**React Hooks & Context (Intro, Rules, State, Effect Hook, Custom Hooks) (15)**

**31. What are the two main rules of hooks in React?**

1. Only call hooks at the top level (not inside loops, conditions, or nested functions).
2. Only call hooks from React function components or custom hooks.

**32. How does useState differ from setting state in class components?**

* Class: this.setState merges state updates.
* Function + useState: setState replaces the state value, so you must merge manually if it’s an object.

**33. How do you update state based on the previous value using useState?**

setCount(prevCount => prevCount + 1);

**34. What are some common use cases for the useEffect hook?**

* Data fetching
* Subscribing to events (e.g., WebSocket)
* Updating document title
* Setting up timers

**35. How do you clean up side effects in useEffect?**  
Return a cleanup function from useEffect:

useEffect(() => {

const id = setInterval(() => console.log('Tick'), 1000);

return () => clearInterval(id);

}, []);

**36. What happens if you forget to provide a dependency array in useEffect?**  
The effect runs after every render, which can cause infinite loops for state-updating effects.

**37. What is the difference between useContext and prop drilling?**

* Prop drilling: Passing data down through many levels via props.
* useContext: Accessing shared state directly from any child without passing props.

**38. How do you create a React Context provider and consumer using hooks?**

const ThemeContext = React.createContext();

function App() {

const [theme, setTheme] = useState('dark');

return (

<ThemeContext.Provider value={{ theme, setTheme }}>

<Child />

</ThemeContext.Provider>

);

}

function Child() {

const { theme } = useContext(ThemeContext);

return <div>{theme}</div>;

}

**39. How do you avoid re-renders when passing context values?**

* Use useMemo to memoize context value:

const value = useMemo(() => ({ theme, setTheme }), [theme]);

**40. Give an example of a custom hook for form input handling.**

function useInput(initial) {

const [value, setValue] = useState(initial);

const onChange = e => setValue(e.target.value);

return { value, onChange };

}

**41. What is the difference between useEffect and useLayoutEffect?**

* useEffect: Runs after the browser paints → non-blocking.
* useLayoutEffect: Runs before the paint → can block rendering for DOM reads/writes.

**42. How can you create a custom hook for API fetching?**

function useFetch(url) {

const [data, setData] = useState(null);

useEffect(() => {

fetch(url).then(res => res.json()).then(setData);

}, [url]);

return data;

}

**43. What is the difference between multiple useEffect hooks vs a single one with multiple logics?**  
Multiple useEffect hooks keep unrelated logic separate, making code cleaner and easier to maintain.  
They allow different dependency arrays, preventing unnecessary re-runs.

A single useEffect with multiple logics mixes concerns, which can cause harder debugging and extra renders.

**44. Why can’t hooks be used inside conditional statements?**  
Hooks rely on consistent call order for state mapping; conditional execution would break this order.

**45. How would you share logic between multiple components using hooks?**  
Encapsulate the logic in a custom hook and reuse it across components.

**API Integration with Fetch & Axios (GET, POST, PUT, DELETE) (10)**

**46. What is the difference between Fetch API and Axios in React?**

* Fetch: Native, returns promises, minimal features.
* Axios: External library, supports request cancellation, interceptors, automatic JSON parsing.

**47. How do you make a GET request using Axios in useEffect?**

useEffect(() => {

axios.get('/api/data').then(res => setData(res.data));

}, []);

**48. How do you handle errors in Axios requests?**

axios.get('/api/data')

.catch(err => console.error(err));

**49. How do you send POST requests with JSON body using Axios?**

axios.post('/api/data', { name: 'John' });

**50. What are the differences in default headers between Fetch and Axios?**

* Fetch: Does not set Content-Type automatically for JSON.
* Axios: Automatically sets Content-Type: application/json for objects.

**51. How do you send a PUT request with Axios to update existing data?**

axios.put('/api/data/1', { name: 'Updated' });

**52. How do you delete data from an API using Axios?**

axios.delete('/api/data/1');

**53. How do you cancel an Axios request in progress?**

const controller = new AbortController();

axios.get('/api/data', { signal: controller.signal });

controller.abort();

**54. What is an Axios interceptor and why would you use it?**  
A function that runs before a request or after a response to modify them .It used for auth tokens, logging, or error handling.

**55. How do you handle loading states during API requests in React?**

setLoading(true);

axios.get('/data').then(res => {

setData(res.data);

}).finally(() => setLoading(false));

**Pure Components (5)**

**56. What is a Pure Component in React?**  
A component that re-renders only if its props or state change (shallow comparison).

**57. How do Pure Components improve performance?**  
They prevent unnecessary re-renders by skipping updates when data hasn’t changed.

**58. How is React.memo related to Pure Components in function components?**  
React.memo wraps a functional component to give it Pure Component-like behavior.

**59. What kind of props changes will cause a Pure Component to re-render?**  
Any shallow change in props (primitive values changed, new object/array references).

**60. What are the limitations of Pure Components?**

* Only does shallow comparison (may miss deep changes).
* Can cause bugs if relying on mutable objects.
* Not beneficial for very cheap renders.