Tutoriel issu de https://blog.eleven-labs.com/fr/votre-premiere-pwa/

Première application pwa …

PSB 2021

Utilisation de nodejs et de git.

Source sous github : https://github.com/CaptainJojo/pwa

Le monde du web évolue. Le site web mobile est devenu le plus grand concurrent aux applications natives, et Google l'a bien compris. Si vous avez suivi la conférence de Google à Amsterdam , vous savez que l'avenir des sites web mobiles sont les Progressive Web Apps (le cas contraire, je vous invite à lire mon précédent article sur cet événement, disponible **ici**){:rel="nofollow noreferrer"}. Ça tombe bien, cet article va vous permettre de mettre en place votre première "PWA".

PRÉ-REQUIS :

Avant de commencer ce tutoriel:

- Je vous invite à mettre à jour ou installer nodejs, toutes les explications pour ce faire sont disponibles **ici**.
- Il vous faudra aussi une version de chrome avancée, telle que Canary, disponible **ici**
- Une fois l'installation de votre chrome Canary, je vous invite à installer l'extension suivante, disponible **ici**
- Durant l'ensemble du tutoriel, nous allons suivre le projet git suivant, disponible **ici**

ÉTAPE 1, L'INSTALLATION :

L'installation du projet est très simple (suivez le README) ou les étapes suivantes :

```
git clone git@github.com:CaptainJojo/pwa.git
a remplacer par
git clone https://github.com/CaptainJojo/pwa
```

Puis allez dans le répertoire pwa et lancez l'installation.

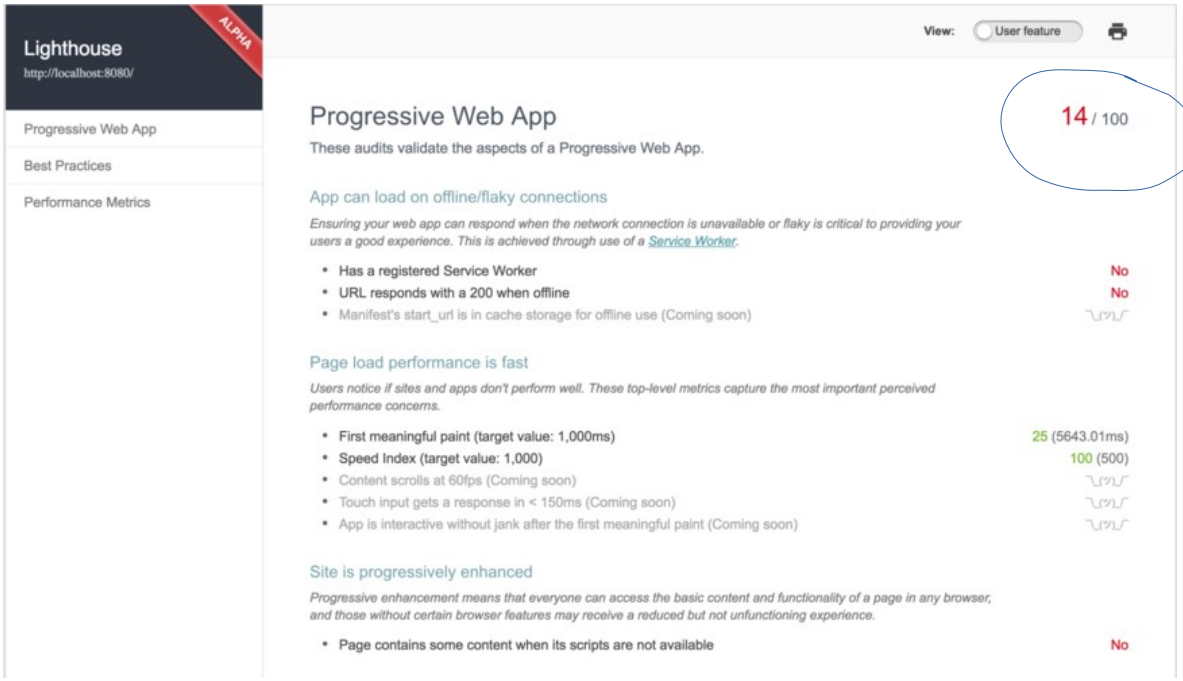
```
cd pwa & npm install
```

Une fois l'installation terminée, vous n'avez plus qu'à lancer le serveur. Attention vérifier que vous n ′avez pas déjà le port 8080 occupé par un autre service.

```
npm start
```

Si tout se passe bien, l'application est disponible sur **cette adresse localhost**.

Vous pouvez alors naviguer dans l'application, son seul but étant d'avoir quelques urls et d'afficher des images (ce qui n'a que peu d'intérêt). Je vous invite à lancer l'extension installée plus tôt. Vous devez arriver sur cette page :



Cette extension génère une note sur 100 pour votre application. Plus la note est proche de 100, plus votre application est une progressive web app. Comme vous pouvez le lire, il y a deux choses principales qui permettent de gagner des points : la mise en oeuvre d'un service worker qui permet d'avoir une application offline et la mise en place d'un manifest notifiant le navigateur que "vous êtes une progressive web app" et donc installable sur votre téléphone. Nous allons commencer par mettre en place le service worker.

ÉTAPE 2, LE SERVICE WORKER :

Le service worker est un simple fichier js à enregistrer dans le navigateur (s'il est compatible), une fois enregistré, il peut lancer du code sans que vous soyez connecté à internet. La première chose à faire est donc d'enregistrer un fichier js dans le navigateur. Nous allons tout d'abord créer un fichier vide sw.js dans le dossier public.

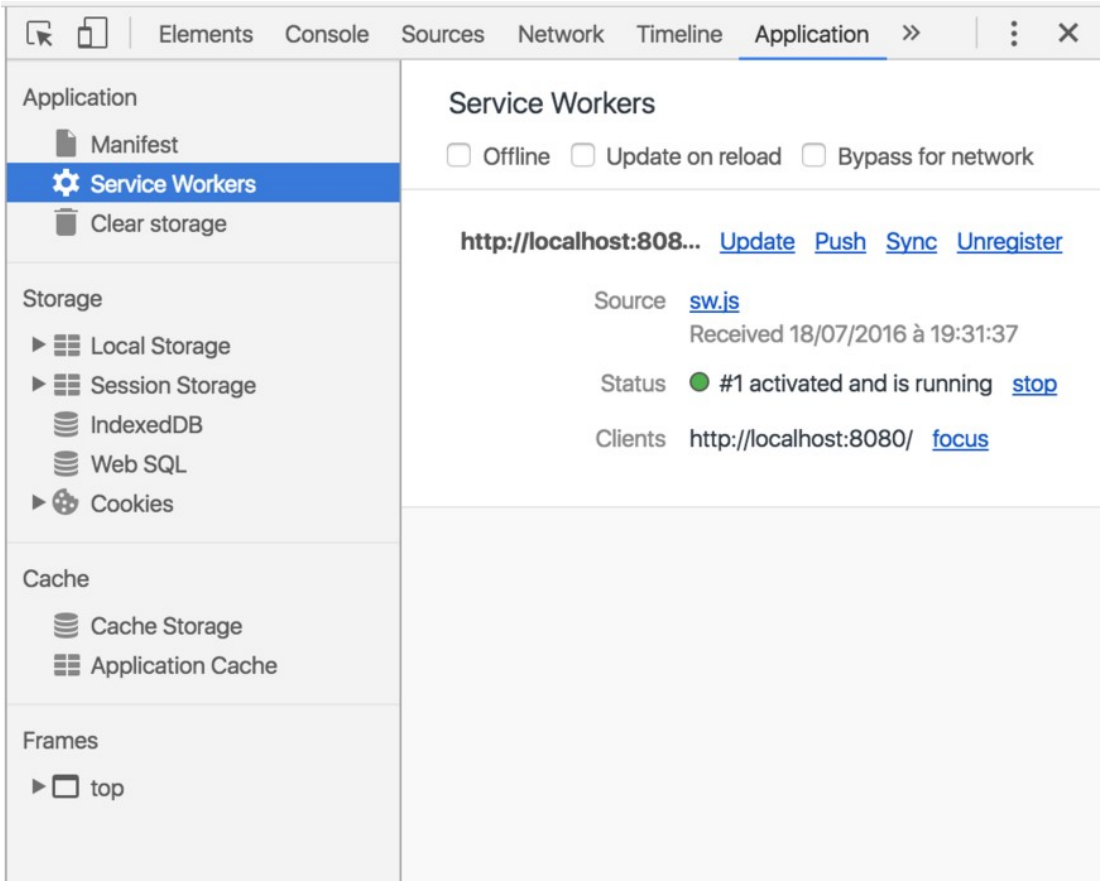
```
touch public/sw.js
```

Puis, pour enregistrer le service worker, il vous suffit d'ajouter le code suivant dans le fichier public/index.html

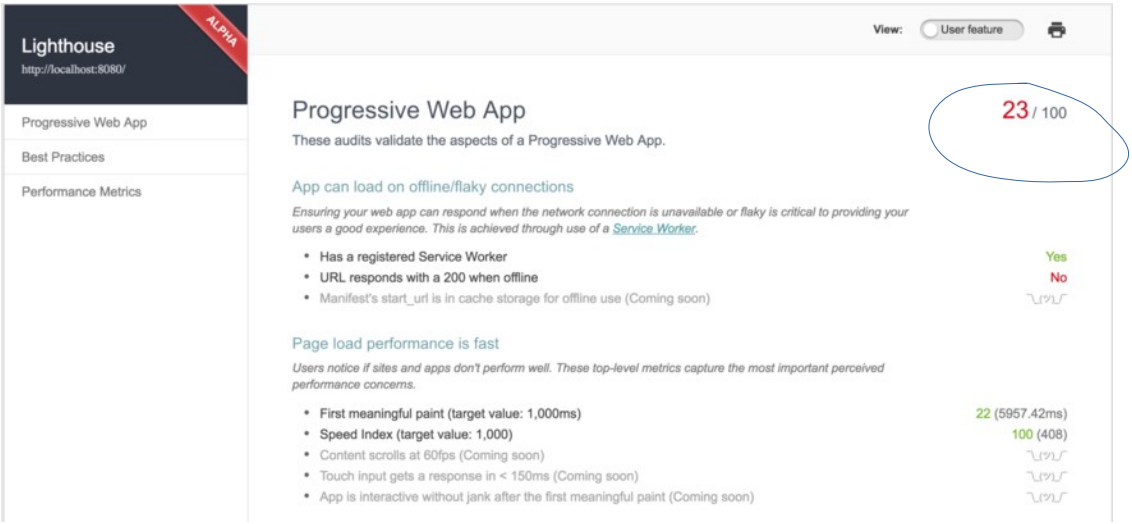
```
<script>;
if('serviceWorker' in navigator) {
  navigator.serviceWorker
           .register('/sw.js')
           .then(function() { console.log("Service Worker Registered"); });
}
</script>;
```

Vous pouvez retrouver cette étape **ici**.

Il vous suffit alors de relancer le serveur. Si vous allez sur http://localhost:8080 et que vous ouvrez l'outil de développement, vous trouverez l'onglet application qui vous permet de gérer l'état de votre PWA. Je vous invite à cliquer sur "Service Workers" pour vérifier que vous avez bien un service enregistré pour votre site.



Fermez l'outil de développement et relancez l'extension Lighthouse.



Bravo, vous avez gagné des points ! Passons à la suite.

ÉTAPE 3, LE OFFLINE :

Maintenant que vous avez enregistré votre service nous allons mettre en cache le site pour vous permettre d'avoir un site visible en offline. Pour cela, il suffit d'agrémenter le fichier sw.js. Si vous lisez la norme du W3C sur le service worker, disponible **ici**, vous verrez qu'il fonctionne comme suit : il lit des événements javascript et, en fonction de ce qui a été lu, il effectue une action. Nous allons commencer par l'événement 'install' qui va vous permettre de mettre en cache l'ensemble des pages statiques de votre site. Voici le code à ajouter dans le fichier public/sw.js :

```
self.addEventListener('install', e => {
  e.waitUntil(
    caches.open('pwa').then(cache => {
      return cache.addAll([
        '/',
        '/sw.js',
        '/index.html',
        '/planets',
        '/bundle.js',
        '/index.css',
        '/cat.png',
        '/duck.png',
        '/donut.png',
        '/racoon.png',
      ])
      .then(() => self.skipWaiting());
    })
  )
});
```

Comme vous pouvez le lire, quand l'événement est lancé, on ouvre un cache au nom 'pwa' et on lui ajoute les fichiers statiques. Si vous relancez l'application, vous pouvez alors mettre en "offline" dans l'outil de développement puis dans l'onglet applications, bien que cela ne devrait pas encore fonctionner car nous n'avons pas pris en compte les appels serveur. Pour cela, vous allez récupérer l'événement 'fetch' qui permet de récupérer ces appels serveurs en question.

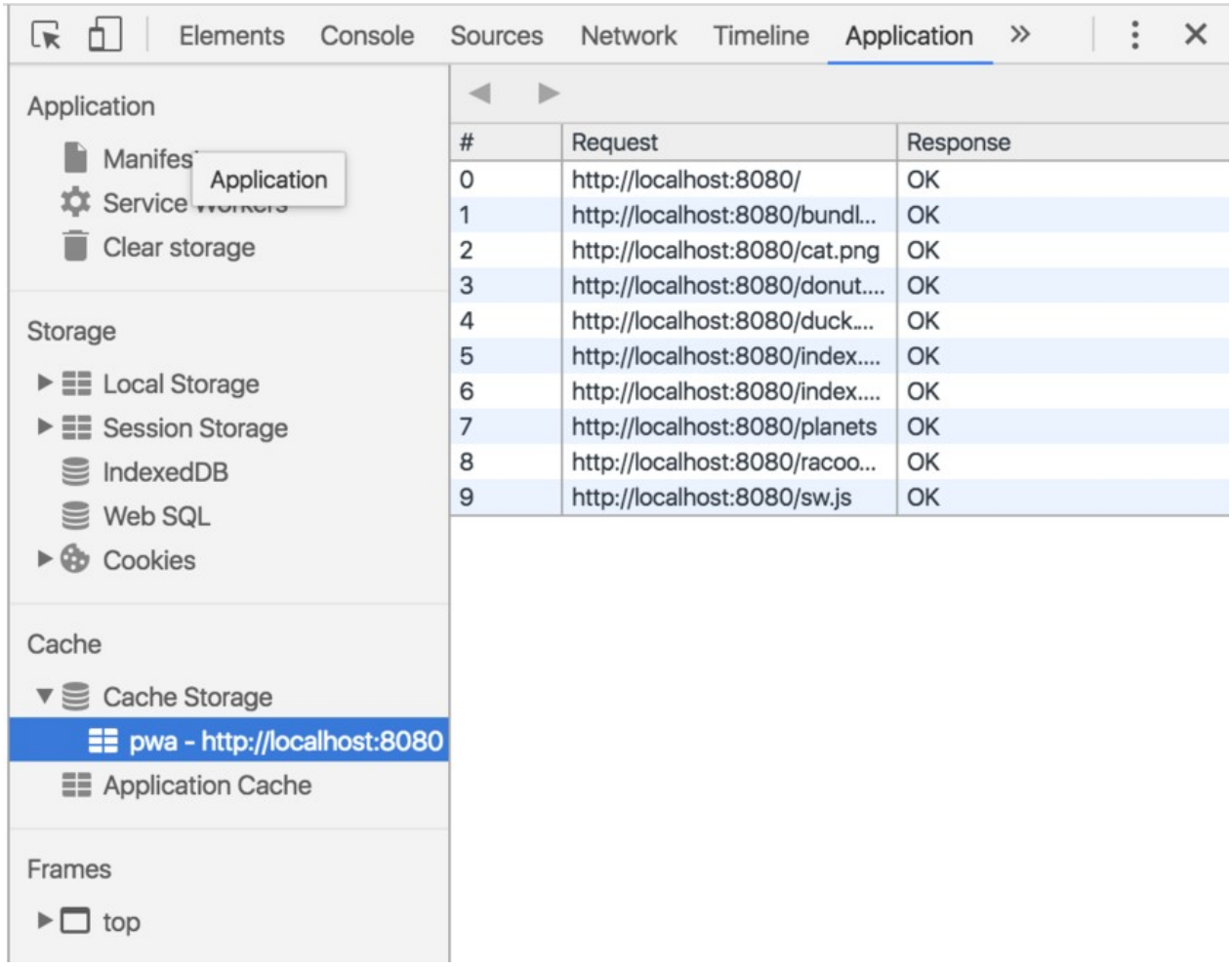
Vous ajoutez dans le fichier public/sw.js, le code suivant :

```
self.addEventListener('fetch', event =>{
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request);
```

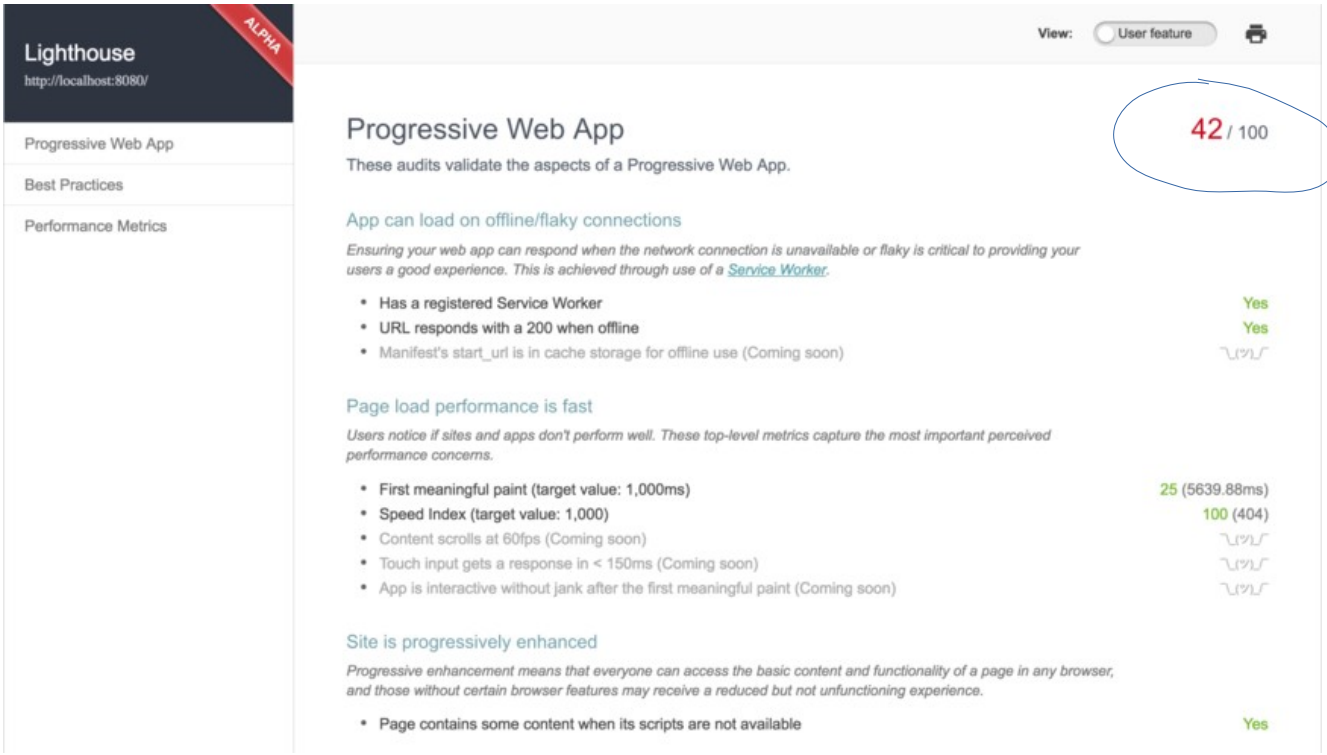
```
    })
  );
});
```

Ce que l'on fait est simple, vous récupérez chaque requête et la mettez dans le cache. Vous retrouverez cette étape **ici**.

Si vous relancez l'application, vous pouvez voir dans l'outil de développement, onglet application, un lien pour voir ce que contient votre cache (Cache Storage), vous y retrouvez l'ensemble des fichiers statiques.



Puis, encore une fois, fermez l'outil de développement, et relancez LightHouse.



Vous êtes sur la bonne voie, mais il y a une méthode encore meilleure pour initialiser votre cache.

ÉTAPE 4, PRECACHE DU SERVICE WORKER :

Comme vous avez pu le voir, il est très fastidieux de mettre chaque fichier statique dans le service worker, Google y a pensé et a mis en place plusieurs outils pour faciliter le développement.

Vous allez utiliser le projet sw-précache disponible sur **github**, ce dernier permet de générer le service worker via un fichier **Gulp**.

Vous n’avez plus qu’à ajouter dans le package.json.

```
"sw-precache": "^3.2.0",
"gulp": "^3.9.1",
```

Et faire un

```
npm install
```

Vous pouvez alors ajouter un fichier Gulpfile.js qui contiendra la configuration pour votre service worker.

```
'use strict';

// Include Gulp &amp;amp; Tools We'll Use
var gulp = require('gulp');

gulp.task('generate-service-worker', function(callback) {
  var path = require('path');
  var swPrecache = require('sw-precache');
```

```
  var rootDir = 'public';

  swPrecache.write(path.join(rootDir, 'sw.js'), {
    staticFileGlobs: [rootDir + '/**/*.{js,html,css,png,jpg,gif}'],
    stripPrefix: rootDir,
    navigateFallback: '/',
    runtimeCaching: [{
      urlPattern: /\/planet/,
      handler: 'cacheFirst'
    }],
    verbose: true

  }, callback);
});
```

En lisant le fichier, vous pouvez voir que lors de l'initialisation du sw-precache vous avez plusieurs clés de configuration. La première est la 'staticFileGlobs' qui permet d'aller chercher l'ensemble des fichiers statiques. Vous avez ensuite le 'runtimeCaching' qui permet de cacher les requêtes qui vont vers le serveur. Il vous suffit de choisir un pattern d'url et une façon de cacher, sachant qu'il existe plusieurs 'handler' :

- cacheFirst : Prend ce qui se trouve dans le cache, s'il est vide, envoie la requête au serveur.
- networkFirst : Envoie le requête au serveur, s'il ne répond pas prend ce qu'il y a dans le cache.
- fastest : Envoie la requête au serveur et va chercher dans le cache, prend celui qui répond en premier.

Pour générer le fichier, il ne vous reste plus qu'à lancer la commande suivante :

```
gulp generate-service-worker
```

Je vous invite à lire le fichier généré que vous pouvez trouver à la place de l'ancien /public/sw.js Si vous relancez l'application normalement vous n'avez aucune modification. Vous pouvez retrouver l'étape **ici**.

ÉTAPE 5, LE MANIFEST :

La mise en place du manifest est une étape simple mais qui permet de signifier aux navigateurs que vous avez créé votre première PWA. Vous pouvez retrouver la spécification sur le site **W3C**.

Je vous donne l'exemple typique que l'on trouve dans tous les bons tutoriels.

```
{
  "name": "My PWA",
  "short_name": "PWA",
  "icons": [{
        "src": "images/touch/icon-128x128.png",
```

```
      "sizes": "128x128",
      "type": "image/png"
    }, {
      "src": "images/touch/apple-touch-icon.png",
      "sizes": "152x152",
      "type": "image/png"
    }, {
      "src": "images/touch/ms-touch-icon-144x144-precomposed.png",
      "sizes": "144x144",
      "type": "image/png"
    }, {
      "src": "images/touch/chrome-touch-icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    }],
  "start_url": "/",
  "display": "standalone",
  "background_color": "#3E4EB8",
  "theme_color": "#2F3BA2"
}
```

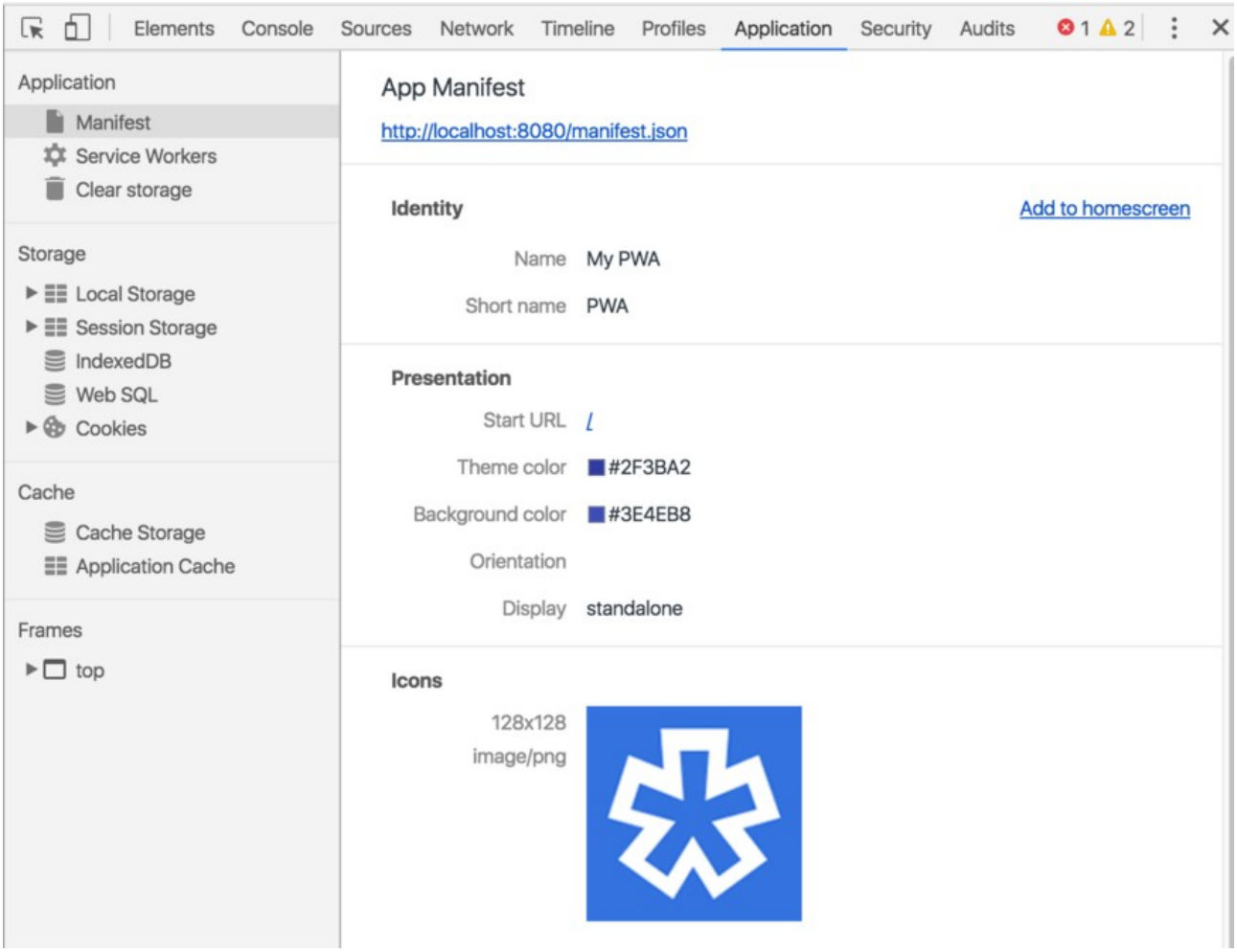
Il est très lisible puisque vous y trouvez le nom de votre application, les icônes utilisés lors de l'installation sur le téléphone et les couleurs pour le splashScreen. La clé 'display' vous permet de choisir l'orientation du téléphone lors de l'installation, soit horizontale, verticale ou 'standalone' qui permet de laisser l'utilisateur choisir.

Une fois le fichier rempli, vous devez signifier son emplacement pour le navigateur. Dans le header de la page, il faut ajouter dans le fichier /public/index.html :

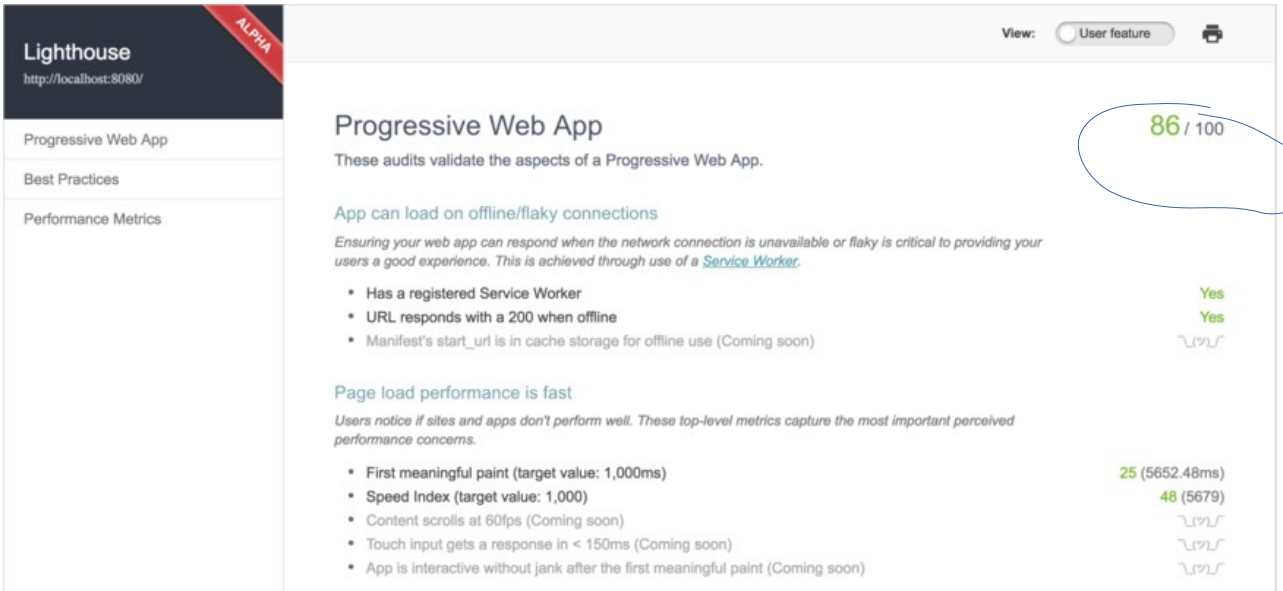
```
<link rel="manifest" href="/manifest.json">;
```

Vous pouvez retrouver cette étape **ici**.

Si vous relancez l'application et que vous allez dans l'outil de développement, onglet application, vous trouverez les propriétés de votre manifest. Il est même possible d'installer votre application en cliquant sur 'Add to homescreen'.



Et encore une fois, fermez l'outil de développement et lancez l'extension LightHouse.



Vous y êtes, votre application est une Progressive Web App ! En conclusion, ce n'est pas compliqué de mettre en place une PWA, maintenant il faut jouer avec, tester le cache, etc… Il existe d'autres

fonctionnalités sympa comme la mise en place des push notifications, le fait de contrôler le bluetooth… Pour en apprendre encore plus, vous pouvez trouver des tutoriels super intéressants dont je me suis inspiré, chez **Google**.

AUTEUR(S)
Jonathan Jalouzot

Lead développeur au @lemondefr, mes technologies sont le symfony depuis 2009, le nodejs, l'angularjs, rabbitMq etc … J'adore les médias et aimerai continuer dans ce secteur plein de surprise. Vous pouvez me retrouver sur les réseaux sociaux: Twitter: @captainjojo42 Instagram: @captainjojo42 Linkedin: https://fr.linkedin.com/in/jonathanjalouzot Github: https://github.com/captainjojo