Engineer

Single-page application

en webapp, der indlæser én side og ændrer den med JavaScript

Hvad det er

En single-page application, også skrevet single-page-app og som regel forkortet SPA, er en webapplikation, der indlæser ét HTML-dokument og derefter ændrer det dokument med JavaScript i stedet for at bede serveren om en ny side. Et klik på et link inde i den får ikke browseren til at forlade siden. Et script fanger klikket, henter de data, den næste visning skal bruge, skriver en del af DOM'en om og opdaterer adresselinjen, så URL'en stadig fortæller, hvor læseren er.

Ingen standard ejer begrebet, og dem, der bygger værktøjerne, siger det selv. Next.js indleder sin guide til emnet med »The definition of a SPA varies.« og fastlægger derefter en definition til eget brug, som Next.js kalder en strict SPA. Den har to kendetegn. Appen leveres af én HTML-fil, og hver route, hvert sideskift og hver datahentning håndteres af JavaScript i browseren. Og der er ingen fulde genindlæsninger af siden: i stedet for at bede om et nyt dokument for hver route ændrer JavaScript på klienten den aktuelle sides DOM og henter data efter behov. Denne artikel bruger den definition, fordi den er den mest direkte, der findes, og fordi den nævner den egenskab, alt andet afhænger af, nemlig et dokument, der aldrig bliver udskiftet.

Browserdelene er standardiserede. HTML-standardens History-interface giver en side `history.pushState()`, som tilføjer en ny post i sessionshistorikken og sætter URL'en uden at indlæse noget, og `history.replaceState()`, som i stedet skriver den aktuelle post om. Når læseren går tilbage eller frem mellem de poster, sender browseren en `popstate`-hændelse til vinduet og overlader det til siden at tegne indholdet igen. Det nyere Navigation API, som standarden præsenterer som efterfølgeren til de klassiske location- og history-API'er, arbejder fra den anden ende: en `navigate`-hændelse udløses, når en navigation begynder, og dens metode `intercept()` gør navigationen til en navigation inden for samme dokument, som siden selv håndterer.

Det modsatte er det almindelige website med flere sider, hvor hvert link beder serveren om et nyt dokument, og browseren smider det gamle væk. Om en sides HTML bliver lavet én gang af en statisk sitegenerator eller ved hver forespørgsel med server-side rendering, er et andet spørgsmål. Det handler om, hvordan HTML'en bliver til, og ikke om, hvad der sker, når den er kommet frem. Next.js udvisker grænsen med vilje. En statisk eksport fra Next.js skriver en HTML-fil for hver route, og skiftet mellem dem sker stadig på klienten.

Brug begrebet i en beslutning

Brug »Hvad er en single-page application?« 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

Fordi dokumentet aldrig bliver udskiftet, overlever alt, hvad der ligger i det, et klik: en halvt udfyldt formular, en video, der afspilles, scrollpositionen i et sidepanel, data, der allerede er hentet. Kun de data, den næste visning skal bruge, skal over netværket, og ikke en hel ny side med sidehoved, stylesheets og scripts. Et skift af visning kan være så lille som at skrive ét område af siden om.

Next.js nævner omkostningerne i samme åndedrag som definitionen. En strict SPA skal ofte indlæse en stor mængde JavaScript, før siden kan blive interaktiv, og datavandfald på klienten, hvor én forespørgsel først kan begynde, når en anden er kommet tilbage, er svære at styre. Alt, der læser siden uden at køre scripts, betaler også. Med én HTML-fil bag hver route får en crawler, der ikke kører JavaScript, det samme næsten tomme dokument, uanset hvilken URL den beder om. Next.js siger det samme om sine egne komponenter, der kun kører i browseren: crawlere, der ikke udfører JavaScript, ser fallbacken i stedet for komponenten.

Svaret fra Next.js er at beholde navigationen på klienten og flytte resten. Next.js deler automatisk JavaScript-bundles op og genererer flere HTML-indgange til forskellige routes, så en besøgende ikke indlæser kode, som en route ikke har brug for. Komponenten `next/link` henter routes på forhånd, hvilket giver de hurtige sideskift fra en strict SPA og samtidig holder applikationens routing-

tilstand i URL'en, hvor den kan linkes til og deles. En statisk eksport går et skridt videre og skriver en færdigrenderet side for hver route, så besøgende får indhold uden at vente på klientens bundle, mens skift mellem sider på klienten forbliver lige så hurtige som i en strict SPA.

Valget træffes heller ikke én gang for alle. Next.js beskriver et projekt, der starter som et statisk site eller endda som en strict SPA, der renderes helt på klienten, og som får serverfunktioner, efterhånden som det vokser, og ifølge Next.js kan eksisterende SPA'er migreres uden at blive skrevet helt om.

Sådan ser det ud i praksis

Routeren nedenfor er skrevet til denne artikel og er den mindste fungerende udgave af idéen. Den forudsætter, at siden har et `<main>` -element, og at serveren svarer på hver sti i appen med det samme HTML-dokument. De to visninger står for det, appen i virkeligheden viser.

```
// router.js
const views = {
  "/": { title: "Orders", text: "Your recent orders." },
  "/settings": { title: "Settings", text: "Delivery preferences." },
};
const main = document.querySelector("main");

function show(path, moveFocus) {
  const view = views[path] ?? views["/"];
  const heading = document.createElement("h1");
  const body = document.createElement("p");
  heading.textContent = view.title;
  heading.tabIndex = -1;
  body.textContent = view.text;
  main.replaceChildren(heading, body);
  document.title = view.title;
  if (moveFocus) heading.focus();
}

document.addEventListener("click", (event) => {
  const link = event.target.closest("a[href]");
  const modified = event.metaKey || event.ctrlKey || event.shiftKey;
  if (!link || link.origin !== location.origin) return;
  if (modified || event.button !== 0) return;
  event.preventDefault();
  history.pushState(null, "", link.href);
  show(location.pathname, true);
});
```

```
window.addEventListener("popstate", () => show(location.pathname, true));
show(location.pathname, false);
```

Klikhandleren overtager almindelige venstreklik på links til samme origin og lader klik med modifikatortaster passere, så det stadig virker at åbne et link i en ny fane. `preventDefault()` stopper browserens egen navigation, og `pushState()` registrerer en ny post i sessionshistorikken med linkets URL. Det midterste argument findes kun af historiske årsager, og derfor er det en tom streng. `show()` udskifter visningen, sætter dokumentets titel og flytter fokus til den nye overskrift, som en `tabIndex` på `-1` gør fokuserbar fra script, mens den holdes uden for tabulatorrækkefølgen. Lytteren på `popstate` tegner den matchende visning igen, når læseren går tilbage eller frem, og den sidste linje tegner den URL, siden blev åbnet på, hvilket kun virker, hvis serveren sender det samme dokument for `/settings` som for `/` . En rigtig app ville også vise sin egen side for adresser, den ikke kender, i stedet for stille at falde tilbage til startsiden, og den ville lade links til filer og til andre dele af sitet være i fred.

I grundlæggerens nuværende arbejde for NextMariner viser Next.js 16-frontenden omkostningssiden af det her på et rigtigt produkt. Portfolio-siden om frontenden beskriver et autentificeret dashboard, der skulle føles hurtigt og samtidig være reelt interaktivt, og nævner, hvad der sker, når alt hentes på klienten: den første indlæsning bliver langsom, og N+1-mønsteret dukker op, hvor hver komponent vågner og sender sin egen request af sted, indtil en enkelt side bliver til en kaskade af rundture. Det er netop det datavandfald på klienten, som Next.js-guiden advarer om, her mødt i praksis. Løsningen beholdt interaktiviteten på klientsiden, hvor den betød noget, og flyttede den første datahentning over på serveren. På dashboardet og de store oversigtslister giver en factory ved navn `createPrefetchedServerPage` hver side videre til klienten med dataene allerede på plads. Siden lægger ikke skjul på resten: de øvrige autentificerede sider renderes stadig på klienten og venter på deres tur, så den beskriver en migrering, der virker og har en oplagt næste side, og ikke en, der er afsluttet.

Hvor det går galt

Tilbage-knappen forlader appen. Hvis et script skifter visning uden at kalde `pushState()` , har sessionshistorikken ingen post for visningerne. Tilbage springer over hver visning, læseren har set, og forlader appen helt, og en genindlæsning lander på den URL, appen først blev åbnet på. Tilføj en post for hver visning, en læser ville kalde en side, brug `replaceState()` til ændringer, der ikke skal blive

selvstændige trin, som et filter, der er ved at blive tastet ind, og håndtér `popstate`. Next.js argumenterer for at holde routing-tilstanden i URL'en af samme grund: det er den, der gør det muligt at linke til en visning og dele den.

Et link direkte til en underside møder en server, der kun kender én fil. I en strict SPA betjener én HTML-fil hver route, men adresselinjen viser `/settings`, og det er den adresse, browseren beder om ved en genindlæsning, eller når nogen åbner et delt link. En server, der kun har `index.html`, svarer med en 404. Konfigurér enten serveren til at svare på hver sti i appen med dokumentet, eller generér en fil pr. route. Den statiske eksport i Next.js gør det sidste: dens `out`-mappe indeholder filer som `index.html`, `404.html` og `blog/post-1.html`, og guiden viser en nginx-regel, `try_files $uri $uri.html $uri/ =404;`, der kobler en pæn URL til den matchende fil.

Fokus bliver, hvor den gamle visning var. `pushState()` ændrer sessionshistorikken og URL'en og intet andet. Når et script har skiftet visningen, bliver tastaturfokus på det link, der blev klikket på, eller går tabt, hvis elementet, der havde det, blev fjernet, og intet fortæller hjælpemiddelteknologi, at en ny visning er kommet, medmindre siden gør det. Sæt titlen, og flyt fokus til den nye visnings overskrift, som eksemplet gør. Navigation API påtager sig selv mere af det arbejde. En navigation, der håndteres med `intercept()`, nulstiller som standard fokus til det første element med `autofocus` eller ellers til body, og når handleren er færdig, signalerer browseren til brugeren, at navigationen er gennemført, hvilket ifølge standarden også omfatter hjælpemiddelteknologi. En app, der selv placerer fokus, fravælger det med `focusReset: "manual"`.

Alt betales på forhånd. En strict SPA skal ofte indlæse en stor mængde JavaScript, før noget er interaktivt, og dens første dataforespørgsler starter først, når den kode kører. Del bundlen op efter route, så en besøgende kun henter det, den aktuelle visning skal bruge, og start de første forespørgsler på serveren. Mønsteret i Next.js henter data i en Server Component uden at afvente dem og sender promiset videre til en Client Component, som pakker det ud med `use()`. Når forespørgslen starter, før resten af appen renderes, kan svaret ifølge guiden streames med det samme, og datavandfald på klienten undgås.

Læs mere

- How to build single-page applications with Next.js — definitionen af en strict SPA, hvad den koster, og de mønstre, Next.js bruger til at beholde navigationen på klienten og flytte arbejde over på serveren.

- HTML Standard: The History interface — `pushState()` , `replaceState()` , og hvordan poster i sessionshistorikken tilføjes og udskiftes.
- HTML Standard: Navigation API — `navigate` -hændelsen, `intercept()` og den håndtering af fokus og scroll, der følger med.
- How to create a static export of your Next.js application — én HTML-fil pr. route, hvad en statisk eksport ikke kan, og pæne URL'er med nginx.

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 en single-page application?« 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/single-page-application/