Module 9

React – Components, state, Props

**Introduction :**

**Q1. What is React.js? How is it different from other JavaScript frameworks and libraries?**

Ans. React.js is an open-source JavaScript library primarily used for building user interfaces, particularly single-page applications (SPAs). Developed and maintained by Meta (formerly Facebook) and a community of developers, it focuses on creating reusable UI components.

React.js differs from other JavaScript frameworks and libraries as it uses a Virtual DOM for efficient updates, promotes a component-based architecture for reusability, and offers flexibility by letting developers choose additional tools like routing or state management, unlike frameworks like Angular that provide an all-in-one solution.

**Q2. Explain the core principles of React such as the virtual DOM and component- based architecture.**

Ans.

1. **Virtual DOM**:

* + React uses a **Virtual DOM**, a lightweight in-memory representation of the real DOM.
  + When the state of a component changes, React updates the Virtual DOM, compares it with the previous version (using a process called reconciliation), and applies only the minimal updates to the actual DOM.
  + This approach improves performance by reducing costly direct DOM manipulations.

1. **Component-Based Architecture:**

* React applications are built using small, reusable components.
* Each component represents a piece of the UI and manages its own logic, state, and rendering.
* Components can be nested, combined, and reused, leading to a modular and maintainable codebase.

**Q3. What are the advantages of using React.js in web development?**

Ans.

1. **Performance**: The Virtual DOM ensures fast updates and rendering.
2. **Reusability**: Component-based architecture promotes modular and reusable code.
3. **Declarative Syntax**: Simplifies UI development by focusing on "what" the UI should do.
4. **Flexibility**: Works well with other libraries and tools.
5. **Strong Community**: Large community and ecosystem for support and resources.

**Task 1 : Set up a new React.js project using create-react-app.**

1. Install Node.js
2. Create a New React App : npx create-react-app my-app
3. Navigate to Your Project Folder : cd my-app
4. Start the Development Server : npm start

**Task 2 : Create a basic component that displays "Hello, React!" on the web page.**

import React from 'react';

function App() {

return (

<div>

<h1>Hello, React!</h1>

</div>

);

}

export default App;

**2. JSX (JavaScript XML) :**

**Q1. What is JSX in React.js? Why is it used?**

Ans. JSX (JavaScript XML) is a syntax extension for JavaScript that looks similar to HTML. It allows you to write HTML-like code within JavaScript, which React then transforms into JavaScript functions.

We use jsx for :

* **Readability**: Makes it easier to visualize the component structure.
* **Integration**: Combines UI structure and logic in one place.
* **Efficiency**: React optimizes JSX into efficient JavaScript code during compilation.

**Q2. How is JSX different from regular JavaScript? Can you write JavaScript inside JSX?**

Ans. JSX allows writing UI elements directly within JavaScript, unlike regular JavaScript.

JSX is compiled into plain JavaScript using tools like Babel before being executed.

Yes, you can write JavaScript expressions inside JSX using curly braces {}.

**Q3. Discuss the importance of using curly braces {} in JSX expressions.**

Ans. Curly braces {} in JSX allow embedding JavaScript expressions directly within the markup.

* **Dynamic Content**: Render variables, functions, or expressions (e.g., {name}, {2 + 2}).
* **Logic**: Use conditional rendering (e.g., {isLoggedIn ? 'Welcome' : 'Login'}).
* **Flexibility**: Integrate JavaScript seamlessly into the JSX syntax for dynamic and interactive UIs.

Without {}, JSX treats everything as plain text.

**Task : 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).**

import React from 'react';

function App() {

const jsx = "JSX is a syntax extension for JavaScript that looks similar to HTML. It allows us to write UI elements in JavaScript, which React can render.";

return (

<div>

<h1>Welcome to JSX</h1>

<p>{jsx}</p>

</div>

);

}

export default App;

1. **Components (Functional & Class Components) :**

**Q1. What are components in React? Explain the difference between functional components and class components.**

Ans. Components are the building blocks of a React application. They are reusable, self-contained pieces of UI that handle rendering, logic, and styling for a specific part of the application.

* **Functional Components**: Simple functions that return JSX and use hooks for state and lifecycle.
* **Class Components**: ES6 classes that extend React.Component and use this.state and lifecycle methods.  
  Functional components are now preferred for their simplicity and ease of use with hooks.

**Q2. How do you pass data to a component using props?**

Ans. In React, props (short for "properties") are used to pass data from a parent component to a child component.

**Parent Component**: Pass data to the child component by setting it as an attribute.

function Parent() {

return <Child name="John" age={30} />;

}

**Child Component**: Access the data in the child component via props.

function Child(props) {

return <h1>{props.name} is {props.age} years old.</h1>;

}

**Q3. What is the role of render() in class components?**

Ans. In React class components, the render() method is responsible for returning the JSX (UI structure) that the component should display.

* **Purpose**: It defines what the component will render on the screen.
* **Automatic Invocation**: React calls the render() method automatically whenever there is a change in the component's state or props.
* **Return JSX**: The render() method must return **JSX** (or null if nothing is to be rendered).

**Task 1: Create a functional component Greeting that accepts a name as a prop and displays "Hello, [name]!".**

import React from 'react';

function Greeting(props) {

return <h1>Hello, {props.name}!</h1>;

}

function App() {

return (

<div>

<Greeting name="Vraj" />

<Greeting name="Patel" />

</div>

);

}

export default App;

**Task 2 : Task 2: o Create a class component WelcomeMessage that displays "Welcome to React!" and a render() method.**

import React, { Component } from 'react';

class WelcomeMessage extends Component {

render() {

return <h1>Welcome to React!</h1>;

}

}

function App() {

return (

<div>

<WelcomeMessage />

</div>

);

}

export default App;

1. **Props and State**

**Q1. What are props in React.js? How are props different from state?**

Ans. Props (short for "properties") are read-only inputs passed from a parent component to a child component. They allow data and event handlers to be shared between components.

Props are for passing data down,

while state is for handling internal component data and Mutable (can be changed within the component). Also Managed and updated within the component itself.

**Q2. Explain the concept of state in React and how it is used to manage component data.**

Ans. State in React is an object used to manage and track data that can change over time within a component. It determines how the component behaves and renders.

**How State is Used to Manage Component Data:**

1. **DeclaringState**:  
   In class components, state is declared in the constructor. In functional components, state is declared using the useState hook.
2. **UpdatingState**:  
   To change the state, you use the setState() method in class components or the setter function from useState in functional components.
3. **Re-rendering:**

When the state changes, React re-renders the component to reflect the updated state in the UI.

**Q3. Why is this.setState() used in class components, and how does it work?**

Ans. In React class components, this.setState() is used to update the component's state. It triggers a re-render of the component with the new state, ensuring the UI reflects the updated data.

How it works :

* **Reactivity**: this.setState() ensures the UI updates when the state changes.
* **Merging**: It merges the updated state with the existing state.
* **Async**: It works asynchronously and may batch multiple updates for performance.

**Task 1 : Create a React component UserCard that accepts name, age, and location as props and displays them in a card format.**

import React from 'react';

// Functional component UserCard

function UserCard(props) {

return (

<div style={styles.card}>

<h2>{props.name}</h2>

<p>Age: {props.age}</p>

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

</div>

);

}

function App() {

return (

<div>

<UserCard name="Vraj Patel" age={21} location="Nikol" />

<UserCard name="Het Patel" age={20} location="Maninagar" />

</div>

);

}

export default App;

**Task 2 : Create a Counter component with a button that increments a count value using React state. Display the current count on the screen.**

import React, { useState } from 'react';

function Counter() {

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

const increment = () => {

setCount(count + 1); // Update state with the new count

};

return (

<div>

<h1>Current Count: {count}</h1>

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

</div>

);

}

function App() {

return (

<div>

<Counter />

</div>

);

}

export default App;

**11. Routing in React (React Router)**

**Q1. What is React Router? How does it handle routing in single-page applications?**

Ans. React Router is a popular library in the React ecosystem for handling routing in single-page applications (SPAs). It allows developers to manage navigation and rendering of different components or views based on the URL path, all while maintaining the benefits of a single-page application where only specific parts of the page are updated without a full page reload.

React Router is a library for managing navigation in single-page applications (SPAs). It uses components like <BrowserRouter> and <Route> to map URLs to specific React components. It handles routing by updating the browser's history and rendering the appropriate components dynamically without a full page reload, ensuring smooth client-side navigation.

**Q2. Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.**

Ans.

**BrowserRouter**:

* Acts as the base wrapper for routing in React apps.
* Uses the browser’s History API to handle URL changes.
* Should wrap your app to enable routing functionality.

**Route**:

* Defines a path and specifies which component to render for that path.
* Renders its children when the URL matches the specified path.

**Link**:

* Provides navigation without a full page reload.
* Replaces traditional anchor (<a>) tags for internal routing in SPAs.

**Switch** :

* Ensures only the first matching route is rendered.
* Used for exclusive routing in React Router v5.

**10. Hooks (useState, useEffect, useReducer, useMemo, useRef, useCallback)**

**Q1. What are React hooks? How do useState() and useEffect() hooks work in functional components?**

Ans.

React Hooks are special functions that let you use React features in functional components. They were introduced in React 16.8 to simplify the development of React applications by avoiding the need for class components in many cases.

UseState(): Allows you to add state in class component.

* state: The current state value.
* setState: A function to update the state.
* initialValue: The initial value of the state.

UseEffect(): Manage side effects in functional components.

* Without a dependency array, useEffect runs after every render.
* With an empty array ([]), it runs only once (like componentDidMount).
* With dependencies ([dep1, dep2]), it runs whenever one of the dependencies changes.

**Q2. What problems did hooks solve in React development? Why are hooks considered an important addition to React?**

Ans.

**Problems Solved by Hooks:**

1. Simplified managing state and side effects in functional components.
2. Enabled reusable stateful logic without "wrapper hell".
3. Reduced boilerplate and complexity in class components.
4. Unified lifecycle handling with useEffect, improving readability and consistency.
5. Encouraged modular, maintainable code by splitting logic into smaller functions.

**Why Hooks Are Important:**

* Improved developer experience with concise, intuitive APIs.
* Promoted functional components as the standard in React.
* Allowed for easier state and side-effect management.
* Enhanced code reusability and reduced complexity.
* Backward-compatible and modernized React development.

**Q3. What is useReducer? How we use in react app?**

Ans.

useReducer is a React Hook used for state management, especially when state logic is complex or involves multiple actions. It provides an alternative to useState.

When to Use:

* For managing complex state transitions.
* When multiple related states are interdependent.

How It Works:

* It takes a reducer function and an initialstate as arguments.
* Returns [state, dispatch].

**Q4. What is the purpose of useCallback & useMemo Hooks?**

Ans.

Theyare React hooks that optimize performance by preventing unnecessary re-creations of functions or computations during re-renders. They are used to memoize values or functions.

UseCallback:

Purpose: Memoizes a function to ensure it remains the same between renders unless its dependencies change.

Prevents re-creation of functions passed as props to child components, avoiding unnecessary re-renders.

UseMemo:

Purpose: Memoizes a computed value to avoid recalculating it during every render unless its dependencies change.

Optimizes expensive calculations by caching results.

**Q5. What’s the Difference between the useCallback & useMemo Hooks?**

Ans.

UseCallback:

Purpose: Memoizes a function

Return value: Returns the memoized function

UseCase: Passing stable callback functions as props

Syntax: useCallback(() => func, [dependencies])

UseMemo:

Purpose: Memoizes a computed value

Return value: Returns the memoized result of a computation

Usecase: Optimizing expensive calculations

Syntax: useMemo() => compute(), [dependencies])

**Q6. What is useRef? How to work in react app?**

Ans: useRef is a React Hook that creates a mutable reference object. This object persists across renders and can be used for Accessing and interacting with DOM elements and Storing mutable values that don’t trigger re-renders when updated.

How useRef Works

* It returns an object with a current property, which you can read or update.
* Updating the current property doesn’t cause the component to re-render.

**12. React – Json-Server And Firebase Real Time Database**

**Q1. What do you mean by RESTful web services?**

Ans. RESTful web services in React JS refer to how React applications communicate with a server using the principles of REST (Representational State Transfer). This typically involves making HTTP requests (like GET, POST, PUT, DELETE) to interact with a backend API. In short, React uses tools like fetch or libraries like axios to call REST APIs, retrieve data, and display it in the UI. It follows RESTful conventions where each endpoint represents a resource, and actions are mapped to standard HTTP methods.

**Q2. What is Json-Server? How we use in React ?**

Ans. **JSON-Server** is a lightweight, full-fledged fake REST API that allows you to quickly set up a mock backend to simulate interactions with a real RESTful API. It is useful for front-end development when you don’t have a backend ready but still need to test the functionality of your React app that requires an API.

**How to use JSON-Server in React:**

1. Install JSON-Server
2. Set up a db.json file
3. Add a script to start JSON-Server
4. Start JSON-Server
5. Make API requests in your React app

**Q3. How do you fetch data from a Json-server API in React? Explain the role of fetch() or axios() in making API requests.**

Ans. To fetch data from a JSON-Server API in React, you can use either the fetch() method or the axios library. Both methods are used to send HTTP requests to an API and retrieve data. Here's how each one works and how to use them in a React application:

**1. Using fetch():**

The fetch() method is a built-in JavaScript function that is used to make HTTP requests. It returns a Promise that resolves with the Response object, which contains the response from the API.

**2. Using axios:**

axios is a popular third-party library that provides an easier and more flexible way to make HTTP requests compared to fetch(). It also automatically parses the response as JSON, handles timeouts, and provides built-in error handling.

**Role of fetch() or axios() in Making API Requests:**

* **fetch()**: It is a native JavaScript function for making HTTP requests. It supports all HTTP methods (GET, POST, PUT, DELETE, etc.) and returns a Promise that resolves to a Response object. You typically have to manually parse the response using .json() or .text() to handle the data.
* **axios()**: It is a third-party library that provides a simpler, more powerful API for making HTTP requests. It automatically parses the response as JSON and offers additional features like request cancellation, built-in error handling, and timeouts. It is more user-friendly than fetch().

**Q4. What is Firebase? What features does Firebase offer?**

Ans. Firebase is a platform developed by Google that provides a suite of cloud-based tools and services designed to help developers build, manage, and grow applications. It is widely used for mobile and web app development, especially for real-time applications. Firebase simplifies back-end infrastructure tasks like databases, user authentication, and hosting, allowing developers to focus more on building their applications rather than managing servers and infrastructure.

Firebase offers a broad set of tools that can handle almost every aspect of app development, from user authentication and real-time databases to cloud storage and analytics. It provides developers with the infrastructure they need to build scalable apps with minimal effort on the backend, making it especially useful for mobile and web developers who want to focus on building features and improving user experience rather than managing infrastructure.

Key Features of Firebase:

1. **Realtime Database**
2. **Cloud Firestore**
3. **Authentication**
4. **Cloud Functions**
5. **Firebase Hosting**
6. **Cloud Storage**
7. **Firebase Analytics**

**Q5. Discuss the importance of handling errors and loading states when working with APIs in React**

**Ans.** When working with APIs in React (or any front-end application), handling errors and loading states is crucial for creating a smooth user experience and maintaining app stability. Here's why each of these aspects is so important:

* **Error Handling**: Helps to catch and display any issues (e.g., network failures, API errors) that might occur during an API call, preventing app crashes and providing clear feedback to users. It also aids in debugging and ensures the app can handle unexpected situations gracefully.
* **Loading States**: Informs users that data is being fetched, preventing confusion and improving UX. It sets expectations about wait times and keeps users engaged while the app is loading data.

By managing both, you improve UX, app reliability, and provide a more seamless interaction with external data sources.

**13. Contex API**

**Q1. What is the Context API in React? How is it used to manage global state across multiple components?**

Ans. The Context API in React is a way to manage global state and share data across multiple components without having to pass props manually at every level of the component tree.

How it works:

1. Create a Context: You create a context using React.createContext(), which creates a provider and a consumer.
2. Provider: The Provider component holds the state and makes it available to all child components that need it.
3. Consumer: The Consumer component or the useContext() hook is used in any child component to access the shared state.

**Q2. Explain how createContext() and useContext() are used in React for sharing state.**

Ans. In React, createContext() and useContext() are used together to share state and data across multiple components without the need to pass props through every level of the component tree. Here's how they work:

**1. createContext()**

createContext() creates a Context object, which is used to define the shared state and the components that can access it.

* **Provider**: The Provider component (created by createContext()) is used to wrap a part of the component tree and provide the context value to its descendants.
* **Consumer**: The Consumer component allows accessing the value in the context, but it's more commonly replaced by the useContext() hook in function components.

**2. useContext()**

useContext() is a React hook that allows a component to consume (access) the value provided by a Context, making it easier to share state without needing to manually pass props.

In short,

**createContext()** is used to create a context that holds the global state.

**useContext()** is a hook that allows components to consume and access the shared state provided by the Provider.