COGNIZANT DN - 4.0 DEEP SKILLING

HANDS ON WEEK-7

REACT

1. **ReactJS-HOL**
2. **List the features of ES6.**

**Answer:**

The main features of ES6 are as follows:

**1. let and const**

* **let** allows block-scoped variable declaration.
* **const** declares constants (block-scoped and cannot be reassigned).

### ****2. Arrow Functions****

* Shorter syntax for writing functions and lexically binds this.

### ****3. Template Literals****

* Allows embedded expressions inside string literals using backticks `.

1. **Explain JavaScript let.**

**Answer:**

The let keyword in JavaScript is used to **declare variables** that are **block-scoped** (i.e., only accessible within the block {} where they are defined). It was introduced in **ES6** as an improvement over var.

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

**Answer:**

In JavaScript, both var and let are used to declare variables, but they differ significantly in terms of scope, hoisting, and redeclaration rules. The var keyword is function-scoped, meaning a variable declared with var is accessible throughout the entire function in which it is declared. On the other hand, let is block-scoped, which means it is only accessible within the block {} where it is defined. This makes let more predictable and safer in scenarios involving loops and conditional blocks.

1. **Explain JavaScript const.**

**Answe:**

The const keyword in JavaScript is used to **declare variables with a constant (unchangeable) value**. It was introduced in **ES6** and is **block-scoped**, just like let.

Some key features of javaScript const:

* 1. **Block Scope**
* Variables declared with const are accessible **only within the block** where they are defined.

2. **Must Be Initialized**

* You **must assign a value** to a const variable at the time of declaration.

1. **Explain ES6 class fundamentals.**

**Answer:**

ES6 introduced the class syntax to JavaScript as a cleaner and more familiar way to create objects and handle inheritance, similar to object-oriented programming languages like Java, C++, or Python. While it's mostly syntactic sugar over JavaScript’s existing **prototype-based inheritance**, it greatly improves readability and structure.

1. **Explain ES6 class inheritance.**

**Answer:**

ES6 introduced a more intuitive and cleaner way to implement **inheritance** in JavaScript using the class and extends keywords. Inheritance allows a class (called the **child** or **subclass**) to acquire the properties and methods of another class (called the **parent** or **superclass**).

**Basic syntax:**

class Parent {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hello, I'm ${this.name}`);

}

}

class Child extends Parent {

constructor(name, age) {

super(name);

this.age = age;

}

introduce() {

console.log(`I'm ${this.name} and I'm ${this.age} years old.`);

}

}

1. **Define ES6 arrow functions.**

**Answer:**

**Arrow functions** in ES6 (ECMAScript 2015) are a **concise way to write functions** using the => syntax. They provide a shorter syntax compared to traditional function expressions and also **lexically bind** this, making them especially useful in certain contexts like callbacks.

**Basic Syntax:**

// Traditional Function

function add(a, b) {

return a + b;

}

// Arrow Function

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

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

**Answer:**

### ****1.**** Set

A **Set** is a collection of **unique values**, meaning it automatically removes duplicates.

**Basic Syntax:**

const mySet = new Set();

mySet.add(1);

mySet.add(2);

mySet.add(2);

console.log(mySet);

### ****2.**** Map

A **Map** is a collection of **key-value pairs** where **keys can be of any type**, unlike objects which only allow string or symbol keys.

**Basic Syntax:**

const myMap = new Map();

myMap.set("name", "Ayushmaan");

myMap.set(1, "one");

console.log(myMap); // Map { 'name' => 'Ayushmaan', 1 => 'one' }

**Practical Question 1:**

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

**ListOfPlayers.js**

*import* *React* *from* '*react*';

*const* ListofPlayers = () *=>* {

*const* players = [

    { *name*: '*Virat*', *score*: 95 },

    { *name*: '*Rohit*', *score*: 82 },

    { *name*: '*Dhoni*', *score*: 60 },

    { *name*: '*KL Rahul*', *score*: 70 },

    { *name*: '*Jadeja*', *score*: 50 },

    { *name*: '*Shami*', *score*: 40 },

    { *name*: '*Bumrah*', *score*: 75 },

    { *name*: '*Ashwin*', *score*: 68 },

    { *name*: '*Surya*', *score*: 85 },

    { *name*: '*Gill*', *score*: 90 },

    { *name*: '*Pant*', *score*: 72 },

  ];

*const* allPlayers = players.map((*player*, *index*) *=>* (

    <li *key*=*{*index*}*>*{*player.name*}* - *{*player.score*}*</li>

  ));

*const* lowScorers = players.filter(*p* *=>* p.score *<* 70).map((*player*, *index*) *=>* (

    <li *key*=*{*index*}*>*{*player.name*}* - *{*player.score*}*</li>

  ));

*return* (

    <div>

      <h2>All Players</h2>

      <ul>*{*allPlayers*}*</ul>

      <h2>Players with score below 70</h2>

      <ul>*{*lowScorers*}*</ul>

    </div>

  );

};

*export* *default* ListofPlayers;

**IndianPlayers.js**

*import* *React* *from* '*react*';

*const* IndianPlayers = () *=>* {

*const* T20 = ['*Virat*', '*Rohit*', '*Surya*', '*Gill*', '*Pant*'];

*const* Ranji = ['*Pujara*', '*Saha*', '*Manoj*', '*Iyer*', '*Karun*'];

*const* allPlayers = [*...*T20, *...*Ranji];

*const* [p1, p2, p3, p4, p5, p6, p7, p8, p9, p10] = allPlayers;

*const* oddTeam = [p1, p3, p5, p7, p9];

*const* evenTeam = [p2, p4, p6, p8, p10];

*return* (

    <div>

      <h2>Odd Team Players</h2>

      <ul>*{*oddTeam.map((*p*, *i*) *=>* <li *key*=*{*i*}*>*{*p*}*</li>)*}*</ul>

      <h2>Even Team Players</h2>

      <ul>*{*evenTeam.map((*p*, *i*) *=>* <li *key*=*{*i*}*>*{*p*}*</li>)*}*</ul>

    </div>

  );

};

*export* *default* IndianPlayers;

**App.js**

*import* *React* *from* '*react*';

*import* *ListofPlayers* *from* '*./ListofPlayers*';

*import* *IndianPlayers* *from* '*./IndianPlayers*';

*function* App() {

*const* flag = *true*; // *change to false to test both views*

*return* (

    <div *className*="*App*">

      <h1> Cricket App</h1>

*{*flag ? <ListofPlayers /> : <IndianPlayers />*}*

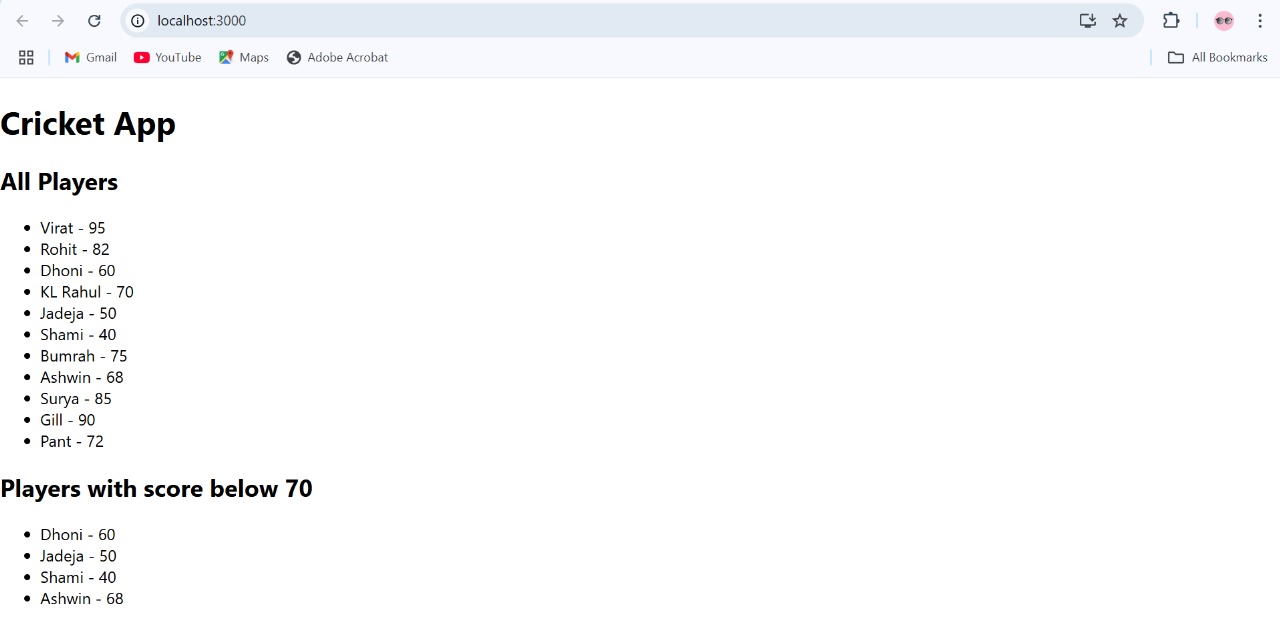
    </div>

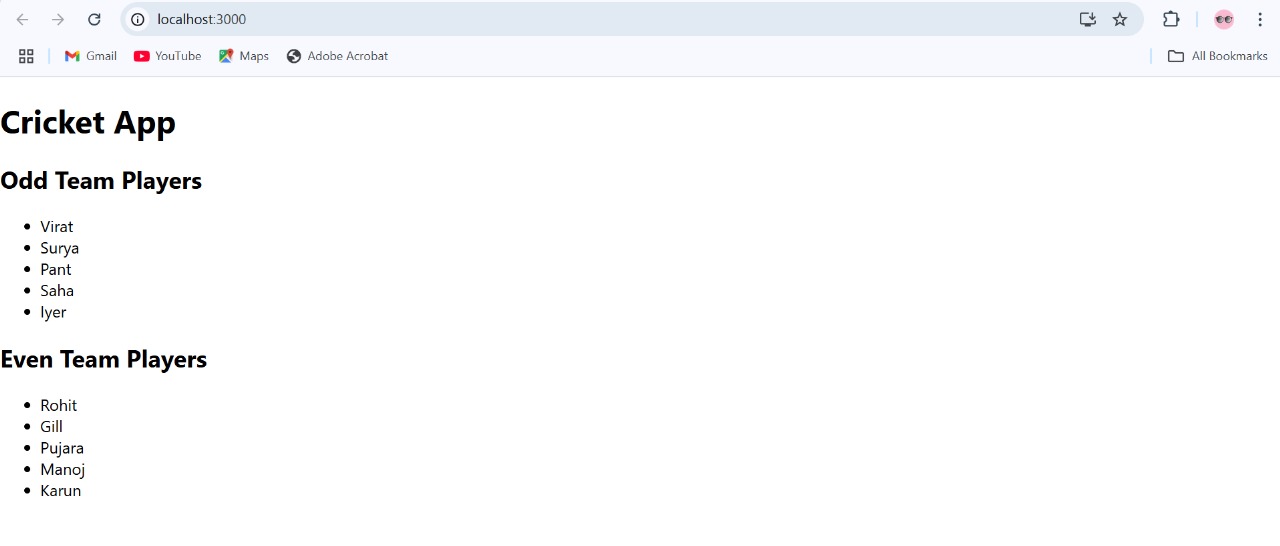
  );

}

*export* *default* App;

**Output:**

****



1. **ReactJS-HOL**
   1. **Define JSX.**

**Answer:**

**JSX** stands for **JavaScript XML**. It is a **syntax extension** for JavaScript used in **React** to describe what the UI should look like. JSX allows you to **write HTML-like code directly within JavaScript**, making your code more readable and expressive.

* 1. **Explain about ECMA Script.**

**Answer:**

**ECMAScript (ES)** is the **standardized scripting language specification** that serves as the foundation for JavaScript. It defines how the language should work so that different browsers and environments can implement it consistently.

* 1. **Explain React.createElement().**

**Answer:**

React.createElement() is a **core function in React** used to **create React elements**. These elements are the **building blocks of the React UI**, and they describe what should be rendered on the screen.

**Basic Syntax:**

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

* 1. **Explain how to create React nodes with JSX.**

**Answer:**

To create **React nodes with JSX,** we simply write HTML like syntax directly inside JavaScript. These JSX expressions are compiled into React nodes using React.createElement().

Example:

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

* 1. **Define how to render JSX to DOM.**

**Answer:**

In React, you render JSX to the **DOM** using the ReactDOM.createRoot() and root.render() methods. JSX defines **what to render**, and ReactDOM tells **where to render** it in the HTML page.

**Basic Syntax:**

import React from 'react';

import ReactDOM from 'react-dom/client';

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

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

root.render(element);

* 1. **Explain how to use JavaScript expressions in JSX.**

**Answer:**

JSX allows you to **embed JavaScript expressions inside curly braces {}**. This makes it easy to write dynamic and interactive user interfaces directly within HTML-like syntax.

**Basic Syntax:**

const name = "Ayushmaan";

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

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

**Answer:**

In JSX (used in React), you can apply inline styles using a **JavaScript object** rather than a string like in HTML. This gives more power and flexibility, especially when styles depend on logic or variables.

**Basic Inline Syntax:**

const element = (

<h1 style={{ color: "blue", backgroundColor: "lightyellow" }}>

Hello, Inline CSS!

</h1>

);

**Practical Question 1:**

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

**App.js**

*import* *React* *from* '*react*';

*function* App() {

*const* office = {

*name*: "*COGNIZANT*",

*rent*: 50000,

*address*: "*Chennai*",

    image: “SORRY!! The Chrome URL was Way too long so I removed it but you can check the image in the output attached ”

  };

*const* rentStyle = {

*color*: office.rent *<* 60000 ? "*red*" : "*green*",

*fontWeight*: "*bold*"

  };

*return* (

    <div *style*=*{*{ *textAlign*: "*center*", *fontFamily*: "*Arial, sans-serif*", *marginTop*: "*20px*" }*}*>

      <h1>Office Space , at Affordable Range</h1>

      <img *src*=*{*office.image*}* *alt*="*Office*" *width*="*400*" />

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

      <p *style*=*{*rentStyle*}*>Rent: Rs. *{*office.rent*}*</p>

      <p>Address: *{*office.address*}*</p>

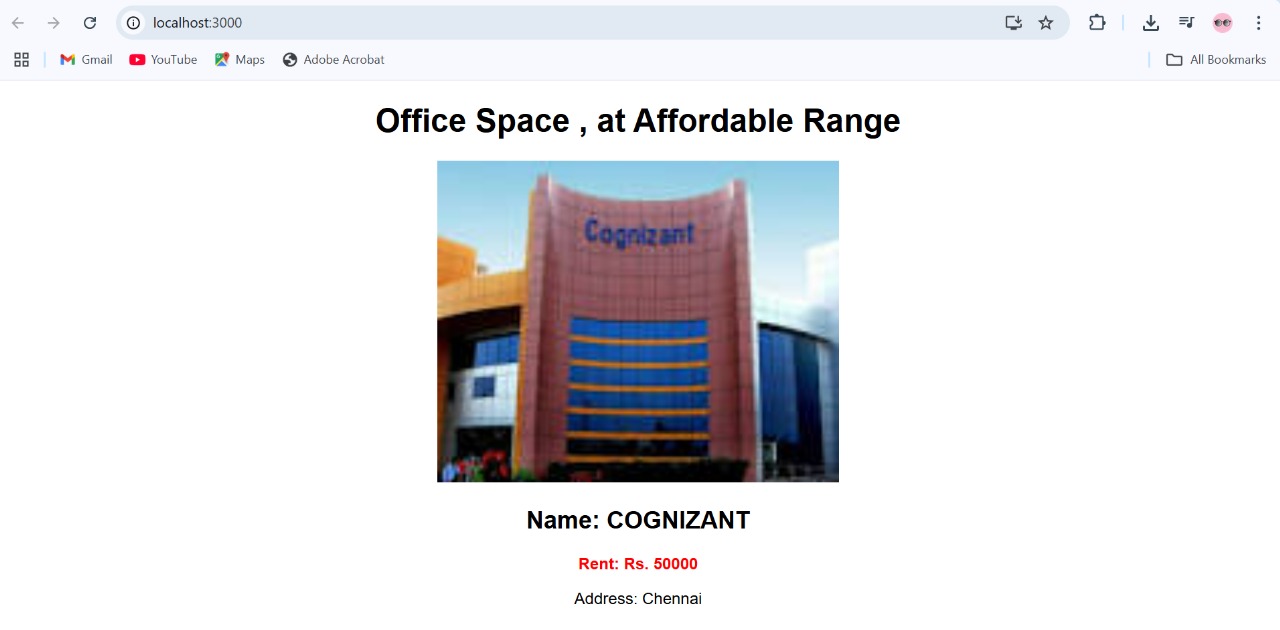
    </div>

  );

}

*export* *default* App;

**Output:**

****

1. **ReactJS-HOL**
   1. **Explain React events.**

**Answers:**

**React events** are **synthetic events** that wrap around the browser’s native events to provide **consistent behavior across different browsers**. React uses **camelCase** syntax and **function handlers** to respond to events like clicks, key presses, form submissions, and more.

* 1. **Explain about event handlers.**

**Answers:**

**Event handlers** in React are **functions that respond to user interactions or system events** like clicking a button, typing in an input field, or submitting a form.

**Basic Syntax:**

function handleClick() {

alert("Button was clicked!");

}

* 1. **Define Synthetic event.**

**Answer:**

In **React**, a **Synthetic Event** is a **cross-browser wrapper** around the browser’s native event system. It is part of React’s event delegation model and ensures that events behave consistently across all browsers.React creates a SyntheticEvent **object** for every event, such as clicks, form submissions, key presses, etc.

**Basic Syntax:**

function App() {

const handleClick = (event) => {

console.log("Synthetic Event:", event);

console.log("Event type:", event.type); // 'click'

};

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

}

* 1. **Identify React event naming convention,**

**Answer:**

In react we use camelCase instead of lower case.

Example: onClick, onChange, onSubmit

**Practical Question 1:**

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

**App.js**

*import* *React*, { *useState* } *from* '*react*';

*function* App() {

*const* [count, setCount] = useState(5);

*const* [amount, setAmount] = useState('');

*const* [currency, setCurrency] = useState('');

*const* handleIncrement = () *=>* {

    setCount(*prev* *=>* prev + 1);

  };

*const* sayHello = () *=>* {

    alert("*Hello! Ayushmaan*");

  };

*const* handleDecrement = () *=>* {

    setCount(*prev* *=>* prev - 1);

  };

*const* sayWelcome = (*message*) *=>* {

    alert(`*Hello!* ${*message*}`);

  };

*const* handleSyntheticEvent = (*e*) *=>* {

    e.preventDefault();

    alert("*I was clicked*");

  };

*const* handleSubmit = (*e*) *=>* {

    e.preventDefault();

*if* (!amount || isNaN(amount)) {

      alert("*Please enter a valid amount*");

*return*;

    }

*const* converted = (parseFloat(amount) / 90).toFixed(2);

    alert(`*₹*${*amount*} *= €*${*converted*}`);

  };

*return* (

    <div *style*=*{*{ *margin*: "*30px*", *fontFamily*: "*Arial*" }*}*>

      <p>*{*count*}*</p>

      <button *onClick*=*{*handleIncrement*}*>Increment</button><br /><br />

      <button *onClick*=*{*handleDecrement*}*>Decrement</button><br /><br />

      <button *onClick*=*{*() *=>* sayWelcome("*Member1*")*}*>Say welcome</button><br /><br />

      <button *onClick*=*{*handleSyntheticEvent*}*>Click on me</button><br /><br />

      <h2 *style*=*{*{ *color*: "*green*" }*}*>Currency Convertor!!!</h2>

      <form *onSubmit*=*{*handleSubmit*}*>

        <label>

          Amount: <input *type*="*text*" *value*=*{*amount*}* *onChange*=*{e* *=>* setAmount(e.target.value)*}* />

        </label>

        <br /><br />

        <label>

          Currency: <input *type*="*text*" *value*=*{*currency*}* *onChange*=*{e* *=>* setCurrency(e.target.value)*}* />

        </label>

        <br /><br />

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

      </form>

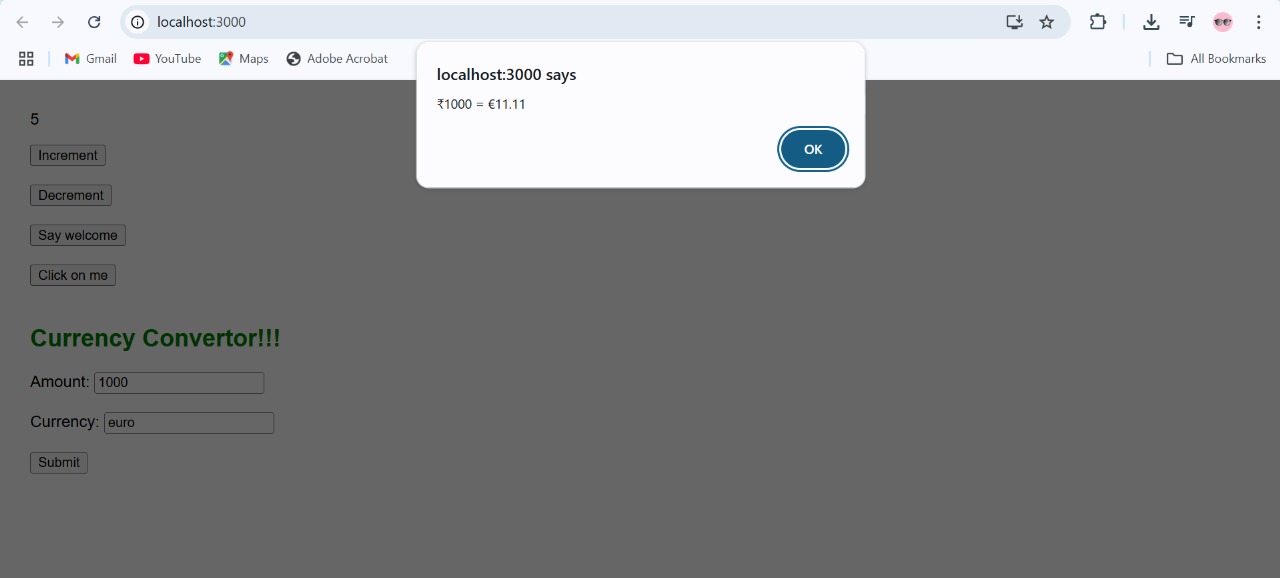
    </div>

  );

}

*export* *default* App;

**Output:**

****

1. **ReactJS-HOL**
2. **Explain about conditional rendering in React.**

**Answer:**

**Conditional rendering** in React means **dynamically showing or hiding parts of the UI** based on certain conditions (like user input, state, or props). It’s similar to using if, else, and ternary operators in JavaScript.

**Using if statement:**

function App() {

const isLoggedIn = true;

let message;

if (isLoggedIn) {

message = <h1>Welcome back!</h1>;

} else {

message = <h1>Please log in.</h1>;

}

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

}

1. **Define element variables.**

**Answer:**

**Element variables** in React are **JavaScript variables** that are used to **store JSX elements**, which can then be conditionally or repeatedly rendered inside the component’s return statement.

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

**Answer:**

Some ways to prevent component from rendering are:

### ****Return**** null ****from a Component****

### Returning null from a component means it renders nothing.

### ****2. Use Conditional Rendering in JSX****

Using conditions inside the parent component to decide whether or not to include the child component.

**Practical Question1:**

**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.**

**App.js**

*import* *React*, { *useState* } *from* '*react*';

*function* GuestPage() {

*return* (

    <div>

      <h2>Welcome, Guest!</h2>

      <p>Explore available flight options below:</p>

      <ul>

        <li> Delhi to Mumbai - ₹4500</li>

        <li> Chennai to Bangalore - ₹3200</li>

        <li> Kolkata to Goa - ₹5000</li>

      </ul>

    </div>

  );

}

*function* UserPage() {

*return* (

    <div>

      <h2>Welcome, User!</h2>

      <p>You can now book your tickets:</p>

      <button>Book Now</button>

    </div>

  );

}

*function* App() {

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

*let* pageContent;

*if* (isLoggedIn) {

    pageContent = <UserPage />;

  } *else* {

    pageContent = <GuestPage />;

  }

*return* (

    <div *style*=*{*{ *textAlign*: '*center*', *padding*: '*20px*' }*}*>

      <h1> Ticket Booking App</h1>

*{*pageContent*}*

      <br />

*{*/\* *Conditional rendering of buttons* \*/*}*

*{*isLoggedIn ? (

        <button *onClick*={() *=>* setIsLoggedIn(*false*)}>Logout</button>

      ) : (

        <button *onClick*={() *=>* setIsLoggedIn(*true*)}>Login</button>

      )*}*

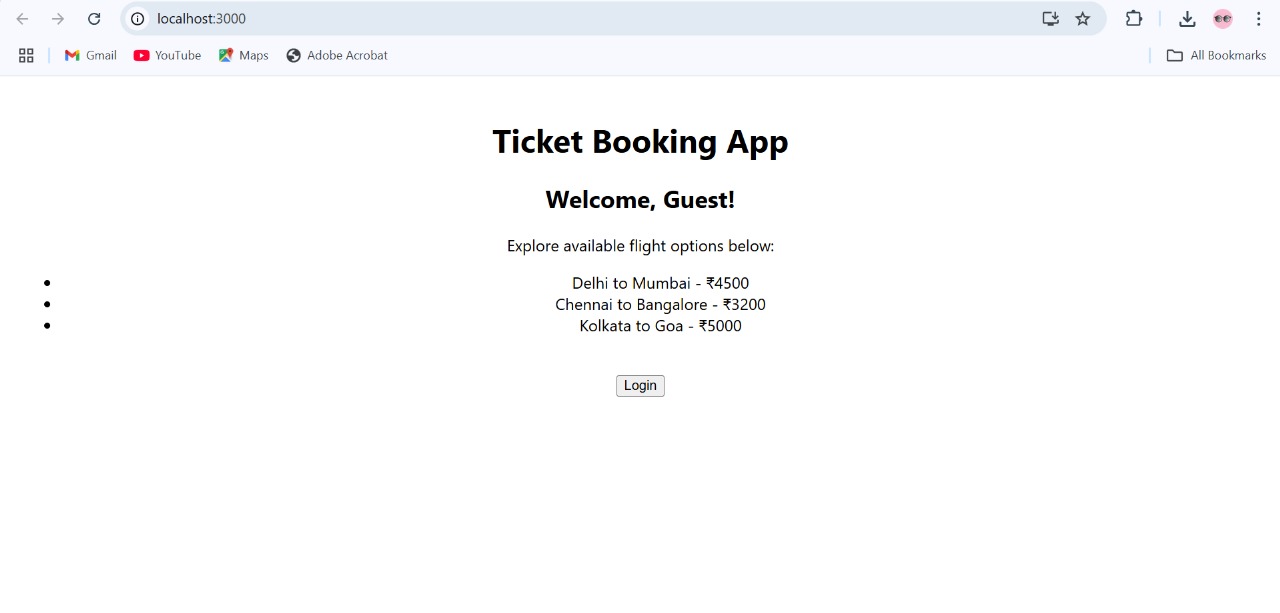
    </div>

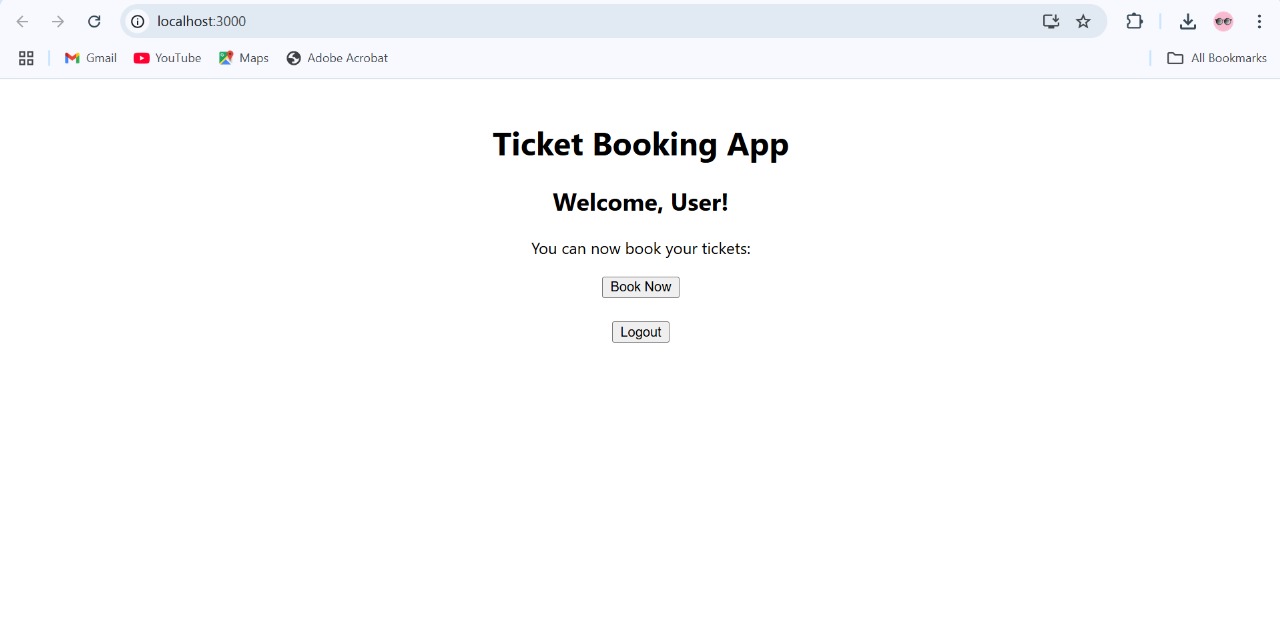
  );

}

*export* *default* App;

**Output:**

****



1. **ReactJS-HOL**
   1. **Explain various ways of conditional rendering.**

**Answer:**

In React, **conditional rendering** is the technique of showing or hiding components or elements based on certain conditions (like if, boolean, state, etc.). It helps create dynamic and responsive user interfaces.

**If statement:**

function App() {

const isLoggedIn = true;

let content;

if (isLoggedIn) {

content = <h2>Welcome, User!</h2>;

} else {

content = <h2>Please log in</h2>;

}

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

}

**Using ternery operator:**

function App() {

const isMember = false;

return (

<div>

{isMember ? <p>Member Access</p> : <p>Guest Access</p>}

</div>

);

}

* 1. **Explain how to render multiple components.**

**Answer:**

In React, you can render multiple components together by including them inside a **parent element** like a <div>, <> (Fragment), or another component. This is useful when building modular and reusable UI structures.

* 1. **Define list component.**

**Answer:**

A **List Component** in React is a component that is used to **display a collection of items** (like names, products, tasks, etc.) by **looping through an array** and rendering JSX for each item.

**Example:**

function NameList() {

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

return (

<ul>

{names.map((name, index) => (

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

))}

</ul>

);

}

* 1. **Explain about keys in React applications.**

**Answer:**

**Keys** are special attributes used in **lists of elements** in React to **uniquely identify** eachitem. They help React **efficiently update, add, or remove items** when the listchanges.

**Example:**

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

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

* 1. **Explain how to extract components with keys.**

**Answer:**

When rendering lists in React, it’s common to **extract each list item into a separate component** for cleaner, reusable, and modular code. In such cases, you must still assign a key **prop** but the key should be passed to the **component instance**, not to an element inside it.

**Example:**

function StudentList() {

const students = ['Ayushmaan', 'Riya', 'Mohit'];

return (

<ul>

{students.map((student, index) => (

<Student key={index} name={student} />

))}

</ul>

);

}

* 1. **Explain React Map, map() function.**

**Answer:**

In React, the map() **function** is commonly used to **render lists of elements or components** dynamically. It’s a **JavaScript array method** that creates a new array by calling a function on each element.

**Basic Syntax:**

function App() {

const names = ['Ayush', 'Riya', 'Mohit'];

return (

<ul>

{names.map((name, index) => (

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

))}

</ul>

);}

**Practical Question1:**

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

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

**BlogDetails.js**

*import* *React* *from* '*react*';

*function* BlogDetails() {

*const* blogs = [

    {

*id*: 1,

*title*: '*React Learning*',

*author*: '*Stephen Biz*',

*content*: '*Welcome to learning React!*'

    },

    {

*id*: 2,

*title*: '*Installation*',

*author*: '*Schewzdenier*',

*content*: '*You can install React from npm.*'

    }

  ];

*return* (

    <div *style*=*{*{ *flex*: 1, *padding*: '*20px*' }*}*>

      <h2>Blog Details</h2>

*{*blogs.map(*blog* *=>* (

        <div *key*={blog.id}>

          <h3>{blog.title}</h3>

          <strong>{blog.author}</strong>

          <p>{blog.content}</p>

        </div>

      ))*}*

    </div>

  );

}

*export* *default* BlogDetails;

**BookDetails.js**

*import* *React* *from* '*react*';

*function* BookDetails() {

*const* books = [

    { *id*: 1, *title*: '*Master React*', *price*: 670 },

    { *id*: 2, *title*: '*Deep Dive into Angular 11*', *price*: 800 },

    { *id*: 3, *title*: '*Mongo Essentials*', *price*: 450 }

  ];

*return* (

    <div *style*=*{*{ *flex*: 1, *padding*: '*20px*', *borderRight*: '*4px solid green*' }*}*>

      <h2>Book Details</h2>

*{*books.map(*book* *=>* (

        <div *key*={book.id}>

          <h3>{book.title}</h3>

          <p>{book.price}</p>

        </div>

      ))*}*

    </div>

  );

}

*export* *default* BookDetails;

**CourseDetails.js**

*import* *React* *from* '*react*';

*function* CourseDetails() {

*const* courses = [

    { *id*: 1, *name*: '*Angular*', *date*: '*4/5/2021*' },

    { *id*: 2, *name*: '*React*', *date*: '*6/3/20201*' }

  ];

*return* (

    <div *style*=*{*{ *flex*: 1, *padding*: '*20px*', *borderRight*: '*4px solid green*' }*}*>

      <h2>Course Details</h2>

*{*courses.map(*course* *=>* (

        <div *key*={course.id}>

          <h3>{course.name}</h3>

          <p>{course.date}</p>

        </div>

      ))*}*

    </div>

  );

}

*export* *default* CourseDetails;

**App.js**

*import* *React* *from* '*react*';

*import* *CourseDetails* *from* '*./CourseDetails*';

*import* *BookDetails* *from* '*./BookDetails*';

*import* *BlogDetails* *from* '*./BlogDetails*';

*function* App() {

*return* (

    <div *style*=*{*{ *display*: '*flex*', *justifyContent*: '*space-around*', *fontFamily*: '*Arial*' }*}*>

      <CourseDetails />

      <BookDetails />

      <BlogDetails />

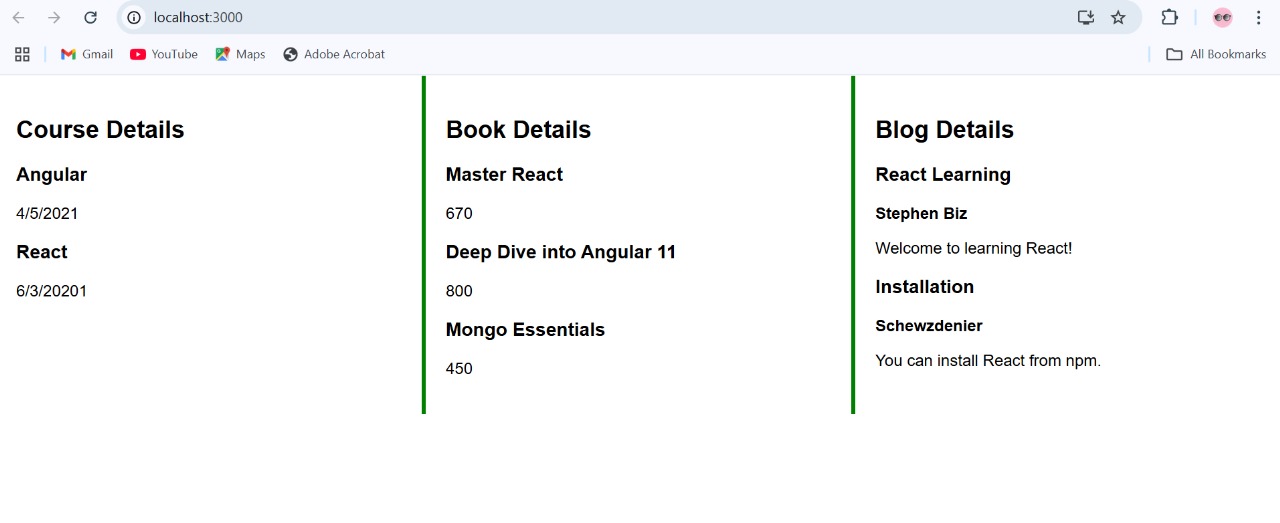
    </div>

  );

}

*export* *default* App;

**Output:**

****