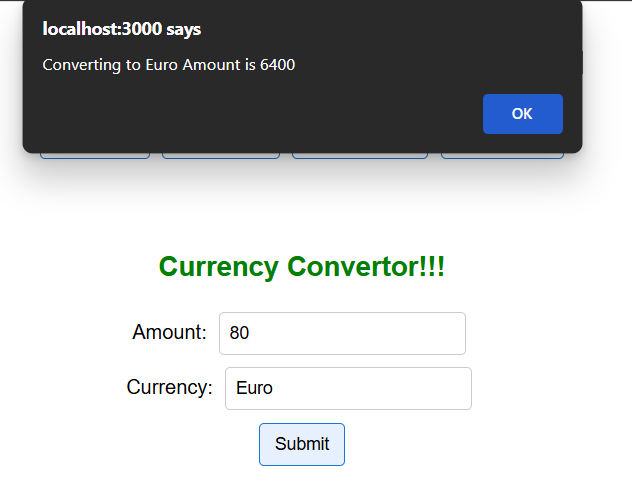
**While “say welcome” pressed**



**While “click on me ” pressed**



**Converting 80 into Euro**



**Ex12: ReactJS-HOL**

**Conditional rendering in React**

Conditional rendering in React is the process of displaying different UI elements or components based on certain conditions or the current state of the application. Just like with JavaScript, you can use standard operators like if, else, and the ternary operator (condition ? true : false) inside your components to decide what should be rendered. This allows you to create dynamic and responsive user interfaces that change based on user interaction or data.

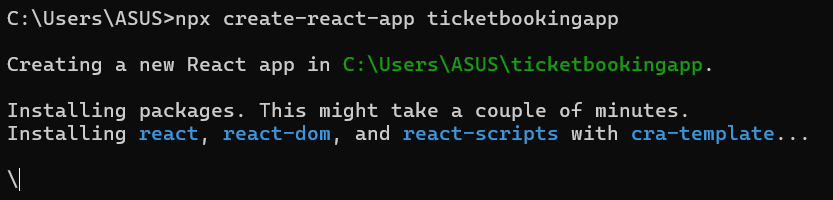
**Element variables**

Element variables are regular JavaScript variables used to hold JSX elements. They are useful for conditionally rendering a part of a component while the rest of the output remains the same. You can use if statements to assign different JSX to the variable based on some condition, and then include that variable inside your final JSX output.

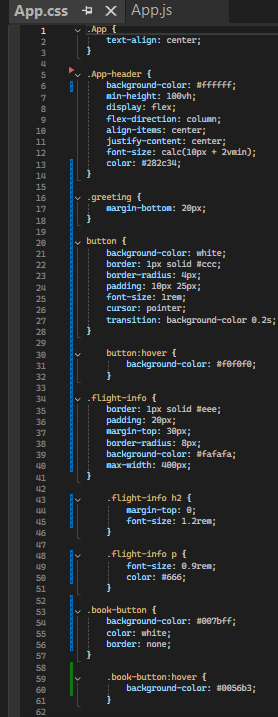
**Prevent components from rendering**

To prevent a component from rendering anything to the DOM, you can have its render method return null. When React sees that a component has returned null, it simply skips rendering it and any of its child components. This is a common pattern for hiding components based on a specific condition without unmounting them completely or cluttering the DOM with empty elements.

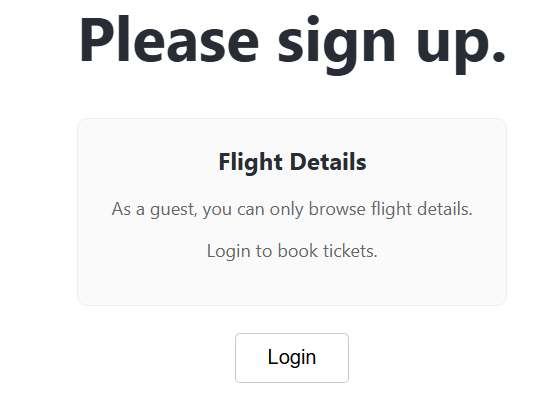
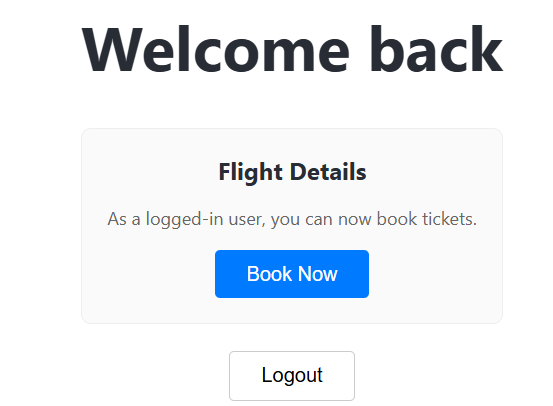
Step1: Create “ticketbookingapp” by below cmd



Step2: Update “App.cs” and “App.js”

Step3: Output



**Ex13: ReactJS-HOL**

**Various ways of conditional rendering**

Conditional rendering allows you to show different UI elements based on specific conditions. The most common ways to do this are:

1. **if/else Statement:** Use an if statement with element variables to decide which component or element to assign to a variable before the return statement.
2. **Ternary Operator (? :):** A concise, inline way to choose between two options directly inside your JSX. It's perfect for simple if/else logic.

{isLoggedIn ? <UserPage /> : <GuestPage />}

1. **Logical && Operator (Short-Circuit And):** A clean way to render something *only if* a condition is true. If the condition is false, it renders nothing.

{showWarning && <WarningBanner />}

1. **switch Statement:** Useful for when you have several different UI options based on a single value.

**Render multiple components**

To render multiple components, you can simply place them one after another in your JSX. However, a component's return statement must always have a **single root element**. If you need to return a list of components, you must wrap them in a container, such as:

* A <div>.
* A <React.Fragment>.
* The shorthand fragment syntax <> ... </>.

**List component**

A "list component" is a component that dynamically renders a list of items by iterating over an array of data. It typically uses an array method like map() to transform each item in the data array into a JSX element (like an <li> or a custom component).

**Keys in React applications**

A key is a special string attribute you must include when creating lists of elements in React. Keys help React identify which items have changed, been added, or been removed. By giving each element a stable and unique key, you allow React to efficiently update the UI without re-rendering the entire list.

* Keys must be **unique among siblings** in the same list.
* The best keys are unique and stable IDs from your data (e.g., user.id).
* Using the array index as a key is not recommended if the list can be reordered, added to, or filtered, as it can lead to bugs and performance issues.

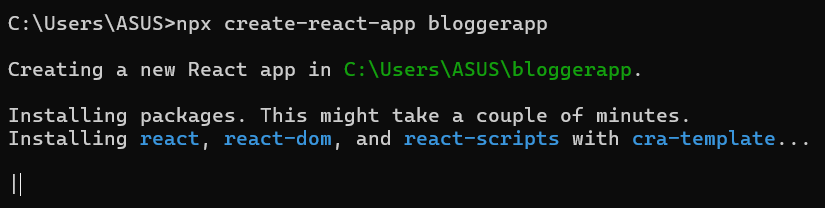
**Extract components with keys**

When you create a list component, the key should always be assigned to the outermost element returned by the .map() loop. If you extract the list item into its own component, the key should be placed on the custom component tag itself, not inside the component's own JSX.

**Map() function**

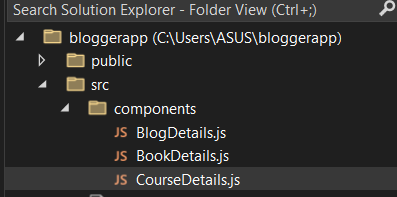
In React, the JavaScript map() function is the standard way to create lists of elements. You call .map() on an array of data, and for each item in the array, the function you provide returns a React element. The map() function then collects all these elements into a new array, which React can render.

Step1: Create project

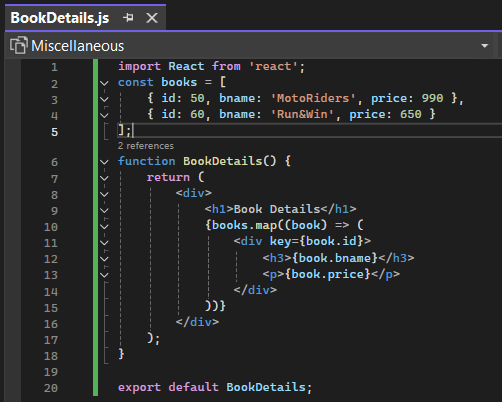


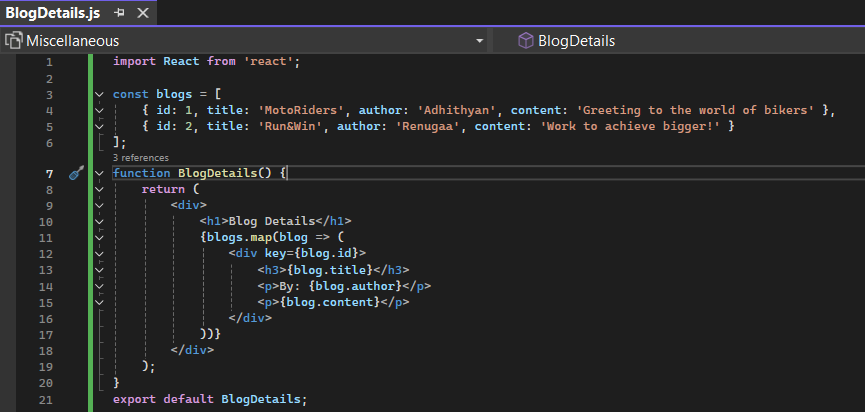
Step2: create folder “components” inside src and add “BookDetails.js”,”BlogDetails.js” &

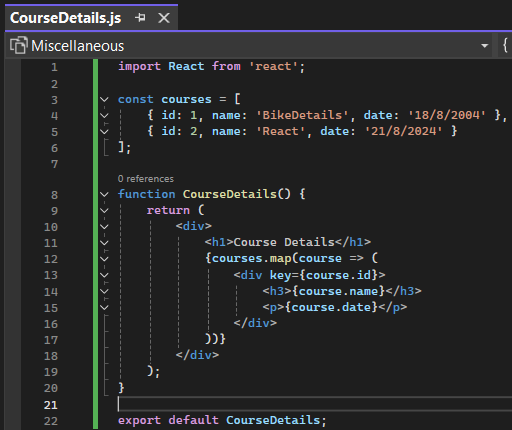
“CourseDetails.js”



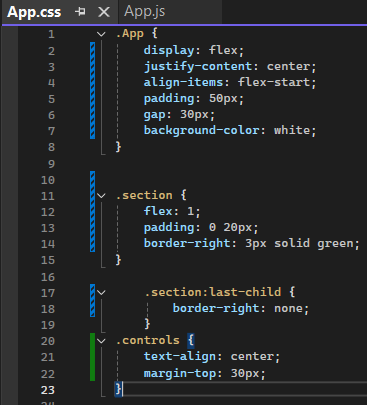
Step3:Type the below code inside “BookDetails.js”,”BlogDetails.js” & “CourseDetails.js”



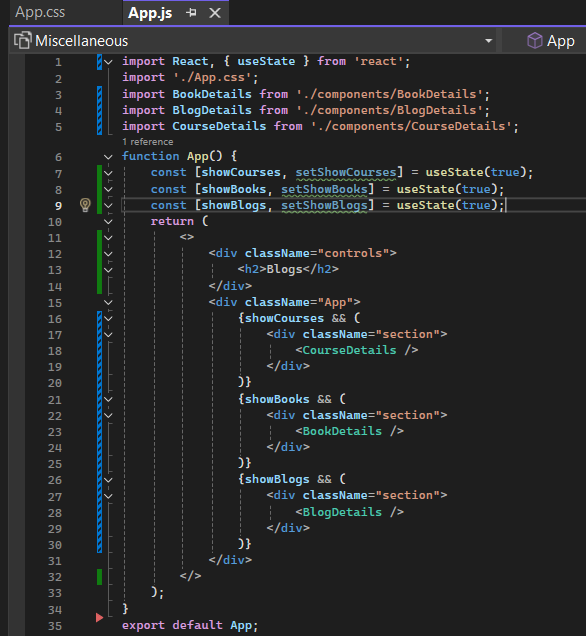




Step4:Update “App.cs” by below



Step5: Update “App.js”



Step6: run “npm start” in terminal

