Engineer

Односторінковий застосунок

вебзастосунок, що змінює одну сторінку засобами JavaScript

Що це таке

Односторінковий застосунок (англійською single-page application, скорочено SPA) — це вебзастосунок, який завантажує один HTML-документ, а далі змінює цей документ засобами JavaScript, замість того щоб просити в сервера нову сторінку. Клік на посилання всередині нього не змушує браузер залишити сторінку. Скрипт перехоплює клік, отримує дані, потрібні для наступного екрана, переписує частину DOM і оновлює адресний рядок, тож URL і далі показує, де перебуває читач.

Жоден стандарт не володіє цим терміном, і ті, хто робить інструменти, це визнають. Next.js починає свій посібник на цю тему словами «The definition of a SPA varies.», а потім задає власне визначення, яке називає strict SPA. У нього дві ознаки. Увесь застосунок віддається одним HTML-файлом, а кожен маршрут, кожен перехід між сторінками і кожне отримання даних обробляє JavaScript у браузері. І повних перезавантажень сторінки немає: замість того щоб запитувати новий документ для кожного маршруту, JavaScript на клієнті змінює DOM поточної сторінки й отримує дані тоді, коли вони потрібні. Ця стаття користується саме цим визначенням, бо воно найпряміше з наявних і називає властивість, від якої залежить усе інше, — документ, який ніколи не замінюється.

Браузерні частини стандартні. Інтерфейс History зі стандарту HTML дає сторінці `history.pushState()`, що додає новий запис в історію сеансу й змінює URL, нічого не завантажуючи, і `history.replaceState()`, що натомість переписує поточний запис. Коли читач переходить між цими записами кнопками «Назад» і «Вперед», браузер надсилає вікну подію `popstate` і залишає перемальовування самій сторінці. Новіший Navigation API, який стандарт подає як наступника класичних API location і history, заходить з

іншого боку: подія `navigate` спрацьовує, коли навігація починається, а її метод `intercept()` перетворює цю навігацію на навігацію в межах того самого документа, яку сторінка обробляє сама.

Протилежність — звичайний багатосторінковий сайт, де кожне посилання просить у сервера свіжий документ, а браузер викидає старий. Чи HTML сторінки створює один раз генератор статичних сайтів, чи серверний рендеринг на кожен запит — окреме питання. Воно про те, як HTML з'являється, а не про те, що відбувається, коли він уже прийшов. Next.js свідомо розмиває цю межу: його статичний експорт записує HTML-файл для кожного маршруту, а переходи між ними однаково відбуваються на клієнті.

Застосуйте поняття до рішення

Застосуйте «Що таке односторінковий застосунок?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Інтерфейси, що швидко завантажуються, працюють на телефоні й поводяться однаково в усіх браузерах ваших клієнтів.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Керувала full-stack-розробкою GIS-карт — full-stack-картографія — від бази даних до браузера

Керувала full-stack-розробкою GIS-карт Зв'язатися з нами →

Чому це важливо

Документ ніколи не замінюється, тож усе, що в ньому є, переживає клік: наполовину заповнена форма, відео, що грає, позиція прокрутки бічної панелі, уже отримані дані. Мережею треба передати лише дані для наступного екрана, а не цілу нову сторінку з шапкою, стилями й скриптами. Зміна екрана може звестися до того, щоб переписати одну ділянку сторінки.

Next.js називає ціну на одному подиху з визначенням. Strict SPA часто мусить завантажити великий обсяг JavaScript, перш ніж сторінка стане інтерактивною, а каскадами запитів на клієнті, коли один запит може початися лише після того, як повернувся інший, важко керувати. Платить і все, що читає сторінку без виконання скриптів. Коли за кожним маршрутом стоїть один HTML-файл, краулер, який не виконує JavaScript, отримує той самий майже порожній документ, хоч би який URL він запитав. Next.js каже те саме про власні компоненти, що працюють лише в браузері: краулери, які не виконують JavaScript, бачать fallback замість компонента.

Відповідь Next.js — зберегти навігацію на клієнті, а решту перенести. Next.js автоматично розбиває JavaScript-бандли й генерує кілька HTML-точок входу для різних маршрутів, тож відвідувач не завантажує коду, який маршруту не потрібен. Компонент `next/link` заздалегідь завантажує маршрути, що дає швидкі переходи між сторінками, як у strict SPA, і водночас тримає стан маршрутизації застосунку в URL, де на нього можна дати посилання й поділитися ним. Статичний експорт іде ще далі й записує повністю відрендерену сторінку для кожного маршруту, тож відвідувачі отримують вміст, не чекаючи клієнтського бандла, а переходи між сторінками на клієнті лишаються такими самими швидкими, як у strict SPA.

І вибір не робиться раз і назавжди. Next.js описує проєкт, який починається як статичний сайт або навіть як strict SPA, що повністю рендериться на клієнті, і в міру зростання додає серверні можливості, а наявні SPA, за словами Next.js, можна перевести на нього без повного переписування.

Як це виглядає на практиці

Маршрутизатор нижче написано для цієї статті, і це найменша робоча версія ідеї. Він припускає, що на сторінці є елемент `<main>` і що сервер на кожен шлях застосунку відповідає тим самим HTML-документом. Два екрани тут замінюють те, що застосунок показує насправді.

```
// 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);
```

Обробник кліків перехоплює звичайні кліки лівою кнопкою на посиланнях того самого походження (origin) і пропускає кліки з клавішами-модифікаторами, тож посилання, як і раніше, можна відкрити в новій вкладці. `preventDefault()` зупиняє власну навігацію браузера, а `pushState()` додає в історію сеансу новий запис з URL посилання. Середній аргумент існує лише з історичних причин, тому тут він порожній рядок. `show()` замінює екран, задає заголовок документа й переводить фокус на новий елемент `h1`, який завдяки `tabIndex` зі значенням `-1` може отримати фокус зі скрипту, але не потрапляє в послідовність переходів клавішею Tab. Слухач `popstate` перемальовує відповідний екран, коли читач іде назад або вперед, а останній рядок малює екран для того URL, з яким сторінку відкрили, і це працює лише тоді, коли сервер віддає для `/settings` той самий документ, що й для `/`. Справжній застосунок ще й показував би власний екран для невідомих адрес, а не тихо повертався б на стартову сторінку, і не чіпав би посилань на файли та на інші частини сайту.

У поточній роботі засновниці в NextMariner фронтенд на Next.js 16 показує ціну цього підходу на справжньому продукті. Сторінка портфоліо про цей фронтенд описує дашборд для автентифікованих користувачів, який мав бути відчутно швидким і водночас по-справжньому інтерактивним, і розповідає, що стається, коли всі дані отримують на клієнті: перше завантаження затягується, і з’являється патерн N+1, коли кожен компонент прокидається й надсилає власний запит, доки одна сторінка не перетвориться на каскад мережевих звернень. Це й є той каскад запитів на клієнті, від якого застерігає посібник Next.js, тільки на практиці. Рішення зберегло клієнтську інтерактивність там, де вона важлива, і перенесло перше отримання даних на сервер. У дашборді й на великих сторінках-каталогах фабрика `createPrefetchedServerPage` передає кожну сторінку клієнту вже з даними на місці. Про решту сторінка портфоліо говорить відверто: інші сторінки для автентифікованих користувачів досі рендеряться на клієнті й чекають своєї черги, тож ідеться про перенесення, яке працює і вже знає, яку сторінку брати наступною, а не про завершений перехід.

Де виникають проблеми

Кнопка «Назад» виводить із застосунку. Якщо скрипт змінює екрани, не викликаючи `pushState()`, в історії сеансу для них немає записів. «Назад» проскакує всі екрани, які бачив читач, і зовсім виводить із застосунку, а перезавантаження веде на той URL, з яким застосунок відкрили першим. Варто додавати запис для кожного екрана, який читач назвав би сторінкою, використовувати `replaceState()` для змін, що не мають ставати окремими кроками, як-от фільтр, який саме набирають, і обробляти `popstate`. Next.js обстоює стан маршрутизації в URL з тієї самої причини: саме це дає змогу послатися на екран і поділитися ним.

Пряме посилання натрапляє на сервер, що знає один файл. У strict SPA один HTML-файл обслуговує кожен маршрут, але в адресному рядку видно `/settings`, і саме цю адресу браузер запитує під час перезавантаження або коли хтось відкриває надіслане посилання. Сервер, у якого є тільки `index.html`, відповідає кодом 404. Вихід — або налаштувати сервер так, щоб на кожен шлях застосунку він віддавав документ, або згенерувати файл для кожного маршруту. Статичний експорт Next.js робить друге: його тека `out` містить файли на кшталт `index.html`, `404.html` і `blog/post-1.html`, а посібник показує правило nginx `try_files $uri $uri.html $uri/ =404;`, яке зіставляє чистий URL з відповідним файлом.

Фокус лишається там, де був старий екран. `pushState()` змінює історію сеансу та URL і більше нічого. Після того як скрипт замінив екран, фокус клавіатури лишається на посиланні, по якому клікнули, або губиться, якщо елемент, що його тримав, видалено, і допоміжні технології дізнаються про новий екран лише тоді, коли сторінка сама їм про це повідомить. Варто задавати заголовок документа і переводити фокус на заголовок нового екрана, як це робить приклад. Navigation API бере більше цієї роботи на себе. Навігація, оброблена через `intercept()`, за замовчуванням скидає фокус на перший елемент з `autofocus`, а якщо такого немає, то на body, і коли обробник завершиться, браузер повідомляє користувачеві про завершення навігації, що, за словами стандарту, охоплює й допоміжні технології. Застосунок, який сам розставляє фокус, відмовляється від цього через `focusReset: "manual"`.

Усе оплачується наперед. Strict SPA часто мусить завантажити великий обсяг JavaScript, перш ніж щось стане інтерактивним, а перші запити даних стартують лише тоді, коли цей код запуститься. Варто розбивати бандл за маршрутами, щоб відвідувач завантажував лише те, що потрібно поточному

екрану, і починати перші запити на сервері. Патерн Next.js для цього отримує дані в Server Component, не чекаючи на результат, і передає проміс у Client Component, який розгортає його через `use()`. Якщо запит стартує ще до того, як відрендериться решта застосунку, відповідь, за словами посібника, може відразу йти потоком, і каскадів запитів на клієнті вдається уникнути.

Що почитати далі

- How to build single-page applications with Next.js — визначення strict SPA, його ціна і патерни, за допомогою яких Next.js зберігає навігацію на клієнті, переносячи роботу на сервер.
- HTML Standard: The History interface — `pushState()`, `replaceState()` і те, як записи історії сеансу додаються та замінюються.
- HTML Standard: Navigation API — подія `navigate`, `intercept()` і обробка фокусу та прокрутки, що йде разом із ними.
- How to create a static export of your Next.js application — один HTML-файл на маршрут, чого не вміє статичний експорт і як віддавати чисті URL через nginx.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке односторінковий застосунок?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/single-page-application/