Les PWA

Détail du cours :

- **Date**: 03/2023
- **OS**: win32 x64

Une **PWA** c'est une **Progressive Web App**, cela correspond à une application web adapté aux besoin du web moderne.

Les avantages d'une PWA est qu'elle est installable en tant qu'icone sur votre bureau. (sur l'écran du mobile)

Cela ne transforme pas votre site en application, cela reste un site. Mais ce n'est pas non plus un simple raccourci.

La PWA une fois lancé aura un navigateur à l'interface réduite, faisant ressemblé votre site à une application.

> Note : Actuellement Firefox n'accepte les PWA que sur mobile, j'utiliserait donc Chrome et l'extension lighthouse pour ce cours.

Elle devra aussi respecter un tas de règles améliorant l'expérience utilisateur :

- Les PWA
  - Discoverable
  - Installable
  - Linkable
  - Network Independent
    - Service Worker
  - Progressive
  - Re-engageable
  - Responsive
  - Safe

Discoverable

C'est à dire qu'elle doit être visitable par les robots des moteurs de recherche et respecter autant que possible le SEO.

Elle doit aussi contenir à sa racine, un fichier nommé `manifest.json`, ce fichier va contenir tout un tas d'informations au sujet du site. Mais au minimum il faudra ceci :

```
{
    "name": "Super PWA",
    "icons":[
        {
            "src": "./icons/pizza512.png",
            "type": "image/png",
            "sizes": "512x512"
        },
```

```
        {
            "src": "./icons/pizza32.png",
            "type": "image/png",
            "sizes": "32x32"
        }
    ],
    "start_url": "./?source=pwa",
    "display": "standalone"
}
```

- Le nom du site,
- les icons en petit et grand format
- un url par défaut (ici un paramètre get indique le début)
- un mode d'affichage pour le navigateur.

Ensuite on ira lier le manifest dans le head de notre site:

```
<link rel="manifest" href="manifest.json">
```

Si dans les outils de développement nous allons dans "**Application**" sur chrome, nous verrons le menu "**manifest**" qui nous rappelle les informations du manifest.

Et il indique aussi que nous n'avons pas de "**service worker**", voyons cela au chapitre suivant.

> Il existe tout un tas d'autres propriétés plus ou moins importante que l'on peut ajouter au manifest. Il y a d'autres formats que le json pour le manifest comme `.webmanifest`

Installable

Pour être installable, un site doit :

- Avoir le manifest dont on a parlé précédemment.
- Être en HTTPS ou localhost.
- Avoir une icone pour représenté l'application (favicon et manifest)
- Un `service worker` permettant à l'application de fonctionner hors ligne.

Ce dernier point est nouveau. Un "**service worker**" est un script js permettant de gérer les caches et autre fonctionnalité permettant que votre application n'est pas ou peu de problème si la connexion vient à manquer.

Le but d'une PWA c'est qu'elle fonctionne même sans connexion. (avec bien sûr des fonctionnalités en moins)

Nous allons créer deux fichiers :

- `sw.js` à la racine.
- `script.js` dans un dossier "script"

Nous allons pour l'instant laisser le premier vide, voyons notre script.js :

```
if("serviceWorker" in navigator)
{
    navigator.serviceWorker.register("./sw.js");
}
```

Tous les navigateurs ne supportent pas "**serviceWorker**", on vérifie donc si on y a accès. Si oui alors nous allons enregistrer notre fichier "**sw.js**".

Si nous regardons à nouveau notre navigateur, il nous indiquera maintenant que la page ne peut pas travailler hors ligne. Voyons cela dans un chapitre suivant.

> Ici nous nous contentons du stricte minimum, mais nous pouvons pousser les fonctionnalités biens plus loin.

Linkable

Ici on va y passer vite fait, puisque cela indique simplement qu'à la différence d'une véritable application qui utilisera une logique précise pour charger ses différents éléments, notre application web aura de simple liens.

Network Independent

Une PWA se doit de pouvoir fonctionner un minimum lorsque la connexion est faible ou non existante. Pour cela on fera attention à 3 points :

- Visiter le site et obtenir son contenu même sans connexion.
- Avoir accès à toute donnée que l'utilisateur a déjà pu accédé par le passé.
- Contrôler ce qui sera montré à l'utilisateur si on charge une page sans connexion.

Pour obtenir ce résultat, on mélangera les technologies suivantes :

- "Service Worker API" pour contrôler les pages
- "Cache API" pour sauvegarder les fichiers en cache
- "Web Storage API" pour sauvegarder des donnée
- "IndexedDB API" pour sauvegarder des donnée aussi

Mais pour commencer, on pourra se contenter de venir ajouter ceci à notre fichier "**sw.js**" :

```
self.addEventListener("fetch", ()=>{});
```

On indique à notre service worker que quand on requête (on va chercher) notre page, on va faire une action particulière. (bien que pour l'instant nous ne faisons rien)

Cela suffira à faire disparaître notre précédent message d'erreur. Notre navigateur est maintenant prêt à installer notre PWA. Sur chrome un petit icone est apparut dans la barre d'URL.

Service Worker

Quelques petites explications sur les services workers.

C'est ce fichier qui s'occupera d'intercepter les requêtes envoyés et interagir avec nos pages.

Si on modifie ce fichier, il ne sera pas tout de suite remplacé lorsque qu'on rechargera la page.

le service worker passe par 3 étapes :

- Installation
- Attente
- Activation

Si il n'y a aucun service worker, les trois s'enchaînent directement. Si il y en a déjà un (par exemple si on en avait déjà un et que l'on a modifié celui ci), Il va s'installer, se mettre en attente, puis quand le précédent aura fini de s'exécuter, alors il prendra sa place, reléguant l'ancienne version aux oubliettes.

On pourrait par exemple demander à notre service worker que lorsque l'on requête une page, sauvegarder en cache ses images et son css afin de pouvoir la recharger même sans connexion.

Progressive

Une bonne PWA doit être capable de fonctionner sur n'importe quel navigateur.

Et pas simplement sur les différents navigateurs, mais aussi sur leurs différentes versions, des plus récentes aux plus anciennes.

Le plus important avec ce qu'on appelle "**Progressive Enhancement**" (Amélioration progressive) ce n'est pas forcément que 100% de vos fonctionnalités soient viable sur les plus vieux navigateurs, mais surtout que votre site en général y fonctionne.

Qu'il n'y ai pas d'erreur provoquant l'arrêt de vos scripts et le mauvais fonctionnement de votre site.

Il faut qu'un utilisateur qui n'a pas mis à jour son navigateur ai une experience plus basique mais toujours fonctionnelle.

Une des pratiques permettant cela, est la "**Feature Detection**", cela consiste en ce que nous avons fait plus haut :

```
if("serviceWorker" in navigator)
```

vérifier l'existence d'une technologie avant de l'utiliser. Cela évitera une erreur si elle n'est pas disponible sur un navigateur.

Une autre pratique est le "**polyfill**", cela consiste en la création de morceau de code permettant de remplacer les fonctionnalités qui ne serait pas disponible sur un navigateur :

```
// Exemple : si la fonction trunc n'existe pas, alors je l'implémente.
if (!Math.trunc) {
  Math.trunc = function(number) {
    return number < 0 ? Math.ceil(number) : Math.floor(number);
  };
```

```
}
// On pourrait imaginer de même avec fetch par exemple en créant un équivalent
avec xmlHttpRequest
```

Cela a pour limite que cela ne fonctionnera qu'avec les propriétés et fonctions. Avant 2020, l'opérateur "??" n'existait pas, et on ne peut pas le remplacer simplement.

Pour cela il faudra utiliser un "**transpiler**". Cet un outil qui a pour rôle de compiler un code dans un autre langage, ou alors pour ce qui nous intéresse ici, dans une version plus ancienne du langage.

Un transpiler connu pour javascript est babel js.

```
// exemple :
const h = element.height??100;
// le code ci-dessus deviendra :
var h = (element.height !== undefined && element.height !== null)?
element.height:100;
```

Re-engageable

Ce point là représente le fait de motiver l'utilisateur à revenir sur votre app. C'est une bonne chose que votre utilisateur ai installé votre PWA, mais si il ne la réouvre jamais et qu'elle n'est jamais mis à jour, cela ne sert à rien.

Pour cela on pourra utiliser 3 technologies :

- "Service Worker" pour contrôler les pages.
- "Web Push API" pour mettre à jour vos données depuis le serveur.
- "Notifications API" pour notifier votre utilisateur et le faire revenir.

Responsive

... Non mais là si je dois encore vous l'expliquer celui ci.

Safe

Là aussi, pas beaucoup de choses à ajouter:

- HTTPS
- CSRF
- XSS
- Brute Force
- SQL Injection

Gardez juste ces termes en tête.