Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Behind Food

Amélioration de l'application mobile sur la face cachée des aliments industriels

Rapport

Semestre 5
2021-2022

Étudiant : Grégory Geinoz
gregory.geinoz@edu.hefr.ch

Professeur : Pascal Bruegger
Conseiller/Mandant : Samuel Fringeli

Hes·so // FRIBOURG FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Version : 2.0
Fribourg, 02.11.2021

Table des matières

1 Introduction

Behind food est une application mobile utilisée dans le cadre d'une exposition sur le développement durable. Elle permet aux visiteurs d'explorer les faces cachées de différents aliments du quotidien au moyen d'un parcours de divers thèmes reliés à ces aliments. Ce parcours donne accès à des images, à des vidéos et à des textes pour illustrer les caractéristiques des aliments concernés. Ces éléments sont mis à jour par l'équipe qui gère l'exposition, au moyen d'une interface backend, et sont accessibles depuis l'application grâce à une API. Dans la version actuelle de l'application, c'est une WebView qui est chargée et qui affiche les images et vidéos au fur et à mesure du parcours de l'utilisateur dans la structure, mais les médias ne sont pas sauvegardés dans le stockage local de l'application, ce qui rend impossible l'utilisation de celle-ci hors-ligne. Comme l'exposition a pour objectif de fonctionner entièrement hors-ligne, il serait nécessaire de faire en sorte que les données affichées soient téléchargées localement, avec un système permettant d'actualiser les dernières modifications effectuées sur le backend par l'utilisateur de l'application.

1.1 Contexte

L'application de base [1] est une application web qui utilise la bibliothèque ZircleUI [2], une interface utilisateur simple mais intelligente avec une navigation zoomable intégrée à travers des cercles, affichée dans une application iOS avec un WebView de SwiftUI. Tout le contenu affiché, y compris le texte, les images et les vidéos, est obtenu à partir d'une API créée par le mandant.

1.2 Objectifs

Le projet est constitué d'objectifs principaux et secondaires. Les objectifs principaux sont à réaliser en priorité.

1.2.1 Objectifs principaux

1. **Application multiplateforme**
   L'idée est d'adapter l'application iOS de base en une application multiplateforme. La technologie utilisée pour ce projet est documentée dans la section "Analyse" de ce document entre Cordova, Flutter, React Native, Ionic ou Xamarin.

2. **Compatibilité hors-ligne**
   L'application doit devenir compatible hors-ligne. Cela signifie qu'elle est capable de vérifier la connexion de l'appareil et de s'adapter selon cette dernière. Elle doit être capable de stocker son contenu, de le récupérer si l'appareil est hors-ligne, et de mettre à jour le contenu si l'API a été mise à jour quand l'appareil est en ligne.

1.2.2 Objectifs secondaires

1. **Mises à jour**
   Ajouter les applications Android et iOS respectivement sur le Play Store et l'App Store et gérer les futures éventuelles mises à jour de ces applications mobiles.

1.3 Outils de collaboration

Le code et les documents officiels de ce projet sont postés sur le GitLab de l'école (autorisations à demander) et les réunions entre les différents acteurs du projet se font soit en présentiel au bureau de M.Bruegger (D20.17) soit avec Microsoft Teams.

1.4 Structure du rapport

Le projet est organisé selon un modèle hybride cascade et agile avec la structure suivante :

1. Analyse
2. Conception
3. Implémentations
4. Tests
5. Conclusion
6. Documentation

L'analyse permet de trouver les directions avec lesquelles les implémentations sont codées. Puisqu'il s'agit d'une amélioration d'une application déjà existante, la conception a déjà été étudiée avec le mandant et ajoute simplement les objectifs à la conception existante. Des diagrammes d'explication du fonctionnement de l'application de base et de l'application améliorée sont tout de même disponible dans la section "Conception" de ce document. Les implémentations sont en fait des sprints référençant chaque objectif du projet. Chaque objectif est implémenté, testé et présenté au professeur et au mandant afin d'être confirmé pour pouvoir passer au sprint suivant. Une fois ces sprints réalisés, la phase de test confirme les implémentations ou permet de les corriger. La phase de documentation se déroule sur l'entier du projet, en commençant par ce document qui est écrit au fur et à mesure de l'avancement du projet.

2 Analyse

2.1 Les solutions multiplateformes

Il s'agit pour ce premier objectif de convertir l'application de base iOS en une version cross-plateforme en utilisant une des technologies décrites dans les chapitres ci-dessous. L'application de base affiche simplement l'application web dans une WebView du WebKit de iOS [3]. Afin de s'orienter au mieux pour le deuxième objectif, il faut que cette technologie supporte non seulement les WebViews mais également une solution de stockage local en cache, comme l'API IndexedDB [4], qui est nécessaire pour la persistance des données au sein d'un navigateur internet. Une autre approche est de tout adapter en code natif pour chaque plateforme pour simplifier l'accès au stockage local des appareils.

2.1.1 Natif

Il s'agit de la façon la plus pragmatique de développer une application sur plusieurs plateformes : simplement créer une application spécialement implémentée pour chaque plateforme. Une application iOS en Swift, une application Android en Java et une application web pour ordinateurs en HTML5, CSS3 et JavaScript. Dans notre cas, les applications web et iOS existent déjà, il suffirait donc de simplement implémenter une version similaire en Android puis d'améliorer l'application web dans le Goal 2. Cette approche amène plusieurs avantages :

- Une disponibilité simplifiée et direct au matériel de chaque appareil (GPS, micro, caméra, stockage interne,...)
- Une UX toujours adaptée à chaque appareil
- Moins besoin de librairie externe open-source pour ajouter des fonctionnalités qui sont déjà disponibles en natif

Évidemment, cette méthode demande autant de code qu'il y a de plateforme à disposition, mais pour ce travail, cela ne devrait pas être un obstacle à prendre en compte. Il y a néanmoins une difficulté à relever, les restrictions de JavaScript. En effet, pour des raisons de sécurité, JavaScript a toujours eu des accès très limités au matériel des plateformes, c'est pourquoi un script ne pourra jamais accéder directement aux disques durs d'un ordinateur, il est entièrement encapsulé dans le navigateur. Il est donc nécessaire d'adapter l'application web pour qu'elle puisse stocker des fichiers dans le cache des navigateurs, car c'est l'unique zone de stockage qu'un script JavaScript pourra accéder. Le côté application web pour ordinateurs ne devraient pas poser de problème de ce côté, mais il s'agit

maintenant de savoir s'il est possible à des WebViews d'accéder au stockage local des téléphones, et si non s'il est possible de stocker des fichiers dans le cache des WebViews. Si ces choses sont impossibles, cela veut dire qu'il faut adapter entièrement en natif l'application web, sans utiliser de WebView. Il est tout à fait possible d'utiliser du JavaScript avec Swift ou Java (avec JavaScriptCore et AndroidJSCore respectivement), l'intégration du framework ZircleUI et la grande majorité de la logique de l'application web existante ne sera donc pas un problème, mais cela demandera beaucoup plus de codage que les solutions suivantes.

2.1.2 Xamarin

Xamarin est une plateforme open-source qui permet de créer des applications mobiles sur iOS, Android et Windows Phone. Le code partagé de l'application s'écrit en C# [5]. Un des frameworks de Xamarin est Xamarin.Forms [6], qui fournit les WebView [7]. Lors de la rédaction de ce document et durant une bonne partie du semestre 5, Xamarin est un sujet majeur du cours "Développement Mobile Cross-Plateforme".
Par exemple, UPS utilise Xamarin pour son application mobile.

2.1.3 Cordova

Apache Cordova est un framework de développement mobile open-source. Il permet de créer des applications multi-plateformes en utilisant les technologies web standards actuelles HTML5, CSS3 et JavaScript. Cordova utilise de base des WebViews afin d'afficher tout l'interface graphique des applications, ce framework est donc un bon choix pour s'implifier l'utilisation des WebViews pour les applications multiplateformes. Le plus gros avantage de Cordova est qu'il met à disposition une grande quantité de plugins (les Core Plugins notamment) permettant l'accès à des composants natifs des plateformes, ce qui est très rassurant pour le bon fonctionnement des deux Goals de ce projet.
Il n'y a plus beaucoup de grosses compagnies qui utilisent Cordova pour leurs applications mobiles. Adobe utilisait Cordova comme base pour leur solution de développement cross-plateforme Adobe PhoneGap, qu'ils ont décidé d'abandonner. Cordova, lui, continue d'exister et reste une référence du cross-plateforme.

2.1.4 Flutter

Flutter est un kit de développement logiciel (SDK) open-source créé par Google. Il permet de créer des applications multiplateformes sur Android, iOS, mais aussi Windows, Linux, Mac et le web à partir d'une seule base de code, le Dart. Le package webview_flutter permet d'importer les WebViews dans une application Flutter. Malheureusement, Flutter bloque IndexedDB et le seul moyen de contourner cette restriction est d'utiliser un plugin qui est en fait un simple wrapper Flutter pour IndexedDB, mais il semble que Flutter ne permette pas de mettre en cache des gros fichiers [8].
Par exemple, eBay Motors, BMW ou encore Google Ads utilisent Flutter pour leurs applications mobiles.

2.1.5 React Native

React Native est un framework de développement mobile open-source créé par Facebook. Le code est écrit en JavaScript puis est rendu en natif. C'est une évolution de React, une librairie JavaScript également créée par Facebook qui permet de créer des interfaces utilisateurs mais uniquement ciblé pour les navigateurs. Mais il s'agit d'un projet relativement récent, ce qui rend la recherche d'information compliquée et la documentation doit encore s'étoffer.
Puisqu'il s'agit d'une invention de Facebook, la grande majorité de leurs applications sont en React Native (Facebook, Instagram,...), mais d'autres tout autant connues (UberEats, Oculus, Discord, Skype, ...) sont également implémentées en React Native.

2.1.6 Ionic

Ionic est un framework en partie open-source de développement cross-plateforme utilisant les technologies web. Initialement, Ionic était basé sur AngularJS et Apache Cordova. Dans sa version 2.0, Ionic utilise Angular 2 qui permet de convertir du TypeScript vers du JavaScript, il est donc nécessaire de coder en TypeScript.
McDonald's Turquie, McLaren, Diesel, Sworkit sont des exemples de grosses applications utilisant Ionic.

2.1.7 Tableau comparatif

Le tableau suivant compare chacune des technologies précédemment présentées avec des critères pondérés afin de choisir au mieux la solution idéale pour ce projet.

Goal 1							
Critères et pondération		Native	Xamarin	Cordova	Flutter	React Native	Ionic
Connaissance personnelle	4	9 (cours)	10 (cours)	10 (cours)	1	1	1
Réutilisabilité du code fourni	5	1	5	10	5	5	5
Communauté et documentation	3	10	8	8	8	3	3
Temps de développement gagné	5	3	5	10	2	2	2
Totaux	**17**	**86**	**114**	**164**	**63**	**48**	**48**

2.1.8 Conclusion

Selon le tableau précédent, Cordova est la solution favorite pour les besoins de ce projet. Même s'il s'agit du doyen de toutes ces solutions, Cordova reste une référence dans le domaine, possède une communauté gigantesque, est particulièrement simple à adapter pour le projet car il s'agit de code d'application web qui crée une application mobile. Je pense donc que cette approche sera la mieux adaptée.

2.2 Les solutions hors-ligne

Afin de rendre l'application existante utilisable sans connexion, il faut que les données récupérées depuis l'API du mandant puissent persister d'une manière ou d'une autre. Pour ce faire, deux possibilités se présentent :

2.2.1 Convertir l'application web en application mobile native

Convertir entièrement l'application web avec une solution multiplateforme vue plus haut, afin d'accéder facilement au stockage de l'appareil mobile utilisé.

2.2.2 Convertir l'application web en application web multiplateforme

Améliorer l'application web actuelle afin qu'elle puisse accéder au cache des navigateurs sur lesquels elle tourne pour y enregistrer les données qu'elle utilise, afin de les récupérer facilement sans connexion dans le cas où le réseau tomberait ou qu'aucune connexion n'est disponible. Tout dépend donc à quel endroit les données sont stockées.
La première solution permet d'utiliser le stockage interne de l'appareil mobile et de sortir du champ d'application d'une WebView, mais demande une bien plus grande quantité de codage et de prise en main, quelque soit la technologie cross-plateforme choisie.
La seconde solution ne demande qu'une adaptation cross-plateforme de l'application mobile actuelle utilisant simplement une WebView afin d'afficher l'application web convertie en application "web-mobile" lancée dans une WebView, mais demande d'utiliser les caches des navigateurs car il n'est pas possible de sortir du champ d'application de ces derniers.
L'idée de la seconde solution est de faire comme cette application web d'exemple qui, lors de la première visite avec une connexion, stocke toutes les vidéos (quelques Mo par vidéos, un peu comme l'application web actuelle) ainsi que les fichiers statiques comme le CSS et le JavaScript dans une IndexedDB. Grâce à ce processus, mettre son appareil en mode avion (ou dans un scénario plus réaliste, perdre sa connexion) n'empêche pas l'utilisation du site. Les vidéos s'affichent et peuvent toujours être lues sans aucun problème. Comme mentionné plus haut, les dernières versions de Safari et Chrome sont les versions 15 et 94, IndexedDB est parfaitement supporté sur ces deux versions récentes qui se mettent pour la grande majorité des appareils automatiquement à jour [9].

En contrôlant la connexion de l'appareil dès le démarrage de l'application [10] [11], il est possible dans le cas où il y a une connexion de vider toute la IndexedDB et de télécharger tous les fichiers de l'API afin de mettre leurs versions à jour dans la IndexedDB. Dans le cas où il n'y a pas de connexion, l'application va directement chercher dans sa IndexedDB les fichiers afin de créer l'application web sans avoir à télécharger quoi que ce soit du net. **Ceci inclut que la toute première utilisation de l'application se fasse avec une connexion internet.** Mais puisque l'objectif secondaire est de déployer cette application sur les stores, les utilisateurs désireux de l'acquérir n'auront pas le choix d'avoir une connexion internet, ce qui rend cette précondition plutôt triviale.

2.2.3 Tableau comparatif

Le tableau suivant compare quelques technologies afin de stocker des données autant du côté natif que du côté web, avec des critères pondérés afin de choisir au mieux la solution idéale pour ce projet. Puisqu'il a décidé dans la section précédente que Cordova est l'approche à utiliser pour ce projet, toutes les solutions présentées ci-dessous sont utilisables avec Cordova.

Goal 2				
Critères et pondération		Plugin File	IndexedDB	Service Workers
Taille maximale des fichiers stockés	5	10 (même que le stockage)	10 (1GB à 50% du stockage)	1 (52MB)
Taille maximale du stockage mis à disposition	5	10	10	1
Persistance en cas de stockage libre bas	5	10	10	10
Communauté et documentation	3	8	8	5
Totaux	**18**	**174**	**174**	**75**

2.2.4 Conclusion

Il y a donc deux solutions qui sont à égalité : Plugin File et IndexedDB. Il faut tester ces deux solutions dans les extrêmes, notamment pour la taille maximale allouée pour l'application et la taille maximale par fichier stocké. Ces tests sont réalisés après le premier sprint d'implémentation, une fois le Goal 1 terminé et validé puisque Cordova sera utilisé dans tous les cas.

3 Conception

3.1 Application de base

Ci-dessous un schéma simplifié de l'application web actuelle dont voici une explication :

1. Une WebView SwiftUI sur un iPhone ou un iPad lance un URL (https://adelente-admin.samf.me/app), ceci est la WebApp hébergée sur le serveur Strapi des mandants
2. La WebApp va ensuite faire un XMLHTTPRequest pour acquérir le fichier JSON fourni par l'API des mandants (également du côté de Strapi)
3. Ce dernier est ensuite décortiqué :
    - Si le média est un texte, il est immédiatement trouvé dans le JSON et est directement placé dans une balise div
    - Si le média est une image ou une vidéo, le JSON fournit une URL qui pointe vers le serveur Strapi qui héberge cette image ou vidéo, puis cet URL est placé dans une balise image ou video qui permet de les afficher

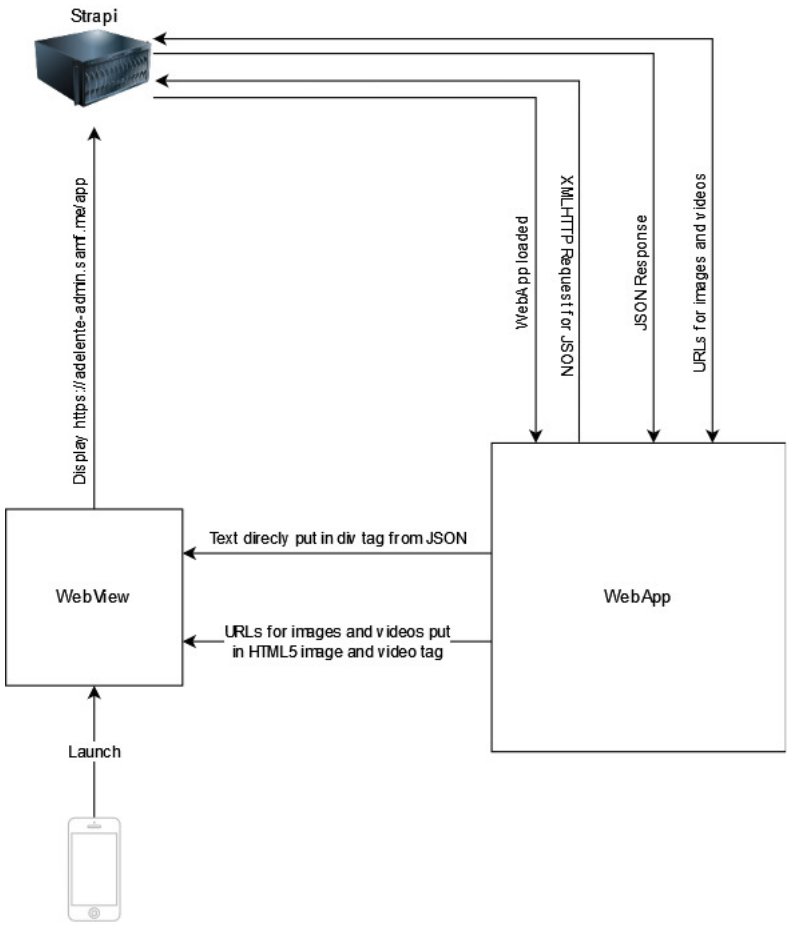

Figure 1 – Application web de base

3.2 Application modifiée

Ci-dessous un schéma du fonctionnement de l'application web cross-plateforme prévue dont voici une explication :

1. Une application web cross-plateforme Cordova nativement affichée dans une WebView, demande le fichier JSON à l'API Strapi
2. Une fois le JSON récupéré, il est comparé au fichier JSON mis en cache.
   - Dans le cas où il n'y aurait pas de JSON dans le cache, il serait caché immédiatement et passerait à la suite (première utilisation), tout le contenu est téléchargé et stocké localement.
   - Dans le cas où un JSON est caché, il est comparé au JSON fraîchement téléchargé. Si une ou plusieurs différences sont observées, elles sont traitées afin de récupérer le nouveau contenu et/ou remplacer le contenu mise à jour et/ou supprimer le contenu retiré.
   - Dans tous les cas, le JSON téléchargé écrase l'ancien JSON dans le cache.
3. Une fois ces contrôles terminés, la WebApp récupère et affiche tout le contenu mise en cache (connexion internet ou non).

Remarque : ce schéma ne couvre pas les points suivants :

1. Si dès le début il n'y a pas de connexion internet disponible, la WebApp récupère immédiatement le JSON et le contenu cachés. Si rien n'est caché, c'est qu'il s'agit d'une première utilisation ou que la plateforme ait effectué un vidage de cache indésirable, il faudra donc afficher un message d'erreur et de demande de connexion internet.
2. Le framework ainsi que les polices utilisés par la WebApp sont également cachés et ce avant le JSON. Ce sont également les premiers éléments récupérés dans le cache afin de pouvoir afficher correctement le contenu.

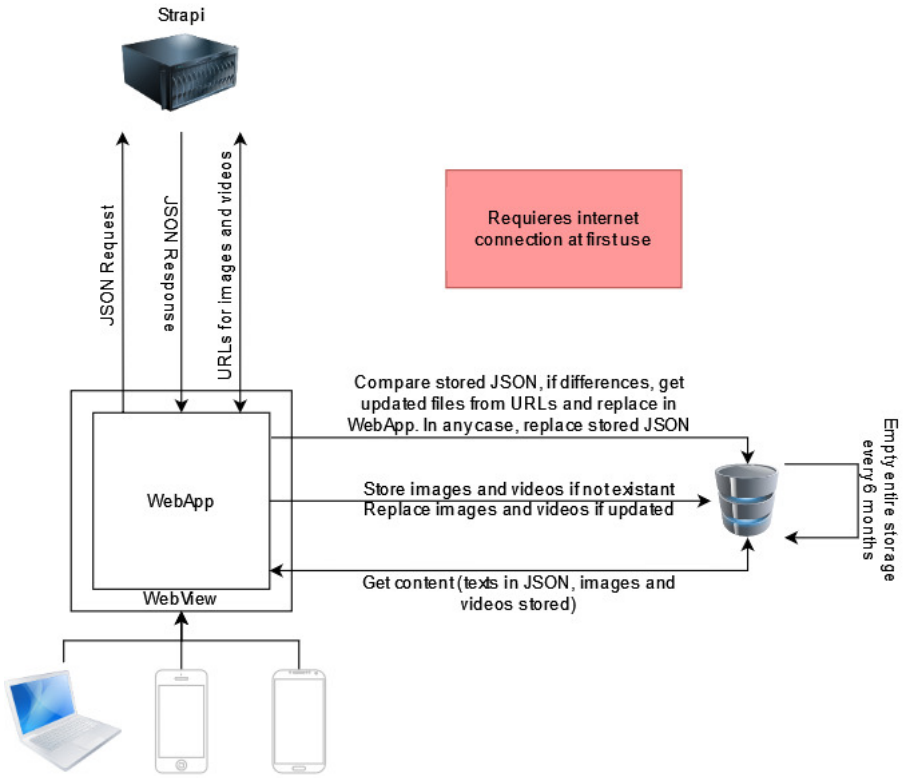

Figure 2 – Application web modifiée

4 Implémentations

5 Tests

6 Conclusion

7 Figures et Tables

7.1 Liste des figures

7.2 Liste des tables

8 Bibliographie

[1] Samuel Fringeli. Behind food. https://github.com/samuel-fringeli/behind-food, 2021. [En ligne; Page disponible le 11-octobre-2021].

[2] Juan Martin. Zircleui. https://zircleui.github.io/docs/, 2019. [En ligne; Page disponible le 11-octobre-2021].

[3] Apple Inc. ios wkwebview. https://developer.apple.com/documentation/webkit/wkwebview, 2020. [En ligne; Page disponible le 11-octobre-2021].

[4] W3C. Indexed database api 3.0. https://www.w3.org/TR/IndexedDB/, 2021. [En ligne; Page disponible le 11-octobre-2021].

[5] Microsoft. Qu'est-ce que xamarin ? https://docs.microsoft.com/fr-fr/xamarin/get-started/what-is-xamarin, 2021. [En ligne; Page disponible le 11-octobre-2021].

[6] Microsoft. Qu'est-ce que xamarin.forms ? https://docs.microsoft.com/fr-fr/xamarin/get-started/what-is-xamarin-forms, 2021. [En ligne; Page disponible le 11-octobre-2021].

[7] Microsoft. Xamarin.forms webview. https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/user-interface/webview?tabs=windows, 2021. [En ligne; Page disponible le 11-octobre-2021].

[8] Hossam Tarek. How do i import the dart indexeddb library in flutter? https://stackoverflow.com/questions/67490597/how-do-i-import-the-dart-indexeddb-library-in-flutter, 2021. [En ligne; Page disponible le 11-octobre-2021].

[9] Alexis Deveria. Can i use... indexeddb. https://caniuse.com/indexeddb, 2021. [En ligne; Page disponible le 11-octobre-2021].

[10] Microsoft. Xamarin.essentials: Connectivity. https://docs.microsoft.com/en-us/xamarin/essentials/connectivity?tabs=android, 2019. [En ligne; Page disponible le 11-octobre-2021].

[11] Apache. cordova-plugin-network-information. https://cordova.apache.org/docs/en/10.x/reference/cordova-plugin-network-information/, 2021. [En ligne; Page disponible le 11-octobre-2021].

[12] WhatIsMyBrowser.com. What's the latest version of chrome? https://www.whatismybrowser.com/guides/the-latest-version/chrome, 2021. [En ligne; Page disponible le 11-octobre-2021].

[13] WhatIsMyBrowser.com. What's the latest version of safari? https://www.whatismybrowser.com/guides/the-latest-version/safari, 2021. [En ligne; Page disponible le 11-octobre-2021].

[14] Apache. Store data - indexeddb. https://cordova.apache.org/docs/en/10.x/cordova/storage/storage.html#indexeddb, 2021. [En ligne; Page disponible le 11-octobre-2021].