EcomLabs

STEP-BY-STEP GUIDE

Server and Client Components in Next.js: the tree

The component tree diagram, the table that tells you whether a component belongs on the server or the client, and a full refactor of a product page: from a client page with useEffect to a Server Component with interactive leaves. With a checklist.

FOR	Next.js projects on the App Router (React 19)
TESTED WITH	Next.js 15.5.23 · React 19.0.0 · Node 22.22.3 · next build, next start, curl and Playwright, locally
UPDATED	November 19, 2026
DOCS	nextjs.org/docs/app/getting-started/server-and-client-components
ARTICLE	ecomlabs.dev/g/rsc-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Step 01	The tree diagram
Step 02	The table: server or client?
Step 03	Before: the whole page on the client
Step 04	After: server with client leaves
Step 05	Check it in your project
When you finish	Checklist

STEP 01

The tree diagram

In the App Router everything is a Server Component by default (there is no directive; `'use server'` is for Server Functions). Code crosses to the client through imports and data through serializable props. This is the tree of the demo's product page; draw yours with the same shape: root and branches on the server, client only at the leaves.

```
layout.tsx                  server
+-- page.tsx (async)        server, owner of Reviews
    +-- h1, price           server
    +-- AddToCart           CLIENT (leaf)
    +-- Tabs                CLIENT (leaf)
        +-- Reviews (async) server, arrives as a prop
```

Owner and parent. `page.tsx` writes `<Reviews />` in its JSX: it is the owner. `Tabs` contains it on screen: it is the parent. Because the owner is a server component, `Reviews` renders on the server and `Tabs` only receives its output.

What reaches the browser on the first load: the **HTML**, with the page already painted and the review text; the **RSC Payload**, with the output of the Server Components, references to the Client Components and their props; and the **JavaScript** for `AddToCart` and `Tabs` (965 bytes uncompressed in the demo). What never travels: the code of `Reviews`, the function that reads the data, and any field you did not pass as a prop.

STEP 02

The table: server or client?

IF THE COMPONENT…	IT GOES ON THE
Reads the database, files or a secret	Server
Uses a heavy library just to render (Markdown, dates)	Server
Only displays data: title, price, description, reviews	Server
Opens and closes with `<details>` or submits a `<form>` to a Server Function	Server
Uses `useState`, `onClick` or `onChange`	Client
Uses `useEffect` (check first that you really need it)	Client
Reads `window`, `localStorage` or `navigator`	Client
Uses a custom hook that does any of the above	Client
Provides a context (provider)	Client, wrapping only `{children}`

If no client row applies, keep it on the server. If one does, move only that part into its own file with `"use client"`.

STEP 03

Before: the whole page on the client

A product page that fetches the product from an API route on mount. It compiles and works, but all of it is client code:

```
"use client";

import { useEffect, useState } from "react";
import { useParams } from "next/navigation";

type Product = { id: string; name: string; price: string; sizes: string[] };

export default function Page() {
  const { id } = useParams<{ id: string }>();
  const [product, setProduct] = useState<Product | null>(null);
  const [size, setSize] = useState("");

  useEffect(() => {
    fetch(`/api/products/${id}`)
      .then((r) => r.json())
      .then((p: Product) => {
        setProduct(p);
        setSize(p.sizes[0]);
      });
  }, [id]);

  if (!product) return <p>Cargando…</p>;

  return (
    <main>
      <h1>{product.name}</h1>
      <p>{product.price}</p>
      {product.sizes.map((s) => (
        <button key={s} aria-pressed={s === size} onClick={() => setSize(s)}>
          {s}
        </button>
      ))}
    </main>
  );
}
```

What we measured in the demo with `next start` and `curl`:

- The initial HTML only holds `<p>Cargando…</p>` (loading). The product name shows up after hydration and the request; a hero image would arrive just as late and hurt LCP.
- It needs a public API route just to feed this page.
- The whole component is browser JavaScript, even though the only interactive part is the size buttons.

STEP 04

After: server with client leaves

The page reads the product on the server and passes each leaf only what it uses. `lib/products.ts` starts with `import "server-only"`.

app/p/[id]/page.tsx

```
import { notFound } from "next/navigation";
import { getProduct } from "@/lib/products";
import AddToCart from "./add-to-cart";
import Tabs from "./tabs";
import Reviews from "./reviews";

export default async function Page({ params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const product = await getProduct(id);
  if (!product) notFound();

  const price = new Intl.NumberFormat("es-MX", { style: "currency", currency: "MXN" })
    .format(product.priceCents / 100);

  return (
    <main>
      <h1>{product.name}</h1>
      <p>{price}</p>
      <AddToCart productId={product.id} sizes={product.sizes} />
      <Tabs
        panels={[
          { label: "Reseñas", content: <Reviews productId={product.id} /> },
          { label: "Envío", content: <p>Envío gratis desde $999.</p> },
        ]}
      />
    </main>
  );
}
```

`tabs.tsx` is a client component that takes the panels as a prop and never imports `Reviews`: the full code is in the article. `add-to-cart.tsx` carries `"use client"` and the size buttons from the previous step, now with the sizes as a prop. `reviews.tsx` is an `async` Server Component that reads the reviews and returns a list. Result: the HTML has the name, the price and the reviews; the API route is gone; the page's JavaScript is only `AddToCart` and `Tabs`.

> **CAUTION**
>
> Do not pass the whole product to a leaf. In the demo, `<AddToCart product={product} />` published an internal field (`costCents`) in the payload and the HTML. TypeScript does not catch it. Do not pass plain functions as props either: the build fails at prerender.

STEP 05

Check it in your project

With `npm run build` and `npm start` running, swap in your own route and field:

```
curl -s http://localhost:3000/p/1 | grep -c costCents
curl -s -H "RSC: 1" http://localhost:3000/p/1 | grep -c costCents
grep -rl "getReviews" .next/static
```

The first two lines count how many times the internal field appears in the HTML and in the payload: it should be `0`. The third looks for the name of a server function in the build's public files: it should list nothing. The payload format is internal to React; use it to inspect, not to depend on.

WHEN YOU FINISH

Checklist

- [ ] Pages and layouts have no `'use client'`.
- [ ] Every client component matches a client row in the table.
- [ ] Interactive parts live in small leaves, each in its own file.
- [ ] Server content enters the client as `children` or another prop, never as an import.
- [ ] Leaves get only the fields they use.
- [ ] No plain function is passed as a prop to a client component.
- [ ] Files that read data or secrets start with `import "server-only"`.
- [ ] First-load data is read by a Server Component, not a `useEffect`.
- [ ] One page's HTML and payload contain no internal fields.