EcomLabs

STEP-BY-STEP GUIDE

Review your 'use client' boundaries

A review to find the 'use client' files that ship extra JavaScript to the browser, plus a complete refactor of a product page: from the directive on the page to the directive on the button. With the commands to measure before and after on Next.js 15.

FOR	Next.js projects on the App Router
TESTED WITH	Next.js 15.5.23 · React 19.0.0 · Node 22 · Windows 11
UPDATED	October 30, 2026
DOCS	nextjs.org/docs/app/getting-started/server-and-client-components
ARTICLE	ecomlabs.dev/g/cliente-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you will review
Step 01	Measure where you start
Step 02	List the files with the directive
Step 03	Decide which ones go
Step 04	Move the boundary down: a refactor example
Step 05	Check what each leaf imports
Step 06	Measure again and compare
Final	Checklist

BEFORE YOU START

What you will review

`"use client"` marks a boundary in the module tree: the file that carries it and everything it imports become client code and ship to the browser. A directive in the wrong place, such as a page or a layout, drags in data, libraries and components that only render HTML.

This guide walks you through a full review: measure, list the marked files, decide which ones should not be, move the boundary down with a real example, and measure again. The commands are for Git Bash, macOS or Linux, and the results come from a local demo on Next.js 15.5.23 with webpack.

STEP 01

Measure where you start

Build the project and keep the route table. On Next.js 15, the **First Load JS** column is the gzipped JavaScript each route downloads on a first visit:

```
npm run build
```

In the demo, the product page with the directive on the page gave this:

```
Route (app)                              Size  First Load JS
└ ○ /top                              1.16 kB         116 kB
+ First Load JS shared by all            103 kB
```

Write down your best-selling routes (product, collection, cart) and their First Load JS. Whatever sits above "shared by all" is what this review can trim.

> **CAUTION**
> Next.js 16 no longer prints the Size and First Load JS columns in `next build`. On 16.1 or later, measure with `npx next analyze` or Lighthouse.

STEP 02

List the files with the directive

This command finds `"use client"` or `'use client'` at the start of a line in the `app` and `components` folders. Adjust the folders to your project:

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

Look at the results inside `app/` first: a `page` or a `layout` on the list is nearly always a boundary set too high. In the demo, `app/top/page.jsx` showed up.

STEP 03

Decide which ones go

For each file on the list, answer these questions in order. The first "yes" justifies the directive:

QUESTION	IF THE ANSWER IS YES
Does it use `useState`, `useReducer` or handlers like `onClick`?	It needs the directive
Does it use `useEffect` or a stateful custom hook?	It needs the directive
Does it use `window`, `localStorage` or another browser API?	It needs the directive
Is it a page or a layout?	Move the interactive part to its own file and drop the directive
Does it only take data and render HTML?	Drop the directive

If a third-party component uses hooks but ships without the directive, do not mark the page: create a file of your own with `'use client'` that re-exports it, and use that from the server.

STEP 04

Move the boundary down: a refactor example

The demo page had `"use client"` in `page.jsx` because the size picker uses state. The refactor has three parts. First, the interactive part lives in its own file, `components/AddToCart.jsx`:

```
"use client";

import { useState } from "react";

export default function AddToCart({ sizes }) {
  const [size, setSize] = useState(sizes[0]);
  const [added, setAdded] = useState(false);
  return (
    <div>
      <label>
        Size
        <select value={size} onChange={(e) => setSize(e.target.value)}>
          {sizes.map((s) => (
            <option key={s}>{s}</option>
          ))}
        </select>
      </label>
      <button type="button" onClick={() => setAdded(true)}>
        Add to cart
      </button>
      {added && <p role="status">Size {size} added</p>}
```

```
    </div>
  );
}
```

Second, anything that wraps server content takes that content as `children`, so it never imports it, `components/Collapsible.jsx`:

```
"use client";

import { useState } from "react";

export default function Collapsible({ title, children }) {
  const [open, setOpen] = useState(true);
  return (
    <section>
      <button type="button" aria-expanded={open} onClick={() => setOpen(!open)}>
        {title}
      </button>
      {open && children}
    </section>
  );
}
```

Third, `app/top/page.jsx` loses its first line and is a Server Component again. The rest of the file does not change: it reads the data on the server, passes the sizes to `AddToCart` and puts `<Reviews>` inside `<Collapsible>`:

```
-"use client";
-
 import { getProduct } from "@/lib/product";
```

`Reviews` has no directive and goes into `Collapsible` as children: it renders on the server even though it sits inside a client component.

STEP 05

Check what each leaf imports

A well-placed leaf can drag everything back with a single import. In the demo, one version of the button imported `formatPrice` from `lib/product.js`, the same module that validates the data with zod:

```
"use client";

import { useState } from "react";
import { formatPrice } from "@/lib/product";
```

Result: 116 kB, the same as with the directive on the page, because zod and the data were back in the browser. To find out whether a library reached the client, search the build output for a string only that library contains:

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

If it shows up, follow the imports of your marked files until you find the module that brings it in, and move the helper the client uses into a separate file with no server dependencies.

Do not import a Server Component from a `'use client'` file: importing it turns it into client code. Pass it as `children` or another prop from a Server Component.

STEP 06

Measure again and compare

Run `npm run build` again and compare with what you wrote down in step 1. These were the demo's results, a lab measurement on Next.js 15.5.23:

VERSION	WHERE THE DIRECTIVE IS	FIRST LOAD JS
Before	`page.jsx`	116 kB
Leaf with a heavy import	`AddToCartWithPrice.jsx`	116 kB
After	`AddToCart.jsx` and `Collapsible.jsx`	103 kB

Also check in the browser that everything still works: the button adds, the collapsible opens and closes, and the console shows no errors.

When you need the full map

On Next.js 15 the official tool is `@next/bundle-analyzer`. Install it with `npm i @next/bundle-analyzer`, wrap your `next.config.js` with `withBundleAnalyzer` and the option `enabled: process.env.ANALYZE === 'true'`, as the Next.js 15 package bundling guide shows, and generate the report with `ANALYZE=true npm run build` (in PowerShell, `$env:ANALYZE="true"; npm run build`).

FINAL

Checklist

- [ ] No page or layout has `'use client'` unless all of it needs to.
- [ ] Every file with the directive uses state, events, effects or browser APIs.
- [ ] Client Components get serializable props.
- [ ] Server content goes into Client Components as `children`.
- [ ] Client-side helpers live in modules with no server libraries or data.
- [ ] Heavy libraries no longer show up in `.next/static/chunks`.
- [ ] First Load JS went down on your best-selling routes and everything still works.