Progressive Web App et service-worker

Table des matières

I - Progressive Web Apps (présentation)

1. PWA (Progressive Web App) – aperçu général

1.1. Présentation des "progressive web apps"

Les "**Progressive Web Apps**" (PWA) sont des **applications web modernes (html/css/js)** , ***pouvant fonctionner*** en mode "***hors connexion***" et axées sur les **mobiles** .
Ces applications "PWA" sont censées rivaliser avec des applications mobiles (natives ou hybrides) .

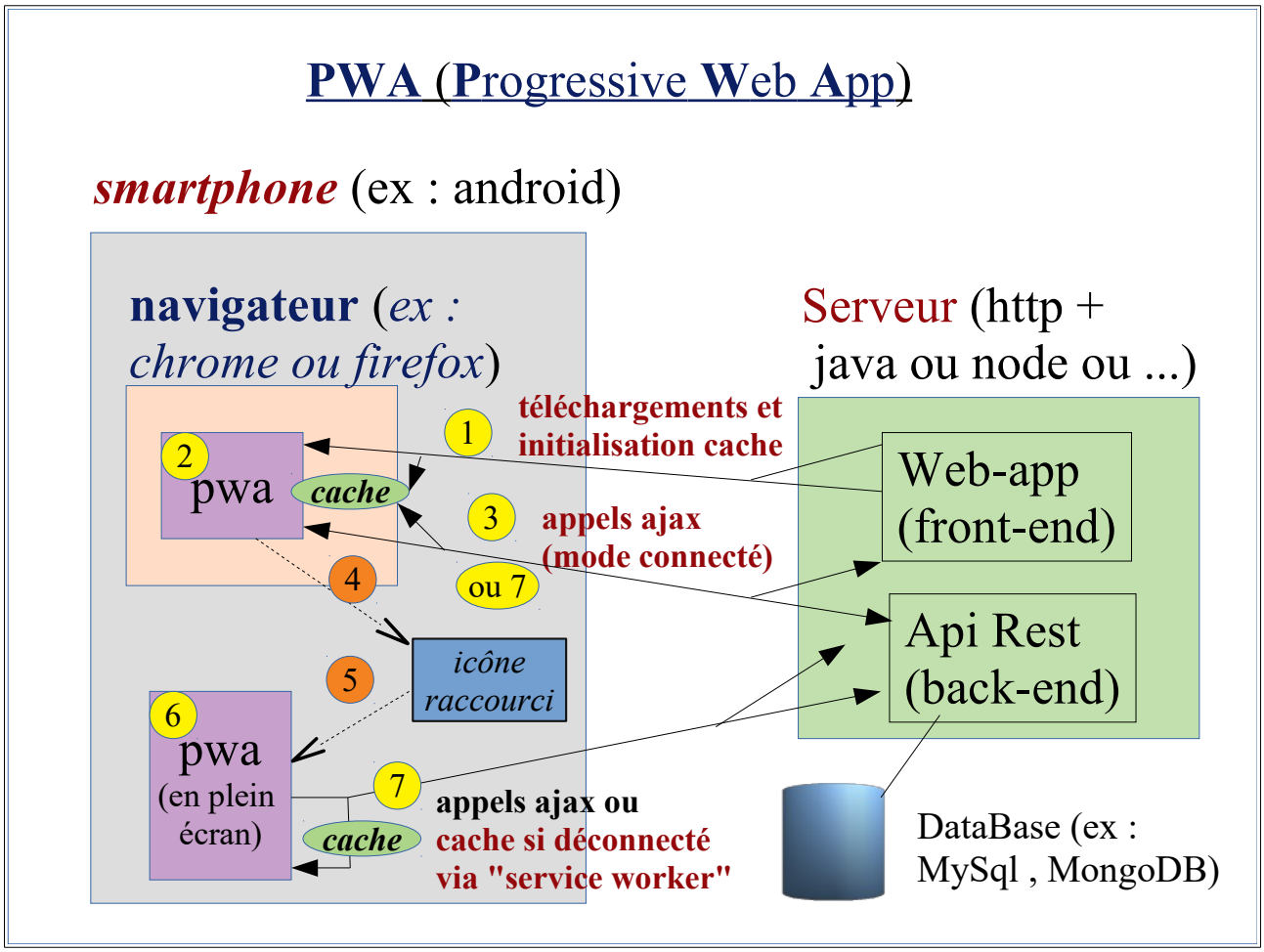
Quelques comparaisons :

	Développement	**Exécution**
Application mobile **native** (ex : java pour Android)	Langage et api spécifique à une plateforme mobile (ex : java et android-sdk ou bien ios/iphone/swift)	application mobile (à télécharger depuis un "store") et à exécuter sur un type précis de smartphone (android ou iphone)
Application mobile **hybride** (ex : *cordova* et/ou *ionic*)	Code source en html/js relativement portable (android , iphone, ...) mais nécessitant une étape de construction qui est moyennent simple avec android et délicate avec ios/iphone	même environnement d'exécution et comportement qu'une application native (avec néanmoins des performances et fonctionnalités quelquefois un peu plus limitées)
PWA (Progressive Web App)	Code source en html/css/js très portable ne nécessitant pas de phase de construction complexe et dépendante d'une plateforme mobile	Une "pwa" s'exécute au sein d'un navigateur de smartphone mais avec le "plein écran" possible et d'autres spécificités. Les fonctionnalités techniques d'une "pwa" seront donc liées aux versions (idéalement récentes) des navigateurs.

Le concept de "pwa" est très moderne et pour l'instant assez peu répandu en 2018/2019 .
L'essor prévu des "pwa" dépendra essentiellement du support offert par les navigateurs des smartphones . Pour l'instant "android , chrome , firefox" avancent clairement dans cette direction .
Apple (ios/iphone) comporte un navigateur supportant partiellement les "pwa" . On verra dans l'avenir les choix stratégiques d'Apple (ouverture ou fermeture) .

Dans "Progressive Web App" le terme "progressif" signifie que l'utilisateur d'un smartphone peut de manière très progressive :

- **naviguer sur internet et utiliser l'application web comme un site ordinaire**
- utiliser potentiellement l'application en mode "**plein écran**" (comme une appli mobile)
- **générer un "icône/raccourci" de lancement sur le fond du bureau** (pour relancer ultérieurement cette appli en mode "navigateur pas encore ouvert")
- **utiliser certaines parties de l'application en mode "déconnecté"** grâce à des "***service -workers***" (*tâches de fond* pouvant faire office de "*cache*" ou "*proxy*")
- **finalement utiliser l'application web aussi facilement que si elle était native** et *disposer en prime d'une réactualisation (mise à jour) automatique du code* (vers la dernière version en date).



1.2. Fonctionnalités d'une "progressive web app"

Une "progressive web app" est avant tout une bonne "web app" s'utilisant bien sur divers navigateurs (PC/desktop, smartphone, tablettes) et qui a en plus certaines fonctionnalités spécifiques au contexte mobile (en mode quelquefois déconnecté) .

Fonctionnalités d'une bonne "web app" :

- **Responsive**
- Lightweight
- Géolocalisation (via navigator.geolocation HTML5)
- etc (intuitive, ergonomique, efficace , ...)

Fonctionnalités *supplémentaires* d'une "progressive web app" :

- **Web App Manifest** (Add to Home Screen)
- **Service Worker**
  - ***cache***
  - ***notifications***
- ....

1.3. Paramétrage fondamental pour "responsive" sur smartphone

index.html

```
<html>
<head>
<title>my pwa</title>
 <link rel="stylesheet" href="./css/bootstrap.min.css">
 <link rel="stylesheet" type="text/css" href="css/styles.css" />
 <link rel="icon" href="./img/pwa-icon_48_48.png" /> <!-- favicon.ico by default -->
 <link rel="manifest" href="manifest.json" />
 <base href="/" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
                <!-- for responsive bootstrap on mobile device -->
</head>
...
</html>
```

La ligne

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

est indispensable pour obtenir un bon comportement "responsive" sur un appareil mobile (ex : smartphone ou tablette) .
Sans celle-ci , ce qui fonctionne bien au sein d'un navigateur sur PC/desktop , ne fonctionne malheureusement pas bien sur un téléphone "android" (c'est tout petit ou bien ça déborde) .

1.4. Quelques exemples d'applications web en mode "PWA"

La plupart des sites web d'actualités fonctionnent aujourd'hui en mode "pwa" (avec cependant plus ou moins de soins apportés au mode "off-line" quelquefois inopérant) :

- le figaro
- le monde
...

1.5. Emulateur android permettant de visualiser les effets "pwa" :

- ~~blueStack 4 ne fonctionne pas bien en mode "pwa"~~
- l'émulateur android proposé par défaut avec "android studio" fonctionne bien
- ....

II - WebManisfest , AddToHomeScreen

1. Web App Manifest / add to home screen

1.1. Génération d'un icône "raccourci" en fond d'écran

De façon à ce qu'un navigateur récent (fonctionnant sur un mobile) puisse proposer la génération d'un icône raccourci sur le fond du bureau , il faut référencer un fichier manifest.json (décrivant l'image de l'icône dans différentes tailles) .

Ce fichier appelé "web app manifest" contiendra quelques informations complémentaires (nom de l'application, ...) .

1.2. Fichier "web app manifest"

Le manifest JSON offre un moyen de décrire un point d'entrée de l'application, afin de proposer à l'utilisateur une expérience d'application native lors du prochain lancement de la PWA :

- Icône dédiée sur la Home (lien URL)
- Lancement en plein écran, ...

Ce fichier est à déposer à la racine du serveur (souvent placé à coté du Service Worker).

Exemple :

manifest.json

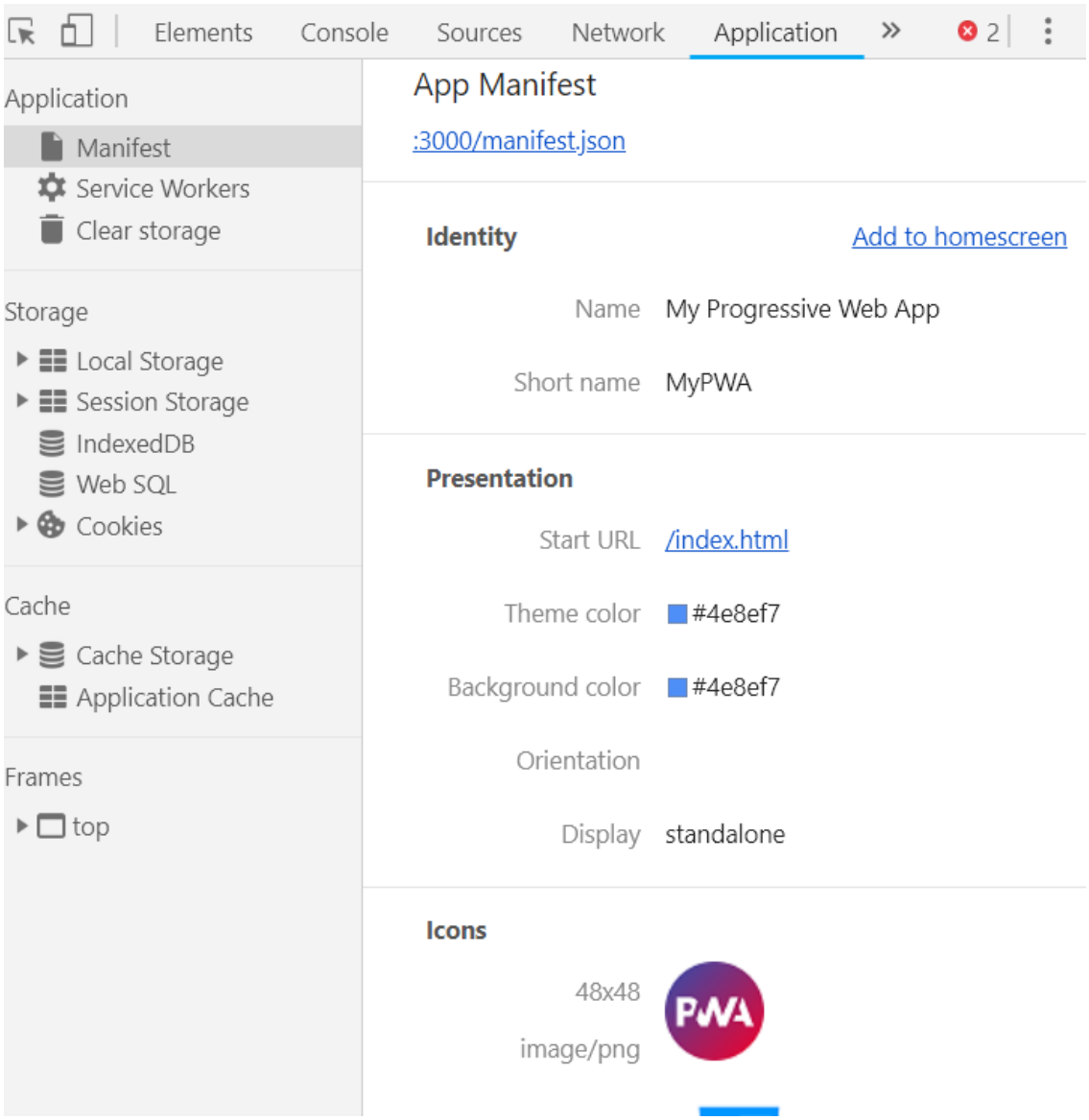
```
{ "name": "My Progressive Web App",
  "short_name": "MyPWA",
  "start_url": "/index.html",
  "icons": [
        {
            "src": "img/pwa-icon_48_48.png",
            "sizes": "48x48",       "type": "image/png"
        },
        {
            "src": "img/pwa_72_72.png",
            "sizes": "72x72",       "type": "image/png"
        },
        {
            "src": "img/pwa-cube_96_96.png",
            "sizes": "96x96",       "type": "image/png"
        },
        {
            "src": "img/pwa-cube_144_144.png",
            "sizes": "144x144",     "type": "image/png"
        },
        {
            "src": "img/pwa-icon_192_192.png",
            "sizes": "192x192"      "type": "image/png"
        }
  ],
  "display": "standalone",
  "background_color": "#4e8ef7",
  "theme_color": "#4e8ef7"
```

```
}
```

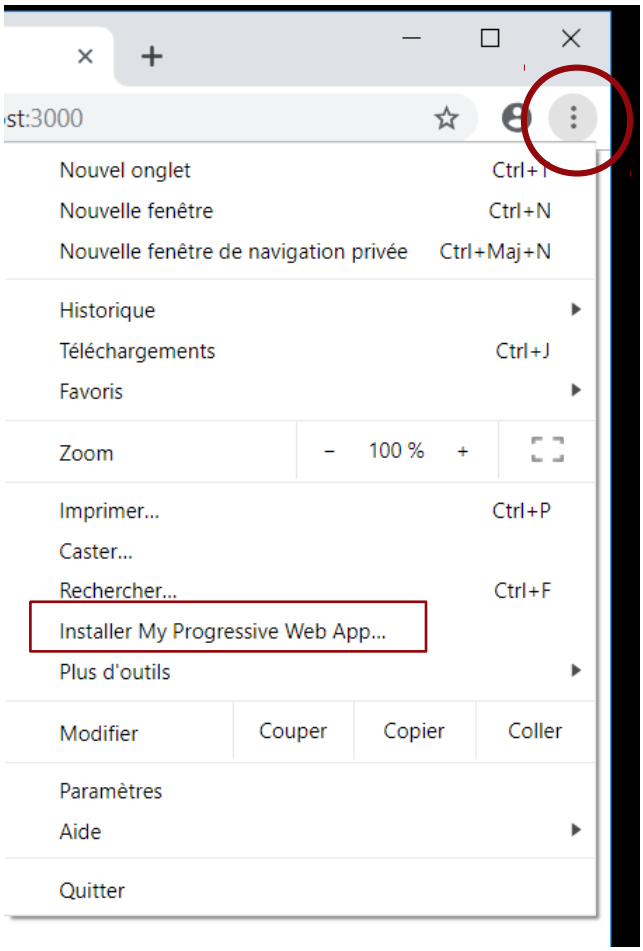
Ce fichier doit être référencé au niveau de **index.html**

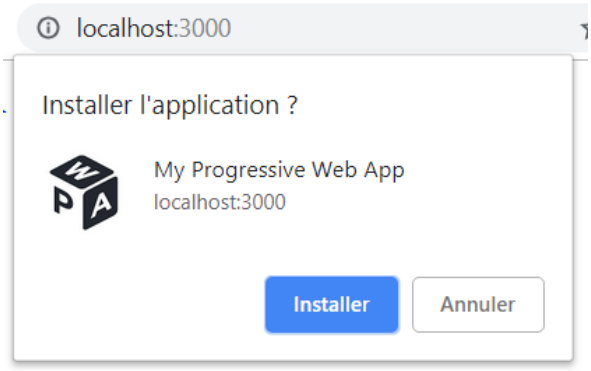
```
<!DOCTYPE html>
<html>  <head>
          <meta charset="UTF-8">
          <title>My progressive Web App</title>
          <link rel="manifest" href="manifest.json">
     </head>
      <body>    ...  </body> </html>
```

Reconnaissance du fichier "***web app manifest***" au sein de la partie "***Application***" de "***outils de développement***" du navigateur "Chrome Desktop" :



1.3. Effets au sein du navigateur "Chrome Desktop" (tests)

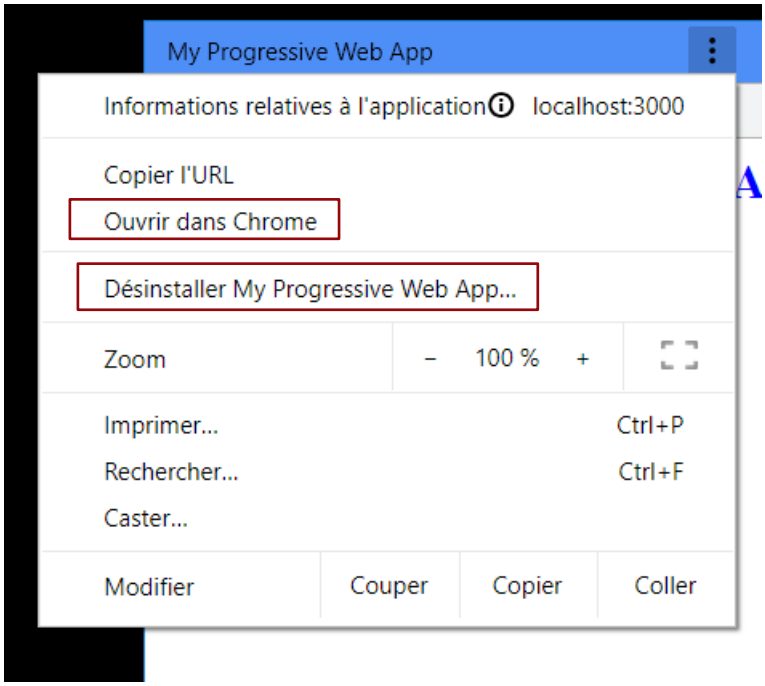




==> cette action génère un icône raccourci sur le fond du bureau :

En cliquant sur ce raccourci , l'application se lance dans une fenêtre spéciale (proche plein-écran) et avec les mêmes fonctionnalités que le navigateur "Chrome" habituel .

Cette application "pwa" comporte un menu contextuel permettant éventuellement la désinstallation de l'application ou bien le lancement de celle ci au sein du navigateur "Chrome" en mode habituel "multi-sites" .



1.4. Attention , pas de "add to home screen" sans service-worker enregistré et gérant les événements "install" et "fetch"

En début de développement (ou de "tp") , il faudra au minimum prévoir le code suivant pour que le navigateur propose le menu "add to home screen" ou "installer la pwa" ou "page/créer un raccourci"

dans index.html (ou client.js) :

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
        navigator.serviceWorker.register('service-worker.js')
       .then(() => navigator.serviceWorker.ready)
       .then(function(registration) {
              // Registration was successful
              console.log('ServiceWorker registration successful with scope: ', registration.scope);
       }, function(err) {
              // registration failed :
              console.log('ServiceWorker registration failed: ', err);
       });
  });
}
```

dans ***service-worker.js***:

```
self.addEventListener('install', function(event) {
console.log('[ServiceWorker] install ***');
```

```
});

self.addEventListener('activate', function(event) {
        console.log('[ServiceWorker] activate ***');
});

self.addEventListener('fetch', function(event) {
        console.log('[ServiceWorker] fetch ***');
});
```

1.5. Effets au sein du navigateur "Chrome pour android"

Lorsqu'une application web est reconnue comme progressive (webmanifest + service-worker + ...) , l'entrée "add to home screen" (ou bien "installer l'application ...") apparaît dans le menu principal du navigateur .

1.6. Effets au sein du navigateur "Firefox pour android"

Lorsqu'une application web est reconnue comme progressive (webmanifest + service-worker + ...) , l'entrée "page / ajouter un raccourcis" (ou bien "....") apparaît dans le menu principal du navigateur . En outre , un icône "maison +" apparaît quelquefois pour signifier un "add to home screen" possible en cliquant dessus .

1.7. Lien ou bouton facilitant l'installation (A2HS)

Pour inviter l'utilisateur à installer l'application en mode A2HS (Add To Home Screen) , il est possible d'ajouter un lien ou bouton qui sera pris en compte par différents navigateurs mobiles (ex : "Chrome pour android" , "Firefox pour android") .

Ce bouton servira simplement à inviter l'utilisateur (via javascript) à gérer l'événement "***beforeinstallprompt***" prévu pour demander l'installation en fond de bureau .

Ce bouton soit idéalement être invisible avant la réception de l'événement .

Exemple de code :

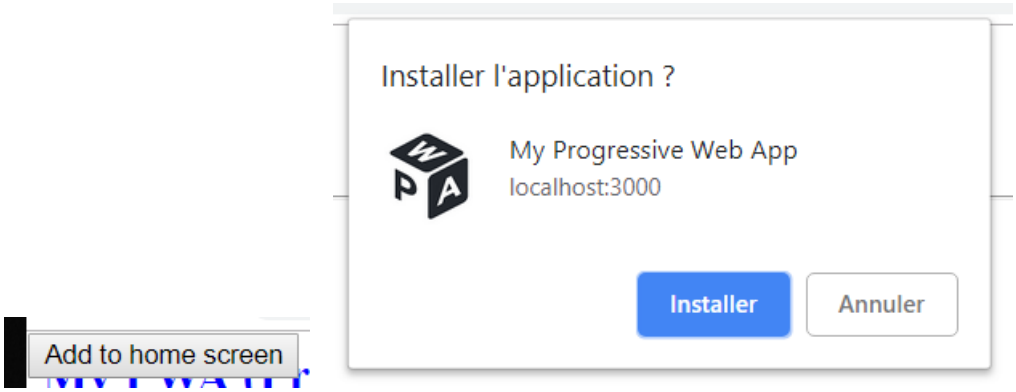
```
<button class="add-button">Add to home screen</button>
```

```
.add-button {
  position: absolute;
  top: 1px;
  left: 1px;
}
```

dans un bloc de script en fin de partie "body" :

```
let deferredPrompt;
const addBtn = document.querySelector('.add-button');
addBtn.style.display = 'none';

window.addEventListener('beforeinstallprompt', (e) => {
  // Prevent Chrome 67 and earlier from automatically showing the prompt
  e.preventDefault();
  // Stash the event so it can be triggered later.
  deferredPrompt = e;
  // Update UI to notify the user they can add to home screen
  addBtn.style.display = 'block';

  addBtn.addEventListener('click', (e) => {
    // hide our user interface that shows our A2HS button
    addBtn.style.display = 'none';
    // Show the prompt
    deferredPrompt.prompt();
    // Wait for the user to respond to the prompt
    deferredPrompt.userChoice.then((choiceResult) => {
       if (choiceResult.outcome === 'accepted') {
         console.log('User accepted the A2HS prompt');
       } else {
         console.log('User dismissed the A2HS prompt');
       }
       deferredPrompt = null;
     });
  });
});
```

Effets (en mode "pré-test") avec "Chrome-Desktop" sous windows :



NB :

L'événement "***beforeinstallprompt***" ne sera déclenché par un ***navigateur mobile récent*** que si les conditions suivantes sont réunies :

- L'appli (PWA) n'est pas déjà installée
- "user engagement heuristic" (l'utilisateur a activement utilisé l'appli au moins 30s)
- L'appli (pwa) comporte le fichier "web app manifest".
- L'appli (pwa) est servie par un serveur sécurisé (https).
- Au moins un "service worker" est enregistré avec un gestionnaire pour l'événement fetch .

Nb : A des fin de "pré-test" avec "Chrome-desktop" on pourra éventuellement (en http et pas https) , rendre visible le bouton dès le début (pas de addBtn.style.display = 'none') .

III - ServiceWorker (principes , essentiel)

1. Service Worker (essentiel)

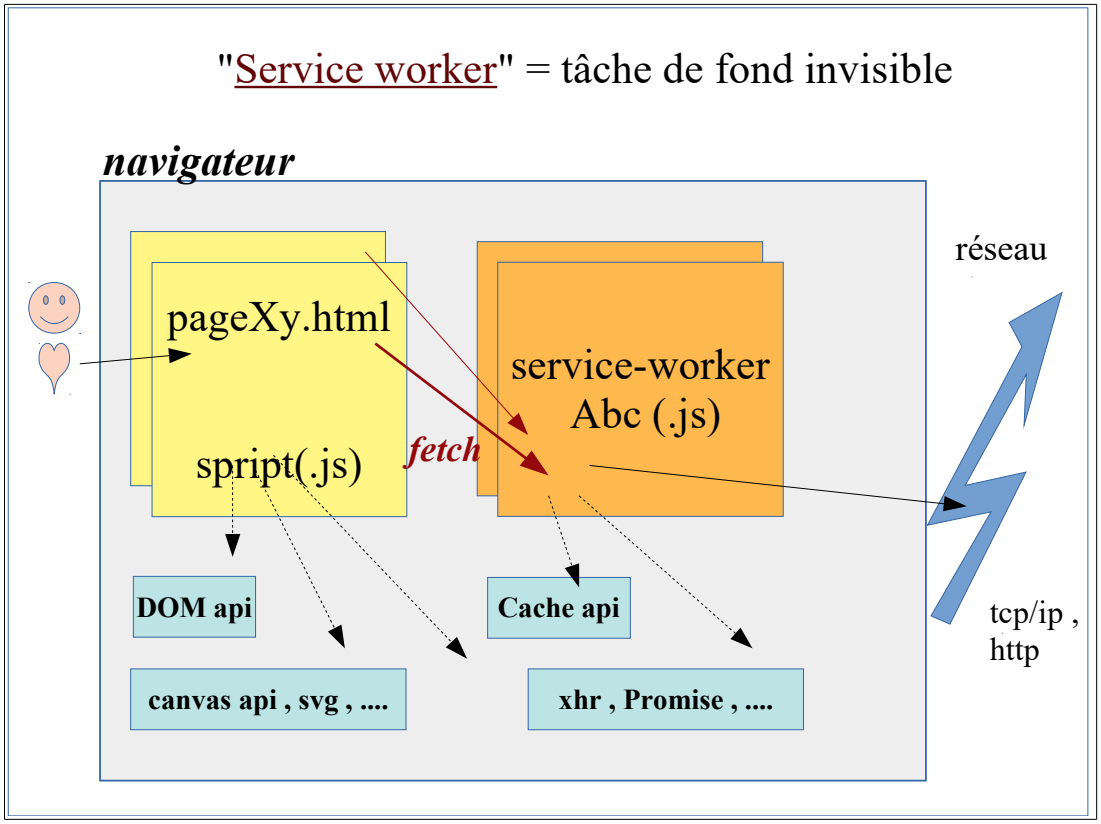
1.1. Présentation des "service worker"

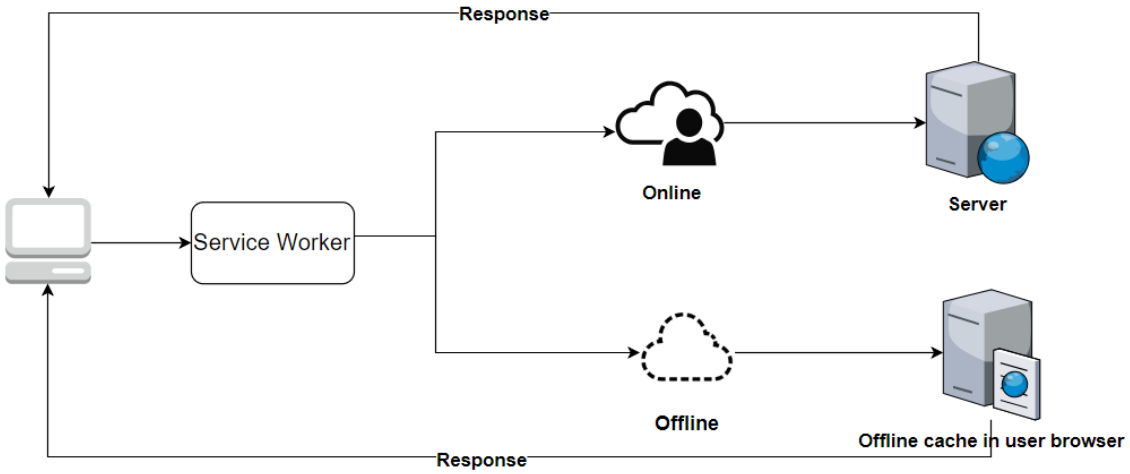
Un "service worker" est un composant complètement invisible qui travaille en tâche de fond et qui n'interagit jamais directement avec l'interface utilisateur (pas de DOM, ...).
Bien que pas directement associé à la structure d'une page HTML , un "service worker" est associé à une partie d'une certaine application web et selon sa portée il sera fonctionnel et accessible depuis une ou plusieurs pages HTML de l'application web (html5/css3/javascript) .

Un "service worker" agira essentiellement comme un proxy (intermédiaire transparent mais à programmer) entre la partie "front end" cliente de l'application web et la fréquente autre partie "back end" (comportant généralement quelques web services "rest" à invoquer via AJAX) .

Ce "service worker" agissant comme un proxy pourra par exemple:
- récupérer certaines données distantes en mode "connexion ouverte"
- les stocker en cache
- restituer ces données en mode "déconnecté / off-line" pour que l'application
puisse au moins afficher quelques choses et fonctionner partiellement lorsque le réseau 4G ou Wifi est inaccessible .

Caractéristiques principales d'un "service-worker" :
- invisible en arrière plan (pas d'accès au DOM)
- asynchrone et non-bloquant (Promise ou ...)
- normalement associé à HTTPS (http déconseillé)





1.2. Portée et enregistrement d'un service-worker

Bien que pas directement associé à une page html spécifique , un service-worker a une **portée** qui est directement liée à l'arborescence des URLs d'un domaine ou d'une application .
Par exemple , un service-worker dont l'enregistrement aura été déclenché par *http(s)://www.xyz.com/my-app/index.html* pourra ensuite être utilisé pour intercepter toutes les requêtes issues des pages suivantes :
http(s)://www.xyz.com/my-app/page2.html

http(s)://www.xyz.com/my-app/foo/pagefoo.html

L'enregistrement d'un service-worker est donc assez souvent effectué depuis la page principale "index.html" .

Exemple d'enregistrement (au sein de index.html) :

```
<script>
if ('serviceWorker' in navigator) {
        window.addEventListener('load', function() {
   navigator.serviceWorker.register('service-worker.js').then(function(registration) {
     // Registration was successful
     console.log('ServiceWorker registration successful with scope: ', registration.scope);
   }, function(err) {
     // registration failed :(
     console.log('ServiceWorker registration failed: ', err);
   });
  });
}
</script>
```

Attention : ne pas confondre "enregistrement / registration" et "(re-)installation" .

La phase d'enregistrement ci-dessus ne fait que demander au navigateur de prendre en compte l'existence du service-worker et de lui associer un domaine (ou portée) .

Dans le cas d'un premier enregistrement , la phase d'installation sera généralement enclenchée juste après par le navigateur.

Dans la cas d'un ré-enregistrement, la phase de (ré-)installation ne sera enclenchée par le navigateur qui si le code du fichier "service-worker.js" a changé depuis l'enregistrement précédent .

NB : il est éventuellement possible d'ajuster la portée (le scope relatif) d'un service worker lors de son enregistrement :

navigator.serviceWorker.register('/worker.js', {scope: '/'})

NB : il ne peut y avoir qu'un seul "service-worker" actif et utilisé par une certaine page html.

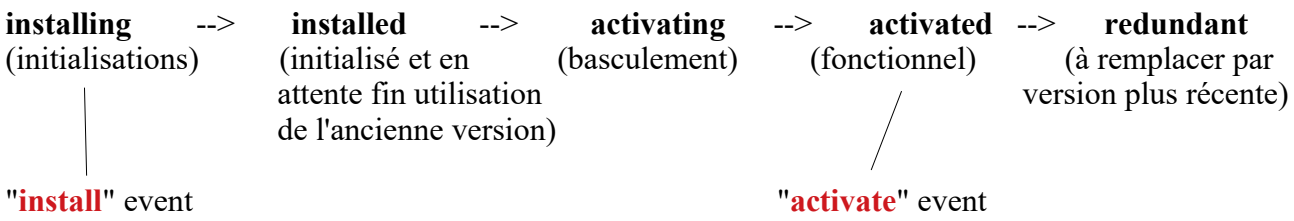
Dans le cas où plusieurs "workers" sont enregistrés (à différents niveaux de l'arborescence) , ce sera le "worker" au plus près de la page active qui sera utilisé .

Un fichier xy-worker.js (enregistré depuis une certaine page) peut cependant faire référence à des sous fichiers "sub-worker-x.js" , "..." via **importScripts()** **et** d'autre part , un worker peut éventuellement fournir de multiples gestionnaires (handler) pour un certain événement (ex : "fetch")

1.3. Cycle de vie d'un "service worker" (événements)

Etant donné que le code d'un "service worker" pourra évoluer au cours du temps (versions de plus

en plus récente) , un cycle de vie élaboré (géré/contrôlé par le navigateur) permettra dans certains cas d'obtenir un basculement contrôlé entre un ancien et un nouveau "service worker" : Attente de la fin des utilisations en cours de l'ancien "service worker" pour pouvoir basculer sur le nouveau .



Phase d'installation (ou de ré-installation) :

Un service worker est un fichier javascript dont l'installation initiale est généralement déclenchée par du code lié à une page html importante (ex : index.html) .
Pour garantir un code régulièrement à jour , un re-téléchargement sera retenté périodiquement (ex : toutes les 24h) et si le code a changé une ré-installation sera alors automatiquement lancée.
Une ré-installation pourra également être déclenchée (si nécessaire , si code modifié) suite à un rechargement de la page html principale .

C'est généralement durant cette phase d'installation (à la réception de l'événement "install") que le service-worker stocke en cache certaines ressources de l'application permettant un ultérieur fonctionnement en mode déconnecté .
Ces ressources peuvent soit être statiques (.html, .css, .js, ...) soit être générées dynamiquement par du code coté serveur (ex : WS-REST java , php ou nodeJs retournant des données "json" ou "...") .

Le nom logique du cache devrait idéalement être de type "my-site-cache-v1" ou "...-v2" , ... de manière à distinguer la version "n" de l'ancienne version "n-1" .

NB : dans le cas d'une ré-installation , la nouvelle version installée (en parallèle) peut temporairement coexister avec l'ancienne version (potentiellement toujours en phase d'exécution) .

Exemple (partie de service-worker.js) :

```
var CACHE_NAME = 'my-site-cache-v1';
var urlsToCache = [
  '/css/styles.css',
  '/js/main.js',
  '/json/sampleData.json'
];


self.addEventListener('install', function(event) {
        console.log('[ServiceWorker] install');
        // Perform install steps
    event.waitUntil(
    caches.open(CACHE_NAME)
      .then(function(cache) {
        //console.log('Opened cache before addAll urlsToCache');
        return cache.addAll(urlsToCache);
```

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

Phase d'activation :

Soit juste après la phase d'installation soit après l'attente de la fin des exécutions des pages html utilisant une ancienne version d'un "service-worker" , la réception de l'événement "**activate**" marque le début de la phase de fonctionnement du "service-worker" .

C'est généralement au moment de la réception de l'événement "activate" que le service-worker pourra supprimer l'ancien cache devenu inutile (en version "n-1") .

Exemple (partie de la version 2 de service-worker.js) :

```
self.addEventListener('activate', function(event) {
  var cacheWhitelist = ['my-site-cache-v2', 'my-other-cache-v2'];
  // delete all existing caches that are not in  cacheWhitelist :
  event.waitUntil(
    caches.keys().then(function(cacheNames) {
      return Promise.all(
        cacheNames.map(function(cacheName) {
          if (cacheWhitelist.indexOf(cacheName) === -1) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

1.4. Fonctionnement d'un "service-worker" activé

Lorsqu'un service-worker a été préalablement installé et activé au sein d'un navigateur , il fonctionnera comme une tâche de fond invisible .

Le principal rôle d'un "**service-worker**" est d'agir comme un "**proxy**" ou "**intercepteur transparent**" au niveau des échanges (requêtes http) qui sont effectués entre une page html et le coté serveur .

Un "service-worker" pourra ainsi intercepter :

- des téléchargements de styles css , images , vidéos , ...
- des téléchargements de fichiers "javascript" (librairies , bundles , scripts, ...)
- des téléchargements de données "json" (statiques ou bien générées dynamiquement via des web services REST)
- autres échanges ("post" , "push" , ...)

Lorsque du code javascript lié à une page html effectue un appel ajax (via XmlHttpRequest) ou bien lorsque suite à l'analyse d'une page html le navigateur a besoin de télécharger des ressources annexes (.css , .js , .json , .jpeg , .png , ...) , un navigateur récent (ex : Firefox ou Chrome) va alors automatiquement déclencher l'événement "**fetch**" au niveau d'un service-worker associé au domaine

(ou site) de la page html .

Lorsque l'événement "**fetch**" est reçu par le service-worker , celui ci a alors (en tant qu'intercepteur) la possibilité de :
- retransmettre (ou pas) la requête au coté serveur (en mode connecté)
- renvoyer directement des fichiers préalablement stockés dans un cache (en mode déconnecté)
- effectuer d'autres tâches

Exemple simple:

Le service-worker élémentaire ci dessous va renvoyer un élément du cache s'il existe ou bien retourner la réponse du coté serveur sinon :

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response
        if (response) {
          return response;
        }
        return fetch(event.request);
      }
    )
  );
});
```

Exemple un peu plus sophistiqué :

Cette version un peu plus sophistiquée (mais encore beaucoup améliorable) agrandi le contenu du cache au fur et à mesure des appels effectués vers le serveur :

```
self.addEventListener('fetch', function(event) {
          console.log('[ServiceWorker] Fetch', event );
          event.respondWith(
    caches.match(event.request)
      .then(function(response) {
        // Cache hit - return response from cache
                  console.log("cache hit for fetch request " + JSON.stringify(event.request));
        if (response) {
          return response;
        }
                  //else standard (network) http request/fetch .

                  // IMPORTANT: Clone the request. A request is a stream and
        // can only be consumed once. Since we are consuming this
        // once by cache and once by the browser for fetch, we need
        // to clone the request.
        var fetchRequest = event.request.clone();

        return fetch(fetchRequest).then(
          function(response) {
```

```
        // Check if we received a valid response
        if(!response || response.status !== 200 || response.type !== 'basic') {
          return response;
        }

        // IMPORTANT: Clone the response. A response is a stream
        // and because we want the browser to consume the response
        // as well as the cache consuming the response, we need
        // to clone it so we have two streams.
        var responseToCache = response.clone();

        caches.open(CACHE_NAME)
          .then(function(cache) {
                                console.log("put request response in cache " +
JSON.stringify(event.request));
            cache.put(event.request, responseToCache);
          });

        return response;
      }
    );//end of fetch(...).then(
  }
 ) //end of caches.matches(...).then(
); //end of event.respondWith
});
```

Principaux événements sur service-worker :

événements	sémantiques
install	début d'installation : le bon moment pour initialiser un cache
activate	début du mode actif : le bon moment pour supprimer ancien cache
fetch	requête à intercepter : on retourne un contenu en cache ou bien on retransmet la requête au serveur ou on effectue une combinaison évoluée .
message	réception d'un message provenant d'un autre script .
sync	synchronisation (maintenant possible , en mode différée) avec le coté serveur lorsque l'on est plus en mode déconnecté : le bon moment pour propager coté serveur des actions effectuées et enregistrées que du coté client .
push	données reçues en mode "***push***" (provenance=coté serveur) . Souvent des notifications

IV - Api workBook pour ServiceWorker/Cache

1. Api "workbox"

De manière à ne pas trop complexifier le code d'un service-worker lorsque les stratégies de gestion du cache sont élaborées (filtrages selon url , mode connecté ou non, ...) , on pourra s'appuyer sur une librairie javascript spécialisée appelée "***workbox***" .

Autrement dit :

- sans l'api "workbox" (ou une api équivalente) , il faut directement coder la gestion des événements "install" , "activate" et "fetch" d'un service worker . C'est une tâche fastidieuse (cohérence à assurer , code asynchrone complexe , plusieurs stratégies à prévoir , ...)

- avec l'api "workbox" (ou une api équivalente) , on a plus besoin de directement gérer les événements "install" / "activate" / "fetch" car c'est déjà pris en charge par cette api de haut niveau. Le seul travail à faire est de paramétrer les éléments à placer (et rechercher) en cache.

NB : L'ancienne version de cette librairie s'appelait "*sw-toolbox*" (https://googlechrome.github.io/sw-toolbox/) , il existait également une api complémentaire "precache" . Ces 2 apis ont fusionné pour donner naissance à l'api "workbox" .

La version récente "workbox" est quant à elle (pour l'instant) accessible au bout de l'url suivante :
https://developers.google.com/web/tools/workbox

Remarque : dans le cas particulier du framework "angular (>=6) " , on pourra utiliser le module "ngsw" déjà intégré à Angular/angular-cli . L'alternative "ngsw" ressemble un peu à "workbox" .

1.1. Principe "route request":

Le principe de fonctionnement de "workbox" consiste à router une "fetch request" vers un gestionnaire "handler" approprié selon certaines règles (correspondances de critères) .

Un "handler" va générer une réponse d'une manière ou d'une autre :
- soit en retournant un contenu présent en cache
- soit en repropageant la requête vers le coté serveur
- ....

La définition (paramétrage) d'une route peut s'effectuer de l'une des 3 manières suivantes :
- comparaison de "string"
- concordance avec "expression régulière"
- filtrage personnalisé via "fonction callback"

Exemples :

```
workbox.routing.registerRoute('/logo.png',handlerQuiVaBien);
workbox.routing.registerRoute('https://some-other-origin.com/logo.png', handler);
```

```
workbox.routing.registerRoute( new RegExp('.+\\.js$'), jsHandler);
```

```
workbox.routing.registerRoute( new RegExp('.+\\.css$'), cssHandler);
```

```
const matchFunction = ({url, event}) => {
// Return true if the route should match
return false;
};

workbox.routing.registerRoute( matchFunction,  handler);
```

handler prédéfini (**workbox.strategies**)

Strategies	***Caractéristiques / comportements***
Stale While Revalidate	Cette stratégie renvoi le contenu du cache si il existe et en tâche de fond tente un appel réseau pour actualiser si possible le contenu du cache.
Network first	Essai d'abord une requête réseau. Il elle réussi , le cache est alimenté avec la réponse également retransmise à l'appelant. Si le réseau est inaccessible , la dernière valeur stockée en cache est retournée
Cache first	Tente d'abord de retourner un contenu en cache si il existe , sinon une requête réseau est déclenchée et le cache est également mis à jour.
Network only	force la réponse à provenir d'un appel réseau/serveur
Cache only	force la réponse à provenir d'un cache

Syntaxe de déclenchement :

```
workbox.routing.registerRoute(  match,  workbox.strategies.staleWhileRevalidate() );
workbox.routing.registerRoute(  match,  workbox.strategies.networkFirst() );
workbox.routing.registerRoute(  match,  workbox.strategies.cacheFirst() );
workbox.routing.registerRoute(  match,  workbox.strategies.networkOnly() );
workbox.routing.registerRoute(  match,  workbox.strategies.cacheOnly() );
```

NB : chaque stratégie peut éventuellement configurée (avec détails) .
Exemple :

```
workbox.strategies.staleWhileRevalidate({
  // Use a custom cache for this route
 cacheName: 'my-cache-name',
 // Add an array of custom plugins (like workbox.expiration.Plugin)
 plugins: [
  new workbox.expiration.Plugin({
      // Cache only 20 items
      maxEntries: 20,
      // Cache for a maximum of a week
      maxAgeSeconds: 7 * 24 * 60 * 60,
    }) , ...
```

```
  ]
});
```

handler personnalisé (via callback):

```
const handler = ({url, fetchEvent}) => {
  return new Response(`Custom handler response.`);
};

workbox.routing.registerRoute(match, handler);
```

2. Precache

NB : Lorsque l'application (dont ***index.html*** est souvent le point d'entrée) est démarrée et que le "service-worker" enregistré a commencé à fonctionner et agir , le contenu des caches s'enrichit petit à petit au fur et à mesure des actions de l'utilisateur (requêtes , navigations, ...) .
Ce s'effectue en appliquant les routes et les stratégies du paragraphe précédent.
Cependant , étant donné que la page "index.html" ne s'appelle pas elle même , certains éléments fondamentaux nécessaires au bon démarrage de l'application pourraient manquer dans le cache et cela empêcherait carrément le fonctionnement ultérieur en mode déconnecté .

On a donc besoin de placer systématiquement certains fichiers en cache (index.html , ...) .
C'est là qu'intervient la fonction ***workbox.precaching.precache***([...]) ;

Exemple :

```
workbox.precaching.precache([
  {
    url: '/index.html',
    revision: 'r2',
  },
  {
    url: '/lib/workbox.3.6.1.js.js',
    revision: 'r1',
  }
]);

workbox.precaching.addRoute();
```

NB : les noms de révisions serviront à distinguer version "courante" et "ancienne(s)" lors d'une mise à jour de certains éléments de l'application : 'r1' à remplacer par 'r2' , 'r2d2' ou 'r3' , ...

==> approfondir si besoin la partie "precache" de l'api "workbox" sur le site officiel (de référence) .

3. Service worker basé api workbox

Plus besoin de gérer directement les événements "install" , "activate" et "fetch" car ceux-ci sont indirectement gérés par l'api workbox .

service-worker.js

```
/*
importScripts('https://storage.googleapis.com/workbox-cdn/releases/3.6.1/workbox-sw.js');
*/
importScripts('./lib/workbox.3.6.1.js');

workbox.precaching.precache([
  {
    url: '/index.html',
    revision: 'r1',
  }
]);
workbox.precaching.addRoute();

workbox.routing.registerRoute(
  new RegExp('.*\.js'),
  workbox.strategies.networkFirst()
);

workbox.routing.registerRoute(
  new RegExp('rest-api-xy/.*'),
  workbox.strategies.networkFirst()
);

workbox.routing.registerRoute(
  // Cache CSS files
  /.*\.css/,
  // Use cache but update in the background ASAP
  workbox.strategies.staleWhileRevalidate({
    // Use a custom cache name
    cacheName: 'css-cache',
  })
);

workbox.routing.registerRoute(
  // Cache image files
  /.*\.(?:png|jpg|jpeg|svg|gif)/,
  // Use the cache if it's available
  workbox.strategies.cacheFirst({
    // Use a custom cache name
    cacheName: 'image-cache',
    plugins: [
      new workbox.expiration.Plugin({
        // Cache only 20 images
        maxEntries: 20,
        // Cache for a maximum of a week
        maxAgeSeconds: 7 * 24 * 60 * 60,
```

```
    })
  ],
 })
);

/*
importScripts('./sub-worker-push-message-sync.js');
*/
```

Attention: **un cache peut en cacher un autre !!!!**

La mise au point de la gestion des caches est assez délicate à cause de la cohabitation temporaire d'un ancien "worker-actif" et d'un nouveau "worker-en-attente-d-activation" .

Pour ne pas perdre sa santé mentale , il est conseillé de :

- ne pas activer les caches en début de développement
- bien vérifier le "worker actif" via la partie "Application / service-worker" des outils de développement de "Chrome-desktop" .
- supprimer si besoin des anciens caches (par exemple via la partie "Application / caches" des outils de développement de "Chrome-desktop").
- ne pas abuser des "bugs" (indirectement introduits par un mauvais paramétrage des caches)

V - message, push , sync (ServiceWorker)

1. Service Worker (push, sync, message)

Rappel des principaux événements sur service-worker :

événements	sémantiques
install	début d'installation : le bon moment pour initialiser un cache
activate	début du mode actif : le bon moment pour supprimer ancien cache
fetch	requête à intercepter : on retourne un contenu en cache ou bien on retransmet la requête au serveur ou on effectue une combinaison évoluée .
message	réception d'un message provenant d'un autre script .
sync	synchronisation (maintenant possible , en mode différée) avec le coté serveur lorsque l'on est plus en mode déconnecté : le bon moment pour propager coté serveur des actions effectuées et enregistrées que du coté client .
push	données reçues en mode "***push***" (provenance=coté serveur) . Souvent des notifications

1.1. Evenement "push"

L'événement "push" d'un service worker sera activé au niveau du service-worker lorsque l'application va recevoir les notifications spontanément envoyées par le coté serveur (après souscription/accord explicite de l'utilisateur) .
Ceci permettra une éventuelle gestion asynchrone/différée de la réception de la notification .

Exemple : ***worker.js***

```
self.addEventListener('push', function(event) {

   // retrieve notification payload :
   var payload = event.data.json();
   console.log('Got push', payload);
   event.waitUntil(self.registration.showNotification(payload.title, {
      body: payload.message,
      icon: './pwa-icon_48_48.png',
      vibrate: [200, 100, 200, 100, 200, 100, 200]
   }));
});
```

Pour que cet exemple puisse fonctionner , il faut proposer à l'utilisateur d'accepter de futures notifications en provenance du coté "serveur/back end" de l'application.
Cet abonnement/souscription aux notifications doit être déclenché par la partie "IHM cliente/html" d'une manière ressemblant au code suivant :

index.html

```
...
<body>
   <h3>Service Worker / push demo</h3>
        <div id="askPushSubscription">
           <input type="button" id="subscribe_btn_id" value="subscribe push" />
   </div>
        <div style="display:none;font-weight:bold;font-style:italic" id="subscriptionDone">
              push subscription is done .
        </div>
        <script src="/client.js"></script>
 </body>
```

client.js

```
// Hard-coded, replace with your public key
const publicVapidKey =
'BG-THsg-ZuqK0np-UlWouA7i38Zhee-obwSKH2Dw7HYDwT6lIKi3l2STtz3TgVDvR0SSfSgWyf4QZk_XJVxsTW4';

//fonction utilitaire (encodage)
function urlBase64ToUint8Array(base64String) {
  const padding = '='.repeat((4 - base64String.length % 4) % 4);
  const base64 = (base64String + padding)
    .replace(/\-/g, '+')
    .replace(/_/g, '/');

  const rawData = window.atob(base64);
  const outputArray = new Uint8Array(rawData.length);

  for (let i = 0; i < rawData.length; ++i) {
    outputArray[i] = rawData.charCodeAt(i);
  }
  return outputArray;
}


var btnSubScribePush = document.getElementById('subscribe_btn_id');
btnSubScribePush.addEventListener("click",function () {
        if ('serviceWorker' in navigator) {
                navigator.serviceWorker.register('/worker.js', {scope: '/'})
                    .then(()=>{
                                console.log('Registered service worker');
                                subscribePush();
                            });
        }
});

function subscribePush() {
console.log('Retrieving PushManager for Subscription');

   // retrieve Service Worker
navigator.serviceWorker.ready.then(function(serviceWorkerRegistration) {

   // subscribe Push Server
```

```
    let pushManager = serviceWorkerRegistration.pushManager;
    pushManager.subscribe({
        userVisibleOnly: true,
        applicationServerKey: urlBase64ToUint8Array(publicVapidKey)
    }).then(function(subscription) {
        console.log('User is subscribed.');
        updateSubscriptionOnServer(subscription); //sous fonction pour initialiser la souscription
    }, function(error) {
        console.error('Failed to subscribe the user', error);
        document.getElementById('subscribe_btn_id').disabled = true; //griser le bouton
    });
},function(error) {
            console.error('Failed to retrieve PushManager', error);
    });
}


function updateSubscriptionOnServer(subscription){
fetch('/subscribe', {
    method: 'POST',
    body: JSON.stringify(subscription),
    headers: {
      'content-type': 'application/json'
    }
  }).then( () => {   console.log('Sent push subscription');
                     document.getElementById('askPushSubscription').style.display='none';
                     document.getElementById('subscriptionDone').style.display='block';
                     })
     .catch( (e) =>  {   console.log(e); }) ;
}
```

Exemple d'implémentation du "push" coté serveur (ex : via nodeJs et web-push) :

dans nouveau répertoire ***pwa_backend*** (ou autre)

```
npm init

npm install --save express@4.16.3 web-push@3.3.0 body-parser@1.18.2 express-static@1.2.5

./node_modules/.bin/web-push   generate-vapid-keys
```

NB : la commande **web-push   generate-vapid-keys** sert à générer un couple clef_privée/clefèpublique nécessaire à la mise en oeuvre du push.
Ces clefs devraient idéalement être stockées dans des variables d'environnement (et surtout être en dehors du code source accessible sur github) . Dans cet exemple simplifié , elle seront directement copiées/collées dans le code source ci-après .

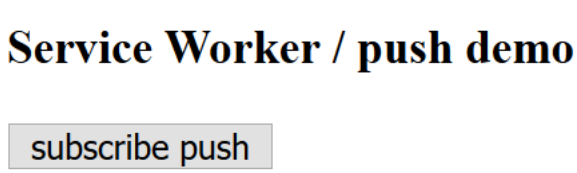
```
=======================================

Public Key:
BG-THsg-ZuqK0np-UlWouA7i38Zhee-obwSKH2Dw7HYDwT6lIKi3l2STtz3TgVDvR0SSfSgWyf4QZk_XJVxsTW4

Private Key:
YH-vyz301BJlcu2XN0H13I8InZgUnbbkVBqRlMp-Xic

=======================================
```

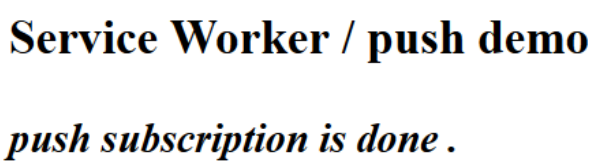
server.js

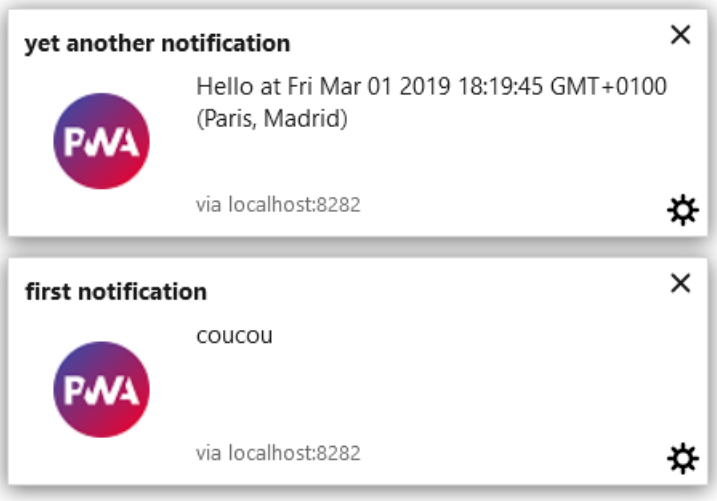
```
//modules to load:
const express = require('express');
const webpush = require('web-push');

const app = express();
//express framework manage basic route in server side with app.get() , app.post() , app.delete() , ...

//NB VapidKeys can be generated by ./node_modules/.bin/web-push generate-vapid-keys

const publicVapidKey =
"BG-THsg-ZuqK0np-UlWouA7i38Zhee-obwSKH2Dw7HYDwT6lIKi3l2STtz3TgVDvR0SSfSgWyf4QZk_XJVxsTW4";
//process.env.PUBLIC_VAPID_KEY;

const privateVapidKey = "YH-vyz301BJlcu2XN0H13I8InZgUnbbkVBqRlMp-Xic";
//process.env.PRIVATE_VAPID_KEY;

// Replace with your email
webpush.setVapidDetails('mailto:xy@zzz.com', publicVapidKey, privateVapidKey);

app.use(require('body-parser').json());

var gSubscription = null;

app.post('/subscribe', (req, res) => {
  const subscription = req.body;
  console.log("received subscription:"+ JSON.stringify(subscription));
  gSubscription = subscription;
  res.status(201).json({});
  const payload = JSON.stringify({ title: 'first notification' , message : 'coucou' });
  console.log(subscription);
  webpush.sendNotification(subscription, payload).catch(error => {
    console.error(error.stack);
  });
  sendAnotherNotificationAfter5s();
});

function sendAnotherNotificationAfter5s(){
 var myIntObj = setTimeout(function () {
        if(gSubscription!=null){
                const payload = JSON.stringify({ title: 'yet another notification' ,
                                                 message: "Hello at " + new Date() });
                webpush.sendNotification(gSubscription, payload)
                      .catch(error => { console.error(error.stack);
                            });
         }
    }, 5000); //déclenchement en différé (dans 5s=5000ms) – cas d'école
}

app.use(require('express-static')('./')); //AFTER OTHER routes to serve static files !!!

app.listen(8282 , function () {
  console.log("simple express node server listening at 8282");
```

```
});
```

(à lancer via node server.js)







Remarques très importantes :

- La technologie de "push" prévue au niveau d'un service-worker est **"web-push" (avec vapid)** et non "~~websocket~~" . Les websockets ont l'inconvénient de ne pas bien s'adapter à l'élasticité et la variabilité d'une architecture cloud .
- La technologie **web-push** nécessite un service intermédiaire (qui va stoker puis relayer) une notification envoyée par un serveur et à destination d'un navigateur client.
- Ce service "cloud" intermédiaire est transparent et spécifique au navigateur utilisé . Il s'agit souvent pour l'instant d'un des 3 suivants :
  - **Firebase cloud messaging (FCM)**
  - **Google cloud Messaging (GCM)**
  - **Apple Push Notifications service (APNs)**
- Bien que transparent (au niveau du code source du serveur et du client) , la passerelle intermédiaire relayant les notifications nécessite un accès internet convenable ("online" , numéro de port "ok" du coté firewall, ...) .
- A la réception d'une notification arrivant en mode "push" , le service-worker utilisera généralement l'api complémentaire "***showNotification***" .

1.2. Evenement "message"

L'événement "message" permet au service-worker de réceptionner des messages quelconques en provenances des clients (pages html) . Via un "MessageChannel" initialisé par un client et véhiculé dans un premier message envoyé , il est possible qu'un service-worker puisse envoyer des messages (dans le sens inverse) vers un client html (voir même vers plusieurs éventuels clients) .

Communication "client -> worker" élémentaire :

Envoi d'un message vers un service-worker :

```
function sendOneWayMessageToWorker(msg){
   navigator.serviceWorker.controller.postMessage("Client says "+msg);
}
```

Réception d'un message au niveau du service-worker :

```
self.addEventListener('message', function(event){
   console.log("SW Received Message: " + event.data);
});
```

Communication bidirectionnelle "client <--> worker" :

coté client (page html) :

```
function callbackDisplayBackMessageSentByWorker(msg){
        console.log("backMessageSentByWorker:" + msg);
}

var msgChannelWithWorker = null;

function sendMessageToWorkerWithReplyCallback(msg,backMsgCallback){
      msgChannelWithWorker = new MessageChannel();
         //MessageChannel existe depuis longtemps dans la plupart des navigateurs.
         //ça peut servir à communiquer entre plusieurs pages html chargées dans différentes "iframe"
         //ça sert ici à communiquer entre un service-worker et une page html)
         //Un objet message channel est un canal de communication en mémoire (pipe)
         //ce que l'on envoi d'un coté (ex: msgChannel.port1) sortira de l'autre coté (ex: msgChannel.port2)

      // Handler for receiving (client side) some reply-message(s) sent back by service-worker
      msgChannelWithWorker.port2.onmessage = function(event){
         if(event.data.error){
            console.log("error when receive backMessage" + event.data.error);
         }else{
            backMsgCallback(event.data);
         }
      };

      // Send message to service worker .
       // Second parameter is array of ports (used worker-side) for sending response:
      navigator.serviceWorker.controller.postMessage("Client says "+msg,
                                                     [msgChannelWithWorker.port1]);
}
```

```
window.addEventListener('load',function() {

        var btnSendMsg = document.getElementById("btnSendMsg");
        btnSendMsg.addEventListener("click",function () {
           var msg = document.getElementById("idMsgToSw").value;
                sendMessageToWorkerWithReplyCallback(msg,
                                                        callbackDisplayBackMessageSentByWorker);
        });

});
```

Coté service-worker :

```
//var arrayOfMessageChannels = [];
var mainMessageChannelPort;

function sendBackMessageToClient(backMsg){
        mainMessageChannelPort.postMessage(backMsg);
}

self.addEventListener('message', function(event) {
         if(event.ports !=null && event.ports[0]!=null){
                  console.log("SW Received message with twoWay channel : " + event.data);
                  mainMessageChannelPort = event.ports[0];
                  sendBackMessageToClient("SW Says a reply message back!");
                  //another sending after 10s = 10000ms :
                 setTimeout(()=>{sendBackMessageToClient("other reply message after 10s")} ,
                           10000);
         }else{
            console.log("SW Received oneWay Message: " + event.data);
         }
});
```

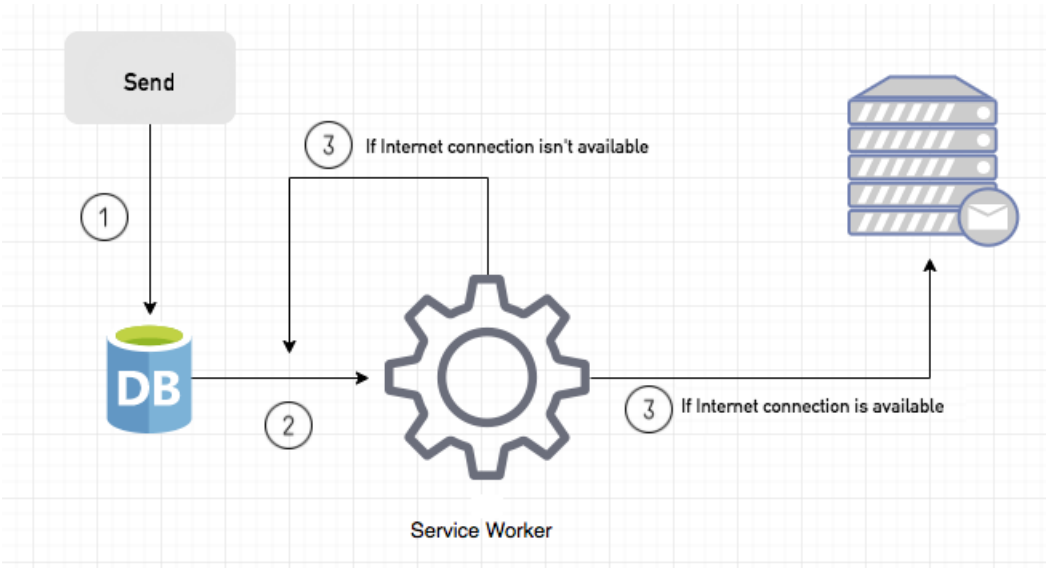
1.3. Evenement "sync"

Attention : l'événement "sync" n'est pas encore géré par tous les navigateurs.
Pour l'instant, il est bien géré via le navigateur "Chrome" . il faudra à court terme trouver des alternatives pour les autres navigateurs (ex : gérer les événements "online" et "offline" du coté page html) .

Evénements proches "oline" et "offline" (du coté "page html") :

```
window.addEventListener('load',function() {

   //nb: la propriété (en readonly) navigator.onLine est tout le temps maintenue à jour par le navigateur
   document.getElementById('onlineOfflineStatus').innerHTML=(navigator.onLine==true)?'online':'offline';

   window.addEventListener('offline', function() {
            //nb: cet événement est déclenché lors d'un changement (mais pas dès le début)
            //exemple : coupure connexion wifi ou ...
           document.getElementById('onlineOfflineStatus').innerHTML='offline';
   });

    window.addEventListener('online', function() {
            //nb: cet événement est déclenché lors d'un changement (mais pas dès le début)
            //exemple : rétablissement connexion wifi ou ...
           document.getElementById('onlineOfflineStatus').innerHTML='online';
    });
}) ;
```

Principes et limitations du "background sync" :



Certaines données saisies dans un formulaire peuvent éventuellement être envoyées au serveur d'une manière "non-immédiate" en cas de réseau déconnecté .
Dans ce cas , les données envoyées sont stockées dans une base de données locales (api "indexedDb") . Lorsque l'événement "sync" sera traité par le service-worker , celui ci aura alors la possibilité de récupérer les données à envoyer dans la base locale (indexedDb) et de les transmettre au serveur (généralement en mode "POST") .

Principale limitation : l'utilisateur n'aura pas un "feed-back" immédiat et sera un certain temps dans l'incertitude concernant la bonne prise en compte des données soumises .

Comportement précis :

Lorsque le coté client_html stocke des données dans la base indexedDB puis enregistre aussitôt une demande de synchronisation (avec tag précis) au niveau du service-worker , il ne se passe rien de plus durant le temps ou le navigateur est en mode déconnecté .

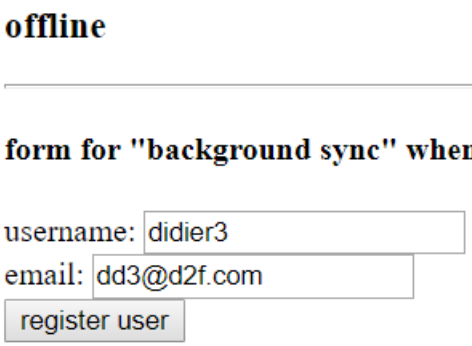
Dès que la connexion internet est rétablie , l'événement "sync" est reçu et traité par le service worker qui peut alors agir en retransmettant au serveur distant les valeurs préalablement stockées en base locale.

Dans le cas particulier où la connexion est "online" dès le début , l'événement "sync" est déclenché immédiatement .

Si l'utilisateur enregistre successivement (en mode "offline") plusieurs demandes de synchronisation avec même nom logique (même tag) et même nom de "objectStore" , alors seule la dernière demande sera retenue (avec dernières valeurs saisies ou bien cumul des valeurs saisies selon le code d'écriture/relecture dans "indexedDB") .

Exemple partiel de page html :

```
<h3 id="onlineOfflineStatus">online or offline</h3>
<hr/>
<h4> form for "background sync" when offline </h4>
<form>
        username: <input type="text" id="idUsername" /> <br/>
        email: <input type="text" id="idEmail" /> <br/>
        <input type="button" value="register user" id="btnRegisterUser" /> <br/>
</form>
<div id="registerMsg"></div>

<script src="/client.js"></script>
```



Exemple de code coté client/js :

```
function registerServiceWorkerSyncOnFormSubmit(registration){
 document.getElementById('btnRegisterUser').addEventListener('click', (event) => {
                                event.preventDefault();
                                saveFormDataInDbPromise().then(function() {
                                        if(registration.sync) {
                                                registration.sync.register('my-registering-sync')
                                                .catch(function(err) {
                                                        return err;
                                                });
                                                console.log('my-registering-sync is register in worker');
                                        } else {
                    // sync isn't there so fallback:
                     console.log("sync is not supported by this browser");
                    document.getElementById("registerMsg").innerHTML=
                                                        "sync is not supported by this browser";
                }
          });
          })
}

function initializeDbForBackroungSync() {
   var myRegisteringDB = window.indexedDB.open('my-registering-db');
   myRegisteringDB.onupgradeneeded = function(event) {
      var db = event.target.result;
      var registeringObjStore = db.createObjectStore("registeringObjStore",
                                                        { autoIncrement: true });
      registeringObjStore.createIndex("username", "username", { unique: true });
      registeringObjStore.createIndex("email", "email", { unique: true });
        console.log('onupgradeneeded ok - registeringObjStore');
   }
        console.log('initializeDB ok - my-registering-db');
}

function saveFormDataInDbPromise() {
   return new Promise(function(resolve, reject) {
      var tmpObj = {
         username: document.getElementById('idUsername').value,
         email: document.getElementById('idEmail').value
```

```
    };

    var myDB = window.indexedDB.open('my-registering-db');
    myDB.onsuccess = function(event) {
      var objStore = this.result.transaction('registeringObjStore', 'readwrite')
                                    .objectStore('registeringObjStore');
      objStore.add(tmpObj);
       console.log("data saved in indexedDB");
      resolve();
    }

    myDB.onerror = function(err) {
       reject(err);
    }
  })
}

initializeDbForBackroungSync();

//********************** serviceWorker.register() for push and sync *************

if ('serviceWorker' in navigator) {
        window.addEventListener('load', function() {
  navigator.serviceWorker.register('/worker.js', {scope: '/'})
        .then(() => navigator.serviceWorker.ready)
        .then(function(registration) {
          // Registration was successful
          console.log('ServiceWorker registration successful with scope: ', registration.scope);
          registerServiceWorkerSyncOnFormSubmit(registration);
      }, function(err) {
          // registration failed :(
          console.log('ServiceWorker registration failed: ', err);
    });
 });
}
```

Exemple de code coté worker :

```
console.log('Loaded service worker!');

function syncToServer() {
   return getIndexedDB()
   .then(sendToServer)
   .catch(function(err) {
      return err;
   })
}

function getIndexedDB() {
   return new Promise(function(resolve, reject) {
      var db = indexedDB.open('my-registering-db');
      db.onsuccess = function(event) {
         this.result.transaction("registeringObjStore").objectStore("registeringObjStore")
                              .getAll().onsuccess = function(event) {
            resolve(event.target.result);
         }
      }
      db.onerror = function(err) {
         reject(err);
      }
   });
}

function sendToServer(userRegistering) {
   return fetch('/registerUser', {
         method: 'POST',
         body: JSON.stringify(userRegistering),
         headers:{
           'Content-Type': 'application/json'
         }
   }).then(function(servResp) {
      return servResp.text();
```

```
  }).catch(function(err) {
     return err;
  })
}

self.addEventListener('sync', function(event) {
        console.log('[ServiceWorker] Sync', event);
        if(event.tag == 'my-registering-sync') {
              event.waitUntil(syncToServer());
  }
});
```

Exemple de code (basique) coté serveur (nodeJs+express):

```
...
app.post('/registerUser', (req, res) => {
        const userRegistering = req.body;
        const userRegisteringAsJsonString = JSON.stringify(userRegistering);
        console.log("received userRegistering:"+ userRegisteringAsJsonString );
        res.status(200).json( userRegisteringAsJsonString );
});
```

NB : dans le cas où le worker du navigateur (ex : firefox pour l'instant) ne supporte pas l'événement sync (et donc pas window.SyncManager) , on pourra prévoir un plan "B" (tenter un envoi immédiat depuis la page sans passer par le "worker" ou bien stoker dans indexedDB des données que la page enverra elle même sur l'événement "online") .

Exemple en ligne pour approfondir ce sujet (plan B / fallback) :

https://davidwalsh.name/background-sync *(explications)*

https://github.com/carmalou/background-sync-example *(code à cloner sur github)*

VI - pwa/serviceWorker intégré dans Angular

1. Service-worker et pwa pour Angular

Les versions récentes (ex : 6, 7) du framework "Angular" de Google prennent maintenant en charge les aspects "pwa" et "service-worker" d'une manière bien intégrée (via "**NGSW**") .

Pour tester les aspects "pwa" et "service-worker" du framework Angular , il faudra :

- effectuer les bonnes configuration (génération via ng add @.../pwa , édition ngsw-config.json, ...)
- construire une version de production via "ng build --prod"
- installer l'application construite sur un serveur http (ex : nginx ou http-server ou ...)
- effectuer des tests via "Chrome Desktop" ou depuis un navigateur mobile (fonctionnant par exemple sur un système Android réel ou simulé/émulé) .

1.1. initialisation "pwa / sw" appli angular

ng add @angular/pwa

Cette commande (de angular-cli) permet d'ajouter tout un tas d'éléments "pwa" et "service-worker" à une application angular :

- ajout du package ***@angular/service-worker*** dans ***package.json*** et configurations associées
- ajout de ***manifest.json*** à la page *index.html* (+ icônes associés dans ***src/assets/icons***)
- création du fichier de configuration ***ngsw-config.json*** (paramétrage des caches)
- le flag ***"serviceWorker" : true*** positionné dans *angular.json* permet de préciser qu'il faudra utiliser le fichier *ngsw-worker.js* comme worker en mode production avec le fichier de paramétrage *ngsw.json* (qui sera généré lors du "*ng build --prod*") .

Attention :

La commande "*ng add @angular/pwa*" fonctionne bien sur une application simple (qui vient d'être générée par exemple par "ng new myapp") mais peut ne pas bien fonctionner sur un projet angular complexe (où beaucoup d'ajout personnalisés ont déjà été réalisés : ng-bootstrap par exemple) .

Il vaut donc mieux déclencher cette commande assez tôt (avant d'ajouter d'autres extensions à package.json) .

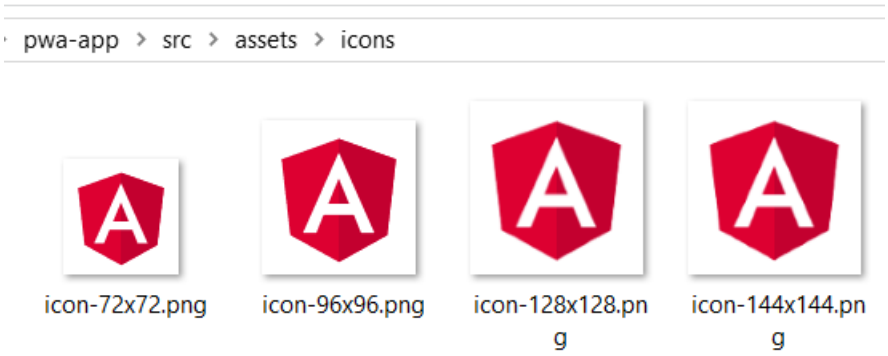
Exemple de dépendance "npm" ajoutée dans ***package.json*** :

```
"dependencies": {
...
"@angular/pwa": "^0.12.4",
...
"@angular/service-worker": "^7.2.0"
}
```

exemple de fichier **manifest.json** généré :

manifest.json ou *manifest.webmanifest*

```
{
    "name": "myapp",
    "short_name": "myapp",
    "theme_color": "#1976d2",
    "background_color": "#fafafa",
    "display": "standalone",
    "scope": "/",
    "start_url": "/",
    "icons": [
        {
        "src": "assets/icons/icon-72x72.png",
        "sizes": "72x72",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-96x96.png",
        "sizes": "96x96",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-128x128.png",
        "sizes": "128x128",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-144x144.png",
        "sizes": "144x144",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-152x152.png",
        "sizes": "152x152",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-192x192.png",
        "sizes": "192x192",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-384x384.png",
        "sizes": "384x384",
        "type": "image/png"
    },
    {
        "src": "assets/icons/icon-512x512.png",
        "sizes": "512x512",
        "type": "image/png"
    }
    ]
}
```



exemple de lien ajouté dans index.html :

```
<link rel="manifest" href="manifest.json">
...
<meta name="theme-color" content="#1976d2">
```

Eléments ajoutés dans le module principal de l'application (***app.module.ts***) :

```
...
import { ServiceWorkerModule } from  '@angular/service-worker';

@NgModule({
declarations: [
AppComponent
],

imports: [
...
ServiceWorkerModule.register('/ngsw-worker.js',
                             { enabled:  environment.production })
],
```

1.2. Configuration du cache (ngsw-config.json)

Le fichier ***src/ngsw-config.json*** (pris en compte lors de la construction d'une application via ***ng build --prod***) permet de préciser quels sont les éléments à placer en cache et les stratégies de mise à jour.

Dans ce fichier les chemins doivent commencer par "/" et seront interprétés en relatif vis à vis de la racine des fichiers sources (placés dans src puis déplacés dans les bundles de production) .

Sauf indication contraire, les expressions des chemins sont basés sur le format "**glob**" (pas de ~~regexp~~):

- * * matches 0 or more path segments.

- `*` matches 0 or more characters excluding `/`.
- `?` matches exactly one character excluding `/`.
- The `!` prefix marks the pattern as being negative, meaning that only files that don't match the pattern will be included.

Example patterns:

- `/**/*.html` specifies all HTML files.
- `/*.html` specifies only HTML files in the root.
- `!/**/*.map` exclude all sourcemaps.

Groupes d'éléments en cache :

assetGroups	fichiers statiques (pour une version précise de l'application) , ex : icônes , images , ... , avec stratégies "*prefetch*" ou "*lazy*"
dataGroups	fichiers (souvent "json") de données fabriqués dynamiquement (souvent via des web services "rest") . paramétrages de type "*maxAge*" , ...

Modes d'installation en cache (pour un des assetGroups):

prefetch : dès le début (au premier démarrage de l'application) , les ressources sont téléchargées et stockées en cache .

lazy : au fur et à mesure des requêtes déclenchées . Les ressources ne sont placées en cache que si elles ont été téléchargées via le fonctionnement normal de l'application piloté par l'utilisateur .

La valeur par défaut de "***installMode***" est "prefetch" .

Modes de mise à jour du cache (pour un des assetGroups):

prefetch : dès le début (au premier rechargement de l'application modifiée) , les ressources modifiées sont téléchargées et stockées en cache .

lazy : au fur et à mesure des requêtes déclenchées . Les ressources ne sont remplacées en cache que si elles ont été téléchargées via le fonctionnement normal de l'application piloté par l'utilisateur .

La valeur par défaut de "***updateMode***" est la valeur de "*installMode*" .

Expression des chemins :

files=... pour les éléments internes à l'application (ex : icônes et petites images dans assets)

urls=... pour les éléments externes à l'application (ex : CDN pour css, fonts, ... et images téléchargées via http)

Paramétrages pour un des dataGroups :

maxSize	nombre maxi d'éléments (réponses) stockés en cache . éviction si taille dépassée
maxAge	durée maxi de validité en cache (ex : 3d12h)
timeout (paramètre facultatif)	durée d'attente d'une réponse réseau (en mode "online" dégradé) avant d'utiliser le contenu du cache en plan B (ex : 5s30u)
strategy	"performance" (pour données évoluant peu) ou "freshness" (pour données importantes à rafraîchir souvent)

unités :

- d: days
- h: hours
- m: minutes
- s: seconds
- u: milliseconds

Exemple de fichier src/**ngsw-config.json** :

```
{
 "index": "/index.html",
 "assetGroups": [
   {
    "name": "app",
    "installMode": "prefetch",
    "resources": {
     "files": [
       "/favicon.ico",
       "/index.html",
       "/*.css",
       "/*.js"
     ]
    }
   }, {
    "name": "assets",
    "installMode": "lazy",
    "updateMode": "prefetch",
    "resources": {
     "files": [
       "/assets/**",
       "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
     ]
    }
   }
 ]
}
```

ajouts classiques :

```
{
 ...
 "dataGroups": [
```

```
  {
   "name": "xy-api",
   "urls": [
          "https://www.mycompany.com/xy"
         ],
     "cacheConfig": {
        "maxSize" : 10000,
        "maxAge" : "7d" ,
        "strategy" : "freshness"  ,
       "timeout" : "5s"
      }
  }, {
   .... (avec "strategy" : "performance" et autre(s) url(s) )
  }
 ]
}
```

1.3. Mise en "pseudo-production" pour effectuer un test

installation (en mode global) via npm du serveur http-server :

```
npm  install  -g  http-server
```

Génération de l'appli avec ses bundles de production dans le répertoire "dist" :

```
ng build --prod
```

Lancement du serveur http avec la prise en charge de l'application angular :

```
http-server  -c-1  dist\myapp\
```

NB : l'option **-c-1** signifie "*désactiver les caches coté serveur http*"
l'url par défaut est *http://localhost:8080*

l'option --proxy http://localhost:8282 ./api-rest-xy permet (si besoin) de configurer une redirection vers une api rest

1.4. Fonctionnalités offertes par ServiceWorkerModule

Les fonctionnalités apportées par le module facultatif *ServiceWorkerModule* sont plutôt secondaires mais peuvent cependant constituer un plus au niveau de l'application .

Via le service **SwUpdate** :

- Savoir si une nouvelle mise à jour est disponible .
- Demander une activation de la mise à jour .
- ...

Via le service **SwPush** :

- ...
- ...

1.5. Notifications sur mises à jour disponibles et activées

```
@Injectable()
export class LogUpdateService {

  constructor(updates: SwUpdate) {

    updates.available.subscribe(event => {
      console.log('current (running) version is', event.current);
      console.log('available (waiting) version is', event.available);
    });

    updates.activated.subscribe(event => {
      console.log('old version was', event.previous);
      console.log('new (activated) version is', event.current);
    });

  }
}
```

autre exemple :

```
...
export class AppComponent implements OnInit {

  constructor(private swUpdate: SwUpdate) {
  }

  ngOnInit(){
    if(this.swUpdate.isEnabled){
        swUpdate.available.subscribe(event => {
                if ( confirm("new version available , would you like to reload it ?") ) {
                        window.location.reload() ;
                }
        });
    }
  }
}
```

et d'autres fonctionnalités de ce genre à étudier sur le site de référence d'angular

https://angular.io/guide/service-worker-communications

1.6. Exemple d'utilisation du service SwPush

...

1.7. Exemple d'enrichissement personnalisé de ngsw

Dans certains cas pointus , il sera peut être nécessaire de faire cohabiter l'implémentation prédéfinie du service-worker de angular avec certaines extensions personnalisées .

Pour atteindre cet objectif , le mode opératoire (à éventuellement ajuster en fonction des besoins) est le suivant :

sw-custom.js *(à ajouter dans src)*

```
(function () {
  'use strict';

  self.addEventListener('...', (event) => {
    ...
  });

}());
```

sw-master.js *(à ajouter dans src)*

```
importScripts('./ngsw-worker.js');
importScripts('./sw-custom.js');
```

Enregistrer le nouveau fichier *sw-master.js* dans les "*assets*" de "*build*" de **angular.json** :

```
...
"assets": [
            "src/favicon.ico",
            "src/assets",
            "src/manifest.json" ,
             "src/sw-master.js"
          ]
...
```

Référencer ***sw-master.js*** plutôt que le prédéfini ***ngsw-worker.js*** dans ***app.module.ts*** :

```
...
ServiceWorkerModule.register('/sw-master.js', { enabled: environment.production })
...
```

ANNEXES

VII - Annexe – DevTools pwa/service-worker

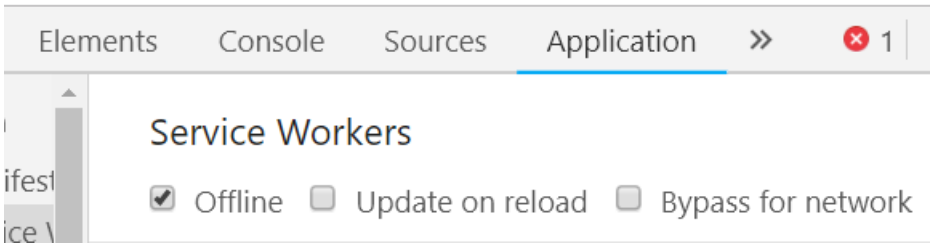
1. Dev tools in Chrome (pwa , service-worker)

Le navigateur "Google Chrome" comporte certains outils intéressants pour contrôler et tester une application "pwa" et son (ou ses) "service-worker(s)" .

Menu "***plus d'outils / outils de développement***" (ou ***"Ctrl"*** + ***"Maj"*** + ***I***)

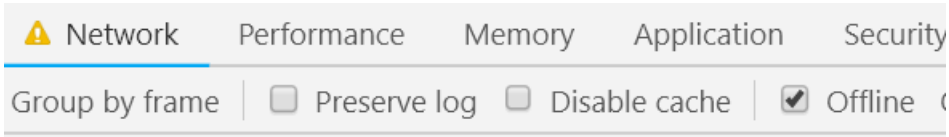
Sélectionner quelquefois l'onglet "***Console***" ou l'onglet "***Network***" .
Sélectionner souvent l'onglet "***Application***" . puis la partie "***service-workers***" .

1.1. Simuler/contrôler une déconnexion internet



cocher (ou décocher) "**offline**" dans l'onglet "***Application***" .

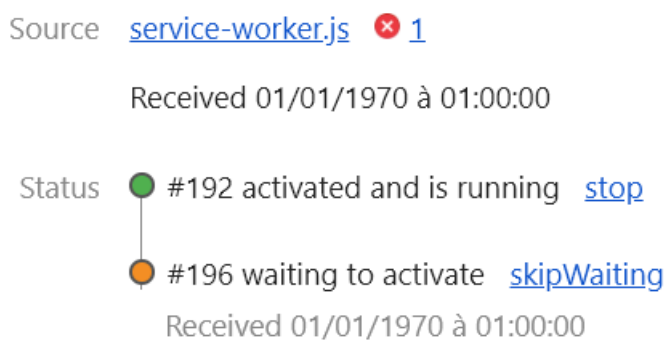
ou bien



cocher (ou décocher) "**offline**" dans l'onglet "***Network***" .

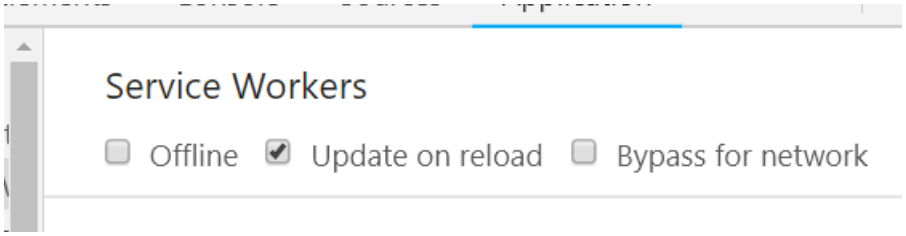
1.2. Changer immédiatement la version active d'un service-worker

Méthode (solution) 1 :



- *stopper éventuellement l'ancienne version (stop)*
- activer immédiatement la nouvelle version (**skipWaiting**)

Méthode (solution) 2 :



et

1) cocher l'option "***update on reload***"
2) effectuer un "refresh" ou "reload" de la page

2. Dev tools in Firefox (pwa , service-worker)

Le navigateur "Mozilla Firefox" comporte pour l'instant très peu d' outils intéressants pour contrôler et tester une application "pwa" et son (ou ses) "service-worker(s)" .

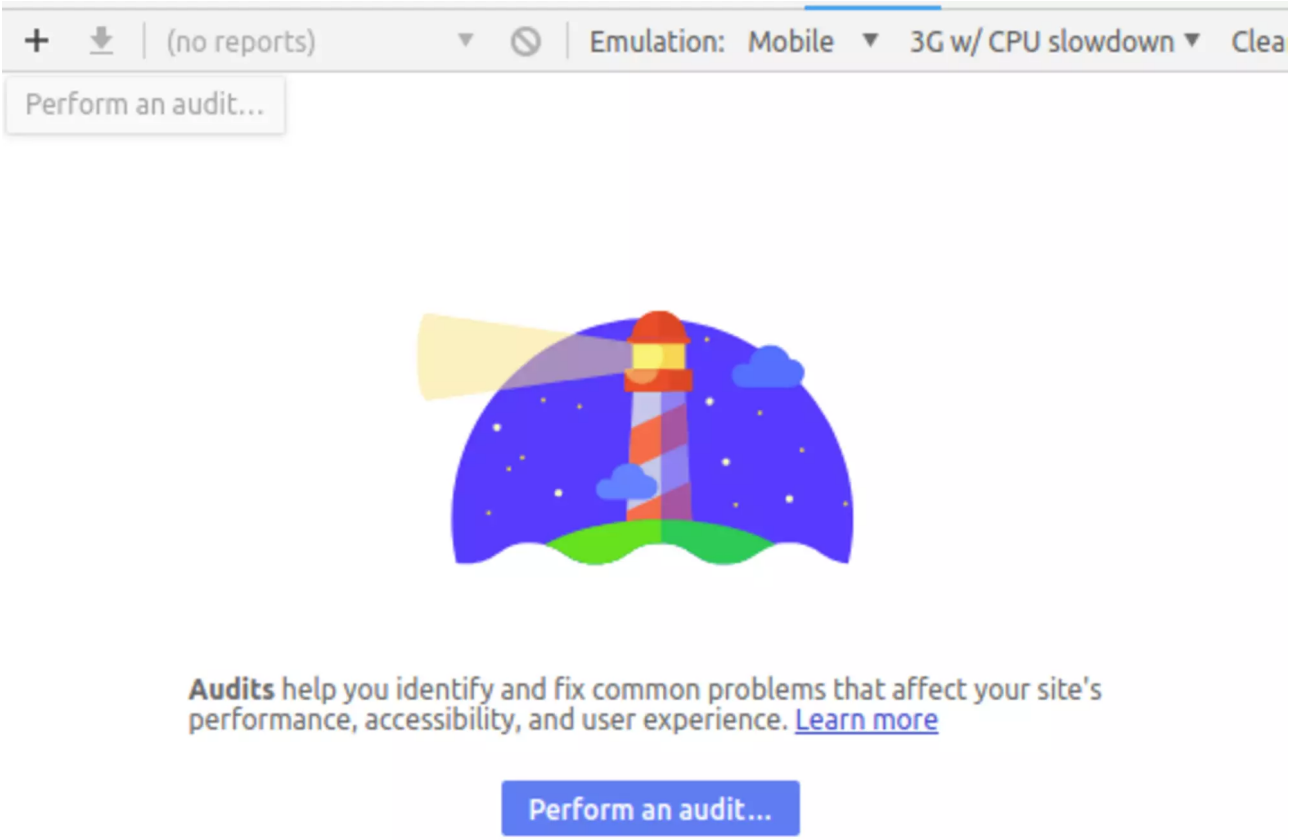
Affaire à suivre (surveiller les prochaines versions) ...

VIII - Annexe – lightHouse / plugin Chrome

1. Audit via Chrome Lighthouse

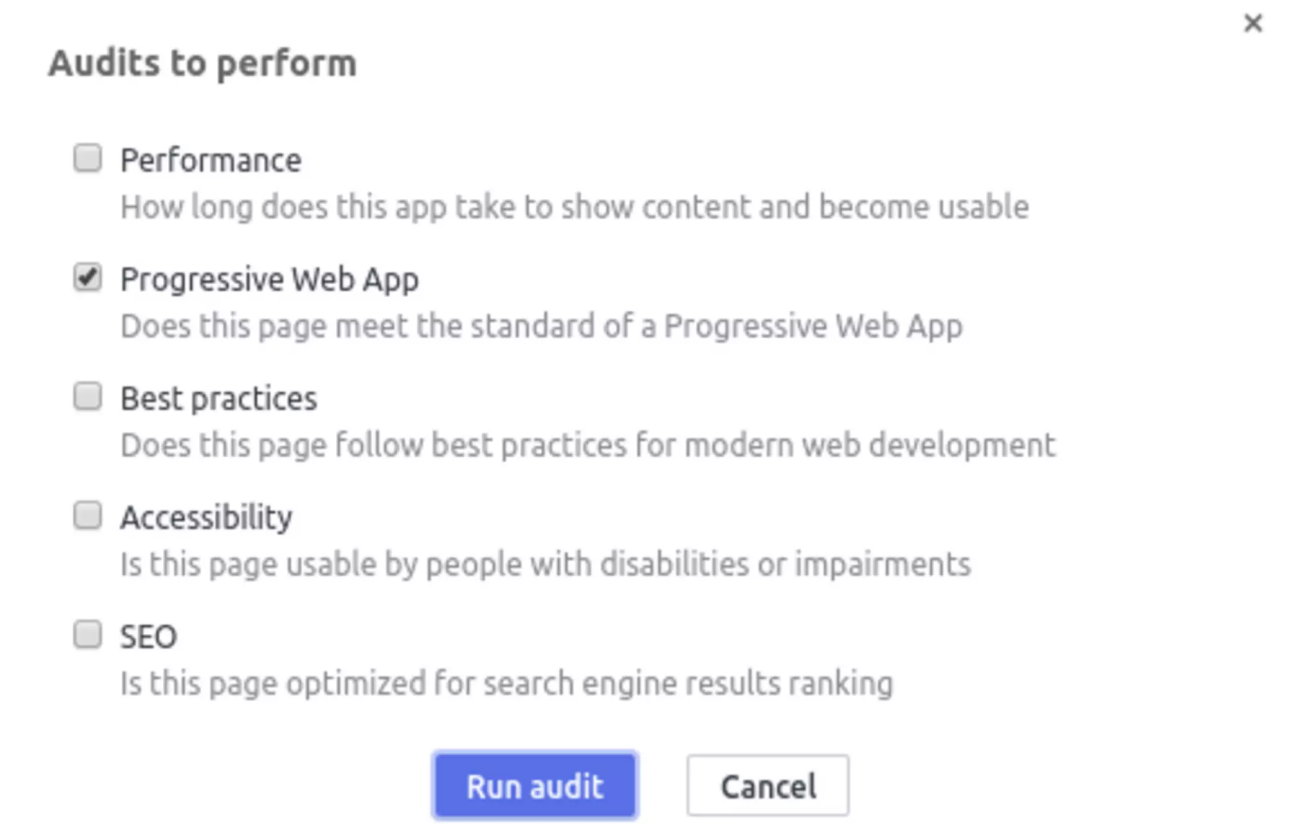
Lighthouse est un **plugin** que l'on peut ajouter au navigateur "**Google Chrome**" de façon à **vérifier à ce qu'une application respecte bien les critères de qualités d'une application "responsive" et/ou "pwa"** .

Une fois installé dans le navigateur "Google Chrome" , l'extension "lighthouse" se comporte et s'utilise comme un nouvel onglet de la partie "outils de développement" .

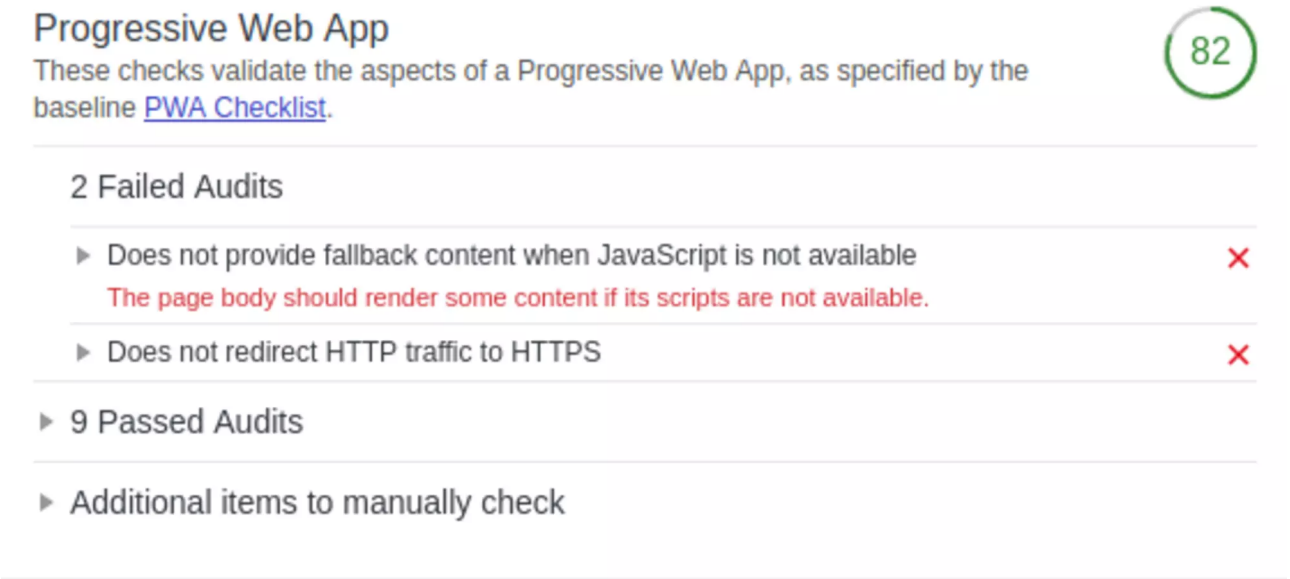


....

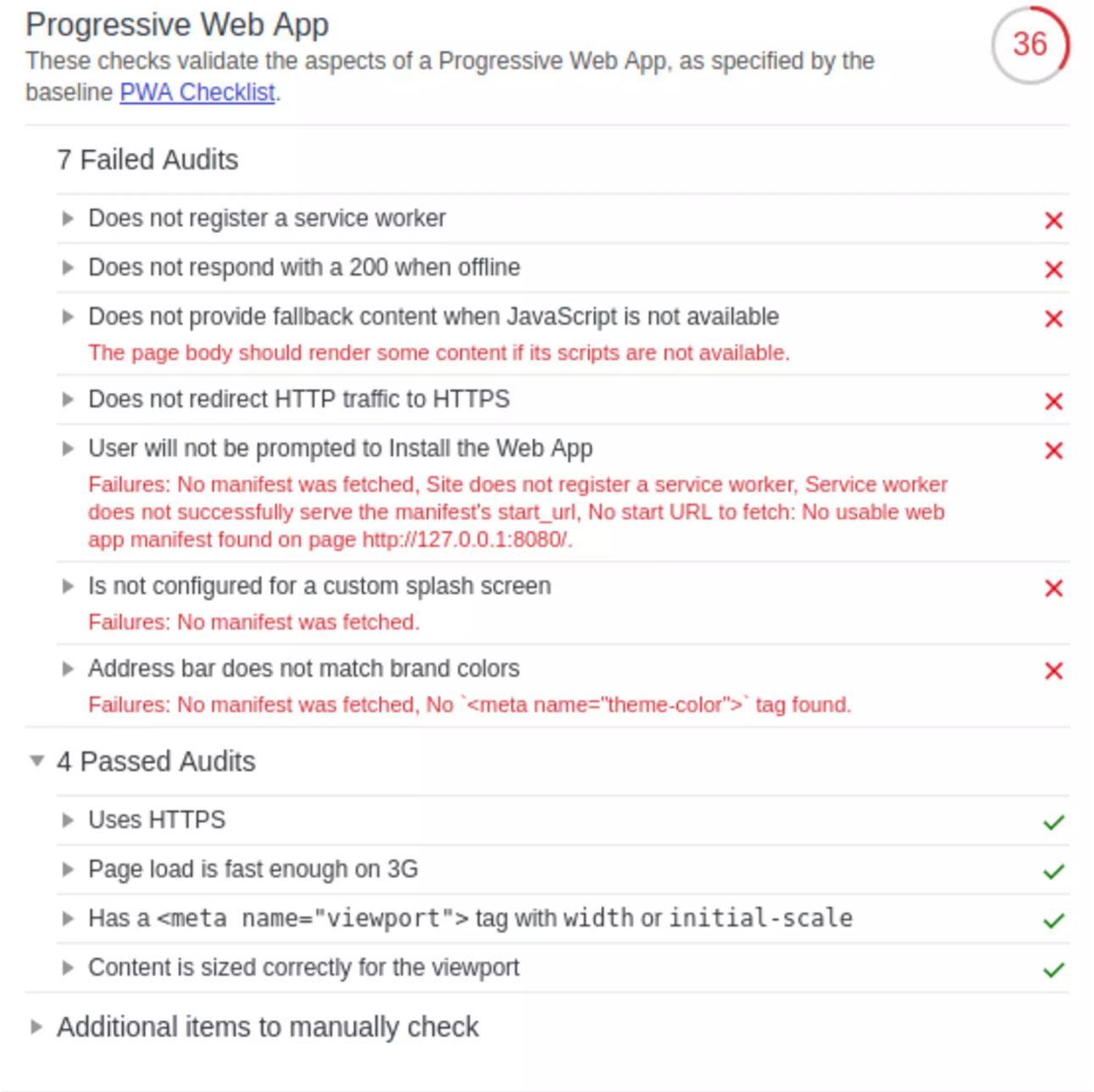
Sélection des aspects à vérifier/auditer :



Exemple d'assez bon résultat :



Exemple d'assez mauvais résultat :



IX - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP