Gagan doddanna

🧠 React Interview Q&A: `useState`, `useEffect`, `async/await`, Data Fetching, and React Query

---

❓Q1: What is `useState` in React?

🗣️ **A:**
`useState` is a React Hook that lets me store and manage data inside a function component.
Whenever I update that state, React re-renders the component with the new value.

📦 **Example:**

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

🧠 **Analogy:**
It's like a whiteboard. I can write a value on it, and React will update the display automatically whenever I change it.

❓Q2: What is `useEffect` used for?

🗣️ **A:**
`useEffect` lets me perform side effects in a component — such as fetching data, setting up subscriptions, or manipulating the DOM.

📦 **Syntax:**

```
useEffect(() => {
  // side effect
}, []);
```

🧠 **Analogy:**
Think of it as a to-do list that runs after the component appears on screen.

❓Q3: What does the `[]` (empty array) mean in `useEffect`?

🗣️ **A:**
The empty array is the dependency array.
If it's empty, the effect runs only once — when the component first mounts (just like `componentDidMount` in class components).

If I add dependencies like `[id]`, it re-runs every time `id` changes.

❓Q4: Can `useEffect` be async?

🗣️ **A:**
No — `useEffect` itself can't be `async` because React expects it to return a cleanup function, not a Promise.

✅ **Best practice:**
I define an async function inside `useEffect`, then call it:

```
useEffect(() => {
  const fetchData = async () => {
    const res = await fetch(url);
    const data = await res.json();
    setState(data);
  };
  fetchData();
}, []);
```

❓ Q5: Why do we use `async/await`?

🗣️ **A:**
`async/await` helps me write asynchronous code (like API calls) in a clean and readable way.

🧠 **Analogy:**
It's like ordering pizza:

- **`async` says "this might take time"**
- **`await` means "pause here until the pizza arrives"**

❓ Q6: How do I show API data and the average salary in React?

🗣️ **A:**

1. **I use `useEffect` to fetch data when the component mounts**
2. **I use `useState` to store the data and average salary**
3. **I use `reduce` to calculate the total, then divide by length**
4. **Then I display it in JSX**

✅ **Snippet:**

```
const total = data.reduce((sum, emp) => sum + emp.salary, 0);

setAverageSalary(total / data.length);
```

❓ Q7: What if I want to re-fetch data after updating something?

🗣️ **A:**
If I update something on the server (like salary), I can either:

✅ **Use a `refreshTrigger` state:**

```
const [refresh, setRefresh] = useState(0);


useEffect(() => {
  fetchData();
}, [refresh]);


// After update
setRefresh(prev => prev + 1);
```

✅ Or, better: Use React Query and call `refetch()` or `invalidateQueries()`.

❓Q8: What is React Query, and why is it useful?

🗣️ **A:**
React Query is a library that handles:

- Data fetching
- Caching
- Auto re-fetching
- Loading and error states

✅ I don't need to manually manage `useState` or `useEffect`. It simplifies complex data flows.

🧠 **Analogy:**
It's like a smart butler — it fetches data, remembers it, refreshes it, and shows a loading spinner while you wait.

❓Q9: How would you fetch data using React Query?

🗣️ **A:**

```
const { data, isLoading } = useQuery({
  queryKey: ['employees'],
  queryFn: fetchEmployees,
});
```

If I mutate data:

```
const mutation = useMutation({...});
```

And on success:

```
queryClient.invalidateQueries(['employees']);
```

❓Q10: What are best practices when fetching data in React?

🗣️ **A:**
✅ Show a loading state
✅ Handle errors with `try/catch`
✅ Use `abortController` to cancel on unmount
✅ Prefer `async/await` over `.then()`
✅ Avoid stale data with functional updates
✅ Use tools like React Query for clean architecture
✅ Use `emp.id` as React key (not `index`)
