Engineer

Next.js

N ett React-ramverk för att bygga webbappar

Vad det är

Next.js är ett ramverk som omsluter React, biblioteket för att bygga användargränssnitt av komponenter, och som står för de delar React lämnar åt varje applikation: routning, rendering på servern, datahämtning och ett bygge som redan är konfigurerat åt dig. Paketet och dess kommando heter båda `next`, som i `next dev` och `next build`.

Den första versionen släpptes 2016, enligt upphovsmännen för att ge ett enkelt sätt att serverrendera React-applikationer. Den tidiga designen vilade på några få principer: ingen konfiguration att göra, filsystemet som API, så att en fil i mappen `pages/` blev en route, automatisk serverrendering och koddelning, och datahämtning som lämnades åt utvecklaren. År 2023 tackade versionsanteckningarna över 2 600 utvecklare vid sidan av ett kärnteam på Vercel.

Next.js 13 lade den 25 oktober 2022 till en `app`-katalog i beta, byggd på React Server Components, med nästlade layouter och strömning. Med Next.js 13.4 den 4 maj 2023 förklarades denna App Router stabil, och dokumentationen har beskrivit två routrar sedan dess. Pages Router presenteras som den ursprungliga routern, som fortfarande stöds och förbättras. De två kör inte ens samma React: App Router levereras med inbyggda canary-versioner av React, medan Pages Router använder den version som projektets `package.json` anger. I App Router renderas sidor och layouter på servern, om inte en fil väljer webbläsaren med direktivet `"use client"`.

Next.js 16 kom den 21 oktober 2025. Versionen gjorde Turbopack till standardbundlare, döpte om filen `middleware.ts` till `proxy.ts` och införde Cache Components, en modell där direktivet `"use cache"` markerar vad som får cachas

och allt annat renderas vid varje anrop. Next.js 16.4, från den 6 oktober 2026, rekommenderar Cache Components för alla appar, slår på dem i nya projekt som skapas med `create-next-app` och säger att de blir standard i Next.js 17.

Använd begreppet i ett beslut

Använd ”Vad är Next.js?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Gränssnitt som laddar snabbt, fungerar i mobilen och beter sig likadant i alla webbläsare dina kunder råkar använda.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Styrde full stack-utvecklingen av GIS-kartor — kartutveckling i full stack, från databas till webbläsare

Styrde full stack-utvecklingen av GIS-kartor Hör av er →

Varför det spelar roll

Frågan som Next.js besvarar är om ett ramverk ska äga routningen, renderingen och servern i en React-applikation, eller om ni sätter ihop de delarna själva och behåller valen.

Det ramverket erbjuder är konvention i stället för konfiguration. Routes kommer från filer, bundlare och kompilatorer ställs in automatiskt, och varje route kan byggas i förväg, renderas på servern vid varje anrop eller renderas i webbläsaren. Server Components håller dataåtkomst och hemligheter på servern och skickar mindre JavaScript till webbläsaren. För en applikation som blandar publika sidor med ett inloggat område och en egen backend är just det valet, route för route, den stora lockelsen.

Den första kostnaden är en server, som för Next.js 16 måste köra Node.js 20.9 eller senare. Med `output: 'export'` skrivs en HTML-fil per route för valfri statisk webbserver, och guiden presenterar det som ett sätt att börja statiskt och lägga till serverfunktioner senare. Guidens lista över det som kräver en Node.js-server täcker ändå mycket av det som skiljer Next.js från andra: Proxy, Server Actions, cookies, rewrites, omdirigeringar, headers, Incremental Static Regeneration och bildoptimering med standardladdaren.

Den andra kostnaden är tempot. Fyra funktionsutgåvor följde på 16.0 inom ett år, och cachningsmodellen står i begrepp att byta standard igen. Enligt anteckningarna till 16.4 ersätter Cache Components det implicita

cachningsbeteendet i tidigare versioner av App Router, så kod som skrevs för Next.js 13–15 cachar efter andra regler. Guiderna på nätet är dessutom delade mellan de två routrarna.

Den tredje kostnaden är att uppgraderingar är säkerhetsarbete. Den 3 december 2025 publicerade projektet en säkerhetsvarning om CVE-2025-66478, en kritisk sårbarhet i protokollet för React Server Components, och uppmanade användarna att genast uppgradera till rättade versioner, och bloggen publicerade ytterligare tre säkerhetsinlägg mellan den 30 september och den 14 oktober 2026. Ett team som väljer Next.js förbinder sig att följa utgåvorna noga. Ett team som publicerar dokument som ser likadana ut för varje besökare betalar för en servermodell som det knappt använder.

Så ser det ut i praktiken

I grundarens arbete hos NextMariner renderar frontenden i Next.js 16 varje yta på sitt eget sätt, och det är valet route för route omsatt i praktiken. Marknadsföringssidor och publika sidor genereras statiskt, de genuint interaktiva delarna förblir klientrenderade, och på den inloggade dashboarden och de stora katalogsidorna hämtar en fabrik, `createPrefetchedServerPage`, sidans data på servern och lämnar över dem så att klienten redan har sina data. Sidorna flyttas över till fabriken en i taget, och sidan om arbetet säger öppet att resten av den inloggade appen fortfarande är klientrenderad och väntar på sin tur efter den: ”en migrering med en fungerande mekanism och en självklar nästa sida, snarare än en avslutad genomgång”. App Router är utformad för just den sortens flytt, och versionsanteckningarna till 13.4 beskriver hur den kan införas stegvis.

En skiktad testsvit ger en frontend i Next.js ett eget lager: 543 testspecifikationer i Vitest och Playwright, däribland 48 end-to-end-filer och 30 webbläsarspecifikationer, vid sidan av separata lager för Go-API:t och databasen. Varje lager verifieras där dess beteende finns i stället för att allt verifieras utifrån genom en webbläsare, och det väger extra tungt i ett ramverk där samma komponentträd körs dels på servern, dels i webbläsaren.

När grundaren i sitt arbete hos NextMariner härdade en applikation hamnade skydden som riktar sig mot webbläsaren i `proxy.ts`, den omdöpta middlewaren. Den sätter en Content-Security-Policy med en nonce per anrop och `strict-dynamic`, plus HSTS och SameSite-cookies. Behörigheten avgörs på ett annat ställe. Varje anrop prövas för sig på servern mot nivå, roll, organisation och avgränsning per fartyg, och det som gränssnittet spärrar räknas enbart som UX.

Sidan sammanfattar uppdelningen i en mening: ”Spärrarna avgör vad du ser; servern avgör vad du får göra.” Uppdelningen stämmer med referensen för Proxy, som kallar funktionen en sista utväg och uppmanar utvecklare att kontrollera behörighet inuti varje Server Function i stället för att enbart förlita sig på Proxy.

Var det går fel

När en omdirigering till inloggningen läggs till laddas inloggningssidan utan sina stilar. Proxy körs på varje anrop om du inte anger någon `matcher`, även för byggda filer under `_next/static`, optimerade bilder under `_next/image` och allt i `public/`, så en omdirigering som var tänkt för sidor fångar också den CSS, det JavaScript och de bilder som inloggningssidan behöver. Ge Proxy en matcher som hoppar över de sökvägarna:

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

Filerna i `public/` delar inget gemensamt prefix, så lägg till deras sökvägar också.

En formulärhandling fungerar för en utloggad användare på en route som ser skyddad ut. Server Functions delar route med sidan där de används: de kommer som POST-anrop till den sidan, så en matcher som hoppar över en sökväg hoppar också över funktionerna där. Referensen för Proxy varnar för att en ändrad matcher, eller en omstrukturering som flyttar en funktion till en annan route, kan ta bort Proxy-skyddet utan att någon märker det. Kontrollera sessionen och användarens behörigheter inuti varje Server Function, och behandla Proxy som ett yttre lager, aldrig som det enda.

En användare sparar en ändring och sidan visar fortfarande det gamla värdet. I Next.js 16 ger `revalidateTag(tag, 'max')` taggade poster ett beteende av typen stale-while-revalidate: den cachade kopian fortsätter att levereras medan en färsk hämtas, vilket passar innehåll som tål slutlig konsistens men överraskar den som just tryckte på Spara. Anropa `updateTag(tag)` från den Server Action som gjorde ändringen; den låter posten löpa ut och läser färska data inom samma anrop. Den gamla formen med ett argument, `revalidateTag(tag)`, är föråldrad och ger nu ett TypeScript-fel.

Uppgraderingens codemod blir klar och bygget fallerar ändå på `params` eller `cookies()`. Next.js 15 gjorde API:erna för anrop asynkrona med en tillfällig synkron kompatibilitet, och Next.js 16 tar bort den. Codemoden `upgrade` gör inte alla nödvändiga ändringar, så kör också `npx @next/codemod@canary next-async-request-api .` och därefter `npx next typegen` för att generera hjälptyperna `PageProps`, `LayoutProps` och `RouteContext`. Samma uppgradering tar bort `next lint` och gör att `next build` slutar linta, så lägg till ESLint eller Biome som ett eget steg.

Läs mer

- Next.js Docs — den officiella dokumentationen, med ett val mellan App Router- och Pages Router-versionen av varje sida.
- Next.js 16 — utgåvan som förde in Cache Components, `proxy.ts` och Turbopack som standardbundlare, med sina bakåtinkompatibla ändringar.
- How to upgrade to version 16 — codemods, versionskrav och varje borttaget eller omdöpt API.
- proxy.js — matchers, var Proxy ligger i ordningen för ett anrop och varför middleware döptes om.
- Next.js 16.4 — argumenten för Cache Components och planen att göra dem till standard i Next.js 17.
- How to create a static export — vilka funktioner som kräver en server och vad `output: 'export'` skriver.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Next.js?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/nextjs/