Engineer

Mejora progresiva

crear páginas que funcionan solo con HTML y luego mejorarlas

Qué es

La mejora progresiva es una forma de construir páginas web por capas, empezando por la que siempre funciona. La base es HTML que incluye todo el contenido y completa por sí solo todas las tareas: los enlaces llevan a un destino, los formularios se envían a un servidor y la página puede leerse y utilizarse sin cargar nada más. Encima se añaden hojas de estilo para organizar la página y darle un aspecto agradable. Los scripts llegan al final para que su uso sea más rápido o fluido. Cada capa mejora la que tiene debajo, y la página no depende de ninguna capa por encima de HTML para cumplir su función. El Service Manual de GOV.UK resume la idea como hacer que una página funcione primero con HTML.

El nombre procede de una charla de Steven Champeon y Nick Finck, *Diseño web inclusivo para el futuro con mejora progresiva*, que, según su artículo, se presentó el 11 de marzo de 2003 en SXSW, en Austin, Texas. El artículo escribe el nombre con mayúsculas, Progressive Enhancement, y lo abrevia como PE. Contrapone la idea a la degradación controlada, la costumbre anterior de diseñar para el navegador más capaz y confiar en que los demás pudieran seguir accediendo al contenido. PE, tal como la describe el artículo, parte de un documento que pueden utilizar los dispositivos menos capaces y añade después la presentación como lógica separada, para que los dispositivos básicos no carguen con lo que reciben los navegadores modernos.

La diferencia entre ambas es la dirección del trabajo. La degradación controlada parte de la experiencia completa y pregunta qué queda cuando faltan partes. La mejora progresiva parte de lo que siempre está disponible y pregunta qué se puede añadir. En un navegador actual con una buena conexión, ambas pueden llegar a la misma página. Se diferencian cuando algo falla: una página construida desde la base sigue teniendo esa base, mientras que una construida desde arriba conserva lo que sus autores se acordaron de proporcionar.

Es una forma de trabajar, no una biblioteca ni una funcionalidad. Nada en HTML, CSS o JavaScript se llama mejora progresiva, pero los estándares permiten aplicarla. Un navegador omite una declaración CSS que no entiende e ignora un script cuyo tipo no reconoce, y la regla `@supports` permite que una hoja de estilo compruebe si una funcionalidad está disponible antes de depender de ella.

Utilice el concepto en una decisión

Aplique «¿Qué es la mejora progresiva?» 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

Las capas fallan de distintas maneras, y el Service Manual de GOV.UK explica cómo. La capa CSS es razonablemente tolerante: un navegador omite una declaración que no reconoce y continúa, aunque un selector que desconoce hace que descarte toda la regla. La capa JavaScript no es tolerante en absoluto. Cuando un script usa una sintaxis o una interfaz que el navegador no tiene, el resto de ese script no se ejecuta. Por tanto, una página que necesita su script para realizar la tarea principal solo es fiable si lo son todos los pasos entre el servidor y quien la lee.

Hay muchos de esos pasos. La misma página enumera lo que puede impedir que CSS o JavaScript se carguen o ejecuten: un error temporal de red, un bloqueador de anuncios, una red de distribución de contenido caída, un fallo de DNS, un error en una actualización del navegador o un cortafuegos corporativo o personal. Nada de eso tiene que ver con un navegador antiguo, y quien lee no puede saber cuál ha sido la causa. Ve un botón que no hace nada. El manual no ofrece cifras sobre la frecuencia de cada caso; su argumento se basa en la lista, no en un porcentaje.

Algunas personas también desactivan funciones deliberadamente. El manual dice que ciertos usuarios desactivan funciones de sus navegadores por decisión propia y que debe respetarse esa elección. El HTML Standard afirma que muchos usuarios desactivan los scripts y, por esa razón y porque las funcionalidades declarativas suelen ser más fáciles de mantener, anima a los autores a usarlas en

lugar de scripts cuando sea posible. Su ejemplo es el elemento `details`, que muestra y oculta una sección de una página sin una línea de JavaScript. El estándar también pide a los autores que sus aplicaciones se degraden de forma controlada cuando no haya soporte para scripts.

Para los servicios del Gobierno británico, el manual convierte esto en una regla: deberían funcionar solo con HTML, y JavaScript, cuando se use, debería limitarse a mejorar HTML y CSS para que las personas puedan seguir utilizando el servicio si el script falla. Esa regla está escrita para los servicios de un gobierno, no para toda la web, pero su razonamiento se aplica a cualquier sitio cuyos autores no puedan elegir quién lo lee.

El equipo también obtiene una ventaja. Cuando una mejora falla, quien usa la página pierde la mejora, pero puede seguir realizando la tarea, de modo que un fallo de un script es una molestia en lugar de una interrupción del servicio. Puede publicarse una nueva función para los navegadores que la admiten sin un plan para los que no, porque estos conservan la página que ya tenían.

Cómo se aplica en la práctica

El HTML Standard ofrece como ejemplo de funcionamiento adecuado sin scripts una tabla cuyos encabezados de columna cambian su ordenación, y es una prueba razonable del enfoque. La base es una tabla cuyos encabezados son enlaces corrientes. Un clic pide al servidor la misma tabla ordenada por esa columna, y el servidor la devuelve en ese orden. Esto funciona en cualquier navegador, con o sin scripts.

```
<table id="orders">
  <thead>
    <tr>
      <th><a href="?sort=date">Date</a></th>
      <th><a href="?sort=total">Total</a></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>2026-10-20</td>
      <td>120</td>
    </tr>
    <tr>
      <td>2026-10-24</td>
      <td>45</td>
    </tr>
```

```
  </tbody>
</table>
<script type="module" src="sort.mjs"></script>
```

El script es la mejora. Encuentra esos mismos enlaces, cancela la navegación y reordena las filas en la página, lo que evita un viaje de ida y vuelta al servidor.

```
const table = document.querySelector("#orders");
const body = table.tBodies[0];
const collator = new Intl.Collator("en", { numeric: true });

function sortBy(column) {
  const text = (row) => row.cells[column].textContent;
  const rows = [...body.rows];
  rows.sort((a, b) => collator.compare(text(a), text(b)));
  body.append(...rows);
}

for (const link of table.tHead.querySelectorAll("a")) {
  link.addEventListener("click", (event) => {
    event.preventDefault();
    sortBy(link.closest("th").cellIndex);
  });
}
```

Si se produce un fallo antes de asociar un manejador de eventos a un enlace, ese enlace sigue funcionando. Si `sort.mjs` nunca llega, los enlaces hacen lo mismo que antes. Si el script lanza una excepción a medio camino, los enlaces de los encabezados a los que no llegó a asociar un manejador siguen recurriendo al servidor. Un navegador que no admite scripts de módulo trata `type="module"` como un tipo de script desconocido y, como dice el HTML Standard, lo ignora. Un script de módulo sin `async` se ejecuta cuando la página ha terminado de analizarse, por lo que la tabla ya existe cuando el script la busca. Este ejemplo no se recupera si un manejador de clic ya asociado lanza una excepción después de que `preventDefault()` haya cancelado la navegación.

El coste recae en el servidor: la base necesita un punto de acceso capaz de ordenar, y las dos formas de ordenación tienen que coincidir. Ese es el coste habitual del enfoque, y se paga una vez. La ventaja buscada es que la tarea principal de la página pueda seguir realizándose recurriendo al servidor; un manejador asociado debe conservar esa vía alternativa si falla su propio procesamiento.

Este sitio hace explícita su base. Su declaración de accesibilidad describe páginas completas en HTML, CSS y SVG sin JavaScript ejecutable. El código de búsqueda se conserva, pero está desactivado, de modo que no se depende de un filtro dentro de la página. La declaración también señala una limitación pendiente: las pequeñas etiquetas que aparecen al pasar el puntero sobre un icono o al enfocarlo no pueden cerrarse con una tecla, lo que exigiría código de navegador que el sitio no distribuye. Por eso solo cumple parcialmente el criterio 1.4.13, Contenido señalado con el puntero o que tiene el foco.

Dónde surgen los problemas

Un bloque `noscript` sustituye a una base funcional. El elemento `noscript` muestra su contenido solo cuando los scripts están desactivados, y el HTML Standard es tajante: «El elemento noscript es una herramienta burda». Los scripts pueden estar activados y el script de la página aun así fallar, en cuyo caso quien la lee no obtiene ni la página con scripts ni la alternativa. Parta de una página que funcione sin el script y deje que este la modifique, como hace el propio ejemplo del estándar cuando su script añade un resultado en vivo a un formulario y elimina su botón de envío.

Una funcionalidad no admitida detiene todo el script. Como la capa JavaScript no tolera fallos, una sola construcción sintáctica nueva en un paquete puede inutilizar todas sus mejoras, incluidas las que habrían funcionado. Mantenga la base independiente del script para que un fallo solo cueste la mejora. Cargue el código moderno con `type="module"`, que los navegadores antiguos omiten, y, si estos necesitan alguna versión mejorada, proporcióneles un paquete clásico separado marcado con `nomodule`, que los navegadores que admiten módulos no ejecutan.

Un selector nuevo inutiliza reglas válidas. Cuando un navegador encuentra un selector que no conoce, descarta toda la regla, incluidas sus declaraciones. Por tanto, enumerar una pseudoclase reciente junto a selectores que todos los navegadores entienden puede eliminar estilos en navegadores que los habrían aplicado. Dé una regla propia a un selector que algunos navegadores podrían no admitir.

Los estilos de reserva se filtran al diseño mejorado. Los anchos escritos para navegadores sin un sistema de disposición más reciente pueden interferir con ese sistema en los navegadores que sí lo tienen. CSS Conditional Rules Level 3 muestra la solución con una disposición flex y un bloque `@supports not (display:`

`flex)` para la alternativa, y señala que las declaraciones de ancho podrían perjudicar la disposición flex, por lo que deben aparecer solo en los estilos para navegadores que no la admiten.

Las imágenes se eligen mediante un script. El HTML Standard señala que, si una página no especifica ninguna imagen en su marcado y deja la elección a un script, quienes tengan los scripts desactivados no descargarán ninguna imagen; y, si un script sustituye una imagen ya especificada en el marcado, el navegador puede iniciar dos descargas. Declare las candidatas en HTML con `srcset` y `sizes`, o con `picture`, y deje que el navegador elija.

Comportamiento sin una base que lo sostenga. La lista de estrategias de Champeon y Finck pide una alternativa para todo comportamiento: un `href` real detrás de cada manejador de clic y una comprobación en el servidor detrás de cada comprobación en el navegador. Un control que solo existe como manejador de clic o un formulario que solo se valida en el navegador carecen por completo de base. Use primero enlaces reales y envíos reales de formularios, valide siempre en el servidor y trate la validación en el navegador como una comodidad añadida.

Tratarlo como una cuestión de navegadores antiguos. Un equipo que cree que la base está destinada a los navegadores antiguos la elimina cuando estos desaparecen. La lista de causas de GOV.UK habla de redes, bloqueadores, cortafuegos y actualizaciones defectuosas, que también afectan a los navegadores actuales. Mantenga la base como una vía utilizada y probada: de vez en cuando cargue la página con los scripts bloqueados y complete su tarea principal de principio a fin.

Lecturas adicionales

- Uso de la mejora progresiva — Orientaciones del Service Manual de GOV.UK, actualizadas por última vez el 27 de septiembre de 2024: las tres capas, cómo falla cada una y por qué los servicios gubernamentales deberían funcionar solo con HTML.
- HTML Standard: scripts — El elemento `script` con `type="module"` y `nomodule`, y el elemento `noscript` con el consejo de no depender de él.
- CSS Conditional Rules Module Level 3 — El borrador de recomendación candidata del W3C del 15 de agosto de 2024 que define `@supports` con sus operadores `not`, `and` y `or`.

- HTML Standard: imágenes — `srcset` , `sizes` y `picture` , y por qué elegir una imagen con un script deja sin imagen a quienes tienen los scripts desactivados.

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 la mejora progresiva?» 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/progressive-enhancement/