DAY 21: EVENING ASSESSMENT

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

Only call hooks at the top level not inside loops/conditions.

Only call hooks from React functions ,components or custom hooks.

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

useState: replaces the state value, doesn’t merge objects.

this.setState: shallow merges the update with previous state.

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/unsubscribing (e.g., events, sockets)

Updating DOM manually

Syncing with localStorage

35. How do you clean up side effects in 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?

Effect runs after every render → possible infinite loops or performance issues.

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

Prop drilling: manually pass props down many levels.

useContext: direct access from context provider, no middle props needed.

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

const ThemeContext = React.createContext();

function ThemeProvider({children}) {

const [theme, setTheme] = useState("light");

return (

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

{children}

</ThemeContext.Provider>

);

}

function Child() {

const {theme} = useContext(ThemeContext);

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

}

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

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

Split contexts (one for state, one for updater).

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 };

}

const name = useInput("");

<input {...name} />

41. What is the difference between useEffect and useLayoutEffect?

useEffect: async, runs after paint → doesn’t block UI.

useLayoutEffect: runs before paint → blocks UI until complete. Used for DOM measurements.

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

function useFetch(url) {

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

useEffect(() => {

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

}, [url]);

return data;

}

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

logics?

Multiple: cleaner, each effect handles one concern.

Single: harder to maintain, mixed concerns.

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

Because hook order must stay consistent between renders. Conditions change order → React loses track of states.

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

Use custom hooks → encapsulate logic, reuse across components.

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

Fetch: built-in, promise-based, minimal, manual JSON parsing.

Axios: external lib, auto JSON parsing, interceptors, cancellation, easier defaults.

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

useEffect(() => {

axios.get("/api/users")

.then(res => setUsers(res.data))

.catch(console.error);

}, []);

48. How do you handle errors in Axios requests?

axios.get("/api")

.catch(err => {

if (err.response) console.error("Server error", err.response);

else console.error("Network error");

});

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

axios.post("/api/users", { name: "Sahil", age: 22 })

.then(res => console.log(res.data));

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

Fetch: only sets Content-Type if body is a string.

Axios: automatically sets Content-Type: application/json for JSON and handles it.

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

axios.put("/api/users/1", { name: "Updated" })

.then(res => console.log(res.data));

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

axios.delete("/api/users/1")

.then(res => console.log("Deleted"));

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

const source = axios.CancelToken.source();

axios.get("/api", { cancelToken: source.token });

source.cancel("Request canceled");

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

axios.interceptors.request.use(config => {

config.headers.Authorization = "Bearer token";

return config;

});

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

const [loading, setLoading] = useState(false);

useEffect(() => {

setLoading(true);

axios.get("/api")

.then(r => setData(r.data))

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

}, []);

56. What is a Pure Component in React?

A component that re-renders only if its props/state actually change

57. How do Pure Components improve performance?

Avoids unnecessary re-renders → saves CPU and improves UI responsiveness.

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

React.memo wraps function components and makes them behave like Pure Components.

const Button = React.memo(({ label }) => <button>{label}</button>);

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

Any prop that fails shallow equality check (===). Objects/arrays/functions with new references cause re-render.

60. What are the limitations of Pure Components?

Only shallow compare → nested objects/arrays still cause re-renders.

Might over-optimize and complicate debugging.