PWA

Progressive Web App

La problématique du service numérique

- Pour accéder à un service numérique, les internautes ont deux solutions : utiliser leur navigateur (site web) ou lancer une application installée sur leur terminal.
- Le développeur doit donc concevoir un site web adapté au mobile mais aussi une application native au système employé (Android, iOS…). Cette stratégie est coûteuse: plusieurs produits sont développés et maintenus en parallèle, et les ponts entre les sites et les applications sont très limités.

Le composant Web View

- Pour répondre à cette problématique, les applications hybrides sont nées. Il s’agit d’embarquer des technologies web dans une coquille vide, un composant appelé WebView. On accède à certaines fonctionnalités native du mobile, la base du code est unique, les compétences en développement web sont pratiquement suffisantes.

- Revers de la médaille : les performances ne sont pas satisfaisantes, le site et l’application sont encore séparés (vous ne pourrez pas, par exemple, synchroniser facilement le contenu d’un panier) et vous n’accédez pas à l’ensemble des fonctionnalités natives du terminal.

- De plus, la valeur ajoutée apportée n'est pas très grande... C'est construire une application avec des éléments importés ; certains éditeurs refusent ce genre de pratique (Apple pour ne pas le nommer).

C’est quoi au juste une PWA ?

- On peut définir une PWA comme « un site Web qui a l’apparence et le comportement d’une application mobile native ».

- Une PWA :

  - s’affiche en plein écran
  - se charge rapidement
  - envoie des notifications push
  - fonctionne hors connexion

Ce que doit être une PWA...

- **Fiable** : charger instantanément et être en mesure de fonctionner (partiellement) même en cas de couverture limitée ou si l'internaute est hors connexion (gestion fine du cache, synchronisations en arrière-plan…)

- **Rapide** : répondre rapidement aux interactions des internautes. La performance est un point clé des PWA, il est essentiel d'intégrer l'optimisation des performances à toutes les étapes du développement.

- **Engageante** : les progressive web apps peuvent être installées, depuis un site web ou un magasin d'applications, et intégrer des mécanismes de ré-engagement tel que les notifications push sur mobile.

Avantages d’une PWA

1) **Proposer** une expérience utilisateur identique à celle d’une application native

2) **Plus**…

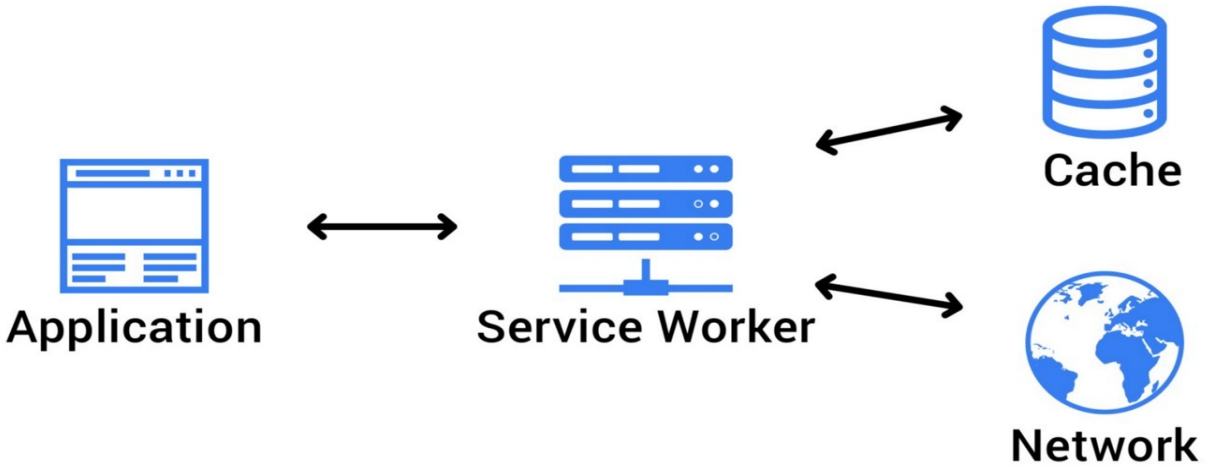
- aucune installation donc moins de mémoire utilisée
- référencement naturel des sites Web
- totalement responsive

Inconvénients d’une PWA

a) Les PWA offrent plus de possibilités actuellement sur Android que sur IOS → normal car « poussé » par Google…

b) Elles consomment plus de batterie car plus complexe → normal car n’étant pas natives…

c) Elles n’utilisent pas pleinement les fonctionnalités du smartphone → normal car n’utilisant que le principe du « service worker », chargé de gérer les requêtes



Comparatif...

	Application web	Progressive Web App	Application mobile (natif)
Device cible	N'importe quel device avec un navigateur web compatible		Se repose sur l'OS du device
Langage de programmation	Web		Swift (iOS) Java (Android)
UI / UX	Sur mesure		Natif
Mode d'exécution	Ouverture dans un navigateur	Ouverture dans un navigateur Ouverture dans une app	Ouverture dans une app
Disponibilité dans le store	Non	Possible	Oui
Mode offline	Non	Oui	Oui
Cross-platform	Oui	Oui	Non
Accès à l'application	A partir de l'url de l'application		Téléchargement via le store
Mises à jour de l'application	Inutiles – Accès direct via le site		Téléchargement via le store
Référencement dans les moteurs de recherche	Oui	Oui	Non

Bases de construction

- **Le manifeste**

  Une PWA possède deux briques essentielles et le premier est un manifeste : un fichier JSON qui décrit les données de base de l'app, pour qu'elle puisse se comporter comme une application native (icône, couleur de fond du splash screen, nom de l'application…).

- **Le service worker**

  Le second élément est le service worker. Le cœur des PWA, c'est un processus JavaScript qui tourne côté client (navigateur), qui s'exécute en dehors de l'application mais qui est relié à l'application.

  Le service worker sert de proxy : quand l'application demande des ressources, les requêtes passent par le service worker. On peut placer des ressources en cache pour gérer les accès hors-ligne. Le service worker tourne même quand l'application n'est pas lancée, ce qui permet de diffuser des notifications sur les terminaux de vos usagers.

Un service : Workbox

- Pour construire ces premiers éléments, vous pouvez utiliser Workbox, un service proposé par Google pour démarrer rapidement votre progressive web app.

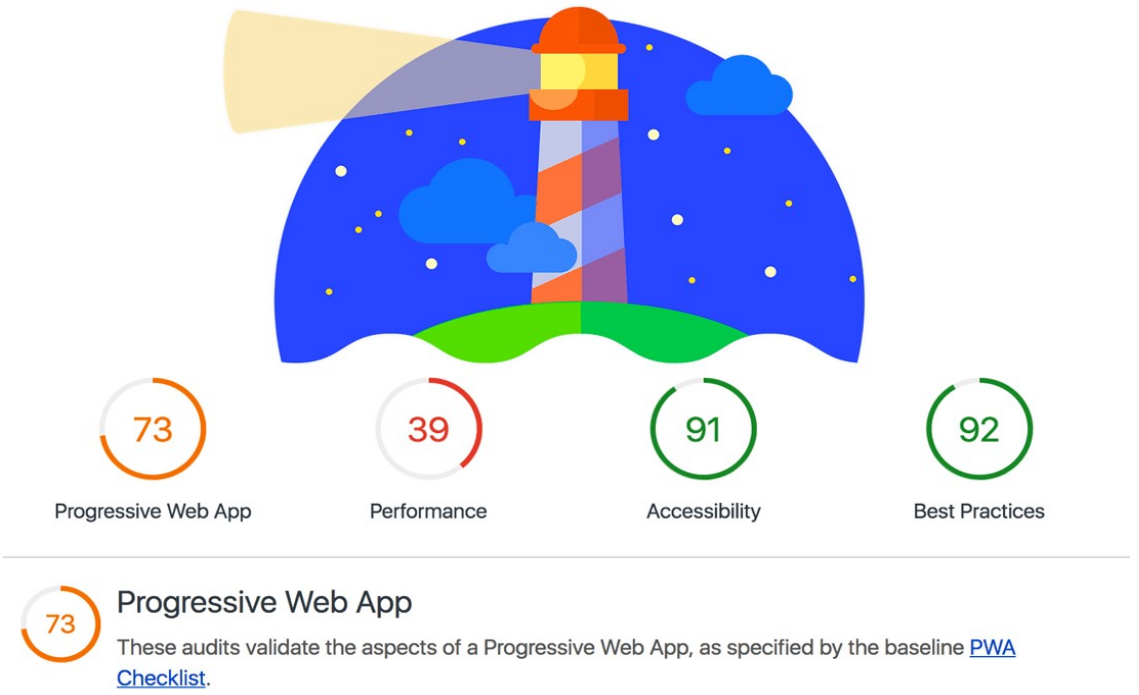
https://developers.google.com/web/tools/workbox/

- Un site d'information possible :

https://progressive-web-apps.fr/

Un outil : Lighthouse

- Cet outil open source sert à l'amélioration de la qualité des pages Web en général et à la détermination du statut PWA d'une application.
- En ligne mais en version beta, elle s'utilise communément sous la forme d'une extension de navigateur, de préférence avec un moteur Chrome.
- https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk



De l’aide...

Voici par exemple (parmi d’autres) un tutoriel en français :

https://avengering.com/tutoriel-pwa-comment-convertir-votre-site-web-en-pwa/

Ou bien utiliser **Angular PWA** pour une application… Angular ! Voici 3 tutoriels :

1) https://medium.com/ngconf/angular-pwa-install-and-configure-858dd8e9fb07
2) https://www.positronx.io/build-progressive-web-app-pwa-with-angular/
3) https://developer.okta.com/blog/2019/01/30/first-angular-pwa

Notez que dans le cas d’une PWA, l’**utilisation d’un serveur Web est obligatoire** pour utiliser le Service Worker !

Voir la démonstration

du cours →