IUT Paris Descartes 2018-2019

Applications Mobiles

Projet Médiathèque Android

Maud GELLEE Groupe 201

Arsène LAPOSTOLET Groupe 202

Sixtine TRAVERSE Groupe 201

Table des matières

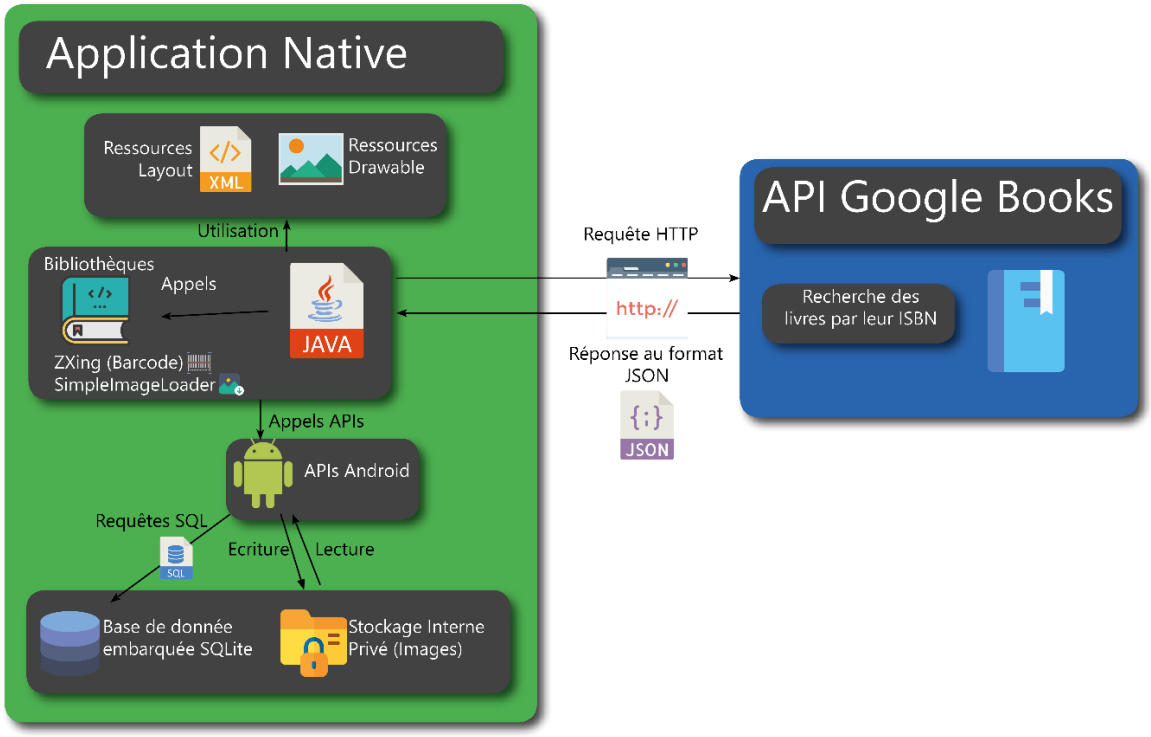
I/ Présentation de l'application

Introduction

Pour ce projet nous avons décidé de développer une application de bibliothèque personnelle, permettant à l'utilisateur de stocker certaines informations sur les livres qu'il possède (titre, auteur, résumé, ...). Ainsi lorsqu'il souhaite acheter un nouveau livre, il peut au préalable vérifier qu'il ne le possède pas déjà. Ce projet nous permettait ainsi de découvrir beaucoup de fonctionnalités. Nous devions utiliser un système de stockage de données pour stocker les livres, utiliser l'API google Books pour récupérer toutes les informations nécessaires facilement et appeler une autre application afin de pouvoir scanner le code barre d'un livre et récupérer directement ses informations grâce à ce dernier.

Architecture

Schéma de l'architecture de l'application



II/ Back End

Stockage des données

Pour le stockage des données, nous avons choisi la base de données embarquée SQLite. En effet les données étant propres à chaque utilisateur et n'ayant pas besoin d'être partagées en réseau, nous avons pensé que ce choix serait le plus judicieux à la fois pour des raisons de performances et d'autonomie, tout appel réseau étant coûteux en batterie.

Nous avons donc implémenté le Pattern DAO (Database Access Object) afin de faire l'interface entre notre classe du modèle *Livre* et la base de données, gérée par un *SQLiteOpenHelper*.

Le schéma de la base de données est extrêmement simple : une seule table qui permet de stocker les livres.

Nous utilisons aussi le stockage interne (privé) pour stocker les images de couverture des livres une fois qu'elles ont été téléchargées depuis le réseau. En effet, lorsqu'un livre est ajouté, son image est téléchargée pour être affichée sur l'interface. Afin de diminuer le trafic réseau et rendre l'application plus fluide, nous avons décidé de télécharger les images et de les stocker dans le stockage interne privé. Ainsi, l'image est téléchargée uniquement lors de son premier affichage, pour les suivants, elle est simplement lue depuis le stockage interne privé. Cette opération est infiniment plus rapide et n'utilise pas le réseau, ce permet d'économiser de la batterie.

Réseau

Nous utilisons différents services réseau dans notre application. La communication réseau (longue et parfois bloquante) est interdite sur le thread de l'UI pour préserver l'expérience de l'utilisateur. Ainsi nous utilisons l'API AsyncTask d'Android, qui encapsule un thread. Nous service réseaux étendent donc la classe AsyncTask en utilisant les paramètres de type selon ceux des paramètres et résultats du services.

De plus, pour traiter le résultat de service au bon moment, nous avons défini une interface fonctionnelle prenant le résultat en paramètre. Ainsi, nous pouvons la déclarer à la volée à l'initialisation du service, qui l'appelle lorsqu'il a fini. Les données sont ainsi traitées par le thread de l'UI au bon moment.

Tout d'abord, nous faisons une requête HTTP à l'API Google Books, qui nous fournit une réponse au format JSON. Nous avons donc utilisé l'API JSONObject afin de parser dans ce résultat les informations qui nous intéressent.

Ensuite nous faisons des appels réseaux pour récupérer les images à leur premier chargement. Nous utilisons à cet effet la bibliothèque OpenSource SimpleImageLoader, qui permet de lire un Bitmap depuis une URL http de façon claire et concise.

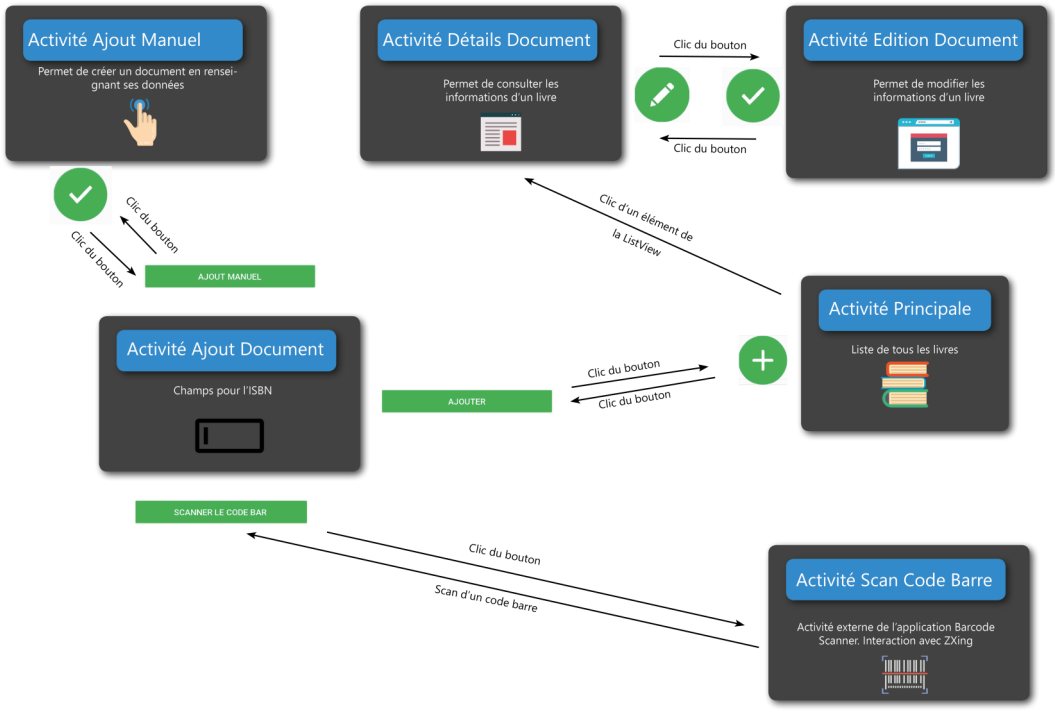
III/ Front End

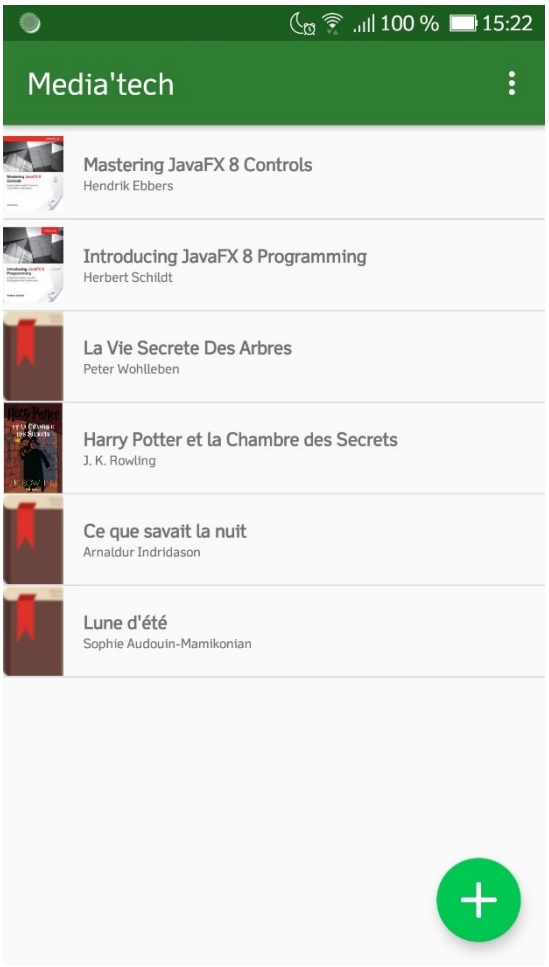
Design

Pour le design des vues, nous avons utilisé principalement des ConstraintsLayout, qui permet de créer rapidement des vues responsive et cohérentes avec le concepteur graphique d'Android studio. Les relations des contrôles avec leurs parents et entre eux permettant de respecter la cohérence de l'interface sur des écrans de toutes taille, et la rend robuste à la rotation.

La navigation dans l'application se fait grâce à des boutons d'actions flottants, qui en plus de donner un *look & feel* assez moderne à l'application sont très ergonomique, car ils prennent peu de place et sont très intuitifs à l'utilisation par leur légende pictographique.

Vues de l'application

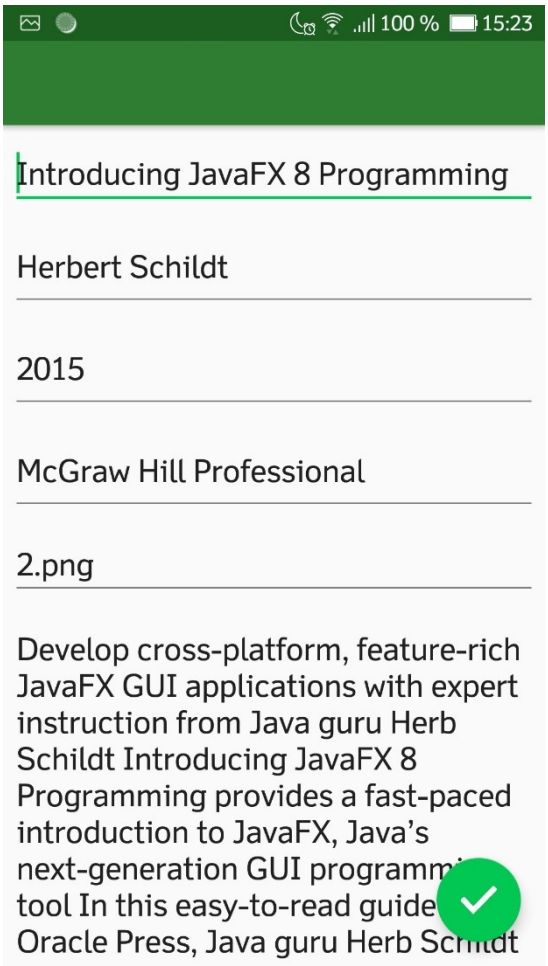




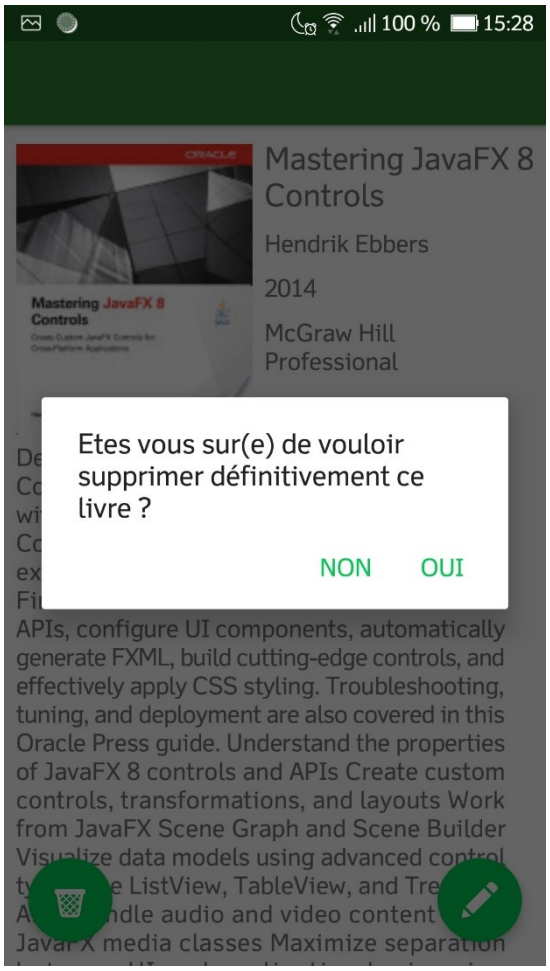
Cette vue est la vue d'accueil de l'application. Elle permet à l'utilisateur de consulter sa bibliothèque, en lui présentant pour chaque livre sa couverture son titre et son auteur. Cette vue est implémentée grâce à un contrôle graphique de ListView. Afin de la conformer à nos donner, nous avons créé un adapteur de liste personnalisé, *CustomAdapter*. Cela nous permet pour chaque indice de la liste de données, de retourner une vue pour la ListView, créée à partir d'un layout paramétré avec ses données puis inflaté pour donner un objet View. Cela permet de créer une ListView agréable et modulable de manière moins verbeuse et plus lisible qu'avec un *SimpleArrayAdapter*. Lors d'un clic sur un élément de la ListView, l'utilisateur accèdes à la vue détails. De plus un bouton d'action flottant permet d'accèder à la vue d'ajout de document.

Cette vue qui est servie à l'utilisateur lorsqu'il clique sur un élément de la ListView, à ce moment-là, le livre correspondant à l'entrée de la ListeView cliqué, est passée en SerializableExtra dans l'Intent. Ainsi, on peut afficher à l'utilisateur diverses informations sur le livre, ainsi que l'image de la couverture. Afin de ne pas bouleverser l'esthétique de la vue, nous avons installé une bibliothèque permettant d'utiliser un contrôle graphique de TextView qui permet de justifier le texte.

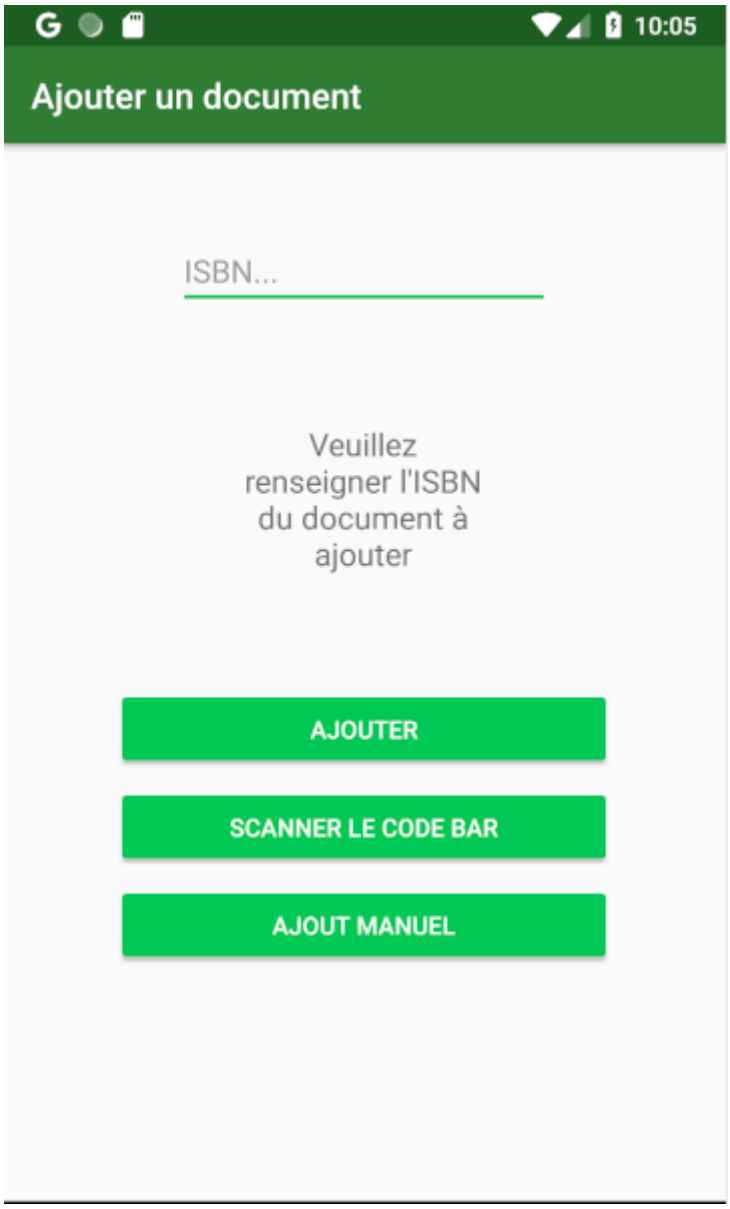
L'utilisateur dispose de deux boutons d'action flottants, l'un qui permet de supprimer le document, l'autre qui permet d'en modifier les informations.



Cette vue permet à l'utilisateur d'éditer les informations concernant le livre à travers des EditText. Il dispose d'un bouton d'action flottant pour valider ses modifications.

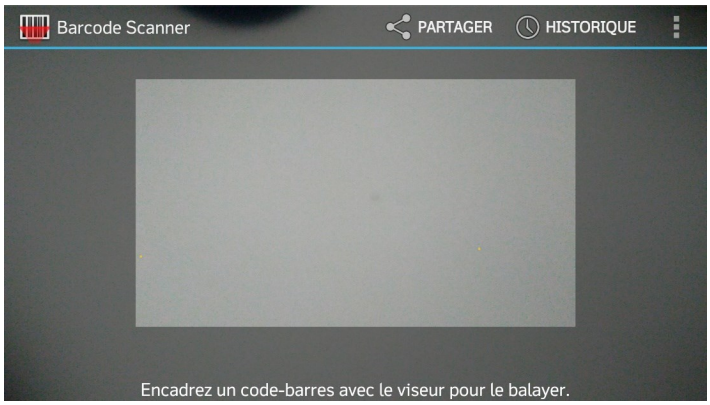


Lorsque l'utilisateur supprime une livre, on lui demande une confirmation, pour éviter toute suppression involontaire de données.

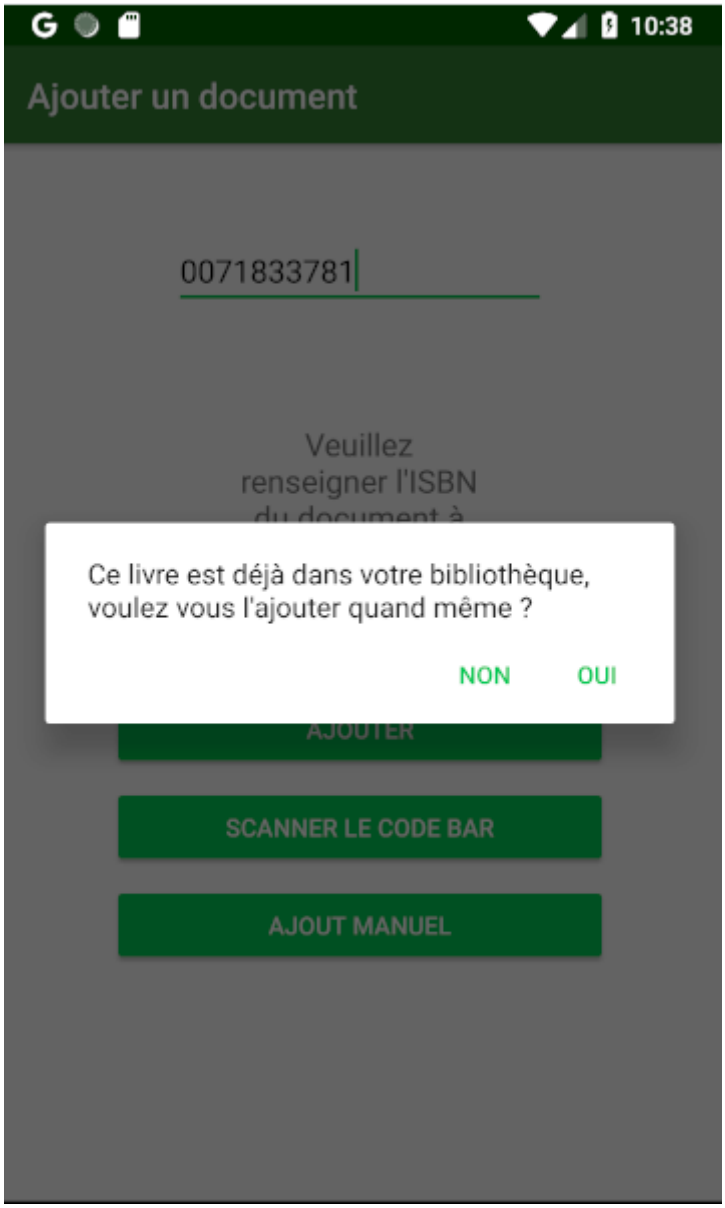


Cette vue permet à l'utilisateur d'ajouter un livre. Il a plusieurs options : taper directement l'ISBN ou alors le scanner grâce au code barre du livre, afin de gagner en temps et en ergonomie, ce qui se fait à partir d'une application tierce. Cela permet de charger les informations du livre à partir de l'API Google Books.

Il peut aussi entrer manuellement informations d'un livre (par exemple si celui-ci est introuvable à partir de son ISBN).



Cette vue, issue de l'application BarcodeScanner et intégrée grâce à la bibliothèque ZXIng permet de scanner le code barre d'un livre pour en lire l'ISBN.



Si le livre ajouté par l'utilisateur est déjà dans sa bibliothèque, un message de confirmation apparaît car les deux cas d'utilisations suivants sont possibles : l'utilisateur scanne le livre pour vérifier s'il le possède pour ne pas le prendre si c'est le cas, ou alors il possède deux fois le même livre, ce qui est également possible.

IV/ Conclusion

Difficultés rencontrées

Nous avons eu quelques problèmes lors du choix du SGBD que nous allions utiliser. Nous avions d'abords dans l'idée d'utiliser FireBase, cependant, après avoir essayé de l'installer sur une de nos machines, l'utilisation de Android Studio était devenue presque impossibles (erreurs inconnues, …) nous avons donc préféré ne pas utiliser ce SGBD et avons donc opté pour SQLite, qui nous semblait plus simple d'utilisation mais aussi plus logique pour notre application, car nous pouvions ainsi stocker les données en interne.

Nous avons aussi quelques difficultés à choisir la méthode de scan du code barre. Plusieurs choix s'offraient à nous (API google, bibliothèque …, etc), chacun avec leurs avantages et leurs inconvénients. Nous voulions initialement utiliser l'API google car nous croyions qu'il existait un lecteur de code bar intégré dans les appareils photos Android, après quelques recherches, nous avons compris que cette dernière requérait un niveau d'API trop haut à notre gout. Nous avons donc préféré utiliser une application tier et la bibliothèque ZXing.

Bilan

Notre application nous a permis d'appréhender la programmation Android et de nous habituer aux concepts d'activités, de vues, … Nous avons pu beaucoup découvrir et les nombreuses recherches que nous avons faites nous ont permis de découvrir qu'il existait de nombreux moyens, parfois très différents, pour arriver à nos fins. Nous avons aussi pu comprendre que le monde d'Android est en constante évolution et qu'il est nécessaire de faire de nombreuses recherches afin de trouver la meilleure manière de faire, et ne pas perdre trop de temps à apprendre une technologie, qui pourra devenir obsolète dans quelques mois. Ce projet nous a ainsi rendu plus autonome et plus efficaces dans nos recherches.

De plus, nous avons pour la première fois fait l'expérience sur un environnement embarqué, ce qui nous a permis d'en découvrir les contraintes, notamment la performance et l'autonomie, et donc la nécessité de se poser des questions de performance nous seulement sur le plan algorithmique comme nous y étions habitués mais aussi sur le plan de l'architecture.

Nous avons réussi à travailler efficacement ensemble, chacun découvrant une technologie et l'expliquant ensuite aux autres personnes du groupe. Nous avons pu ainsi discuter de ce que nous avions trouvé, et ainsi réfléchi ensemble à la manière qui nous semblait la plus simple ou la plus intéressante à découvrir. Pour faciliter le travail de groupe nous avons travaillé avec GIT, nous codions ainsi sur nos branches respectives, et lorsqu'une fonctionnalité marchait comme nous le souhaitions nous pouvions les fusionner, et résoudre les éventuels conflits.

De plus, ce module nous a permit de monter en compétence pour notre PJS4 utilise également une application Android, nous avons donc pu progresser plus rapidement dessus grâce au module Applications Mobiles.

Améliorations Possibles

Nous pourrions facilement transformer notre bibliothèque en médiathèque, en laissant la possibilité à l'utilisateur de ne plus stocker uniquement des livres mais aussi des CD ou des DVDs. Il existe pour cela des API similaire à google Books mais pour les disques et les films. La méthode serait vraisemblablement la même que pour les livres, il nous faudrait simplement ajouter des activités et des vues spéciales pour chaque type de documents stocker, ainsi qu'un menu permettant de choisir quel type de document on veut ajouter ou quel type de document on souhaite consulter comme par exemple une vue avec différents onglets.

Il serait aussi imaginable de donner à l'utilisateur l'option de prendre une photo comme couverture d'un livre.