**WEEK 7(Hands -On Exercises)**

**ReactJS HOL 09:**

**Objectives:**

* List the features of ES6
* Explain JavaScript let
* Identify the differences between var and let
* Explain JavaScript const
* Explain ES6 class fundamentals
* Explain ES6 class inheritance
* Define ES6 arrow functions
* Identify set(), map()

**Key Features of ES6 (ECMAScript 2015)**

1. **Block-scoped variables**: let and const declarations
2. **Arrow functions**: Shorter function syntax
3. **Classes**: Syntactical sugar over JavaScript's prototype-based inheritance
4. **Enhanced object literals**: Shorter syntax for object properties
5. **Template literals**: String interpolation with backticks
6. **Destructuring assignment**: Unpack values from arrays/objects
7. **Default parameters**: Function parameters with default values
8. **Rest and spread operators**: ... syntax
9. **Modules**: import and export syntax
10. **Promises**: For asynchronous programming
11. **New built-in methods**: For strings, arrays, numbers, etc.
12. **Map and Set**: New data structures

**JavaScript let**

The let keyword declares block-scoped variables:

javascript

let x = 10;

if (true) {

let x = 20;

console.log(x);

}

console.log(x);

**Differences Between var and let**

| **Feature** | var | let |
| --- | --- | --- |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Hoisted (undefined) | Hoisted but not initialized (TDZ) |
| Re-declaration | Allowed in same scope | Not allowed |
| Global object | Becomes property | Doesn't become property |

**Example:**

**javascript**

function varLetExample() {

console.log(a);

console.log(b);

var a = 1;

let b = 2;

{

var a = 3;

let b = 4;

console.log(a);

console.log(b);

}

**JavaScript const**

The const keyword declares block-scoped constants:

**javascript**

const PI = 3.14159;

const obj = { name: 'John' };

obj.name = 'Jane';

**ES6 Class Fundamentals**

ES6 classes are syntactic sugar over JavaScript's prototype-based inheritance:

**javascript**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

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

}

static info() {

return 'This is a Person class';

}

}

const john = new Person('John', 30);

console.log(john.greet());

console.log(Person.info());

**ES6 Class Inheritance**

Classes can extend other classes using extends:

javascript

class Employee extends Person {

constructor(name, age, jobTitle) {

super(name, age);

this.jobTitle = jobTitle;

}

greet() {

return `${super.greet()} and I'm a ${this.jobTitle}`;

}

}

const dev = new Employee('Jane', 25, 'Developer');

console.log(dev.greet());

**ES6 Arrow Functions**

Arrow functions provide a concise syntax and lexical this binding:

javascript

const add = function(a, b) {

return a + b;

};

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

const square = x => x \* x;

const greet = () => 'Hello';

const multiply = (a, b) => {

const result = a \* b;

return result;

};

function Timer() {

this.seconds = 0;

setInterval(() => {

this.seconds++;

}, 1000);

}

**Set and Map**

Set is a collection of unique values:

javascript

const mySet = new Set();

mySet.add(1);

mySet.add(2);

mySet.add(1);

console.log(mySet.size);

console.log(mySet.has(2));

mySet.delete(2);

**Map**

Collection of key-value pairs (keys can be any type):

javascript

const myMap = new Map();

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

myMap.set(1, 'number one');

myMap.set({}, 'object key');

console.log(myMap.get('name'));

console.log(myMap.size);

myMap.delete(1);

**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.**

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

**App.js**

import React from "react";

import ListOfPlayers from "./components/ListOfPlayers";

import ScoreBelow70 from "./components/ScoreBelow70";

import { OddPlayers } from "./components/OddPlayers";

import { EvenPlayers } from "./components/EvenPlayers";

import ListOfIndianPlayers from "./components/ListOfIndianPlayers";

function App() {

const flag = false; // change to true or false to toggle views

const players = [

{ name: "Jack", score: 50 },

{ name: "Michael", score: 70 },

{ name: "John", score: 40 },

{ name: "Ann", score: 61 },

{ name: "Elisabeth", score: 61 },

{ name: "Sachin", score: 95 },

{ name: "Dhoni", score: 100 },

{ name: "Virat", score: 84 },

{ name: "Jadeja", score: 64 },

{ name: "Raina", score: 75 },

{ name: "Rohit", score: 80 }

];

const IndianTeam = ["Sachin1", "Dhoni2", "Virat3", "Rohit4", "Yuvaraj5", "Raina6"];

const T20Players = ["First Player", "Second Player", "Third Player"];

const RanjiTrophyPlayers = ["Fourth Player", "Fifth Player", "Sixth Player"];

const IndianPlayers = [...T20Players, ...RanjiTrophyPlayers];

if (flag === true) {

return (

<div>

<h1>List of Players</h1>

<ListOfPlayers players={players} />

<hr />

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

<ScoreBelow70 players={players} />

</div>

);

} else {

return (

<div>

<h1>Indian Team</h1>

<h2>Odd Players</h2>

<OddPlayers players={IndianTeam} />

<hr />

<h2>Even Players</h2>

<EvenPlayers players={IndianTeam} />

<hr />

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

<ListOfIndianPlayers IndianPlayers={IndianPlayers} />

</div>

);

}

}

export default App;

**ListOfPlayers.js**

import React from "react";

const ListOfPlayers = ({ players }) => {

return (

<ul>

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

<li key={index}>

Mr. {item.name} <span>{item.score}</span>

</li>

))}

</ul>

);

};

export default ListOfPlayers;

**ScoreBelow70.js**

import React from "react";

const ScoreBelow70 = ({ players }) => {

const filtered = players.filter((item) => item.score < 70);

return (

<ul>

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

<li key={index}>

Mr. {item.name} <span>{item.score}</span>

</li>

))}

</ul>

);

};

export default ScoreBelow70;

**OddPlayers.js**

import React from "react";

export const OddPlayers = ({ players: [first, , third, , fifth] }) => {

return (

<div>

<li>First : {first}</li>

<li>Third : {third}</li>

<li>Fifth : {fifth}</li>

</div>

);

};

**EvenPlayers.js**

import React from "react";

export const EvenPlayers = ({ players: [, second, , fourth, , sixth] }) => {

return (

<div>

<li>Second : {second}</li>

<li>Fourth : {fourth}</li>

<li>Sixth : {sixth}</li>

</div>

);

};

**ListOfIndianPlayers.js**

import React from "react";

const ListOfIndianPlayers = ({ IndianPlayers }) => {

return (

<ul>

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

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

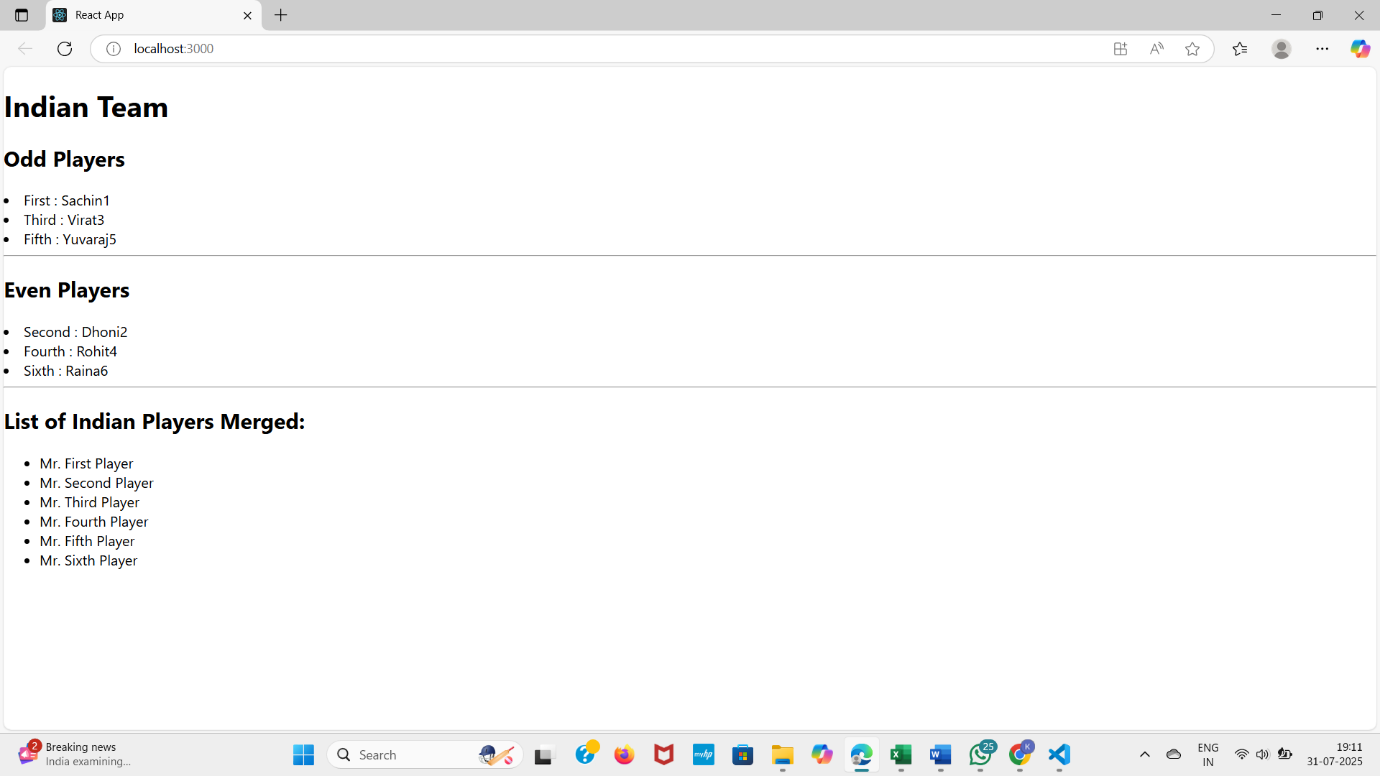
))}

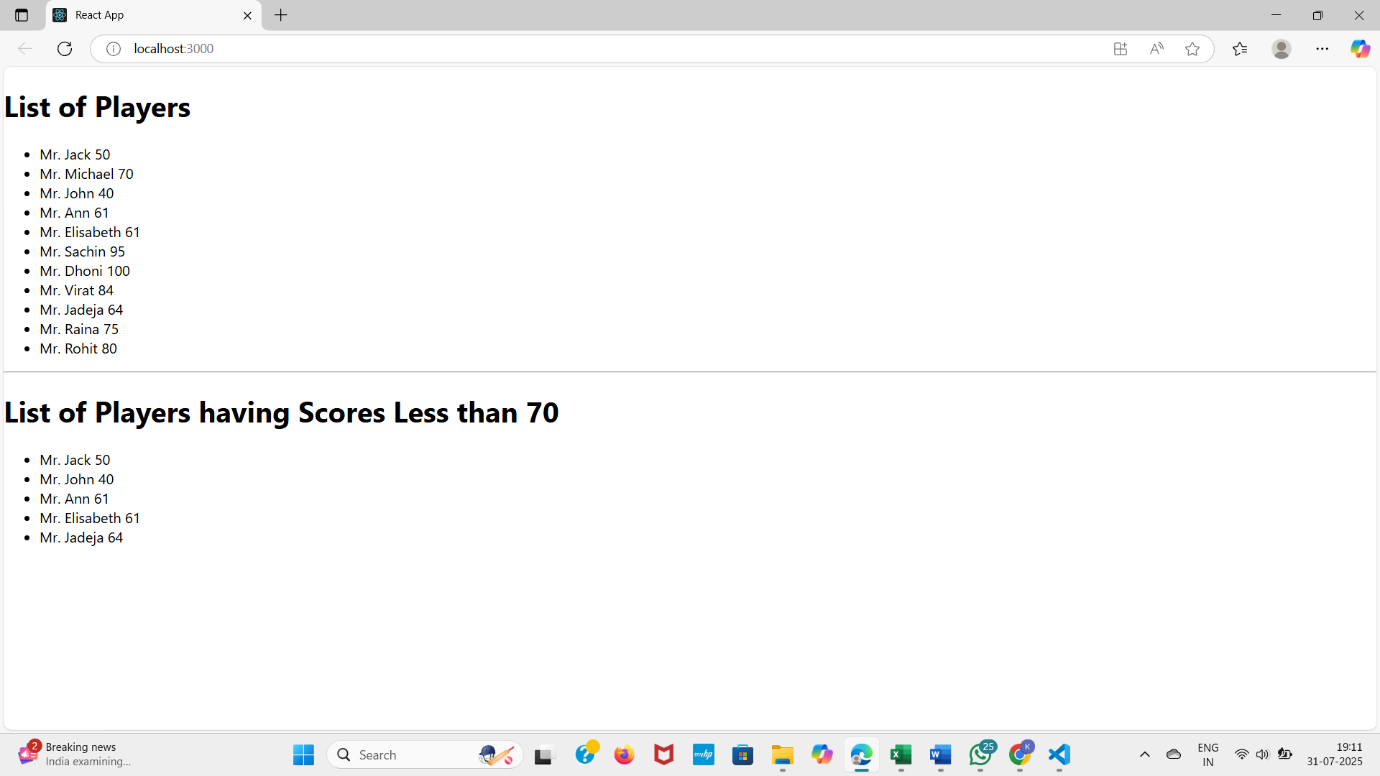
</ul>

);

};

export default ListOfIndianPlayers;

**OUTPUT**



**ReactJS- HOL10**

**Objectives**

* Define JSX
* Explain about ECMA Script
* Explain React.createElement()
* Explain how to create React nodes with JSX
* Define how to render JSX to DOM
* Explain how to use JavaScript expressions in JSX

**Definition of JSX**

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code in your JavaScript files. It's not valid JavaScript by itself but gets transformed into regular JavaScript function calls (specifically React.createElement()) during the build process.

**Key characteristics:**

* Looks like HTML/XML but is actually JavaScript
* Provides syntactic sugar for creating React elements
* Not required to use React, but makes code more readable
* Must be transpiled to JavaScript before browsers can understand it

**ECMAScript (ES) Overview**

ECMAScript is the standardized scripting language specification that JavaScript is based on. Key points:

* JavaScript is an implementation of the ECMAScript standard
* ECMAScript versions are abbreviated as ES (e.g., ES5, ES6/ES2015)
* ES6 (2015) introduced major new features (classes, modules, arrow functions, etc.)
* New versions are released annually (ES2016, ES2017, etc.)
* JSX is not part of ECMAScript - it's a separate syntax extension

React.createElement()

The fundamental way to create React elements without JSX:

javascript

React.createElement(

type,

[props],

[...children]

);

Example:

javascript

const element = React.createElement(

'h1',

{ className: 'greeting' },

'Hello, world!'

);

This is equivalent to the JSX:

jsx

const element = <h1 className="greeting">Hello, world!</h1>;

Creating React Nodes with JSX

Basic JSX syntax rules:

1. Basic elements:

const heading = <h1>Welcome to React</h1>;

1. Nested elements:

const container = (

<div className="app">

<header>

<h1>My App</h1>

</header>

<main>Content goes here</main>

</div>

);

1. Self-closing tags

const image = <img src="logo.png" alt="Logo" />;

1. Attributes (use camelCase for DOM properties):

const input = <input type="text" readOnly={true} />;

Rendering JSX to the DOM

To render JSX to the DOM:

1. Create a root element in your HTML:

html

<div id="root"></div>

1. Use ReactDOM.render() (or createRoot in React 18+):

javascript

ReactDOM.render(

<App />,

document.getElementById('root')

);

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

root.render(<App />);

Using JavaScript Expressions in JSX

You can embed any JavaScript expression in JSX using curly braces {}:

1. Variables:

const name = 'Alice';

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

1. Expressions:

const total = <p>Total: {5 + 10}</p>;

1. Function calls:

function formatName(user) {

return `${user.firstName} ${user.lastName}`;

}

const user = { firstName: 'John', lastName: 'Doe' };

const element = <h1>Hello, {formatName(user)}!</h1>;

1. Conditional rendering:

const isLoggedIn = true;

const message = (

<div>

{isLoggedIn ? <p>Welcome back!</p> : <p>Please sign in.</p>}

</div>

);

**Using Inline CSS in JSX**

In JSX, inline styles are specified as an object with camelCase properties:

1. Basic syntax:

const divStyle = {

color: 'blue',

backgroundColor: 'lightgray',

padding: '10px',

borderRadius: '5px'

};

const styledDiv = <div style={divStyle}>Styled content</div>;

1. Directly in element:

<h1 style={{ color: 'red', fontSize: '24px' }}>Hello</h1>

1. Dynamic styles:

jsx

const isError = true;

const messageStyle = {

color: isError ? 'red' : 'green',

fontWeight: 'bold'

};

<p style={messageStyle}>{isError ? 'Error!' : 'Success!'}</p>

**Explain how to use inline CSS in JSX**

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

Create an element to display the heading of the page.

Attribute to display the image of the office space

Create an object of office to display the details like Name, Rent and Address.

Create a list of Object and loop through the office space item to display more data.

**To apply Css, Display the color of the Rent in Red if it’s below 60000 and in Green if it’s above 60000**.

**App.js**

import React from "react";

import "./App.css";

import officeImg from "./assets/office.jpg";

function App() {

const heading = "Office Space";

const officeList = [

{ Name: "DBS", Rent: 50000, Address: "Chennai" },

{ Name: "WeWork", Rent: 70000, Address: "Bangalore" },

{ Name: "Regus", Rent: 55000, Address: "Hyderabad" },

{ Name: "IndiQube", Rent: 80000, Address: "Pune" },

];

return (

<div className="App">

<h1>{heading}, at Affordable Range</h1>

<img src={sr} width="25%" height="25%" alt="Office Space" />

<hr />

{officeList.map((office, index) => {

const rentClass = office.Rent <= 60000 ? "textRed" : "textGreen";

return (

<div key={index} className="officeBox">

<h2>Name: {office.Name}</h2>

<h3 className={rentClass}>Rent: Rs. {office.Rent}</h3>

<h3>Address: {office.Address}</h3>

</div>

);

})}

</div>

);

}

export default App;

**App.css**

.App {

padding: 20px;

font-family: Arial, sans-serif;

}

.textRed {

color: red;

font-weight: bold;

}

.textGreen {

color: green;

font-weight: bold;

}

.officeBox {

border: 1px solid #ccc;

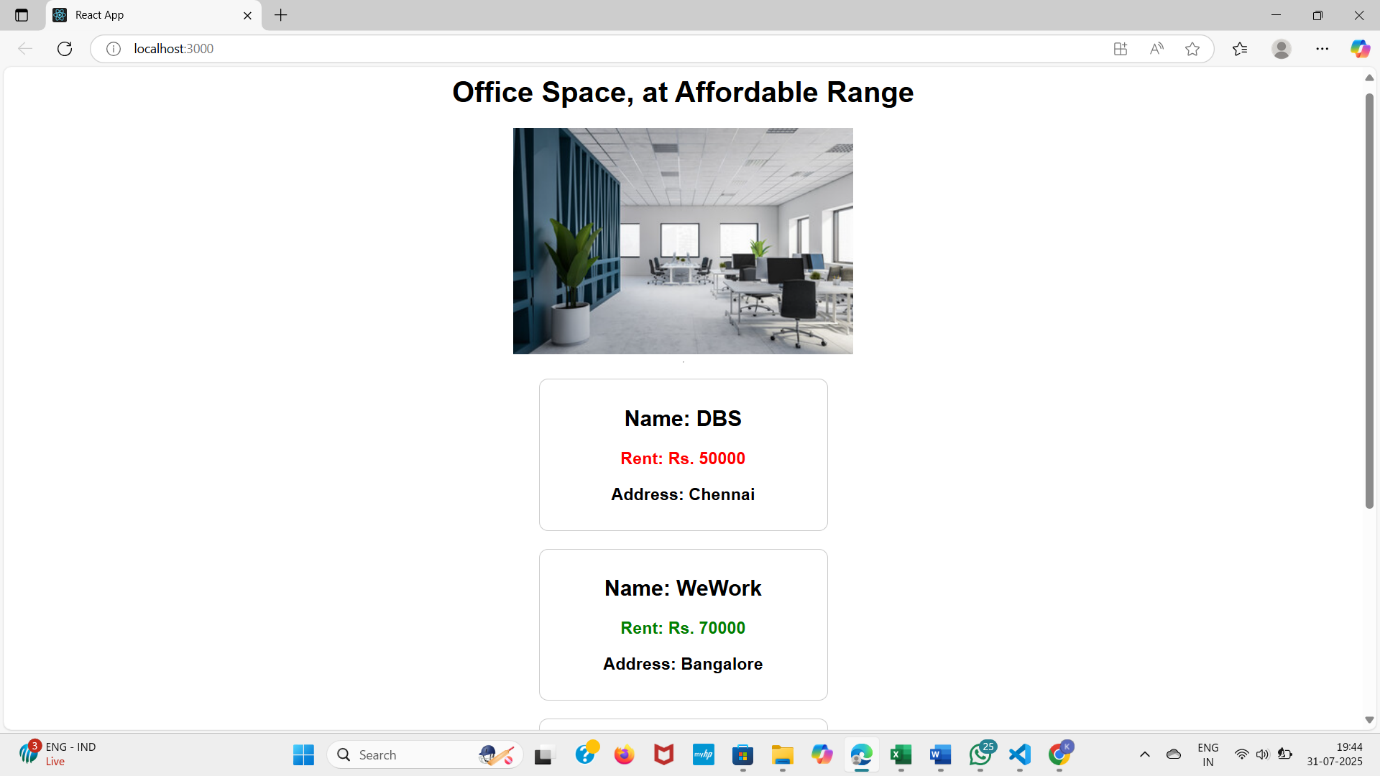
border-radius: 10px;

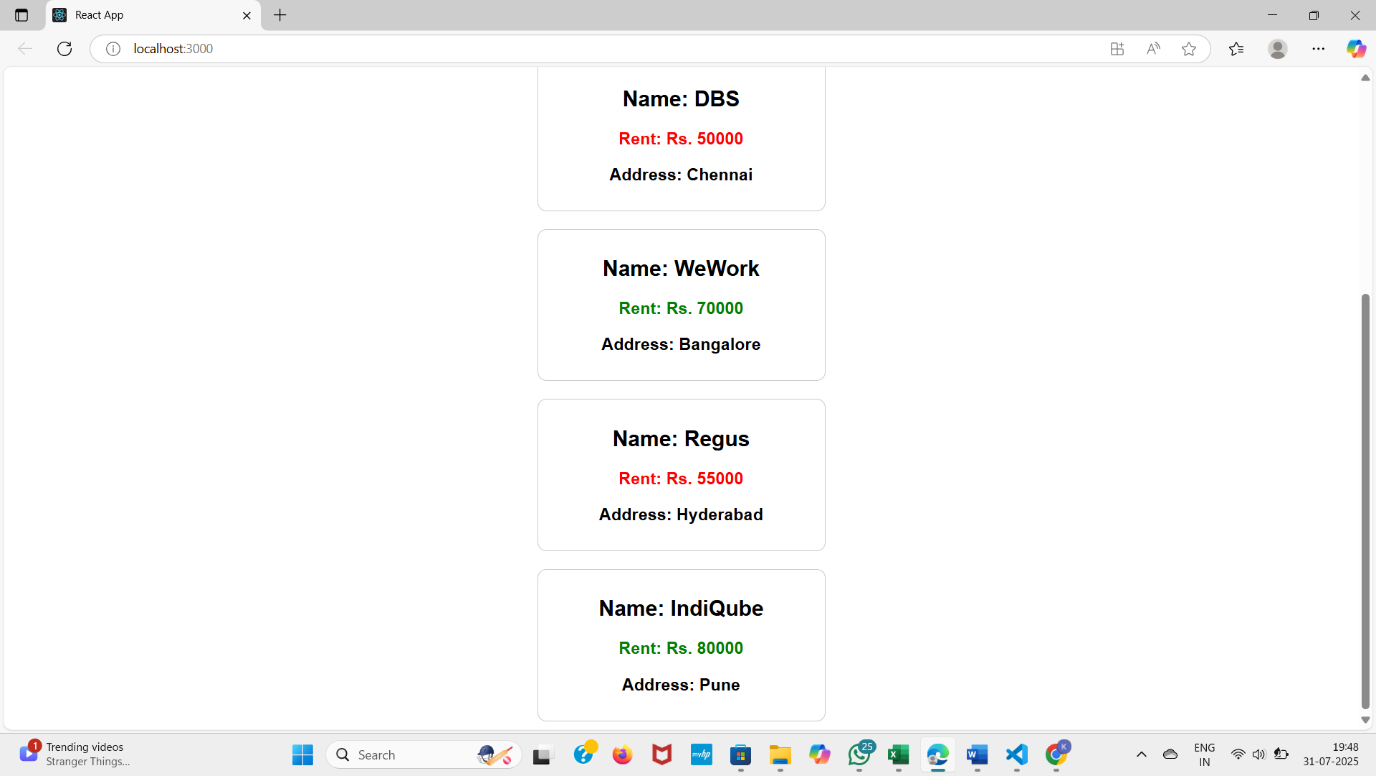
padding: 10px;

margin: 10px;

width: 300px;

}

**OUTPUT:**



**ReactJS- HOL 11**

**Objectives**

* **Explain React events**
* **Explain about event handlers**
* **Define Synthetic event**
* **Identify React event naming convention**

**React Events and Event Handling**

**React Events Overview**

React events are wrappers around native browser events that provide cross-browser compatibility. Key characteristics:

* Use camelCase naming (onClick instead of onclick)
* Pass functions as event handlers rather than strings
* Receive synthetic event objects that wrap native events
* Work similarly to DOM events but with some important differences

Basic example:

function Button() {

function handleClick() {

console.log('Button clicked');

}

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

}

**Event Handlers in React**

Event handlers in React are functions that are called in response to user interactions:

**1. Inline function:**

<button onClick={() => console.log('Clicked')}>Click</button>

**2. Class method (in class components):**

class MyComponent extends React.Component {

handleClick() {

console.log('Clicked');

render() {

return <button onClick={this.handleClick.bind(this)}>Click</button>;

}

}

**3. Arrow function (avoids binding issues):**

class MyComponent extends React.Component {

handleClick = () => {

console.log('Clicked');

}

render() {

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

}

}

**4. Function component with handler:**

function MyComponent() {

const handleClick = () => {

console.log('Clicked');

};

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

}

**Synthetic Events**

React's **SyntheticEvent** is a cross-browser wrapper around the browser's native event:

Key features:

* Normalizes events across different browsers
* Has the same interface as native events (stopPropagation(), preventDefault())
* Events are pooled (for performance) so they can't be accessed asynchronously
* If you need to access the event asynchronously, call event.persist()

Example:

function handleClick(event) {

event.preventDefault();

event.stopPropagation();

event.persist();

setTimeout(() => {

console.log(event.type);

}, 100);

}

**React Event Naming Convention**

**React events follow these naming conventions:**

| **DOM Event** | **React Event** |
| --- | --- |
| onclick | onClick |
| onchange | onChange |
| onsubmit | onSubmit |
| onblur | onBlur |
| onfocus | onFocus |
| onmouseover | onMouseOver |
| onkeydown | onKeyDown |
| onload | onLoad |
| onerror | onError |

Key patterns:

1. **CamelCase**: All events are named using camelCase
2. **"on" prefix**: All event handler props start with "on"
3. **Different behavior**: Some events work differently than their DOM counterparts (e.g., onChange fires for every keystroke)

Common Event Examples

**Form submission:**

function Form() {

const handleSubmit = (event) => {

event.preventDefault();

console.log('Form submitted');

};

return (

<form onSubmit={handleSubmit}>

<button type="submit">Submit</button>

</form>

);

}

**Input change:**

function InputField() {

const [value, setValue] = useState('')

const handleChange = (event) => {

setValue(event.target.value);

};

return <input type="text" value={value} onChange={handleChange} />;

}

**Mouse events:**

function MouseTracker() {

const handleMouseEnter = () => console.log('Mouse entered');

const handleMouseLeave = () => console.log('Mouse left');

return (

<div

onMouseEnter={handleMouseEnter}

onMouseLeave={handleMouseLeave}

>

</div>

);

}

**Key events:**

function KeyLogger() {

const handleKeyDown = (event) => {

if (event.key === 'Enter') {

console.log('Enter pressed');

}

};

return <input onKeyDown={handleKeyDown} />;

}

**Create a React Application “eventexamplesapp” to handle various events of the form elements in HTML.**

1.Create “Increment” button to increase the value of the counter and “Decrement” button to decrease the value of the counter. The “Increase” button should invoke multiple methods.

* 1. To increment the value

Say Hello followed by a static message

2. Create a button “Say Welcome” which invokes the function which takes “welcome” as an argument.

3. Create a button which invokes synthetic event “OnPress” which display “I was clicked”

Create a “CurrencyConvertor” component which will convert the Indian Rupees to Euro when the Convert button is clicked.

Handle the Click event of the button to invoke the handleSubmit event and handle the conversion of the euro to rupees.

**Counter.js**

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

const increment = () => {

setCount(prev => prev + 1);

sayHello();

};

const decrement = () => {

setCount(prev => prev - 1);

};

const sayHello = () => {

alert("Hello! This is a static message.");

};

const sayWelcome = (message) => {

alert(message);

};

const onPress = () => {

alert("I was clicked");

};

return (

<div style={{ textAlign: 'center', marginTop: '40px' }}>

<h2>Counter: {count}</h2>

<button onClick={increment}>Increment</button>&nbsp;

<button onClick={decrement}>Decrement</button><br /><br />

<button onClick={() => sayWelcome("Welcome!")}>Say Welcome</button><br /><br />

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

</div>

);

}

export default Counter;

**2. Create Currency Converter Component (CurrencyConvertor.js)**

import React, { useState } from 'react';

function CurrencyConvertor() {

const [rupees, setRupees] = useState('');

const [euro, setEuro] = useState('');

const handleSubmit = () => {

const converted = (parseFloat(rupees) / 90).toFixed(2); // Example conversion

setEuro(converted);

};

return (

<div style={{ textAlign: 'center', marginTop: '40px' }}>

<h2>Currency Converter</h2>

<input

type="number"

value={rupees}

placeholder="Enter amount in ₹"

onChange={(e) => setRupees(e.target.value)}

/>

<button onClick={handleSubmit}>Convert</button>

{euro && <p>Amount in €: {euro}</p>}

</div>

);

}

export default CurrencyConvertor;

**3. App Component (App.js)**

import React from 'react';

import Counter from './components/Counter';

import CurrencyConvertor from './components/CurrencyConvertor';

function App() {

return (

<div className="App">

<h1 style={{ textAlign: 'center' }}>Event Examples App</h1>

<Counter />

<CurrencyConvertor />

</div>

);

}

export default App;

**4. CSS Styling (App.css)**

.App {

font-family: Arial, sans-serif;

padding: 20px;

}

input {

margin-right: 10px;

padding: 6px;

}

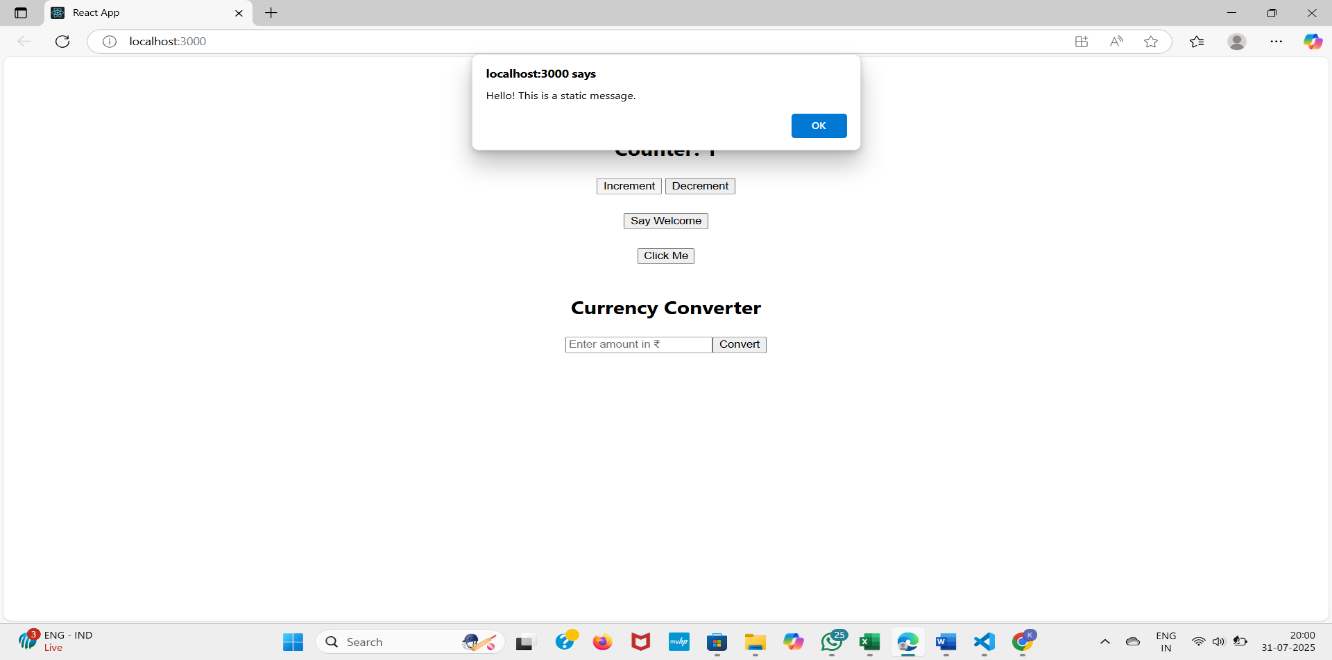
button {

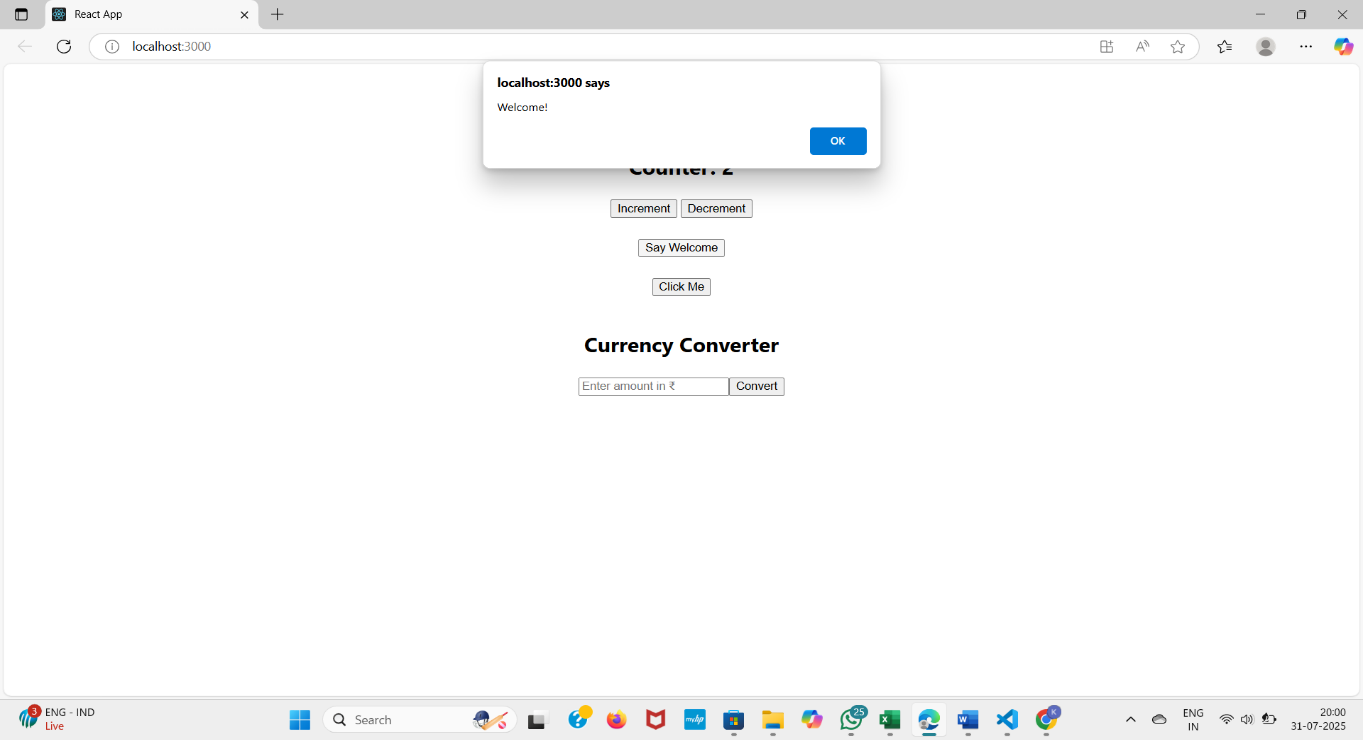
padding: 6px 10px;

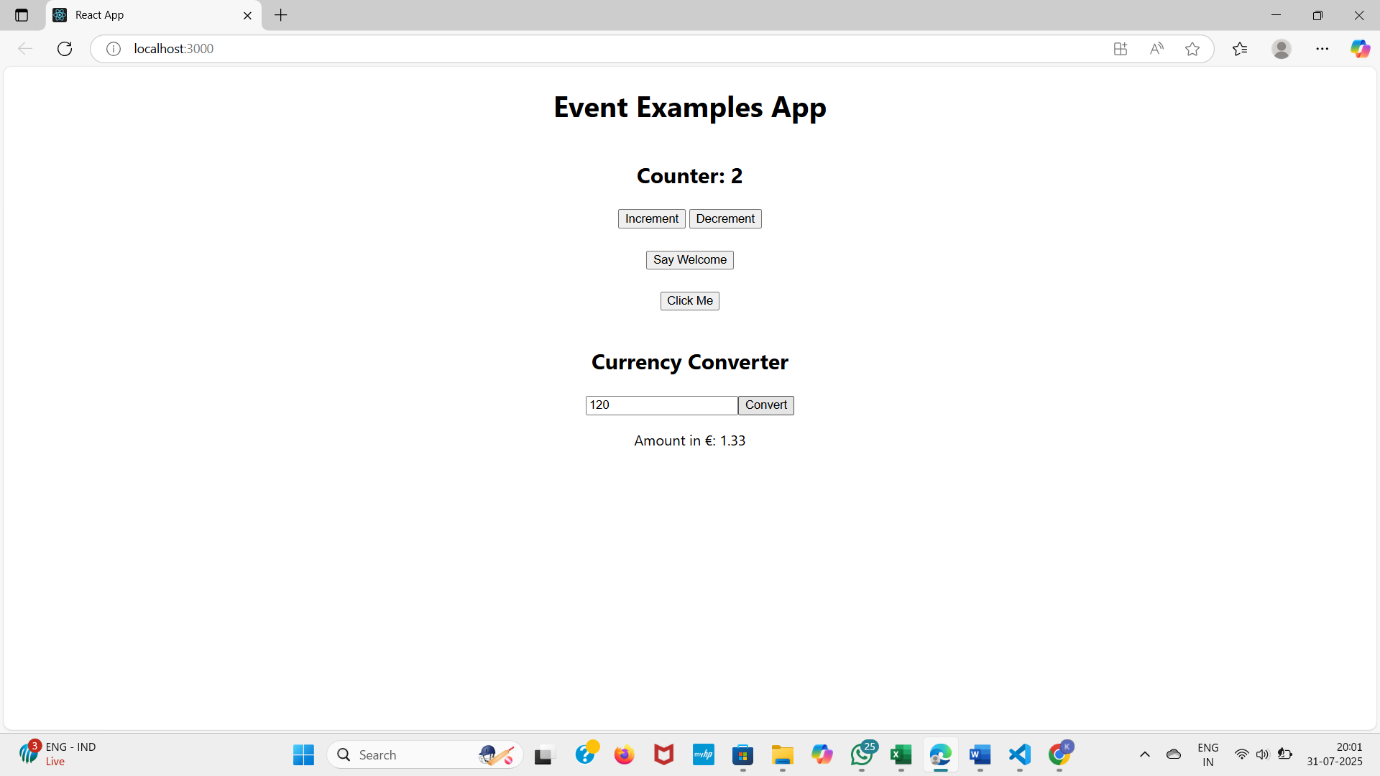
margin: 5px;

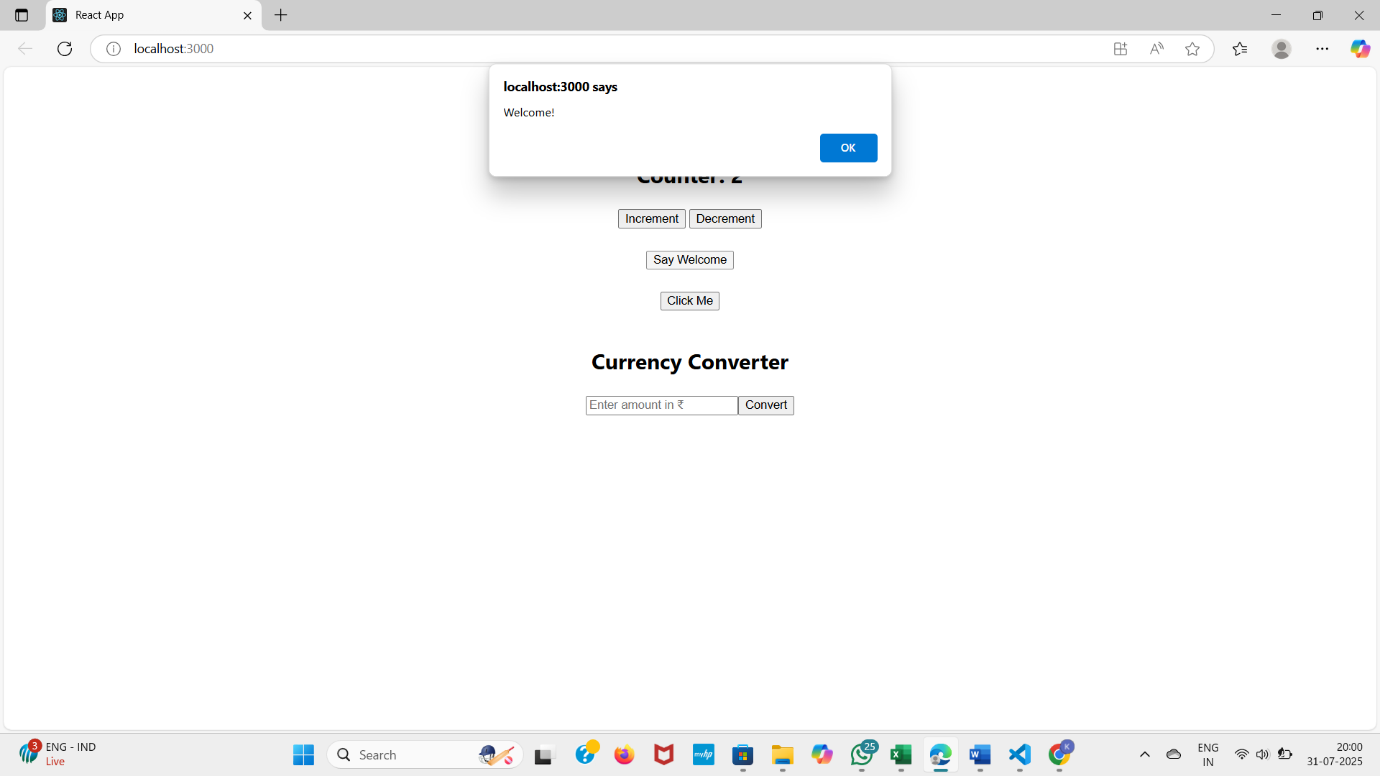
cursor: pointer;

}

**OUTPUT**







**ReactJS – HOL 12**

**Objectives**

* **Explain about conditional rendering in React**
* **Define element variables**
* **Explain how to prevent components from rendering**

**Conditional Rendering in React**

**Conditional Rendering Overview**

Conditional rendering in React allows you to display different UI elements based on certain conditions. There are several approaches to implement conditional rendering:

1. If/Else Statements

function Greeting({ isLoggedIn }) {

if (isLoggedIn) {

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

} else {

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

}

}

1. Ternary Operator (Conditional ? : )

function Greeting({ isLoggedIn }) {

return (

<div>

{isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}

</div>

);

}

3. Logical && Operator

function Notification({ messages }) {

return (

<div>

<h1>Messages</h1>

{messages.length > 0 &&

<p>You have {messages.length} unread messages.</p>

}

</div>

);

}

4. Switch Statements

function StatusIndicator({ status }) {

switch (status) {

case 'loading':

return <LoadingSpinner />;

case 'error':

return <ErrorDisplay />;

case 'success':

return <SuccessMessage />;

default:

return <DefaultStatus />;

}

}

**Element Variables**

Element variables are a way to store JSX elements in variables for conditional rendering:

function LoginControl({ isLoggedIn }) {

let button;

if (isLoggedIn) {

button = <LogoutButton />;

} else {

button = <LoginButton />;

}

return (

<div>

<Greeting isLoggedIn={isLoggedIn} />

{button}

</div>

);

}

Benefits of element variables:

* Keeps JSX clean and readable
* Allows complex conditional logic before rendering
* Can be reused in multiple places

Preventing Component Rendering

There are several ways to prevent a component from rendering:

1. Return null from render

function WarningBanner({ warn }) {

if (!warn) {

return null;

}

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

}

2. Short-circuit evaluation

function UserProfile({ user }) {

return (

<div>

{user && (

<div>

<h1>{user.name}</h1>

<p>{user.email}</p>

</div>

)}

</div>

);

}

3. Early return in component

function DataDisplay({ data }) {

if (!data) {

return <LoadingIndicator />;

}

if (data.error) {

return <ErrorDisplay error={data.error} />;

}

return <DataView data={data} />;

}

Important notes about preventing rendering:

* Returning null from a component's render method doesn't affect lifecycle methods
* The component still goes through its lifecycle (componentDidMount,
* componentDidUpdate, etc.)
* This approach is different from CSS display: none which hides the element but still renders it

4. Conditional rendering with children

function ConditionalWrapper({ condition, wrapper, children }) {

return condition ? wrapper(children) : children;

}

<ConditionalWrapper

condition={shouldWrap}

wrapper={children => <div className="wrapper">{children}</div>}

>

<Content />

</ConditionalWrapper>

These techniques give you fine-grained control over what gets rendered in your React components based on application state and props**.**

**Create a React Application named “ticketbookingapp” 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.**

**App.js**

import React, { useState } from "react";

import GuestPage from "./components/GuestPage";

import UserPage from "./components/UserPage";

import "./App.css";

function App() {

const [isLoggedIn, setIsLoggedIn] = useState(false);

const handleLoginToggle = () => {

setIsLoggedIn(!isLoggedIn);

};

return (

<div className="App">

{isLoggedIn ? (

<>

<UserPage />

<button onClick={handleLoginToggle}>Logout</button>

</>

) : (

<>

<GuestPage />

<button onClick={handleLoginToggle}>Login</button>

</>

)}

</div>

);

}

export default App;

**GuestPage.js**

import React from "react";

function GuestPage() {

return (

<div>

<h2>Please sign up.</h2>

<p>Here are the available flights:</p>

<ul>

<li> Chennai to Hyderabad - ₹2500</li>

<li> Delhi to Mumbai - ₹3200</li>

<li> Bangalore to Pune - ₹1800</li>

</ul>

</div>

);

}

export default GuestPage;

**UserPage.js**

import React from "react";

function UserPage() {

return (

<div>

<h2>Welcome back</h2>

<p>You can now book your flights.</p>

<button>Book Now</button>

</div>

);

}

export default UserPage;

**App.css (Styling)**

.App {

text-align: center;

margin-top: 100px;

font-family: 'Segoe UI', sans-serif;

}

button {

margin-top: 20px;

padding: 10px 16px;

font-size: 16px;

cursor: pointer;

background-color: #007bff;

color: white;

border: none;

border-radius: 6px;

}

button:hover {

background-color: #0056b3;

}

h2 {

font-size: 28px;

margin-bottom: 10px;

}

ul {

list-style-type: none;

padding-left: 0;

}

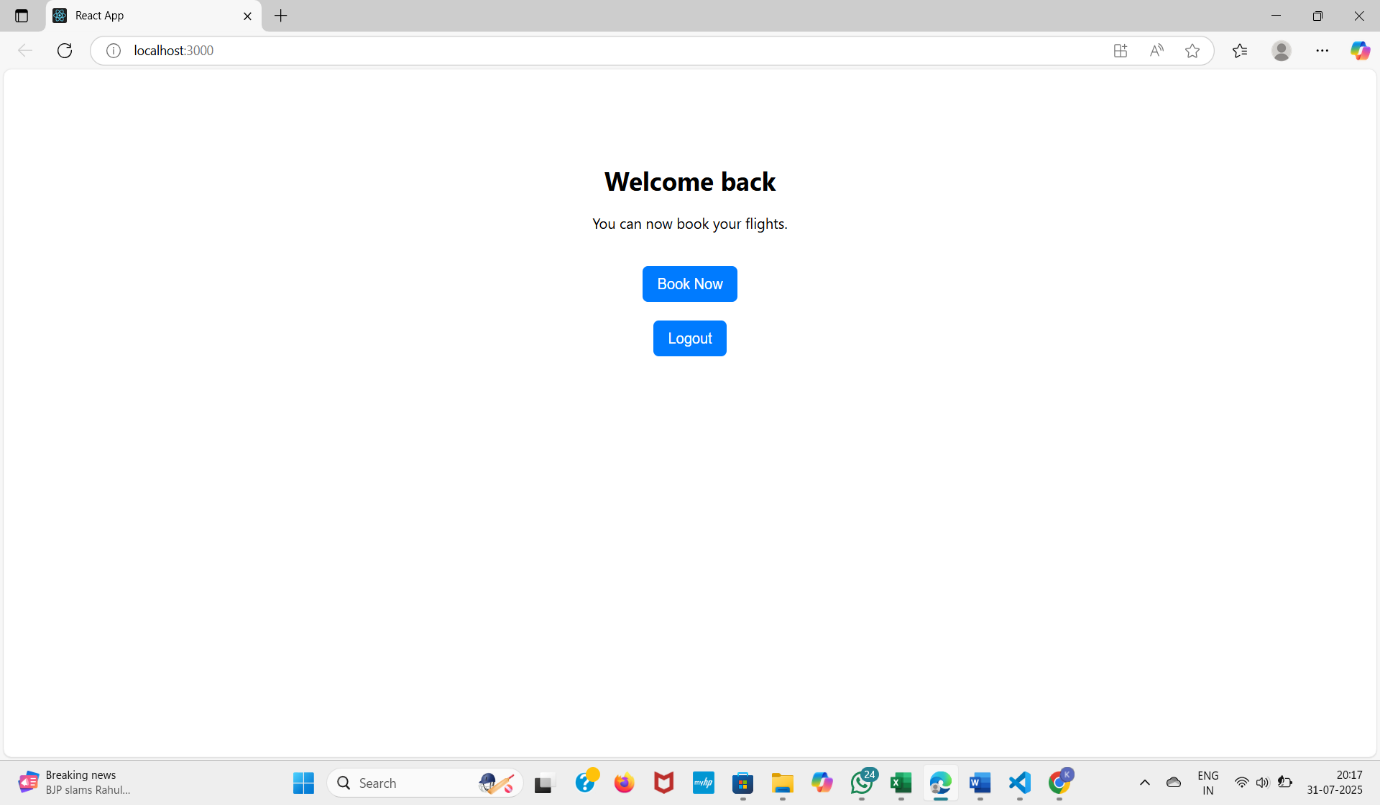
li {

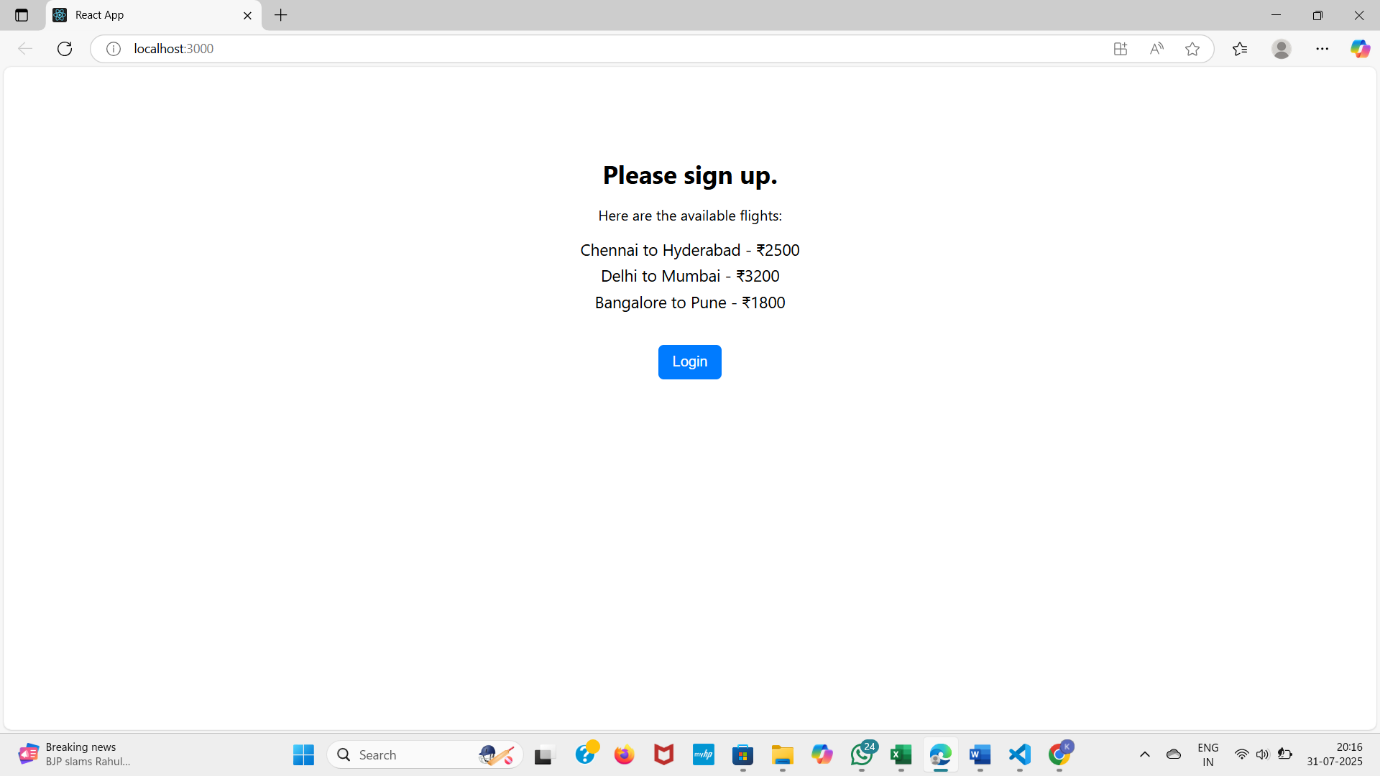
font-size: 18px;

margin: 5px 0;

}

**OUTPUT**





**ReactJS – HOL 13**

**Objectives**

* **Explain various ways of conditional rendering**
* **Explain how to render multiple components**
* **Define list component**
* **Explain about keys in React applications**
* **Explain how to extract components with keys**
* **Explain React Map, map() function**

React Conditional Rendering Cheatsheet

3 Ways to Render Conditionally

1. Ternary Operator (Best for simple either/or

{isLoggedIn ? <Dashboard /> : <Login />}

1. Logical && (Best for single condition)

{hasItems && <Cart items={cartItems} />}

1. Element Variables (Best for complex logic)

let content;

if (isLoading) content = <Spinner />;

else if (error) content = <Error />;

else content = <Data />;

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

Preventing Rendering

Return null to render nothing:

jsx

function Popup({ show }) {

if (!show) return null;

return <div className="popup">Message</div>;

}

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

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

**CourseDetails.js**

import React from 'react';

const CourseDetails = ({ show }) => {

if (!show) return null; // 1. Using `if` condition

const courses = [

{ name: "Angular", date: "4/5/2021" },

{ name: "React", date: "6/3/20201" },

];

return ( <div style={{ marginRight: "20px" }}> <h2>Course Details</h2>

{courses.map((course, idx) =>

course.name ? ( // 2. Ternary operator

<div key={idx}>

<strong>{course.name}</strong><br />

<span>{course.date}</span>

<br /><br />

</div>

) : null

)}

</div>

);

};

export default CourseDetails;

**BookDetails.js**

import React from 'react';

const BookDetails = ({ visible }) => {

return visible && ( // 3. Logical && operator

<div style={{ margin: "0 20px", borderLeft: "3px solid green", paddingLeft: "20px" }}>

<h2>Book Details</h2>

<p><strong>Master React</strong><br />670</p>

<p><strong>Deep Dive into Angular 11</strong><br />800</p>

<p><strong>Mongo Essentials</strong><br />450</p>

</div>

);

};

export default BookDetails;

**BlogDetails.js**

import React from 'react';

function BlogDetails(props) {

const { shouldRender } = props;

const renderBlog = () => { // 4. Using return from function

if (!shouldRender) return null;

return ( <div style={{ marginLeft: "20px", borderLeft: "3px solid green", paddingLeft: "20px" }}>

<h2>Blog Details</h2>

<div>

<strong>React Learning</strong><br />

<em>Stephen Biz</em><br />

<p>Welcome to learning React!</p>

</div>

<div>

<strong>Installation</strong><br />

<em>Schwezdenier</em><br />

<p>You can install React from npm.</p>

</div>

</div>

);

};

return renderBlog();

}

export default BlogDetails;

**App.js**

import React, { useState } from 'react';

import CourseDetails from './CourseDetails';

import BookDetails from './BookDetails';

import BlogDetails from './BlogDetails';

function App() {

const [showAll, setShowAll] = useState(true);

return ( <div style={{ display: "flex", justifyContent: "center", marginTop: "50px", fontFamily: "Arial" }}>

{/\* Conditional Rendering via props \*/}

<CourseDetails show={showAll} />

<BookDetails visible={showAll} />

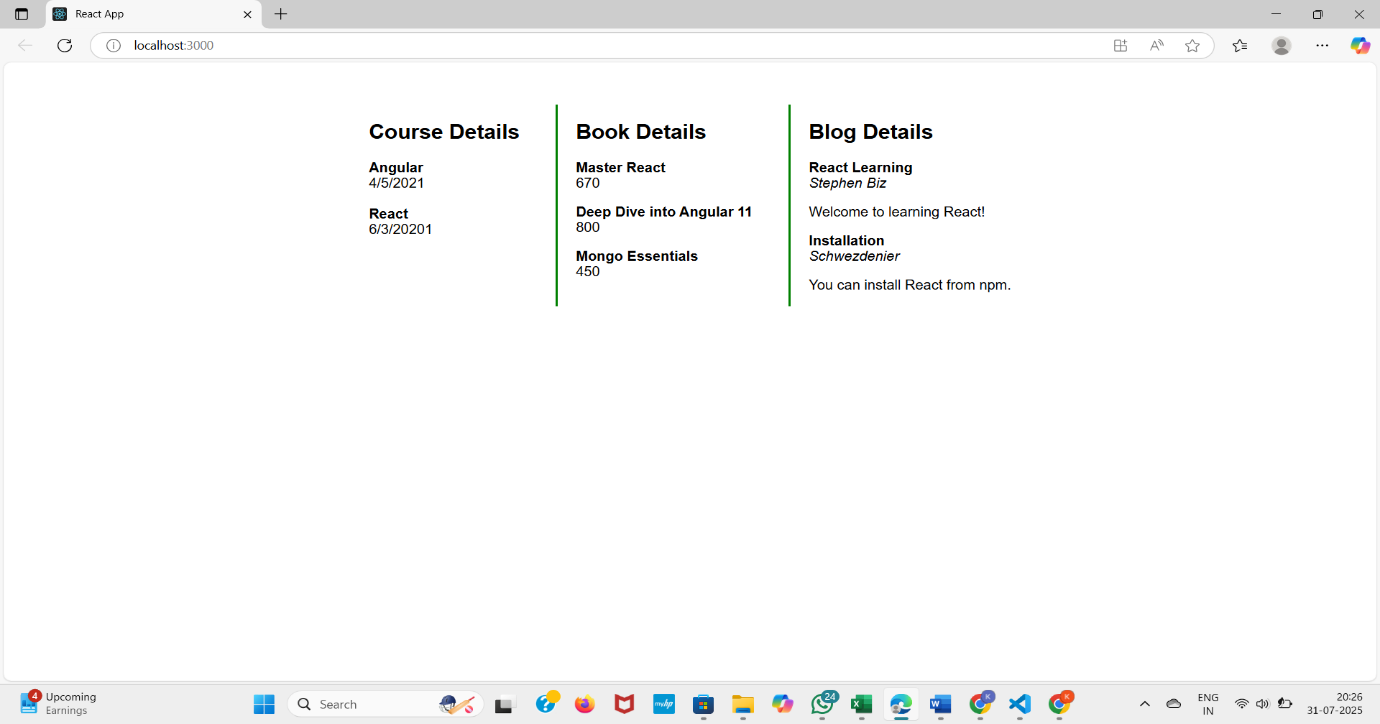
<BlogDetails shouldRender={showAll} />

</div>

);

}

export default App;

**OUTPUT**