Engineer

Прогресивний вебзастосунок

вебсайт, який можна встановити й запускати як застосунок

Що це таке

Прогресивний вебзастосунок (англійською Progressive Web App, скорочено PWA) — це вебсайт, зроблений так, щоб браузер міг його встановити й запускати як застосунок: з власного значка, у власному вікні й часто так, що частина його працює і без мережі. При цьому він увесь час лишається звичайним вебсайтом. Магазин застосунків тут ні до чого, а сторінки — той самий HTML, CSS і JavaScript, який браузер інакше завантажив би у вкладці.

Жоден стандарт не визначає цього терміна. Стандарти визначають дві частини, з яких його складають. Перша — маніфест вебдодатка, JSON-файл, описаний у специфікації W3C Web Application Manifest: він дає застосунку назву, значки, початковий URL і режим відображення. Планку ця специфікація ставить низько: «Будь-який вебсайт є вебзастосунком, який можна встановити». Браузер може запропонувати спосіб такий сайт встановити, і встановлений застосунок потім відкривається з налаштуваннями з маніфесту. У режимі відображення `standalone` він має бути схожим на нативний застосунок — це може означати окреме вікно і значок у меню програм, — а браузер прибирає свій стандартний інтерфейс, наприклад адресний рядок.

Друга частина — service worker зі специфікації W3C Service Workers. Це скрипт, який браузер виконує окремо від сторінки: він стоїть між сторінкою й мережею, бачить запити, які надсилає сторінка, і може відповідати на них із кешів, якими керує сам. Саме це дає вебзастосунку змогу й далі віддавати свої сторінки без мережі. Ці кеші належать скрипту: специфікація тримає їх цілком окремо від звичайного HTTP-кешу браузера, і змінюються вони лише тоді, коли їх змінює скрипт.

У Web Application Manifest слово «progressive» не трапляється жодного разу, а специфікація Service Workers згадує його один раз — у примітці про те, як Chromium розпізнає PWA. Посібник Next.js про PWA радить прогресивне покращення (progressive enhancement), щоб застосунок і далі добре працював у браузері, який не підтримує частини його PWA-можливостей. У цьому сенсі кожна частина необов'язкова. Браузер, який ігнорує маніфест або взагалі не підтримує service workers, однаково показує вебсайт.

Прогресивний вебзастосунок — це й не окремий різновид вебсайту. Односторінковий застосунок, статичний сайт і сайт із серверним рендерингом можуть однаково додати маніфест і service worker. Термін описує, що браузер може зробити із сайтом після завантаження, а не те, як з'являється його HTML.

Застосуйте поняття до рішення

Застосуйте «Що таке прогресивний вебзастосунок?» 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

Посібник Next.js наводить три аргументи. Оновлення виходять одразу, без очікування схвалення в магазині застосунків. Одна кодова база працює на всіх платформах. І застосунок отримує можливості, яких люди чекають від нативних застосунків, як-от встановлення на головний екран і push-сповіщення. Усі три випливають із того, що застосунок і далі є вебсайтом: нову версію можна опублікувати без окремого випуску в магазині застосунків; коли кожен пристрій перейде на неї, залежить від кешування й оновлення service worker.

Робота без мережі — причина, з якої service worker узагалі існує. Розділ специфікації про мотивацію каже, що вебзастосунки традиційно виходили з того, що мережа досяжна, і що це ставить вебвміст у невигідне становище порівняно з іншими технологічними стеками. Service worker покликаний виправити цей дисбаланс. Коли він є, потяг у тунелі чи корабель далеко від берега вже не мусять закінчуватися сторінкою помилки браузера.

Обидві половини незалежні. Посібник Next.js зазначає, що для пропозиції встановити застосунок підтримка офлайну не потрібна, а специфікація маніфесту прибрала власний член `serviceworker` . Застосунок може встановлюватися без service worker, а service worker може навчити звичайний сайт працювати офлайн без жодного маніфесту.

Спільне в обох половин — HTTPS. Next.js називає сайт, який віддається через HTTPS, однією з двох умов, без яких застосунок не встановити на головний екран телефона; друга умова — коректний маніфест. Специфікація Service Workers вимагає, щоб service workers працювали в захищених контекстах, тобто на практиці через HTTPS, а для розробки дозволено localhost.

Як це виглядає на практиці

Робочий PWA складається з трьох невеликих файлів, і всі вони написані для цієї статті. Маніфест дає застосунку назву, вибирає режим відображення й перелічує значки:

```
{
  "name": "Field Notes",
  "short_name": "Notes",
  "start_url": "/",
  "display": "standalone",
  "icons": [
    { "src": "/icon-192.png", "sizes": "192x192", "type": "image/png" },
    { "src": "/icon-512.png", "sizes": "512x512", "type": "image/png" }
  ]
}
```

Кожна сторінка підключає маніфест і реєструє service worker, але лише в браузері, який це підтримує:

```
<link rel="manifest" href="/manifest.webmanifest" />
<script>
  if ("serviceWorker" in navigator) {
    navigator.serviceWorker.register("/sw.js");
  }
</script>
```

А сам service worker зберігає невеликий набір файлів для роботи офлайн:

```
// sw.js
const CACHE = "shell-v2";
const SHELL = ["/offline.html", "/app.css", "/app.js"];

async function precache() {
  const cache = await caches.open(CACHE);
  await cache.addAll(SHELL);
}

async function dropOldCaches() {
  const names = await caches.keys();
  const old = names.filter((name) => name !== CACHE);
  await Promise.all(old.map((name) => caches.delete(name)));
}

async function respond(request) {
  if (request.mode === "navigate") {
    try {
      return await fetch(request);
    } catch {
      return caches.match("/offline.html");
    }
  }
  const hit = await caches.match(request);
  return hit ?? fetch(request);
}

self.addEventListener("install", (event) => {
  event.waitUntil(precache());
});

self.addEventListener("activate", (event) => {
  event.waitUntil(dropOldCaches());
});

self.addEventListener("fetch", (event) => {
  if (event.request.method !== "GET") return;
  event.respondWith(respond(event.request));
});
```

Маніфест має розширення `.webmanifest`, зареєстроване для цього формату, і віддається з медіатипом `application/manifest+json`. Service worker лежить у корені, бо його область дії (scope) за замовчуванням — каталог, у якому лежить скрипт: worker за адресою `/js/sw.js` міг би керувати лише сторінками під `/js/`, якщо сервер не надішле заголовок `Service-Worker-Allowed`. Під час встановлення він відкриває кеш з назвою `shell-v2` і зберігає туди офлайн-сторінку, таблицю стилів і скрипт. `addAll()` відхиляє проміс, якщо будь-який із цих файлів повернеться зі статусом помилки, а оскільки

цей проміс передано в `waitUntil()`, разом із ним зривається й усе встановлення, і напівзаповненого кешу не лишається. Під час активації worker видаляє всі кеші з іншою назвою. Переходи між сторінками спершу йдуть у мережу й повертаються до збереженої офлайн-сторінки, коли мережа не відповідає. Усе інше віддається з кешу, якщо воно там є, і завантажується з мережі, якщо немає. Запитів, відмінних від `GET`, він не чіпає.

Компанія описує повну реалізацію цього на справжній платформі. Запис у портфоліо пояснює, навіщо це знадобилося: багато користувачів платформи — моряки й працівники на виїзді, які весь час у телефоні, часто без мережі або зі слабким зв'язком, тож застосунок треба було зробити таким, щоб його встановлювали, як звичайну програму, і щоб він працював і тоді, коли пропадає сигнал. Запис охоплює обидві половини. Застосунок встановлюється на головний екран зі значками кількох розмірів, екраном-заставкою та кольорами теми. Service worker підключено за допомогою плагіна next-pwa, а кешування Workbox під час виконання віддає шрифти, зображення, аудіо, відео й ресурси з CDN за принципом «спершу кеш», бо між запитами все це лишається незмінним, а поруч працюють заголовки HTTP cache-control. Оскільки service workers потребують HTTPS, встановлення й роботу без мережі перевіряли на справжніх пристроях, і саме через HTTPS. Результат, про який ідеться в записі: люди встановлюють застосунок і користуються ним без мережі, а повторні відвідини беруться з кешу.

Половина, що стосується встановлення, теж виглядає інакше з погляду читача. На сайті є сторінка про те, як установити його як застосунок, і вона пояснює, де кожен браузер ховає потрібну команду: «Поділитися» й «Додати на початковий екран» у Safari на iPhone, невеликий значок праворуч в адресному рядку в Chrome, Edge, Brave, Opera та Vivaldi на комп'ютері і «Додати в Dock» у Safari на macOS Sonoma чи новішій. Там також сказано, що iOS сама ніколи не пропонує встановлення. Firefox надає окремі вікна вебзастосунків у Windows, починаючи з версії 143 або з версії 150, якщо Firefox встановлено з Microsoft Store, але не в macOS чи Linux. Сторінка також прямо каже, що дає встановлення цього сайту: значок і вікно, але не копію для роботи офлайн, бо без мережі сайт працює лише тоді, коли в нього є service worker.

Де виникають проблеми

Користувачі застрягають на старій версії. Згідно зі специфікацією Service Workers, об'єкти кешу не оновлюються, якщо автор явно цього не попросить, і термін їхньої дії не спливає, доки автор їх не видалить. Тож worker, який спершу дивиться в кеш, віддаватиме вчорашню таблицю стилів, доки скрипт не скаже інакше. Додайте версію в назву кешу, змінюйте її щоразу, коли змінюються закешовані файли, і видаляйте старі кеші під час активації, як це робить приклад. Оновлення запускає саме зміна скрипту: браузер побайтово порівнює щойно завантажений `sw.js` зі встановленим. Але й тоді новий worker чекає, доки старим не перестануть користуватися всі вкладки й вікна, якщо тільки він не викличе `skipWaiting()`, тож користувач із відкритою вкладкою працює з попередньою версією стільки, скільки ця вкладка лишається відкритою. Посібник Next.js віддає `sw.js` із заголовком `Cache-Control: no-cache, no-store, must-revalidate`, щоб користувачі завжди отримували найновішу версію самого worker.

На ноутбуці працює, а на тестовому телефоні — ні. `navigator.serviceWorker` існує лише в захищених контекстах. На сторінці, яку віддають через звичайний HTTP, звична перевірка підтримки мовчки дає false, worker так і не реєструється, а сторінка про це мовчить. Специфікація дозволяє браузерам пускати localhost, 127.0.0.0/8 і ::1/128 для розробки, тому worker, який справно працює на власній машині розробника, може зникнути, щойно телефон відкриє той самий застосунок через HTTP за адресою цієї машини в локальній мережі. Next.js пропонує `next dev --experimental-https` для локального тестування через HTTPS. Перевіряйте встановлення й роботу офлайн на реальних пристроях і через HTTPS, перш ніж вважати бодай одне з них готовим.

Обробник fetch, який нічого не робить. Специфікація Service Workers фіксує, що деякі сайти реєструють обробник `fetch` з порожнім тілом лише для того, щоб Chromium розпізнав їх як PWA. Тепер специфікація виявляє обробники на зразок `() => {}` і заохочує браузери попереджати, що вони зайві й можуть погіршувати продуктивність. Напишіть обробник, який справді відповідає на запити, або не додавайте його зовсім.

Розрахунок на пропозицію встановити. Чи пропонувати встановлення і як саме, вирішує браузер. Next.js пише, що сучасні браузери автоматично пропонують встановити застосунок, щойно сайт має коректний маніфест і HTTPS, і радить не будувати власну кнопку встановлення на `beforeinstallprompt`, бо ця подія не працює однаково в усіх браузерах і на всіх

платформах, зокрема в Safari на iOS. Специфікація маніфесту відтоді взагалі прибрала цю подію. Режим відображення й початковий URL теж лише рекомендаційні: браузер, який не підтримує `standalone`, переходить на `minimal-ui`, а потім на `browser`, тобто звичайну вкладку. Поясніть користувачам, як установити застосунок, як це робить інструкція для iOS у посібнику Next.js, і розраховуйте на те, що застосунок відкриють і у вкладці.

Що почитати далі

- Web Application Manifest — формат маніфесту, його члени, режими відображення і що означає встановити й видалити застосунок.
- Service Workers — життєвий цикл worker, область дії, правила оновлення, вимога захищеного контексту і Cache API.
- Web Application Manifest: Installable web applications — як браузер встановлює вебзастосунок, запускає його й дає користувачеві знову його видалити.
- Service Workers: Understanding cache lifetimes — чому об’єкти кешу не оновлюються й не застарівають, доки цього не зробить скрипт.
- How to build a Progressive Web Application (PWA) with Next.js — файл маніфесту, push-сповіщення, інструкції зі встановлення для iOS, HTTPS під час розробки і заголовки безпеки для service worker.

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 прогресивний вебзастосунок?» 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-web-app/