h e g

Haute école de gestion
Genève

Les PWA comme solution au développement WEB et mobile ?

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Ruben CARVALHO

Conseiller au travail de Bachelor :

Rolf HAURI, enseignant HES

Haute École de Gestion de Genève, le 9 juillet 2020

Haute École de Gestion de Genève (HEG-GE)

Filière IG

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre « Bachelor of Science HES-SO en Informatique de gestion ».

L'étudiant a envoyé ce document par email à l'adresse remise par son conseiller au travail de Bachelor pour analyse par le logiciel de détection de plagiat URKUND, selon la procédure détaillée à l'URL suivante : https://www.urkund.com .

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 9 juillet 2021

Ruben CARVALHO

Remerciements

Je voudrais remercier en premier lieu mon enseignant HAURI Rolf, qui m'a donné le sujet de mon travail de Bachelor et qui m'a suivi durant la réalisation de celui-ci. En second lieu, je voudrais remercier un élève, FAVRE Alan, qui a relu mon travail et corrigé les éventuelles fautes. Pour terminer, je voudrais remercier les trois personnes qui ont généreusement donné de leur temps afin de répondre à mes questions durant une interview.

Résumé

J’ai dû réaliser plusieurs étapes durant mon travail de Bachelor afin de répondre au mieux à ma problématique. N’ayant aucune expérience dans le domaine des Progressive Web App, j’ai dû effectuer quelques recherches, afin de me renseigner au mieux sur cette technologie.

Tout d’abord, je me suis renseigné sur l’histoire de cette technologie, est-elle récente, qui l’a créé, est-elle beaucoup utilisée ou existe-t-il des applications qui utilisent déjà cette technologie. Je me suis aperçu qu’il y avait des applications déjà existantes et utilisées par plusieurs milliers de personnes. Pour le moment, il y a surtout les réseaux sociaux, tel que Twitter ou les e-commerces, tel que DWShop qui utilise les PWA, chacun pour des raisons différentes, mais toujours avec un bilan positif pour toutes personnes ayant mis en place une PWA.

Ensuite, j’ai décidé de faire une comparaison de plusieurs technologies frontends afin de voir quels seraient les points positifs et les points négatifs de chacune. J’ai d’abord commencé par expliquer les applications natives et les applications hybrides, car c’est ce qui est le plus utilisé par les smartphones de nos jours. Puis, j’ai parlé de quelques langages de programmation, tels que React ou Flutter, utile pour développer une application native multiplateformes sur smartphones ou Electronjs, qui permet de faire des applications bureaux multiplateformes sur Windows, Linux ou MacOS.

Pour finir, j’ai dû réaliser une PWA, mais avant de la développer, j’ai regardé quelques tutos sur YouTube, afin de connaître les ressources dont j’avais besoin pour mettre en place une PWA. J’ai pu voir les différentes étapes que je devais réaliser, je me suis lancé et j’ai créé ma première PWA.

Table des matières

Liste des tableaux

Liste des figures

1. Introduction

Pour mon travail de Bachelor, j'ai décidé de choisir un sujet qui m'intéresse et dans lequel je veux évoluer professionnellement. J'ai donc choisi le domaine des applications WEB/Mobile, plus précisément le design UI/UX. Après avoir effectué quelques recherches sur le sujet, je n'ai pas trouvé comment développer mon idée afin qu'elle soit assez claire et précise pour en faire un travail de Bachelor.

J'ai pris contact avec l'enseignant HAURI Rolf, afin de lui demander s'il n'avait pas une idée de sujet qui concernait les applications WEB/Mobile afin d'en faire mon travail de Bachelor. C'est à ce moment qu'il m'a parlé de Progressive Web App (PWA). Je me suis renseigné sur le sujet, ayant suivi les deux modules à option qui sont 644-16 - Développement d'application mobile et 645-26 - Développement de clients riches - Librairies avancées Javascript, cela m'a directement intéressé car je pourrais utiliser mon expérience afin de pouvoir comparer les PWA par rapport aux autres technologies frontend.

L'évolution des téléphones mobiles a fait émerger le marché des applications mobiles, de nos jours il est indispensable de penser à développer une application mobile quand on veut développer une application web. Avant de se lancer dans le développement de celle-ci, il est important de connaître les besoins afin de choisir les technologies adéquates. Par mon travail de Bachelor, j'essaie de savoir si les PWA peuvent potentiellement être la solution au développement WEB/Mobile ?

Figure 1 – Progressive Web App

(Anjali Bhatt, 2021, TYPO3 CMS)

2. Progressive Web App

2.1 Définition

Une progressive web app est une application web qui permet à l'aide de code informatique (HTML[1], CSS[2], JS[3]) d'avoir une application qui peut être ajouté sur un smartphone ou un ordinateur et qui ressemble à une application native. En effet, grâce à l'aide des navigateurs modernes, il est possible à partir d'un site web d'avoir une application similaire aux applications natives assez facilement sans les contraintes des stores mobiles. PWA offre un système de mode offline, le push de notification et l'utilisation du hardware mobile comme une application native.

2.2 Histoire

Le terme Progressive Web App fut créé en 2015 par Google, mais le concept du PWA n'est pas nouveau. Pour comprendre l'histoire du PWA, il faut revenir à l'année 2000, lorsque XMLHttpRequest fut inventé. En effet, cette technologie permet de communiquer avec des serveurs en envoyant des requêtes via une URL HTTP sans recharger la page afin de récupérer des données des serveurs.

Cinq ans plus tard, nous avons l'arrivé d'AJAX l'acronyme *d'asynchronous JavaScript and XML*, qui permet d'offrir des applications web plus rapide et interactives. La particularité d'AJAX, en utilisant XMLHttpRequest, est de pouvoir envoyer ou récupérer des données d'un serveur de manière asynchrone sans interférer avec l'affichage et le comportement de la page web. La création de ces différentes technologies a changé la façon dont nous avions l'habitude de construire et d'imaginer les applications web.

En 2007, Steve Jobs fut la première personne à présenter le concept du PWA devant le monde entier durant la conférence Apple d'introduction de l'iPhone. A ce moment, Steve Jobs était convaincu que des applications développées en web serait utilisé sur l'iPhone et ainsi rendre l'iPhone populaire chez tout le monde, utilisateur comme développeur.

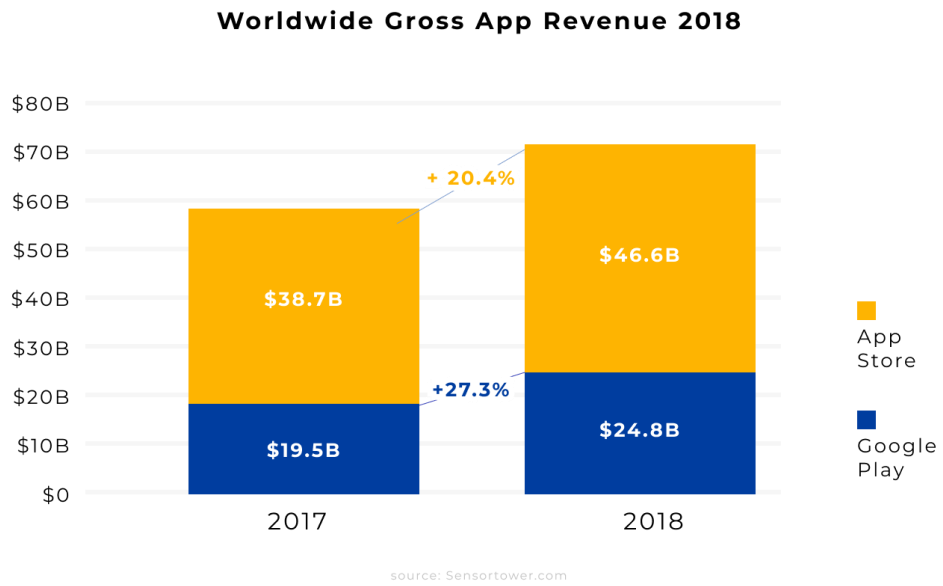
> *« (...) you can write amazing Web 2.0 and Ajax apps that look exactly and behave exactly like apps on the iPhone. And these apps can integrate perfectly with iPhone services. And guess what?*
>
> *There's no SDK that you need! You've got everything you need if you know how to write apps using the most modern web standards to write amazing apps for the iPhone today.*

[1] Langage de programmation de balisage conçu pour représenter les pages web
[2] Langage de programmation permettant de faire du design sur les pages web
[3] Langage de programmation de scripts employé dans les pages web interactives

So developers, we think we've got a very sweet story for you. You can begin building your iPhone apps today » *(Steve Jobs 2007, Conférence Apple)*

Il n'a jamais fait mention d'un App Store durant cette conférence. Finalement, le SDK pour iPhone fut annoncé en octobre 2007 et au lieu d'avoir des PWA, en juillet 2008, Apple présente l'App Store. Principalement, pour des raisons financières.

Figure 2 – Montant des stores mobiles pour l'année 2017 et 2018



(sensortower.com, 2019)

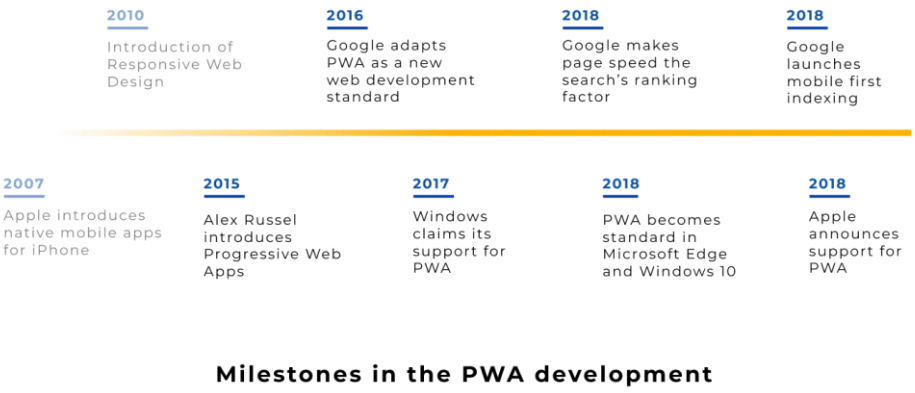
Pendant une décennie les applications natives ont dominé le marché du téléphone, ce qui a permis à Apple et à Google de devenir leaders du marché. En effet, les applications universelles ont été oubliées et pendant ce temps le responsive design a été introduit à partir de 2010. Le responsive design aide pour l'affichage des pages web sur plusieurs appareils, en adaptant la taille de la page web en fonction de la taille de l'écran avec du CSS. C'est notamment grâce au responsive design que l'idée de progressive web app revient.

En 2015, Frances Berriman et Alex Russell vont observer un nouveau style d'application web qui donne une meilleure expérience utilisateur, notamment grâce à l'évolution des navigateurs web modernes. En effet, ces navigateurs offrent une nouvelle perspective aux applications web, en leur permettant de quitter les onglets des navigateurs afin de donner une impression d'application autonome.

Ils ont décidé d'appeler ces nouveaux types d'application web « Progressive Web App ». Un an plus tard, lors de la conférence Google IO, Eric Bidelman, Senior Staff Developers

Programs Engineer, a présenté les Progressive Web Apps comme un nouveau standard dans le développement web.

Figure 3 – Histoire du PWA



(divante.com)

2.3 Comment fonctionne une PWA

Dans le développement web nous avons plusieurs approches possibles quand on veut développer un produit. Avant de commencer le développement de ce produit nous devons passer par plusieurs étapes, une de ces étapes, est le maquettage du produit. Pour réaliser le maquettage, nous avons besoin de savoir sur quel environnement sera utilisé ce produit. Est-ce que ce produit sera uniquement déployé en tant que site web, ou alors uniquement en tant d'application mobile ou alors les deux. Quand on voit la place que prend le téléphone mobile dans nos vies, il semble impensable de déployer un produit sur une application web sans l'application mobile. Malheureusement, pour des petites entreprises ou des start-ups aux moyens limités, il semble difficile de développer une application web en plus d'une application mobile.

Pour des personnes qui ne connaissent pas le coût de développement d'une application mobile, le coût dépend du temps de développement de l'application. Le développement d'une application mobile prend de 3 à 6 mois selon sa complexité. Pour une simple application mobile, on peut compter 300-350 heures ce qui fait environ 3 mois et pour les plus complexes 750-1000 heures ce qui fait environ 6 mois. Le prix moyen pour le développement d'une application mobile est de 40CHF/heure, ce qui fait 12'000 CHF pour une application mobile simple et peut aller jusqu'à 40'000CHF pour une application complexe.

Concernant le coût pour le développement d'un site web, les prix varient entre 100 et 120 CHF/heure. Les prix sont plus élevés étant donné que la demande est plus forte. Plusieurs types de site web sont possible, le moins cher étant les sites fait à l'aide d'un CMS[4], les plus cher sont les sites web personnalisés. Comme pour les applications mobiles, on peut compter 1 mois de développement pour un site simple et jusqu'à 3 à 6 mois pour un site web plus complexe et personnalisé. Le prix pour un site peut aller de 1'500 CHF à 50'000 CHF, voir beaucoup plus selon la demande et la complexité du site.

Les prix qui ont été utilisé sont une moyenne, on peut trouver des prix moins élevés comme l'inverse. Les prix varient énormément, selon l'expérience du développeur ou le taux de la demande. Les prix changent aussi si le développement du site est fait dans une agence, souvent plus cher, ou via un Freelancer. Le temps de développement est souvent moins long par une agence, car un Freelancer a souvent un autre travail.

Cependant, il existe un moyen simple pour avoir les deux pour le prix d'un. Il y a deux conditions, la première condition est que le site web et l'application mobile soit le même produit. La deuxième est que le site web doit être responsive design. En effet, l'implémentation d'une PWA à un site web est simplement de permettre aux utilisateurs finaux de pouvoir ajouter le site web sur un appareil qui donnerait une impression d'utiliser une application native.

Si le site web permet l'ajout de celui-ci sur un appareil, tel qu'un smartphone ou une tablette, on pourra voir, sur Android, une bannière indiquant si on veut ajouter le site à l'écran d'accueil. En cliquant dessus, l'application sera ajoutée à l'écran d'accueil du smartphone ou de la tablette.

Sur IOS, on peut uniquement avertir les utilisateurs finaux qu'il est possible d'ajouter le site à l'écran d'accueil de l'iPhone, mais si on souhaite réellement l'ajouter, il y a plusieurs étapes à réaliser. Il faut être sur le navigateur Safari puis cliquer sur l'icône « Partager » qui se trouve en bas au milieu du navigateur. Pour finir, il suffit de cliquer sur « Ajouter à l'écran d'accueil » afin d'avoir l'application sur le smartphone ou la tablette.

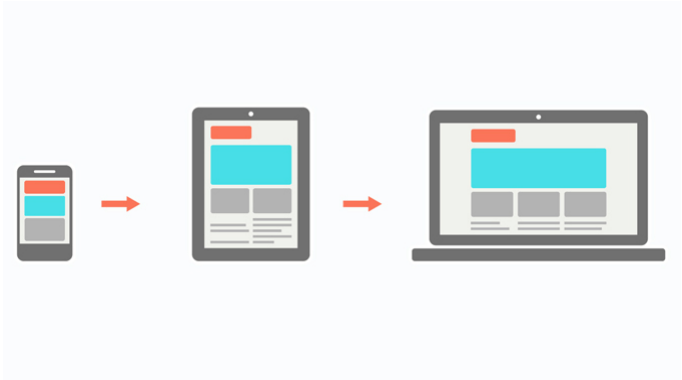
Il est également possible, d'avoir une application sur un ordinateur. La méthode d'ajout d'une PWA sur un ordinateur dépend simplement du navigateur, par exemple sur le navigateur Google Chrome, une icône apparaît à droite dans la barre URL du navigateur pour indiquer que le site web peut être installer en tant qu'application sur l'ordinateur.

[4] Programme permettant de créer un site internet, un blog ou encore un site e-commerce

2.4 Exemple de PWA existante

De nos jours, énormément d'entreprise, principalement les réseaux sociaux et le e-Commerce, ont opté pour la technologie de PWA. Une des raisons principales est l'ajout d'une fonctionnalité supplémentaire à moindre coût. En effet, les entreprises qui ont eu une approche mobile-first dès la création de leur site web, n'auront aucun mal à ajouter une PWA, alors que pour les autres, si on veut utiliser le plein potentiel d'une PWA, il faut reconcevoir son site web sur smartphone en ayant une approche mobile-first. Une approche mobile-first comme son nom l'indique, c'est le fait de concevoir son site web sur smartphone avant de le concevoir sur les autres plateformes.

Figure 4 – Approche mobile-first



(divante.com)

2.4.1 Réseaux sociaux

2.4.1.1 Twitter

Twitter décide d'implémenter une PWA. On se demande ce que Twitter a à gagner en développant une PWA, alors qu'ils ont une application mobile qui est utilisée par plus de 145 millions de personnes par jour. La force d'une PWA est son utilisation rapide, ainsi que la mise en place de celle-ci par un site web qui est déjà entièrement responsive design. En avril 2017, la PWA Twitter Lite a vu son apparition et après une annonce de Twitter indiquant de télécharger la PWA via le « Add to HomeScreen », plus de 250'000 utilisateurs jours utilisent Twitter Lite.

> *« Twitter Lite is now the fastest, least expensive, and most reliable way to use Twitter. The web app rivals the performance of our native apps but requires less than 3% of the device storage space compared to Twitter for Android. »*
>
> *(Nicolas Gallagher, the Engineering Lead for Twitter Lite)*

En effet, Twitter Lite utilise moins de donnée que l'application native notamment grâce à la mise en cache de certaine donnée. La taille de la PWA prend beaucoup moins de place sur le téléphone que l'application native, nous avons 600KB pour l'application PWA contre 23.5MB pour l'application native.

2.4.1.2 Pinterest

Il y a plusieurs raisons sur le fait que Pinterest décide d'implémenter une PWA. La raison principale était que leur site web sur smartphone soit une très mauvaise UX (User Expérience). En 2017, le groupe de Pinterest décide de monter une équipe afin de reconcevoir leur site web sur le smartphone avec une approche mobile-first en y implémentant une PWA. Cet immense changement fut conséquent mais il apporte beaucoup d'opportunités par l'ajout d'une PWA comme l'on peut le constater par le verdict un an après par l'équipe de Pinterest Engineering.

> *« Weekly active users on the mobile web have increased 103% year-over-year overall, with a 156% increase in Brazil and a 312% increase in India. On the engagement side, session length increased by 296%, the number of Pins seen increased by 401% and people were 295% more likely to save a Pin to a board. Those are amazing in and of themselves, but the growth front is where things really shone. Logins increased by 370%, and new signups increased by 843% year-over-year. Since we shipped the new experience, mobile web has become the top platform for new signups. And for fun, in less than 6 months since fully shipping, we already have 800 thousand weekly users using our PWA like a native app (from their home screen) »* *(The Pinterest Engineering Team 2018, Article)*

Nous avons d'autres réseaux sociaux, qui ont succombé à l'énorme potentiel qu'offre les PWA, principalement pour améliorer l'expérience utilisateur, nous avons Tinder qui a vu une augmentation du temps passé sur Tinder, notamment sur les sessions de messages plus longues en utilisant l'application sur ordinateur.

2.4.2 e-Commerce

2.4.2.1 DW Shop

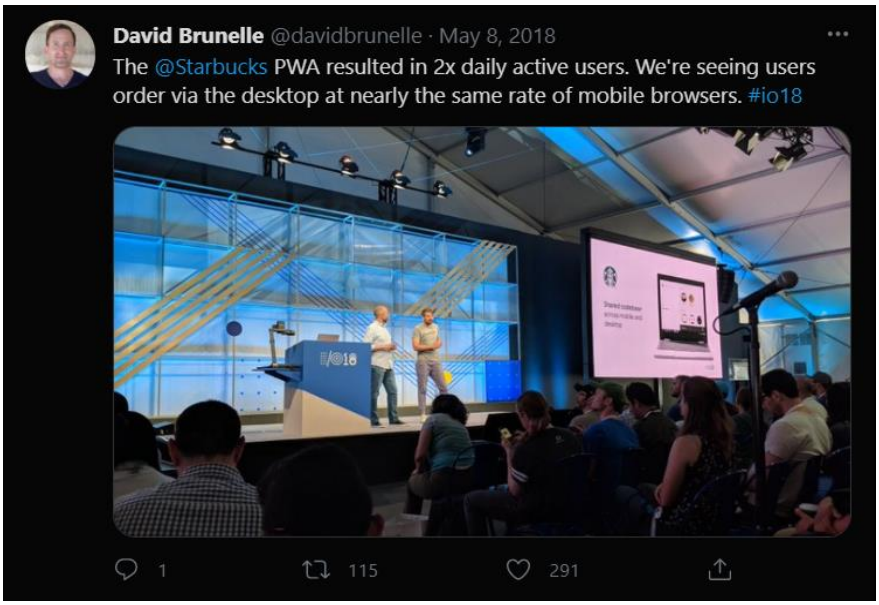
DW Shop, une sous-branche de groupe DW Creative, qui fut créé par le célèbre coiffeur polonais Dariusz Wójcik, fait face comme beaucoup de e-Commerce, au problème de l'expérience utilisateur sur téléphone mobile qui est assez catastrophique. La plupart des utilisateurs qui ont essayé d'acheter ou commander des articles sur téléphones mobiles ont vite abandonné ou tout simplement changé de plateforme. Pour pallier à ce problème, beaucoup de marque de e-Commerce utilise les réseaux sociaux, comme Instagram pour DW Shop, qui correspond à la quasi-totalité de leur clientèle. Le but de DW Shop est de convaincre leur clientèle d'Instagram a utilisé une plateforme sur téléphone mobile, pour cela ils comptent utiliser une PWA.

En utilisant, les Frameworks Vue Storefront comme technologie frontend et Magento 2 comme backend e-Commerce pour le store, DW Shop, veut convaincre par une meilleure expérience utilisateur.

> *« In a short time, the DW Shop achieved 75% of its traffic from Instagram, and thanks to PWA technology, it has seamlessly turned these users into clients directly on mobile devices »* *(ALEKSANDRA KWIECIEŃ 2018, Article)*

On peut voir énormément d'entreprise e-Commerce mettre en place une PWA afin d'avoir une augmentation de leurs clientèles ou du temps passé sur l'application. Prenons l'exemple de l'entreprise Starbucks, qui a implémenté une PWA, en voyant le nombre d'utilisateur actif multiplié par deux sur leur application. Le nombre de commande sur l'application du téléphone mobile ou de l'ordinateur est quasi-similaire.

Figure 5 – Twitte Starbucks résultat PWA



(David Brunelle, 2018)

Vous pourrez voir d'autres entreprises qui ont décidé d'implémenter une PWA, ainsi que des statistiques sur la page suivante, https://www.pwastats.com/ .

2.5 Comparaison de technologie frontend

2.5.1 Application native

Une application native est une application mobile développé spécifiquement pour un des systèmes d'exploitation utilisés par les smartphones, les plus connus étant iOS et Android.

Généralement, un développement spécifique dans le langage de programmation adapté pour chaque système d'exploitation était obligatoire. On utilise le langage de programmation Objectif-C ou Swift pour les développements iOS et le langage Java ou Kotlin pour Android. Initialement, Il était nécessaire d'avoir autant d'applications, qu'il y a de systèmes d'exploitation, mais cela a évolué avec les langages de programmation, tel que React ou Flutter, qui ont permis en utilisant un seul langage de programmation de développer une seule application déployable sur plusieurs systèmes d'exploitation.

Les applications natives permettent d'utiliser toutes les fonctionnalités hardware de l'appareil mobile, comme le GPS, l'accéléromètre ou encore l'appareil photo. En effet, cela permet de développer des applications plus riches que les applications web qui ont un accès restreint aux différentes fonctionnalités hardware d'un appareil.

Par ailleurs, contrairement au PWA qui peuvent être installée à partir d'un navigateur web, les applications natives ne peuvent être installées que par l'intermédiaire des stores mobiles, tel que Google Play ou App Store. Elles sont également soumises aux règles des stores mobiles, qui ont un contrôle sur les applications, par exemple le droit de refus sur l'ajout d'une application qui ne respecte pas toutes les règles du store. Une part des revenus de l'application est également retiré par le store qui permet à l'application d'exister dans son écosystème et donc d'être utilisée.

2.5.1.1 Avantages et inconvénients

Tableau 1 – Avantages et inconvénients application native

Application native

Avantages	Inconvénients
Facilité d'intégration au système d'exploitation	Coût plus onéreux
Mise à jour possible lors du lancement de l'application	Dépendance des applications aux stores mobiles
Tests de l'application via un émulateur. Android Studio, Expo	Développement spécifique pour chaque OS, même si Flutter ou React facilite cela
Installation depuis un store mobile. App Store, Google Play	Application plus lourde qu'une PWA
Utilisation de toutes les fonctionnalités hardware d'un appareil	
Push de notification ou message reçu sur l'application	
Expérience de navigation adapté à l'OS[5], ce qui donne une meilleure expérience utilisateur.	

[5] Système d'exploitation

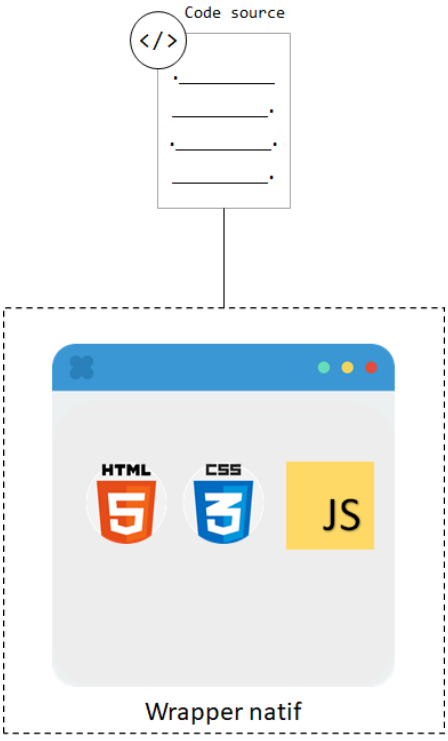
Communication entre plusieurs applications installées sur le même appareil possible
Possible utilisation d'application en mode hors connexion

2.5.2 Application hybride

Une application hybride est à la base un site web ou une application web, qui utilise donc des technologies web, tel que HTML, CSS et JS, qu'on veut transformer en application « native » afin qu'elle puisse être déployée sur les différents stores mobiles et installée sur un smartphone comme une application native.

A la différence des applications natives qui doivent être développées dans deux langages de programmation différents, les applications hybrides sont développées sur un seul langage de programmation. Elles sont donc beaucoup plus simples à maintenir qu'une application native, notamment dû aux langages de programmation utilisés par les applications hybrides, qui sont connues par le plus grand monde, HTML, CSS et JS.

Figure 6 – Application hybride



Afin de déployer une application hybride sur un store mobile et qu'elle puisse être utilisée comme une application native, il nous faut joindre le contenu de cette application web à un « wrapper » ou à une enveloppe native. Il est également possible d'ajouter un code natif à notre application hybride afin d'utiliser certaines fonctionnalités hardware de l'appareil comme une application native.

Nous avons différents logiciels qui peuvent aider à développer une application hybride. Voici une liste.

- Apache Cordova
- Monaca
- Adobe PhoneGab

(The Net Ninja, 2019)

2.5.2.1 Avantages et inconvénients

Tableau 2 – Avantages et inconvénients application hybride

Application hybride

Avantages	Inconvénients
Coût de l'application souvent moins cher qu'une application native	Plus lourd qu'une application native, car au lancement de l'application c'est d'abord le wrapper natif qui est lancé, puis le contenu web qui est affiché
Maintenance plus rapide et plus simple, donc moins de frais, cause : un seul code source	Peut parfois engendrer des bugs, car ce n'est pas du code totalement natif
Le déploiement des applications hybrides sont plus rapides sur les stores mobiles que les applications natives	Expérience utilisateur moins bonne car il peut y avoir des problèmes de fluidités
Les mises à jour des applications hybrides non pas besoin d'être vérifiées par les stores mobiles, donc plus rapide à mettre à jour, uniquement si c'est le code non-natif qui est mis à jour	
Permet d'utiliser les fonctionnalités hardware comme une application native	

2.5.3 Technologie frontend possible

2.5.3.1 React

React est une bibliothèque JavaScript libre qui a été développée par Facebook en 2013. Le but de React est de faciliter la création des applications web monopages avec un seul langage de programmation.

En 2015, React Native fait son apparition, c'est un Framework qui permet de développer des applications cross-plateforme Android et iOS. En effet, React Native permet au développeur d'utiliser React avec les fonctionnalités natives des systèmes d'exploitation comme Android ou iOS.

Il y a de nombreuses applications, comme Facebook, UberEat ou encore AirBNB, qui sont utilisées par plusieurs millions de personnes chaque jour et qui ont été codé entièrement en React Native pour les plateformes iOS et Android. Un langage plutôt simple à prendre en main, car c'est une bibliothèque JavaScript avec un langage de balisage pour les composants, qui est connu par la plupart des développeurs web.

Exemple de code React Native (Page HelloWorld)

```
import React from 'react';
import { Text, View } from 'react-native';

const YourApp = () => {
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems:
"center" }}>
      <Text>
        Try editing me! 🎉
      </Text>
    </View>
  );
}

export default YourApp;
```

2.5.3.2 Flutter

Flutter est un kit de développement de logiciel (SDK) d'interface utilisateur open-source créé par Google. Il est utilisé pour développer des applications multi-plateforme à partir d'une seule base de code pour Android, iOS, Linux, Mac, Windows, Google Fuchsia et le web. À la différence de React, qui utilise ReactJS pour toute les plateformes web et React Native pour toutes les plateformes mobile et tablette, Flutter utilise le langage Dart pour toutes les plateformes.

En 2018, une première version « stable » du Framework Flutter 1.0 a été annoncé et publié lors de l'événement Flutter Live de Google. L'une des dernières grosses mises à jour a été annoncé le 3 mars 2021 par la release de son SDK en version 2.0.0, en réglant de nombreux bugs et problèmes de jeunesse.

Le langage étant très jeune, il reste des choses à améliorer, mais dans l'ensemble il est très stable et permet de faire de très bonne application. Il y a notamment une application My BMW qui a été développé en interne par BMW pour les plateformes iOS et Android, entièrement à l'aide de Flutter.

Exemple de code Flutter (Page HelloWorld)

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

2.5.3.3 Electronjs

Electron est un Framework qui permet de développer des applications de bureau en utilisant des langages web, qui sont HTML, CSS et JS. En intégrant, Chromium, un navigateur web et Node.js, une plateforme JavaScript coté serveur, Electron permet de créer des applications multi-plateformes qui fonctionne sous Windows, Linux et MacOS en utilisant une base de JavaScript. Il n'est pas nécessaire de connaître des langages de développement natif pour développer avec Electron.

La première version « stable » d'Electron 1.0.0 fut ajouté le 11 mai 2016. La dernière version stable en date d'Electron 13.1.6 était le 6 juillet 2021, notamment pour régler certains bugs de crash sur MacOS en utilisant le code `app.quit()`. Electron est un Framework stable régulièrement mise à jour, qui est facilement prise en main, comme pour React, par la plupart des développeurs web notamment grâce à l'utilisation de langages web pour créer des applications de bureau natif.

De très grosse application de bureau connu, comme Facebook Messenger, Twitch ou en encore Microsoft Teams, utilisé chaque jour par des millions de personnes, entièrement développé avec Electron.

Exemple de code Electron (Module - créer une fenêtre de navigation native)

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow () {
  // Créer la fenêtre de navigation.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
```

```
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  // et charger l'index.html de l'application.
  mainWindow.loadFile('index.html')

  // Ouvrir les outils de développement.
  mainWindow.webContents.openDevTools()
}
```

2.6 Avantages et Inconvénients

Tableau 3 – Avantages et inconvénients PWA

Progressive Web App

Avantages
Permet d'avoir du multi-plateforme en développement uniquement un site web
Coute moins cher en temps et en argent à développer qu'une application native
Avoir les avantages d'une application sur téléphone mobile sans les contraintes d'un store mobile
Mode hors-ligne qui permet d'utiliser l'application après une perte de connexion internet
Améliore l'expérience utilisateur, grâce à la mise en cache des fichiers de l'application
Le push de notification qui renforce l'interaction avec les utilisateurs de l'application
Fonctionne uniquement sur le protocole HTTPS, donc apporte une sécurité accrue
Maintenable très facilement, car les changements fait sur le site web sont directement répertoriés sur l'application
Le référencement par Google est renforcé étant donné qu'elle se trouve sur un moteur de recherche et que l'algorithme les mets en valeurs
Application moins lourde qu'une application native, prend beaucoup moins de place sur un téléphone
Inconvénients
Pas encore assez connu par le plus grand monde, les personnes sont encore trop habituées au store mobile qui domine le marché depuis plus de 10 ans

Limitation de l’utilisation de certaines fonctionnalités du téléphone mobile pour les PWA, selon le système d’exploitation. Peut-être contourné, notamment le push de notification sur IOS, mais demande un développement assez compliqué pour être adapté au système d’exploitation IOS
Pas encore très stable sur tous les navigateurs, peut avoir certain problème de fluidité selon les navigateurs. Marche très bien sur Chrome
Ne peut pas encore utiliser la 3D, donc pas intéressant pour certain jeu mobile

https://bluemarketing.fr/progressive-web-apps-pwa-les-avantages-de-cette-nouvelle-generation-dapplications-mobiles/

https://fr.followanalytics.com/blog/applications-natives-vs-pwa-comment-choisir/

https://stackoverflow.com/questions/63819485/sending-push-notifications-to-ios-from-pwa

https://www.izooto.com/blog/ios-safari-push-notifications-in-2021

(Vous pensez qu’un tableau qui compare les fonctionnalités qu’une PWA peut utiliser sur le téléphone mobile, avec celle d’une application native, est une bonne idée ?)

3. Mettre en place une PWA (Théorie)

3.1 Manifest

Manifest est un document au format JSON, qui fournit des informations sur l'application comme son nom, les icônes que l'application va utiliser et URL sur lequel l'application va s'ouvrir quand elle sera lancée. Ce fichier va indiquer au navigateur comment l'application devra s'afficher et se lancer quand il sera ajouté à l'écran d'accueil d'un appareil. Les navigateurs qui sont compatible avec les PWA sont Chrome, Edge, Firefox, UC Browser, Opera, et le Samsung browser. Safari est partiellement compatible.

3.1.1 Mots-clés JSON

3.1.1.1 Name/Short_name

C'est le nom que l'application aura quand elle sera ajoutée à l'écran d'accueil d'un appareil. Si jamais le nom de l'application est trop long pour l'afficher avec l'application, alors c'est le short_name qui sera utilisé.

Exemple

```
"name" : "My First App", "short_name" : "1st App"
```

3.1.1.2 Description

Une description pour indiquer le but de l'application.

Exemple

```
"description" : "My First App than help me to understand PWA"
```

3.1.1.3 Dir

Spécifie l'orientation de lecture du texte du nom, du short_name et de la description. Il peut aider à afficher correctement les langues qui se lisent de droite à gauche.

```
"dir" : "rtl",
"lang" : "ar",
"short_name" : "التطبيق من أنا!"
```

Les valeurs possibles :

- **Ltr** (de gauche à droite)
- **Rtl** (de droite à gauche)

- **Auto** (conseille au navigateur d'utiliser l'algorithme bidirectionnel Unicode pour mieux comprendre la direction du texte.)

3.1.1.4 Lang

Pour indiquer la langue de l'application. Souvent ensemble avec le mot-clé **dir**.

3.1.1.5 Background_color

Pour indiquer la couleur de fond que l'application utilisera. Afin d'améliorer l'expérience utilisateur, cette couleur est parfois utilisée si le fichier manifest est chargé avant le fichier de style, cela permet une transition plus discrète d'un fond d'une couleur jusqu'au chargement du style.

3.1.1.6 Start_url

L'URL qui sera chargé lors du lancement de l'application qui est sur l'écran d'accueil.

3.1.1.7 Display

Permet de définir quel affichage aura l'application lors de son lancement. Par exemple, il est possible de cacher la barre URL et le navigateur afin d'avoir une application autonome.

Exemple

```
"display" : "standalone"
```

Les valeurs possibles :

- **Fullscreen** – Ouvre une application en cachant le navigateur et prend tout l'espace disponible par l'écran.
- **Standalone** – Permet d'avoir l'affichage d'une application autonome. Quand l'application est lancée, elle est séparée du navigateur et cache tout ce qui peut avoir un rapport avec les éléments du navigateur, tel que la barre URL.
- **Minimal-ui** – Ce mode est assez similaire à l'affichage du mode standalone, la seule différence est qu'il permet d'avoir le minimum afin de gérer la navigation, tel que l'action de back ou de refresh.
- **Browser** – Mode par défaut, ouvre simplement l'application sur le navigateur.

3.1.1.8 Icons

Quand un utilisateur ajoute une PWA à son écran d'accueil, il est possible définir un ensemble d'icône qui peuvent être utilisé sur l'écran d'accueil, sur la page de démarrage, ou autre.

Il est possible d'utiliser des **maskables icons**[6], surtout si on utilise l'application sur Android, en ajoutant `"purpose" : "any maskable"` dans le mot-clé **icons**.

Les mots-clés nécessaires :

- **Src** – Défini le chemin où se trouve l'image
- **Sizes** – Défini la taille de l'image en pixel
- **Type** – Le type de l'image

Exemple

```
"icons" : [
    {
      "src" : "/images/icons-192.png",
      "type" : "image/png",
      "sizes" : "192x192"
    },
    {
      "src" : "/images/icons-512.png",
      "type" : "image/png",
      "sizes" : "512x512"
    }
  ],
```

3.1.2 Ajouter le fichier manifest à une page web

Souvent le fichier manifest est nommé manifest.json. Pour ajouter ce fichier manifest sur toutes les pages web de la PWA, il suffit d'ajouter la balise `<link>` sur chaque page web.

Exemple

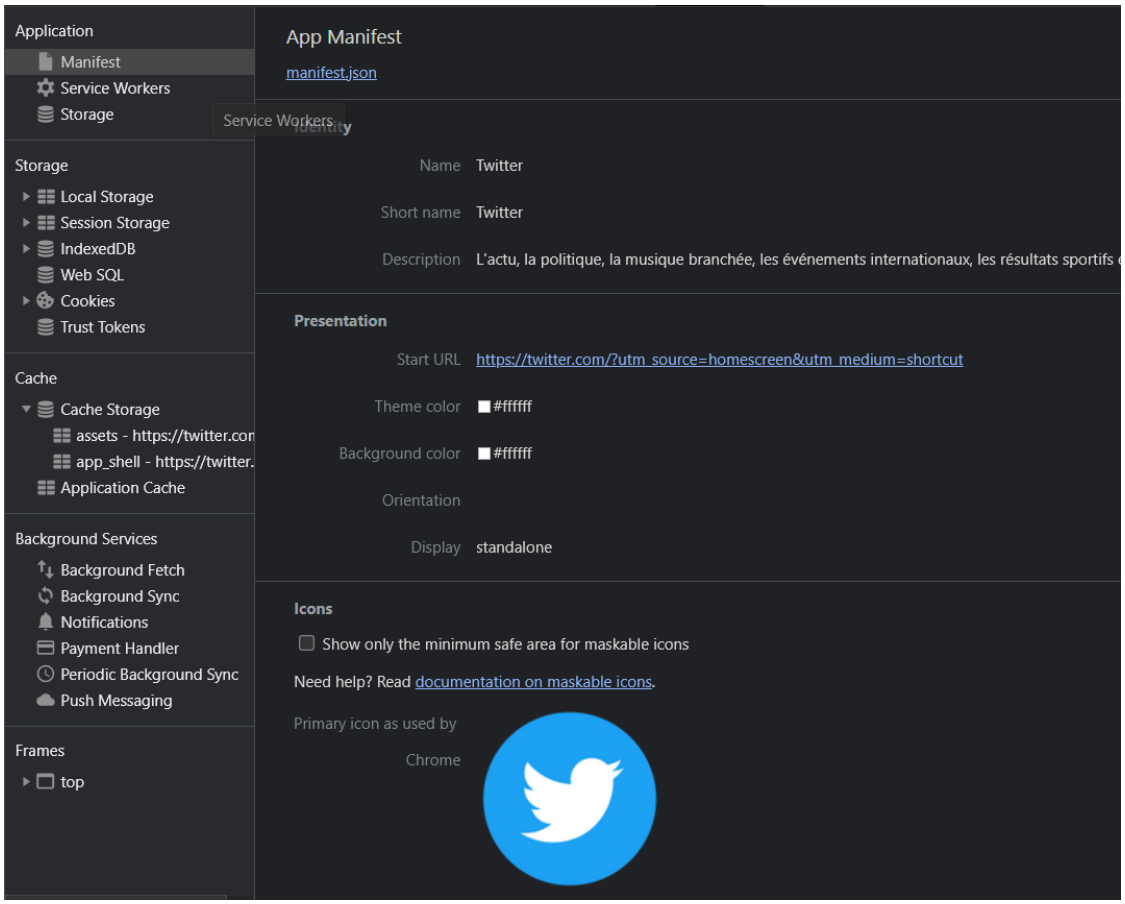
```
<link rel="manifest" href="/manifest.json">
```

3.1.3 Test Manifest sur Chrome

Le navigateur de Google permet de vérifier si les différents paramètres qui ont été défini sur le fichier manifest sont correctes. En effet, à l'aide du DevTools de Chrome, on a un onglet « Application » qui permet de vérifier le fichier manifest qui est lié à l'application.

[6] https://web.dev/maskable-icon/

Figure 7 – Google Chrome Manifest Twitter

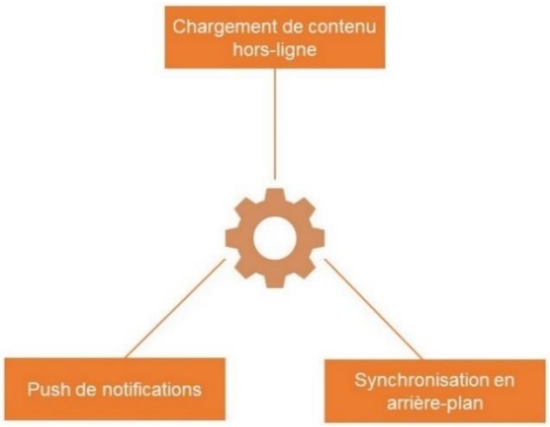


3.2 Service Worker

Le service worker est l'élément le plus important d'une PWA, il est le cœur de l'application, il est donc important de comprendre comment fonctionne ce service si on veut comprendre les PWAs.

Le service worker va permettre de faire tout ce qu'on attend d'une application moderne qui s'exécute sur un téléphone mobile, tel que le chargement de contenu en mode hors-ligne, afin qu'il soit toujours possible d'ouvrir et de voir le contenu de l'application sans connexion internet à l'aide des fichiers mis en cache. On peut également faire de la synchronisation en arrière-plan, de sorte que si un utilisateur final veuille mettre à jour des informations ou simplement ajouter de nouvelles informations, ceci sera effectué en arrière-plan une fois que l'utilisateur aura de nouveau accès à une connexion internet sans bloquer l'utilisateur. Il est également possible de faire du push de notifications, afin que l'application puisse tout simplement avertir l'utilisateur d'un changement sur celle-ci ou d'un potentiel message que l'utilisateur pourrait recevoir sur l'application.

Figure 8 – Fonctionnalités importantes des PWA

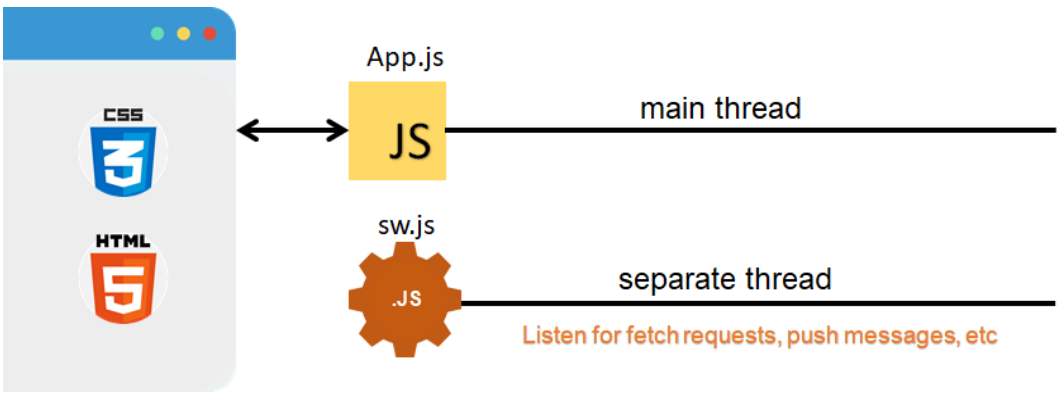


(The Net Ninja, 2019)

Les services workers sont simplement des fichiers javascript, mais ils fonctionnent différemment des fichiers javascript qu'on utilise pour un site web. Quand on crée un site web, on utilise différentes technologies tel que l'HTML, le CSS ou le JS. Le javascript qu'on a l'habitude d'utiliser pour créer un site web peut interagir directement avec le DOM[7] et fonctionne sur le flux principal. Contrairement au service worker qui n'a aucun accès au DOM, fonctionne sur un autre flux du navigateur et écoute différents événements définis dans le service worker et réagit à ceux-ci.

Les services workers ne peuvent être exécuté que dans un contexte sécurisé, donc ils ont besoin du protocole HTTPS pour fonctionner.

Figure 9 – Explication service worker



(The Net Ninja, 2019)

[7] Interface de programmation qui permet à des scripts d'examiner et de modifier le contenu du navigateur web

3.2.1 Enregistrer le service worker

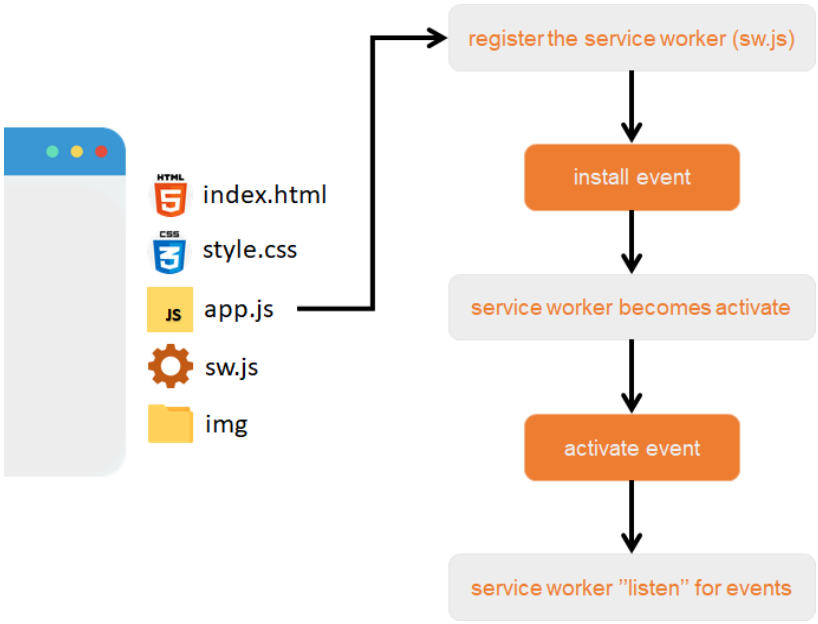
Si le navigateur prend en charge le service worker, il est nécessaire de l'enregistrer afin qu'il puisse être exécuté. Pour cela, il est nécessaire de mettre le code suivant dans un fichier javascript, tel que le fichier app.js.

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

3.2.2 Le cycle de vie

Pour commencer, une fois que le service worker est enregistré, le fichier sw.js sera automatiquement téléchargé, puis installé et pour finir activé. Ces étapes sont répétées dans l'ordre jusqu'à ce que le service worker soit prêt à écouter les différents évènements qui peuvent être déclenché sur l'application et y réagir.

Figure 10 – Cycle de vie service worker



(The Net Ninja, 2019)

3.2.2.1 Évènement d'installation

L'événement d'installation se déclenche uniquement s'il y a eu un changement sur le fichier du service worker ou alors à chaque nouveau lancement de l'application. Cet événement va s'occuper de toutes les actions qui sont nécessaires de faire au chargement de l'application, tel que mettre certains fichiers en cache.

Il suffit de mettre le code suivant dans le fichier du service worker afin de faire les actions nécessaires au moment où l'évènement d'installation sera déclenché.

```
self.addEventListener('install', evt => {
  console.log("service worker has been installed!");
});
```

3.2.2.2 Évènement d'activation

L'événement d'activation se déclenche une fois que l'événement d'installation est terminé. Cet évènement est surtout utile pour supprimer tous les fichiers qui ne sont plus nécessaires ou pour nettoyer l'application.

Le code est similaire à l'évènement d'installation, la différence est le mot-clé qu'on va écouter afin de faire une action particulière au moment où cet évènement est déclenché.

```
self.addEventListener('activate', evt => {
  console.log("service worker has been activated!");
});
```

3.2.2.3 Évènement Fetch

Un évènement fetch est également possible, il est déclenché lorsqu'une requête HTTP est émise par l'application. En effet, c'est très utile car ça nous permet d'intercepter des requêtes et d'y répondre de façon personnalisée. Voici un exemple d'utilisation.

```
self.addEventListener('fetch', evt => {
    console.log('Ressource récupérée '+evt.request.url);
});
```

Il y a plusieurs ressources qu'on peut récupérer, par exemple un fichier demandé en particulier, une copie d'un fichier mise en cache ou un bout de code JavaScript qui fera quelque chose de particulier, beaucoup de choix s'offre au développeur.

3.3 IndexedDB

Pour rendre une PWA ou un site web plus rapide et toujours fonctionnel en mode hors-ligne. Il faudrait un système permettant d'avoir une base de données coté client, qui donnerait la possibilité de manipuler certaines données de l'application sans devoir communiquer par le réseau afin d'atteindre une base de données sur un serveur externe. Il serait possible de le faire avec le Web Storage, cependant, il ne permet que de stocker 10 Mo ce qui représente une assez faible quantité de données.

La solution au problème est IndexedDB, qui est une API[8] coté client qui permet de stocker une importante quantité de données structurées, incluant des fichiers ou blob, contrairement au Web Storage, qui permet le stockage de petite quantité de données avec une clé et une valeur.

IndexedDB est un système de gestion de base de données transactionnel, c'est différent des SGBD[9] relationnel basé sur SQL[10], qui est une base de données, utilisant un système de table avec un nombre de colonne fixe. IndexedDB est plutôt une base de données orienté objet sur JavaScript, similaire au système de base de données NoSQL, comme MongoDB. Il utilise un système de paires, avec une clé et une valeur. La valeur est l'objet de donnée et la clé est la propriété de cet objet.

Les transactions qui sont effectué via IndexedDB sont asynchrone, donc le site web ne sera pas bloquée entre chaque transaction. Il était aussi possible d'utiliser indexedDB de façon synchrone, mais il a été retiré.

La sécurité est l'élément le plus important pour IndexedDB qui peut contenir les données sensibles d'un site web, il est donc très important que seul le site web puisse accéder à son propre IndexedDB. Pour cela, IndexedDB a établi une **Same Origin Policy (« politique de même origine »)**, qui oblige le domaine, le protocole de couche d'application et le port à être identiques, sans quoi les données ne seront pas disponibles.

IndexedDB est disponible sur tous les navigateurs modernes.

3.3.1 Mode hors-ligne

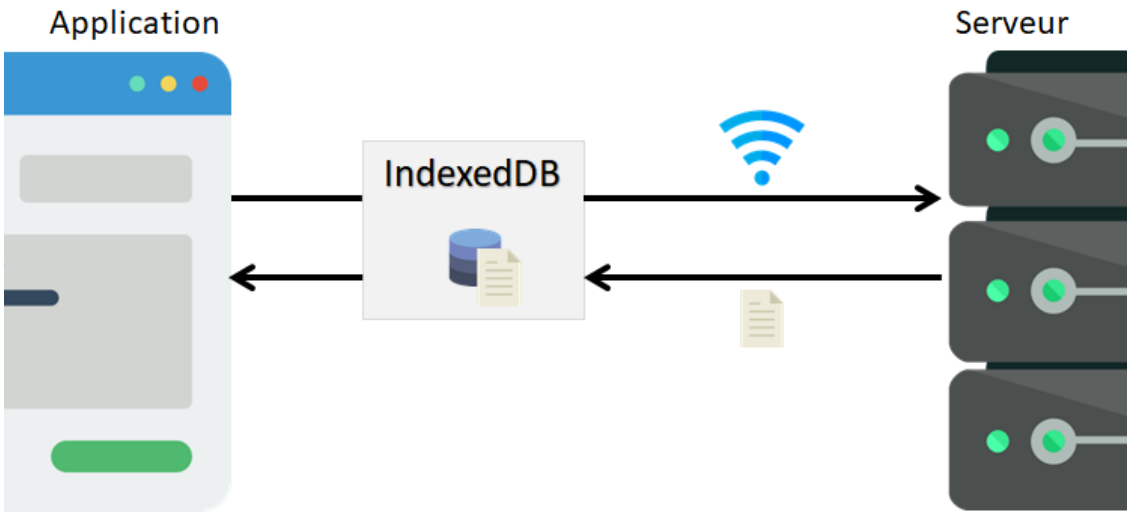
Il y a plusieurs utilisations possibles concernant IndexedDB, mais la principale utilisation étant dès le chargement du sites web avoir une copie de la base de données ou une partie de celle-ci sur IndexedDB. Ensuite, pour chaque transaction qu'il est nécessaire de faire, il faut toujours passer par indexedDB d'abord, ainsi si une perte de connexion internet à lieu, l'utilisateur ne le remarquera pas. C'est ce qu'on appelle une utilisation en mode hors-ligne. Il est possible de le voir sur les images ci-dessous.

Une première transaction à lieu lors du chargement du site web, afin d'afficher les données nécessaires à l'utilisateur et de faire une copie de la base de données.

[8] Un produit ou un service qui permet de communiquer des informations à un autre produit ou service
[9] Logiciel qui permet de stocker des informations dans une base de données
[10] Langage de programmation permettant de manipuler les données et les systèmes de bases de données relationnelles

Maintenant le but, étant que chaque modification effectuée sur la base de données, soit aussi modifier sur IndexedDB afin d'avoir une persistance des données.

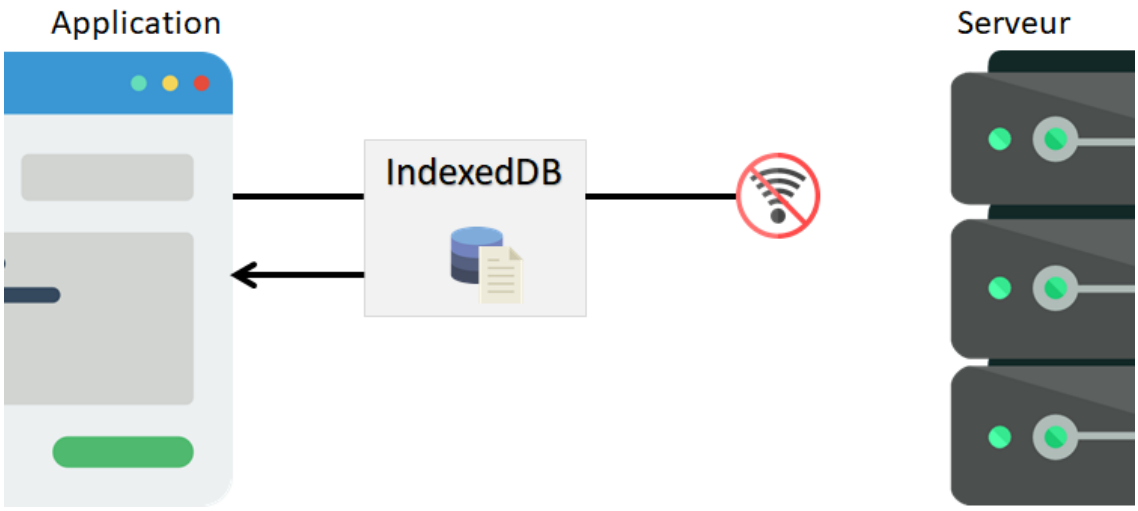
Figure 11 – Persistance des données IndexedDB en ligne



(The Net Ninja, 2019)

Si une perte de connexion internet arrive, le but est d'utiliser les données qui ont été stocké sur IndexedDB, pour que l'utilisateur ne se rend pas compte qu'une perte de connexion à eu lieu. Comme vous pouvez le voir sur l'image ci-dessous.

Figure 12 – Persistance des données IndexedDB hors-ligne



Une fois que la connexion internet a été rétabli, il est nécessaire, si une modification à eu lieu sur IndexedDB de le répertorié également sur la base de données coté serveur afin de toujours garder une persistance des données des deux côtés.

Il est nécessaire de mettre en place à l'aide des commandes de IndexedDB, cette persistance des données et cette utilisation lors d'une perte de la connexion internet. Il est possible de le faire automatique à l'aide d'une seule commande, avec Firebase.

3.4 Cache Storage

Le système de cache permet une utilisation de l'application, tel que la navigation ou l'affichage d'une page web en mode hors-ligne, uniquement si la page web a été visité par l'utilisateur quand une connexion internet était établi.

À plusieurs reprises, quand on a problème de connexion internet, on peut voir sur les navigateurs une erreur indiquant qu'il n'y a aucun accès à internet. Pour éviter ce genre d'erreur sur le navigateur et continuer à utiliser l'application, il suffit d'utiliser le système de cache, qui va stocker coté client tous les fichiers nécessaires au fonctionnement du site web.

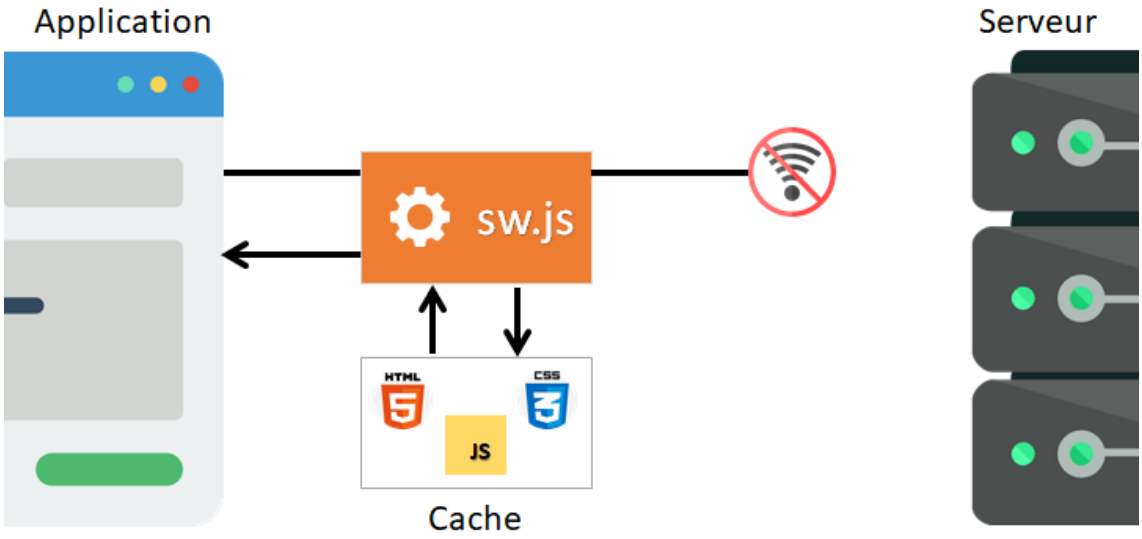
Concernant la gestion du système de cache, cela dépend des développeurs, mais il est possible de créer plusieurs caches, qui vont contenir des fichiers différents. Par exemple, il est possible d'avoir un cache, qu'on va nommer « statique », qui va stocker tous les fichiers de la premier page web au lancement du site web. On peut ensuite avoir un autre cache, qu'on va nommer « dynamique », qui va stocker tous les fichiers des autres pages web visitées au fur et à mesure par l'utilisateur. Ainsi, lors d'une perte de la connexion internet, on peut toujours avoir accès au site, ce qui donne une bonne expérience utilisateur.

Pour une meilleure expérience utilisateur, il est recommandé de créer une page à afficher si jamais l'utilisateur n'a plus accès à internet et qu'il n'a pas visité toutes les pages du sites web, afin de ne pas avoir une erreur affichée par le navigateur.

3.4.1 Mode hors-ligne

Comme on peut le voir sur l'image ci-dessous, la gestion du cache est laissée au service worker pour une PWA. En dehors du mode hors-ligne qui a été expliqué au-dessus et qui est assez explicite sur l'image. Un des buts du cache, c'est d'améliorer la rapidité de l'application et de diminuer le nombre d'interaction avec le serveur en utilisant les fichiers qui sont dans le cache pour gérer la navigation et afficher les pages web.

Figure 13 – Explication de l'utilisation du cache hors-ligne



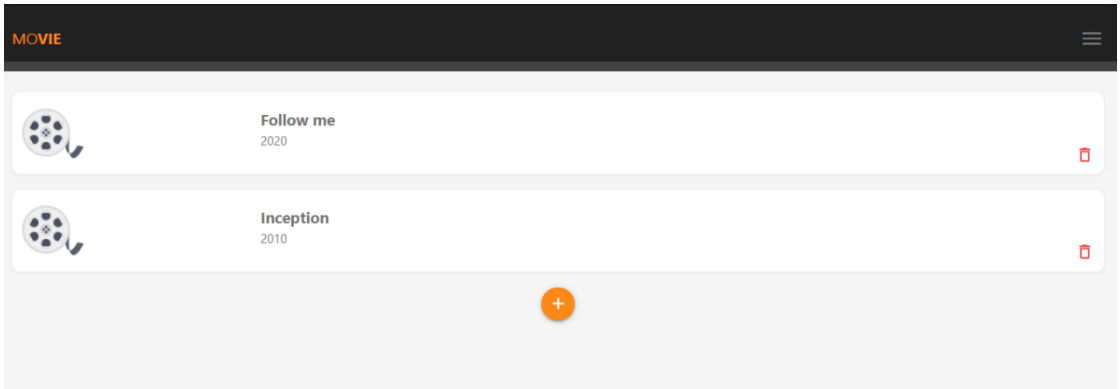
(The Net Ninja, 2019)

4. Création d'une PWA

La meilleure façon pour comprendre une nouvelle technologie frontend est par la pratique, j'ai donc développé une application PWA. Comme je ne connaissais pas les méthodes à réaliser pour mettre en place une PWA, j'ai décidé de regarder une playlist d'une trentaine de vidéo, expliquant le fonctionnement et le code à mettre en place pour développer une PWA.

Le code qui est écrit sur ces vidéos est aussi sur un git. Pour une question de simplicité, j'ai pris le code du site web existant sur le git, que j'ai modifié, sans le code pour ajouter une PWA au site web. Au démarrage de mon développement j'avais donc trois fichiers HTML, deux fichiers CSS et deux fichiers JS.

Figure 14 – PWA Movie



4.1 Manifest

Pour cette étape, j'ai créé un fichier que j'ai appelé manifest.json. Ensuite, j'ai indiqué tous les paramètres que le navigateur web à besoin, comme j'ai expliqué dans le chapitre 3.1. Les paramètres qui sont importants, c'est le "display":"standalone" qui va donner cette impression d'être sur une application native et "orientation":"portrait-primary" qui indique l'orientation de l'application.

```
{
  "name":"Movie",
  "short_name":"Movie",
  "start_url":"/index.html",
  "display":"standalone",
  "background_color":"#FF8816",
  "theme_color":"#FF8816",
  "orientation":"portrait-primary",
}
```

Il y a aussi toutes les tailles d'images, étant donné que la PWA peut-être ajoutée sur plusieurs plateformes, qui ont donc des tailles d'écrans différentes, il est important d'avoir plusieurs tailles d'images.

```
"icons": [
    {
      "src": "./img/maskable_icons/maskable_icon_x72.png",
      "type": "image/png",
      "sizes": "72x72",
      "purpose": "any maskable"
    },
    {…},
    {
      "src": "./img/maskable_icons/maskable_icon_x512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose": "any maskable"
    }
]
```

Il est important d'ajouter le fichier manifest.json à toutes les pages du site web. Le code suivant doit être ajouté entre dans la balise `<head>`.

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

4.2 Service Workers

Pour cette étape, j'ai créé deux fichiers JS, un qui s'appelle app.js et un second qui s'appelle sw.js. Le fichier app.js va pouvoir interagir avec le DOM et le fichier sw.js va gérer le service worker.

Il faut enregistrer le service worker auprès du navigateur, afin qu'il soit ajouté au site web. Pour cela, il faut mettre le code suivant sur un fichier qui a accès au DOM, donc pour ce cas, je l'ai fait sur le app.js. Je vérifie si le navigateur web peut utiliser le service worker, si c'est le cas, alors je l'enregistre.

```
if('serviceWorker' in navigator) {
  navigator.serviceWorker.register("./sw.js")
  .then((reg) => console.log("service worker registered!", reg))
  .catch((err) => console.log("service worker not registered!", err));
}
```

Ensuite, il va falloir faire quelque chose sur le service worker pour chaque état ou évènement de celui-ci qui va être déclenché. Comme j'ai expliqué dans le chapitre 3.2.2, les évènements sont déclenchés dans l'ordre et sous certaines conditions. Pour le premier évènement qui est « install », je vais ajouter différents fichiers dans le cache dès le lancement de l'application.

```
self.addEventListener('install', evt => {
  console.log("service worker has been installed!");
  //Pre-caching static data from the server
  evt.waitUntil(
    caches.open(staticCacheName).then(cache => {
      console.log("caching shell assets");
      cache.addAll(assets);
    })
  );
});
```

La code `waitUntil` va attendre que tous les fichiers soit ajouté dans le cache pour continuer.

La variable `staticCacheName` est seulement le nom du cache. Comme je l'ai déjà expliqué, il est possible de créer plusieurs caches.

La variable `assets` contient tous les fichiers qui seront ajoutés au cache `staticCacheName`.

```
const assets = [
  '/',
  '/index.html',
  '/js/app.js',
  '/js/ui.js',
  '/js/materialize.min.js',
  '/css/materialize.min.css',
  '/css/styles.css',
   ...
  '/pages/fallback.html',
];
```

Pour la suite, nous avons l'évènement « activate » qui va servir surtout à nettoyer le cache, en supprimant un ancien cache et en ajoutant le nouveau en utilisant un système de versionning qui sera expliqué au chapitre 4.4. J'ai ajouté un cache supplémentaire qui sera surtout utile durant l'évènement « fetch » qui est `dynamicCacheName`. Comme son nom l'indique c'est un cache qui va rajouter des fichiers au fur et à mesure que l'utilisateur visite les différentes pages web. C'est un cache qui évolue contrairement au `staticCacheName` qui lui est rempli uniquement au lancement de l'application.

```
self.addEventListener('activate', evt => {
  console.log("service worker has been activated!");
  //cache versioning
  evt.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys
        .filter(key => key !== staticCacheName
       && key !== dynamicCacheName)
        .map(key => caches.delete(key)))
    })
  );
});
```

Pour finir, l'évènement « fetch », va être utilisé pour le mode hors-ligne ou pour rendre l'application plus rapide, grâce aux fichiers qui sont dans les deux caches.

En effet, lorsque qu'un évènement « fetch » est déclenché, je vais regarder si une des ressources demandées correspond à une des ressources qui se trouve dans l'un des caches. Si c'est le cas, je retourne cette ressource, ainsi, il n'est pas nécessaire de passer par le serveur pour obtenir cette ressource. Si ce n'est pas le cas, j'ajoute cette ressource au cache dynamique que je récupère du serveur, puis je la retourne au client.

Cependant, si l'utilisateur n'a pas de connexion internet et que cette ressource n'est pas dans le cache, alors il sera impossible de la retourner. Pour éviter une erreur du

navigateur web indiquant que l'utilisateur n'a pas de connexion internet, il suffit de créer une page HTML, indiquant qu'il y a une erreur et qu'il est impossible de retourner cette ressource, car l'utilisateur est hors-ligne.

Dans ce code, uniquement les évènements qui ne correspondent pas à l'URL « firestore.googleapis.com » vont être traités, car je ne veux pas enregistrer dans le cache des ressources qui peuvent provenir de la base de données de Firestore.

```
self.addEventListener('fetch', evt => {
  //console.log("fetch event", evt);
  if(evt.request.url.indexOf('firestore.googleapis.com') === -1) {
    evt.respondWith(
      caches.match(evt.request).then(cacheResp => {
        return cacheResp || fetch(evt.request).then(fetchResp => {
          return caches.open(dynamicCacheName).then(cache => {
            cache.put(evt.request.url, fetchResp.clone());
            limitCacheSize(dynamicCacheName, 15);
            return fetchResp;
          })
        });
      }).catch(() => {
        if(evt.request.url.indexOf('.html') > -1) {
          return caches.match('./pages/fallback.html');
        }
      })
    );
  }
});
```

4.3 Cache Storage

Concernant le cache, j'ai utilisé deux sortes de caches comme j'ai déjà expliqué. Un qui va se remplir au lancement de l'application et un autre au fur et à mesure dynamiquement. Pour gérer le cache, j'utilise un système de versionning manuel. A chaque fois qu'il y a un changement sur un des fichiers qui peuvent être sur un des deux caches, alors j'incrémente le chiffre dans le nom du cache manuellement.

```
const staticCacheName = "site-static-v8";
const dynamicCacheName = "site-dynamic-v8";
```

Cependant, comme les variables du cache se trouvent sur le fichier du service worker et qu'un changement dans ce fichier entraine le déclenchement de l'évènement « install » puis de l'évènement « activate », qui s'occupe de nettoyer le cache, ainsi l'ancien fichier du cache sera supprimé et remplacé par le nouveau.

4.4 Mode Hors-ligne

Pour le mode hors-ligne, il y a deux systèmes qui ont été utilisés. Le premier c'est celui du cache, qui prend le relai en cas de perte de connexion, pour tout ce qui est frontend de l'application. Le deuxième, concerne la base de données du navigateur IndexedDB, qui en cas de perte de connexion internet va s'occuper de l'affichage et modification des données. Lorsque la connexion est rétablie, les changements qui ont été effectués sur les données vont être appliqués sur la base de données coté serveur. Ce système qui vérifie que la persistance des données soit appliquée, est géré automatiquement par Firebase avec cette simple ligne de code ci-dessous.

```
db.enablePersistence()
  .catch(err => {
    if(err.code == 'failed-precondition') {
        // probably multiple tabs open at once
        console.log("persistence failed");
    } else if (err.code == "umimplemented") {
        console.log("persistence is not available");
    }
  });
```

Ce code se trouve dans un fichier qui se nomme db.js, tout ce qui a un rapport avec la base de données est écrit dans ce fichier.

4.5 Bannière d'ajout sur smartphone

4.5.1 Android

Pour Android, une bannière indiquant que l'application web qui est ouvert sur un navigateur web, peut-être ajouté à l'écran d'accueil du smartphone apparaît automatiquement après avoir respecté certains critères d'une PWA. En cliquant sur cette bannière, Android vous demande si vous voulez ajouter l'application sur l'écran d'accueil du smartphone.

4.5.2 iOS

Pour iOS, il n'existe pas de bannière qui permette d'ajouter directement l'application sur l'écran d'accueil, mais il est possible d'avertir l'utilisateur. J'ai implémenté deux codes, un qui permet de savoir si l'application web est sur iOS et un autre pour savoir si l'application est en mode standalone, donc déjà ajouté à l'écran d'accueil.

```
const isIos = () => {
  const userAgent = window.navigator.userAgent.toLowerCase();
  return /iphone|ipad|ipod/.test( userAgent );
```

```
}
```

```
const isInStandaloneMode = () =>
('standalone' in window.navigator)
&& (window.navigator.standalone);
```

Avec ces informations je peux savoir quand la bannière en HTML doit s'afficher ou non.

```
if (isIos() && !isInStandaloneMode()) {
  let popup = document.getElementById("myPopup");
  popup.classList.toggle("show");
}
```

4.6 Firebase

J'utilise Firebase comme backend et serveur externe, donc toute la gestion des données ainsi que l'hébergement de l'application sera effectué sur Firebase.

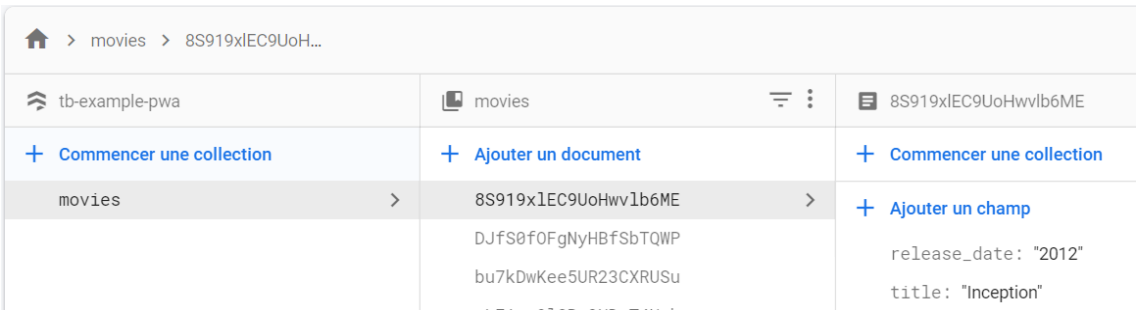
Depuis la plateforme web de Firebase, il est possible de créer un nouveau projet. Dans ce nouveau projet, il est possible de gérer l'hébergement et les données de son application. D'autres modules sont également possible, comme un système de push de notification, mais pour cette application, j'ai uniquement utilisé deux modules.

4.6.1 Firestore Database

Pour la gestion des données de l'application, je vais utiliser un module de Firebase, Firestore Database, qui est une base de données NoSQL.

Tout d'abord, j'ai commencé par créer la structure de ma base de données, étant une base NoSQL, le système fonctionne via une clé unique et les objets de cette clé unique sont les valeurs. J'ai créé une collection « movies » qui va contenir chaque film et les données liées à ce film. On peut voir sur l'image ci-dessous la structure de ma base de données.

Figure 15 – Structure de la base de données - Firestore



Ensuite, il faut récupérer les données qui sont dans la base, afin de les afficher dans l'application. Pour cela, on veut être averti de chaque changement qui s'opère dans la base de données, afin d'y appliquer une action. Pour cette application, je regarde uniquement, s'il y a une insertion ou une suppression d'un objet dans ma collection « movies » et j'applique une action liée à ce changement. Il suffit de mettre ce code dans le fichier db.js.

```
db.collection('movies').onSnapshot((snapshot) => {
  //console.log(snapshot.docChanges());
  snapshot.docChanges().forEach(change => {
    //console.log(change, change.doc.data());
    if(change.type === 'added') {
      renderMovie(change.doc.data(), change.doc.id);
    }
    if(change.type === 'removed') {
      removeMovie(change.doc.id);
    }
  });
});
```

4.6.1.1 Affichage des films

Pour la récupération des données et l'affichage de celle-ci, j'ai créé une fonction dans le fichier ui.js, qui se nomme `renderMovie()`. Cette fonction va seulement afficher les films sur l'application. Elle va être appelée une fois au lancement de l'application, si la collection « movies » contient des films et une deuxième fois, à chaque nouveau film qui est ajouté dans la collection.

```
const renderMovie = (data, id) => {
  const html = `
    <div class="card-panel movie white row" data-id="${id}">
      <img src="./img/reel.png" alt="recipe thumb">
      <div class="movie-details">
        <div class="movie-title">${data.title}</div>
        <div class="movie-release_date">${data.release_date}</div>
      </div>
        <div class="movie-delete">
          <i class="material-icons remove" data-
id="${id}" name="delete">delete_outline</i>
        </div>
    </div>`;

    movies.innerHTML += html;
};
```

Comme on peut le voir ci-dessus le code de balisage, qui est en orange, est du HTML, c'est ainsi que l'affichage des films est effectué. La variable `html` va contenir le rendu d'un film. La ligne de code `movies.innerHTML += html` va simplement ajouter le rendu de ce film à l'intérieur d'une autre balise qui se trouve sur une page HTML, qui est index.html pour cette application. Voici la balise qui se trouve dans le fichier index.html.

```
<div class="movies container grey-text text-darken-1"></div>
```

On peut voir dans cette balise `<div>`, une classe qui se nomme `movies`, c'est grâce à cette classe qu'on va pouvoir récupérer cette balise avec du code JavaScript et ajouter le rendu d'un film entre la balise avec le code `innerHTML`. Le code `+=` signifie simplement qu'on va ajouter les prochains films à la suite.

```
const movies = document.querySelector('.movies');
```

En effet, on peut voir que la fonction `renderMovie()` est appelée dans une boucle, donc tant que l'affichage de tous les films n'a pas eu lieu alors la fonction va continuer d'être appelée.

4.6.1.2 Insertion d'un film

L'insertion d'un film se fait via un formulaire en HTML, l'utilisateur va entrer les informations nécessaires dans les champs, puis à l'aide du JavaScript je vais récupérer ces informations et les ajouter à la collection « movies ». En HTML, on utilise une balise `<form>`, je vais récupérer ce formulaire, afin d'avoir les informations qu'il contient et les ajouter dans une variable, qui se nomme `form`. Toute cette partie et ce qui va suivre se trouve dans le fichier db.js.

```
const form = document.querySelector('form');
```

Maintenant, si l'utilisateur a validé le formulaire en cliquant sur le bouton ajouter du formulaire, alors un évènement « submit » est déclenché. Ci-dessous c'est le code du bouton en HTML qui se trouve dans le fichier index.html.

```
<input type="submit" value="Add" class="btn-small add-btn" />
```

Lorsque que l'évènement est déclenché, c'est que l'utilisateur vient de faire l'insertion d'un nouveau film via le formulaire. Je récupère les informations du formulaire, qui sont le titre du film et sa date de sortie, que je stocke dans une variable, qui se nomme `movie`.

Je vais ensuite ajouter ce nouveau film à la collection « movies » et vider les champs du formulaire. Ci-dessous on peut voir le code qui s'occupe de l'insertion du film.

```
form.addEventListener('submit', evt => {
  evt.preventDefault();

  const movie = {
    title: form.title.value,
    release_date: form.release_date.value,
  };

  db.collection('movies').add(movie)
  .catch(err => console.log(err));

  form.title.value = '';
  form.release_date.value = '';
});
```

4.6.1.3 Suppression d'un film

Pour la suppression d'un film, il y a deux étapes. La première, c'est de le supprimer de la collection « movies », la deuxième c'est de le supprimer du HTML, afin que l'utilisateur ne voie plus le film qu'il a décidé de supprimer.

Pour que l'utilisateur puisse supprimer un film lorsque qu'il le désire, j'ai ajouté une icône de corbeille sur chaque film, la clé unique du film est stockée sur le paramètre `data-id` de la balise de l'icône `<i>`.

```
<i class="material-icons" data-id="${id}">delete_outline</i>
```

Lorsque que l'utilisateur va cliquer sur un film, l'évènement « click » est déclenché, cependant aucune action n'est effectuée, car il faut cliquer sur l'icône de la corbeille pour que le film soit supprimé. Pour savoir sur quel élément du film l'utilisateur a cliqué, je regarde dans le `tagName` qui est un paramètre lié à la balise. Le `tagName` de la balise `<i>` est un `I` majuscule. Si l'utilisateur à bien cliqué sur l'icône, alors je peux supprimer le film qui se trouve dans la collection, à l'aide du `data-id` qui contient la clé unique du film. Ce code se trouve dans le fichier db.js.

```
const movieContainer = document.querySelector('.movies');
movieContainer.addEventListener('click', evt => {
  if(evt.target.tagName === 'I') {
    const id = evt.target.getAttribute('data-id');
    db.collection('movies').doc(id).delete();
  }
```

```
});
```

Il faut aussi supprimer le film dans le HTML, pour cela j'ai créé une fonction `removeMovie()` qui se déclenche lorsque l'évènement « removed » est déclenché par la collection « movies ». Je récupère le film ayant été supprimé de la collection avec sa clé unique et je le supprime simplement du HTML. Ce code se trouve dans le fichier ui.js.

```
const removeMovie = (id) => {
  const movie = document.querySelector(`.movie[data-id=${id}]`);
  movie.remove();
};
```

4.6.2 Hosting

Pour l'hébergement de mon application, j'utilise le module Hosting de Firebase. Personnellement, j'utilise Visual Studio Code, pour développer mes applications web. Il contient un terminal sur la racine de mon application et je peux écrire les différentes commandes pour créer mon hébergement.

Première étape avant de créer l'hébergement, j'ai dû installer node.js sur mon ordinateur. Pour vérifier si node.js était bien installé ou s'il était déjà installé sur mon ordinateur, j'ai tapé la commande `$ node -v` dans un terminal.

Maintenant, il est important d'installer le module firebase-tools globalement sur son ordinateur, afin que les commandes de firebase soient reconnues par son ordinateur. Pour cela, j'ai tapé dans un terminal, la commande `$ npm install -g firebase-tools`.

Dernière étape avant de déployer l'application sur l'hébergeur de Firebase, il faut se connecter à un compte Google. Pour cela, j'ai tapé la commande `$ firebase login` dans un terminal, qui va ouvrir une fenêtre de login, afin de pouvoir se connecter à un compte Google.

Il est nécessaire d'initialiser l'hébergeur pour savoir ce qui va être déployé. Pour cela, j'ai tapé la commande `$ firebase init`. Plusieurs questions vont être posées. La première question était de savoir ce qui allait être initialiser, j'ai choisi « Hosting ». La deuxième était de savoir si je voulais lier un projet Firebase existant ou si je voulais en créer un. Comme j'ai déjà créé un projet, je voulais lier un projet déjà existant, j'ai donc choisi mon projet qui est tb-example-pwa. Maintenant, il m'indique qu'un dossier public va être créé, c'est dans ce dossier qu'il faudra mettre tous les fichiers de l'application pour pouvoir l'héberger.

Après avoir déplacé tous les fichiers de l'application dans le dossier public, pour déployer l'application, j'ai tapé la commande `$ firebase deploy`.

Firebase a créé deux domaines qui sont liés à mon application, pour pouvoir accéder à mon application, il suffit d'aller sur un de ces domaines, qui sont https://tb-example-pwa.web.app/ et https://tb-example-pwa.firebaseapp.com/.

Il est conseillé avant de déployer des changements qui seront effectués sur l'application d'émuler l'application localement, en tapant la commande `$ firebase emulators:start`, ainsi on peut voir les changements qui sont effectué sur l'application et déployer l'application par la suite. L'adresse locale pour accéder à l'application après avoir tapé la commande est localhost:5000.

5. Interview

5.1 Interview 1

5.1.1 Profil

Tableau 4 – Profil 1

Charles SCHMID

Situation	Travail	Entreprise
Étudiant 4ème année en DUAL – HEG Genève	Développeur informatique	Lysoft SA

5.1.2 Questionnaire

Tableau 5 – Questionnaire 1

Questions	Réponses
Quelles sont les technologies frontend que vous utilisez pour le développement d'application web et/ou mobile ?	ReactJs, React Native, VanillaJs, Android Studio, Ionic
Pour quelle raison vous n'utilisez pas la technologie des PWA ?	Manque de connaissance, pas de réel besoin des clients jusqu'à maintenant
(Explication) + Maintenant que vous connaissez plus la technologie des PWA, pensez-vous qu'il vous sera utile et comptez-vous l'utiliser ?	Je pense que ça pourrait être intéressant pour certains projets
La technologie des PWA est-il la solution pour le développement web/mobile ? **Oui/Non** Si oui : - Est-ce qu'une PWA peut remplacer une application native mobile ? Si non : - Pourquoi ?	Cela dépend de ce que tu veux réellement, j'aurais tendance à dire qu'il sera plus facile de développer une page web qu'une application native à première vue. Si un client me demande une app vitrine pour son magasin pourquoi pas. Mais si tu dois commencer à utiliser les senseurs de l'appareil pour le geofencing, les notifications push, l'interaction avec d'autres app comme ton calendrier, ça commence à devenir plus compliqué.

5.2 Interview 2

5.2.1 Profil

Tableau 6 – Profil 2

Micael CID

Situation	Travail	Entreprise
Employé	Full Stack Developper	IEC

5.2.2 Questionnaire

Tableau 7 – Questionnaire 2

Questions	Réponses
Quelles sont les technologies frontend que vous utilisez pour le développement d'application web et/ou mobile ?	Angular, Blazor, React, Vue, PWA
Pour quel type d'application web et/ou mobile utilisez-vous la technologie des PWA ?	J'ai dû réaliser une application pour une agence de taxi en ligne. J'ai utilisé la technologie de PWA, car elle s'accordait parfaitement à la demande du mandant. Une application rapide après le chargement initial grâce au pré-caching, au mode hors-ligne et en plus du push notification, qui ajoute une interaction avec le client
La technologie des PWA est-il la solution pour le développement web/mobile ? **Oui/Non**	PWA peut-être une solution, mais pour le moment, il est impossible qu'une PWA soit la solution pour le mobile principalement.
Si oui : - Est-ce qu'une PWA peut remplacer une application native mobile ?	PWA peut remplacer une app native, dans certains cas. Notamment dans le eCommerce, on peut voir de grosse structure utiliser les PWA. Principalement pour des questions de temps de développement et d'argents, les PWA sont imbattables à ce niveau-là.
Si non : - Pourquoi ?	Une application native pour le moment est plus performante qu'une PWA sur mobile

5.3 Interview 3

5.3.1 Profil

Tableau 8 – Profil 3

Marin VERSTRAETE

Situation	Travail	Entreprise
Employé	Développeur	WIDE Agency

5.3.2 Questionnaire

Tableau 9 – Questionnaire 3

Questions	Réponses
Quelles sont les technologies frontend que vous utilisez pour le développement d'application web et/ou mobile ?	Personnellement et à mon travail j'utilise Laravel et/ou Symfony (frameworks fullstack PHP) pour le backend et VueJS/React pour le frontend. Pas de dev mobile natif pour ma part, sauf si on compte une SPA/PWA comme tel
Pour quel type d'application web et/ou mobile utilisez-vous la technologie des PWA ?	J'utilise les PWA pour des apps web très lourdes en logique métier, ça permet d'avoir une app très rapide après le chargement initial, grâce à la mise en cache, mode hors-ligne, etc. et d'éviter le rechargement de la page
La technologie des PWA est-il la solution pour le développement web/mobile ? **Oui/Non**	Je ne pense pas qu'il existe une réponse absolue, cela dépend des besoins de l'application
Si oui : - Est-ce qu'une PWA peut remplacer une application native mobile ?	Une PWA peut remplacer une app native si les besoins ne sont pas trop intensifs niveau performance et permissions, dans ce cas une PWA est plus simple à mettre en place initialement et à maintenir.
Si non : - Pourquoi ?	Dans le cas où les performances sont cruciales (calcul intensif, jeu vidéo, etc.) et/ou l'app a besoin de permissions spéciales (accès aux fichiers, etc.) une app native est nécessaire.

6. Conclusion

Dans l'ensemble, je suis très satisfait de mon travail de Bachelor et de son déroulement. En effet j'ai pu échanger, partager et discuter durant les interviews avec des personnes qui travaillent dans le domaine du développement web et ainsi ajouter une plus-value importante à mon travail.

Ce travail m'a permis d'approfondir mes connaissances dans le développement web, même si j'ai développé une PWA simple, j'ai pu apprendre sur les technologies web en général. Du développement frontend avec des langages de programmation, tels que HTML, CSS ou JS jusqu'au développement backend en utilisant des modules de Firebase, comme Firestore Database ou Hosting.

Personnellement, avant de réaliser ce travail, je ne connaissais pas l'existence des Progressive Web App, ainsi que toutes les fonctionnalités qu'un navigateur web moderne propose. J'ai pu apprendre une bonne façon d'utiliser le cache, ce qui va me servir dans mes développements futurs ou encore l'utilisation d'indexedDB avec le mode hors-ligne.

Pour répondre à ma problématique, je pense que les PWA sont une solution pour les applications web et mobile dans une certaine mesure. Cette technologie est encore jeune, mais très prometteuse, malgré les restrictions de certains OS pour l'utilisation des fonctionnalités hardware, je pense que cette technologie va évoluer en bien dans le futur et les OS vont être obligés d'accepter les PWA, notamment grâce à l'appui de Google.

Je savais déjà dans quel domaine de l'informatique je voulais travailler, mais grâce à la réalisation de ce travail, j'en suis maintenant convaincu, le domaine du web est ce qui me correspond le mieux.

Bibliographie

DIVANTE, 2019. THE PWA BOOK [en ligne]. Wrocław : Divante eCommerce Software House. [Consulté le 7 juillet 2021]. Disponible à l'adresse : https://divante.com/pwabook/ [accès gratuit]

The Ninja Net, 2019. PWA Tutorial for Beginners [playlist]. YouTube [en ligne]. 6 mai 2019. [Consulté le 9 juillet 2021]. Disponible à l'adresse : https://www.youtube.com/watch?v=4XT23X0Fjfk&list=PL4cUxeGkcC9gTxqJBcDmoi5Q2pzDusSL7

Sam Richard & Pete LePage, 2020. What are Progressive Web Apps?. Web.dev [en ligne]. 6 janvier 2020. Dernière modification de la page le 24 février 2020. [Consulté le 10 avril 2021]. Disponible à l'adresse : https://web.dev/what-are-pwas/

Ajax (informatique). Wikipédia : l'encyclopédie libre [en ligne]. Dernière modification de la page le 4 novembre 2020 à 12 :17. [Consulté le 12 avril 2021]. Disponible à l'adresse : https://fr.wikipedia.org/wiki/Ajax_(informatique)

XMLHttpRequest. Wikipédia : l'encyclopédie libre [en ligne]. Dernière modification de la page le 5 juillet 2021 à 15 :11 (UTC). [Consulté le 12 avril 2021]. Disponible à l'adresse : https://en.wikipedia.org/wiki/XMLHttpRequest

XMLHttpRequest. Wikipédia : l'encyclopédie libre [en ligne]. Dernière modification de la page le 5 juillet 2021 à 15 :11 (UTC). [Consulté le 12 avril 2021]. Disponible à l'adresse : https://en.wikipedia.org/wiki/XMLHttpRequest#:~:text=The%20XMLHttpRequest%20object%20was%20accessible,released%20on%20June%205%2C%202002.

Swisstomato. Combien coûte une application?. Swisstomato [en ligne]. [Consulté le 28 juin 2021]. Disponible à l'adresse : https://swisstomato.ch/2020/06/11/developper-une-application-combien-ca-coute/

Team igeneve. TARIF HORAIRE D'UN DÉVELOPPEUR (EN SUISSE). Le journal du geek [en ligne]. [Consulté le 28 juin 2021]. Disponible à l'adresse : https://igeneve.com/le-journal-du-geek/tarif-horaire-dun-developpeur-en-suisse/

GEEKWORKERS. Geek workers [en ligne]. [Consulté le 28 juin 2021]. Disponible à l'adresse : https://geekworkers.ch/developpement-application-mobile-ios-android/

IONOS, 2019. IndexedDB : tutoriel pour la mémoire dans le navigateur. IONOS [en ligne]. 7 mai 2019. [Consulté le 1 juillet 2021]. Disponible à l'adresse : https://www.ionos.fr/digitalguide/sites-internet/developpement-web/indexeddb/

React. 2021. Documentation. ReactJS [en ligne]. [Consulté le 3 juillet 2021]. Disponible à l'adresse : https://fr.reactjs.org/docs/getting-started.html

React. 2021. Documentation. React Native [en ligne]. [Consulté le 3 juillet 2021]. Disponible à l'adresse : https://reactnative.dev/docs/getting-started

iamshaunjp, 2019. iamshaunj/pwa-tutorial. GitHub [en ligne]. 1 mai 2019. [Consulté le 17 avril 2021]. Disponible à l'adresse : https://github.com/iamshaunjp/pwa-tutorial/tree/lesson-2

Apple Inc, 2018. Notification Programming Guide for Websites. Developper Apple [en ligne]. Dernière modification de la page le 9 avril 2018. [Consulté le 2 juillet 2021]. Disponible à l'adresse : https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/NotificationProgrammingGuideForWebsites/PushNotifications/PushNotifications.html#//apple_ref/doc/uid/TP40013225-CH3-SW1

Paige Copper, 2020. 25 statistiques Twitter à connaître en 2020. Hootsuite [en ligne]. 1 janvier 2020. [Consulté le 2 juillet 2021]. Disponible à l'adresse : https://blog.hootsuite.com/fr/statistiques-twitter/

Google Developpers, 2017. Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage. Web [en ligne]. Dernière modification de la page le 3 novembre 2017. [Consulté le 2 juillet 2021]. Disponible à l'adresse : https://developers.google.com/web/showcase/2017/twitter

Thomas Le Coz, 2019. La création d'application mobile hybride iOS et Android. Nativio [en ligne]. 3 septembre 2019. [Consulté le 2 juillet 2021]. Disponible à l'adresse : https://nativio.net/creation-application-mobile-hybride/

Annexe 1 : fichier manifest.json

```
{
  "name":"Movie",
  "short_name":"Movie",
  "start_url":"/index.html",
  "display":"standalone",
  "background_color":"#FF8816",
  "theme_color":"#FF8816",
  "orientation":"portrait-primary",
  "icons": [
    {
      "src": "./img/maskable_icons/maskable_icon_x72.png",
      "type": "image/png",
      "sizes": "72x72",
      "purpose": "any maskable"
    },
    {
      "src": "./img/maskable_icons/maskable_icon_x96.png",
      "type": "image/png",
      "sizes": "96x96",
      "purpose": "any maskable"
    },
    {
      "src": "./img/maskable_icons/maskable_icon_x128.png",
      "type": "image/png",
      "sizes": "128x128",
      "purpose": "any maskable"
    },
    {
      "src": "./img/maskable_icons/maskable_icon_x192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"
    },
    {
      "src": "./img/maskable_icons/maskable_icon_x384.png",
      "type": "image/png",
      "sizes": "384x384",
      "purpose": "any maskable"
    },
    {
      "src": "./img/maskable_icons/maskable_icon_x512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose": "any maskable"
    }
  ]
}
```

Annexe 2 : fichier sw.js

```
const staticCacheName = "site-static-v9";
const dynamicCacheName = "site-dynamic-v8";

const assets = [
  '/',
  '/index.html',
  '/js/app.js',
  '/js/ui.js',
  '/js/materialize.min.js',
  '/css/materialize.min.css',
  '/css/styles.css',
  '/img/maskable_icons/maskable_icon_x192.png',
  '/img/reel.png',
  'https://fonts.googleapis.com/icon?family=Material+Icons',
  'https://fonts.gstatic.com/s/materialicons/v87/flUhRq6tzZclQEJ-Vdg-
IuiaDsNc.woff2',
  '/pages/fallback.html',
];

// cache size limit function
const limitCacheSize = (name, size) => {
  caches.open(name).then(cache => {
    cache.keys().then(keys => {
      if(keys.length > size){
        cache.delete(keys[0]).then(limitCacheSize(name, size));
      }
    })
  })
}

//install service worker
self.addEventListener('install', evt => {
  console.log("service worker has been installed!");
  //Pre-caching static data from the server
  evt.waitUntil(
    caches.open(staticCacheName).then(cache => {
      console.log("caching shell assets");
      cache.addAll(assets);
    })
  );
});

//activate event
self.addEventListener('activate', evt => {
  console.log("service worker has been activated!");
  //cache versioning
  evt.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys
        .filter(key => key !== staticCacheName && key !== dynamicCache
Name)
```

```
        .map(key => caches.delete(key)))
    })
  );
});

//fetch event
self.addEventListener('fetch', evt => {
  //console.log("fetch event", evt);
  if(evt.request.url.indexOf('firestore.googleapis.com') === -1) {
    evt.respondWith(
      caches.match(evt.request).then(cacheResp => {
        return cacheResp || fetch(evt.request).then(fetchResp => {
          return caches.open(dynamicCacheName).then(cache => {
            cache.put(evt.request.url, fetchResp.clone());
            limitCacheSize(dynamicCacheName, 15);
            return fetchResp;
          })
        });
      }).catch(() => {
        if(evt.request.url.indexOf('.html') > -1) {
          return caches.match('./pages/fallback.html');
        }
      })
    );
  }
});
```

Annexe 3 : fichier app.js

```
if('serviceWorker' in navigator) {
  navigator.serviceWorker.register("./sw.js")
    .then((reg) => console.log("service worker registered!", reg))
    .catch((err) => console.log("service worker not registered!", err)
);
}

// Detects if device is on iOS
const isIos = () => {
  const userAgent = window.navigator.userAgent.toLowerCase();
  return /iphone|ipad|ipod/.test( userAgent );
}

// Detects if device is in standalone mode
const isInStandaloneMode = () => ('standalone' in window.navigator) &&
 (window.navigator.standalone);

// Checks if should display install popup notification:
if (isIos() && !isInStandaloneMode()) {
  let popup = document.getElementById("myPopup");
  popup.classList.toggle("show");
}
```

Annexe 4 : fichier db.js

```
//offline data
db.enablePersistence()
  .catch(err => {
    if(err.code == 'failed-precondition') {
        // probably multiple tabs open at once
        console.log("persistence failed");
    } else if (err.code == "umimplemented") {
        console.log("persistence is not available");
    }
  });

//real-time listener
db.collection('movies').onSnapshot((snapshot) => {
  //console.log(snapshot.docChanges());
  snapshot.docChanges().forEach(change => {
    //console.log(change, change.doc.data());
    if(change.type === 'added') {
      renderMovie(change.doc.data(), change.doc.id);
    }
    if(change.type === 'removed') {
      removeMovie(change.doc.id);
    }
  });
});

// add new movie
const form = document.querySelector('form');
form.addEventListener('submit', evt => {
  evt.preventDefault();

  const movie = {
    title: form.title.value,
    release_date: form.release_date.value,
  };

  db.collection('movies').add(movie)
  .catch(err => console.log(err));

  form.title.value = '';
  form.release_date.value = '';
});

//delete a movie
const movieContainer = document.querySelector('.movies');
movieContainer.addEventListener('click', evt => {
  if(evt.target.tagName === 'I') {
    const id = evt.target.getAttribute('data-id');
    db.collection('movies').doc(id).delete();
  }
});
```

Annexe 5 : fichier ui.js

```
const movies = document.querySelector('.movies');

document.addEventListener('DOMContentLoaded', function() {
  // nav menu
  const menus = document.querySelectorAll('.side-menu');
  M.Sidenav.init(menus, {edge: 'right'});
  // add movie form
  const forms = document.querySelectorAll('.side-form');
  M.Sidenav.init(forms, {edge: 'left'});
});

//render movie data
const renderMovie = (data, id) => {
  const html = `
    <div class="card-panel movie white row" data-id="${id}">
      <img src="./img/reel.png" alt="recipe thumb">
      <div class="movie-details">
        <div class="movie-title">${data.title}</div>
        <div class="movie-ingredients">${data.release_date}</div>
      </div>
        <div class="movie-delete">
          <i class="material-icons remove" data-
id="${id}" name="delete">delete_outline</i>
        </div>
    </div>`;

    movies.innerHTML += html;
};

//remove movie from DOM
const removeMovie = (id) => {
  const movie = document.querySelector(`.movie[data-id=${id}]`);
  movie.remove();
};
```