**Créer une Progessive Web Application (PWA)**

# 1. PRESENTATION

Une PWA est avant tout une application Web utilisant HTML/CSS et Javascript.

Cependant, le mode PWA apporte 2 fonctions supplémentaires :

1. Une installation de type « Installation d'une application », avec icône sur le bureau (et possibilité de désinstaller)
2. Le fonctionnement hors connexion au réseau Internet.

## Hors connexion ?

Le site web déclaré PWA est copié sur le terminal utilisé, donc réutilisable sans connexion réseau.
On peut le programmer également pour stocker des données venant d'un site (Cookies ou IndexedDB).
Ces données seront accessibles hors connexion, et seront mises à jour dès que la connexion au réseau sera possible.

## Mécanismes à mettre en place pour une PWA :

Pour transformer un site Web en application PWA, il faut :

- Un serveur HTTPS (Sécurisé) ou pour la phase de développement, un serveur http « localhost ».
- Une image pour l'icône sur le bureau. Cette image doit être déclinée en plusieurs tailles.
- Un fichier « Manifeste » qui contiendra les informations nécessaires à l'installation de l'application.
- Un « Service Worker » : un programme en JavaScript qui intercepte les requêtes réseau et décide si la page demandée doit être prise dans le cache ou chargée à partir de la connexion réseau (si elle existe).

## Pas de panique :

Le fichier «Manifeste » est très simple à créer en utilisant un site qui peut le générer pour vous (Ex : https://app-manifest.firebaseapp.com/ ). C'est un fichier au format JSON.
Il y a beaucoup d'exemples de « Service Worker » sur internet. Il suffira d'en récupérer un et de l'adapter à ses besoins.

## Vanilla JS :

Il est très compliqué de trouver un tuto PWA pour « Vanilla JavaScript » …
La plupart des tuto s'appliquent au monde du NodeJS, avec tous ses outils associés (WMR Google, Polymer, React, ReactNative, Vue.js, Express, …). Ces outils demandent une spécialisation du programmeur.

**Nous resterons très attachés au JavaScript pur (Vanilla JS), sans autre outil de développement.**
C'est le meilleur moyen de comprendre les mécanismes de base qui vous serviront ensuite si vous optez pour une carrière de développeur Web. Il sera alors temps de vous spécialiser sur un des outils de production du marché. Tous ces outils de toute façon exigent que vous connaissiez la base !

Exemple de site d'aide pour les PWA :
https://www.ice-dev.com/comment-faire/comment-ajouter-la-fonctionnalite-pwa-a-votre-site/

Site ayant servi de modèle pour ce TP :
https://levelup.gitconnected.com/build-a-pwa-using-only-vanilla-javascript-bdf1eee6f37a

# 2. C'EST PARTI !

## Testez l'application Web fournie : IndexedDB

→Dispo sur les serveurs habituels (LaboSN, et 78.122.136.24)

Bon ce n'est déjà pas si mal … Mais ce n'est pas encore PWA.

Par contre, on gère déjà l'absence de connexion au serveur pour obtenir les températures : celles-ci sont stockées dans une base de données JS{*indexedDB*} sur le navigateur.

TRAVAIL :
Analyser le code pour vous familiariser avec **IndexedDB**,
Avec le navigateur Chrome : Retrouvez les données stockées en allant dans le « mode développeur F12 », onglet Appli → Base de données indexée

### Installez un service http://localhost ou https

Si votre développement doit être testé sur votre PC, sans serveur https, vous devez installer un serveur http de base sur votre PC :

Soit un Wamp (facile, rapide, classique, mais un peu lourd),
Soit un serveur basé sur NodesJS (Installer NodeJS, NPM, YARN, et serve : Voir le site *Gitconnected* ci-dessus)

Je vous donne aussi accès au site externe https://lycvauv.ydns.eu/ qui vous est accessible en FTP et Mysql. Il s'agit en fait du serveur à l'@IP : 78.122.136.24. Le https ne fonctionne que si vous utilisez l'url.

Votre travail : préparez l'environnement de votre choix en suivant les instructions ci-dessus.

## 3. TRANSFORMATION DE L'APPLICATION EN PWA

Etant donné que l'application permet d'écouter des radios internet, on se doute bien que ce ne sera pas possible hors connexion …
Mais le PWA nous garantira que l'on peut ouvrir l'application même si le site web qui l'héberge est hors d'atteinte car toutes les pages de l'application seront préchargées dans un cache spécifique à l'application.
D'ailleurs, une fois l'application PWA installée, vous constaterez dans votre menu « Application » que celle-ci occupe de la place sur votre stockage.

### Créez votre icône d'application

Trouvez une image carrés de votre chois et retravaillez-la avec un logiciel de dessin (MS Paint ou autre) pour obtenir :

- 1 fichier PNG de 512px par 512px
- 1 fichier PNG de 100px par 100px
- 1 fichier PNG d'environ 144 à 199px de côté.

### Créez votre MANIFEST.JSON

Créez un fichier dans votre dossier d'application, en l'appelant « manifest.json »
Copiez le texte JSON suivant :

```
{
  "name": "Mon appli PWA",
  "short_name": "MonApp",
  "theme_color": "#2196f3",
  "background_color": "#2196f3",
  "display": "standalone",
  "start_url": ".",
  "icons": [
    {
      "src": "icons/icon.png",
      "sizes": "100x100",
      "type": "image/png"
    },
     {
      "src": "icons/icon144.png",
      "sizes": "144x144",
      "type": "image/png"
    }

  ],
  "splash_pages": null
}
```

Modifiez le manifeste pour l'adapter à votre application et à vos icones.

## Relier l'application au manifeste :

Dans votre fichier **index.html** de votre application, ajoutez dans le <head> la ligne suivante :

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

Tester dans les outils de développement (rubrique Application) que le manifeste est bien reconnu.

## Mettre en service le « Service Worker »

On vous donne un Service Worker « standard »

Dans votre **index.js**, la première chose à faire est de le démarrer, en ajoutant ces lignes :

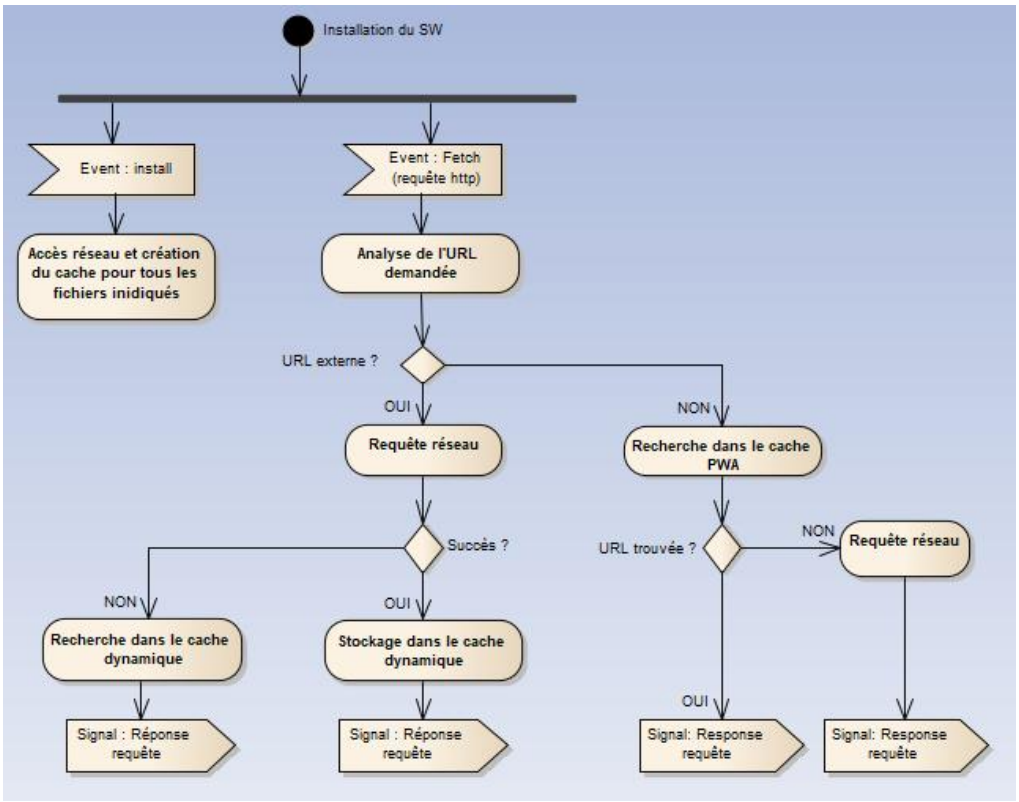
```
window.addEventListener('load', ()=>{

// Inscription du service worker
      if('serviceWorker' in navigator){
            try {
              navigator.serviceWorker.register('./serviceWorker.js');
            } catch (error) {
              console.log("Service Worker Registration Failed");
            }
      }
      else
            console.log("No SW");
});
```

## Analyse du code du Service Worker (SW)

On rappelle le principe : Le SW est exécuté en arrière plan, il intercepte toutes les requêtes réseau, à la façon d'un proxy. Il décide ensuite s'il va utiliser le cache de l'application, ou s'il va effectuer une requête réseau.

Le diagramme d'activité suivant détaille les opérations :



Code exemple :

```
const staticAssets=[
    './',
       './index.html',
    './style.css',
    './index.js',
    './images/background.jpg',
       './sounds/confetti.mp3',
       './manifest.json',
       './images/icons/icon144.png',
       './images/icons/icon.png'
];

//////////////////////////////////////////////////////////////////////
// Installation du Service Worker
//(événement produit automatiquemet lors de l'inscription dans index.js)
//////////////////////////////////////////////////////////////////////
self.addEventListener('install', async event=>{
       console.log("SW installé !");

       // Sauvegarde en cache perso des éléments importants du site
    const cache = await caches.open('cacheRadio');
    cache.addAll(staticAssets);
});

////////////////////////////////////////////
// Comportement du SW pour chaque requete http
////////////////////////////////////////////
self.addEventListener('fetch', event => {
       const req = event.request;
    const url = new URL(req.url);
       console.log('url:',req.url, 1, url.pathname);



       if(url.origin === location.origin){
              // Ressources internes au site : cache d'abord
        event.respondWith(cacheFirst(req));
    } else {
              //Ressource externe au site : reseau d'abord
        event.respondWith(newtorkFirst(req));
    }
});

////////////////////////////////////////////////////
// Recherche dans TOUS les caches, sinon fetch reseau
////////////////////////////////////////////////////
async function cacheFirst(req){
    const cachedResponse = await caches.match(req);
       console.log('url:',req.url,2);
       console.log("Dans un des caches :", cachedResponse);
    return cachedResponse || fetch(req);
}

//////////////////////////////////////////////////////////////
// Recherche sur le réseau et enregistre dans le cache dynamique
//////////////////////////////////////////////////////////////
async function newtorkFirst(req){
    const cache = await caches.open('dynamic-cache');
console.log(3);
    try {
        const res = await fetch(req);
              console.log("Réseau OK, Mis en cache dynamique : ", res);
        cache.put(req, res.clone());
        return res;
    } catch (error) { // Pas de réseau, on puise dans le cache dynamique
              let rep = await cache.match(req);
              console.log("Hors réseau, Essai dans cache dynamique :", req.url, " reponse:", rep);
        return rep;
    }
}
```

Votre travail :
Analysez le code en regard du diagramme d'activité. Ensuite :

- Vérifiez que la liste des fichiers à inclure dans le cache PWA est correcte
- Vérifiez dans les outils de développement que votre Service Worker est bien en activité (vert)

## Installez l'application

Avec Chrome Windows, si vous démarrez votre application à partir d'un serveur HTTPS, vous verrez apparaitre un bouton qui vous invite à installer l'application. Sur Android, le message apparait en bas de l'écran.