REACTJS

**1 What is React.js? How is it different from other JavaScript frameworks and libraries?**

React.js is a JavaScript library developed by Facebook for building user interfaces, particularly for single-page applications. It uses a component-based architecture and the virtual DOM for efficient updates. Unlike frameworks like Angular, which provide a full-fledged solution, React focuses solely on the view layer, offering more flexibility in how developers structure their applications.

**2 Explain the core principles of React such as the virtual DOM and component- based architecture.**

**Virtual DOM**: React uses a virtual representation of the real DOM. It updates the virtual DOM first and then efficiently updates the real DOM based on the differences, enhancing performance.

**Component-Based Architecture**: React applications are built using reusable components, each managing its own state and logic. This modular approach promotes code reusability and maintainability.

**3: What are the advantages of using React.js in web development?**

Performance improvements through the virtual DOM.

Reusable and composable components.

Strong community support and a large ecosystem of libraries and tools.

Enhanced developer experience with JSX and developer tools.

LAB EXERCISE

**Set up a new React.js project using create-react-app.**

**Create a basic component that displays "Hello, React!" on the web page.**

Install create-react-app globally: npm install -g create-react-app

Create a new project: create-react-app my-app

Navigate to the project directory: cd my-app

Start the development server: npm start

**Create a basic component that displays "Hello, React!" on the web page:**

import React from 'react';

function App() {

return (

<div className="App">

<h1>Hello, React!</h1>

</div>

);

}

export default App;

JSX (JavaScript XML)

**1: What is JSX in React.js? Why is it used?**

JSX is a syntax extension that allows developers to write HTML-like code within JavaScript. It improves readability and maintainability by providing a clear structure for defining UI components.

**2: How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?**

JSX combines HTML-like syntax with JavaScript. You can embed JavaScript expressions within JSX using curly braces {}, allowing for dynamic content.

**3: Discuss the importance of using curly braces {} in JSX expressions.**

Curly braces {} in JSX are used to embed JavaScript expressions, making the UI dynamic and interactive. They allow for conditional rendering, looping, and function calls within the markup.

LAB EXERCISE

**Create a React component that renders the following JSX elements:**

**A heading with the text "Welcome to JSX".**

**A paragraph explaining JSX with dynamic data (use curly braces to insert variables).**

Components (Functional & Class Components)

**1: What are components in React? Explain the difference between functional components and class components.**

Components are the building blocks of a React application. Functional components are simple functions that return JSX, while class components are ES6 classes that extend React.Component and include a render method.

**2: How do you pass data to a component using props?**

Data can be passed to a component using props, which are received as parameters in functional components or through this.props in class components.

**3: What is the role of render() in class components?**

The render() method in class components returns the JSX to be rendered on the screen. It is a required method in every class component.

LAB EXERCISE

1. **Create a functional component Greeting that accepts a name as a prop and**

**displays "Hello, [name]!".**

1. **Create a class component WelcomeMessage that displays "Welcome to React!" and**

**a render() method.**

Props and State

**1: What are props in React.js? How are props different from state?**

Props are read-only data passed from parent to child components. They are immutable within the child component. State, on the other hand, is mutable and managed within the component itself, allowing for dynamic updates.

**2: Explain the concept of state in React and how it is used to manage component data.**

State is an object that holds dynamic data specific to a component. It is managed within the component and can be updated using this.setState() in class components or the useState hook in functional components.

**3: Why is this.setState() used in class components, and how does it work?**

this.setState() is used to update the state in class components. It schedules an update to the component's state and re-renders the component with the new state.

LAB EXERCISE

**1 Create a React component UserCard that accepts name, age, and location as**

**props and displays them in a card format.**

**2 Create a Counter component with a button that increments a count value using**

**React state. Display the current count on the screen.**

Handling Events in React

**1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.**

In React, events are handled using synthetic events, which are wrapper objects around the browser's native event. This ensures cross-browser compatibility and provides a consistent API.

**2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.**

**onClick**: Handles click events.

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

**onChange**: Handles input change events.

<input type="text" onChange={handleInputChange} />

**onSubmit**: Handles form submission events.

<form onSubmit={handleSubmit}>

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

</form>

**3: Why do you need to bind event handlers in class components?**

In class components, this is not automatically bound to instance methods. Therefore, you need to bind event handlers in the constructor to ensure they have the correct this context.

LAB EXERCISE

Conditional Rendering

**1: What is conditional rendering in React? How can you conditionally render**

**elements in a React component?**

**Conditional Rendering**: It is a technique used in React to display different components or elements based on certain conditions. React allows you to use JavaScript conditional statements (if-else, ternary operators, logical &&) to render components or elements conditionally.

You can use various methods for conditional rendering:

**If-Else Statements**:

if (isLoggedIn) {

return <LogoutButton />;

} else {

return <LoginButton />;

}

**Ternary Operators**:

return isLoggedIn ? <LogoutButton /> : <LoginButton />;

**2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.**

If-Else Statements: These can be used inside the render method to conditionally render components.

Ternary Operators: A concise way to render one of two values based on a condition.

Logical && Operator: Renders the right-hand operand if the condition on the left-hand side is true.

LAB EXERCISE

**1 Create a component that conditionally displays a login or logout button based on the user’s login status.**

**2 Implement a component that displays a message like "You are eligible to vote" if the**

**user is over 18, otherwise display "You are not eligible to vote."**

**Lists and Keys**

**1: How do you render a list of items in React? Why is it important to use keys when rendering lists?**

In React, you can render a list of items using the map() function. Each item in the list should be rendered as a React element.

**Keys**: Keys are important for identifying which items have changed, been added, or removed. They help React optimize the rendering process by allowing it to efficiently update only the parts of the DOM that have changed.

**2: What are keys in React, and what happens if you do not provide a unique key?**

**Keys**: Keys are unique identifiers assigned to each element in a list. They should be unique among siblings.

If you do not provide a unique key, React will have trouble determining which items have changed, potentially leading to performance issues and unexpected behavior in the application.

LAB EXERCISE

**1 Create a React component that renders a list of items (e.g., a list of fruit names). Use**

**the map() function to render each item in the list.**

**2 Create a list of users where each user has a unique id. Render the user list using**

**React and assign a unique key to each user.**

Forms in React

**1: How do you handle forms in React? Explain the concept of controlled components.**

In React, forms are handled by maintaining the form's state and updating it based on user input. Controlled components are components where the form data is handled by the component's state.

**2: What is the difference between controlled and uncontrolled components in React?**

**Controlled Components**: These are components where form data is controlled by the component's state. The input elements are bound to state variables, and changes are handled by event handlers.

**Uncontrolled Components**: These are components where form data is handled by the DOM itself. Input elements maintain their own state, and refs are used to access the values.

LAB EXERCISE

**1 Create a form with inputs for name, email, and password. Use state to control the form and display the form data when the user submits it.**

**2 Add validation to the form created above. For example, ensure that the email input contains a valid email address.**

Lifecycle Methods (Class Components)

**1: What are lifecycle methods in React class components? Describe the phases of a component’s lifecycle.**

Lifecycle Methods: These are special methods in React class components that allow developers to hook into specific moments in a component's lifecycle (mounting, updating, unmounting). They offer control over what happens at different stages.

Phases of a Component’s Lifecycle:

Mounting: The component is being created and inserted into the DOM.

Common methods: constructor(), componentDidMount()

Updating: The component is being re-rendered as a result of changes to props or state.

Common methods: shouldComponentUpdate(), componentDidUpdate()

Unmounting: The component is being removed from the DOM.

Common method: componentWillUnmount()

**2: Explain the purpose of componentDidMount(), componentDidUpdate(),and componentWillUnmount().**

**componentDidMount()**: This method is called once, immediately after the component is added to the DOM. It's ideal for initializing data (e.g., making API calls).

**componentDidUpdate()**: This method is invoked immediately after updating occurs. It can be used for operations like updating the DOM in response to prop or state changes.

**componentWillUnmount()**: This method is called just before the component is removed from the DOM. It's used for cleanup activities such as invalidating timers, canceling network requests, or cleaning up subscriptions.

LAB EXERCISE

**1: Create a class component that fetches data from an API when the component mounts using componentDidMount(). Display the data in the component.**

**2: Implement a component that logs a message to the console when it updates using componentDidUpdate(). Log another message when the component unmounts using componentWillUnmount().**

Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)

**1: What are React hooks? How do useState() and useEffect() hooks work in functional components?**

**React Hooks**: Hooks are functions that let you use state and other React features in functional components. They eliminate the need for class components.

**useState()**: This hook allows you to add state to functional components. It returns an array containing the current state value and a function to update it.

**useEffect()**: This hook lets you perform side effects in functional components. It takes a function and an optional array of dependencies. The function runs after every render by default.

**2: What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

**Problems Solved**:

Reusing stateful logic without changing component hierarchy.

Avoiding complex wrapper components.

Simplifying the handling of side effects.

**Importance**: Hooks simplify code by allowing the use of state and other React features in functional components. They make it easier to share logic and manage side effects.

**3: What is useReducer ? How we use in react app?**

**useReducer**: This hook is an alternative to useState for managing complex state logic. It accepts a reducer function and an initial state, and returns the current state and a dispatch function to trigger state updates.

**4: What is the purpose of useCallback & useMemo Hooks?**

 **useCallback**: This hook returns a memoized version of a callback function. It's useful for optimizing performance by preventing unnecessary re-creations of functions.

 **useMemo**: This hook returns a memoized value. It re-computes the value only when one of its dependencies changes, optimizing performance by avoiding costly calculations on every render.

**5: What’s the Difference between the useCallback & useMemo Hooks?**

**useCallback**: Memoizes a function.

**useMemo**: Memoizes a value.

**6 : What is useRef ? How to work in react app?**

**useRef**: This hook returns a mutable ref object with a .current property. It's commonly used to access and interact with DOM elements directly or to persist values across renders without causing re-renders.

LAB EXERCISE

**1.Create a functional component with a counter using the useState() hook. Include buttons to increment and decrement the counter.**

**2. Use the useEffect() hook to fetch and display data from an API when the component mounts.**

**3.Create react app with use of useSelector & useDispatch.**

**4.Create react app to avoid re-renders in react application by useRef ?**

Routing in React (React Router)

**1: What is React Router? How does it handle routing in single-page applications?**

React Router: It is a library for managing navigation and rendering components based on the URL in a React application. It allows for declarative routing and handles client-side navigation, making single-page applications behave more like traditional multi-page applications.

Handling Routing: React Router uses components like BrowserRouter, Route, Link, and Switch to define routes, render components, and enable navigation without refreshing the page.

**2: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.**

BrowserRouter: A router implementation that uses the HTML5 history API to keep the UI in sync with the URL.

Route: Renders a component based on the matching path.

Link: Used for navigation between different routes without reloading the page.

Switch: Renders the first child Route that matches the current location.

**LAB EXERCISE**

**1. Set up a basic React Router with two routes: one for a Home page and one for an About page. Display the appropriate content based on the URL.**

**2. Create a navigation bar using React Router’s Link component that allows users to switch between the Home, About, and Contact pages.**

**React – JSON-server and Firebase Real Time Database**

**1: What do you mean by RESTful web services?**

**RESTful Web Services**: REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful web services use HTTP methods (GET, POST, PUT, DELETE) to perform CRUD operations and exchange data between clients and servers in a stateless manner.

**2: What is Json-Server? How we use in React ?**

JSON-server: JSON-server is a library that allows you to create a full fake REST API with zero coding. It's useful for prototyping and testing applications.

Usage in React: JSON-server can be used as a backend to provide mock data for a React application. You can fetch data from the JSON-server API using HTTP requests (e.g., fetch or axios).

**3: How do you fetch data from a Json-server API in React? Explain the role of fetch() or axios() in making API requests.**

Fetching Data: You can use fetch() or axios() to make HTTP requests to the JSON-server API and retrieve data.

**fetch() Example**:

useEffect(() => {

fetch('http://localhost:3000/data')

.then(response => response.json())

.then(data => setData(data));

}, []);

**axios() Example**:

useEffect(() => {

axios.get('http://localhost:3000/data')

.then(response => setData(response.data));

}, []);

**4: What is Firebase? What features does Firebase offer?**

 **Firebase**: Firebase is a platform developed by Google that provides a suite of cloud-based services to help developers build and manage applications.

 **Features**:

* Real-time database
* Authentication
* Cloud Firestore
* Cloud Storage
* Hosting
* Cloud Functions

**5: Discuss the importance of handling errors and loading states when working with APIs in React .**

**Handling Errors and Loading States**: It's important to handle errors and loading states to improve the user experience. Displaying a loading spinner while data is being fetched and showing error messages when requests fail help provide feedback to the user.

**LAB EXERCISE**

**1. Create a React component that fetches data from a public API (e.g., a list of users) and displays it in a table format.**

**Create a React app with Json-server and use Get , Post , Put , Delete & patch method on Json-server API.**

**2 Create a React app crud and Authentication with firebase API.**

**Implement google Authentication with firebase API.**

**3 Implement error handling and loading states for the API call. Display a loading spinner while the data is being fetched.**

**Context API**

**1: What is the Context API in React? How is it used to manage global state across multiple components?**

**Context API**: It provides a way to pass data through the component tree without having to pass props down manually at every level. It's used to manage global state or settings that need to be accessed by multiple components.

**2: Explain how createContext() and useContext() are used in React for sharing state.**

**createContext()**: Creates a context object that holds the current value.

**useContext()**: A hook that allows components to consume the context value.

LAB EXERCISE

**1. Create a simple theme toggle (light/dark mode) using the Context API. The theme state should be shared across multiple components.**

**2. Use the Context API to create a global user authentication system. If the user is logged in, display a welcome message; otherwise, prompt them to log in.**

State Management (Redux, Redux-Toolkit or Recoil)

**1: What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.**

 **Context API**: It provides a way to pass data through the component tree without having to pass props down manually at every level. It's used to manage global state or settings that need to be accessed by multiple components. It's particularly useful for themes, user authentication, or any other settings that need to be accessible across the entire application.

 **Usage**: The Context API uses two primary functions: createContext() to create a context object and Provider to pass the value to components. Components that need the context data use the Consumer component or the useContext() hook to access the context value.

**2: How does Recoil simplify state management in React compared to Redux?**

**createContext()**: This function creates a context object that holds the value to be shared across components.

const MyContext = React.createContext(defaultValue);

**useContext()**: This hook allows a component to access the context value.

const value = useContext(MyContext);

LAB EXERCISE

1 Create a simple counter application using Redux for state management. Implement actions to increment and decrement the counter.

2 Build a todo list application using Recoil for state management. Allow users to add, remove, and mark tasks as complete.

3 Build a crud application using Redux-Toolkit for state management. Allow users to add, remove, delete and update.