**Digital Nurture 4.0 Deep Skilling**

**Week 7**

**Hands on 1**

Create a React Application named “cricket app” with the following components:

1. List of Players

* 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.

2. Indian Players

1. Display the Odd Team Player and Even Team players using the Destructuring features of ES6
2. Declare two arrays T20players and Ranji Trophy 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.

**Objectives**

**1.Features of ES6 (ECMAScript 2015):**

1. let and const for block-scoped variables
2. Arrow functions ()=>{}
3. Template literals using backticks (```)
4. Classes and class inheritance
5. Default parameters in functions
6. Destructuring assignment (arrays & objects)
7. Spread and rest operators (...)
8. Promises for asynchronous operations
9. Modules (import/export syntax)
10. Map and Set data structures

**2.JavaScript let:**

* let is used to declare **block-scoped variables**.
* It **prevents redeclaration** in the same scope.
* Supports **temporal dead zone** (variable can't be accessed before declaration).

**Example:**

let x = 10;

x = 20; // allowed

let x = 30; // Error (redeclaration not allowed in same scope)

**3.Difference Between var and let:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **var** | **Let** |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Hoisted with undefined | Hoisted but not initialized (TDZ) |
| Redeclaration | Allowed | Not allowed in same scope |
| Global Object | Attached to window object | Not attached |

**4.JavaScript const:**

* Used to declare **block-scoped constants**.
* Must be **initialized** at the time of declaration.
* Cannot be **reassigned**, but **object contents can be modified**.

**Example:**

const a = 5;

a = 10; // Error

const obj = { name: "A" };

obj.name = "B"; //Allowed

**5.ES6 Class Fundamentals:**

* Introduced to simplify **object-oriented programming** in JavaScript.
* Uses class keyword and constructor() method.

**Example:**

class Person {

constructor(name) {

this.name = name;}

greet() {

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

}

}

**6.ES6 Class Inheritance:**

* Achieved using extends and super().
* Allows child classes to inherit from parent classes.

**Example:**

class Animal {

constructor(type) {

this.type = type;

}

}

class Dog extends Animal {

constructor(name) {

super("dog");

this.name = name;

}}

**7.Arrow Functions:**

* Shorter syntax for writing functions.
* Do **not bind their own this**.
* Cannot be used as constructors.

**Example:**

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

**8.Set() and Map() in ES6:**

**Set:**

* Collection of **unique values**
* No duplicates allowed.

**Example:**

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

**Map:**

* Collection of **key-value pairs**
* Keys can be of **any data type**.

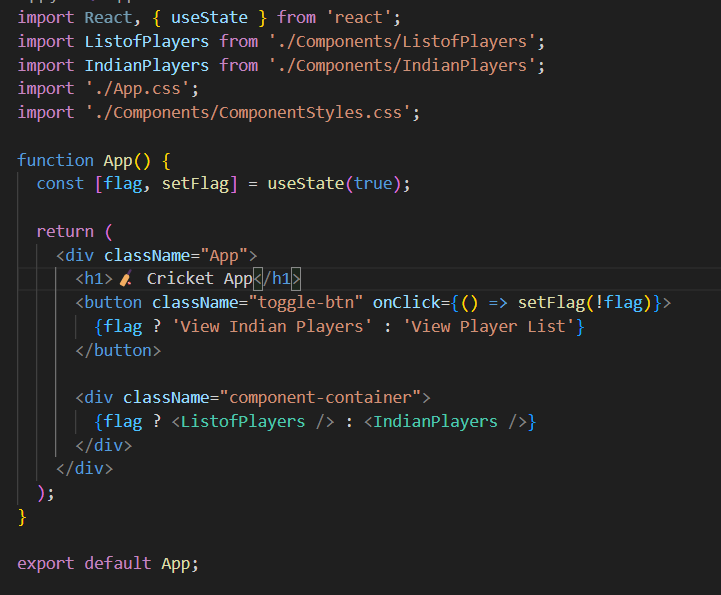
**Example:**

const myMap = new Map();

myMap.set('name', 'Alice');

myMap.set(1, 'One');

**Codes:**

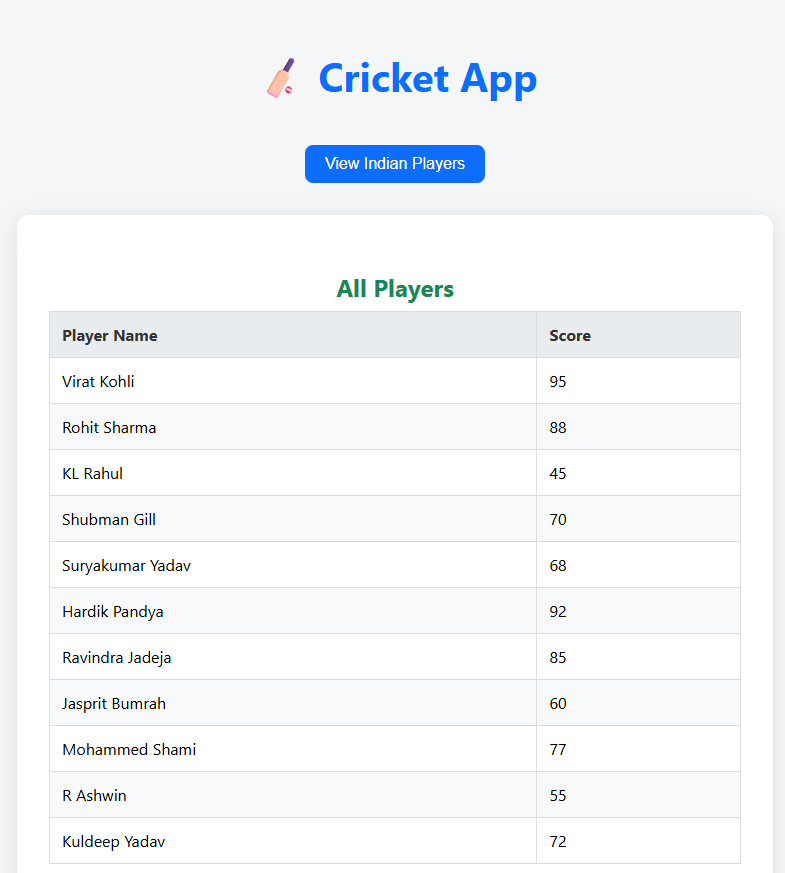
. 

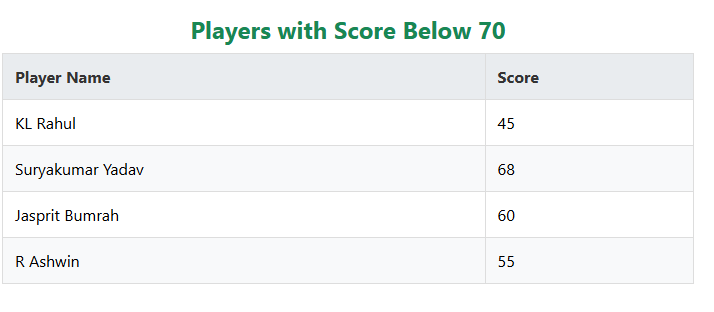




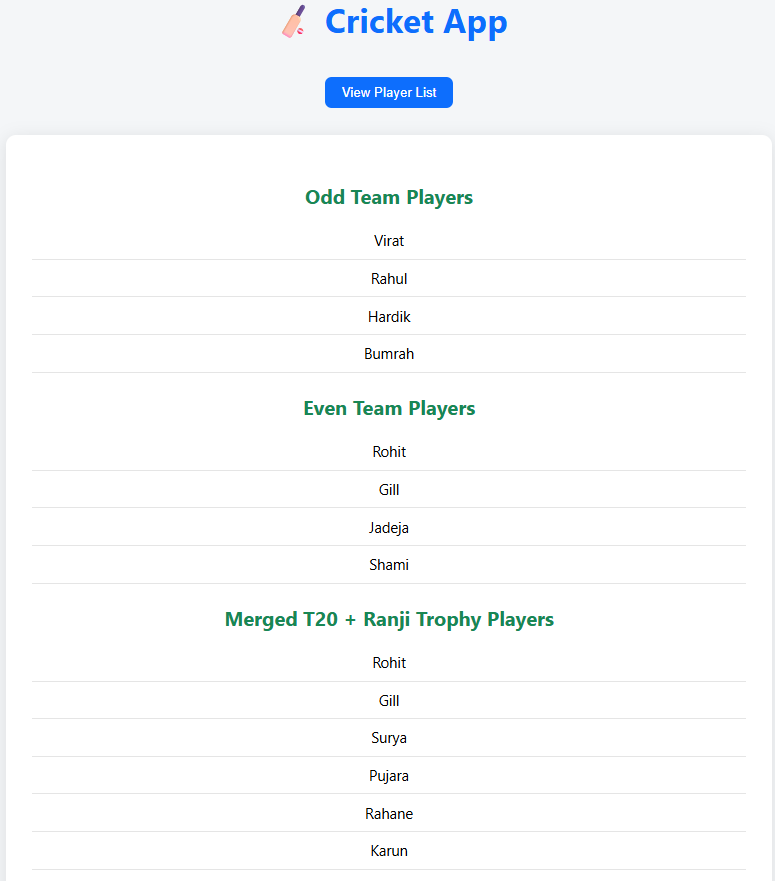
**Output :**

When flag = true

****

****

When flag=false

****

**Hands on 2**

Create a React Application named “office space rental app” which uses React JSX to create elements, attributes and renders DOM to display the page.

1. Create an element to display the heading of the page.
2. Attribute to display the image of the office space
3. Create an object of office to display the details like Name, Rent and Address.
4. Create a list of Object and loop through the office space item to display more data.
5. To apply css, Display the color of the Rent in Red if it’s below 60000 and in Green if it’s above 60000.

**Objectives**

**1.Define JSX:**

JSX (JavaScript XML) is a **syntax extension for JavaScript** used with React. It allows writing **HTML-like code inside JavaScript** files to describe the UI components in a readable way.

**Example:**

const element = <h1>Hello, JSX!</h1>;

JSX is **not valid JavaScript** by itself it gets **transpiled to React.createElement() calls** behind the scenes.

**2.Explain about ECMAScript (ECMA Script):**

ECMAScript is the **standardized specification** that JavaScript follows. It defines the core features of the language, like variables, functions, loops, classes, etc.

* ES6 (ECMAScript 2015) introduced modern features like:
  + let, const
  + Arrow functions
  + Classes and modules
  + Promises
  + Destructuring
  + Spread/rest operators

**3.Explain React.createElement():**

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

**Syntax:**

React.createElement(type, props, ...children)

**Example:**

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

This is equivalent to writing:

<h1 className="title">Hello React</h1>

JSX is internally converted to React.createElement() calls.

**4.Explain How to Create React Nodes with JSX:**

React nodes (elements) can be created using JSX directly in the render method or function.

**Example:**

const element = <h2>Welcome to React</h2>;

You can also nest elements:

const element = (

<div>

<h1>Hello</h1>

<p>This is a paragraph.</p>

</div>

);

**5.Define How to Render JSX to DOM:**

JSX can be rendered to the actual browser DOM using ReactDOM.render().

**Example:**

import React from 'react';

import ReactDOM from 'react-dom';

const element = <h1>Hello World</h1>;

ReactDOM.render(element, document.getElementById('root'));

* This will render the element inside the HTML element with id="root".

**6.Explain How to Use JavaScript Expressions in JSX:**

You can embed JavaScript expressions inside JSX using **curly braces {}**.

**Example:**

const name = "Alice";

const element = <h1>Hello, {name}</h1>;

You can also use functions and arithmetic:

<h2>{5 + 3}</h2>

<p>{getGreeting()}</p>

Note: You cannot use statements like if, for directly only expressions.

**7.Explain How to Use Inline CSS in JSX:**

Inline styles in JSX are written as a **JavaScript object**, not a regular string.

**Example:**

const divStyle = {

color: 'blue',

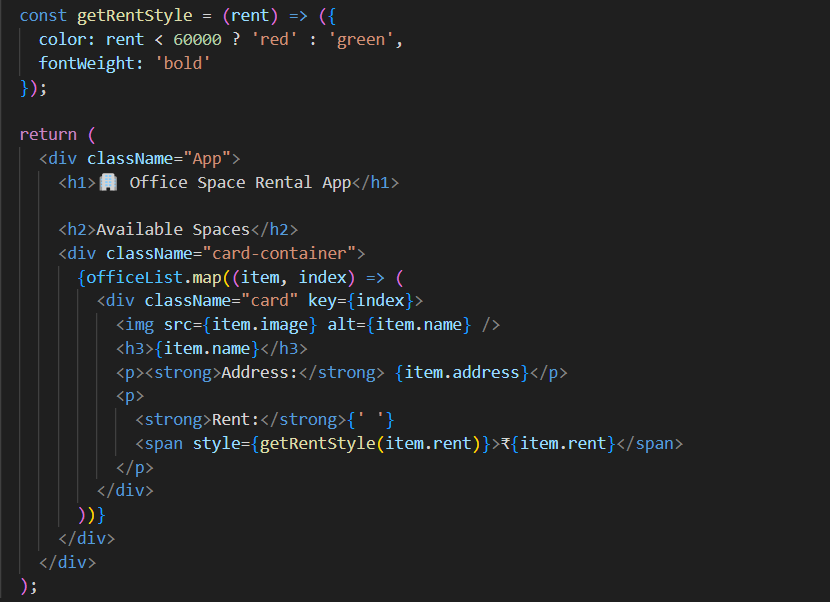
backgroundColor: 'lightgray',

fontSize: '20px'

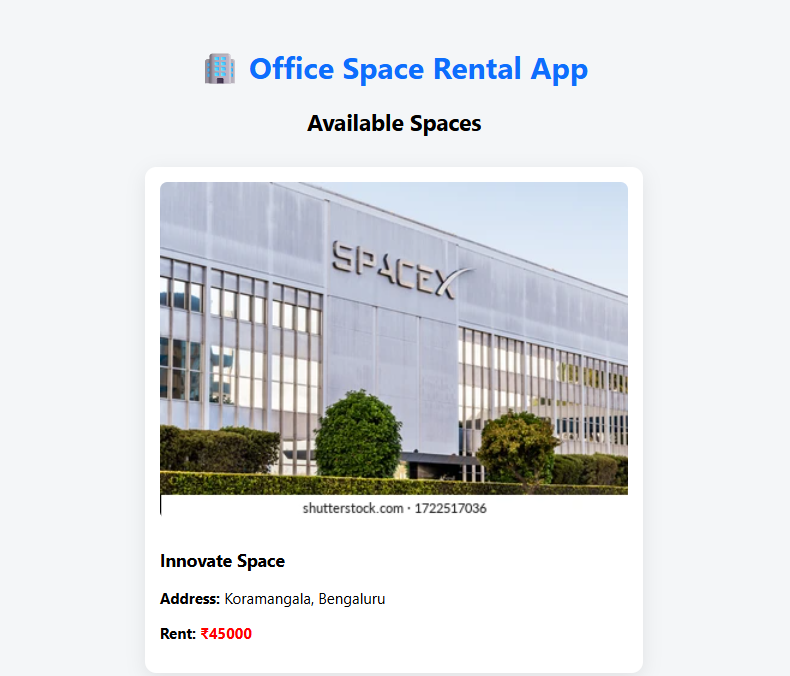
};

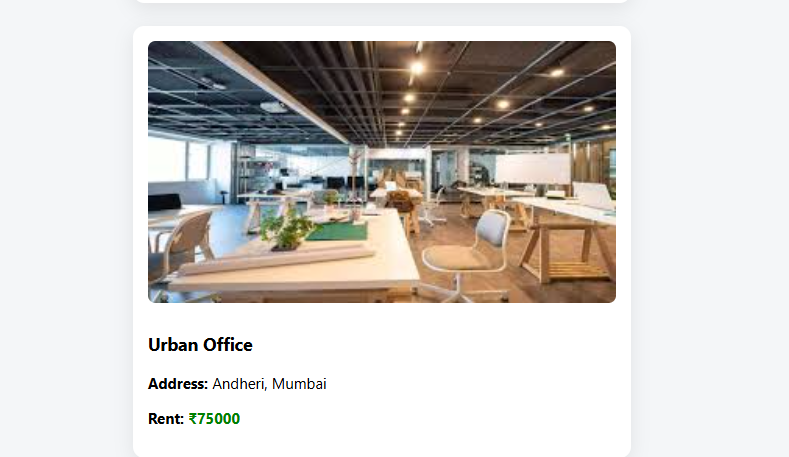
const element = <div style={divStyle}>Styled Text</div>;

CSS properties use **camelCase** (e.g., backgroundColor instead of background-color).

****

**Output :**

****

****

**Hands on 3**

Create a React application named **"event examples app"** to handle various form events.

 Include an **"Increment"** button to increase a counter and a **"Decrement"** button to decrease it.

 The "Increment" button should invoke multiple methods:

* One method to increment the counter value.
* Another method to display a static message like **"Hello Member1"**.

 Create a **"Say Welcome"** button that invokes a function taking **"Welcome"** as an argument and displays it.

 Add a **"Click on me"** button that demonstrates a **synthetic event**, showing the message **"I was clicked!"** when triggered.

 Create a separate component named **"Currency Convertor"**.

* This component should include two input fields: one for **amount in Indian Rupees** and another for the **target currency** (e.g., Euro).
* Include a **"Submit"** button that, when clicked, triggers the **handle Submit** event.
* The event handler should simulate or alert a conversion of the given amount from rupees to the specified currency.

**Objectives**

**1.Explain React Events:**

React events are the **React version of DOM events** (like click, submit, change, etc.) that allow components to **respond to user interactions**.

React uses a **cross-browser wrapper** around native events called the **Synthetic Event system**, which ensures consistency across all browsers.

**Example:**

<button onClick={handleClick}>Click Me</button>

**2.Explain About Event Handlers:**

Event handlers are functions that are **called when an event occurs**. In React, these are passed as props like onClick, onChange, etc., and are usually defined as arrow functions or class methods.

**Example:**

function handleClick() {

alert('Button clicked!');

}

return <button onClick={handleClick}>Click Me</button>;

* Event handlers in React can also access the **event object** as a parameter.

**3.Define Synthetic Event:**

A Synthetic Event is a **React wrapper** around the browser’s native event, ensuring consistent behavior across different browsers.

React creates its own event system for performance and compatibility, and this system wraps native DOM events in SyntheticEvent objects.

**Example:**

function handleClick(event) {

console.log(event); // SyntheticEvent

}

* Synthetic Events pool events for performance. You must call event.persist() to retain the event object asynchronously.

**4.Identify React Event Naming Convention:**

|  |  |
| --- | --- |
| **HTML DOM Event** | **React Equivalent** |
| onclick | onClick |
| onchange | onChange |
| onmouseover | onMouseOver |
| onsubmit | onSubmit |

* React event names use **camelCase** (e.g., onClick, not onclick).
* The handler value is **passed as a function**, not a string.

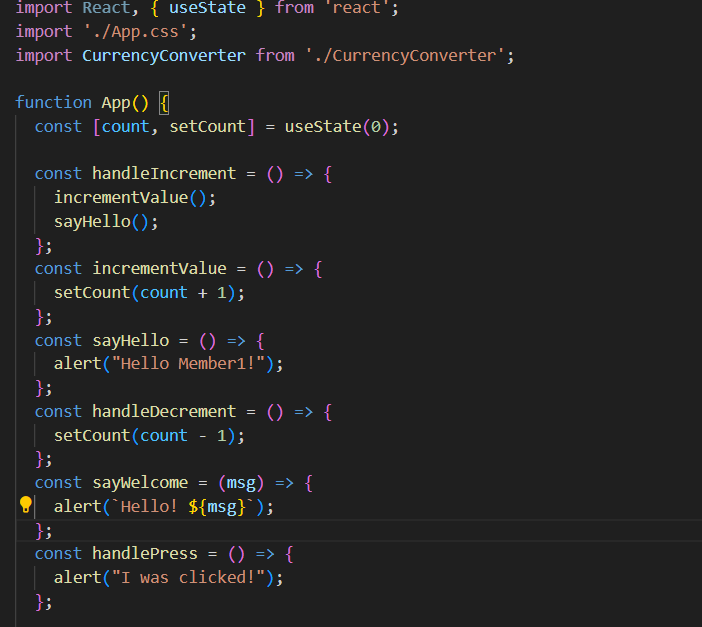
**Correct:**

<button onClick={handleClick}>Click</button>

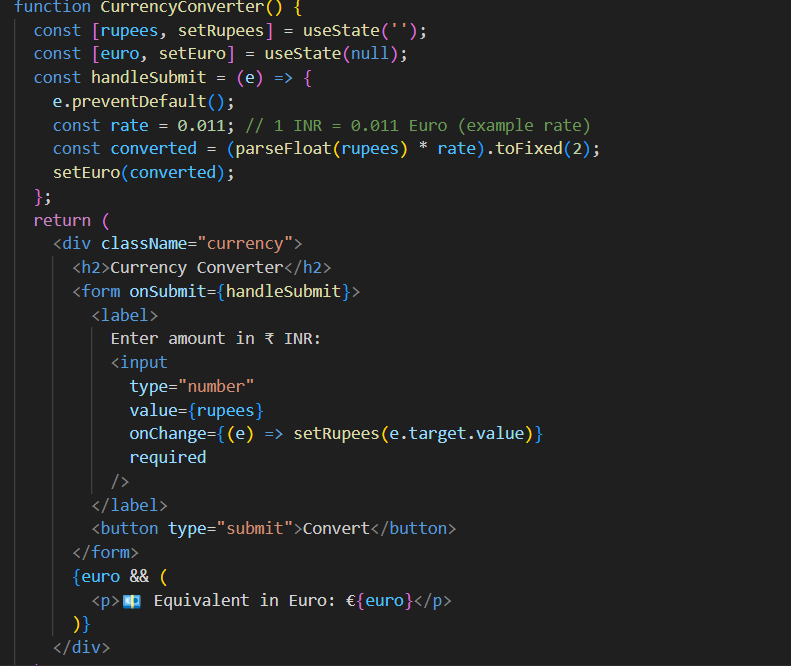
**Incorrect:**

<button onclick="handleClick()">Click</button> // Invalid in React

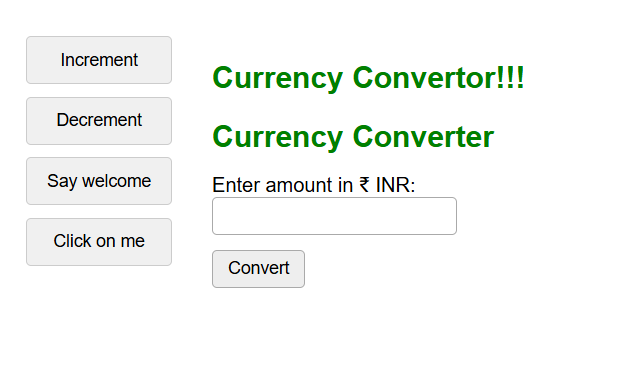
**Code :**

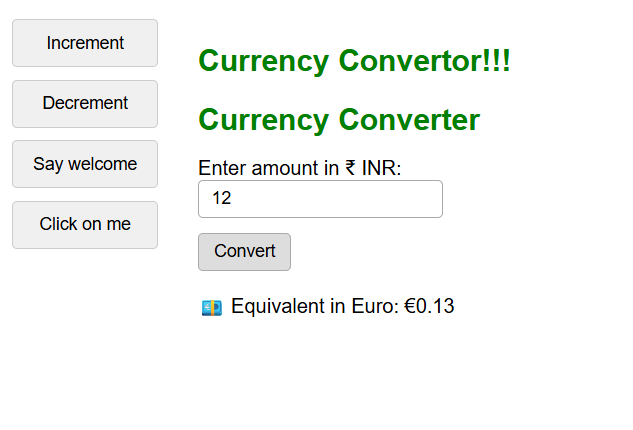


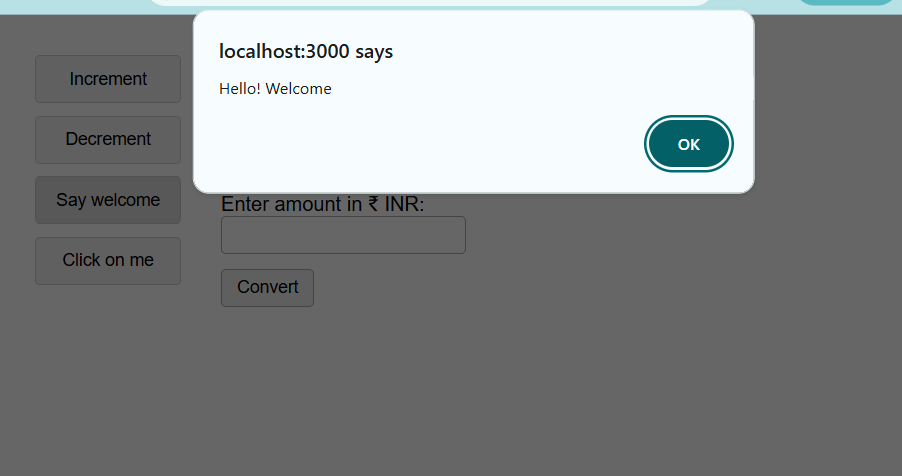


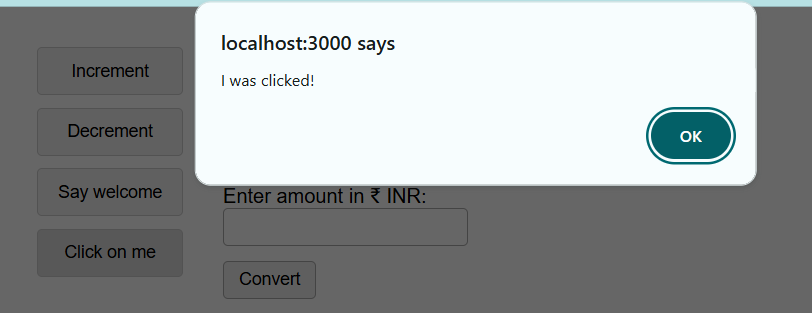
****

**Output :**

****

****

****

****

**Hands on 4**

Create a React Application named “ticket booking app” where the guest user can browse the page where the flight details are displayed whereas the logged in user only can book tickets.

The Login and Logout buttons should accordingly display different pages. Once the user is logged in the User page should be displayed. When the user clicks on Logout, the Guest page should be displayed.

**Objectives**

**1.Explain About Conditional Rendering in React:**

Conditional rendering in React means showing different **UI elements or components** based on certain conditions (just like if statements in JavaScript).

React supports conditional rendering using:

1. **If-else statements**
2. **Ternary operators**
3. **Logical && operators**

**Examples:**

1. Using if:

if (isLoggedIn) {

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

} else {

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

}

2. Using ternary operator:

return (

<div>

{isOnline ? <p>Online</p> : <p>Offline</p>}

</div>

);

3.Using && operator:

{unreadMessages.length > 0 && <h2>You have messages</h2>}

**2.Define Element Variables:**

Element variables are **JavaScript variables** that can store **React elements**, which can later be conditionally rendered.

They’re useful when you want to store a block of JSX and render it conditionally.

**Example:**

let message;

if (isLoggedIn) {

message = <h1>Welcome User</h1>;

} else {

message = <h1>Please Login</h1>;

}

return <div>{message}</div>;

**3.Explain How to Prevent Components from Rendering:**

To prevent a component from rendering, you can:

**1. Return null:**

In the component’s render or return block, simply return null when you want to **skip rendering**.

function WarningBanner(props) {

if (!props.showWarning) {

return null; // component does not render anything

}

return <div className="warning">Warning!</div>;

}

**2. Use conditional rendering in parent:**

Only render the component if a condition is true.

{isVisible && <MyComponent />}

This will render MyComponent only if isVisible is true.

**Code:**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer screen

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Output :**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Hands on 5**

Estimated time to complete this lab: **60 minutes.**

Create a React App named “bloggerapp” in with 3 components.

1. Book Details
2. Blog Details
3. Course Details

Implement this with as many ways possible of Conditional Rendering.

**Objectives:**

**1.Explain Various Ways of Conditional Rendering:**

React offers several ways to conditionally render content:

1. **Using if-else statements**  
   Used outside JSX, typically before the return.

if (isLoggedIn) {

return <Dashboard />;

} else {

return <Login />;

}

1. **Using ternary operator ? :**  
   Used directly in JSX.

{isOnline ? <p>Online</p> : <p>Offline</p>}

1. **Using logical AND (&&) operator**  
   Renders the right side only if the left side is true.

{notifications.length > 0 && <Notification />}

1. **Using switch-case inside functions**  
   Useful for rendering based on multiple cases.

**2.Explain How to Render Multiple Components:**

React allows you to render multiple components inside a single parent component using:

**1.React Fragments (<>...</>)**:

return (

<>

<Header />

<Content />

<Footer />

</>

);

**2.Returning an array of components**:

return [

<li key="1">Item 1</li>,

<li key="2">Item 2</li>

];

**3.JSX group inside a div or other element**:

return (

<div>

<NavBar />

<Main />

</div>

);

**3. Define List Component:**

A List Component in React is a component that **renders a collection of items** using the JavaScript map() function.

**Example:**

function PlayerList({ players }) {

return (

<ul>

{players.map(player => (

<li key={player.id}>{player.name}</li>

))}

</ul>

);

}

**4.Explain About Keys in React Applications:**

**Keys** are unique identifiers assigned to elements in a list to help React **track which items changed, added, or removed**.

* Improve **rendering performance**.
* Keys must be **unique and stable** (don't use array index if data can change).

**Example:**

{items.map(item => (

<li key={item.id}>{item.name}</li>

))}

**5.Explain How to Extract Components with Keys:**

When mapping over a list and rendering components, each child component must have a unique key especially if you **extract the component** to another function or file.

**Example:**

function Player({ player }) {

return <li>{player.name}</li>;

}

function PlayerList({ players }) {

return (

<ul>

{players.map(p => (

<Player key={p.id} player={p} />

))}

</ul>

);

}

Key must be added to the component *at the place where it’s created*, not inside the child.

**6. Explain React map() / JavaScript map() Function:**

The map() function is a **JavaScript array method** used in React to **iterate over a list and render UI elements** dynamically.

**Syntax:**

array.map((item, index) => {

return <Component />;

});

**React Example:**

const names = ['Alice', 'Bob', 'Charlie'];

const nameList = names.map((name, index) => <li key={index}>{name}</li>);

map() is preferred in React for rendering **repeated JSX structures** based on data.

**Code :**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Output :**

**A black screen with white text

Description automatically generated**