Mock Test

Topic: react hook
Difficulty: Intermediate
Total Questions: 5
Time Allowed: 10 minutes

Instructions:

1. Attempt all questions
2. Each question carries equal marks
3. Time allowed: 10 minutes

1. Explain the potential performance implications of using the `useEffect` hook with an empty dependency array (`[]`). How can this be optimized if you need to run an effect only once after the initial render?

A) It runs only once on mount, which is efficient. No optimization needed.
B) It runs on every render, causing unnecessary re-renders. Optimization requires using a ref to track component mount status.
C) It runs only once on mount, but can still cause performance issues if the effect is computationally expensive. Optimization might involve memoization or lazy loading.
D) It will throw an error. Empty dependency arrays are invalid.

2. You need to implement a custom hook that fetches data from an API and manages loading and error states. Describe the structure of this hook, including the use of states and the `useEffect` hook.

A) Use only useState for loading and error states, no useEffect needed.
B) Use useState for data, loading and error states, and useEffect to fetch data on mount and handle any errors.
C) Use useReducer for managing state transitions and useEffect to fetch data.
D) Use useRef to store the fetched data and useState for loading and error states.

3. What is the purpose of the `useCallback` hook and when would you prefer it over a simple function definition within a component?

A) It prevents unnecessary re-renders by memoizing the function.
B) It prevents memory leaks.
C) It is only used for asynchronous operations.
D) It's identical to a simple function; no practical difference.

4. Describe a scenario where `useMemo` is more efficient than directly calculating a value within a component's render function.

A) When calculating a simple value that doesn't involve complex calculations.
B) When the value depends on props that frequently change.
C) When calculating an expensive value that depends on props that change

infrequently.
D) When dealing with asynchronous operations.

5. How can you use custom hooks to improve code reusability and maintainability in a React application?

A) Custom hooks cannot improve code reusability.
B) Extract common logic related to state management, side effects, or data fetching into reusable custom hooks.
C) Custom hooks are only for complex state management.
D) Custom hooks only work with useEffect.

Answer Key

1. Correct Answer: C
Explanation: While an empty dependency array ensures the effect runs only once on mount, a computationally expensive effect could still impact performance. Memoization or lazy initialization of resources within the effect can improve efficiency.

2. Correct Answer: C
Explanation: While B is functional, using useReducer for managing state transitions leads to cleaner and more maintainable code, especially for complex state logic involved in data fetching.

3. Correct Answer: A
Explanation: useCallback memoizes the function, preventing unnecessary re-creation if the dependencies haven't changed. This is crucial for preventing unnecessary re-renders of child components that depend on that function as a prop.

4. Correct Answer: C
Explanation: useMemo memoizes the result of an expensive computation. If the dependencies haven't changed, it returns the cached value, avoiding redundant calculations. This is particularly beneficial when the computation is expensive and the dependencies rarely change.

5. Correct Answer: B
Explanation: Custom hooks encapsulate reusable logic, promoting cleaner, more maintainable code by reducing redundancy and improving readability across components.