

Partie 8: le offline

Dans cette partie finale nous allons nous concentrer sur comment faire pour que notre application ne ressemble pas à ça... :



Aucun accès à Internet

Voici quelques conseils :

- Vérifiez les câbles réseau, le modem et le routeur.
- Reconnectez-vous au réseau Wi-Fi
- [Exécutez les diagnostics réseau de Windows](#)

ERR_INTERNET_DISCONNECTED

...quand elle ne trouve pas de connexion internet !

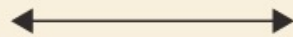
Service workers

C'est grâce aux *Service Workers* que nous allons réussir à faire cela.

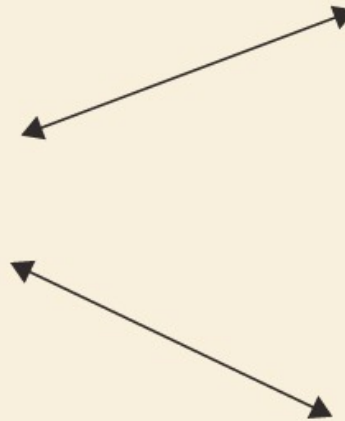
- Premièrement il faut bien comprendre où un service workers va se positionner dans notre architecture. Juste ici :



Page



Service Worker



Server



Cache

- Notre service worker va agir en soi comme un proxy par lequel toutes les requêtes faites par l'app vont passer. À ce moment-là, nous pourrons donc décider de faire ce que nous voulons avec ces requêtes (les stopper, les laisser passer, les mettre en cache, les transformer, etc.) et donc les contrôler.

ATTENTION: nous allons voir ici seulement un service worker qui permet de faire marcher notre PWA en mode offline. Puisque les seules ressources que nous allons chercher à travers le réseau sont des ressources statiques, le SW va se "contenter" d'aller chercher ses ressources une première fois quand il y a du réseau puis va les mettre dans un cache pour qu'elles soient réutilisées en offline. Le SW va aussi aller rechercher les ressources statiques quand il retrouvera du réseau. Il va comparer le cache et les données qu'il a reçus, et seulement si elles sont différentes, il va les remplacer.

- Pour commencer un peu de pratique pour mettre en place un SW.

Il faut créer un nouveau fichier nommé *sw.js*

Notre architecture de dossier ressemblera donc à ça (*italique* = dossier / **gras** = fichier)

- *dist*
 - *assets*
 - *style*
 - *lib*
 - **index.html**
 - **main.js**
 - **sw.js**

- Il faut maintenant enregistrer (dire à notre main.js qu'un service worker existe bien) notre SW.

```
// dans main.js juste après les import
if ("serviceWorker" in navigator) {
  navigator.serviceWorker.register("sw.js"); //C'est une promesse = asynchronicité
} else {
  console.log("not working");
}
```

Lors de son déploiement **asynchrone**, le SW va passer par deux phases: son installation puis son activation.

Nous pouvons donc détecter différents événements lors de son déploiement.

Nous allons agir lors d'un événement spécial (le "installed", qui veut dire que le déploiement du service worker est fini) pour par la suite recharger la page. Ce rechargement permettra à une portion du code de notre fichier sw.js de s'activer (à voir au point suivant).

- Il faut donc maintenant chercher l'événement "installed"

```
const sw = async () => {
  const registration = await navigator.serviceWorker.register("sw.js");
  registration.addEventListener("updatefound", (evt) => {
    evt.target.installing.addEventListener("statechange", (evt) => {
      if (evt.target.state === "installed") location.reload(); // il est là !
    });
  });
};

if ("serviceWorker" in navigator) {
  sw();
} else {
  console.log("SW not working / allowed");
}
```

Reste à coder notre Service Worker.

```
const CACHE_NAME = "cinetheque";
const CACHE_VERSION = "1";

// à chaque fois que l'utilisateur va recharger la page un événement fetch va être lancé
self.addEventListener("fetch", (event) => {
  if (event.request.method !== "GET") return; // si ce n'est pas un GET je retourne
  const fetchCacheFirst = async () => {
    const cache = await caches.open(`${CACHE_NAME}_${CACHE_VERSION}`); // j'ouvre le cache
    const cached = await cache.match(event.request); // si le cache matche la requête, ...
    if (cached) return cached; // ... le cache est retourné
    let response = await fetch(event.request); // si le cache matche pas la réponse direction le network
    if (response) cache.put(event.request, response.clone()); // si le network envoie une réponse elle est mise dans le cache
    return response; // et la réponse est renvoyé
  };
  event.respondWith(fetchCacheFirst());
});

// quand un nouveau service worker est en place (changement de version), il efface le cache
self.addEventListener("activate", (event) => {
  const clearOldCache = async () => {
    let keys = await caches.keys(); // je prends les clefs du cache
    for (const key of keys) caches.delete(key); // et une à une je les delete
  };
  event.waitUntil(clearOldCache());
});
```


À la fin de cette démarche, vous pouvez aller inspecter votre code => sous application => sous Cache => Cache Storage et vous pourrez voir la liste des fichiers mis en cache. De plus vous pouvez changer la version du cache et recharger la page, vous verrez le cache précédent à disparu remplacé par la nouvelle version.

Vous pouvez maintenant vous mettre en mode offline et la pwa marche !

DISCLAIMER: Les service workers ont un pouvoir presque infini sur la manipulation de requêtes. L'exemple donné ici est un petit et simple exemple pour faire marcher une app en offline.

Ressources

[Register service worker](#)

[Service worker API](#)

[Web.dev](#)

[Global Doc MDN](#)

[Monterail](#)

[Service Workers explained](#)

[Le caching](#)

Web Manifest

Voici le moment venu de faire de notre PWA une app installable et cela grâce au grand web manifest !

Un manifest web est un simple fichier JSON comportant les informations pour l'installation de notre PWA.

```

{
  "name": "Ciné★thèque",
  "short_name": "Ciné★",
  "description": "A film library to put all the movies you have seen",
  "start_url": "/index.html",
  "icons": [
    {
      "src": "/assets/android-chrome-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/assets/android-chrome-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    },
    {
      "src": "/assets/favicon-32x32.png",
      "type": "image/png",
      "sizes": "32x32"
    },
    {
      "src": "/assets/favicon-16x16.png",
      "type": "image/png",
      "sizes": "16x16"
    },
    {
      "src": "/assets/apple-touch-icon.png",
      "type": "image/png",
      "sizes": "180x180"
    }
  ],
  "theme_color": "#ff0000",
  "background_color": "grey",
  "display": "fullscreen",
  "orientation": "portrait-primary"
}

```

- Vous pouvez copier-coller le code de la slide précédente dans un nouveau fichier appelé *manifest.webmanifest* et le mettre au même niveau que le *main.js* et que le *sw.js*, puis le lié à votre HTML avec cette ligne de code:

```
<link rel="manifest" href="./manifest.webmanifest" />
```

- Pour continuer, vous pouvez copier-coller le dossier **assets** présent dans *solutions* -> *SolutionFinale*, il contient les différentes icônes de l'application.

Et vous pouvez copier-coller ces 3 lignes dans *index.html* pour avoir un favicon

```
<link
  rel="apple-touch-icon"
  sizes="180x180"
  href="./assets/apple-touch-icon.png"
/>
<link
  rel="icon"
  type="image/png"
  sizes="32x32"
  href="./assets/favicon-32x32.png"
/>
<link
  rel="icon"
  type="image/png"
  sizes="16x16"
  href="./assets/favicon-16x16.png"
/>
```

Ressources

Web manifest Doc MDN

Web.dev - web manifest mis en place

Last but not least










Dernier des derniers points: le petit bonus = les Shortcuts !

Nous pouvons ajouter un bout de code pour que quand l'utilisateur appuiera longtemps sur l'icône de l'app sur mobile, il puisse choisir un raccourci. Nous allons en mettre un pour rajouter un film directement !

```
// à rajouter dans le web manifest
"shortcuts" : [
  {
    "name": "Add movies",
    "url": "/#add"
  }
],
```


Déploiement

Pour déployer votre PWA et pouvoir tester son installation sur mobile, il faudra avoir déployé l'app sur un serveur. Quelle chance que GitHub nous laisse faire ça grâce à GitHub Pages. Comme je vous avais demandé au tout début, votre repository Github porte le nom *votrPseudo.github.io*. Après avoir déployé (commit and push) votre code sur GitHub (en étant connecté à votre compte et en ayant donc l'interface graphique de GitHub sous vos yeux) vous pourrez voir si vous cliquez là...:

NF01 final3			✓ a458abe 21 hours ago	🕒 10 commits
	assets	final3		21 hours ago
	lib	final3		21 hours ago
	style	final3		21 hours ago
	th	moving		21 hours ago
	index.html	final3		21 hours ago
	main.js	final3		21 hours ago
	manifest.webmanifest	final3		21 hours ago
	package.json	final3		21 hours ago
	sw.js	final3		21 hours ago

Help people interested in this repository understand your project by adding a README.

[Add a README](#)

No description, website, or topics provided.

☆ 0 stars

👁 0 watching

🔗 0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Environments 1

 github-pages Active

Languages



... l'avancement du déploiement de votre code sur les serveurs de GitHub.

Quand cette étape sera achevée (= un vu vert) vous pourrez vous rendre sur l'URL dans un nouvel onglet de votre browser: votrepseduo.github.io. Normalement vous aurez une réponse 404. Si vous rajoutez votrepseduo.github.io/dist cela devrait marcher et vous devriez voir votre PWA.

Cependant pour que la PWA soit installable il faut qu'elle soit en lien direct avec un nom de domaine n'ayant pas de sous-dossier. Ici nous en avons un (=dist), il faut donc modifier l'architecture de votre repository pour que l'`index.html` soit à la racine.

La nouvelle architecture devrait ressembler à ça (*italique* = dossier / **gras** = fichier), le dist a simplement disparu, car il n'était utile que pour le live serveur.

- *assets*
- *style*
- *lib*
- **index.html**
- **main.js**
- **sw.js**

Voilà , vous venez de terminer le cours d'introduction au PWA. N'oubliez pas qu'il existe pléthore de possibilité pour faire évoluer cette PWA, je vous invite à continuer en autodidacte et je vous laisse avec quelques dernières ressources !

Bonne suite !

Ressources globales

[Apprendre les PWA de A-Z](#)

[Idées des web APIs pour développer une PWA](#)

[PWA like a native app](#)