Engineer

Next.js

N a React framework for building web apps

What it is

Next.js is a framework that wraps React, the library for building user interfaces out of components, and supplies the parts React leaves to each application: routing, rendering on the server, data fetching and a build that is configured for you. The package and its command are both called `next`, as in `next dev` and `next build`.

It was first released in 2016, in its authors' words "to provide an easy way to server-render React applications". The early design rested on a few principles: zero setup, the filesystem as an API, so that a file in a `pages/` folder became a route, automatic server rendering and code splitting, and data fetching left to the developer. By 2023 the release notes credited over 2,600 developers alongside a core team at Vercel.

Next.js 13, on October 25, 2022, added an `app` directory in beta, built on React Server Components, with nested layouts and streaming. Next.js 13.4, on May 4, 2023, declared this App Router stable, and the documentation has described two routers ever since. The Pages Router is "the original router, still supported and being improved". They differ even in which React they run: the App Router ships React canary releases built in, while the Pages Router uses whatever version the project's `package.json` names. In the App Router, pages and layouts render on the server unless a file opts into the browser with a `"use client"` directive.

Next.js 16 arrived on October 21, 2025. It made Turbopack the default bundler, renamed the `middleware.ts` file to `proxy.ts`, and introduced Cache Components, a model in which a `"use cache"` directive marks what may be cached and everything else is rendered at request time. Next.js 16.4, on October

6, 2026, recommends Cache Components for every app, switches them on for new projects made with `create-next-app`, and says they "will become the default in Next.js 17".

Use the concept in a decision

Put "What is Next.js?" to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The decision Next.js answers is whether to let a framework own the routing, rendering and server of a React application, or to assemble those pieces yourself and keep the choices.

What it offers is convention in place of configuration. Routes come from files, bundlers and compilers are set up automatically, and each route can be built ahead of time, rendered on the server for each request or rendered in the browser. Server Components keep data access and secrets on the server and send less JavaScript to the browser. For an application that mixes public pages with a signed-in area and its own backend, that per-route choice is the main attraction.

The first cost is a server, running Node.js 20.9 or later for Next.js 16. Setting `output: 'export'` writes one HTML file per route for any static web server, and the guide pitches it as a way to start static and add server features later. Its list of what needs a Node.js server, though, covers much of what sets Next.js apart: Proxy, Server Actions, cookies, rewrites, redirects, headers, Incremental Static Regeneration and image optimisation with the default loader.

The second cost is pace. Four feature releases followed 16.0 within a year, and the caching model is about to change its default again. The 16.4 notes say Cache Components replace "the implicit caching behaviors of the previous App Router versions", so code written for Next.js 13 to 15 caches by different rules. Tutorials found online are split between the two routers as well.

The third cost is that upgrades are security work. On December 3, 2025, the project published an advisory for CVE-2025-66478, a critical vulnerability in the React Server Components protocol, telling users to upgrade to patched versions immediately, and the blog carried three more security posts between September 30 and October 14, 2026. A team that adopts Next.js signs up to follow its releases closely. A team publishing documents that read the same for every visitor pays for a server model it hardly uses.

What it looks like in practice

In the founder's work under NextMariner, the Next.js 16 frontend renders each surface its own way, which is the per-route choice put to work. Marketing and public pages are statically generated, the genuinely interactive parts stay client-rendered, and on the authenticated dashboard and the big directory listings a factory, `createPrefetchedServerPage`, fetches the page's data on the server and hands it to the client already populated. Pages move onto the factory one at a time, and the page is candid that the rest of the authenticated app is still client-rendered and queued behind it: "a migration with a working mechanism and an obvious next page, rather than a finished sweep." The App Router was designed for that kind of move; the 13.4 release notes describe adopting it incrementally.

A layered test suite gives a Next.js frontend a layer of its own: 543 Vitest and Playwright specs, including 48 end-to-end files and 30 browser specs, beside separate layers for the Go API and the database. Each layer is checked where its behaviour lives rather than everything through a browser, which matters in a framework whose component tree runs partly on the server and partly in the browser.

When the founder, working under NextMariner, hardened an application, the browser-facing protections went into `proxy.ts`, the renamed middleware. It sets a Content-Security-Policy with a per-request nonce and `strict-dynamic`, plus HSTS and SameSite cookies. Authorisation lives elsewhere: entitlements, covering tier, role, organisation and ship scoping, are re-checked on the server on every request, and the frontend's gates count as UX only. That split matches the Proxy reference, which calls the feature a last resort and tells developers to verify authorisation "inside each Server Function rather than relying on Proxy alone".

Where it goes wrong

After a sign-in redirect is added, the login page loads without its styles. Without a `matcher`, Proxy runs on every request, including built files under `_next/static`, optimised images under `_next/image` and everything in `public/`, so a redirect meant for pages also catches the CSS, JavaScript and images the login page needs. Give Proxy a matcher that skips those paths:

```
// proxy.ts
export const config = {
    matcher: ["/((?!_next/static|_next/image|favicon.ico).*)"],
};
```

Files from `public/` share no prefix, so add their paths as well.

A form action works for a signed-out user on a route that looks protected. Server Functions have no routes of their own: they arrive as POST requests to the page where they are used, so a matcher that skips a path skips the functions there as well. The Proxy reference warns that changing the matcher, or moving a function to another route during a refactor, "can silently remove Proxy coverage". Check the session and the user's permissions inside every Server Function, and treat Proxy as an outer layer, never the only one.

A user saves a change and the page still shows the old value. In Next.js 16, `revalidateTag(tag, 'max')` gives tagged entries stale-while-revalidate behaviour: the cached copy keeps being served while a fresh one is fetched, which suits content that can tolerate eventual consistency and surprises the person who just pressed Save. Call `updateTag(tag)` from the Server Action that made the change; it expires the entry and reads fresh data within the same request. The old one-argument `revalidateTag(tag)` is deprecated and is now a TypeScript error.

The upgrade codemod finishes and the build still fails on `params` or `cookies()`. Next.js 15 made the request APIs asynchronous with temporary synchronous compatibility, and Next.js 16 removes it. The `upgrade` codemod does not run every migration, so also run `npx @next/codemod@canary next-async-request-api .`, then `npx next typegen` to generate the `PageProps`, `LayoutProps` and `RouteContext` helpers. The same upgrade removes `next lint` and stops `next build` from linting, so add ESLint or Biome as a step of its own.

Further reading

- Next.js Docs — the official documentation, with a switch between the App Router and Pages Router versions of every page.
- Next.js 16 — the release that brought Cache Components, `proxy.ts` and Turbopack as the default bundler, with its breaking changes.
- How to upgrade to version 16 — the codemods, the version requirements and every removed or renamed API.
- proxy.js — matchers, where Proxy sits in the request order and why middleware was renamed.
- Next.js 16.4 — the case for Cache Components and the plan to make them the default in Next.js 17.
- How to create a static export — which features need a server, and what `output: 'export'` writes.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is Next.js?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/nextjs/