Engineer

Прогресивне покращення

побудова сторінки шарами, де основою є чистий HTML

Що це таке

Прогресивне покращення — це спосіб будувати вебсторінки шарами, починаючи з того шару, який працює завжди. Основа — HTML, що містить увесь вміст і сам по собі виконує кожне завдання: посилання кудись ведуть, форми надсилаються на сервер, а сторінку можна читати й використовувати, навіть якщо більше нічого не завантажилося. Таблиці стилів лягають згори, щоб розкласти сторінку й зробити її приємною на вигляд. Скрипти додаються останніми й роблять її швидшою або зручнішою в користуванні. Кожен шар покращує той, що під ним, і сторінка не залежить від жодного шару над HTML, щоб виконувати свою роботу. Посібник GOV.UK Service Manual зводить ідею до того, щоб спершу змусити сторінку працювати з HTML. Англійською це називають progressive enhancement.

Назва походить від доповіді, яку прочитали Steven Champeon і Nick Finck, *Inclusive Web Design For the Future with Progressive Enhancement*. Згідно з її письмовою версією, доповідь прозвучала 11 березня 2003 року на SXSW в Остіні, штат Техас. Текст пише Progressive Enhancement з великих літер і скорочує до PE. Він протиставляє ідею плавній деградації (graceful degradation) — давнішій звичці будувати для найспроможнішого браузера й сподіватися, що решта все ж дістанеться до вмісту. PE, як описує текст, починається з документа, яким можуть користуватися найменш спроможні пристрої, а потім додає оформлення як окрему логіку, щоб прості пристрої не обтяжувало те, що отримують сучасні браузери.

Відмінність між ними — у напрямку роботи. Плавна деградація відштовхується від повного досвіду й питає, що залишиться, коли частини його бракуватиме. Прогресивне покращення відштовхується від того, що є завжди, і питає, що можна додати. У новому браузері на доброму з’єднанні обидва підходи можуть привести до тієї самої сторінки. Різниця

проявляється, коли щось ламається: сторінка, побудована знизу, досі має свою основу, а сторінка, побудована згори, має лише те, що автори не забули передбачити.

Це спосіб роботи, а не бібліотека чи функція. Ніщо в HTML, CSS або JavaScript не називається прогресивним покращенням, але стандарти залишають для нього місце. Браузер пропускає CSS-оголошення, якого не розуміє, і ігнорує скрипт, тип якого не знає, а правило `@supports` дає таблиці стилів змогу перевірити, чи є функція, перш ніж на неї покладатися.

Застосуйте поняття до рішення

Застосуйте «Що таке прогресивне покращення?» 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

Шари ламаються по-різному, і GOV.UK Service Manual пояснює, як саме. Шар CSS досить поблажливий: браузер пропускає оголошення, якого не розпізнає, і працює далі, але незнайомий селектор змушує його відкинути все правило. Шар JavaScript не прощає нічого. Коли скрипт використовує синтаксис або інтерфейс, якого браузер не має, решта цього скрипту не виконується. Тож сторінка, якій для головного завдання потрібен скрипт, надійна лише настільки, наскільки надійна кожна ланка між сервером і читачем.

А ланок чимало. Та сама сторінка перелічує причини, чому CSS і JavaScript можуть не завантажитися або не виконатися: тимчасова помилка мережі, блокувальник реклами, недоступна CDN, збій DNS, помилка в оновленні браузера, корпоративний чи особистий брандмауер. Жодна з них не пов’язана зі старим браузером, і читач не бачить, яка саме спрацювала. Усе, що бачить читач, — кнопка, яка нічого не робить. Посібник не наводить цифр про те, як часто трапляється кожна з них; його аргумент спирається на перелік, а не на відсоток.

Деякі читачі також вимикають щось навмисно. Посібник пише, що деякі користувачі свідомо вимикають функції у своїх браузерах і що їхній вибір слід поважати. HTML Standard пише, що багато користувачів вимикають скрипти, і з цієї причини, а також тому, що декларативні механізми часто легше підтримувати, закликає авторів використовувати їх замість скриптів, де це можливо. Його приклад — елемент `details`, який показує й ховає частину сторінки без жодного рядка JavaScript. Стандарт також просить авторів дбати про те, щоб їхні застосунки коректно деградували, коли підтримки скриптів немає.

Для британських державних сервісів посібник перетворює це на правило: ними має бути можливо користуватися лише з HTML, а JavaScript, де його використовують, має тільки покращувати HTML і CSS, щоб люди могли користуватися сервісом, навіть коли скрипт не спрацює. Правило написане для сервісів одного уряду, а не для всього вебу, але його аргументи стосуються будь-якого сайту, чиїх читачів автори не можуть обирати.

Команда теж дещо виграє. Коли покращення ламається, читач утрачає покращення, а не завдання, тож помилка в скрипті стає прикрістю, а не збоєм сервісу. Нову функцію можна випустити для браузерів, які її підтримують, без окремого плану для решти, бо ті зберігають сторінку, яку вже мали.

Як це виглядає на практиці

Стандарт HTML сам наводить приклад того, що працює без скриптів: таблицю, яку пересортовують заголовки стовпців, і це добра перевірка підходу. Основа — таблиця зі звичайними посиланнями в заголовках. Клік просить сервер надіслати ту саму таблицю, відсортовану за цим стовпцем, і сервер повертає її в такому порядку. Це працює в будь-якому браузері, зі скриптом чи без нього.

```
<table id="orders">
  <thead>
    <tr>
      <th><a href="?sort=date">Дата</a></th>
      <th><a href="?sort=total">Сума</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>
```

Скрипт — це покращення. Він знаходить ті самі посилання, скасовує перехід і переставляє рядки прямо на сторінці, що заощаджує запит до сервера й назад.

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

Якщо збій стається до додавання обробника, відповідне посилання залишається працездатним. Якщо `sort.mjs` так і не надійде, посилання робитимуть те саме, що й раніше. Якщо скрипт завершиться з помилкою на півдорозі, заголовки, до яких він не встиг додати обробники, і далі використовуватимуть сервер. Браузер, що не підтримує скрипти-модулі, сприймає `type="module"` як незнайомий тип скрипту й, як каже Стандарт HTML, ігнорує його. Скрипт-модуль без `async` виконується, коли сторінку вже повністю розібрано, тож таблиця вже існує, коли скрипт її шукає. Цей приклад не відновлює роботу, якщо доданий обробник натискання кидає помилку після того, як виклик `preventDefault()` скасував перехід.

Ціну платить сервер: основі потрібна кінцева точка, що вміє сортувати, і обидва способи сортування мають давати однаковий результат. Це звична ціна підходу, сплачена один раз. Очікувана перевага — сторінка, головне завдання якої можна й далі виконувати через резервний шлях на сервері; доданий обробник має зберігати цей шлях, якщо його власна робота завершується збоєм.

Цей сайт чітко називає свою основу. Заява про доступність описує повні сторінки на HTML, CSS і SVG без виконуваного JavaScript. Код пошуку збережено, але вимкнено, тож сторінки не залежать від фільтра. Заява називає й обмеження: маленькі підписи, що з'являються при наведенні на значок або його фокусуванні, не можна прибрати клавішею. Для цього був би потрібен код у браузері, якого сайт не постачає. Тому критерій 1.4.13 Content on Hover or Focus виконується лише частково.

Де виникають проблеми

Блок `noscript` замість основи. Елемент `noscript` показує свій вміст лише тоді, коли скрипти вимкнено, і Стандарт HTML каже про це прямо: «Елемент noscript — грубий інструмент». Скрипти можуть бути ввімкнені, а скрипт сторінки все одно впасти, і тоді читач не отримає ні сторінки зі скриптом, ні запасного варіанта. Почніть зі сторінки, що працює без скрипту, і дайте скрипту її змінити, як це робить власний приклад стандарту: його скрипт додає до форми живий результат і прибирає з неї кнопку надсилання.

Одна непідтримувана функція зупиняє весь скрипт. Оскільки шар JavaScript не стійкий до помилок, один фрагмент новішого синтаксису в бандлі може потягнути за собою всі покращення з того самого бандла, навіть ті, що інакше спрацювали б. Тримайте основу незалежною від скрипту, щоб збій коштував лише покращення. Завантажуйте сучасний код через `type="module"`, який старіші браузери пропускають, а якщо старішим браузерам узагалі потрібна покращена версія, дайте їм окремий класичний бандл з атрибутом `nomodule`, який браузери з підтримкою модулів не виконують.

Новий селектор тягне за собою справні правила. Браузер, що зустрічає незнайомий селектор, відкидає все правило разом з оголошеннями. Тож новий псевдоклас, поставлений у список разом із селекторами, які

розуміють усі браузери, може прибрати стилі з браузерів, що інакше їх застосували б. Давайте селектору, який деякі браузери можуть не підтримувати, окреме правило.

Запасні стилі просочуються в покращену розкладку. Ширини, задані для браузерів без новішої системи розкладки, можуть конфліктувати з цією системою в браузерах, які її мають. CSS Conditional Rules Level 3 показує розв'язок на прикладі флекс-розкладки та блоку `@supports not (display: flex)` для альтернативи й зауважує, що оголошення ширини можуть зашкодити флекс-розкладці. Тому їм місце лише в стилях для браузерів без неї.

Зображення обирає скрипт. Стандарт HTML зауважує: якщо сторінка не вказує жодного зображення в розмітці й залишає вибір скрипту, то для читачів із вимкненими скриптами не завантажиться жодне зображення, а якщо скрипт замінює зображення, яке розмітка вже вказала, браузер може почати два завантаження. Перелічіть кандидатів у HTML через `srcset` і `sizes` або через `picture` і дайте браузерові вибрати.

Поведінка без нічого під нею. Список стратегій у тексті Champeon і Finck вимагає запасного варіанта для будь-якої поведінки: справжнього `href` за кожним обробником кліку і перевірки на сервері за кожною перевіркою в браузері. Елемент керування, що існує лише як обробник кліку, або форма, яку перевіряють лише в браузері, не мають основи взагалі. Спершу використовуйте справжні посилання та справжнє надсилання форм, завжди перевіряйте дані на сервері, а перевірку в браузері вважайте зручністю, доданою згори.

Думка, що це стосується старих браузерів. Команда, яка вважає, що основа потрібна для старих браузерів, відмовляється від неї, щойно ці браузери зникають. Перелік причин у GOV.UK стосується мереж, блокувальників, брандмауерів і зламаних оновлень, а з ними стикаються й сучасні браузери. Підтримуйте основу як шлях, яким користуються й який перевіряють: час від часу завантажуйте сторінку із заблокованими скриптами й проходьте її головне завдання від початку до кінця.

Що почитати далі

- Using progressive enhancement — настанови GOV.UK Service Manual, востаннє оновлені 27 вересня 2024 року: три шари, як ламається кожен із них і чому державні сервіси мають працювати лише з HTML.

- HTML Standard: Scripting — елемент `script` з `type="module"` і `nomodule` та елемент `noscript` з порадою на нього не покладатися.
- CSS Conditional Rules Module Level 3 — Candidate Recommendation Draft W3C від 15 серпня 2024 року, що визначає `@supports` з `not`, `and` і `or`.
- HTML Standard: Images — `srcset`, `sizes` і `picture`, і чому вибір зображення скриптом підводить читачів із вимкненими скриптами.

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 прогресивне покращення?» 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/progressive-enhancement/