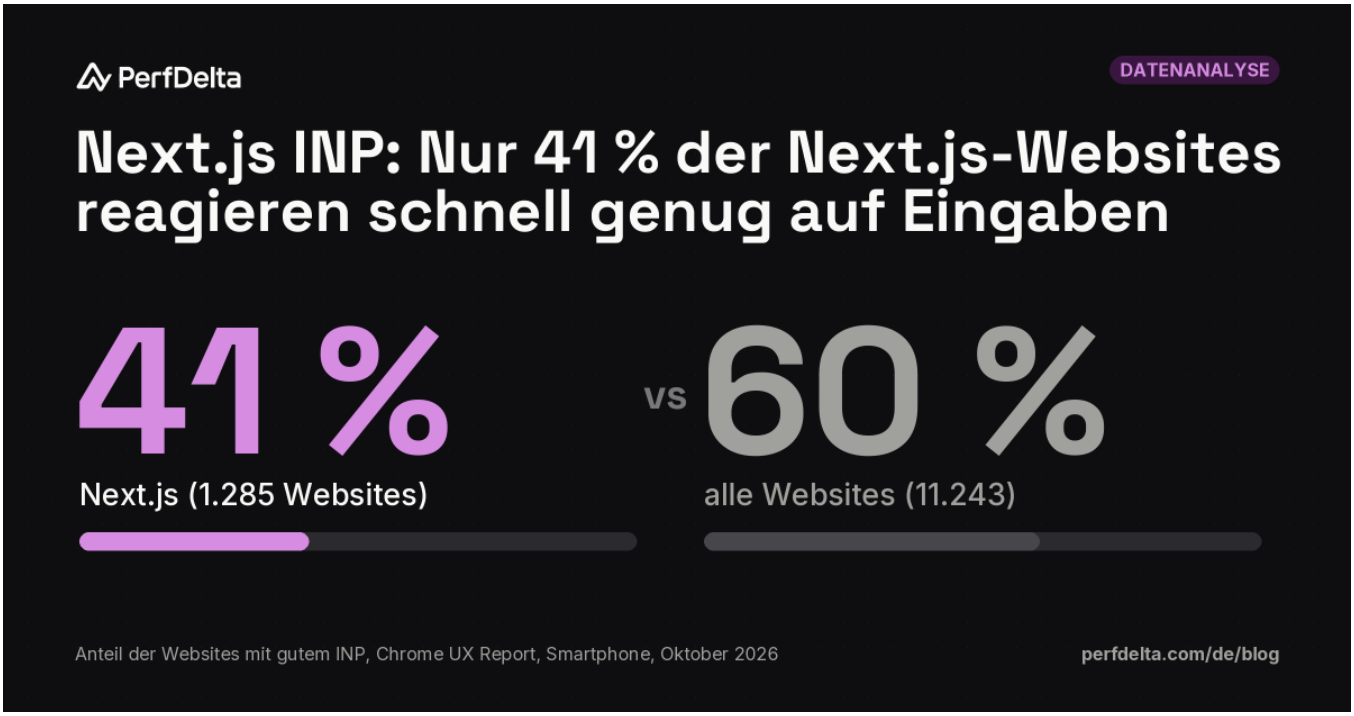
Next.js INP: Nur vier von zehn Next.js-Websites haben einen guten Interaction to Next Paint

Veröffentlicht am 06.11.2026 von Christian Schreiber



Next.js gilt als Framework für schnelle Websites: Server-Rendering, Bildoptimierung und Code-Splitting sind eingebaut. In unseren Felddaten laden Next.js-Websites auch etwa so schnell wie der Durchschnitt. Zurück fallen sie bei der Reaktionszeit. Von den 1.285 Next.js-Websites mit INP-Daten hatten auf Smartphones nur 41,1 Prozent einen guten Interaction to Next Paint (INP). Über alle 11.243 Websites unserer Stichprobe mit INP-Daten waren es 60,2 Prozent.

Der INP ist das Core Web Vital mit dem größten Abstand. 26,1 Prozent der Next.js-Websites bestanden alle drei Core Web Vitals, bei allen Websites waren es 36,4 Prozent. Hätte jede Website einen guten INP, lägen die Quoten bei 45,9 und 50,1 Prozent: Der Abstand schrumpfte von zehn auf vier Prozentpunkte.

Der INP misst, wie schnell eine Seite auf Tippen und Klicken reagiert

Der Interaction to Next Paint erfasst jedes Tippen, jeden Klick und jeden Tastendruck während eines Seitenaufrufs und misst die Zeit, bis der Browser das nächste Bild zeigt. Der ausgewiesene Wert liegt nahe an der langsamsten Interaktion des Seitenaufrufs. Als gut gelten 200 Millisekunden oder weniger, als schlecht mehr als 500 Millisekunden. Wie der Wert zustande kommt, erklärt unser Ratgeber zum Interaction to Next Paint.

Ein langsamer INP fühlt sich an wie ein Lichtschalter mit Verzögerung: Sie drücken, nichts passiert, und Sie drücken noch einmal. Die Ursache ist meist JavaScript, das im Moment der Interaktion den Hauptthread des Browsers belegt oder als Reaktion darauf viel Arbeit erledigen muss. Auf großen Seiten kommt die Stil- und Layoutberechnung für das nächste Bild hinzu.

Next.js-Websites liegen beim INP zurück, nicht beim LCP

Die Tabelle vergleicht die Next.js-Websites mit allen Websites unserer Stichprobe. Die Werte beruhen auf dem 75. Perzentil echter Smartphone-Besuche aus dem Chrome UX Report. Anteile beziehen sich auf die Websites mit Daten zur jeweiligen Metrik:

Felddaten Smartphone	Next.js	alle Websites
guter INP (höchstens 200 ms)	41,1 %	60,2 %
guter LCP (höchstens 2,5 s)	56,9 %	59,0 %
guter CLS (höchstens 0,1)	70,5 %	76,6 %
alle drei Core Web Vitals gut	26,1 %	36,4 %
LCP und CLS gut, INP nicht	19,8 %	13,7 %
TTFB, Median der p75-Werte	875 ms	841 ms
bfcache-Quote, Median	35 %	56 %

Ladezeit (LCP) und Serverantwort (TTFB) liegen nah am Durchschnitt. Jede fünfte Next.js-Website hat gute Lade- und Layoutwerte und verfehlt die Core Web Vitals allein wegen des INP. Die Layoutstabilität ist etwas schwächer, und die bfcache-Quote liegt deutlich niedriger, dazu weiter unten mehr.

Das Muster ist keine Eigenheit von Next.js. Nuxt rendert ebenfalls auf dem Server und hydriert die Seite danach mit JavaScript und zeigt in unserer Stichprobe einen ähnlichen Wert (227 Websites mit INP-Daten, 46,7 Prozent guter INP). Das deutet darauf hin, dass die gemeinsame Architektur eine Rolle spielt. Sie ist aber nicht der einzige Faktor: Auch manche Shopsysteme ohne ein solches Framework haben in unseren Daten niedrige INP-Werte. Art der Website und Zahl der Drittanbieter-Skripte zählen also mit.

Hydration und große Client-Bundles blockieren den Hauptthread

Eine serverseitig gerenderte Next.js-Seite sieht fertig aus, sobald das HTML ankommt. Bedienbar ist sie erst, wenn React das JavaScript der Client-Komponenten geladen und mit dem HTML verbunden hat. Dieser Vorgang heißt Hydration. Ein Tippen während der Hydration muss oft warten, bis der Hauptthread frei ist. Das ist die Eingabeverzögerung, der erste der drei Teile des INP. Suspense-Grenzen erlauben React, große Seiten in kleineren Stücken zu hydrieren und den Bereich vorzuziehen, mit dem der

Besucher interagiert.

Nach der Hydration machen typischerweise drei Dinge Interaktionen langsam:

- **Große Client-Bundles.** Jede mit `"use client"` markierte Komponente und jedes Modul, das diese Datei importiert, landet im Browser und wird dort geparst und ausgeführt. Ein `"use client"` weit oben im Baum zieht deshalb viel Code ins Bundle. Server-Komponenten, die als `children` übergeben werden, bleiben auf dem Server.
- **Teure Neuberechnungen.** Eine Zustandsänderung in einem Context oder einer Komponente nahe der Wurzel kann große Teile der Seite in einer einzigen langen Aufgabe neu rendern, bevor der Browser die Reaktion zeichnen kann.
- **Skripte von Drittanbietern.** Tag-Manager, Consent-Werkzeuge, Chat-Widgets und A/B-Tests teilen sich denselben Hauptthread und konkurrieren mit React um ihn.

Sechs Maßnahmen verbessern den INP von Next.js-Websites

1. **Client-Komponenten klein halten.** Im App Router sind Komponenten Server-Komponenten, solange nichts anderes angegeben ist. Setzen Sie `"use client"` an die interaktiven Blätter des Baums, etwa an einen Button oder einen Filter, statt an ganze Bereiche. Was im Browser ankommt, zeigt der Bundle-Analyzer: `next analyze` für Turbopack-Builds seit Next.js 16.1, das Plugin `@next/bundle-analyzer` für Webpack-Builds (Optimizing package bundling, Next.js-Dokumentation).
2. **Nicht dringende Updates als Transition markieren.** Mit `useTransition` oder `useDeferredValue` rendert React eine aufwendige Aktualisierung, etwa das Filtern einer langen Liste, in unterbrechbaren Schritten auf dem Hauptthread und bearbeitet neue Eingaben zuerst (useTransition, React-Dokumentation). Das hilft nur beim Rendern durch React, nicht bei langsamem synchronem Code im Event-Handler selbst.
3. **Sofort eine Rückmeldung zeigen.** Zeigen Sie im ersten Bild nach der Interaktion einen gedrückten Zustand, einen Ladeindikator oder ein Skelett aus `loading.js` und erledigen Sie die schwere Arbeit danach.
4. **Lange Aufgaben aufteilen.** Arbeit, die länger als 50 Millisekunden läuft, in Abschnitte teilen und dazwischen an den Browser abgeben, etwa mit `scheduler.yield()`, in Chrome seit Version 129 verfügbar, in anderen Browsern mit `setTimeout` als Ersatz (Optimize long tasks, web.dev, Dezember 2024).
5. **Drittanbieter-Skripte reduzieren und zeitlich steuern.** Entfernen Sie Tags, die niemand nutzt. Mit `next/script` und der Strategie `lazyOnload` laden Skripte, die für die erste Interaktion nicht nötig sind, erst, wenn der Browser untätig ist. Ihre Arbeit läuft später trotzdem auf dem Hauptthread. Das verschiebt die Last nur und ersetzt nicht das Entfernen von Tags.

6. **Selten genutzte Komponenten erst bei Bedarf rendern.** Lagern Sie Dialoge, Karussells und Karten mit `next/dynamic` aus und rendern Sie sie erst, wenn sie geöffnet werden. Eine dynamisch importierte Komponente, die schon beim Seitenaufruf gerendert wird, lädt und hydriert trotzdem sofort.

Der Ratgeber zu Next.js Core Web Vitals behandelt zusätzlich Bilder, Schriften und Rendering-Strategien.

Dynamische Next.js-Seiten verfehlen oft den bfcache

Der zweite Befund betrifft den Zurück-Button. Bei der mittleren Next.js-Website kamen nur 35 Prozent der Zurück- und Vor-Navigationen aus dem bfcache. Bei 42,0 Prozent der 1.023 Next.js-Websites mit genügend Verlaufsnavigationen lag die Quote unter 25 Prozent, über alle Websites bei 28,8 Prozent.

Ein wahrscheinlicher Grund ist eine Voreinstellung des Frameworks. Laut Next.js-Dokumentation senden dynamisch gerenderte Seiten den Header `Cache-Control: private, no-cache, no-store, max-age=0, must-revalidate` (Self-Hosting, Next.js-Dokumentation). Laut Chrome war `no-store` im HTML der größte einzelne Grund für verpasste bfcache-Treffer (Enabling bfcache for Cache-Control: no-store, Chrome for Developers, September 2025). Chrome lässt solche Seiten inzwischen nur unter bestimmten Bedingungen in den bfcache. Seiten ohne persönliche Inhalte lassen sich oft statisch rendern oder per Revalidierung aktualisieren. Unsere Auswertung zur bfcache-Quote erklärt die übrigen Blocker.

Seitenwechsel in Single-Page-Apps verändern das Bild nur wenig

Eine Next.js-Anwendung wechselt Seiten im Browser. Für den Chrome UX Report umfasst ein Seitenaufruf deshalb oft mehrere Routenwechsel, und der INP deckt mehr Interaktionen ab als auf einer klassischen Website. Chromes experimentelle Messung von Soft Navigations trennt diese Seitenaufrufe auf. Bei der mittleren Next.js-Website veränderte sie den INP nur um vier Prozent, dieser Effekt erklärt den Abstand also nicht.

Auch Zurück-Navigationen innerhalb der Anwendung laufen im Browser und erscheinen im Chrome UX Report nicht als Zurück-Navigation. Die bfcache-Quote oben beschreibt deshalb vollständige Seitenaufrufe, etwa wenn ein Besucher von einer anderen Website zurückkommt.

Den INP Ihrer Next.js-Website prüfen

Der Website Speed Test zeigt den INP Ihrer Website aus echten Smartphone-Besuchen, neben Ihren Wettbewerbern und Woche für Woche. Bis zu fünf Websites vergleichen Sie direkt im Core Web Vitals Benchmark. Wie Next.js-Websites insgesamt abschneiden, zeigt die Next.js Pagespeed-Statistik.

Methode und Grenzen dieser Auswertung

- **Daten:** 11.396 Smartphone-Datensätze aus dem Chrome UX Report, je Origin einmal gezählt, Datenzeitraum bis 31. Oktober 2026, abgerufen von PerfDelta. Jeder Datensatz fasst echte Chrome-Besuche aus einem gleitenden Zeitraum von 28 Tagen zusammen. Anteile guter Werte beziehen sich auf die Origins mit Daten zur jeweiligen Metrik, die Bestehensquote auf die Origins mit allen drei Werten.
- **Plattform:** erkannt am Technologie-Stack der Startseite in unserer jüngsten Messung. 1.293 Origins wurden als Next.js erkannt. Websites mit erkanntem CMS, auch ein Headless-CMS mit Next.js als Frontend, zählen unter diesem CMS. Die Felddaten gelten für die gesamte Origin.
- **bfcache:** nur Origins, bei denen Zurück- und Vor-Navigationen mindestens zwei Prozent der Seitenaufrufe ausmachten (1.023 Next.js-Websites, insgesamt 8.868).
- **Soft Navigations:** Verhältnis des experimentellen INP mit Soft Navigations zum regulären INP, Median über 1.282 Next.js-Websites.
- **Zusammenhang, keine Ursache:** Next.js-Websites unterscheiden sich vom Rest der Stichprobe nicht allein im Framework. Viele sind Shops, Webanwendungen oder Marketing-Websites mit zahlreichen Drittanbieter-Skripten. Die Daten zeigen, wie Next.js-Websites in der Praxis abschneiden, nicht, wie schnell Next.js sein kann.
- **Stichprobe:** Die Websites stammen aus Tests und Studien von PerfDelta, größere Marken sind überrepräsentiert. Die Ergebnisse sind nicht repräsentativ für das gesamte Web. Weitere Zahlen finden Sie in den Website Speed Statistiken.