EcomLabs

GUÍA PASO A PASO

useEffect para datos derivados

Cinco ejemplos de antes y después para quitar los useEffect que solo copian datos: total del carrito, filtros, nombre completo, ordenamiento y conteos. Cada uno medido en Chromium con React 19, y cuándo sí usar useMemo.

PARA	React 18 o 19, con o sin Next.js
PROBADO CON	React 19.0 · Chromium 151 · Playwright 1.62 · Windows 11
ACTUALIZADA	30 de octubre de 2026
DOCUMENTACIÓN	react.dev/learn/you-might-not-need-an-effect
ARTÍCULO	ecomlabs.dev/g/effect
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	La regla y cómo la medimos
Paso 01	Encuentra los efectos que solo copian datos
Paso 02	Total del carrito
Paso 03	Filtros
Paso 04	Nombre completo
Paso 05	Ordenamiento
Paso 06	Conteos
Paso 07	Agrega useMemo solo si lo mediste
Al terminar	Checklist

ANTES DE EMPEZAR

La regla y cómo la medimos

Si un valor se puede calcular a partir de props o estado, no lo guardes en otro estado ni lo actualices con `useEffect`: calcúlalo durante el render. Con el efecto, React renderiza con el valor viejo, aplica ese resultado al DOM, corre el efecto y vuelve a renderizar.

Los cinco ejemplos de esta guía son componentes reales. Los empaquetamos con esbuild y React 19.0, los corrimos en Chromium 151 con Playwright y contamos renders con un clic o una tecla de verdad, sin StrictMode. En los cinco, la versión con efecto hizo 2 renders al montar y 2 al actualizar; la calculada en el render, 1 y 1. El texto final fue el mismo: la diferencia está en el render de más y en el commit intermedio, que muestra datos desfasados.

PASO 01

Encuentra los efectos que solo copian datos

Busca cada `useEffect` y mira la línea siguiente. Lo corrimos sobre el archivo de los ejemplos; en tu proyecto, usa `grep -rn -A1 "useEffect(() => {" src`:

```
$ grep -n -A1 "useEffect(() => {" examples.jsx
6:  useEffect(() => {
7-    setTotal(items.reduce((sum, item) => sum + item.price * item.qty, 0));
--
21:  useEffect(() => {
22-    setVisible(products.filter((p) => p.category === category && p.stock > 0));
```

Si el efecto solo llama a un `set` con algo calculado de props o estado, es candidato. Si habla con la red, el DOM o un widget externo, déjalo.

PASO 02

Total del carrito

```
// 1. Cart total — before
export function CartTotalBefore({ items }) {
  const [total, setTotal] = useState(0);
  useEffect(() => {
    setTotal(items.reduce((sum, item) => sum + item.price * item.qty, 0));
  }, [items]);
  return <p>{items.length} items · total {(total / 100).toFixed(2)}</p>;
}

// 1. Cart total — after
export function CartTotal({ items }) {
  const total = items.reduce((sum, item) => sum + item.price * item.qty, 0);
```

```
  return <p>{items.length} items · total {(total / 100).toFixed(2)}</p>;
}
```

Con el efecto, al agregar un producto el primer commit mostró «3 items · total 12.50» cuando el total ya era 15.00.

PASO 03

Filtros

```
// 2. Filters — before
export function ProductListBefore({ products, category }) {
  const [visible, setVisible] = useState([]);
  useEffect(() => {
    setVisible(products.filter((p) => p.category === category && p.stock > 0));
  }, [products, category]);
  return <ul>{visible.map((p) => <li key={p.id}>{p.name}</li>)}</ul>;
}

// 2. Filters — after
export function ProductList({ products, category }) {
  const visible = products.filter((p) => p.category === category && p.stock > 0);
  return <ul>{visible.map((p) => <li key={p.id}>{p.name}</li>)}</ul>;
}
```

Con el efecto, la lista empieza vacía y el primer commit al montar no muestra productos.

PASO 04

Nombre completo

```
// 3. Full name — before
  const [firstName, setFirstName] = useState("Ana");
  const [lastName, setLastName] = useState("López");
  const [fullName, setFullName] = useState("");
  useEffect(() => {
    setFullName(firstName + " " + lastName);
  }, [firstName, lastName]);
// 3. Full name — after
  const [firstName, setFirstName] = useState("Ana");
  const [lastName, setLastName] = useState("López");
  const fullName = firstName + " " + lastName;
```

Es el ejemplo de la guía de React. El JSX con los dos campos es igual en las dos versiones. Con el efecto, al escribir «Beatriz» el primer commit todavía decía «Ana López».

PASO 05

Ordenamiento

```
// 4. Sorting — before
export function SortedProductsBefore({ products, sortBy }) {
  const [sorted, setSorted] = useState([]);
  useEffect(() => {
    setSorted(products.toSorted((a, b) => a[sortBy] - b[sortBy]));
  }, [products, sortBy]);
  return <ol>{sorted.map((p) => <li key={p.id}>{p.name}</li>)}</ol>;
}

// 4. Sorting — after
export function SortedProducts({ products, sortBy }) {
  const sorted = products.toSorted((a, b) => a[sortBy] - b[sortBy]);
  return <ol>{sorted.map((p) => <li key={p.id}>{p.name}</li>)}</ol>;
}
```

CUIDADO

No ordenes con `sort()` los arreglos que llegan por props o estado: lo modifica en su lugar. `toSorted()` devuelve un arreglo nuevo.

PASO 06

Conteos

```
// 5. Counts — before
export function OrderCountsBefore({ orders }) {
  const [pending, setPending] = useState(0);
  const [shipped, setShipped] = useState(0);
  useEffect(() => {
    setPending(orders.filter((o) => o.status === "pending").length);
    setShipped(orders.filter((o) => o.status === "shipped").length);
  }, [orders]);
  return <p>{pending} pending · {shipped} shipped</p>;
}

// 5. Counts — after
export function OrderCounts({ orders }) {
  const pending = orders.filter((o) => o.status === "pending").length;
  const shipped = orders.filter((o) => o.status === "shipped").length;
  return <p>{pending} pending · {shipped} shipped</p>;
}
```

Los dos `set` del efecto se agrupan en un solo render extra, pero siguen siendo dos copias que mantener.

PASO 07

Agrega useMemo solo si lo mediste

Mide el cálculo con `console.time` en un build de producción. Si suma 1 ms o más y sus dependencias casi no cambian, guárdalo con `useMemo`:

```
const sorted = useMemo(() => products.toSorted((a, b) => a.price - b.price), [products]);
```

En nuestra prueba, ordenar 20,000 productos tardó entre 3 y 5 ms por vez. Sin `useMemo`, cada tecla del buscador lo volvía a ordenar; con `useMemo`, cero veces. `useMemo` no acelera el primer render.

> **CUIDADO**
> En desarrollo, StrictMode llama dos veces a cada componente (y a la función de `useMemo`). No midas rendimiento ahí: usa producción.

AL TERMINAR

Checklist

- [ ] Ningún estado guarda algo que se pueda calcular de props u otro estado.
- [ ] No quedan efectos cuyo único trabajo sea llamar a un `set`.
- [ ] Los ordenamientos usan `toSorted()` o una copia.
- [ ] `useMemo` solo envuelve cálculos medidos en 1 ms o más.
- [ ] Lo que causa el usuario va en el manejador del evento.
- [ ] Para reiniciar un componente cuando cambia una prop, usas `key`.
- [ ] Comparaste el texto que se ve antes y después del cambio.