**Module 9 – Introduction to React.js**

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

Ans:- React.js is a JavaScript library for building fast and interactive user interface. It is component-based, uses virtual DOM for better performance, and is mainly used for single-page applications (SPAs)

* **Library, Not a Framework** – React focuses only on the **UI**, while Angular and Vue are full frameworks.
* **Virtual DOM** – React updates UI **faster** than Angular and jQuery, which use the **real DOM**.
* **One-way Data Binding** – React is **more predictable** than Angular’s **two-way binding**.
* **Component-Based** – Reusable components make development **easier** and **scalable**.
* **More Flexible** – React works with many tools like Redux, React Router, and Next.js, unlike Angular, which has built-in features.

**Q2: Explain the core principles of React such as the virtual DOM and componentbased architecture.**

Ans:-

1. **Virtual DOM (VDOM)**

* A **lightweight copy** of the real DOM.
* React updates only the **changed parts**, making it **faster** than directly modifying the real DOM.

**2. Component-Based Architecture**

* UI is broken into small, reusable components.
* Each component manages its own **state and logic**, making development **easier and scalable**.

**3. One-Way Data Binding**

* Data flows **from parent to child**, making it **predictable and easier to debug**.

**4. Declarative UI**

* You describe **what the UI should look like**, and React **automatically updates** it when data changes.

**5. State Management**

* Manages dynamic data using useState, Redux, or Context API.

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

Ans:- 1. **Fast Performance**

* Uses **Virtual DOM** for efficient updates, making UI changes faster than traditional methods.

2. **Reusable Components**

* UI is built with **independent, reusable components**, reducing code duplication and improving maintainability.

3. **One-Way Data Binding**

* Ensures **better control over data flow**, making debugging and tracking changes easier.

4. **Declarative UI**

* React updates the UI automatically when data changes, leading to **cleaner and more readable code**.

5. **Strong Community Support**

* Backed by **Meta (Facebook)** and a large developer community, ensuring regular updates and extensive resources.

6. **Easy to Learn and Use**

* Uses simple **JavaScript (JSX)** syntax, making it easier for developers familiar with JS to learn.

7. **SEO-Friendly**

* React improves **SEO performance** with **server-side rendering (SSR)** using Next.js.

8. **Rich Ecosystem & Flexibility**

* Works with tools like **Redux, React Router, and Next.js**, making it highly adaptable for different projects.

**JSX(JavaScript XML)**

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

Ans:- JSX is (JavaScript XML) is a syntax extension for JavaScript used to describe the UI structure. It allows developers to write HTML-like code inside JavaScript. JSX is not mandatory in React, But it makes the code more readable and easier to write.

* **Better Read ability** - JSX looks like HTMl, Making the UI code more readable and intuitive.
* **Easier to Write** – Instead of using complex React.createElement() call, JSX simplifies UI structured.
* **Boosts Performance** – JSX is complied into optimized JavaScript using Babel, Improving rendering efficiency.
* **Prevents injection Attack** – React escapes values before rendering to prevent XSS (Cross Site Scripting).

Examples

Without JSX:- const element = React.createElement("h1", null, "Hello, JSX!");

With JSX:- const element = <h1>Hello, JSX!</h1>;

**Q2: How is JSX different from regular JavaScript? Can you write JavaScript insideJSX?**

Ans:- JSX is an extensions of JavaScript that allows writing HTML-like syntax inside Java Script files. However, there are key differences,

**Feature :-** 1. Syntax , 2. Rendering Elements, 3. Attributes, 4. Expression.

**JSX :-** Uses an HTML-like syntax withing JavaScript, Uses <div>, <h1> etc , like HTML, Uses className inside of class htmlFor inside of for, Can include JavaScript expression I Side {}

**Regular JavaScript :-** Uses pure JavaScript syntax, Uses React.createElement(), Uses standard JavaScript object properties, Doesn’t support inline JSX-like Syntax.

\* **Can You Write JavaScript Inside JSX?**

**->** Yes! You can embed JavaScript expression inside JSX using {} . However, you cannot write statements like if, for, or while directly inside JSX

**Example**

**(1) Allowed (Expression inside {} ):-**

const name = “Dhaval”;

const element = <h1>Hello {name}</h1>

**(2) Not Allowed (Statements inside JSX) :-**

const elemet = <h1>{if (true) “Hello”}</h1>

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

Ans :- In JSX curly braces {} are used to embed JavaScript expression inside the markup. They allow dynamic context rendering within components.

**1. Embedding JavaScript Expression**

* You can use variables, functions, or expression inside {}.

**Example**

const name = “Dhaval”;

return <h1>Hello,{name}</h1>

2. **Dynamic Rendering**

* You can insert calculations, condition, and functions calls.

**Example**

return <h1>2 + 2 = {2 + 2} </h1>

3**. Conditional Rendering (Ternary Operator)**

* {} allow inline conditions using the ternary (? :) operator.

**Example**

const isLoggedIn = true;

return <h1>{ isLoggedIn ? “Welcome back” : “Please Log In”}</h1>

**Components (Functional & Class Components)**

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

Ans :- In React components are the building blocks of a users Interfaces (UI). They allow developers to break down complex UI into smaller, reusable pieces. A React component can be thought of as JavaScript functions or class that reruns JSX (JavaScript XML) to render UI elements.

**1. class Components**

-> **stateful** :- can manage state.

- > **lifecycle** :- Access to lifecycle method’s.

-> **Verbose** :- More boilerplate code not preferred anymore.

**2. Function Components**

-> Initially stateless.

-> Can use Hooks for state and effect’s.

-> Sampler and more concise and more popular.

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

Ans :- In React , you can pass data to a component using props (properties), which allow you to send values from a parent component to child components.

**1. Define the Props in the Parent Component**

* Pass values as attributes to child component

**2. Accesses the Props in the Child Components**

* Use **props** in function component or **this.props** in class component.

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

Ans :- class component , the render () method is **responsible for returning the JSX (UI) that should be displayed on the screen.** It is a required method in class component and gets called automatically whenever the component mounts or updates.

**Points of render()**

-> **Return JSX :-** It must return a single React element (or null).

-> **Pure Function:-** It should not modify state or interact with external APIs.

-> **Re-renders on State/Props Changes :-** if the component’s state or props change, render() runs again.

-> **Cannot Update State Directly :-** It should be a pure functions without side effects (like API calls or state updates)

**Props and State**

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

Ans :- Props (short for “properties”) in React are used to data from a parent component to a child component. They are immutable, meaning they cannot be changed by the child component that receives them

**1. State**

-> Local and mutable data within a component.

-> Initialization within the components.

-> can change over time.

-> Cause re-render when update.

-> Managed using useState in functional components.

**2. Props**

-> Passed into a complete from its parent.

-> Read-only (immutable) within the receiving components.

-> Allow parent-to-child component communication.

-> changes in props can also cause a re-render.

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

Ans :- State in React is an objects that holds data specific to a component. It allows components to be dynamics and interactive by strong and managing changing values over time.

React state is mutable and can be update using the useState hook in functional components or this.setState in class components.

**Example**

React provides the useState hooks to manage state in functions component

import { useState } from "react";

const Counter = () => {

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

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

};

export default Counter;

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

Ans :- class component, this.setState() is used to update the state should never be modified directly. If state is updated using this.state = newvalue, React will not re-render the component.

* **Reason for Using this.setState()**

**1. Ensure Components Re- renders**

-> Automatically trigger a re-render with update state.

**2. Merges State Updates**

-> Only updates the specified properties, keeping the rest unchanged.

**3. Handle Asynchrouns Update’s**

-> React may batch updates for performance.

Examples

this.setState() is used inside a class component to update the state objects

class WelcomeMessage extends Component {

  constructor(props) {

    super(props);

    this.state = { count: 0 };

  }

  increment = () => {

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

  };

  render() {

    return (

      <>

        <h1>{this.state.count}</h1>

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

      </>

    );

  }

}

export default WelcomeMessage;

**Handling Events in React**

**Q1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.**

Ans :- In Vanila JavaScript, event handling is done using method’s like:

* addEventListener to attach event listeners to element
* inline event handler’s (e.g., onclikc=”myFunction ()” in HTML).
* Event delegation using event.target to handle multiple element efficient.

Example :-

document. getElementById (“btn”). addEventListener(“clikc”,function(){

Alert(“btn click”)

})

-> In React , events are handle using event handlers passed as props

* React uses camaleCase instead of lowercase (onClikc instead of onclick).
* You define event handlers as functions inside components.
* React uses Synthetic Events for optimization.

Example:-

function App() {

const handleClick = () => {

alert("Button clicked!");

};

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

}

\* **Synthetic Events in React**

-> React does not use native browser event’s directly. Instead, it warps them in a cross-browser SyntheticEvent system.

**Q2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.**

Ans :-

-> **onClikc** – Triggered when an element is clicked.

-> **onChange** - Triggered when the value of an input field changes.

-> **onSubmit** – Triggered when a form is submitted.

-> **onMouseEnter** – Triggered when mouse enters an element

-> **onMouseLeave** – Triggered when mouse leaves an element.

-> **onKeyDown** – Triggered when a key is pressed.

-> **onFocus** – Triggered when an input field gains focus.

Example:-

**1. onClick**

-> function ClickExample() {

const handleClick = () => {

alert("Button clicked!");

};

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

}

export default ClickExample;

**2. onChange**

-> import { useState } from "react";

function ChangeExample() {

const [text, setText] = useState("");

const handleChange = (event) => {

setText(event.target.value);

};

return (

<div>

<input type="text" value={text} onChange={handleChange} />

<p>You typed: {text}</p>

</div>

);

}

export default ChangeExample;

**3. onSubmit**

-> import { useState } from "react";

function SubmitExample() {

const [name, setName] = useState("");

const handleSubmit = (event) => {

event.preventDefault(); // Prevents page refresh

alert(`Submitted: ${name}`);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

value={name}

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

placeholder="Enter your name"

/>

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

</form>

);

}

export default SubmitExample;

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

Ans :- class components, event handlers do not automatically bind to the components instance (this). If you directly pass an events handler like this.handleClikc,this inside the method will be undefined.

This happens because JavaScript treats event handlers as callback functions, and when they are passed to an event, their this context is lost.

**Conditional Rendering**

**Q1: What is conditional rendering in React? How can you conditionally render elements in a React component?**

Ans:- Conditional rendering in React means **dynamically displaying UI elements** based condition , like user authentication, loadings states, or data availability.

You can conditionally render elements using different techniques based on your use case.

-> if-else

-> Ternary Operator

-> Logical AND

-> switch-case

-> IIFE (Immediately Invoked Functions Expressions)

Example :-

function Login({isLogin}){

If(isLogin){

Alert(“welcome back”)

}else {

Alert(“please login”)

}

}

export default Login

**Q2: Explain how if-else, ternary operators, and && (logical AND) are used in JSXfor conditional rendering.**

Ans : - JSX doesn’t support direct if-else statement inside JSX. Instead, we use deffrent approaches like if-else before JSX, ternary operators, and logical AND (&&) to conditionally render elements.

**1. Using if-else (Before JSX)**

-> React does not allow if-else inside JSX, so you must use it before returning JSX.

**2. Using Ternary Operator (? :) (Inside JSX)**

-> For inline conditions, you can use the ternary operator inside JSX

**3. Using Logical AND (&&) (For One Condition)**

-> Use && When you only need to render something if a condition is true , and render nothing otherwise

**Lists and Keys**

**Q1: How do you render a list of items in React? Why is it important to use keyswhen rendering lists?**

Ans :- You can render a list of items using the .map() method. This method allows to iterate over an array and return JSX elements dynamically.

**\* Important to Use keys when Rendering List**

-> Keys in React are usd to uniquely identify element in a list They helps React efficiently update and re-render component by tracking changes

**1. Performance Optimization**

-> Helps React quickly identify which item’s changed, added, or removed.

**2. Prevents Unnecessary Re-Renders**

-> Without keys, React may re-renders the entire list instead of updating only changed items.

**3. Avoid UI Bugs**

-> Incorrect or missing keys can cause UI inconsistencies, such as incorrect state update.

**Q2: What are keys in React, and what happens if you do not provide a unique key?**

Ans :- Keys In React are identifiers assigned to elements when rendering lists. They help React efficiently update and re-render components by identifying which items have changed, been added , or removed.

**\* If You Do Not Provide a Unique Key**

-> If you don’t provide a unique key , React will show a warning in the console:” Each child in a list should have a unique “key” props.”

**Forms in React**

**Q1: How do you handle forms in React? Explain the concept of controlled components.**

Ans : - In React forms are handled using state to control input values. This ensure that React manages the form data instated of the browsers.

**\* Controlled Components :**

-> A controlled component is a form element whose value is controlled By React state.

**Q2: What is the difference between controlled and uncontrolled components in React?**

Ans :-

**1. Controlled Components**

-> **State Management** - Managed by React using useState or useReducer

-> **Data Handling** – React state holds the form values.

-> **Real-Time Updates** – Updates state on every change ( onChange event).

-> **Validation** – Easy to validate in real-time.

-> **Example** – Uses value and onChange

**2. Uncontrolled Components**

-> **State Management** - Managed by the DOM using ref.

-> **Data Handling** – Form values are accessed via ref.

-> **Real-Time Updates** – Does not update React state on every change.

-> **Validation** – Validation needs to be done manually.

-> **Example** – Uses ref to access values.

**Lifecycle Methods (Class Components)**

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

Ans : - Lifecycle methods in React class component are special methods that get called at different stages of a component’s existence. These methods allowed developer to control an respond to changes in the component lifecycle, sach as when it is created , or removed from the DOM.

\* **Phases of a Component’s Lifecycle**

**1. Mounting (Component Creation & Insertion in DOM)**

* **constructor()** – Initialization state and props
* **render ()** – Return JSX to display the UI
* **componentDidMount()** – Runs after the component is added to the DOM (used for API calls, event listeners,etc.).

**2. Updating (When the component updates due to state/props changes)**

* **shouldComponentUpdate()** - Decide whether the component should re-render.
* **render()** – Updates the UI with new Data.
* **componentDidUpdate()** – Runs after the update is applied (used for fetching new data updating DOM, etc).

**3. Unmounting (When the component is removed from the DOM)**

* **componentWillUnmount()** – Runs before the component is removed (used for cleanup tasks like removing event listeners).

**Q2: Explain the purpose of componentDidMount(), componentDidUpdate(),and componentWillUnmount().**

Ans : -

**1. componentDidMount() (Runs after the component is added to the DOM)**

* Purpose: Used for initializing tasks like API class, setting up event listeners, or updating the DOM.
* Runs only once after the first render.

**2. componentDidUpdate(prevProps,preveState) (Runs after state or props change)**

* Purpose : Used for performing side effects after a component updates, like fetching new data when props changes.
* Runs after every re-renders (unless prevented by shouldComponentUpdate()).

**3. componentWillUnmount() (Runs before the component is removed from the DOM)**

* Propose : - Used for cleanup task like removing event listeners, canceling API class, or learing timers.
* Runs only once before the component is destroyed.

**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 allow functional components to use and lifecycle features without needing a class component. Introduced in React 16.8, hooks make functional components more powerful and easier to manage.

**1. useState() Hooks (Manages State in Functional Components)**

-> The useState () kook lets you add state to functional component.

**2. useEffect() Hooks (Handles Side Effect like API Calls, Subscription,etc)**

-> The useEffect () hook allows us to perform side effects in functional components (similar to componentDidMount) , componentDidUpdate, and componentWillUnmount in class component.

**Q2: What problems did hooks solve in React development? Why are hooks.**

Ans :- Before **React Hooks** (introduced in **React 16.8**), developers primarily used **class components** to manage state and lifecycle methods. However, class components had several issues:

1. **Complex State Management**
   * Class components required this.state and this.setState, making state management **verbose** and **harder to read**.
   * Hooks like useState() simplify state handling in functional components.
2. **Messy Lifecycle Methods**
   * In class components, lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount) **split logic across multiple methods**, making code **harder to maintain**.
   * useEffect() allows handling side effects **in one place** and avoids unnecessary lifecycle method usage.
3. **Reusability Issues**
   * Class components made it **difficult to reuse stateful logic** because lifecycle methods were tied to classes.
   * Hooks enable **custom hooks**, allowing developers to **reuse logic easily**.
4. **Tightly Coupled Logic**
   * In class components, different pieces of logic (e.g., fetching data, handling events) were **scattered across lifecycle methods**, making components **harder to read**.
   * Hooks allow **separating concerns** inside functional components.
5. **Performance Issues**
   * Unnecessary re-renders and **memory usage** in class components could slow down performance.
   * useMemo() and useCallback() help **optimize performance** in functional components.

**\* why Are Hooks Important?**

1. **Simplifies Code** → Functional components with hooks are shorter and easier to read **than class components.**
2. **Better Reusability** → Custom hooks allow code reuse without complex patterns like Higher-Order Components (HOCs).
3. **Improved Performance** → Hooks prevent unnecessary re-renders and optimize state management.
4. **No Need for Classes** → Hooks allow developers to use state and lifecycle features in functional components.

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

Ans :- useReducre is a React hook used for managing complex state logic in functional components. It is an alternative to useState, especially when the state has multiple sub-vlaues or complex updates.

1. Import useReducer from React.

2. Define a reducer function to handles state updates.

3. Initialize the state.

4 Use useReducer() in the component.

5 Dispatch actions to update state.

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

Ans: -

**1. useCallback()**

-> memoizes a function so that is not recreated on every render.

-> When passing a function as prop to child components (prevents re-creation on every render).

-> When using a function inside useEffect() to avoid unnecessary re-runs.

**2. useMemo()**

-> useMemo() memoize the result of a function to avoid re-calculating expensive computation’s on every render.

-> When computations are expensive (e.g, filtering large list, complex calculations).

-> When derived state values should only be recalculated when dependencies change.

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

Ans :- Both useCallback and usememo are used for performance optimization in react, but they serve different purposes.

**1. useCallback()**

-> **Purpose** – Memoizes a function

-> **Return** – a function

-> **Use Case** – Prevents unnecessary functions recreation.

-> **Example Usage** – Passing functions as props to child components.

-> **Re-runs when?** - Any dependency in [] changes.

**2. useMemo()**

-> **Purpose** – Memoizes a computed value

-> **Return** – a value (result of computation)

-> **Use Case** – Prevents expensive re-computation recreation.

-> **Example Usage** – Passing functions as props to child components.

-> **Re-runs when?** - Any dependency in [] changes.

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

Ans :- useRef is React Hooks that allows to you to persist values across renders without causing a re-render. It is commonly used to reference DOM elements directly or store mutable values that do not trigger re-renders when update’s.

**1. Accessing DOM Elements (Direct Manipulation)**

-> You can use useRef to get references to a DOM element , like an input field.

**2. Storing Mutable Vlaues Without Re-Rendering**

-> Unlike useState, updating useRef doesn’t trigger a re-render.

**3. Stroring Previous State Value**

-> You can use useRef to track the pervious state value.

**Routing in React (React Router)**

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

Ans :- React Router is a standard library for routing in React applications. It enables navigation between different component while maintaining the single-page-application (SPA) behavior. Instead of making full-page reloads, it updates the UI dynamically based on the URL, providing a smooth user experience.

**\* React Router Handle Rooting in SPAs?.**

-> React Router uses Client-Side Routing (CSR), meaning it manages vanigation without sending requests to the server’s for every page change. Here’s how it works:

**1. History API & Virtual DOM:**

* React Router leverages the History API (pushState,replaceState) to update the browser’s URL without reloading the page.
* When the URL changes, React updates only the components that need to be rendered.

**2. Route Matching:**

* It checks the current URL and matches it with defined routes.
* Based on the matching route, it renders the respective React components.

**3. Nested Routing:**

* You can define nested routes for better organization.
* Example: / dashboard -> main dashboards, /dashboard/profiles -> Profile section inside the dashboard.

**4. Dynamic Routing:**

* React Router allows dynamics parameters in the URL (/product/:id).
* It extracts the value and presses it to the components.

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

Ans: -

**1. BrowserRouter**

* In is the main wrapper component that enables routing in a React app.
* In listings to URL changes and renders the appropriate component.

**2. Route**

* Used to defined specific paths and the components to render when the URL matches.

**3. Link**

* Used for navigating between pages without pages full-page reloades.
* Works like an <a> tag but doesn’t trigger a server request.

**4. Switch (Deprecated in React Router v6, replaced by Routes)**

* Used to group multiple Route Components.
* Ensured that only the first matching route was rendered.

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

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

Ans :- RESTful Web Services are web-based APIs that follow the principles of REST (Representation State Transfer). They allow communication between client and server using standard HTTP methods.

**1. Stateless**

-> Each request from a client to a server must contain all the necessary information; the server does not store any client context.

**2. Clint-Server Architectures**

-> The lient and server are independent, allowing better scalability and flexibility.

**3. Uniform Interface**

-> RESTful services use standard HTTP method’s

* GET -> Retrieve data
* POST -> Create a new recourse
* PUT -> Update an existing recourse
* DELETE -> Remove a resource

**4. Resource-Based**

-> Everything in REST is considered a resource and is accessed using URLs (e.g, /user/1 to get user data)

**5. Use of JSON/XML**

-> Data is typically exchanged in JSON (Java Script Object Notation) or XML format, with JOSN begin more popular due to its lightweight nature.

**6. Cacheable**

-> Response can be cached to improve performance.

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

Ans :- JSON-Server is a lightweight Node.sj package that provides a moke REST API without writing back end code. Its useful for testing and property ping frontend application without setting up a full backend.

**1. Install JOSN-Server**

-> npm install json-server --save-dev

**2. Create a Fake Database (db.json)**

-> In your React project, create a db.json file and add some sample data:

**3. Start JSON-Server**

-> npx json-server --watch db.json --port 5000

**4. Fetch Data from JSON-Server in React**

-> Now, use **fetch()** or **Axios** to get data inside your React component.

**5. Add Data Using POST Request**

-> To add a new user to JSON-Server:

**6. Update Data Using PUT Request**

-> To update an existing user:

**7. Delete Data Using DELETE Request**

-> To delete a user:

**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 : - In React , you can fetch data from a JSON-Server API using either fetch() (built-in JavaScript functions) or axios (a third-party HTTP client).

**1. Fetch Data Using fetch()**

-> The fetch() functions is a built-in JavaScript methods used for making network requests.

**\* Pros of fetch():**

* Built into JavaScript (no need to indtall extra libraries).
* Simple and lightweight.
* Does not automatically handle errors or requests cancelation.

**2. Fetch Data Using axios()**

-> axios is a third-party library that provides a cleaner and easier way to handle API requests.

-> Install axios First

-> npm install axios

**\* Pros of axios()**

* Supports automatic JSON parsing (no need for .json()).
* Handle errors better with built-in error handling.
* Supports request cancellation, timeouts, and interceptors.
* Requires installing an external library.

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

Ans :- Firebase is a backend-as-a-Service (BaaS) platform by Google that provided various backend services for mobile and web applications. It helps developers build, deploy and scales apps without managing a complex backend.

It is widely used in React, Angular, Vue.js, and native mobile apps for authentication, databases, could functions, and more.

\* Features of Firebase

-> Firebase offers a variety of features categorized into different sections.

**1. Authentication (User management)**

-> Firebase Authentication -> Provides easy-to-use authentication methods

* Email/Password Login
* Google,Facebook, Twitter, GitHub Login
* Phone Number Authentication
* Anonymous Authentication

**2. Realtime Database (NoSQL Database)**

-> Firebase Realtime Database -> A cloud-based NoSQL database that syncs data In real-time across user’s.

**3. Firestore (Cloud Database)**

-> Cloud Firestore -> A more advanced NoSQL database that supports:

* Real-time syncing
* Offline support
* Scalability (better then Realtime Database)

**4. Cloud Storage**

-> Firebase Cloud Storage -> Used for storing and serving files (images, video’s, PDFs).

**5. Firebase Hosting**

-> Fast & secure hosting for web applications with free SSL certificates.

-> Supports React,Vue.js,Angular , and static websites.

**6. Cloud Functions (Serverless Backend)**

-> Run backend logic without managing a sever.

-> Used for processing payments, sending notifications, etc.

**7. Firebase Analytics**

-> Tracks user behavior in real-time.

-> Helps with performance monitoring and crash reports.

**8. Firebase Cloud Messaging (FCM)**

-> Send push notifications to mobile and web apps.

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

Ans :- When working with APIs in React, it’s essential to handle errors loading states properly to ensure a smooth user experience. Without Proper handling, users might see a blank screen, outdated data, or experience app crashes.

* **Loading State ->** Shows a spinner or message while fetching data.
* **Error Handling ->** Displays an error message if the API fails.
* **Success State ->** Shows the fetched data after a successful request.

**Context 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 built-in feature that provides a way to manage global state across multiple components without having to pass props manually at level (props drillings). It is mainly used when state need state needs to be shared among deeply nested components.

**How Context API Work**

Context API has three main parts

**1. React.createContext()**

-> Create a Contrxt object.

**2. Provider**

-> Provides the state to child components.

**3. Consumer or useContext() Hook**

-> Accesses the shared state I child components.

\* **How to Use Context API for Global State Management.**

**Step 1: Create a Context**

**Step 2: Create a Provider Component**

-> The Provider Component will wrap the part of the ap that needs access to the global state

**Step 3: Wrap Your App with the Provider**

-> Now, wrap your main App.js or any parent component with userProvider.

**Step 4: Use useContext() in Child Components**

-> Now, any child component inside UserProvider can access the global 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 to state across multiple component without props drilling.

**1. createContext() – Create a Context**

-> createContext() is used to create a Context a Context object that holds shered data.

**2. UserContext.Provider – Providing the state**

-> To share state, warp the required components with UserContext.provider.

-> The Provider Accept a value, prop , which is the shared state.

**3. useContext() – Accessing Shared State**

-> Instead of using Consumer, the useContext() hooks makes it easier to access shared data.

**State Management (Redux, Redux-Toolkit or Recoil)**

**Q1: What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store**

Ans:- React is a state management library used in JavaScript applications, particularly with React, to manage and centerlize application state. It helps in handling complex state logic, making state predictable, easy to debug, and maintable.

\* **Why Use Redux in React?**

**1. Centralized State Management: -** Keeps all the application states in a single store, making state management easier.

**2. Predictable: -** Uses pure functions (reducers) to update the state, ensuring predictable behaviors.

**3. Easier Debugging & Time Travel: -** Tools like Redux DevTools allow developers to track and revert state changes.

**4. Improves Performance: -** Prevents unnecessary re-render by managing state effectly.

**5. Scalability: -** Ideal for large-scale applications with complex state interactions.

**Core Concepts of Redux**

**1. Actions**

* Auctions are plain JavaScript objects that describe what should happen in the application.
* They must have a type property, which tells the reducer what kind of operation to perform.
* Action can also carry data using a payload.

**2. Reducers**

* A pure function that takes the current state and an action, then returns a new state.
* It never modifies the existing state but returns a new object.

**3. Store**

* The store is a centralized container that holds the application state.
* It Provides methods like:

**1. getState() ->** Get current state

**2. dispatch(action) ->** Send actions to modify state

**3. subscribe(listener) ->** Listen for state changes

**Q2: How does Recoil simplify state management in React compared to Redux?**

Ans:- Recoil is Morden state management library for React that simplifies managing global state compared to Redux. It provides a more intuitive and React-frendly approach to state management by leveraging Atmos and selectors instated of a complex store structure.

**1. Less Boilerplate: -** No need for action reducers, or a critical store like Redux.

**2. Simpler API: -** Uses atoms (state units) and selectors (computed state) instated of dispatching auctions,

**3. Better Performance: -** Only component using an atom re-render, avoiding unnecessary updates like in Redux

**4. Built-in Async Support : -** No need for middleware (like Redux Thunk) to handle async operations.

**5. React-Friendly : -** Work naturally with React hooks (useRecoilState,useRecoilValue).