**Engineer**

# Server-side rendering

at danne en sides HTML på serveren ved hver forespørgsel

## Hvad det er

Server-side rendering, som regel forkortet SSR, betyder, at en sides HTML bliver dannet på serveren i det øjeblik, nogen beder om den. Browseren modtager et færdigt dokument med indholdet på plads og kan vise det, før noget script har kørt. I et JavaScript-framework kører de samme komponenter derefter en gang til i browseren, som overtager den HTML, den fik tilsendt, og gør den interaktiv.

Begrebet svarer på et spørgsmål om timing: hvornår, og hvor, bliver HTML'en lavet? Der er tre almindelige svar. En statisk sitegenerator laver den én gang, når sitet bygges, og sender den samme fil til alle. Server-side rendering laver den på serveren ved hver forespørgsel, så siden kan afhænge af, hvem der spørger, og af hvad data siger lige nu. Klientside-rendering, ofte kaldt CSR, sender et næsten tomt dokument og en pakke JavaScript, der bygger siden i browseren. Rigtige applikationer blander ofte alle tre, én route ad gangen.

Next.js siger det på samme måde. Dokumentationen nævner dynamic rendering som et andet navn for SSR og siger, at sidens HTML genereres ved hver forespørgsel. I Pages Router vælger en side det ved at eksportere en asynkron funktion ved navn `getServerSideProps`, som serveren kalder ved hver forespørgsel. Søsterfunktionen `getStaticProps` gør det samme arbejde én gang, når sitet bygges. React leverer maskineriet nedenunder. Med API'erne i `react-dom/server` bliver et træ af komponenter til HTML på serveren, enten som en strøm, med `renderToPipeableStream` til Node.js og `renderToReadableStream` til Web Streams, eller i ét stykke med den ældre `renderToString`, som React opfører som et legacy-API med begrænset funktionalitet.

Den anden halvdel foregår i browseren og hedder hydrering. Next.js beskriver den første indlæsning sådan: HTML'en giver en hurtig, ikke-interaktiv forhåndsvisning, og derefter hydrerer JavaScript komponenterne. Hydrering

definerer Next.js som Reacts måde at hæfte eventhandlere på DOM'en, så den statiske HTML bliver interaktiv. Reacts reference for `hydrateRoot` lægger ikke skjul på byttehandlen. HTML-øjebliksbilledet får appen til at se ud, som om den indlæses hurtigere, og siden ser færdig ud et øjeblik, før den er det.

### Brug begrebet i en beslutning

Brug »Hvad er server-side rendering?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

**Se vores ydelser** Kontakt os →

### Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

**Har styret full-stack-udviklingen af GIS-kort** Kontakt os →

## Hvorfor det betyder noget

Indholdet er med i det første svar. En læser på en langsom forbindelse ser tekst og billeder, mens JavaScript stadig er undervejs, og en crawler, der ikke kører JavaScript, ser den samme side som et menneske og ikke en tom skal. Browseren behøver ikke først at køre et script og derefter hente data, før den kan tegne rigtigt indhold.

Siden kan også afhænge af forespørgslen, og det er netop det, der adskiller den fra et statisk build. Next.js anbefaler `getServerSideProps` til data, der er personlige for brugeren eller først kendes ved forespørgslen, for eksempel en authorization-header eller den besøgendes placering, og henviser alt andet til generering, når sitet bygges. Data hentes tæt på, hvor de bor. Koden i funktionen kører kun på serveren, dens imports holdes ude af klientens bundle, og den kan spørge en database eller et CMS direkte. Én tur til serveren kan erstatte de forespørgsler, en browser ellers ville sende den ene efter den anden, når dens scripts var indlæst.

Omkostningerne er spejlbilledet af et statisk sites besparelser. Der skal køre en server, og den gør renderingsarbejdet igen ved hver forespørgsel. Svaret kan først begynde, når data er på plads, så uden streaming bestemmer det langsomste dataopslag tempoet for hele siden. Og rendering på serveren fjerner ikke JavaScript: browseren henter stadig hver komponent, den skal hydrere, og indtil hydreringen er færdig, kan en knap være tegnet på skærmen uden at gøre noget.

Det gør det til et valg pr. side og ikke pr. projekt. En side, der er ens for alle læsere, vinder ikke meget ved at blive renderet igen ved hver forespørgsel, mens en side, der skifter med læseren eller med data, der flytter sig fra minut til minut, er der, hvor SSR gør sin server fortjent. Dette site ligger i den statiske ende. Dets teknologivalg er statisk først, hvor det passer, så disse sider ikke kræver afvikling af applikationskode ved hver forespørgsel, og Hugo bygger sitet.

## Sådan ser det ud i praksis

Siden herunder er skrevet til denne artikel og bruger Pages Router. Den viser en indlogget kundes ordrer, sender en besøgende uden session videre til login-siden og beder alle caches om ikke at gemme resultatet. `readSession` og `db` står for det, applikationen bruger til sessioner og data.

```
// pages/orders.js
export async function getServerSideProps({ req, res }) {
  const session = await readSession(req.cookies.session);
  if (!session) {
    return { redirect: { destination: "/sign-in", permanent: false } };
  }

  const rows = await db.orders.listFor(session.userId);
  res.setHeader("Cache-Control", "no-store");

  return {
    props: {
      orders: rows.map((o) => ({
        id: o.id,
        number: o.number,
        status: o.status,
      })),
    },
  };
}

export default function Orders({ orders }) {
  return (
    <ul>
      {orders.map((o) => (
        <li key={o.id}>
          {o.number}: {o.status}
        </li>
      ))}
    </ul>
  );
}
```

Next.js kører funktionen ved hver forespørgsel til `/orders`. `req.cookies` er forespørgslens cookie-header, allerede parset. En `redirect` med `permanent: false` er en midlertidig omdirigering og ikke en permanent. Det, der kommer tilbage under `props`, skal kunne klare `JSON.stringify`, og det når også browseren som en del af den første HTML, så React kan hydrere listen med de samme data, som serveren brugte. Derfor skærer funktionen hver række ned til de tre felter, siden viser, i stedet for at sende databasens rækker direkte videre. Når en læser senere kommer via et `next/link`-link, beder Next.js serveren om at køre funktionen igen og får props tilbage som JSON uden en fuld sideindlæsning.

Next.js 16-frontenden fra grundlæggerens aktuelle arbejde hos NextMariner viser det problem, serverrendering løser. Portfolio-siden om frontenden tager udgangspunkt i et autentificeret dashboard, der skulle føles hurtigt og samtidig være interaktivt. Henter man alt i browseren, bliver den første indlæsning langsom, og N+1-mønsteret følger med: hver komponent vågner og sender sin egen request af sted, så en enkelt side bliver til en kaskade af rundture. På dashboardet og på de store oversigtslister henter en factory ved navn `createPrefetchedServerPage` nu sidens data på serveren og giver klienten de data, der allerede er hentet, så komponenterne starter med deres data i stedet for hver især at skulle spørge efter dem. De sider, der er flyttet over på den, indlæses med én rundtur i stedet for en hel storm af dem. Projektet holder fast i de to andre former, hvor de passer: offentlige sider genereres statisk, fordi de ikke ændrer sig fra bruger til bruger, og de reelt interaktive dele renderes fortsat på klienten. Der er også dokumenteret en strategi for cache-invalidering for hver dataforespørgsel, så data forbliver friske, uden at appen henter det, den allerede har, en gang til.

## Hvor det går galt

**Serveren og browseren er uenige.** Hydrering forventer, at browserens første rendering giver det samme resultat som serverens. En komponent, der tjekker `typeof window`, kalder `window.matchMedia` under renderingen eller skriver det aktuelle klokkeslæt, renderer én ting på serveren og en anden i browseren. Reacts liste over årsager nævner også forskellige data på de to sider og overskydende whitespace inde i rodelementet. I bedste fald gør en uoverensstemmelse hydreringen langsommere, og i værste fald ender eventhandlere på de forkerte elementer, så Reacts reference for `hydrateRoot` siger det ligeud: »You should treat mismatches as bugs and fix them.« Render det samme på begge sider. Hvor browseren virkelig skal vise noget andet, renderer

man serverens version først og skifter i en effect efter hydreringen, hvilket koster en ekstra rendering. `suppressHydrationWarning` hører til en uundgåelig forskel som et tidsstempel, og den virker kun ét niveau ned.

**Hemmeligheder følger med i props.** Alt, der returneres som props, skrives ind i den første HTML, så browseren kan hydrere, og Next.js-dokumentationen advarer mod at sende noget følsomt den vej. En funktion, der returnerer en hel brugerpost, giver dens password-hash og interne flag til enhver, der åbner sidens kildekode. Returnér kun de felter, siden viser, sådan som eksemplet ovenfor gør.

**En delt cache serverer én læsers side til den næste.** Next.js-dokumentationen viser caching af serverrenderede sider med `Cache-Control: public, s-maxage=10, stale-while-revalidate=59` . Kopieret over på en personlig side inviterer den header et CDN eller en proxy foran serveren til at gemme svaret og give det til den, der spørger bagefter. RFC 9111 definerer `public` som tilladelse til, at en cache gemmer et svar, den ellers ville afvise, og `s-maxage` som en levetid beregnet til delte caches. Hver af dem ophæver desuden reglen om, at en delt cache ikke må genbruge svaret på en forespørgsel, der havde en `Authorization` -header med. En side bygget af cookies eller loginoplysninger skal have `private` , som forbyder delte caches at gemme den, eller `no-store` , som forbyder alle caches det.

**Det langsomste opslag holder siden tilbage.** Renderes siden i ét stykke, sender den intet, før hvert eneste opslag bag den er færdigt, så ét langsomt opslag forsinker sidehovedet, navigationen og alt det andet. Reacts streamingrenderer deler siden ved `<Suspense>` -grænser. Den sender skallen, alt uden for grænserne, så snart den er klar, med fallbacks på plads, og streamer derefter hver resterende del med et lille inline-script, der sætter den ind. Der er én hage ved at sende tidligt: når streamingen er begyndt, kan statuskoden ikke længere ændres, så en manglende post eller et afvist tjek skal være afgjort, før skallen sendes, og `onShellError` er stedet, hvor en fejlstatus sættes.

## Læs mere

- Server-side Rendering (SSR) — Next.js om HTML dannet ved hver forespørgsel, og hvordan `getServerSideProps` adskiller sig fra `getStaticProps` .
- getServerSideProps — hvornår den skal bruges, hvad der når klienten i den første HTML, og caching med `Cache-Control` .
- Server React DOM APIs — de streaming- og legacy-API'er, React tilbyder til at rendere komponenter til HTML på serveren.

- hydrateRoot — hvordan React overtager serverrenderet HTML, og de almindelige årsager til uoverensstemmelser under hydrering.
- renderToPipeableStream — skallen, `<Suspense>`-grænser og forbeholdet om statuskoder ved streaming.
- RFC 9111: HTTP Caching — direktiverne `public`, `private`, `s-maxage` og `no-store` og reglerne for delte caches.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

**Se, hvad det koster** Kontakt os →

## Gør begreberne praktiske

Spørg, hvordan »Hvad er server-side rendering?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

**Se vores ydelser**

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/server-side-rendering/