1. What is the purpose of useCallback & useMemo Hooks?

ANS: Both useCallback and useMemo are React Hooks that help improve the performance of your app by preventing unnecessary re-renders or recalculations.

useCallback

- Memoized a function so that it doesn't get re-created on every render.
- When you pass a function as a prop to a child component, and you want to ensure it doesn't trigger unnecessary renders of that child component.
- React re-creates functions on every render.
- useCallback ensures that the same function instance is reused, avoiding unnecessary updates.

useMemo

- Memorized the result of calculation or computation so that it doesn't get re-calculated on every render.
- When you have expensive(time-consuming) computations and you only want to re-calculate the result if specific inputs changes.
- Saves processing power by skipping unnecessary recalculations.

useCallback: Memoized functions.

useMemo: Memoized values.

2. What is the purpose of useCallback & useMemo Hooks?

ANS: The purpose of the useCallback hook is to memoized a function so that it is not re-created on every render unless its dependencies change.

- It helps optimize performance, especially when passing functions as props to child components.
- Unnecessary re-render of child components that depend on the same function.
- Avoids re-creating functions that do the same thing, saving memory and processing time.

useMemo:

- The purpose of the useMemo hook is to memoized the result of an expensive calculation and recompute it only when its dependencies change.
- This prevents costly recalculations during every render.
- Optimizes performance by avoiding unnecessary recalculations.
- Useful for computations that take a lot of time or resources.

3. What is useRef ? How to work in react app?

ANS: useRef is React Hook that creates a mutable object to store a value or reference that persists across renders.

- It provides a way to directly access or modify a DOM element or store any value without triggering a re-render.

Features

1. Holds a reference to a value:

- The object returned by useRef has a ". current" property where you can store any mutable value.
- Updating ". current" does not cause re-renders.

2. Access DOM elements directly:

- You can use useRef to reference and manipulate a DOM element, bypassing React's state and props system.

```javascript
const refContainer = useRef(initialValue);
```

  - refcontainer is an object with a ". current" property.
  - initialValue is the starting value of the reference.

Use Cases for useRef:

- accessing and modifying DOM elements (e.g., input focus, scrolling).
- Storing mutable values that don't need to trigger re-renders (e.g., timers, instance variables).
- Tracking previous values (e.g., a previous state or prop)

- useRef is not for the triggering the re-renders.
- If you want to trigger a re-render, use "useState" instead.
- Changes to .current are not visible in the UI unless explicitly read or updated during render.

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

ANS: JSON-Server is a simple, lightweight tool that provides a full fake REST API file as the database.

- It allows developers to quickly prototype and test applications without setting up a real backed.
- It is widely used for development purposes, especially in React projects.

Key features:

- Provides a REST API for CRUD operations.
- Uses a JSON file (e.g., db.json) as the database.

How to use JSON-Server:

Step 1: install JSON-Server. (npm i json-server@0.17.4)

Step 2: create a db.json file.

- This will act as your fake database.

Step 3: start JSON-Server

Why use JSON-Server in React Projects?

- Set up a backend API in seconds.
- Test your React app's API calls without building a real backend.

5. 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 a React app, you can use either the

- fetch API (built in JavaScript)
- axios

both allow you to make HTTP request to interact with the JSON-Server.

1. Using fetch()

- The fetch function is a built-in JavaScript function for making HTTP requests. It returns a promise that resolves to the Response object.

How it works:

- fetch("URL"): Sends a GET request to the JSON-Server API.
- .then((response) => response.json()): Converts the response to JSON format.
- .then((data) => setData(data)): Updates the React state with the fetched data.
- .catch((error) => console.log(error)): Catches and handles errors.

2. Using axios

- axios is a third-party library for making HTTP requests.

- It simplifies API requests by providing an easy-to-use syntax and automatic handling of JSON data.
- npm install axios.

How it works:

- **axios.get("URL"):** Sends a GET request to the JSON-Server.
- **response.data:** Axios automatically parses JSON responses, so you directly access the data.
- **.then((response) => setData(response.data)):** Updates the React state with the fetched data.
- **.catch((error) => console.log(error**)): Catches and handles errors.

Role of fetch() or axios() in API requests

- Both fetch and axios allow you to make HTTP requests like GET, POST, PUT, DELETE.
- They help retrieve data from an API, process it, and update your app's state.
- Both handle network errors and server-side errors so you can respond gracefully in your app.
- You can use them to send data to the backend.

6. 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 feature that allows you to create and manage a global state that can be shared across multiple components in a React app without having to pass props manually through every level of the component tree.

- It provides a way to avoid **"prop drilling"**—where props are passed down multiple layers of components, even if intermediate components don't use them.

How it works:

- creates a context object.
- Wraps the part of your component tree that needs access to the global state.
- Passes the state value to all its child components via the value prop.
- Access the data provided by the context.
- Use the useContext Hook.

Benefits

- No need to pass props down multiple levels of components.
- Make it easier to manage and share state across compoents.
- Simpler than using libraries like Redux for small to medium-sized projects.