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

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

- Only call hooks at the top level
- Only call hooks from React functions

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

useState returns a state value and a setter function. replaces the value completely

In class components, this.state is an object and you use this.setState (which merges updates).

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

```
const [count, setCount] = useState(0);
setCount(prev => prev + 1);
```

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

Fetching data, Subscribing to events, Updating the document title, Setting timers/intervals

35. How do you clean up side effects in useEffect?

```
useEffect(() => {
  const id = setInterval(() => {}, 1000);
  return () => clearInterval(id);
}, []);
```

36. What happens if you forget to provide a dependency array in useEffect?

The effect runs after every render, which may cause performance issues or infinite loops.

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

- useContext: lets you access data directly from context.
- Prop drilling: you pass props through multiple components unnecessarily.

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

```
const UserContext = React.createContext();

function App() {
  return (
    <UserContext.Provider value="Raksha">
      <Child />
    </UserContext.Provider>
  );
}

function Child() {
  const user = React.useContext(UserContext); // consume
  return <h3>Hello {user}</h3>;
}
```

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

- Memoize values with useMemo.
- Memoize callbacks with useCallback.

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, reset: () => setValue(initial) };
}
```

```
function Form() {
  const name = useInput("");
  return (
    <>
      <input {...name} placeholder="Enter name" />
      <p>Name: {name.value}</p>
    </>
  );
}
```

41. What is the difference between useEffect and useLayoutEffect?

- useEffect: Runs asynchronously after the browser has painted the DOM. This means the browser can update the screen and display content to the user before the code inside useEffect is executed.
- useLayoutEffect: Runs synchronously after all DOM mutations but before the browser paints the screen. This means the code inside useLayoutEffect blocks the browser's rendering until it completes.

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

```
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  useEffect(() => {
    let isMounted = true;
    fetch(url).then(res => res.json()).then(json => {
      if (isMounted) {
        setData(json);
        setLoading(false);
      }
```

```
    });
    return () => { isMounted = false; };
  }, [url]);
  return { data, loading };
}
function Users() {
  const { data, loading } = useFetch("https://jsonplaceholder.typicode.com/users");
  return loading ? <p>Loading…</p> : <ul>{data.map(u => <li key={u.id}>{u.name}</li>)}</ul>;
}
```

43. What is the difference between multiple useEffect hooks vs a single one with multiple logics?

- Multiple: Each useEffect has its own independent dependency array. This allows for precise control over when each specific side effect re-runs. If only one dependency changes, only the useEffect that relies on that dependency will execute, preventing unnecessary re-renders or side effects.
- Single: f a single useEffect has multiple logics, its dependency array might become large and encompass all dependencies for all the contained logics. This can lead to unnecessary re-runs of all the logics within that useEffect even if only a single, unrelated dependency changes.

44. Why can’t hooks be used inside conditional statements?

Because hook order must remain consistent across renders. If used conditionally, React can’t match hook state correctly.

45. How would you share logic between multiple components using hooks?

By extracting the logic into a custom hook and reusing it in different components.

```
function useWindowWidth() {
  const [w, setW] = useState(window.innerWidth);
  useEffect(() => {
    const onResize = () => setW(window.innerWidth);
    window.addEventListener("resize", onResize);
    return () => window.removeEventListener("resize", onResize);
  }, []);
```

```
  return w;
}
function CompA() {
  const width = useWindowWidth();
  return <p>CompA width: {width}</p>;
}
function CompB() {
  const width = useWindowWidth();
  return <p>CompB width: {width}</p>;
}
```

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

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

- Fetch API is built-in, returns promises, needs manual JSON parsing.
- Axios is external, automatically parses JSON, has better error handling and extra features like interceptors.

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

```
import React, { useEffect, useState } from "react";
import axios from "axios";
function Users() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    axios.get("https://jsonplaceholder.typicode.com/users")
      .then(res => setUsers(res.data))
      .catch(err => console.error(err));
  }, []);
```

```
  return <ul>{users.map(u => <li key={u.id}>{u.name}</li>)}</ul>;
}
```

48. How do you handle errors in Axios requests?

```
axios.get("/wrong-url")
  .then(res => console.log(res))
  .catch(err => {
    if (err.response) {
      console.error("Server error:", err.response.status);
    } else if (err.request) {
      console.error("No response:", err.request);
    } else {
      console.error("Error:", err.message);
    }
  });
```

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

```
axios.post('/api/users', { name: "Raksha", age: 22 })
  .then(res => console.log(res.data));
```

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

- Fetch → doesn't set Content-Type automatically.
- Axios → automatically sets Content-Type: application/json for JSON requests.

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

```
axios.put('/api/users/1', { name: "Updated Name" })
  .then(res => console.log(res.data));
```

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

```
axios.delete("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => console.log("Deleted:", res.status));
```

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

```
import axios from "axios";
const controller = new AbortController();
axios.get("https://jsonplaceholder.typicode.com/users", {
  signal: controller.signal
})
.then(res => console.log(res.data))
.catch(err => console.log("Cancelled:", err.message));
setTimeout(() => controller.abort(), 100);
```

54. What is an Axios interceptor and why would you use it?

- A function that runs before a request or response.
- Useful for adding auth tokens, logging, or handling errors globally.

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

```
function Posts() {
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(false);
  useEffect(() => {
    setLoading(true);
    axios.get("https://jsonplaceholder.typicode.com/posts")
      .then(res => setPosts(res.data))
      .catch(err => console.error(err))
      .finally(() => setLoading(false));
  }, []);
```

```
  if (loading) return <p>Loading…</p>;
  return <ul>{posts.slice(0,5).map(p => <li key={p.id}>{p.title}</li>)}</ul>;
}
```

Pure Components

56. What is a Pure Component in React?

A Pure Component is a React component that re-renders only if its state or props change shallowly. It implements shouldComponentUpdate with a shallow comparison.

57. How do Pure Components improve performance?

They avoid unnecessary re-renders by skipping updates when props and state are unchanged, which reduces rendering cost in large apps.

58. How is React.memo related to Pure Components in function components?

React.memo is the functional equivalent of PureComponent. It memoizes the component and re-renders only when props change.

```
const MyComponent = React.memo(function MyComponent(props) {
  return <div>{props.value}</div>;
});
```

59. What kind of props changes will cause a Pure Component to re-render?

When primitive values change and when object/array references change.

60. What are the limitations of Pure Components?

- Only does shallow comparison, so deep object/array changes might not trigger re-renders.
- May cause stale renders if props are mutated instead of replaced.