EcomLabs

GUÍA PASO A PASO

Revisa tus fronteras de 'use client'

Una revisión para encontrar los archivos con 'use client' que mandan JavaScript de más al navegador, y un refactor completo de una ficha de producto: de la directiva en la página a la directiva en el botón. Con los comandos para medir el antes y el después en Next.js 15.

PARA	Proyectos Next.js con App Router
PROBADO CON	Next.js 15.5.23 · React 19.0.0 · Node 22 · Windows 11
ACTUALIZADA	30 de octubre de 2026
DOCUMENTACIÓN	nextjs.org/docs/app/getting-started/server-and-client-components
ARTÍCULO	ecomlabs.dev/g/cliente
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a revisar
Paso 01	Mide el punto de partida
Paso 02	Lista los archivos con la directiva
Paso 03	Decide cuáles sobran
Paso 04	Baja la frontera: ejemplo de refactor
Paso 05	Revisa lo que importa cada hoja
Paso 06	Mide otra vez y compara
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a revisar

`"use client"` marca una frontera en el árbol de módulos: el archivo que la lleva y todo lo que importa se convierte en código de cliente y viaja al navegador. Una directiva en el lugar equivocado, por ejemplo en una página o un layout, arrastra datos, librerías y componentes que solo pintan HTML.

Esta guía te lleva por una revisión completa: medir, listar los archivos marcados, decidir cuáles sobran, bajar la frontera con un ejemplo real y volver a medir. Los comandos son de Git Bash, macOS o Linux, y los resultados son de una demo local con Next.js 15.5.23 y webpack.

PASO 01

Mide el punto de partida

Construye el proyecto y guarda la tabla de rutas. En Next.js 15, la columna **First Load JS** es el JavaScript, con gzip, que descarga cada ruta en la primera visita:

```
npm run build
```

En la demo, la ficha de producto con la directiva en la página dio esto:

```
Route (app)                              Size  First Load JS
├ ○ /arriba                           1.19 kB         117 kB
+ First Load JS shared by all            103 kB
```

Anota las rutas que más venden (ficha, colección, carrito) y su First Load JS. Lo que está por encima de «shared by all» es lo que puedes recortar con esta revisión.

> **CUIDADO**
> Next.js 16 ya no imprime las columnas Size y First Load JS en `next build`. Si estás en la 16.1 o posterior, mide con `npx next analyze` o con Lighthouse.

PASO 02

Lista los archivos con la directiva

Este comando busca `"use client"` o `'use client'` al principio de una línea en las carpetas `app` y `components`. Ajusta las carpetas a tu proyecto:

```
grep -rlE "^[\"']use client[\"']" app components
```

Revisa primero los resultados dentro de `app/`: una `page` o un `layout` en la lista casi siempre es una frontera demasiado alta. En la demo apareció `app/arriba/page.jsx`.

PASO 03

Decide cuáles sobran

Para cada archivo de la lista, contesta estas preguntas en orden. La primera respuesta «sí» justifica la directiva:

PREGUNTA	SI LA RESPUESTA ES SÍ
¿Usa `useState`, `useReducer` o manejadores como `onClick`?	Necesita la directiva
¿Usa `useEffect` o un hook propio con estado?	Necesita la directiva
¿Usa `window`, `localStorage` u otra API del navegador?	Necesita la directiva
¿Es una página o un layout?	Saca lo interactivo a su propio archivo y quita la directiva
¿Solo recibe datos y pinta HTML?	Quita la directiva

Si un componente de terceros usa hooks pero no trae la directiva, no marques la página: crea un archivo tuyo con `'use client'` que lo reexporte y úsalo desde el servidor.

PASO 04

Baja la frontera: ejemplo de refactor

La ficha de la demo tenía `"use client"` en `page.jsx` porque el selector de talla usa estado. El refactor tiene tres partes. Primero, lo interactivo vive en su propio archivo, `components/AgregarAlCarrito.jsx`:

```
"use client";

import { useState } from "react";

export default function AgregarAlCarrito({ tallas }) {
  const [talla, setTalla] = useState(tallas[0]);
  const [agregado, setAgregado] = useState(false);
  return (
    <div>
      <label>
        Talla
        <select value={talla} onChange={(e) => setTalla(e.target.value)}>
          {tallas.map((t) => (
            <option key={t}>{t}</option>
          ))}
        </select>
      </label>
      <button type="button" onClick={() => setAgregado(true)}>
        Agregar al carrito
      </button>
      {agregado && <p role="status">Talla {talla} agregada</p>}
    </div>
```

```
  );
}
```

Segundo, lo que envuelve contenido de servidor recibe ese contenido como `children`, así no lo importa, `components/Desplegable.jsx`:

```
"use client";

import { useState } from "react";

export default function Desplegable({ titulo, children }) {
  const [abierto, setAbierto] = useState(true);
  return (
    <section>
      <button type="button" aria-expanded={abierto} onClick={() => setAbierto(!abierto)}>
        {titulo}
      </button>
      {abierto && children}
    </section>
  );
}
```

Tercero, `app/arriba/page.jsx` pierde su primera línea y vuelve a ser un Server Component. El resto del archivo no cambia: lee los datos en el servidor, pasa las tallas a `AgregarAlCarrito` y mete `<Resenas>` dentro de `<Desplegable>`:

```
-"use client";
-
 import { obtenerProducto } from "@/lib/producto";
```

`Resenas` no lleva directiva y entra en `Desplegable` como children: se renderiza en el servidor aunque se vea dentro de un componente de cliente.

PASO 05

Revisa lo que importa cada hoja

Una hoja bien puesta puede volver a arrastrar todo con un solo import. En la demo, una versión del botón importó `formatoPrecio` desde `lib/producto.js`, el mismo módulo que valida los datos con zod:

```
"use client";

import { useState } from "react";
import { formatoPrecio } from "@/lib/producto";
```

Resultado: 116 kB, casi lo mismo que con la directiva en la página, porque zod y los datos volvieron al navegador. Para saber si una librería llegó al cliente, busca un texto que solo ella contenga en la salida del build:

```
grep -rl "ZodError" .next/static/chunks
```

Si aparece, sigue los imports de tus archivos con la directiva hasta encontrar el módulo que la trae, y mueve el helper que usa el cliente a un archivo aparte, sin dependencias de servidor.

No importes un componente de servidor desde un archivo con `'use client'`: al importarlo, se vuelve de cliente. Pásalo como `children` o como prop desde un componente de servidor.

PASO 06

Mide otra vez y compara

Vuelve a correr `npm run build` y compara con lo que anotaste en el paso 1. Estos fueron los resultados de la demo, una medición de laboratorio con Next.js 15.5.23:

VERSIÓN	DÓNDE ESTÁ LA DIRECTIVA	FIRST LOAD JS
Antes	`page.jsx`	117 kB
Hoja con un import pesado	`AgregarConPrecio.jsx`	116 kB
Después	`AgregarAlCarrito.jsx` y `Desplegable.jsx`	103 kB

Prueba también en el navegador que todo sigue funcionando: el botón agrega, el desplegable abre y cierra, y no hay errores en la consola.

Si necesitas el mapa completo

En Next.js 15, la herramienta oficial es `@next/bundle-analyzer`. Instálalo con `npm i @next/bundle-analyzer`, envuelve tu `next.config.js` con `withBundleAnalyzer` y la opción `enabled: process.env.ANALYZE === 'true'`, como indica la guía de package bundling de Next.js 15, y genera el reporte con `ANALYZE=true npm run build` (en PowerShell, `$env:ANALYZE="true"; npm run build`).

AL TERMINAR

Checklist

- [ ] Ninguna página ni layout lleva `'use client'` si no lo necesita entera.
- [ ] Cada archivo con la directiva usa estado, eventos, efectos o APIs del navegador.
- [ ] Los componentes de cliente reciben props serializables.
- [ ] El contenido de servidor entra en los componentes de cliente como `children`.
- [ ] Los helpers del cliente viven en módulos sin librerías ni datos de servidor.
- [ ] Las librerías pesadas ya no aparecen en `.next/static/chunks`.
- [ ] First Load JS bajó en las rutas que más venden y todo sigue funcionando.