Day-21 Evening Assessment

**React Hooks & Context:**

31. 1. Only call hooks at the top level.

2. Only call hooks inside react functions.

32. useState returns a state value and updater function. Class state is an object merged with setState; useState replaces state directly.

33. setCount(prev => prev+1);

34. common uses: data fetching, subcriptions, timers, DOM manipulating, syncing state with local storage.

35. Returning a clean up function inside useEffect:

useEffect(() => {

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

return () => clearInterval(id);

}, []);

36. Without dependency array, effect runs after every render, possibly causing infinite loops.

37. useContext provides values directly without passing through intermediate components.

38. const MyContext = React.createContext();

function Provider ({children }) {

const [value, setValue] = useState(“Hello”);

return (

<MyContext.Provider value = {value}>{children}</MyContext.Provider>

);

}

function Consumer(){

const value = useContext(MyContext);

return <p>{value}</p>;

}

39. use useMemo for context values or split contexts to avoid unnecessary re-renders.

40. Custom form hook:

function useInput(initial){

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

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

return {value, onChange};

}

41. useEffect runs after paint, non-blocking; useLayoutEffect runs synchronously before paint, blocking render.

42. function useFetch(url){

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

useEffect(() => {

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

}, [url]);

return data;

}

43. Multiple useEffecthooks separate concerns and make code clearer vs one large effect mixing all logic.

44. Hooks rely on call order; conditions break the order, causing React to lose tracks of hook state.

45. Create custom hooks to extract logic and reuse across components.

**API Integration:**

46. Fetch is built-in, minimal, and returns promises; Axios is a library with features like interceptors, automatic JSON, cancellation, and shorter syntax.

47. useEffect(() => {

Axios.get(“/api/data”).then(res => setData(res.data));

}, []);

48. use .catch(err => console.error(err)) or try/catch with async/await.

49. axios.post(“/api/users”, {name: “John”});

50. Fetch doesn’t set Content-Type: application/json automatically; Axios sets JSON headers by default.

51. axios.put(“/api/users/1”, {name: “updated”});

52. axios.delete(“/api/users/1”);

53. const source = axios.CancelToken.source();

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

source.cancel(“Request canceled”);

54. Interceptors allow modifying requests/responses globally.

55. Track loading with state:

setLoading(true);

axios.get(“/api”).then(res => {setData(res.data);}).finally(() => setLoading(false));

**Pure Components:**

56. A pure component is a class component that implements shouldComponentUpdate with a shallow prop/state comparison.

57. They skip re-rendering when props/state haven’t changed, improving performance.

58. React.memo is the functional component equivalent of PureComponent.

59. PureComponent re-renders when shallow comparison detects changes in props.

60. Limitations : shallow comparison only, may add complexity.