Nathan BOISARD
Maxime LERAITRE

Réalisation d'une application de gestion de ToDoList
- ToDouDou -

Introduction

L'objectif de la séquence 3 est de reprendre le travail de la séquence 2 et d'ajouter un mode "offline" en mettant en place une persistance de données sur le téléphone à l'aide d'une base SQLite et de la libraire Room.

Améliorations annexes depuis la séquence 2 :

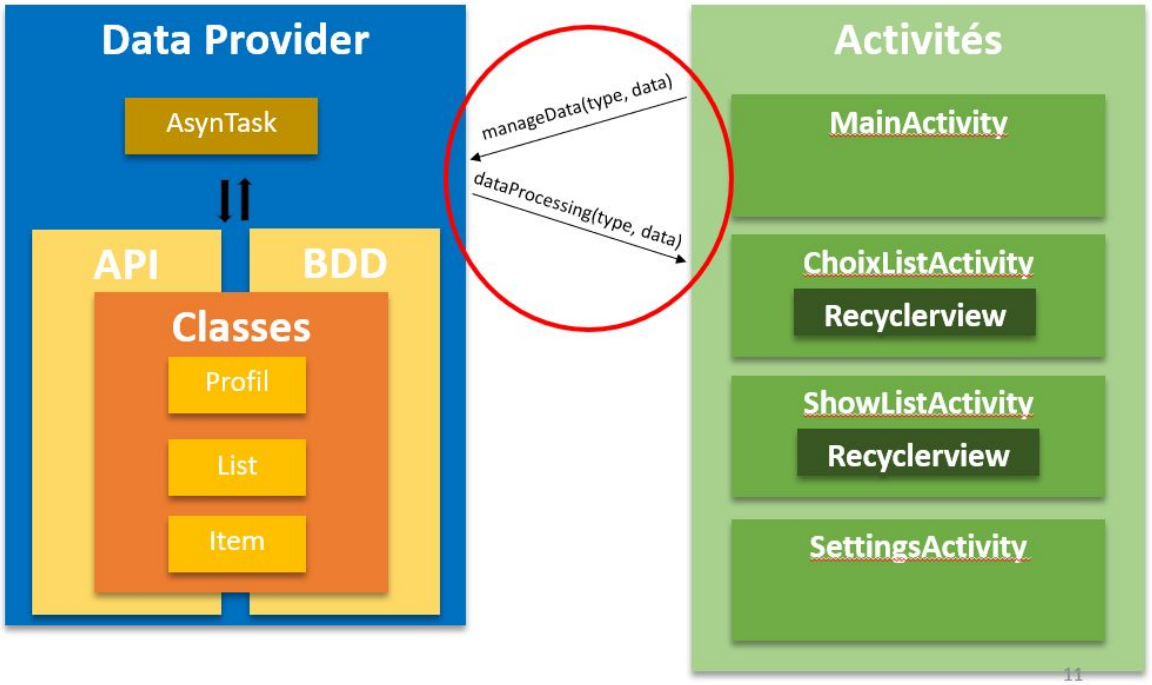
- Mise en place d'une classe 'Converter' qui permet de convertir une liste ou un item au format JSON (à partir de l'objet JSON récupéré depuis l'API) en une liste ou item de notre classe Java. Avant, c'étaient des méthodes propres aux activity affichant les liste et item qui se chargeaient de la conversion.
- Un problème d'UX a été détecté et résolu. Le problème était dû à la connection automatique. En effet, quand l'utilisateur se trouvait sur l'activité affichant les listes, s'il appuyait sur la touche retour de son téléphone, l'application lançait la MainActivity de connexion, qui le redirigeait automatiquement vers l'activité affichant les listes. L'utilisateur voit donc un changement d'écran inutile et un temps de latence dû au au chargement des listes depuis l'API (qui étaient finalement déjà chargées), ce qui n'est pas agréable.
  La solution a été simplement de modifier le comportement par défaut de l'application lorsque l'utilisateur appuie sur la touche retour en faisant quitter l'application (comportement identique à l'appuie de la touche HOME). Pour cela, il suffit de rajouter ce code dans l'activity affichant les listes :7

```
@Override
public void onBackPressed(){
    Intent a = new Intent(Intent.ACTION_MAIN);
    a.addCategory(Intent.CATEGORY_HOME);
    a.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(a);
}
```

Nathan BOISARD
Maxime LERAITRE

Améliorations importantes pour la séquence 3 :

Afin de mieux gérer les données, nous avons créé une classe DataProvider qui va cacher aux activity la complexité d'accès aux données de l'API et la BDD (l'accès aux données est complexe car on doit passer par des async tasks). DataProvider a été conçu pour mieux aborder le problème de la gestion du mode offline, l'idée étant de permettre aux activity de seulement demander des données au DataProvider, sans se soucier si ce sont les données de l'API ou de la BDD : pour les activity c'est transparent.
Techniquement, voici comment on réalise la procédure pour qu'elle soit facile à utiliser :



DataProvider possède une interface et un attribut de type l'interface. Dans chaque activity (sauf settings), on possède une référence vers le singleton DataProvider, et dans le onStart de chaque activity, on va définir (redéfinir) l'interface du DataProvider en appelant "setCustomObjectListener" ainsi que la fonction onDataReady de l'interface. onDataReady est juste un appel à la fonction dataProcessing(type, data).

L'idée est alors qu'à chaque fois qu'une activity a besoin d'une donnée, elle appelle la fonction manageData qui prend en argument une constante (définie dans la classe Constant, cf ci-après) qui correspond à ce que souhaite l'activity, ainsi que des datas optionnelles. manageData va alors s'occuper d'appeler la bonne fonction du dataProvider, fonction qui va faire appel ensuite à des asyns task. Une fois l'asyn task terminé, soit une autre est déclenché (pour pouvoir agir à la fois sur l'API et la BDD) soit on appelle manuellement la fonction onDataReady de l'attribut privée du singleton dataProvider en lui fournissant en paramètre le type de retour (définie dans Constant), ainsi que des données sous le forme (Object... obj). Ainsi, dans l'activity, cela va déclencher un appel à

dataProcessing(type, data) qui va filtrer le type des datas à l'aide du type de retour et réagir en conséquence (par exemple, ajouter un item aux items courants).

```
public class Constant {

    public static final int CONNEXION = 1;
    public static final int CONNEXION_ECHEC = 2;

    public static final int ECHEC_LIAISON_API = 3;

    public static final int GET_LIST = 4;
    public static final int POST_LIST = 5;
    public static final int DELETE_LIST = 6;

    public static final int GET_ITEM = 7;
    public static final int POST_ITEM = 8;
    public static final int DELETE_ITEM = 9;
    public static final int CHECK_ITEM = 10;

    public static final int ACTION_IMPOSSIBLE = 11;
}
```

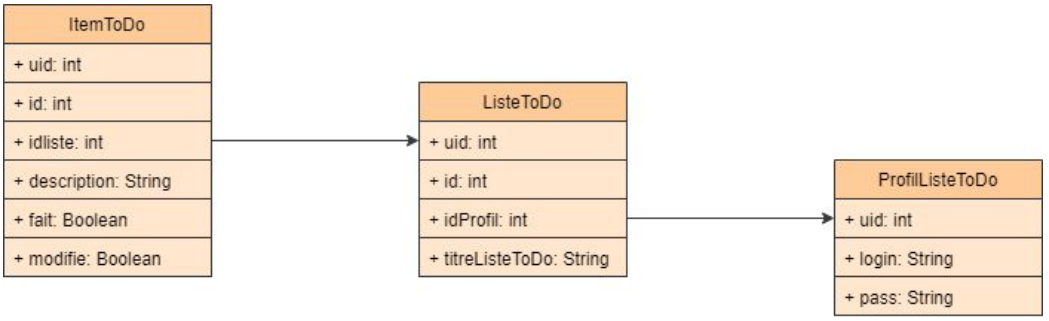
Tout ce processus permet de bien gérer les différents cas, et de migrer l'appel aux async task depuis l'activity vers une classe annexe.

De plus, si dans l'activité des items on appelle manageData(Constant.GET_ITEMS, idList), l'activity ne connait l'état du réseau, c'est le DataProvider qui décide quoi retourner à l'activity : soit les items de la bdd, soit ceux de l'API.
Et par exemple, dans notre application, on ne peut pas ajouter/supprimer des list/items en mode hors ligne, mais si l'utilisateur essaye de le faire, l'activity fait appel à manageData, mais le DataProvider indiquera dans le type de retour de l'interface listenner que Constant.ACTION_IMPOSSIBLE.
Pour plus de détails, nous nous sommes inspirés de ceci :
https://guides.codepath.com/android/Creating-Custom-Listeners

Nathan BOISARD
Maxime LERAITRE

Analyse

Base de donnée :



La base de donnée comporte trois tables, reliées par des correspondances entre clés primaires et clés étrangères. La table ProfilListeToDo contient les login et mots de passe des utilisateurs, la table ListeToDo contient les To-Do listes et la table ItemToDo contient les tâches.
L'application a accès à la base de données via l'ORM Room. Elle sert principalement lorsque l'utilisateur n'est pas connecté au réseau. Dans le cas contraire, on récupère les données en ligne et on met à jour la base de donnée au fur et à mesure des actions de l'utilisateur.
Lorsqu'un utilisateur se connecte avec succès, on récupère l'uid lié à son profil :

```
"SELECT uid FROM profillistetodo WHERE pseudo = :pseudo AND pass = :pass"
```

L'application passe ensuite à l'activité ChoixListeActivity et affiche les To-Do listes de l'utilisateur, préalablement récupérées dans la base de donnée :

```
"SELECT * FROM listetodo WHERE idProfil = :idUser"
```

Lorsque l'utilisateur clique sur une liste, l'application passe à l'activité ShowListActivity et affiche les items récupérés sur la base de donnée :

```
"SELECT * FROM itemtodo WHERE itemtodo.idliste = :id"
```

Sur les activités ChoixListActivity et ShowListActivity, si l'utilisateur possède du réseau, il peut créer, et supprimer des listes ou items. Lorsqu'il ne dispose pas d'une connexion réseau et qu'il modifie un item (en le passant marquant comme fait par exemple), l'attribut `modifie` de l'item est passé à `true` , indiquant ainsi qu'il a été modifié en local et qu'il faudra le synchroniser quand l'utilisateur aura une connexion réseau :

```
"UPDATE itemtodo SET modifie = :modif WHERE itemtodo.uid = :uid"
```

Perspectives

Certains points peuvent encore être améliorés dans l'application :

- ➢ L'UX manque de fluidité et certaines fonctionnalités pourraient être ajoutées afin de proposer une gestion plus claire et personnalisée des listes (cf remarques des comptes-rendus précédents)
- ➢ L'accès aux ToDo listes en mode hors ligne implique de stocker le mot de passe de l'utilisateur sur le téléphone. Celui-ci n'est pas hashé et cela pose des problèmes de sécurité.
- ➢ L'architecture de l'application ne respecte pas les schémas standards et cela peut poser problème lors de la modification de certains éléments car les tâches ne sont pas cloisonnées.
- ➢ Les mots de passe sont stockés en clair dans la base de donnée, pour plus de sécurité, il faudrait les hasher.
- ➢ Il est impossible d'ajouter ou supprimer des listes/items en mode hors ligne, ce qui n'est pas favorable à une bonne expérience utilisateur.

Conclusion

Cette dernière séquence nous a amenés à compléter notre application afin qu'elle gère la sauvegarde de données en ligne et en local. Nous avons maintenant une application fonctionnelle et qui se rapproche de plus en plus de celles que l'on peut trouver sur le Play Store. Il y a certe des améliorations de sécurité et d'expérience utilisateur à faire mais le résultat permet une gestion basique de To-Do listes de manière fiable.
Ce mini-projet "fil rouge" nous a permis de nous familiariser avec le fonctionnement et l'architecture d'android et de cerner les problématiques particulières qu'ils impliquent à l'aide d'un cas concret et intéressant.

Bibliographie

Sitographie :

- Documentation de l'API : https://documenter.getpostman.com/view/375205/S1TYVGTa?version=latest
- Inspiration pour le Data Provider : https://guides.codepath.com/android/Creating-Custom-Listeners
- AsyncTask : https://stackoverflow.com/questions/9671546/asynctask-android-example
- Room : https://developer.android.com/topic/libraries/architecture/room