CHEAT SHEET DE HOOKS DE REACT – USO, EJEMPLOS Y ÁRBOL DE DECISIONES

ÁRBOL RÁPIDO DE DECISIONES

1. ¿El valor cambia y afecta la UI?

- Simple/local → useState
- Complejo/muchas acciones → useReducer
- Estado externo/store → useSyncExternalStore

2. ¿Necesito hacer algo después del render?

- Sí → useEffect
- Callback estable para efectos → useEffectEvent
- Medir layout antes de pintar → useLayoutEffect
- CSS-in-JS → useInsertionEffect

3. ¿Valor que persiste sin causar renders?

- Sí → useRef

4. ¿Cálculo pesado derivado?

- Sí → useMemo

5. ¿Callback estable para hijos?

- Sí → useCallback

6. ¿Compartir datos sin props?

- Sí → useContext

7. ¿API imperativa a un hijo?

- Sí → useImperativeHandle

8. ¿UI optimista?

- Sí → useOptimistic

9. ¿Diferir valor para que la UI no se trabe?

- Sí → useDeferredValue

10. ¿Debug de custom hooks?

- Sí → useDebugValue

11. ¿Necesito ID único para accesibilidad?

- Sí → useId

12. ¿Modelo de acciones en formularios/server actions?

- Sí → useActionState

HOOKS DETALLADOS

useState

- Propósito: manejar estado local simple.
- Cuándo usar: toggles, inputs, flags, contadores.
- No usar: cuando hay muchas transiciones → useReducer.

useReducer

- Propósito: manejar estado complejo con lógica predecible.
- Útil para: formularios, listas con filtros, paginación, loading/error.
- Ventaja: lógica centralizada y escalable.

useEffect

- Propósito: efectos secundarios (fetch, timers, suscripciones).
- Cuándo usar: conectar React con el mundo externo.
- No usar: para cálculos derivados → usar useMemo.

useEffectEvent

- Propósito: callbacks estables usados dentro de efectos.
- Útil cuando: dependencias del efecto causan bucles o recreaciones innecesarias.

useLayoutEffect

- Propósito: medir layout antes del pintado.
- Usar solo cuando: el usuario vería un “flash” con useEffect.

useInsertionEffect

- Propósito: manejar inserción de estilos CSS antes del render.

- Solo para librerías CSS-in-JS.

useMemo

- Propósito: memorizar cálculos costosos.
- Útil para: filtros, sorting, cálculos pesados.
- No usar: para todo, solo cuando hay problemas de rendimiento.

useCallback

- Propósito: memorizar funciones.
- Útil cuando: se pasan callbacks a hijos memoizados.

useRef

- Propósito: guardar valores que no causan re-render.
- Útil para: timers, DOM refs, valores previos.

useContext

- Propósito: compartir estado global sin “prop drilling”.
- Útil para: settings, auth, idioma.

useDeferredValue

- Propósito: suavizar valores para inputs que disparan cálculos pesados.
- Útil para: búsqueda en listas grandes.

useOptimistic

- Propósito: UI optimista (mostrar cambios antes del server).

useSyncExternalStore

- Propósito: conectarse a stores externas (Redux/Zustand nativo).
- Útil para: estado global sólido en concurrent rendering.

useImperativeHandle

- Propósito: exponer métodos imperativos desde un hijo.
- Útil cuando: se usan componentes controlados con forwardRef.

useId

- Propósito: IDs únicos estables.

- Útil para: accesibilidad, labels, aria.

useActionState

- Propósito: modelar acciones y resultados (muy usado en Server Actions).

useDebugValue

- Propósito: debug de custom hooks en React DevTools.

REGLAS PARA PENSAR COMO REACT

1. Primero piensa en datos → UI
2. Decide “estado o derivado”
3. Evita useEffect para lo que puedes calcular directamente
4. Usa optimizaciones solo cuando la UI sufra realmente
5. Mantén el estado en el lugar correcto (prop drilling mínimo)
