**WEEK – 7(HandsOn)**

**React**

**9. ReactJS-HOL**

**1. List the features of ES6**

* Block-scoped variables: let and const
* Arrow functions: Concise function syntax (() => {})
* Classes: Native class syntax for object-oriented code
* Template literals: String interpolation with backticks and ${}
* Destructuring assignment: Easy extraction from arrays/objects
* Default, rest, and spread operators:
  + Default function parameters
  + Rest (...args) to collect
  + Spread (...array) to expand
* Modules: import and export syntax for modular JS
* Promises: Simplified asynchronous programming
* Enhanced object literals: Shorter syntax/methods
* for...of loops: Iterating over collections
* Map and Set data structures
* Symbol type for unique object property keys

**2. Explain JavaScript let**

* let is used to declare variables that are **block-scoped** (only accessible within the block they are declared in, e.g., inside {}).
* Variables declared with let **can be updated**, but **cannot be redeclared** in the same scope.
* Helps prevent bugs from unintentional variable hoisting and scope issues.

**Example:**

let x = 10;

if (true) {

let x = 20;

// This x is different from the x above

}

**3. Identify the differences between var and let**

| **Feature** | **var** | **let** |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped |
| Redeclaration | Can be redeclared in same scope | Cannot redeclare in same scope |
| Hoisting | Hoisted and initialized as undefined | Hoisted but not initialized |
| Reassignment | Allowed | Allowed |

**Example:**

if(true) {

var a = 1; // available outside the block

let b = 2; // only available inside the block

}

console.log(a); // 1

console.log(b); // error: b is not defined

**4. Explain JavaScript const**

* const is used to declare block-scoped constants.
* Must be initialized at declaration.
* Cannot be reassigned. However, if the variable holds an object or array, the object’s properties or array’s elements can still be modified.

**Example:**

const PI = 3.14;

PI = 3.1416; // Error

const arr = [1,2,3];

arr.push(4); // Allowed

**5. Explain ES6 class fundamentals**

* ES6 introduced a native class syntax for creating objects and handling inheritance, making OOP in JS more intuitive.
* Classes include:
  + **Constructor:** Special method for creating and initializing objects.
  + **Methods:** Functions attached to the class.

**Example:**

class Person {

constructor(name) {

this.name = name;

}

greet() {

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

}

}

**6. Explain ES6 class inheritance**

* ES6 allows classes to **inherit** from other classes using the extends keyword.
* The super keyword calls the parent class’s constructor or methods.

**Example:**

class Person {

constructor(name) { this.name = name; }

}

class Student extends Person {

constructor(name, roll) {

super(name); // Calls Person's constructor

this.roll = roll;

}

}

**7. Define ES6 arrow functions**

* A shorter syntax for writing functions.
* **Does not bind its own this** (inherits from the parent scope).
* Cannot be used as constructors.

**Example:**

// Traditional

function add(a, b) {

return a + b;

}

// Arrow

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

**8. Identify set(), map()**

**Set**: Collection of unique values (no duplicates).

Methods: .add(), .has(), .delete()

* + Example:

const mySet = new Set([1,2,3]);

mySet.add(4); // Set {1, 2, 3, 4}

**Map**: Collection of key-value pairs, where keys can be any type.

* Methods: .set(key, value), .get(key), .has(key)
* Example:

const myMap = new Map();

myMap.set('a', 1);

myMap.get('a'); // 1

CODE:

import React, { useState } from "react";

function App() {

  const [showListOfPlayers, setShowListOfPlayers] = useState(true);

  // Sample players array as per the image

  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 },

  ];

  // Filter players with score less than 70

  const lowScorers = players.filter(player => player.score < 70);

  return (

    <div>

      <button onClick={() => setShowListOfPlayers(!showListOfPlayers)}>

        Switch View

      </button>

      {showListOfPlayers && (

        <>

          <h1>List of Players</h1>

          <ul>

            {players.map((player, idx) => (

              <li key={idx}>

                {player.name} {player.score}

              </li>

            ))}

          </ul>

          <hr />

          <h1>List of Players having Scores Less than 70</h1>

          <ul>

            {lowScorers.map((player, idx) => (

              <li key={idx}>

                {player.name} {player.score}

              </li>

            ))}

          </ul>

        </>

      )}

      {/\* Here you can show the alternate view (when flag is false) \*/}

      {/\* {!showListOfPlayers && (

        <AlternateOutput />

      )} \*/}

    </div>

  );

}

export default App;

**src/App.js:**

// src/App.js

import React from 'react';

import ListOfPlayers from './ListOfPlayers';

import IndianPlayers from './IndianPlayers';

function App() {

  const flag = true; // change this to false to test conditional rendering

  return (

    <div className="App">

      <h1>Welcome to Cricket App</h1>

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

    </div>

  );

}

export default App;

**src/App.js:**

import React, { useState } from "react";

function App() {

const [showListOfPlayers, setShowListOfPlayers] = useState(false);

const indianPlayers = [

"Sachin1", // 0 (First)

"Dhoni2", // 1 (Second)

"Virat3", // 2 (Third)

"Rohit4", // 3 (Fourth)

"Yuvaraj5", // 4 (Fifth)

"Raina6", // 5 (Sixth)

];

// Odd (index 0,2,4): 0, 2, 4

const oddPlayers = [

{ position: "First", name: indianPlayers[0] },

{ position: "Third", name: indianPlayers[2] },

{ position: "Fifth", name: indianPlayers[4] },

];

// Even (index 1,3,5): 1, 3, 5

const evenPlayers = [

{ position: "Second", name: indianPlayers[1] },

{ position: "Fourth", name: indianPlayers[3] },

{ position: "Sixth", name: indianPlayers[5] },

];

// Merged list from two arrays

const T20players = [

"Mr. First Player",

"Mr. Third Player",

"Mr. Fifth Player",

];

const RanjiPlayers = [

"Mr. Second Player",

"Mr. Fourth Player",

"Mr. Sixth Player",

];

const mergedPlayers = [...T20players, ...RanjiPlayers];

return (

<div>

<button onClick={() => setShowListOfPlayers(!showListOfPlayers)}>

Switch View

</button>

{!showListOfPlayers && (

<>

<h1>Odd Players</h1>

<ul>

{oddPlayers.map((player, idx) => (

<li key={idx}>

{player.position} : {player.name}

</li>

))}

</ul>

<hr />

<h1>Even Players</h1>

<ul>

{evenPlayers.map((player, idx) => (

<li key={idx}>

{player.position} : {player.name}

</li>

))}

</ul>

<hr />

<h1>List of Indian Players Merged:</h1>

<ul>

{mergedPlayers.map((player, idx) => (

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

))}

</ul>

</>

)}

{/\* The flag==true view goes here \*/}

</div>

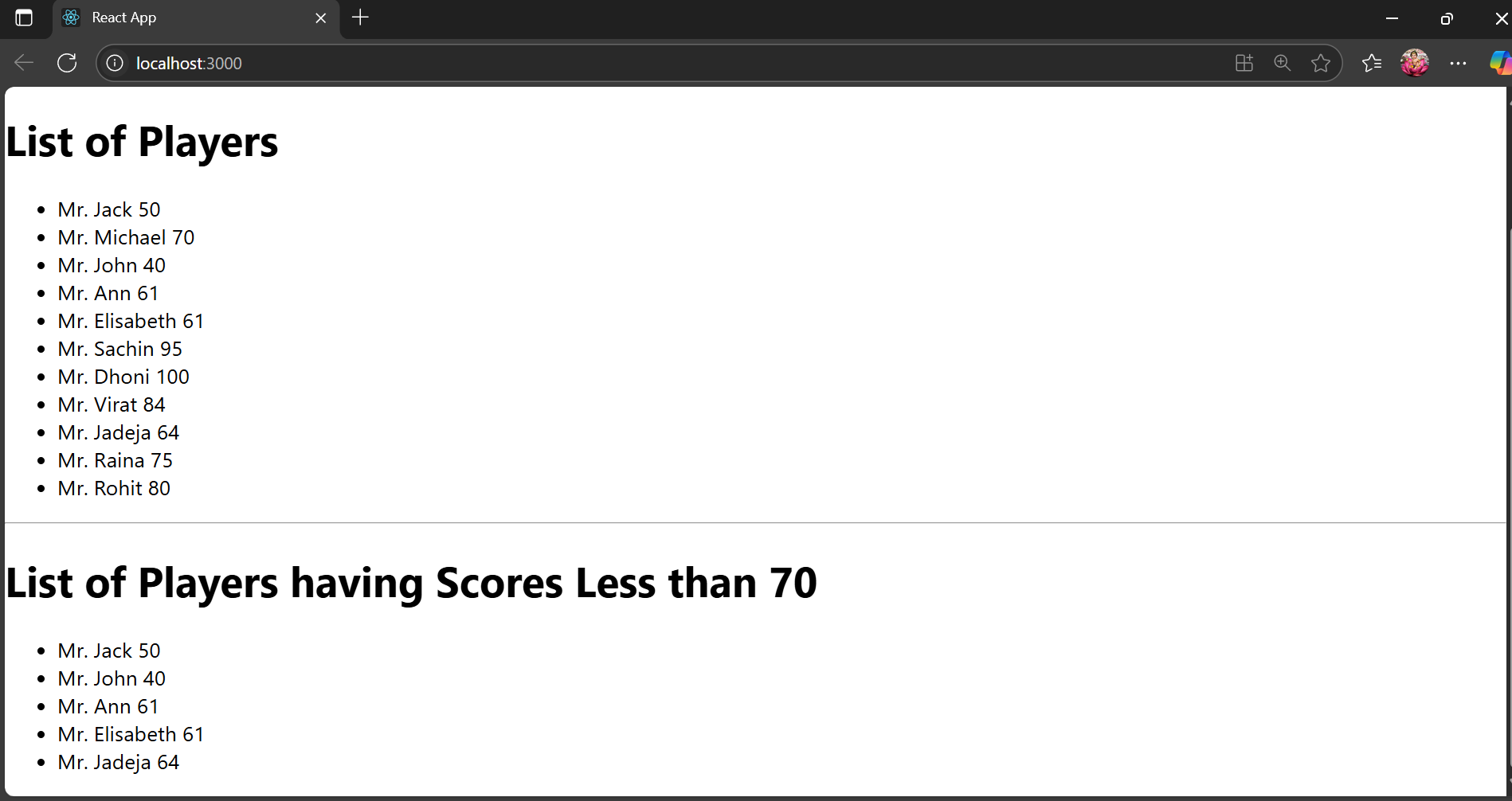
);

}

export default App;

**OUTPUT:**

When flag=true



When flag=false

