**WEEK 7**

**REACT**

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

**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. Display the Odd Team Player and Even Team players using the Destructuring features of ES6. 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.**

**List the features of ES6:**

ES6, also known as ECMAScript 2015, introduced several modern features to JavaScript including let and const for block-scoped variables, arrow functions, classes and inheritance, template literals, default parameters, destructuring, spread/rest operators, modules (import/export), and new data structures like Set and Map. These improvements made JavaScript more powerful, cleaner, and closer to other object-oriented languages.

**Explain JavaScript let:**

The let keyword in JavaScript allows you to declare block-scoped variables. Unlike var, let is not hoisted to the top of its scope and cannot be redeclared in the same block. It’s useful for variables whose values may change and ensures better scoping within loops, conditionals, and functions.

**Identify the differences between var and let:**

var is function-scoped and allows redeclaration, while let is block-scoped and does not allow redeclaration in the same block. Variables declared with var are hoisted and initialized as undefined, but let is hoisted without initialization, causing a Reference Error if accessed before declaration. let is preferred in modern JavaScript for better scoping and cleaner code.

**Explain JavaScript const:**

const is used to declare block-scoped variables whose values cannot be reassigned after initialization. It ensures constant bindings. While the reference to an object or array declared with const cannot be changed, the contents (properties or elements) of those objects or arrays can still be modified. It’s ideal for declaring constants or fixed references.

**Explain ES6 class fundamentals:**

ES6 introduced class syntax to simplify object-oriented programming in JavaScript. A class is a blueprint for creating objects and can contain a constructor and methods. It’s syntactic sugar over JavaScript’s prototype-based inheritance, providing a cleaner and more familiar way for developers to create reusable and structured code using object-oriented principles.

**Explain ES6 class inheritance:**

ES6 supports class inheritance using the extends and super keywords. A child class inherits properties and methods from a parent class using extends, and super() is used inside the constructor to call the parent’s constructor. This enables code reuse and modular design in object-oriented JavaScript, making it easier to manage complex applications.

**Define ES6 arrow functions:**

Arrow functions provide a concise way to write functions using the => syntax. They do not bind their own this, arguments, or super, making them ideal for callbacks and short functions. Arrow functions reduce boilerplate code and are often used in array methods like map, filter, and reduce.

**Identify set() and map():**

Set and Map are new data structures introduced in ES6. A Set stores unique values of any type and automatically removes duplicates. A Map stores key-value pairs and allows any data type as a key. Both offer better performance and flexibility compared to traditional objects or arrays for certain use cases.

**CODE:**

**//ListOfPlayers.js:**

import React from 'react';

const ListofPlayers = () => {

const players = [

{ name: 'Mr. Jack', score: 50 },

{ name: 'Mr. Michael', score: 70 },

{ name: 'Mr. John', score: 40 },

{ name: 'Mr. Ann', score: 61 },

{ name: 'Mr. Elisabeth', score: 61 },

{ name: 'Mr. Sachin', score: 95 },

{ name: 'Mr. Dhoni', score: 100 },

{ name: 'Mr. Virat', score: 84 },

{ name: 'Mr. Jadeja', score: 64 },

{ name: 'Mr. Raina', score: 75 },

{ name: 'Mr. Rohit', score: 80 }

];

const filteredPlayers = players.filter(player => player.score < 70);

return (

<div>

<h2>List of Players</h2>

<ul>

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

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

))}

</ul>

<hr />

<h2>List of Players having Scores Less than 70</h2>

<ul>

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

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

))}

</ul>

</div>

);

};

export default ListofPlayers;

**//App.js:**

import React from 'react';

import './App.css';

import ListofPlayers from './ListofPlayers';

function App() {

return (

<div className="App">

<ListofPlayers />

</div>

);

}

export default App;

**OUTPUT:**

**WHEN TRUE:**



**WHEN FALSE:**



1. **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.**

**Define JSX:**

JSX stands for JavaScript XML. It allows developers to write HTML-like syntax directly in JavaScript, making the structure of UI components more readable and easier to maintain. JSX is not valid JavaScript and needs to be transpiled (e.g., by Babel) into React.createElement() calls. It simplifies the development of React applications.

**Explain about ECMAScript:**

ECMAScript (ES) is the standard specification on which JavaScript is based. It defines language syntax and features. Versions like ES5 and ES6 (ECMAScript 2015) brought improvements like let, const, arrow functions, and classes. ES6 and later versions are widely used in modern JavaScript and React development for writing efficient, cleaner code.

**Explain React.createElement():**

React.createElement() is the core function used by React to create virtual DOM elements. JSX code like <h1>Hello</h1> is transformed into React.createElement('h1', null, 'Hello'). It takes three arguments: the element type, properties (props), and children. It enables React to efficiently update and render UI components in the browser.

**Explain how to create React nodes with JSX:**

React nodes are created using JSX syntax, which looks like HTML written inside JavaScript. For example, <div><h1>Hello</h1></div> creates a tree of React nodes. JSX makes it easy to visualize the component structure and is converted internally into virtual DOM elements using React.createElement().

**Define how to render JSX to DOM:**

JSX is rendered to the real DOM using ReactDOM.render(). This method takes a JSX element and a DOM container (usually a <div id="root">) and mounts the virtual DOM to the actual DOM. This rendering is what displays your React UI in the browser.

**Explain how to use JavaScript expressions in JSX:**

JavaScript expressions can be embedded in JSX using curly braces {}. This allows dynamic values to be displayed inside HTML-like code. For example: <p>{2 + 2}</p> renders as "4". You can use expressions like variables, function calls, conditionals, or mathematical operations.

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

Inline CSS in JSX is written using JavaScript objects with camelCase properties. The style is passed using the style attribute. For example: <h1 style={{ color: 'blue', fontSize: '24px' }}>Hello</h1>. You can also store the style object in a variable and use it conditionally for dynamic styling.

**CODE:**

**//OfficeCard.js**

import React from 'react';

const OfficeCard = ({ office }) => {

const rentStyle = {

color: office.rent < 60000 ? 'textred' : 'textgreen',

fontWeight: 'bold'

};

return (

<div style={{ border: '1px solid #aaa', padding: '10px', margin: '10px', width: '250px' }}>

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

<img src={office.image} alt={office.name} width="200" height="100" />

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

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

</div>

);

};

export default OfficeCard;

**//App.js:**

import React from 'react';

import OfficeCard from './OfficeCard';

function App() {

const officeList = [

{ name: "PrimeSpace", rent: 45000, address: "Chennai", image: "https://via.placeholder.com/150" },

{ name: "TechHub", rent: 75000, address: "Bangalore", image: "https://via.placeholder.com/150" },

{ name: "BizNest", rent: 60000, address: "Mumbai", image: "https://via.placeholder.com/150" }

];

return (

<div>

<h1>Office Space Rental Listings</h1>

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

<OfficeCard key={index} office={office} />

))}

</div>

);

}

export default App;

**//Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

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

root.render(<App />);

**OUTPUT:**



1. **Create a React Application “eventexamplesapp” to handle various events of the form elements in HTML. 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. To increment the value. Say Hello followed by a static message.**

**Explain React events:**

React events are similar to HTML DOM events but are handled via React's synthetic event system. These events provide a consistent and cross-browser wrapper around the native events, allowing developers to attach handlers to elements using JSX. Common events include onClick, onChange, and onSubmit.

**Explain about event handlers:**

Event handlers are functions that are triggered when a specific event occurs, such as a click or input change. In React, event handlers are passed as props to elements using camelCase syntax (e.g., onClick). They allow developers to control user interaction and update component state accordingly.

**Define Synthetic event:**

Synthetic events in React are wrapper objects around the browser’s native events. They combine the behavior of different browsers into a single API, ensuring consistency. Examples include SyntheticMouseEvent for mouse actions or SyntheticKeyboardEvent for keyboard input, and they are passed as parameters to event handlers.

**Identify React event naming convention:**

React follows camelCase naming for events, unlike HTML which uses lowercase. For example, onclick in HTML becomes onClick in React. Additionally, event handlers in React are passed as functions, not as strings like in traditional HTML.

**CODE:**

**//Welcome.js:**

import React from 'react';

const Welcome = () => {

const sayWelcome = (msg) => {

alert(msg);

};

return (

<div>

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

</div>

);

};

export default Welcome;

**//Counter.js:**

import React, { Component } from 'react';

class Counter extends Component {

constructor() {

super();

this.state = {

count: 0

};

}

increment = () => {

this.setState({ count: this.state.count + 1 });

this.sayHello();

};

decrement = () => {

this.setState({ count: this.state.count - 1 });

};

sayHello = () => {

alert("Hello! Value increased.");

};

render() {

return (

<div>

<h2>Counter Value: {this.state.count}</h2>

<button onClick={this.increment}>Increment</button>

<button onClick={this.decrement}>Decrement</button>

</div>

);

}

}

export default Counter;

**//SyntheticEvent.js:**

import React from 'react';

const SyntheticEvent = () => {

const handleClick = (e) => {

alert("I was clicked");

console.log(e); // synthetic event

};

return (

<div>

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

</div>

);

};

export default SyntheticEvent;

**//CurrencyConverter.js:**

import React, { useState } from 'react';

const CurrencyConvertor = () => {

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

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

const handleSubmit = () => {

const result = parseFloat(rupees) / 88; // Example conversion rate

setEuro(result.toFixed(2));

};

return (

<div>

<h3>Currency Converter (INR to EURO)</h3>

<input

type="number"

placeholder="Enter amount in INR"

value={rupees}

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

/>

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

{euro && <p>Converted Amount: €{euro}</p>}

</div>

);

};

export default CurrencyConvertor;

**//App.js:**

import React from 'react';

import Counter from './Counter';

import Welcome from './Welcome';

import SyntheticEvent from './SyntheticEvent';

import CurrencyConvertor from './CurrencyConvertor';

function App() {

return (

<div className="App">

<h1>Event Handling in React</h1>

<Counter />

<Welcome />

<SyntheticEvent />

<CurrencyConvertor />

</div>

);

}

export default App;

**OUTPUT:**









1. **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.**

**Explain about conditional rendering in React:**

Conditional rendering in React is the technique of rendering different components or elements based on certain conditions, such as user login status, form submission, or state. This can be done using if-else, ternary operators, logical &&, or element variables. It enables dynamic and interactive user interfaces.

**Define element variables:**

Element variables in React are JavaScript variables that store JSX elements. You can use them to conditionally render parts of a component by assigning different JSX code to the variable based on some logic and including that variable in your return statement.

**Explain how to prevent components from rendering:**

To prevent a component from rendering, you can return null from the component’s render method or conditionally exclude it using if statements or logical operators in JSX. This is useful for hiding UI elements without removing them from the component hierarchy.

**CODE:**

**//Userpage.js:**

import React from 'react';

const UserPage = () => {

return (

<div>

<h2>Welcome User!</h2>

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

<button>Book Ticket</button>

</div>

);

};

export default UserPage;

**//GuestPage.js:**

import React from 'react';

const GuestPage = () => {

return (

<div>

<h2>Welcome Guest!</h2>

<p>Here are the available flight details:</p>

<ul>

<li>Chennai to Delhi - ₹4500</li>

<li>Mumbai to Bangalore - ₹3500</li>

</ul>

</div>

);

};

export default GuestPage;

**App.js:**

import React, { useState } from 'react';

import GuestPage from './GuestPage';

import UserPage from './UserPage';

function App() {

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

const handleLoginLogout = () => {

setIsLoggedIn(prev => !prev);

};

// Element variable

let page;

if (isLoggedIn) {

page = <UserPage />;

} else {

page = <GuestPage />;

}

return (

<div className="App">

<h1>Flight Ticket Booking</h1>

<button onClick={handleLoginLogout}>

{isLoggedIn ? 'Logout' : 'Login'}

</button>

{page}

</div>

);

}

export default App;

**OUTPUT:**





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

**Book Details**

**Blog Details**

**Course Details**

**Implement this with as many ways possible of Conditional Rendering.**

**Explain various ways of conditional rendering:**

In React, conditional rendering can be implemented using different methods such as if-else statements, ternary operators, logical &&, and element variables. These techniques control what content gets displayed based on component state or props, enabling dynamic UI updates without reloading the page.

**Explain how to render multiple components:**

Multiple components in React can be rendered inside a parent component using JSX. Components can be returned side by side in a single div or React Fragment (<> </>). This modular structure improves reusability and separation of concerns.

**Define list component:**

A list component in React is a component that displays a set of items, usually rendered by iterating over an array using the map() function. Each item is typically represented as a child component for better organization.

**Explain about keys in React applications:**

Keys are unique identifiers assigned to elements when rendering lists in React. They help React track which items have changed, been added, or removed, improving performance. Keys must be unique and stable — using item IDs is recommended over array indices.

**Explain how to extract components with keys:**

When rendering lists, each item can be passed as props to a child component. Keys are added to the component instances (not inside the child). This allows React to efficiently re-render only the changed items.

**Explain React map() function:**

The map() function is used to iterate over an array and return a new array of JSX elements. In React, it’s used to generate components dynamically, such as rendering lists of books or blogs.

**CODE:**

**//BookDetails.js:**

import React from 'react';

const BookDetails = ({ books }) => {

return (

<div>

<h2>Book Details</h2>

<ul>

{books.map((book, index) => (

<li key={book.id}>{book.title} by {book.author}</li>

))}

</ul>

</div>

);

};

export default BookDetails;

**//BlogDetails.js:**

import React from 'react';

const BlogDetails = ({ blogs }) => {

return (

<div>

<h2>Blog Details</h2>

{blogs.map((blog) => (

<div key={blog.id}>

<h4>{blog.title}</h4>

<p>{blog.summary}</p>

</div>

))}

</div>

);

};

export default BlogDetails;

**//CourseDetails.js:**

import React from 'react';

const CourseDetails = ({ courses }) => {

return (

<div>

<h2>Course Details</h2>

<ul>

{courses.map((course) => (

<li key={course.id}>{course.name} - ₹{course.price}</li>

))}

</ul>

</div>

);

};

export default CourseDetails;

**//App.js:**

import React, { useState } from 'react';

import BookDetails from './BookDetails';

import BlogDetails from './BlogDetails';

import CourseDetails from './CourseDetails';

function App() {

const [view, setView] = useState('book');

const books = [

{ id: 1, title: 'React Basics', author: 'Dan Abramov' },

{ id: 2, title: 'Learning JSX', author: 'Andrew Clark' }

];

const blogs = [

{ id: 1, title: 'Why React is Fast', summary: 'Because of Virtual DOM.' },

{ id: 2, title: 'React vs Angular', summary: 'Comparing two frameworks.' }

];

const courses = [

{ id: 1, name: 'React Bootcamp', price: 2000 },

{ id: 2, name: 'Fullstack JS', price: 3500 }

];

let content; // Using element variable

if (view === 'book') {

content = <BookDetails books={books} />;

} else if (view === 'blog') {

content = <BlogDetails blogs={blogs} />;

} else {

content = <CourseDetails courses={courses} />;

}

return (

<div>

<h1>Blogger App</h1>

<button onClick={() => setView('book')}>Books</button>

<button onClick={() => setView('blog')}>Blogs</button>

<button onClick={() => setView('course')}>Courses</button>

{content} {/\* Using conditional element variable \*/}

</div>

);

}

export default App;

**OUTPUT:**

