**Student Name:** HARINEE N J

**Registration No:** 22CSR063

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**9. ReactJS-HOL**

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

ES6 brought revolutionary changes to JavaScript, making it more modern and developer-friendly. Key features include:

* Block-scoped variables with let and const for better variable management
* Arrow functions providing concise syntax and lexical this binding
* Template strings for easier string interpolation and multi-line text
* Destructuring assignment to extract data from arrays and objects efficiently
* Default function parameters to handle missing arguments gracefully
* Rest/Spread syntax for flexible function parameters and array operations
* ES6 Classes offering cleaner object-oriented programming syntax
* Module system with import/export for better code organization
* Enhanced object literals with shorthand properties and methods
* Promise API for better asynchronous programming
* New data types like Symbol, Map, Set, and WeakMap
* Generator functions for creating iterators with function\* syntax

**2. JavaScript let Keyword**

The let keyword declares block-scoped variables, providing better control over variable accessibility compared to var. Variables declared with let:

* Are confined to the block {} where they're declared
* Cannot be redeclared in the same scope
* Are hoisted but remain in a "temporal dead zone" until declaration

**Example**

if (true) {

let message = "Hello World";

console.log(message); => Hello World

}

console.log(message); => ReferenceError

**3. Differences between var and let**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **var** | **let** |
| **Scope** | Function-scoped or globally-scoped | Block-scoped |
| **Hoisting** | Hoisted and initialized with undefined | Avoid in modern JavaScript |
| **Redeclaration** | Allows redeclaration in same scope | Prevents redeclaration in same scope |
| **Temporal Dead Zone** | No TDZ | Has TDZ before declaration |
| **Best Practice** | Avoid in modern JavaScript | Preferred for mutable variables |

**4. JavaScript const Keyword**

The const keyword creates constants - variables that cannot be reassigned after declaration. Key characteristics:

* Block-scoped like let
* Must be initialized at declaration time
* The binding is immutable, but object/array contents can be modified
* Cannot be redeclared or reassigned

**Example**

const PI = 3.14159;

const user = { name: "Alice", age: 25 };

user.age = 26; // this is allowed

user = {}; // this is not allowed

**5. ES6 Class Fundamentals**

ES6 classes provide a cleaner, more intuitive syntax for creating objects and implementing inheritance. Classes are syntactic sugar over JavaScript's existing prototype-based inheritance. A class is created using the class keyword. It can have a constructor() method to initialize properties, and additional methods can be defined inside the class. Classes are not hoisted, and the syntax is more readable compared to older function-based prototypes.

**Example**

class Vehicle {

constructor(brand, model) {

this.brand = brand;

this.model = model;

}

getInfo() {

return `${this.brand} ${this.model}`;

}

static compareVehicles(v1, v2) {

return v1.brand === v2.brand;

}}

**6. ES6 Class Inheritance**

Class inheritance allows one class to inherit properties and methods from another using extends. The super() keyword calls the parent constructor and methods.

**Example**

class Animal {

constructor(species) {

this.species = species;

}

makeSound() {

return "Some generic animal sound";

}

}

class Cat extends Animal {

constructor(species, breed) {

super(species);

this.breed = breed;

}

makeSound() {

return "Meow";

}}

**7. ES6 Arrow Functions**

Arrow functions provide a concise way to write functions with automatic this binding from the enclosing scope. They're especially useful for callbacks and functional programming.

**Example**

const square = x => x \* x;

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

const processData = (data) => { const processed = data.map(x => x \* 2); return processed.filter(x => x > 10); };

**8. Set() and Map() Data Structures**

**Set()**: A collection that stores unique values of any type.

**Example**

const uniqueNumbers = new Set([1, 2, 2, 3, 3, 4]);

console.log(uniqueNumbers); => Set {1, 2, 3, 4}

uniqueNumbers.add(5);

console.log(uniqueNumbers.has(3)); => true

**Map()**: A collection of key-value pairs where keys can be any type.

**Example**

const userRoles = new Map();

userRoles.set('john123', 'admin');

userRoles.set('sarah456', 'editor');

userRoles.set(1, 'guest');

console.log(userRoles.get('john123')); => 'admin'

**Code:**

**App.js**

import React from 'react';

import './App.css';

import MenuList from './components/MenuList';

import RestaurantInfo from './components/RestaurantInfo';

function App() {

const showMenu = true; // Toggle between MenuList and RestaurantInfo

return (

<div className="App">

<header className="app-header">

<h1>Delicious Bites Restaurant</h1>

</header>

{showMenu ? <MenuList /> : <RestaurantInfo />}

</div>

);

}

export default App;

**MenuList.js**

import React from 'react';

const MenuList = () => {

const menuItems = [

{ name: 'Margherita Pizza', price: 299, category: 'Italian' },

{ name: 'Chicken Biryani', price: 349, category: 'Indian' },

{ name: 'Caesar Salad', price: 199, category: 'Healthy' },

{ name: 'Chocolate Brownie', price: 149, category: 'Dessert' },

{ name: 'Grilled Salmon', price: 449, category: 'Seafood' },

{ name: 'Veggie Burger', price: 229, category: 'American' },

{ name: 'Tom Yum Soup', price: 179, category: 'Thai' },

{ name: 'Tiramisu', price: 189, category: 'Dessert' },

{ name: 'Pad Thai', price: 269, category: 'Thai' },

{ name: 'Chicken Wings', price: 259, category: 'American' }

];

const affordableItems = menuItems.filter(item => item.price < 250);

return (

<div style={{ padding: '30px', fontFamily: 'Georgia, serif' }}>

<h2>Complete Menu</h2>

<div className="menu-grid">

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

<div key={index} className="menu-item">

<h3>{item.name}</h3>

<p>Category: {item.category}</p>

<p className="price">₹{item.price}</p>

</div>

))}

</div>

<h2>Budget-Friendly Options (Under ₹250)</h2>

<div className="affordable-list">

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

<div key={index} className="affordable-item">

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

<span className="price-tag">₹{item.price}</span>

</div>

))}

</div>

</div>

);};export default MenuList;

**RestaurantInfo.js**

import React from 'react';

const RestaurantInfo = () => {

const staff = ['Manager-Alex', 'Chef-Maria', 'Waiter-John', 'Cashier-Emma', 'Cook-David', 'Host-Lisa'];

const morningShift = staff.filter((\_, index) => index % 2 === 0);

const eveningShift = staff.filter((\_, index) => index % 2 !== 0);

const shiftLabels = ['First', 'Second', 'Third'];

const kitchenStaff = ['Head Chef', 'Sous Chef', 'Prep Cook'];

const frontStaff = ['Manager', 'Host', 'Server'];

const allStaff = [...kitchenStaff, ...frontStaff];

return (

<div style={{ padding: '30px', fontFamily: 'Arial, sans-serif' }}>

<h2>Morning Shift Team</h2>

<ul className="shift-list">

{morningShift.map((person, i) => (

<li key={i}>

<strong>{shiftLabels[i] || 'Additional'}:</strong> {person}

</li>

))}

</ul>

<h2> Evening Shift Team</h2>

<ul className="shift-list">

{eveningShift.map((person, i) => (

<li key={i}>

<strong>{shiftLabels[i] || 'Additional'}:</strong> {person}

</li>

))}</ul>

<hr />

<h2> Complete Staff Directory:</h2>

<ul className="staff-directory">

{allStaff.map((role, i) => (

<li key={i}>Position: {role}</li>

))}

</ul>

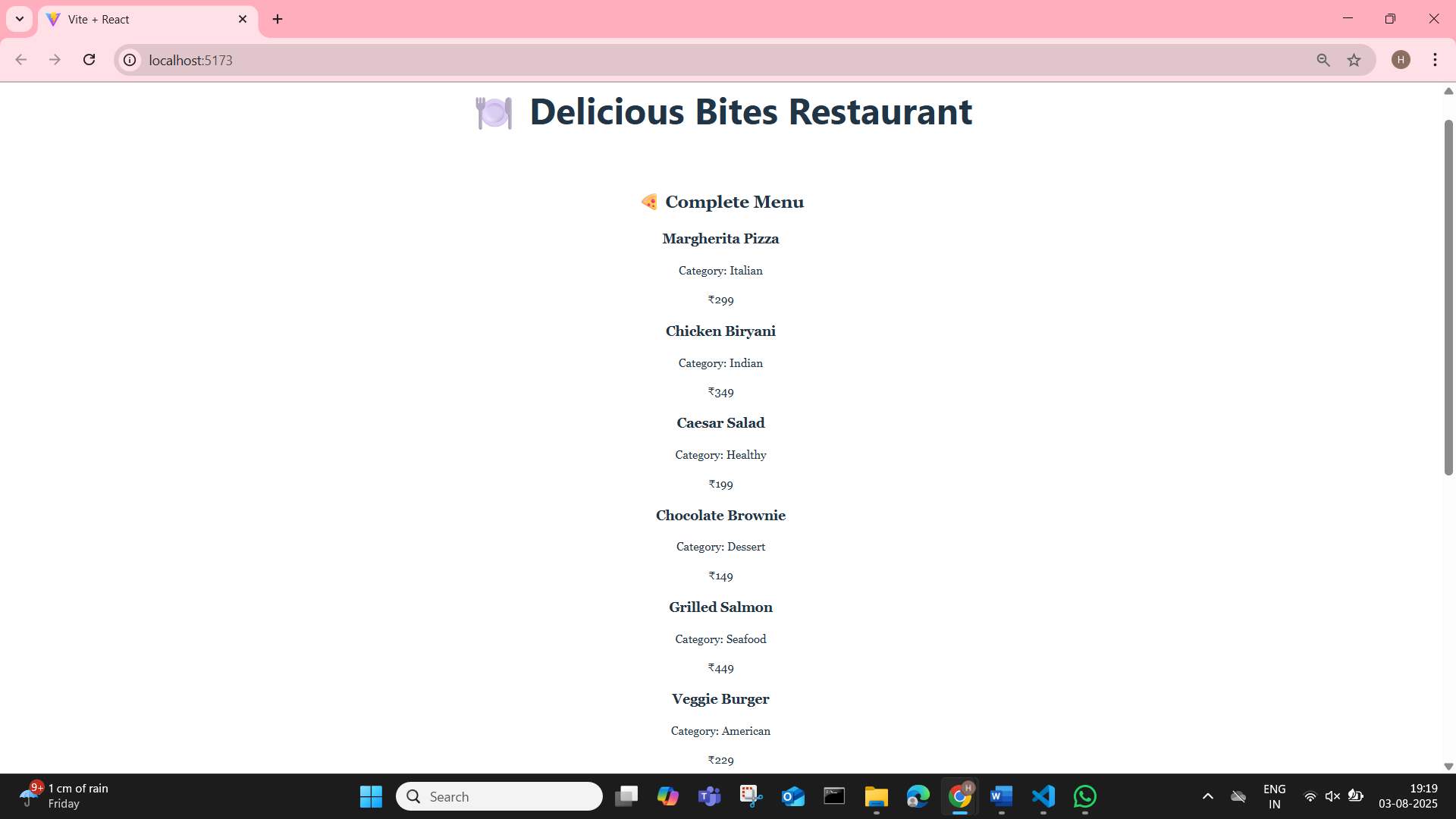
</div>

);};

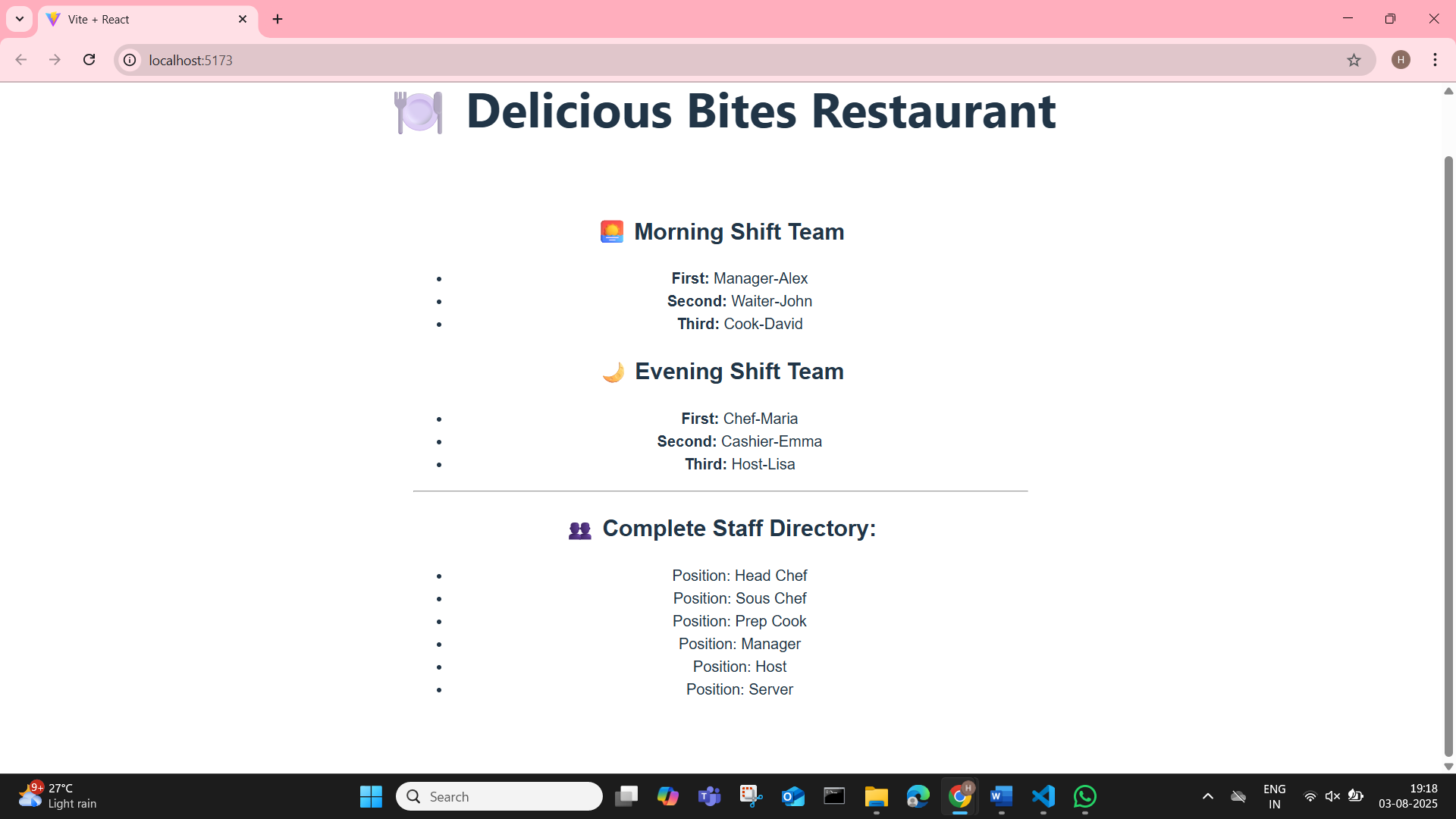
export default RestaurantInfo;

**OUTPUT**

When **showMenu = true**



When **showMenu = false**



**10. ReactJS-HOL**

**1. JSX Definition**

JSX (JavaScript XML/Extension) is a syntax extension for JavaScript that allows writing HTML-like code within JavaScript. It makes React components more readable and enables developers to describe UI structure declaratively. JSX gets transpiled to React.createElement() calls by tools like Babel.

**2. ECMAScript in React Context**

ECMAScript serves as the foundation for JavaScript, and React heavily utilizes modern ES6+ features. React applications commonly use ES6 modules, arrow functions, destructuring, classes, and async/await patterns. This integration makes React code more concise, maintainable, and follows modern JavaScript best practices.

**3. React.createElement() Method**

React.createElement() is the core function that creates React elements (virtual DOM nodes). It takes three parameters: element type, props object, and children. JSX is syntactic sugar that compiles to these function calls.

**Example**

React.createElement('div', { className: 'container' }, 'Hello World');

**4. Creating React Nodes with JSX**

React nodes are created by writing JSX expressions that describe the desired UI structure.

**Example**

const welcomeElement = <h1>Welcome to Our App</h1>;

const userGreeting = <p>Hello, {username}!</p>;

**5. Rendering JSX to DOM**

React 18+ uses createRoot() to render JSX into the browser DOM

**Example**

import { createRoot } from 'react-dom/client';

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

root.render(<App />);

**6. JavaScript Expressions in JSX**

Any valid JavaScript expression can be embedded in JSX using curly braces {}

**Example**

const username = "Alice";

const age = 28;

const element = <h1>Hello {username}, you are {age} years old!</h1>;

**7. Inline CSS in JSX**

CSS styles in JSX are written as JavaScript objects with camelCase properties

**Example**

const buttonStyle = {

backgroundColor: 'blue',

fontSize: '16px',

padding: '10px 20px'

};

const element = <button style={buttonStyle}>Click Me</button>;

**Code:**

**App.js**

import './App.css';

function App() {

const movieCollection = [

{

title: "Inception",

rating: 8.8,

genre: "Sci-Fi Thriller",

director: "Christopher Nolan",

poster: "https://images.unsplash.com/photo-1489599732522-c2efe209e5a6?w=300&h=400&fit=crop"

},

{

title: "The Godfather",

rating: 9.2,

genre: "Crime Drama",

director: "Francis Ford Coppola",

poster: "https://images.unsplash.com/photo-1440404653325-ab127d49abc1?w=300&h=400&fit=crop"

},

{

title: "Interstellar",

rating: 8.6,

genre: "Space Drama",

director: "Christopher Nolan",

poster: "https://images.unsplash.com/photo-1446776877081-d282a0f896e2?w=300&h=400&fit=crop"

}

];

return (

<div className="App">

<h1 className="main-title"> CineStream Platform</h1>

<div className="movie-gallery">

{movieCollection.map((movie, index) => (

<div key={index} className="movie-card">

<img src={movie.poster} alt={movie.title} className="movie-poster" />

<h2>{movie.title}</h2>

<p><strong>Director:</strong> {movie.director}</p>

<p><strong>Genre:</strong> {movie.genre}</p>

<p className={movie.rating >= 9.0 ? 'excellent-rating' : 'good-rating'}>

Rating: {movie.rating}/10

</p>

</div>

))}

</div>

</div>

);

}

export default App;

**App.css**

.App {

text-align: center;

padding: 25px;

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

min-height: 100vh;

}

.main-title {

font-size: 36px;

color: white;

margin-bottom: 30px;

text-shadow: 2px 2px 4px rgba(0,0,0,0.3);

}

.movie-gallery {

display: flex;

justify-content: center;

flex-wrap: wrap;

gap: 25px;

}

.movie-card {

background: white;

border-radius: 15px;

padding: 20px;

width: 280px;

box-shadow: 0 8px 25px rgba(0,0,0,0.15);

transition: transform 0.3s ease;

}

.movie-card:hover {

transform: translateY(-10px);

}

.movie-poster {

width: 100%;

height: 200px;

object-fit: cover;

border-radius: 10px;

margin-bottom: 15px;

}

.excellent-rating {

color: #28a745;

font-weight: bold;

font-size: 18px;

.good-rating {

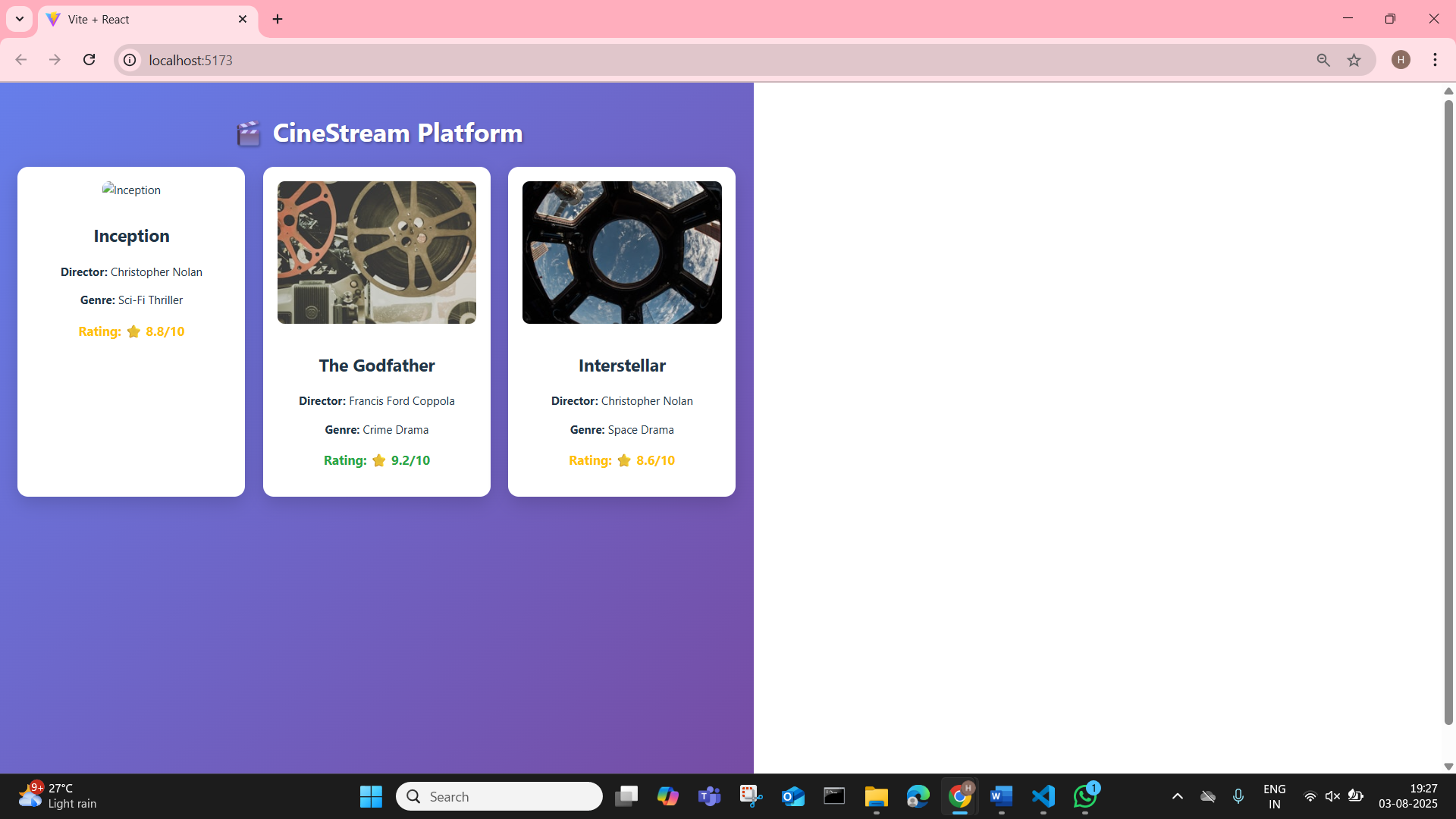
color: #ffc107;

font-weight: bold;

font-size: 18px;

}

**OUTPUT:**



**11. ReactJS-HOL**

**1. React Events Overview**

React events enable components to respond to user interactions like clicks, form submissions, keyboard input, and mouse movements. React implements a cross-browser event system called SyntheticEvents that wraps native DOM events, providing consistent behavior across different browsers and platforms.

**2. Event Handlers Explained**

Event handlers are functions that execute when specific events occur. In React, these are typically defined as methods within components or as separate functions, then attached to JSX elements via props.

**Example**

function handleButtonClick() {

console.log("Button was clicked!");

}

<button onClick={handleButtonClick}>Press Me</button>

**3. Synthetic Event Definition**

SyntheticEvent is React's wrapper around native browser events. It provides the same interface as native events but works consistently across all browsers. SyntheticEvents are pooled for performance optimization and automatically cleaned up after event handling.

**4. React Event Naming Conventions**

React follows camelCase naming for event handlers, differing from HTML's lowercase approach:

* React: onClick, onChange, onSubmit, onMouseOver
* HTML: onclick, onchange, onsubmit, onmouseover

Event handlers receive function references, not strings like in HTML.

**App.js**

import React, { useState } from 'react';

import './App.css';

function App() {

const [username, setUsername] = useState('');

const [greetMsg, setGreetMsg] = useState('');

const [hovered, setHovered] = useState(false);

const handleChange = (e) => {

setUsername(e.target.value);

};

const handleSubmit = (e) => {

e.preventDefault();

setGreetMsg(`Hello, ${username}!`);

setUsername('');

};

const handleMouseOver = () => {

setHovered(true);

};

const handleMouseOut = () => {

setHovered(false);

};

return (

<div className="container">

<h1>React Event Naming Demo</h1>

<form onSubmit={handleSubmit}>

<input

type="text"

placeholder="Enter your name"

value={username}

onChange={handleChange}

required

/>

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

</form>

{greetMsg && <p className="greet-msg">{greetMsg}</p>}

<div

className="hover-box"

onMouseOver={handleMouseOver}

onMouseOut={handleMouseOut}

>

{hovered ? "You're hovering over me!" : "Hover over this box"}

</div>

</div>

);

}

export default App;

**App.css**

body {

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

background-color: #eef2f3;

margin: 0;

padding: 0;

}

.container {

max-width: 500px;

margin: 50px auto;

background-color: white;

padding: 30px;

border-radius: 12px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

text-align: center;

}

input {

padding: 10px;

width: 60%;

margin-right: 10px;

border-radius: 5px;

border: 1px solid #ccc;

}

button {

padding: 10px 15px;

background-color: #28a745;

border: none;

color: white;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

.greet-msg {

font-size: 1.2em;

color: #333;

margin-top: 20px;

}

.hover-box {

margin-top: 30px;

padding: 20px;

background-color: #d1ecf1;

border: 2px dashed #0c5460;

border-radius: 10px;

transition: background-color 0.3s ease;

}

**12. ReactJS-HOL**

**1. Conditional Rendering Techniques**

Conditional rendering allows displaying different UI elements based on application state, user permissions, or other conditions. It works similarly to regular JavaScript conditions (like if, ternary ? :, or logical &&) and helps React decide what to display depending on the component's state or received props. React supports several approaches for implementing conditional logic in components.

**Example:**

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

Here, Dashboard is shown if isLoggedIn is true; otherwise, Login is displayed.

**2. Element Variables**

Element variables store JSX content in variables before rendering, helping organize complex conditional logic outside the return statement

**Example**

let statusMessage;

if (isOnline) {

statusMessage = <span className="online">Online</span>;

} else {

statusMessage = <span className="offline">Offline</span>;

}

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

**3. Explain how to prevent components from rendering**

React provides different ways to stop a component from rendering when it's not needed. One approach is to return null from a component — React will not render anything in this case. Another way is using conditional logic in JSX so that the component only renders under certain conditions.  
**Common ways include:**

* **Returning null in the component itself:**

if (!props.shouldDisplay) {

return null;

}

* **Using short-circuit logic:**

{isVisible && <MyComponent />}

* **Excluding it in the parent JSX entirely based on logic.**  
  This approach improves performance and avoids rendering unnecessary components in the UI.

**Code**

**App.js**

import React, { useState } from 'react';

import GuestView from './components/GuestView';

import UserView from './components/UserView';

import './App.css';

function App() {

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

const login = () => setIsLoggedIn(true);

const logout = () => setIsLoggedIn(false);

return (

<div className="App">

<header>

<h1>🏨 Hotel Booking App</h1>

{isLoggedIn ? (

<button className="logout-btn" onClick={logout}>Logout</button>

) : (

<button className="login-btn" onClick={login}>Login</button>

)}

</header>

{isLoggedIn ? <UserView /> : <GuestView />}

</div>);}export default App;

**HotelList.js**

import React from 'react';

import '../App.css';

const hotels = [

{ id: 1, name: 'Taj Palace', location: 'Mumbai', price: 8500, rating: 5 },

{ id: 2, name: 'ITC Grand', location: 'Chennai', price: 6500, rating: 4.5 },

{ id: 3, name: 'Oberoi', location: 'Delhi', price: 9000, rating: 5 }

];

function HotelList({ allowBooking }) {

return (

<div style={{ textAlign: 'center' }}>

<h3>Available Hotels</h3>

<div className="hotel-list">

{hotels.map(hotel => (

<div key={hotel.id} className="hotel-card">

<h4>{hotel.name}</h4>

<p>Location: {hotel.location}</p>

<p>Price per night: ₹{hotel.price}</p>

<p>Rating: {hotel.rating}</p>

{allowBooking && <button className="book-btn">Book Now</button>}

</div>

))}

</div>

</div>

);

}

export default HotelList;

**GuestView.js**

import React from 'react';

import HotelList from './HotelList';

function GuestView() {

return (

<div style={{ textAlign: 'center' }}>

<h2>Welcome Guest!</h2>

<p>Please log in to book a hotel room.</p>

<HotelList allowBooking={false} />

</div>

);

}

export default GuestView;

**UserView.js**

import React from 'react';

import HotelList from './HotelList';

function UserView() {

return (

<div style={{ textAlign: 'center' }}>

<h2>Welcome Back!</h2>

<HotelList allowBooking={true} />

</div>

);

}

export default UserView;

**App.css**

.App {

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

padding: 20px;

}

header {

text-align: center;

margin-bottom: 30px;

}

.login-btn,

.logout-btn {

padding: 10px 15px;

border: none;

color: white;

background-color: #007bff;

border-radius: 5px;

cursor: pointer;

}

.login-btn:hover,

.logout-btn:hover {

background-color: #0056b3;

}

.hotel-list {

display: flex;

gap: 20px;

flex-wrap: wrap;

justify-content: center;

padding: 10px;

}

.hotel-card {

background-color: #fff8dc;

border: 2px solid #daa520;

border-radius: 12px;

padding: 16px;

width: 250px;

text-align: center;

box-shadow: 2px 2px 10px rgba(0,0,0,0.1);

transition: transform 0.2s;

}

.hotel-card:hover {

transform: scale(1.05);

}

.book-btn {

margin-top: 10px;

background-color: #28a745;

color: white;

border: none;

padding: 8px 12px;

border-radius: 8px;

cursor: pointer;

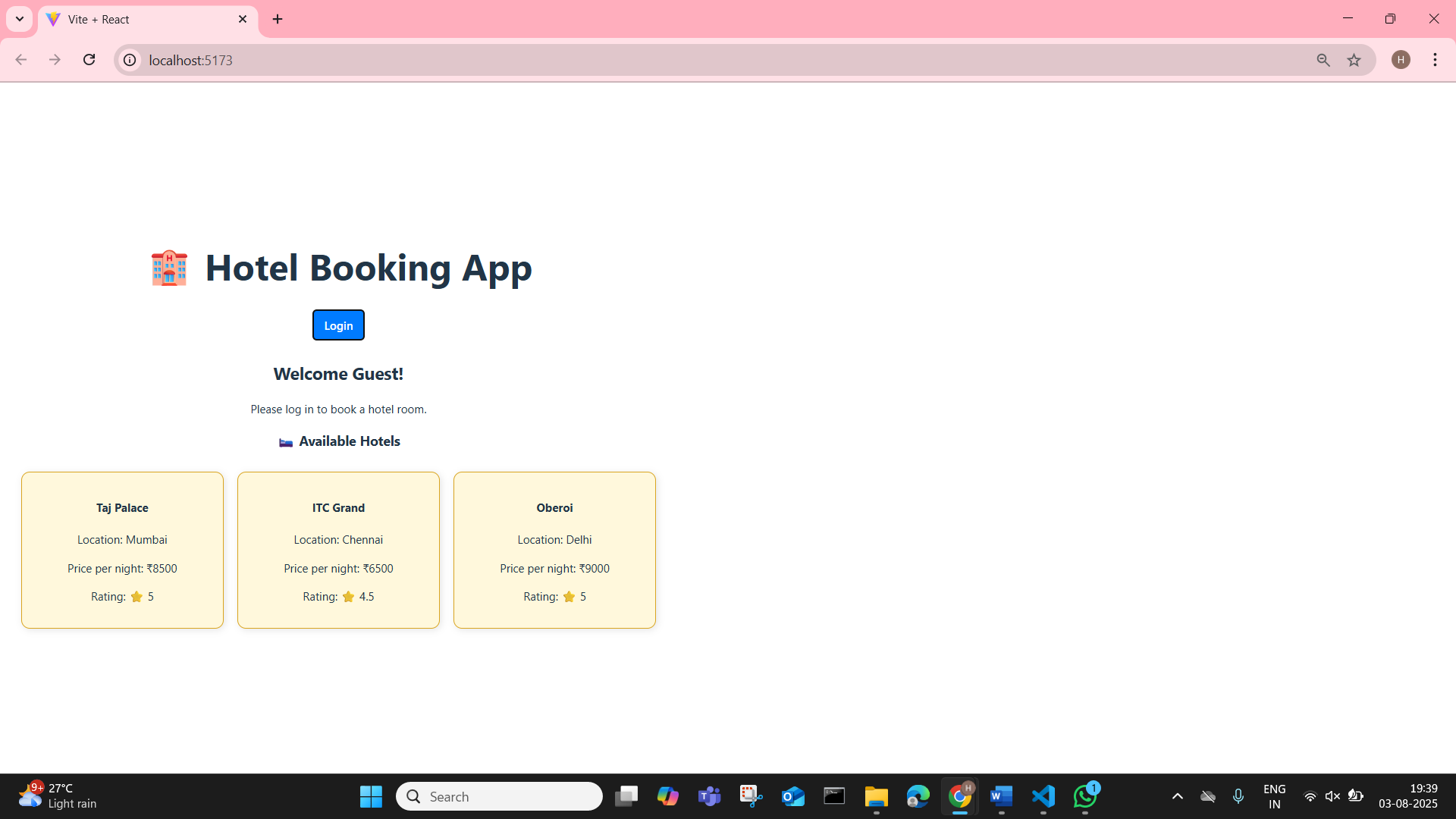
}

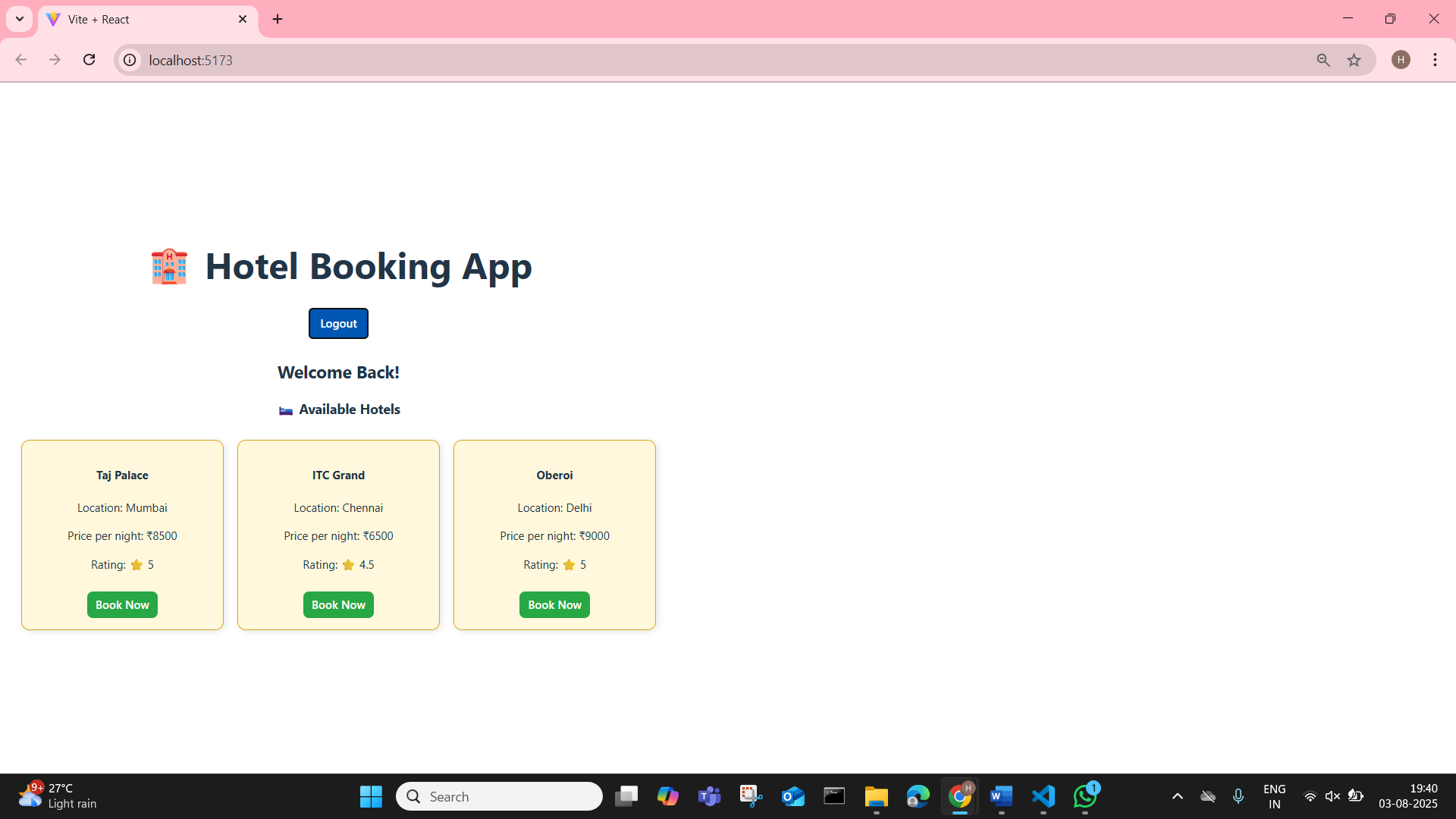
.book-btn:hover {

background-color: #218838;

}

**OUTPUT:**





**13. ReactJS-HOL**

**1.Explain various ways of conditional rendering**  
Conditional rendering means showing different UI elements based on conditions like login status, user role, etc. React supports multiple methods to handle this.

**Common ways:**

1. **If-Else Statement**  
   Good for more structured conditions before the return.  
   *Example:*

if (isLoggedIn) {

return <Dashboard />;

} else {

return <LoginPage />;

}

1. **Ternary Operator**  
   Useful inside JSX to choose between two elements.  
   *Example:*

{isAdmin ? <AdminPanel /> : <UserPanel />}

1. **Logical AND (&&)**  
   Render something only if a condition is true.  
   *Example:*

{showNotification && <Notification />}

1. **IIFE (Immediately Invoked Function)**  
   When conditions are complex but you want to use them directly in JSX.  
   *Example:*

{(() => {

if (status === 'loading') return <Loading />;

if (status === 'done') return <Done />;

return <Error />;

})()}

1. **Switch Statement**  
   Clean for handling multiple distinct values.  
   *Example:*

switch (view) {

case 'home': return <Home />;

case 'about': return <About />;

case 'contact': return <Contact />;

default: return null;

}

**2.Explain how to render multiple components**  
Rendering multiple components in React means displaying more than one component together within a single parent. This can be done in the following ways:

* Wrap the components in a single enclosing element like a div, section, or React.Fragment (<>...</>).
* Example:

function App() {

return (

<>

<Header />

<MainContent />

<Footer />

</>

);

}

This helps maintain a clean structure and keeps the component tree readable.

**3.Define list component**  
A list component is used to display a list of items using the map() function. Instead of hardcoding elements, you loop through an array and render each item dynamically.

*Example:*

function CityList({ cities }) {

return (

<ul>

{cities.map((city, index) => (

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

))}

</ul>

);

}

**4.Explain about keys in React applications**  
In React, keys are identifiers used when rendering lists to help the framework keep track of individual elements. They are essential for efficient updating and correct behavior during re-renders.

* A key tells React which items changed, were added, or removed.
* Each item in a list should have a unique key.
* The best choice for a key is a stable, unique ID from your data.
* Avoid using array indexes as keys unless there is no other option.

**Example:**

items.map(item => (

<Item key={item.id} value={item} />

))

Using keys correctly ensures that React updates only the changed elements instead of re-rendering the entire list.

**5.Explain how to extract components with keys**When rendering a list, you can split each item into its own component (this is called **component extraction**). In such cases, the key must be assigned **where the list is being mapped**, not inside the child component. This ensures React can track and update each component efficiently.

Assign key in the parent component that calls map(), not in the extracted child component.

**Example:**

function UserCard({ user }) {

return <div>{user.name}</div>;

}

function UserList({ users }) {

return (

<div>

{users.map(user => (

<UserCard key={user.id} user={user} />

))}

</div>

);

}

**6.Explain React Map, map() function**  
The map() function is used in React to iterate over an array and return a list of JSX elements. It's one of the most commonly used methods for rendering UI from dynamic data.

Always provide a unique key when rendering multiple elements using map().

**Different Example:**

const students = ['Arun', 'Beena', 'Chetan'];

const studentList = students.map((student, idx) => (

<div key={idx}>Student: {student}</div>

));

You can use this list inside a component like:

function Classroom() {

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

}

This will render each student’s name inside a div, and React will track each one using the provided key.

**Code:**

**App.js**

import React, { useState } from 'react';

import Gadgets from './components/Gadgets';

import Groceries from './components/Groceries';

import Clothing from './components/Clothing';

import './App.css';

function App() {

const [category, setCategory] = useState('');

return (

<div className="main-container">

<h1>Product Explorer</h1>

<div className="button-group">

<button onClick={() => setCategory('gadgets')}>Gadgets</button>

<button onClick={() => setCategory('groceries')}>Groceries</button>

<button onClick={() => setCategory('clothing')}>Clothing</button>

</div>

{category === 'gadgets' ? <Gadgets /> :

category === 'groceries' ? <Groceries /> :

category === 'clothing' ? <Clothing /> :

<p className="info-msg">Please choose a category to view products.</p>}

</div>

);}

export default App;

**Gadgets.js**

import React from 'react';

const Gadgets = () => {

return (

<div className="box gadgets-box">

<h2>Latest Gadgets</h2>

<ul>

<li>Smartphone - ₹25,000</li>

<li>Smartwatch - ₹5,000</li>

<li>Laptop - ₹55,000</li>

</ul>

</div>

);

};

export default Gadgets;

**Groceries.js**

import React from 'react';

const Groceries = () => {

return (

<div className="box groceries-box">

<h2>Daily Groceries</h2>

<ul>

<li>Apples - ₹120/kg</li>

<li>Milk - ₹55/litre</li>

<li>Rice - ₹45/kg</li>

</ul>

</div>

);};

export default Groceries;

**Clothing.js**

import React from 'react';

const Clothing = () => {

return (

<div className="box clothing-box">

<h2>Trendy Clothing</h2>

<ul>

<li>T-Shirt - ₹499</li>

<li>Jeans - ₹1,299</li>

<li>Jacket - ₹2,499</li>

</ul>

</div>

);

};

export default Clothing;

**App.css**

body {

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

background-color: #f9f9f9;

margin: 0;

padding: 20px;

}

.main-container {

text-align: center;

}

.button-group {

margin: 20px;

}

.button-group button {

margin: 0 10px;

padding: 10px 15px;

border: none;

border-radius: 8px;

cursor: pointer;

background-color: #007bff;

color: white;

font-size: 14px;

.button-group button:hover {

background-color: #0056b3;

}

.box {

margin: 30px auto;

width: 60%;

padding: 20px;

border-radius: 10px;

text-align: left;

.gadgets-box {

background-color: #e3f2fd;

border-left: 5px solid #2196f3;

}

.groceries-box {

background-color: #e8f5e9;

border-left: 5px solid #4caf50;

.clothing-box {

background-color: #fff3e0;

border-left: 5px solid #ff9800;

}

h2 {

color: #333;

}

ul {

list-style-type: square;

padding-left: 20px;

}

.info-msg {

color: gray;

font-size: 16px;

}

**OUTPUT:**

