Engineer

application monopage

Application web modifiant une seule page avec JavaScript

Ce que c’est

Une application monopage, appelée en anglais single-page application ou single-page app et généralement abrégée en SPA, est une application web qui charge un document HTML, puis le modifie avec JavaScript au lieu de demander une nouvelle page au serveur. Un clic sur un lien interne ne décharge pas la page. Un script intercepte le clic, récupère les données nécessaires à la vue suivante, réécrit une partie du DOM et met à jour la barre d’adresse : l’URL indique ainsi toujours où se trouve le lecteur.

Le terme ne relève d’aucune norme, et les concepteurs des outils le reconnaissent. Next.js commence son guide par « La définition d’une SPA varie », puis fixe la sienne, qu’il appelle une SPA stricte. Elle possède deux caractéristiques. Un seul fichier HTML dessert toute l’application, et JavaScript dans le navigateur gère chaque route, chaque transition de page et chaque requête de données. La page n’est jamais entièrement rechargée : lors d’un changement de route, les scripts du client réécrivent le DOM de la page déjà ouverte et chargent les données au besoin, plutôt que de demander un nouveau document au serveur. Cet article retient cette définition, la plus directe proposée : elle nomme la propriété dont tout le reste dépend, un document qui n’est jamais remplacé.

Les mécanismes du navigateur sont normalisés. L’interface History de HTML Standard donne à une page `history.pushState()`, qui ajoute une entrée à l’historique de session et modifie l’URL sans rien charger, et `history.replaceState()`, qui réécrit plutôt l’entrée actuelle. Lorsque le lecteur utilise Précédent ou Suivant entre ces entrées, le navigateur déclenche un événement `popstate` sur la fenêtre et laisse à la page le soin de refaire l’affichage. La Navigation API, plus récente et présentée par la norme comme la relève des API classiques de gestion de l’adresse et de l’historique, aborde la

question par l'autre bout : un événement `navigate` se déclenche au début d'une navigation, et sa méthode `intercept()` la transforme en navigation au sein du même document, prise en charge par la page.

À l'opposé se trouve le site multipage ordinaire : chaque lien demande un nouveau document au serveur, et le navigateur abandonne l'ancien. Que le HTML d'une page soit produit une fois par un générateur de site statique ou à chaque requête par rendu côté serveur est une question distincte. Elle porte sur la production du HTML, pas sur ce qui se passe après sa réception. Next.js brouille volontairement cette frontière : son export statique écrit un fichier HTML pour chaque route, et les déplacements entre ces pages continuent de se faire côté client.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'une application monopage ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des interfaces rapides, utilisables sur téléphone et cohérentes dans tous les navigateurs de vos clients.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Développement cartographique GIS full-stack — cartographie full-stack, de la base au navigateur

Développement cartographique GIS full-stack Contactez-nous →

Pourquoi c'est important

Le document n'étant jamais remplacé, ce qu'il contient survit au clic : un formulaire à moitié rempli, une vidéo en cours de lecture, la position de défilement d'un panneau latéral, des données déjà récupérées. Seules les données nécessaires à la vue suivante traversent le réseau, pas une nouvelle page entière avec son en-tête, ses styles et ses scripts. Un changement de vue peut se limiter à réécrire une seule zone de la page.

Next.js présente les coûts en même temps que la définition. Une SPA stricte doit souvent charger beaucoup de JavaScript avant de devenir interactive, et les cascades de données côté client — une requête ne pouvant commencer qu'au retour d'une autre — sont difficiles à gérer. Tout ce qui lit la page sans exécuter de scripts en paie aussi le prix. Avec le même fichier HTML derrière toutes les routes, un robot sans JavaScript reçoit le même document presque vide quelle que soit l'URL demandée. Next.js fait la même remarque pour ses composants exclusivement exécutés dans le navigateur : un outil d'indexation sans scripts reçoit le contenu de repli, pas le composant.

Sa réponse est de conserver la navigation côté client et de déplacer le reste. Next.js découpe automatiquement les paquets JavaScript et génère plusieurs points d'entrée HTML vers différentes routes : le visiteur ne charge ainsi pas le code inutile à une route. Son composant `next/link` précharge les routes. Les transitions semblent alors aussi immédiates que dans une SPA entièrement rendue côté client, tout en conservant l'état de routage de l'application dans l'URL pour pouvoir créer un lien et la partager. Un export statique va plus loin et écrit une page entièrement rendue par route : les visiteurs reçoivent le contenu sans attendre le paquet client, tandis que les transitions côté client restent aussi rapides que dans une SPA stricte.

Le choix n'est pas non plus définitif. Next.js décrit un projet qui commence comme site statique, voire comme SPA stricte entièrement rendue côté client, puis ajoute des fonctions serveur au fil de sa croissance. Il indique qu'une SPA existante peut migrer sans être réécrite de zéro.

À quoi cela ressemble en pratique

Le routeur ci-dessous, écrit pour cet article, est la plus petite version fonctionnelle de ce principe. Il suppose que la page possède un élément `<main>` et que le serveur répond à chaque chemin de l'application avec le même document HTML. Les deux vues tiennent lieu de ce que l'application afficherait réellement.

```
// 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);
```

Le gestionnaire de clic prend en charge les clics gauches ordinaires sur les liens de même origine et laisse passer ceux accompagnés de touches modificatrices : ouvrir un lien dans un nouvel onglet reste donc possible. `preventDefault()` bloque la navigation du navigateur, et `pushState()` inscrit une nouvelle entrée dans l'historique de session avec l'URL du lien. Son argument central n'existe que pour des raisons historiques, d'où la chaîne vide. `show()` remplace la vue, définit le titre du document et place le focus sur le nouveau titre : un `tabIndex` de `-1` permet de le cibler par script tout en l'excluant de l'ordre de tabulation. L'écouteur `popstate` redessine la vue correspondante lorsque le lecteur utilise Précédent ou Suivant. La dernière ligne affiche la vue de l'URL ayant servi à ouvrir la page : cela ne fonctionne que si le serveur envoie le même document pour `/settings` que pour `/` . Une véritable application afficherait aussi sa propre vue de page introuvable plutôt que de revenir silencieusement à l'accueil, et laisserait intacts les liens vers des fichiers ou d'autres parties du site.

Dans le travail actuel de la fondatrice pour NextMariner, le frontend Next.js 16 montre les coûts de cette approche sur un produit réel. La présentation de cette réalisation décrit un tableau de bord derrière une authentification, qui devait sembler rapide et rester réellement interactif. Elle nomme ce qui se passe lorsque toutes les données sont récupérées côté client : le premier chargement traîne et le schéma N+1 apparaît. Chaque composant se réveille et envoie sa propre requête, jusqu'à transformer une page en cascade d'allers-retours. C'est la cascade de données côté client contre laquelle le guide Next.js met en garde, rencontrée en pratique. La correction a conservé l'interactivité côté client là où elle importait et déplacé la première récupération sur le serveur. Sur le tableau de bord et les grandes listes d'annuaire, une fabrique appelée `createPrefetchedServerPage` remet chaque page au client avec ses données déjà en place. La présentation précise le reste sans détour : les autres pages authentifiées sont encore rendues côté client et attendent leur tour. Elle décrit donc une migration qui fonctionne et dont la prochaine page à migrer est évidente, pas une migration terminée.

Où les problèmes surviennent

Le bouton Précédent fait quitter l'application. Si un script remplace les vues sans appeler `pushState()`, l'historique de session ne les contient pas. Précédent saute toutes les vues parcourues et fait quitter l'application ; un rechargement revient à l'URL d'ouverture initiale. Ajoutez une entrée pour chaque vue que le lecteur considérerait comme une page, utilisez `replaceState()` pour les modifications qui ne doivent pas constituer des étapes distinctes, comme un filtre en cours de saisie, et gérez `popstate`. Next.js recommande pour la même raison de conserver l'état de routage dans l'URL : c'est ce qui permet de créer un lien vers une vue et de la partager.

Un lien profond rencontre un serveur qui ne connaît qu'un fichier. Dans une SPA stricte, un fichier HTML dessert toutes les routes, mais la barre d'adresse affiche `/settings`. C'est ce que le navigateur demande au rechargement ou à l'ouverture d'un lien partagé. Un serveur qui ne possède que `index.html` répond 404. Configurez-le pour répondre à chaque chemin de l'application avec le document, ou générez un fichier par route. L'export statique de Next.js suit la seconde voie : son dossier `out` contient des fichiers comme `index.html`, `404.html` et `blog/post-1.html`. Son guide propose une règle nginx, `try_files $uri $uri.html $uri/ =404;`, qui associe une URL lisible au fichier correspondant.

Le focus reste à l'emplacement de l'ancienne vue. `pushState()` ne modifie que l'historique de session et l'URL. Une fois la vue remplacée par un script, le focus clavier reste sur le lien activé, ou se perd si l'élément qui le portait a été supprimé. Rien n'annonce la nouvelle vue aux technologies d'assistance à moins que la page le fasse. Définissez le titre et placez le focus sur le titre de la nouvelle vue, comme dans l'exemple. La Navigation API prend davantage de ces opérations en charge. Une navigation gérée par `intercept()` réinitialise le focus par défaut sur le premier élément portant `autofocus`, ou sinon sur le corps du document. Une fois le gestionnaire terminé, le navigateur signale l'achèvement à l'utilisateur, ce qui inclut les technologies d'assistance selon la norme. Une application qui place elle-même le focus désactive ce comportement avec `focusReset: "manual"`.

Tout est payé au départ. Une SPA stricte doit souvent charger beaucoup de JavaScript avant la moindre interaction, et ses premières requêtes de données ne commencent qu'à l'exécution de ce code. Découpez le paquet par route, pour que le visiteur ne télécharge que ce dont la vue actuelle a besoin, et lancez les premières requêtes sur le serveur. Le modèle proposé par Next.js récupère les données dans un Server Component sans les attendre, puis passe la promesse à

un Client Component, qui en extrait la valeur avec `use()`. La requête commençant avant le rendu du reste de l'application, explique le guide, la réponse peut immédiatement commencer à être transmise en flux. Le client évite ainsi une cascade de requêtes attendant chacune la précédente.

Pour aller plus loin

- Construire des applications monopages avec Next.js — définition d'une SPA stricte, coûts et modèles utilisés par Next.js pour conserver la navigation côté client tout en déplaçant le travail sur le serveur.
- HTML Standard : interface History — `pushState()`, `replaceState()` et ajout ou remplacement d'entrées dans l'historique de session.
- HTML Standard : Navigation API — événement `navigate`, méthode `intercept()` et gestion associée du focus et du défilement.
- Créer un export statique d'une application Next.js — un fichier HTML par route, les limites d'un export statique et la desserte d'URL lisibles avec nginx.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'une application monopage ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/single-page-application/