Engineer

Single-Page-Anwendung

Web-App, die eine Seite lädt und per JavaScript verändert

Was es ist

Eine Single-Page-Anwendung, auch Single-Page-App und meist kurz SPA genannt, ist eine Webanwendung, die ein HTML-Dokument lädt und es anschließend mit JavaScript verändert, statt beim Server eine neue Seite anzufordern. Ein Klick auf einen internen Link entlädt die Seite nicht. Ein Skript fängt den Klick ab, lädt die für die nächste Ansicht benötigten Daten, schreibt einen Teil des DOM um und aktualisiert die Adressleiste. Die URL zeigt dadurch weiterhin die aktuelle Position in der Anwendung an.

Kein Standard definiert den Begriff verbindlich, und die Hersteller der Werkzeuge räumen das ein. Next.js beginnt seinen Leitfaden dazu mit „Die Definition einer SPA variiert“ und legt dann für sich selbst eine sogenannte strikte SPA fest. Sie hat zwei Eigenschaften. Eine einzige HTML-Datei bedient die ganze Anwendung, und JavaScript im Browser kümmert sich um jede Route, jeden Seitenwechsel und jede Datenanfrage. Außerdem wird die Seite nie ganz neu geladen: Bei einem Routenwechsel schreiben Skripte auf dem Client das DOM der bereits geöffneten Seite um und laden Daten nach Bedarf, statt vom Server ein neues Dokument anzufordern. Dieser Artikel verwendet diese Definition, weil sie die direkteste verfügbare ist und die Eigenschaft benennt, von der alles andere abhängt: ein Dokument, das nie ersetzt wird.

Die Browserbausteine sind standardisiert. Die History-Schnittstelle des HTML-Standards stellt einer Seite `history.pushState()` bereit. Der Aufruf fügt dem Sitzungsverlauf einen Eintrag hinzu und setzt die URL, ohne etwas zu laden. `history.replaceState()` überschreibt stattdessen den aktuellen Eintrag. Wechselt jemand mit Zurück oder Vorwärts zwischen diesen Einträgen, löst der Browser am Fenster ein `popstate` -Ereignis aus und überlässt der Seite das Neuzeichnen. Die neuere Navigation API, die der Standard als Nachfolgerin der klassischen Location- und History-APIs vorstellt, setzt am anderen Ende an: Zu Beginn einer

Navigation wird ein `navigate`-Ereignis ausgelöst. Seine Methode `intercept()` macht daraus eine Navigation innerhalb desselben Dokuments, die die Seite selbst behandelt.

Das Gegenteil ist die gewöhnliche Website mit mehreren Seiten. Jeder Link fordert dort ein neues Dokument vom Server an, und der Browser verwirft das alte. Ob das HTML einer Seite einmal durch einen statischen Site-Generator oder bei jeder Anfrage durch serverseitiges Rendering entsteht, ist eine andere Frage. Sie betrifft die Erzeugung des HTML und nicht das, was nach seiner Ankunft im Browser geschieht. Next.js verwischt die Grenze bewusst: Sein statischer Export schreibt für jede Route eine HTML-Datei, und der Wechsel zwischen ihnen findet trotzdem auf dem Client statt.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Single-Page-Anwendung?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Oberflächen, die schnell laden, auf dem Smartphone funktionieren und sich in allen Browsern Ihrer Kunden gleich verhalten.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Full-Stack-GIS-Kartenentwicklung geleitet — Full-Stack-Kartenentwicklung von der Datenbank bis zum Browser

Full-Stack-GIS-Kartenentwicklung geleitet Kontakt aufnehmen →

Warum es wichtig ist

Weil das Dokument nie ersetzt wird, übersteht alles darin einen Klick: ein halb ausgefülltes Formular, ein laufendes Video, die Scrollposition einer Seitenleiste, bereits abgerufene Daten. Nur die für die nächste Ansicht benötigten Daten müssen durchs Netz, keine ganz neue Seite samt Kopfbereich, Stilen und Skripten. Ein Ansichtswechsel kann sich auf das Umschreiben eines einzigen Bereichs der Seite beschränken.

Next.js nennt die Kosten im selben Atemzug wie die Definition. Eine strikte SPA muss oft viel JavaScript laden, bevor die Seite interaktiv wird. Datenanfragekaskaden auf dem Client, bei denen eine Anfrage erst nach der Antwort auf eine andere beginnen kann, sind schwer zu steuern. Auch für Systeme, die die Seite ohne Skriptausführung lesen, hat das Nachteile. Steht hinter jeder Route eine einzige HTML-Datei, erhält ein Crawler ohne JavaScript stets dasselbe fast leere Dokument, egal welche URL er anfragt. Next.js sagt

dasselbe über seine ausschließlich im Browser ausgeführten Komponenten: Wer die Seite ohne Skriptausführung indexiert, erhält den Ersatzinhalt statt der Komponente.

Die Antwort von Next.js besteht darin, die clientseitige Navigation zu behalten und den Rest zu verlagern. Next.js teilt JavaScript-Bundles automatisch auf und erzeugt mehrere HTML-Einstiegspunkte für verschiedene Routen. Besucher laden so keinen Code, den eine Route nicht braucht. Seine Komponente `next/link` lädt Routen im Voraus. Routenwechsel wirken dadurch so unmittelbar wie in einer vollständig auf dem Client gerenderten SPA. Zugleich bleibt der Routing-Zustand der Anwendung in der URL und lässt sich verlinken und teilen. Ein statischer Export geht weiter und schreibt für jede Route eine vollständig gerenderte Seite. Besucher erhalten dadurch Inhalt, ohne auf das Client-Bundle zu warten, während clientseitige Seitenwechsel so schnell wie in einer strikten SPA bleiben.

Die Entscheidung fällt auch nicht ein für alle Mal. Next.js beschreibt ein Projekt, das als statische Website oder sogar als vollständig clientseitig gerenderte strikte SPA beginnt und beim Wachsen Serverfunktionen ergänzt. Eine bestehende SPA könne umziehen, ohne von Grund auf neu geschrieben zu werden.

Wie es in der Praxis aussieht

Der folgende, für diesen Artikel geschriebene Router ist die kleinste funktionsfähige Fassung der Idee. Er setzt ein `<main>` -Element auf der Seite voraus und einen Server, der jeden Anwendungspfad mit demselben HTML-Dokument beantwortet. Die beiden Ansichten stehen für das, was die Anwendung tatsächlich anzeigt.

```
// 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);
```

Der Klick-Handler übernimmt einfache Linksklicks auf Links zum selben Origin und lässt modifizierte Klicks durch. Ein Link lässt sich deshalb weiterhin in einem neuen Tab öffnen. `preventDefault()` unterbindet die eigene Navigation des Browsers, und `pushState()` trägt einen neuen Eintrag mit der Link-URL in den Sitzungsverlauf ein. Sein mittleres Argument existiert nur aus historischen Gründen und ist deshalb eine leere Zeichenfolge. `show()` ersetzt die Ansicht, setzt den Dokumenttitel und verschiebt den Fokus auf die neue Überschrift. Ein `tabIndex` von `-1` macht sie per Skript fokussierbar, hält sie aber aus der Tab-Reihenfolge heraus. Der Listener für `popstate` zeichnet beim Zurück- oder Vorwärtsgehen die passende Ansicht neu. Die letzte Zeile zeichnet die zur anfänglichen URL gehörende Ansicht. Das funktioniert nur, wenn der Server für `/settings` dasselbe Dokument wie für `/` sendet. Eine echte Anwendung würde außerdem eine eigene Nicht-gefunden-Ansicht zeigen, statt still auf die Startseite zurückzufallen, und Links zu Dateien und anderen Teilen der Website unangetastet lassen.

In der aktuellen Arbeit der Gründerin bei NextMariner zeigt das Frontend mit Next.js 16 diese Kosten an einem echten Produkt. Der zugehörige Portfolioeintrag beschreibt ein Dashboard mit Anmeldung, das sich schnell anfühlen und wirklich interaktiv bleiben musste. Er benennt auch, was passiert, wenn alle Daten im Client abgerufen werden: Der erste Ladevorgang zieht sich hin, und das N+1-Muster tritt auf. Jede Komponente wird aktiv und sendet ihre eigene Anfrage, bis eine einzelne Seite eine Kaskade von Hin- und Rückübertragungen auslöst. Damit begegnet einem in der Praxis genau die Kaskade clientseitiger Datenabrufe, vor der der Next.js-Leitfaden warnt. Die Lösung behielt clientseitige Interaktivität dort bei, wo sie wichtig war, und

verlagerte den ersten Datenabruf auf den Server. Für das Dashboard und die großen Verzeichnislisten übergibt eine Factory namens `createPrefetchedServerPage` jede Seite mit bereits vorhandenen Daten an den Client. Der Eintrag benennt offen, was noch aussteht: Die übrigen Seiten mit Anmeldung werden weiterhin im Client gerendert und warten, bis sie an der Reihe sind. Er beschreibt also eine funktionierende Migration, bei der die nächste umzustellende Seite klar ist, die aber noch nicht abgeschlossen ist.

Wo es schiefgeht

Die Zurück-Schaltfläche verlässt die Anwendung. Tauscht ein Skript Ansichten ohne `pushState()` aus, gibt es dafür keinen Eintrag im Sitzungsverlauf. Zurück überspringt jede gesehene Ansicht und verlässt die Anwendung ganz. Ein Neuladen landet bei der URL, mit der die Anwendung zuerst geöffnet wurde. Legen Sie für jede Ansicht, die ein Leser als Seite verstehen würde, einen Eintrag an. Verwenden Sie `replaceState()` für Änderungen, die keine eigenen Schritte werden sollen, etwa einen beim Tippen veränderten Filter, und behandeln Sie `popstate`. Next.js empfiehlt aus demselben Grund, den Routing-Zustand in der URL zu halten: So lässt sich eine Ansicht verlinken und teilen.

Ein Link auf eine Unterseite trifft auf einen Server, der nur eine Datei kennt. In einer strikten SPA bedient eine HTML-Datei jede Route. Die Adressleiste zeigt aber `/settings`, und genau das fordert der Browser beim Neuladen oder Öffnen eines geteilten Links an. Ein Server mit nur einer `index.html` antwortet mit 404. Konfigurieren Sie ihn so, dass er jeden Anwendungspfad mit dem Dokument beantwortet, oder erzeugen Sie eine Datei pro Route. Der statische Export von Next.js tut Letzteres: Sein Ordner `out` enthält Dateien wie `index.html`, `404.html` und `blog/post-1.html`. Der Leitfaden zeigt eine nginx-Regel, `try_files $uri $uri.html $uri/ =404;`, die eine saubere URL der passenden Datei zuordnet.

Der Fokus bleibt bei der alten Ansicht. `pushState()` ändert den Sitzungsverlauf und die URL, sonst nichts. Nach einem skriptgesteuerten Ansichtswechsel bleibt der Tastaturfokus auf dem angeklickten Link oder geht verloren, wenn sein Element entfernt wurde. Keine Meldung informiert assistive Technik über die neue Ansicht, sofern die Seite das nicht selbst tut. Setzen Sie den Titel und verschieben Sie den Fokus wie im Beispiel auf die Überschrift der neuen Ansicht. Die Navigation API übernimmt davon mehr selbst. Eine mit `intercept()` behandelte Navigation setzt den Fokus standardmäßig auf das erste Element mit `autofocus`, andernfalls auf den Dokumentkörper. Nach Abschluss des Handlers

meldet der Browser dem Nutzer den Abschluss. Nach dem Standard schließt das assistive Technik ein. Eine Anwendung, die den Fokus selbst setzt, schaltet das automatische Zurücksetzen mit `focusReset: "manual"` ab.

Alles wird im Voraus bezahlt. Eine strikte SPA muss oft viel JavaScript laden, bevor irgendetwas interaktiv ist. Ihre ersten Datenanfragen beginnen erst, wenn dieser Code läuft. Teilen Sie das Bundle nach Routen auf, damit Besucher nur den für die aktuelle Ansicht benötigten Code laden, und beginnen Sie die ersten Anfragen auf dem Server. Das Next.js-Muster dafür beginnt den Datenabruf in einer Server Component, ohne dessen Abschluss abzuwarten, und reicht das Promise an eine Client Component weiter. Diese entpackt es mit `use()`. Weil die Anfrage schon vor dem Rendern der restlichen Anwendung beginnt, kann laut Leitfaden sofort die Übertragung der Antwort als Datenstrom beginnen. Dem Client bleibt eine Kaskade erspart, in der jede Anfrage auf die vorherige wartet.

Weiterführende Lektüre

- Single-Page-Anwendungen mit Next.js bauen — Die Definition einer strikten SPA, ihre Kosten und Next.js-Muster, die clientseitige Navigation beibehalten und Arbeit auf den Server verlagern.
- HTML Standard: die History-Schnittstelle — `pushState()`, `replaceState()` und das Hinzufügen und Ersetzen von Einträgen im Sitzungsverlauf.
- HTML Standard: Navigation API — Das Ereignis `navigate`, `intercept()` und die damit verbundene Fokus- und Scrollsteuerung.
- Einen statischen Export einer Next.js-Anwendung erzeugen — Eine HTML-Datei pro Route, die Grenzen eines statischen Exports und das Ausliefern sauberer URLs mit nginx.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Single-Page-Anwendung?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/single-page-application/