PERRET Pierre-Yves
M2 MIAGE ISI
2021-2022

Rapport de projet

Développement d'applications mobiles

X4IP110

Avant propos

Ce projet devait être réalisé en binôme cependant Alexandre Cellier a été placé en arrêt au début du projet. J'ai donc réalisé l'ensemble du projet (recherche, mise en place technique et écriture de ce rapport) seul.

I. Introduction

Dans le cadre du module de Développement d'applications mobiles, j'ai réalisé un projet permettant de mettre en avant mon apprentissage de VueJS et de Capacitor pour réaliser une application PWA.
Pour cela, j'ai réalisé une application de gestion de bibliothèque fonctionnant sur web, tablette et smartphone Android, et utilisant la même base de code. Cette application embarque des fonctionnalités utilisables uniquement sur une plateforme native (APK installé sur smartphone ou tablette Android) comme le scan ou le partage.

II. L'application

1. Description

Pour ce projet, j'ai donc décidé de réaliser une application permettant de gérer sa bibliothèque. Concrètement après avoir effectué une recherche par ISBN, mots clés ou scan du code barre, il est possible d'ajouter ce livre à sa bibliothèque virtuelle. Ensuite, l'utilisateur peut indiquer s'il a lu le livre, auquel cas il pourra lui donner une note sur 5, laisser un avis mais aussi indiquer si son livre est disponible ou non (par exemple, il pourra indiquer en commentaire s'il a prêté son livre et à qui il l'a adressé, ou tout autre motif).

2. L'API

Après la définition globale du projet, il a fallut choisir l'API à utiliser. Après de longues recherches, j'ai trouvé un certain nombre d'API répondant à cet objectif, mais la plupart avec des contraintes telles que la limitation du nombre d'appel ou le coût. J'ai donc décidé d'utiliser l'API de google qui est très complète pour sa quantité de livre. Elle fourni une grande quantité de livres anglais, cependant, elle est au contraire assez pauvre en informations pour beaucoup de livres français (manque d'images, de notes, etc.).

3. Les plugins Capacitors

Pour répondre à la consigne, j'ai du définir au moins 2 plugins Capacitor pour ce projet.

J'ai choisi d'utiliser dans un premier temps le plugin « BarcodeScanner » permettant de scanner un code barre. Dans mon projet, cela permet de rechercher un livre simplement en scannant son code barre.
Son intégration a été relativement simple à mettre en place au vu de la documentation à disposition. A noter tout de même une limitation graphique ne permettant pas de limiter la zone de recherche du code barre.

Le deuxième plugin que j'ai mis en place est « CapacitorStorage » permettant la persistance des données. Dans mon projet, cela permet de sauvegarder les livres que j'ajoute à ma bibliothèque.
J'avais commencé la réalisation de cette persistance à l'aide de « Dexie » cependant, ce plugin ne semble pas fonctionner sur une plateforme native comme Android.
J'ai donc utilisé « CapacitorStorage » qui fonctionne à la fois sur web et sur Android.
Son intégration est assez simple à mettre en place encore une fois grâce à la documentation importante. A noter que ce plugin ne peut sauvegarder que des chaines de caractères et il a donc fallut transformer les objets livre en texte JSON afin d'assurer leur sauvegarde.
Les deux plugins sont relativement semblables même si « Dexie » permet de réaliser des requêtes plus complexe alors que sur le plugin capacitor il est obligatoire d'extraire toutes les entrées dans un tableau, puis de réaliser un filtre sur ce dernier.

Enfin, j'ai intégré « Share Capacitor », un plugin permettant de faire un partage sur l'ensemble des applications le permettant (sms, Facebook, Twitter, etc.) uniquement sur une plateforme native.
Son intégration a été très simple car la documentation est complète et qu'il a peu de fonctionnalité à implémenter.

III. Choix d'implémentation

1. Appels API / Base de données

Dans le but de centraliser les appels vers l'API et la base de données, j'ai mis en place un service pour ces deux fonctionnalités. Ces classes permettent de récupérer les données à partir de n'importe quel composant sans avoir à réimplémenter l'appel vers l'API ou la base de données. Cela permet de factoriser le code et de limiter les redondances qui sont complexes à maintenir.

2. Gestion des messages popup

Afin d'avertir l'utilisateur qu'une action s'est correctement effectuée ou non, j'ai mis en place un système de message popup. Pour cela, j'ai dans un premier temps réalisé un service en Singleton permettant d'ajouter une popup, d'en supprimer une ou de toutes les effacer. A l'ajout, le message et la couleur de la popup sont insérés dans un tableau consultable par le composant en charge d'afficher les popups. C'est cette utilisation par tableau qui nécessite l'utilisation d'un singleton sans lequel chaque instance aurait son propre tableau.
La mise en place de ce service permet de simplifier l'utilisation car il suffit d'appeler la méthode *show()*, avec le message à afficher et la classe permettant de jouer sur les couleurs, sur l'instance du singleton pour voir afficher la notification.
Après 5 secondes, la notification est détruite par le service pour ne pas encombrer l'affichage.

IV. Axes d'amélioration

Dans le cas d'une mise en production de cette application, j'ai identifié plusieurs points méritant une attention ou une amélioration.

Pour commencer, lors de l'affichage de la liste des livres présents dans la bibliothèque, il serait intéressant d'intégrer une pagination afin de garantir de bonnes performances même dans le cas où l'utilisateur possède beaucoup de livres. Cependant, cela n'a pas de sens dans le cas actuel car le plugin Capacitor de stockage ne permet pas de limiter le retour et de filtrer réellement une requête. Comme il est nécessaire de récupérer

l'ensemble des livres pour vérifier s'ils correspondent à la requête, une pagination ne changerait en rien les performances de l'application. Il faudrait donc utiliser un autre système de stockage.

Ensuite, l'utilisation d'une API plus complète est primordiale pour une utilisation agréable de l'application. Il faudrait ainsi utiliser une API payante mais plus complète comme celle proposée par la société « Electre Data Service » (API utilisée par l'application Gleeph).

Un autre point intéressant serait de mettre en place un système de commentaires visibles par l'ensemble de la communauté. La note des livres pourrait également être collaborative. Cette fonctionnalité nécessite cependant la mise en place d'un backend, d'une gestion d'utilisateur et d'une base de données, ce qui est beaucoup plus « lourd » que le fonctionnement actuel.

Enfin, on peut imaginer des propositions de livres similaires afin d'orienter l'utilisateur en fonction des livres qu'il a aimé. Cette option nécessite une base d'informations importantes sur les livres ainsi que la mise en place d'un algorithme de recommandation.

V. Bilan

Tout au long de ce projet, je n'ai pas eu de difficultés majeure et j'ai donc pu avancer de manière régulière.
Ce projet m'a permis d'apprendre les bases du Framework VueJS et de complété mes connaissances dans le Framework Angular car certains mécanisme peuvent s'y appliquer.
Ce projet m'a également permis de mieux comprendre le fonctionnement des PWAs.