Engineer

Single-page application

en nettapp som laster én side og endrer den med JavaScript

Hva det er

En single-page application, også kalt enkeltsideapp og som regel forkortet til SPA, er en webapplikasjon som laster inn ett HTML-dokument og deretter endrer det dokumentet med JavaScript i stedet for å be serveren om en ny side. Et klikk på en lenke inne i den får ikke nettleseren til å forlate siden. Et skript fanger opp klikket, henter de dataene neste visning trenger, skriver om en del av DOM-en og oppdaterer adresselinjen, slik at URL-en fortsatt forteller hvor leseren er.

Ingen standard eier begrepet, og de som lager verktøyene, sier det selv. Next.js åpner veiledningen sin om emnet med «The definition of a SPA varies.» og fastsetter deretter en definisjon til eget bruk, som Next.js kaller en strict SPA. Den har to kjennetegn. Appen leveres av én HTML-fil, og hver rute, hver sideovergang og hver datahenting håndteres av JavaScript i nettleseren. Og det er ingen fullstendige omlastinger av siden: i stedet for å be om et nytt dokument for hver rute endrer JavaScript på klienten DOM-en til den gjeldende siden og henter data etter behov. Denne artikkelen bruker den definisjonen fordi den er den mest direkte som finnes, og fordi den nevner egenskapen alt annet avhenger av, nemlig et dokument som aldri blir byttet ut.

Nettleserdelene er standardiserte. History-grensesnittet i HTML-standarden gir en side `history.pushState()`, som legger til en ny oppføring i økthistorikken og setter URL-en uten å laste inn noe, og `history.replaceState()`, som i stedet skriver om den gjeldende oppføringen. Når leseren går tilbake eller fram mellom disse oppføringene, sender nettleseren en `popstate`-hendelse til vinduet og overlater til siden å tegne innholdet på nytt. Det nyere Navigation API, som standarden presenterer som etterfølgeren til de klassiske location- og history-API-ene, arbeider fra den andre enden: en `navigate`-hendelse utløses når en navigasjon begynner, og metoden `intercept()` gjør navigasjonen om til en navigasjon innenfor det samme dokumentet, som siden håndterer selv.

Det motsatte er det vanlige nettstedet med flere sider, der hver lenke ber serveren om et nytt dokument og nettleseren kaster det gamle. Om HTML-en til en side blir laget én gang av en statisk nettstedsgenerator eller ved hver forespørsel med server-side rendering, er et annet spørsmål. Det handler om hvordan HTML-en blir til, og ikke om hva som skjer når den har kommet fram. Next.js visker ut grensen med vilje. En statisk eksport fra Next.js skriver en HTML-fil for hver rute, og overgangen mellom dem skjer fortsatt på klienten.

Bruk begrepet i en beslutning

Bruk «Hva er en single-page application?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Grensesnitt som laster raskt, fungerer på mobilen og oppfører seg likt i alle nettlesere kundene dine måtte bruke.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Styrte fullstack-utviklingen av GIS-kart — fullstack-kartutvikling, fra database til nettleser

Styrte fullstack-utviklingen av GIS-kart Ta kontakt →

Hvorfor det er viktig

Fordi dokumentet aldri blir byttet ut, overlever alt som ligger i det, et klikk: et halvt utfylt skjema, en video som spilles av, rulleposisjonen i et sidepanel, data som allerede er hentet. Bare dataene neste visning trenger, må over nettverket, og ikke en hel ny side der toppen, stilarkene og skriptene følger med på nytt. Et bytte av visning kan være så lite som å skrive om ett område av siden.

Next.js nevner kostnadene i samme åndedrag som definisjonen. En strict SPA må ofte laste inn store mengder JavaScript før siden kan bli interaktiv, og kaskader av forespørsler på klienten, der én forespørsel først kan starte når en annen har kommet tilbake, er vanskelige å styre. Alt som leser siden uten å kjøre skript, betaler også. Med én HTML-fil bak hver rute får en søkerobot som ikke kjører JavaScript, det samme nesten tomme dokumentet uansett hvilken URL den ber om. Next.js sier det samme om sine egne komponenter som bare kjører i nettleseren: søkeroboter som ikke kjører JavaScript, ser fallbacken i stedet for komponenten.

Svaret fra Next.js er å beholde navigasjonen på klienten og flytte resten. Next.js deler automatisk opp JavaScript-bundlene og genererer flere HTML-inngangspunkter til forskjellige ruter, slik at en besøkende ikke laster inn kode en rute ikke trenger. Komponenten `next/link` forhåndshenter ruter, noe som gir de raske sideovergangene fra en strict SPA og samtidig holder applikasjonens

rutingtilstand i URL-en, der den kan lenkes til og deles. En statisk eksport går et steg videre og skriver en ferdig rendret side for hver rute, slik at besøkende får innhold uten å vente på klientens bundle, mens overganger mellom sider på klienten forblir like raske som i en strict SPA.

Valget tas heller ikke én gang for alle. Next.js beskriver et prosjekt som starter som et statisk nettsted, eller til og med som en strict SPA som rendres helt på klienten, og som får serverfunksjoner etter hvert som det vokser, og ifølge Next.js kan eksisterende SPA-er migreres uten å skrives helt om.

Slik ser det ut i praksis

Ruteren nedenfor er skrevet for denne artikkelen og er den minste versjonen av ideen som fungerer. Den forutsetter at siden har et `<main>` -element, og at serveren svarer på hver sti i appen med det samme HTML-dokumentet. De to visningene står for det appen egentlig 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);
```

Klikkhåndtereren tar over vanlige venstreklikk på lenker til samme origin og slipper gjennom klikk med modifikatortaster, slik at det fortsatt går an å åpne en lenke i en ny fane. `preventDefault()` stopper nettleserens egen navigasjon, og `pushState()` registrerer en ny oppføring i økthistorikken med lenkens URL. Det midterste argumentet finnes bare av historiske grunner, og derfor er det en tom streng. `show()` bytter ut visningen, setter dokumentets tittel og flytter fokus til den nye overskriften, som en `tabIndex` på `-1` gjør fokuserbar fra skript, mens den holdes utenfor tabulatorrekkefølgen. Lytteren på `popstate` tegner den tilsvarende visningen på nytt når leseren går tilbake eller fram, og den siste linjen tegner den URL-en siden ble åpnet på, noe som bare fungerer hvis serveren sender det samme dokumentet for `/settings` som for `/`. En ekte app ville også vise sin egen side for adresser den ikke kjenner, i stedet for å falle stille tilbake til startsiden, og den ville la lenker til filer og til andre deler av nettstedet være i fred.

I grunnleggerens nåværende arbeid hos NextMariner viser Next.js 16-frontenden kostnadssiden av dette på et ekte produkt. Porteføljesiden om frontenden beskriver et innlogget dashbord som skulle føles raskt og samtidig være virkelig interaktivt, og forteller hva som skjer når alt hentes på klienten: den første innlastingen blir treg, og N+1-mønsteret dukker opp, der hver komponent sender sin egen forespørsel når den våkner, helt til én side er blitt en kaskade av rundturer. Det er nettopp kaskaden av forespørsler på klienten som Next.js-veiledningen advarer mot, her møtt i praksis. Løsningen beholdt interaktiviteten på klientsiden der den betydde noe, og flyttet den første datahentingen over på serveren. På dashbordet og de store katalogoversiktene gir en fabrikk som heter `createPrefetchedServerPage`, hver side videre til klienten med dataene allerede på plass. Siden legger ikke skjul på resten: de andre innloggede sidene rendres fortsatt på klienten og venter på tur, så den beskriver en migrering som virker og har en opplagt neste kandidat, og ikke en som er fullført.

Hvor det går galt

Tilbake-knappen forlater appen. Hvis et skript bytter visning uten å kalle `pushState()`, har økthistorikken ingen oppføring for visningene. Tilbake hopper over hver visning leseren har sett, og forlater appen helt, og en omlasting lander på den URL-en appen først ble åpnet på. Legg til en oppføring for hver visning en leser ville kalt en side, bruk `replaceState()` til endringer som ikke skal bli egne

steg, som et filter som er i ferd med å bli skrevet inn, og håndter `popstate`. Next.js argumenterer for å holde rutingtilstanden i URL-en av samme grunn: det er den som gjør det mulig å lenke til en visning og dele den.

En lenke rett til en underside møter en server som bare kjenner én fil. I en strict SPA betjener én HTML-fil hver rute, men adresselinjen viser `/settings`, og det er den adressen nettleseren ber om ved en omlasting, eller når noen åpner en delt lenke. En server som bare har `index.html`, svarer med en 404. Konfigurer enten serveren til å svare på hver sti i appen med dokumentet, eller generer en fil per rute. Den statiske eksporten i Next.js gjør det siste: `out`-mappen inneholder filer som `index.html`, `404.html` og `blog/post-1.html`, og veiledningen viser en nginx-regel, `try_files $uri $uri.html $uri/ =404;`, som kobler en pen URL til den tilsvarende filen.

Fokuset blir værende der den gamle visningen var. `pushState()` endrer økthistorikken og URL-en og ingenting annet. Når et skript har byttet visningen, blir tastaturfokuset værende på lenken som ble klikket, eller går tapt hvis elementet som hadde det, ble fjernet, og ingenting forteller hjelpemiddelteknologi at en ny visning har kommet, med mindre siden gjør det. Sett tittelen, og flytt fokus til overskriften i den nye visningen, slik eksemplet gjør. Navigation API tar på seg mer av dette arbeidet selv. En navigasjon som håndteres med `intercept()`, nullstiller som standard fokus til det første elementet med `autofocus` eller ellers til body, og når håndtereren er ferdig, signaliserer nettleseren til brukeren at navigasjonen er fullført, noe som ifølge standarden også omfatter hjelpemiddelteknologi. En app som plasserer fokus selv, velger det bort med `focusReset: "manual"`.

Alt betales på forhånd. En strict SPA må ofte laste inn store mengder JavaScript før noe er interaktivt, og de første dataforespørslene starter først når den koden kjører. Del opp bundlen etter rute, slik at en besøkende bare laster ned det den gjeldende visningen trenger, og start de første forespørslene på serveren. Mønsteret i Next.js henter data i en Server Component uten å vente på dem og sender promiset videre til en Client Component, som pakker det ut med `use()`. Når forespørselen starter før resten av appen rendres, kan svaret ifølge veiledningen strømmes med en gang, og kaskader av forespørsler på klienten unngås.

Les mer

- How to build single-page applications with Next.js — definisjonen av en strict SPA, hva den koster, og mønstrene Next.js bruker for å beholde navigasjonen på klienten og flytte arbeid over på serveren.
- HTML Standard: The History interface — `pushState()`, `replaceState()` og hvordan oppføringer i økthistorikken legges til og byttes ut.
- HTML Standard: Navigation API — `navigate`-hendelsen, `intercept()` og håndteringen av fokus og rulling som følger med.
- How to create a static export of your Next.js application — én HTML-fil per rute, hva en statisk eksport ikke kan, og pene URL-er med nginx.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en single-page application?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/single-page-application/