**List the features of ES6**

- let and const declarations  
- Arrow functions  
- Template literals  
- Default parameters  
- Destructuring assignment  
- Spread and Rest operators  
- Promises  
- Classes and Inheritance  
- Modules (import/export)  
- Enhanced object literals

**Explain JavaScript let**

The `let` keyword is used to declare block-scoped variables. It prevents redeclaration within the same scope and allows reassignment. Unlike `var`, `let` is not hoisted to the top of its scope in the same way.  
  
Example:  
let count = 10;  
count = 20;

**Identify the differences between var and let**

- `var` is function-scoped; `let` is block-scoped.  
- `var` gets hoisted and initialized as `undefined`; `let` is hoisted but not initialized.  
- `let` prevents redeclaration in the same scope.  
  
Example:  
function test() {  
 if (true) {  
 var x = 10;  
 let y = 20;  
 }  
 console.log(x);   
 console.log(y);   
}

**Explain JavaScript const**

`const` is used to declare constants. The value assigned to a `const` variable cannot be changed after initialization. It is block-scoped.  
  
Note: For objects and arrays, the reference is constant, but internal values can be mutated.  
  
Example:  
const PI = 3.14;  
PI = 3.15;   
  
const user = { name: "Alex" };  
user.name = "John";

**Explain ES6 class fundamentals**

ES6 introduced the `class` syntax for creating objects and handling inheritance more cleanly. Classes are syntactic sugar over JavaScript’s existing prototype-based inheritance.  
  
Example:  
class Person {  
 constructor(name) {  
 this.name = name;  
 }  
 greet() {  
 return `Hello, ${this.name}`;  
 }  
}

**Explain ES6 class inheritance**

ES6 supports class-based inheritance using `extends` and `super` keywords.  
  
Example:  
class Animal {  
 speak() {  
 return "Animal speaks";  
 }  
}  
  
class Dog extends Animal {  
 speak() {  
 return "Dog barks";  
 }  
}  
  
const d = new Dog();  
d.speak();

**Define ES6 arrow functions**

Arrow functions provide a shorter syntax for writing functions and do not bind their own `this` context.  
  
Example:  
const add = (a, b) => a + b;  
const greet = name => `Hello, ${name}`;

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

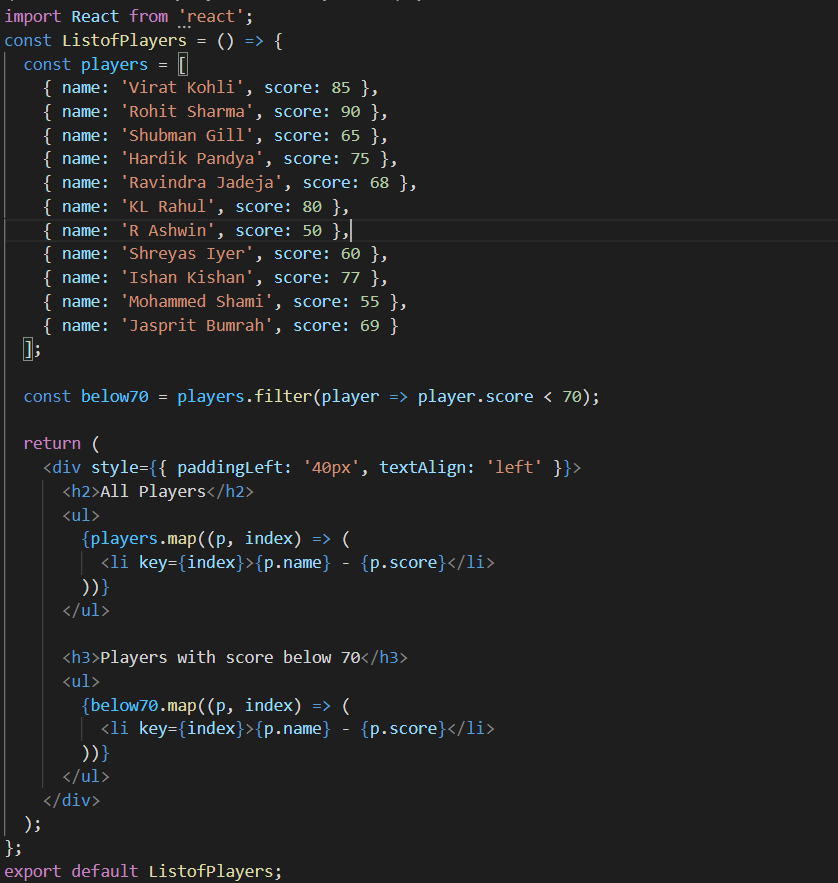
- Set: A collection of unique values.  
- Map: A collection of key-value pairs with keys of any type.  
  
Example:  
const s = new Set([1, 2, 2, 3]);   
  
const m = new Map();  
m.set('a', 1);  
m.set('b', 2);  
console.log(m.get('a'));

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

1. 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.

**ListofPlayers.js**

****

**IndianPlayers.js**

This component demonstrates array merging and team division using index-based destructuring logic.

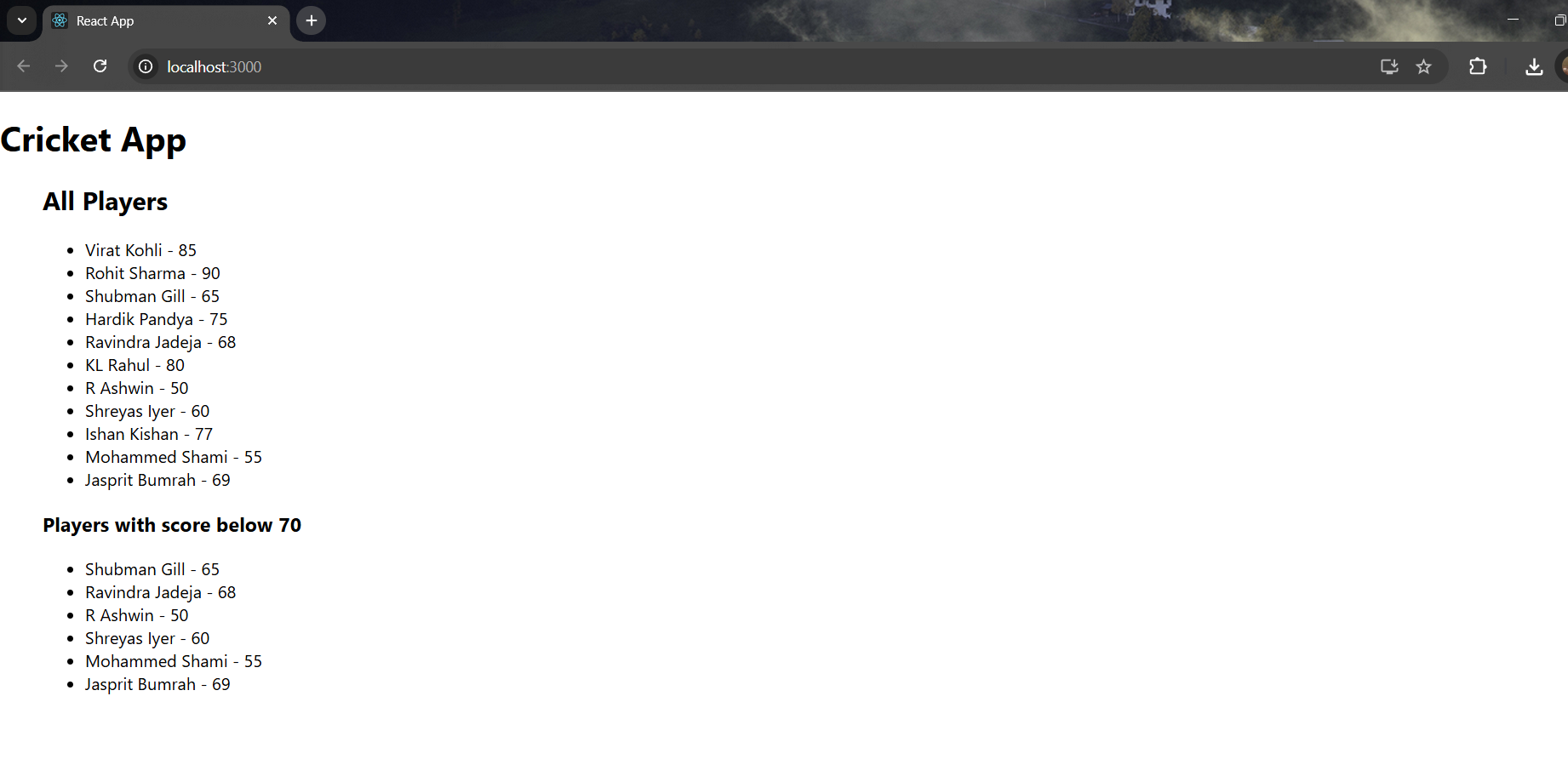
****

**App.js**

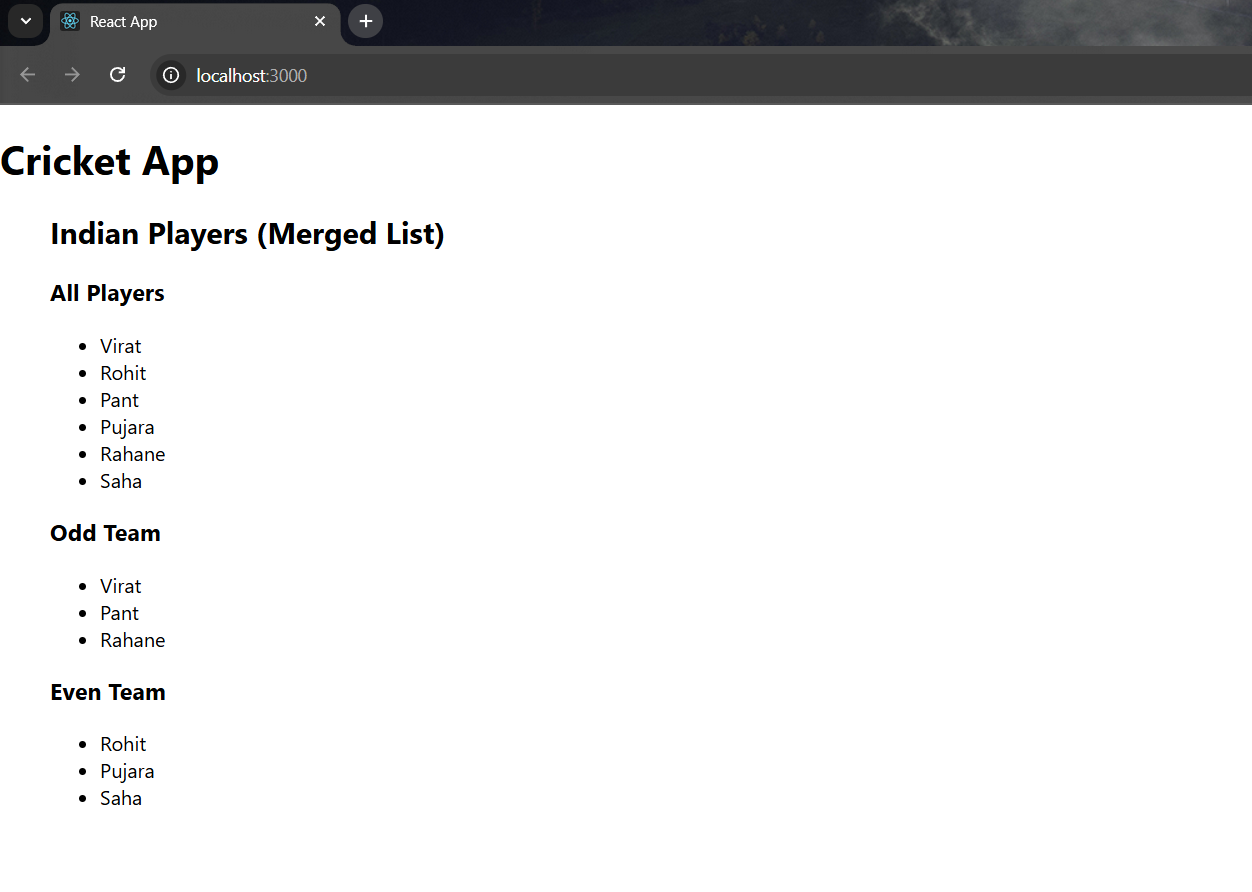
****

**OUTPUT:**

When Flag=true

****

When Flag=false

****