Optimización con useMemo, useCallback y memo

Introducción

React renderiza componentes frecuentemente. Esto puede llevar a renderizados innecesarios, afectando el rendimiento.

¿Qué es `useMemo`?

`useMemo` memoriza un valor calculado.
Evita ejecutar cálculos costosos en cada render.

```
const resultado = useMemo(() => calcular(a, b), [a, b]);
```

Cuándo usar `useMemo`

- Cálculos costosos (filtrado, ordenamiento).
- Derivar valores de props o estado.
- Mejorar rendimiento en listas grandes.

Cuándo NO usar `useMemo`

- Cálculos simples.
- Dependencias que no cambian.
- Si no mejora el rendimiento (profiling primero).

¿Qué es `useCallback`?

`useCallback` memoriza funciones.

```
const manejarClick = useCallback(() => {
  hacerAlgo();
}, []);
```

Evita recrear funciones en cada render.

Cuándo usar `useCallback`

- Pasar callbacks a componentes hijos memorizados.
- En dependencias de otros hooks.
- Eventos que disparan renders innecesarios.

Cuándo NO usar `useCallback`

- Funciones pequeñas/no compartidas.
- Si no hay memoización en hijos.
- Uso excesivo puede ensuciar el código.

¿Qué es `React.memo`?

`React.memo` memoriza un componente.

```
const MiComponente = React.memo(({ nombre }) => (
  <div>{nombre}</div>
));
```

Evita renders si las props no cambian.

Cuándo usar `React.memo`

- Componentes funcionales puros.
- Props simples y estables.
- Evitar re-renders en listas.

Cuándo NO usar `React.memo`

- Props cambian frecuentemente.
- Componentes con efectos secundarios.
- Props complejas (objetos, funciones sin memoización).

Errores comunes

Hook	Error común
`useMemo`	Usarlo para cosas triviales
`useCallback`	No evita renders por sí solo
`React.memo`	No sirve si props cambian cada vez

Recomendaciones

✅ Mide antes de optimizar (profiling).
✅ Usa `useMemo` para cálculos pesados.
✅ Usa `useCallback` para funciones en props.
✅ Usa `React.memo` para componentes puros.

Evita la sobre-optimización innecesaria.