**Objectives**

**List the features of ES6**

ES6 (ECMAScript 2015) introduced major enhancements to JavaScript, including:

* let and const for block-scoped variables
* Arrow functions
* Classes and inheritance
* Template literals
* Default and rest parameters
* Destructuring (arrays, objects)
* Spread operator
* Promises
* Modules (import/export)
* Collections: Map, Set

**Explain JavaScript let**

* let declares a block-scoped variable (only accessible inside its {} block).
* Unlike var, let does not hoist to the top of the function scope with initialization.
* Helps prevent accidental variable overwrites.

Example:

if (true) {

let x = 10; // x exists only here

}

// console.log(x); // Error: x is not defined

**Identify the differences between var and let**

|  |  |  |
| --- | --- | --- |
| Feature | var | let |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Hoisted & initialized as undefined | Hoisted but uninitialized (TDZ) |
| Redeclaration | Allowed within same scope | Not allowed in same scope |

**Explain JavaScript const**

* Declares a block-scoped constant.
* Value must be assigned at declaration.
* The reference cannot be changed, but object properties can still be mutated.

Example:

const PI = 3.14;

// PI = 3.15; // Error

const obj = { a: 1 };

obj.a = 2; // Allowed: properties can change

**Explain ES6 class fundamentals**

* Classes provide a clearer, OOP-like syntax to create objects.
* constructor method is used for initialization.
* Methods defined inside a class are placed on its prototype.

Example:

class Person {

constructor(name) {

this.name = name;

}

greet() {

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

}

}

const p = new Person("Alice");

console.log(p.greet()); // Hello, Alice

**Explain ES6 class inheritance**

* Use extends to inherit from another class.
* Use super() to call the parent constructor.

Example:

class Animal {

constructor(name) {

this.name = name;

}

}

class Dog extends Animal {

bark() {

return `${this.name} says woof!`;

}

}

const d = new Dog("Buddy");

console.log(d.bark()); // Buddy says woof!

**Define ES6 arrow functions**

* Shorter syntax for functions: () => {}.
* Do not bind their own this, which means this refers to the outer scope.
* Great for callbacks and concise code.

Example:

const add = (a, b) => a + b;

console.log(add(3, 4)); // 7

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

**Set**

* Collection of **unique values**.
* Fast lookups to check if an element exists.
* No duplicate entries allowed.

Example:

const set = new Set();

set.add(1);

set.add(2);

set.add(1); // ignored

console.log(set); // Set {1, 2}

**Map**

* Key-value pairs collection with **any type of key** (objects, functions, etc.).
* Maintains insertion order.

Example:

const map = new Map();

map.set('a', 1);

map.set('b', 2);

console.log(map.get('a')); // 1

**Code :**

**--App.js**

import React from 'react';

import ListofPlayers from './components/ListofPlayers';

import IndianPlayers from './components/IndianPlayers';

function App() {

  const flag = false; // Toggle between true/false to test both components

  return (

    <div className="App">

      <h1>React App</h1>

      {flag ? <ListofPlayers /> : <IndianPlayers />}

    </div>

  );

}

export default App;

**--ListofPlayers.js**

import React from 'react';

const ListofPlayers = () => {

  const players = [

    { name: 'Mr. Jack', score: 50 },

    { name: 'Mr. Michael', score: 70 },

    { name: 'Mr. John', score: 40 },

    { name: 'Mr. Ann', score: 61 },

    { name: 'Mr. Elisabeth', score: 61 },

    { name: 'Mr. Sachin', score: 95 },

    { name: 'Mr. Dhoni', score: 100 },

    { name: 'Mr. Virat', score: 84 },

    { name: 'Mr. Jadeja', score: 64 },

    { name: 'Mr. Raina', score: 75 },

    { name: 'Mr. Rohit', score: 80 },

  ];

  // All Players List

  const allPlayers = players.map((player, index) => (

    <li key={index}>{player.name} {player.score}</li>

  ));

  // Players having scores less than 70

  const lowScorers = players

    .filter(player => player.score < 70)

    .map((player, index) => (

      <li key={index}>{player.name} {player.score}</li>

    ));

  return (

    <div>

      <h2>List of Players</h2>

      <ul>{allPlayers}</ul>

      <h2>List of Players having Scores Less than 70</h2>

      <ul>{lowScorers}</ul>

    </div>

  );

};

export default ListofPlayers;

**--IndianPlayers.js**

import React from 'react';

const IndianPlayers = () => {

  const oddPlayers = ['Sachin1', 'Virat3', 'Yuvraj5'];

  const evenPlayers = ['Dhoni2', 'Rohit4', 'Raina6'];

  // Destructuring for label

  const [first, third, fifth] = oddPlayers;

  const [second, fourth, sixth] = evenPlayers;

  // Merge T20 + Ranji Players

  const T20Players = ['Mr. First Player', 'Mr. Second Player'];

  const RanjiPlayers = ['Mr. Third Player', 'Mr. Fourth Player', 'Mr. Fifth Player'];

  const merged = [...T20Players, ...RanjiPlayers];

  return (

    <div>

      <h2>Odd Players</h2>

      <ul>

        <li>First: {first}</li>

        <li>Third: {third}</li>

        <li>Fifth: {fifth}</li>

      </ul>

      <h2>Even Players</h2>

      <ul>

        <li>Second: {second}</li>

        <li>Fourth: {fourth}</li>

        <li>Sixth: {sixth}</li>

      </ul>

      <h2>List of Indian Players Merged:</h2>

      <ul>

        {merged.map((player, index) => (

          <li key={index}>{player}</li>

        ))}

      </ul>

    </div>

  );

};

export default IndianPlayers;

**--index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

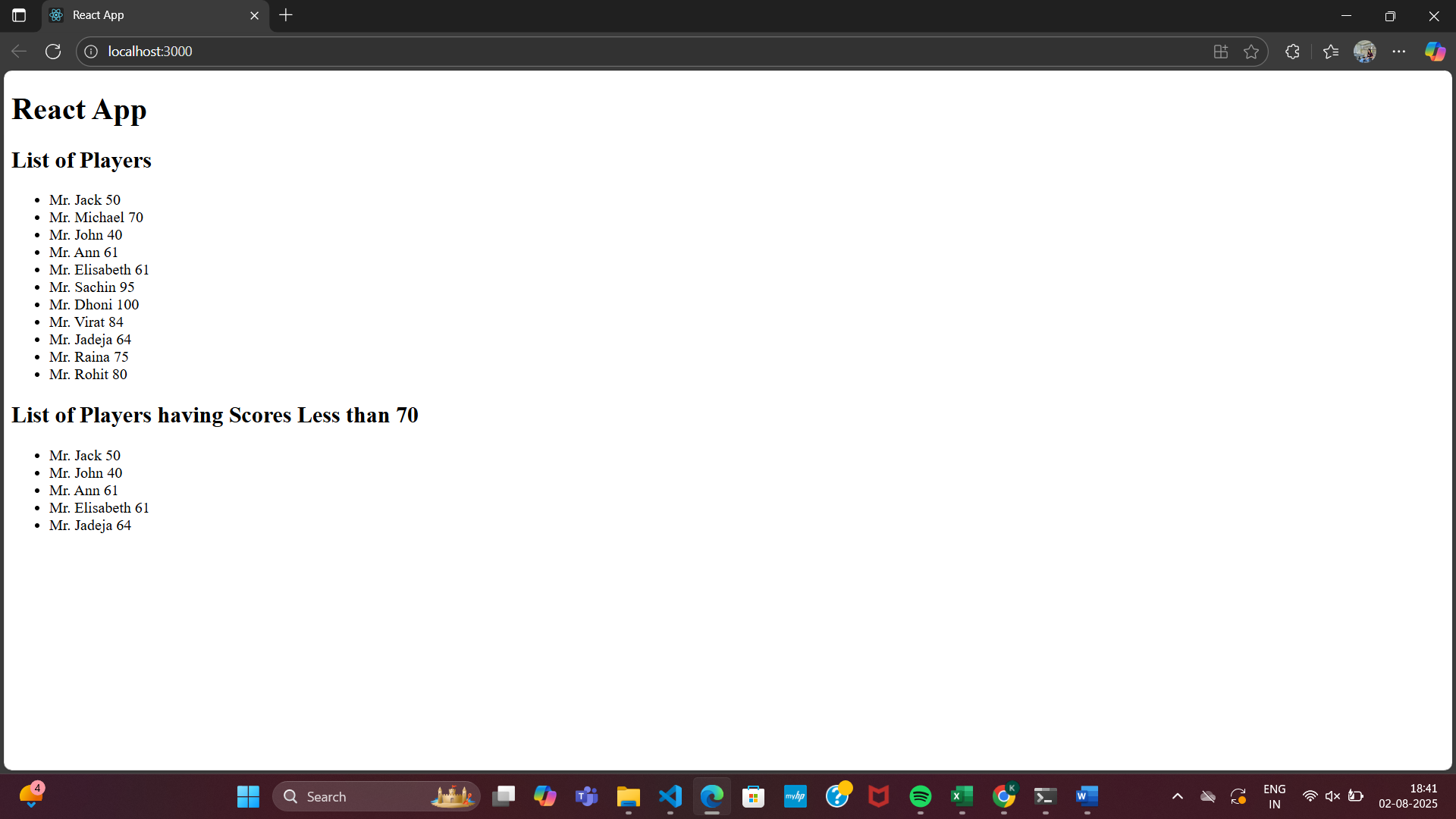
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);

**Output :**

**When flag = true**



**When flag = false**

