Engineer

Application web progressive

Site installable et utilisable comme une application

Ce que c’est

Une application web progressive, en anglais progressive web app ou Progressive Web App, généralement abrégée en PWA, est un site web conçu pour qu’un navigateur puisse l’installer et l’exécuter comme une application : depuis sa propre icône, dans sa propre fenêtre, et souvent avec une partie de ses fonctions encore disponible sans réseau. Elle reste en permanence un site web ordinaire. Rien ne provient d’une boutique d’applications, et ses pages reposent sur le même HTML, le même CSS et le même JavaScript qu’un navigateur chargerait dans un onglet.

Aucune norme ne définit ce terme. Les normes définissent les deux éléments sur lesquels il repose. Le premier est le manifeste d’application web, un fichier JSON décrit par Web Application Manifest du W3C, qui donne à l’application un nom, des icônes, une URL de démarrage et un mode d’affichage. Cette spécification fixe une condition d’admissibilité très simple : « Tout site web est une application web installable. » Un navigateur peut proposer un moyen de l’installer, puis l’application installée s’ouvre avec les réglages du manifeste. Dans le mode d’affichage `standalone`, elle est censée ressembler à une application native, ce qui peut inclure sa propre fenêtre et une icône dans le lanceur d’applications. Le navigateur omet alors son interface habituelle, comme la barre d’URL.

Le second élément est le service worker, défini par la spécification Service Workers du W3C. C’est un script que le navigateur exécute séparément de la page, placé entre celle-ci et le réseau : il voit les requêtes de la page et peut y répondre depuis des caches qu’il contrôle. C’est ce qui permet à une application web de continuer à servir ses pages hors connexion. Ces caches appartiennent au script : la spécification les sépare entièrement du cache HTTP ordinaire du navigateur, et ils ne changent que lorsque le script les modifie.

Web Application Manifest n'emploie jamais le mot « progressive », et la spécification Service Workers l'emploie une fois, dans une note sur la façon dont Chromium reconnaît une PWA. Le guide PWA de Next.js recommande l'amélioration progressive, afin que l'application continue de bien fonctionner dans un navigateur qui ne prend pas en charge certaines de ses fonctions PWA. En ce sens, chaque élément est facultatif. Un navigateur qui ignore le manifeste ou ne prend pas en charge les service workers affiche quand même le site web.

Une application web progressive n'est pas non plus une catégorie distincte de site. Une application monopage, un site statique et un site rendu sur le serveur peuvent chacun ajouter un manifeste et un service worker. Le terme décrit ce qu'un navigateur peut faire d'un site après son chargement, pas la façon dont son HTML est produit.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'une application web progressive ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des interfaces rapides, utilisables sur téléphone et cohérentes dans tous les navigateurs de vos clients.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Développement cartographique GIS full-stack — cartographie full-stack, de la base au navigateur

Développement cartographique GIS full-stack Contactez-nous →

Pourquoi c'est important

Le guide Next.js présente trois avantages. Les mises à jour sont diffusées immédiatement, sans attendre l'approbation d'une boutique d'applications. Une même base de code dessert toutes les plateformes. L'application reçoit aussi des fonctions attendues des applications natives, comme l'installation sur l'écran d'accueil et les notifications push. Ces trois avantages découlent du fait qu'elle reste un site web : une nouvelle version peut être publiée sans sortie distincte dans une boutique d'applications ; le moment où chaque appareil l'adopte dépend de la mise en cache et des mises à jour du service worker.

Le service worker existe pour le fonctionnement hors connexion. La section de motivation de la spécification explique que les applications web ont traditionnellement supposé le réseau accessible et que les autres plateformes applicatives gèrent donc mieux les interruptions de connexion. Les service

workers donnent aux applications web un moyen de les gérer elles aussi. Avec un tel script, se trouver dans un train en tunnel ou sur un navire hors de portée des côtes ne conduit plus nécessairement à la page d'erreur du navigateur.

Les deux moitiés sont indépendantes. Le guide Next.js précise que les propositions d'installation ne nécessitent pas de prise en charge du fonctionnement hors connexion, et la spécification du manifeste a supprimé son propre membre `serviceworker`. Une application peut être installable sans service worker, et un service worker peut faire fonctionner un site ordinaire hors connexion sans aucun manifeste.

Les deux moitiés ont HTTPS en commun. Next.js cite un site servi en HTTPS parmi les deux conditions nécessaires pour installer une application sur l'écran d'accueil mobile, l'autre étant un manifeste valide. La spécification Service Workers exige que les service workers s'exécutent dans des contextes sécurisés, ce qui signifie en pratique HTTPS, avec une exception pour localhost en développement.

À quoi cela ressemble en pratique

Trois petits fichiers, tous écrits pour cet article, constituent une PWA fonctionnelle. Le manifeste nomme l'application, choisit le mode d'affichage et énumère ses icônes :

```
{
  "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" }
  ]
}
```

Chaque page référence le manifeste et enregistre le service worker, mais seulement dans un navigateur qui le prend en charge :

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

```
    navigator.serviceWorker.register("/sw.js");
  }
</script>
```

Le service worker conserve ensuite un petit ensemble de fichiers pour l'utilisation hors connexion :

```
// 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));
});
```

Le manifeste utilise `.webmanifest`, l'extension de fichier enregistrée à cet effet, et est servi avec le type de média `application/manifest+json`. Le service worker est placé à la racine car son périmètre est par défaut le répertoire de son script : un worker dans `/js/sw.js` ne pourrait donc contrôler que les pages sous `/js/`, sauf si le serveur envoyait un en-tête `Service-Worker-Allowed`. À l'installation, il ouvre un cache nommé `shell-v2` et y conserve la page hors connexion, la feuille de style et le script. `addAll()` rejette la promesse si l'un de ces fichiers est renvoyé avec un statut d'erreur. Comme cette promesse a été transmise à `waitUntil()`, toute l'installation échoue avec elle, au lieu de laisser un cache à moitié rempli. À l'activation, il supprime tous les caches portant un autre nom. Les navigations vers des pages passent d'abord par le réseau et se rabattent sur la page hors connexion en cache si le réseau ne répond pas. Tout le reste est servi depuis le cache lorsque la ressource y figure, et récupéré sur le réseau sinon. Les requêtes autres que `GET` ne sont pas traitées.

L'entreprise décrit une réalisation complète sur une plateforme réelle. La page du portfolio explique le besoin : beaucoup d'utilisateurs de la plateforme sont des gens de mer et du personnel de terrain sur téléphone, souvent hors connexion ou avec une connexion faible. L'application devait donc s'installer comme une application native et continuer de fonctionner quand le signal disparaît. La réalisation couvre les deux moitiés. L'application s'installe sur l'écran d'accueil avec des icônes de plusieurs tailles, un écran de démarrage et des couleurs de thème. Le service worker est configuré par l'extension next-pwa, et Workbox met en cache à l'exécution les polices, les images, les fichiers audio et vidéo et les ressources CDN, puis les sert en donnant la priorité au cache, puisqu'ils ne changent pas d'une requête à l'autre, en complément d'en-têtes HTTP de contrôle du cache. Les service workers exigeant HTTPS, l'installation et le fonctionnement hors connexion ont été testés en HTTPS sur de vrais appareils. Le résultat rapporté est une application que les utilisateurs installent et continuent d'utiliser hors connexion, l'application se chargeant depuis le cache lors des visites suivantes.

L'installation se présente aussi différemment du côté du lecteur. La page expliquant comment installer ce site comme application indique où chaque navigateur place la commande : le menu Share (partage) et Add to Home Screen (ajout à l'écran d'accueil) dans Safari sur iPhone, une petite icône à droite de la barre d'adresse dans Chrome, Edge, Brave, Opera et Vivaldi sur ordinateur, et Add to Dock (ajout au Dock) dans Safari sur macOS Sonoma ou version ultérieure. Elle précise aussi qu'iOS ne propose jamais spontanément l'installation. Firefox fournit des fenêtres d'application web distinctes sous Windows à partir de la version 143, ou de la version 150 si Firefox a été installé depuis le Microsoft

Store, mais pas sous macOS ni sous Linux. La page explique également clairement ce que donne l'installation de ce site : une icône et une fenêtre, mais pas de copie hors connexion, car le fonctionnement hors connexion nécessite un service worker.

Où les problèmes surviennent

Les utilisateurs conservent l'ancienne version. Selon la spécification Service Workers, les objets de cache ne sont mis à jour que si l'auteur le demande explicitement, et ils n'expirent que si l'auteur les supprime. Un worker donnant la priorité au cache sert donc la feuille de style d'hier jusqu'à ce que son script en décide autrement. Inscrivez une version dans le nom du cache, changez-la chaque fois que les fichiers en cache changent et supprimez les anciens caches à l'activation, comme dans l'exemple. C'est la modification du script qui déclenche une mise à jour, car le navigateur compare octet par octet le `sw.js` nouvellement récupéré à celui qui est installé. Même alors, le nouveau worker attend qu'aucun onglet ni aucune fenêtre n'utilise l'ancien, sauf s'il appelle `skipWaiting()`. Un utilisateur qui garde un onglet ouvert conserve donc la version précédente tant qu'il reste ouvert. Le guide Next.js sert `sw.js` avec `Cache-Control: no-cache, no-store, must-revalidate`, pour que les utilisateurs obtiennent toujours la dernière version du worker lui-même.

Cela fonctionne sur l'ordinateur portable, mais pas sur le téléphone de test. `navigator.serviceWorker` n'existe que dans les contextes sécurisés. Sur une page servie en HTTP simple, le contrôle habituel de prise en charge renvoie silencieusement faux, le worker n'est jamais enregistré et rien dans la page ne le signale. La spécification permet aux navigateurs d'autoriser localhost, 127.0.0.0/8 et ::1/128 en développement. Un worker qui fonctionne sur la machine du développeur peut donc manquer lorsqu'un téléphone ouvre la même application en HTTP à l'adresse de cette machine sur le réseau local. Next.js propose `next dev --experimental-https` pour les essais locaux en HTTPS. Testez l'installation et le fonctionnement hors connexion sur de vrais appareils, en HTTPS, avant de considérer l'un ou l'autre comme achevé.

Un écouteur fetch qui ne fait rien. La spécification Service Workers indique que certains sites enregistrent un écouteur `fetch` au corps vide uniquement pour que Chromium les reconnaisse comme application web progressive. Elle détecte désormais les écouteurs comme `() => {}` et encourage les navigateurs à avertir qu'ils sont inutiles et peuvent nuire aux performances. Écrivez un gestionnaire qui répond aux requêtes, ou omettez cet écouteur.

Compter sur une proposition d'installation. Proposer ou non l'installation, et sous quelle forme, relève du navigateur. Next.js indique que les navigateurs modernes affichent automatiquement une proposition dès qu'un site possède un manifeste valide et utilise HTTPS. Le guide déconseille de construire un bouton d'installation personnalisé sur `beforeinstallprompt`, car cet événement ne fonctionne pas dans tous les navigateurs et sur toutes les plateformes, notamment dans Safari sur iOS. La spécification du manifeste a depuis supprimé cet événement. Le mode d'affichage et l'URL de démarrage sont également indicatifs : un navigateur qui ne prend pas en charge `standalone` se rabat sur `minimal-ui`, puis sur `browser`, un onglet ordinaire. Expliquez aux utilisateurs comment installer l'application, comme le font les instructions iOS du guide Next.js, et prévoyez qu'elle puisse être ouverte dans un onglet.

Pour aller plus loin

- Web Application Manifest — le format du manifeste, ses membres, les modes d'affichage et le sens de l'installation et de la désinstallation.
- Service Workers — le cycle de vie du worker, son périmètre, ses règles de mise à jour, l'exigence d'un contexte sécurisé et l'API Cache.
- Web Application Manifest : applications web installables — comment un navigateur installe une application web, la lance et permet à l'utilisateur de la supprimer.
- Service Workers : comprendre la durée de vie des caches — pourquoi les objets de cache ne se mettent jamais à jour et n'expirent jamais sans instruction du script.
- Créer une application web progressive (PWA) avec Next.js — un fichier manifeste, les notifications push, les instructions d'installation pour iOS, HTTPS en développement et les en-têtes de sécurité du service worker.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'une application web progressive ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/progressive-web-app/