Engineer

aplicación de una sola página

App web que carga una página y la cambia con JavaScript

Qué es

Una aplicación de una sola página, también llamada app de una sola página y normalmente abreviada como SPA, es una aplicación web que carga un documento HTML y después lo modifica con JavaScript en lugar de pedir una página nueva al servidor. Al pulsar un enlace interno, no se descarta el documento abierto. Un script intercepta el clic, obtiene los datos que necesita la siguiente vista, reescribe parte del DOM y actualiza la barra de direcciones, de modo que la URL sigue indicando dónde se encuentra quien la consulta.

Ninguna norma define el término, y quienes crean las herramientas lo reconocen. Next.js empieza su guía sobre el tema con «La definición de una SPA varía» y establece una para su propio uso, que denomina SPA estricta. Tiene dos rasgos. Un solo archivo HTML sirve a toda la aplicación, y JavaScript en el navegador se encarga de cada ruta, cada transición entre páginas y cada solicitud de datos. Además, la página nunca se recarga por completo: cuando cambia la ruta, los scripts del cliente reescriben el DOM de la página que ya está abierta y cargan los datos cuando hacen falta, en lugar de pedir un documento nuevo al servidor. Este artículo utiliza esa definición porque es la más directa disponible e identifica la propiedad de la que depende todo lo demás: un documento que nunca se sustituye.

Los componentes del navegador sí están normalizados. La interfaz History del estándar HTML proporciona a una página `history.pushState()`, que añade una entrada al historial de la sesión y establece la URL sin cargar nada, y `history.replaceState()`, que reescribe la entrada actual. Cuando se retrocede o avanza entre esas entradas, el navegador envía un evento `popstate` a la ventana y deja que la página se encargue de redibujarse. La API Navigation, más reciente y presentada por el estándar como sucesora de las API clásicas de ubicación e

historial, actúa desde el otro extremo: se emite un evento `navigate` cuando empieza una navegación, y su método `intercept()` la convierte en una navegación dentro del mismo documento que la propia página gestiona.

Lo opuesto es el sitio web convencional de varias páginas, donde cada enlace pide un documento nuevo al servidor y el navegador descarta el anterior. Que el HTML de una página se genere una vez mediante un generador de sitios estáticos o en cada solicitud mediante renderizado en el servidor es una cuestión distinta: trata de cómo se produce el HTML, no de lo que sucede después de recibirlo. Next.js difumina esa frontera deliberadamente: su exportación estática escribe un archivo HTML para cada ruta, pero la navegación entre ellas sigue ocurriendo en el cliente.

Utilice el concepto en una decisión

Aplique «¿Qué es una aplicación de una sola página?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Interfaces que cargan rápido, funcionan en el teléfono y se comportan igual en los navegadores que utilizan sus clientes.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Dirección del desarrollo integral de mapas GIS — cartografía integral, de la base de datos al navegador

Dirección del desarrollo integral de mapas GIS Póngase en contacto →

Por qué importa

Como el documento nunca se sustituye, todo lo que contiene sobrevive a un clic: un formulario a medio rellenar, un vídeo en reproducción, la posición de desplazamiento de un panel lateral o datos ya obtenidos. Solo tienen que cruzar la red los datos que necesita la siguiente vista, no una página completamente nueva con su cabecera, estilos y scripts. Un cambio de vista puede consistir únicamente en reescribir una región de la página.

Next.js expone los costes junto con la definición. Una SPA estricta a menudo tiene que cargar una gran cantidad de JavaScript antes de que la página sea interactiva, y las cascadas de solicitudes de datos en el cliente, donde una solicitud solo puede empezar cuando otra ha terminado, son difíciles de gestionar. También hay un coste para cualquier sistema que lea la página sin ejecutar scripts. Si todas las rutas utilizan un solo archivo HTML, un rastreador que no ejecute JavaScript recibe el mismo documento casi vacío sea cual sea la

URL que solicite. Next.js señala lo mismo sobre sus componentes que solo se ejecutan en el navegador: cualquier sistema que indexe la página sin ejecutar scripts recibe el contenido alternativo, no el componente.

Su respuesta consiste en mantener la navegación en el cliente y trasladar el resto. Next.js divide automáticamente los paquetes de JavaScript y genera múltiples puntos de entrada HTML a distintas rutas, para que quien visite el sitio no cargue código que una ruta no necesita. Su componente `next/link` precarga rutas, lo que hace que los cambios de ruta parezcan tan inmediatos como en una SPA renderizada por completo en el cliente, mientras mantiene el estado del enrutamiento de la aplicación en la URL, que se puede enlazar y compartir. Una exportación estática va más allá y escribe una página completamente renderizada para cada ruta, de modo que las visitas reciben contenido sin esperar al paquete del cliente, mientras que las transiciones entre páginas en el cliente siguen siendo tan rápidas como en una SPA estricta.

Tampoco es una decisión definitiva. Next.js describe un proyecto que empieza como sitio estático, o incluso como SPA estricta renderizada íntegramente en el cliente, y añade funciones de servidor a medida que crece. También indica que una SPA existente puede migrar sin reescribirse desde cero.

Cómo se aplica en la práctica

El enrutador siguiente, escrito para este artículo, es la versión funcional más pequeña de la idea. Supone que la página tiene un elemento `<main>` y que el servidor responde a todas las rutas de la aplicación con el mismo documento HTML. Las dos vistas representan lo que la aplicación muestre en realidad.

```
// 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);
```

El manejador de clics toma el control de los clics normales con el botón izquierdo sobre enlaces al mismo origen y deja pasar los clics modificados, por lo que sigue siendo posible abrir un enlace en otra pestaña. `preventDefault()` detiene la navegación propia del navegador, y `pushState()` registra una nueva entrada en el historial de la sesión con la URL del enlace; su argumento central solo existe por razones históricas, de ahí que sea una cadena vacía. `show()` sustituye la vista, establece el título del documento y mueve el foco al nuevo encabezado, al que un `tabIndex` de `-1` permite recibir el foco desde un script sin incluirlo en el orden de tabulación. El escuchador de `popstate` redibuja la vista correspondiente al retroceder o avanzar, y la última línea dibuja la vista de la URL con la que se abrió la página. Esto solo funciona si el servidor envía el mismo documento para `/settings` que para `/`. Una aplicación real también mostraría su propia vista de página no encontrada en lugar de volver en silencio a la página inicial, y dejaría intactos los enlaces a archivos y a otras partes del sitio.

En el trabajo actual de la fundadora en NextMariner, el frontend de Next.js 16 muestra estos costes en un producto real. Su ficha en el portafolio describe un panel con autenticación que debía resultar rápido y mantener una interactividad real, y señala qué ocurre cuando todos los datos se obtienen en el cliente: la primera carga se vuelve lenta y aparece el patrón N+1, en el que cada componente se activa y envía su propia solicitud hasta que una sola página se convierte en una cascada de intercambios de ida y vuelta. Es un caso práctico de la cascada de solicitudes de datos en el cliente de la que advierte la guía de Next.js. La solución mantuvo la interactividad en el cliente donde importaba y trasladó la primera obtención de datos al servidor. En el panel y en los grandes listados del directorio, una factoría llamada `createPrefetchedServerPage` entrega cada página al cliente con los datos ya preparados. La ficha es clara sobre lo que

queda: las demás páginas con autenticación aún se renderizan en el cliente y esperan su turno. Describe, pues, una migración que funciona y cuya siguiente página por migrar está clara, pero que todavía no ha terminado.

Dónde surgen los problemas

El botón Atrás sale de la aplicación. Si un script cambia las vistas sin llamar a `pushState()`, el historial de la sesión no tiene entradas para ellas. Atrás se salta todas las vistas que se han mostrado y sale por completo de la aplicación, y una recarga vuelve a la URL con la que se abrió inicialmente. Añada una entrada para cada vista que se consideraría una página, utilice `replaceState()` para los cambios que no deban convertirse en pasos propios, como un filtro mientras se escribe, y gestione `popstate`. Next.js defiende mantener el estado del enrutamiento en la URL por la misma razón: permite compartir una vista y enlazarla.

Un enlace profundo encuentra un servidor que solo conoce un archivo. En una SPA estricta, un archivo HTML sirve a todas las rutas, pero la barra de direcciones muestra `/settings`, y eso es lo que pide el navegador al recargar o al abrir un enlace compartido. Un servidor que solo tiene `index.html` responde con un 404. Configure el servidor para responder a cada ruta de la aplicación con ese documento, o genere un archivo por ruta. La exportación estática de Next.js hace lo segundo: su carpeta `out` contiene archivos como `index.html`, `404.html` y `blog/post-1.html`, y su guía muestra una regla de nginx, `try_files $uri $uri.html $uri/ =404;`, que asocia una URL limpia con el archivo correspondiente.

El foco permanece donde estaba la vista anterior. `pushState()` cambia el historial de la sesión y la URL, y nada más. Después de que un script sustituya la vista, el foco del teclado permanece en el enlace pulsado, o se pierde si se ha eliminado el elemento que lo tenía, y nada avisa a la tecnología de apoyo de que ha llegado una vista nueva si no lo hace la propia página. Establezca el título y mueva el foco al encabezado de la nueva vista, como hace el ejemplo. La API Navigation asume una parte mayor de esta tarea. Una navegación gestionada con `intercept()` restablece el foco de forma predeterminada en el primer elemento con `autofocus` o, si no lo hay, en el cuerpo del documento. Una vez terminado el manejador, el navegador comunica al usuario que ha finalizado, lo que, según el estándar, incluye a la tecnología de apoyo. Una aplicación que sitúe el foco por su cuenta desactiva ese comportamiento mediante `focusReset: "manual"`.

Todo se paga de antemano. Una SPA estricta a menudo tiene que cargar una gran cantidad de JavaScript antes de que nada sea interactivo, y sus primeras solicitudes de datos solo empiezan cuando se ejecuta ese código. Divida el paquete por rutas, para que cada visita descargue solo lo que necesita la vista actual, e inicie las primeras solicitudes en el servidor. El patrón de Next.js para esto inicia la obtención de datos en un Server Component sin esperar a que termine y pasa la promesa a un Client Component, que extrae su valor con `use()`. Como la solicitud empieza antes de renderizar el resto de la aplicación, explica la guía, la respuesta puede empezar a enviarse por partes de inmediato, y el cliente evita una cascada de solicitudes en la que cada una espera a la anterior.

Lecturas adicionales

- Cómo crear aplicaciones de una sola página con Next.js — la definición de SPA estricta, sus costes y los patrones que utiliza Next.js para mantener la navegación en el cliente y trasladar trabajo al servidor.
- Estándar HTML: la interfaz History — `pushState()`, `replaceState()` y cómo se añaden y sustituyen las entradas del historial de la sesión.
- Estándar HTML: API Navigation — el evento `navigate`, `intercept()` y la gestión del foco y el desplazamiento que los acompaña.
- Cómo crear una exportación estática de su aplicación Next.js — un archivo HTML por ruta, lo que no puede hacer una exportación estática y cómo servir URL limpias con nginx.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es una aplicación de una sola página?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

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

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/single-page-application/