Le Guillou Eve
Thomas Iona

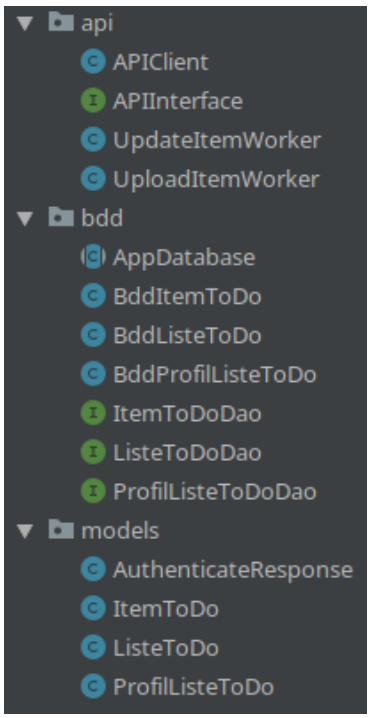
Compte-rendu Application TodoList
Séquence 3

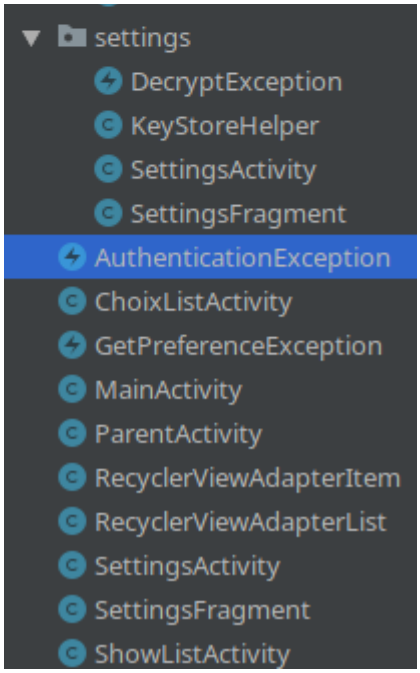
Introduction

Le mini-projet du fil rouge des séance était le développement d'une application android en Java permettant de gérer des todo listes associées à des profils d'utilisateur. Dans cette troisième séquence nous devions gérer la persistance des données sur le téléphone à l'aide d'une base de donnée et de la librairie room. L'objectif était de pouvoir accéder à ses ToDoListes en hors-ligne et pouboir (dé)cocher des items en hors-ligne.

En résumé, on dispose d'une activité MainActivity sur lequel l'utilisateur rentre son pseudo et son mot de passe la première fois qu'il se connecte. Ceux-ci sont stocké dans les préférences de l'application, avec le mot de passe chiffré. Les fois suivantes si l'appareil à internet, l'application lance un appel à l'API pour vérifier si l'utilisateur existe et si le mot de passe correspond. Si oui, l'API renvoie un hash, et l'utilisateur passe à l'activité ChoixListActivity en chargeant les données depuis l'API. Si l'appareil n'a pas internet la même activié est lancée sans authentification mais en chargeant les données depuis une base de donnée locale. L'activité ChoixListActivity affiche les listes correspondant au profil. On peut en ajouter si on dispose d'internet ou aller sur une liste déjà existante. Dans l'activité « ShowListActivity » on affiche les tâches à faire incluses dans la liste et leur état (faites ou non). De même on peut y ajouter des items si on dispose d'internet. Avec ou sans connexion, on pourra toujours cocher ou décocher les items éxistant. Enfin une « SettingsActivity » et un boutton déconnexion sont accessibles à tout instant en utilisant le menu de l'actionBar.

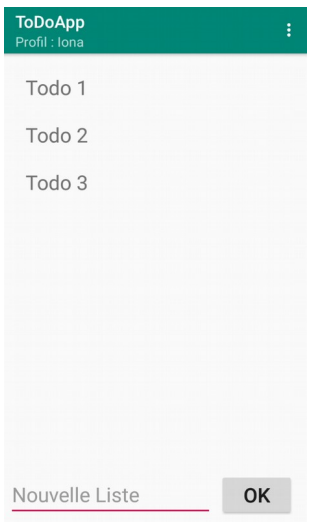
Dans nos sources on dispose donc des classes suivantes qu'on détaillera dans l'analyse en fonction des modifications effectuées :



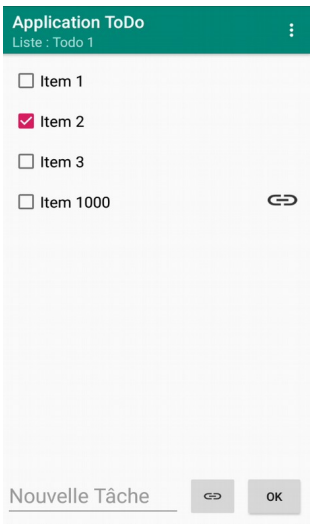


Les différentes activités n'ont pas changé d'apparence :

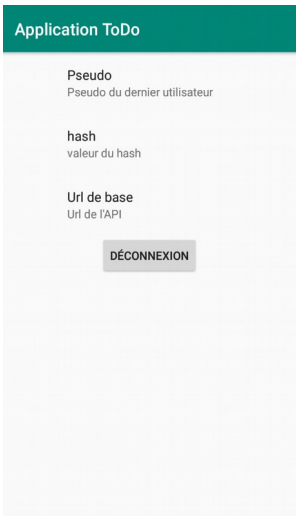
MainActivity



ChoixListActivity



ShowListActivty



SettingsActivity

Le logo est également toujours le même :

Fonctionnalités mise en place dans la séquence 3 :

- Mise en place d'une base donnée pour avoir de la persistance de données en hors-ligne

- Possibilité d'accéder aux listes et items déjà chargés en hors-ligne

- Possibilité de (dé)coher les items en hors-lignes

1. Analyse

i. ParentActivity

Permet de définir des variables et fonctions communes à nos différentes activités, hors settingsActivity.

ii. MainActivity (activité de connexion)

MainAcitvity est l'activité servant à la connexion. La première fois, on y entre son pseudo et mot de passe. Ceux-ci sont sauvegardés dans les préférences. On stockera également le hash à chaque nouvelle connexion. Le mot de passe est stocké une fois hashé grâce à une paire de clé gérée par le AndroidKeyStore. (cf CR séquence2)

Si l'appareil dispose d'internet et si un utilisateur est enregistré dans les préférences, l'application essaiera de l'authentifier automatiquement auprès de 'API, récupérera le hash et passera à l'activité suivante. Si on ne dispose pas d'internet on utilisera les données provenant de la base de donnée pour remplir les activités suivantes.

Si internet n'est pas disponible et que personne n'est connecté, l'activité mainActivity apparaîtra avec un bouton de connexion désactivé et un message invite l'utilisateur à se connecter à internet pour pouvoir s'authentifier.

iii. SettingsActivity

Cette activité n'a pas changé. Elle est est accessible grâce au menu de l'actionBar. On peut accéder au menu Préférences, on peut entrer son pseudo, son hash d'identification et l'URL de base de l'API.

Un bouton log out est disponible pour changer proprement d'utilisateur.

iv. ChoixListActivity

Depuis cette activité, on peut accéder aux Listes Todo déjà créées et en ajouter de nouvelles lorsqu'on dispose d'internet.

Le fonctionnement pour l'utilisateur est similaire au fonctionnement précédent. Les modifications apportée sont la présence d'un connectivityManager et le chargement depuis l'API ou la BDD selon la disponibilité du réseau.

Dans OnResume on met en place le connectivityManager dans la fonction checkConnectivity. On récupère le service du système corespondant, on initialise le booléen isConnected et on attache à se connectivityManager un NetworkCallback définit par nos soins et nommer connectivity callback qui sera appelé à chaque fois que la

capacité d' l'appareil à accéder à internet sera modifié. Ce callback est responsable de la mise à jour de la variable isConnected, de réactiver le bouton pour ajouter une liste et de mettre à jour la liste des listesToDo depuis l'API à chaque fois que l'appareil récupère une connexion internet. En cas de perte de connexion, il désactive le bouton et met à jour isConnected. Également dans OnResume, selon la valeur du boolen isConnected, on chargera les données des listes depuis l'API ou la BDD.

Les méthodes convertToBDD permettent de créer, à partir d'objets issus des classes de notre package models, des objets spécifiques aux classes de la base de données de Room. Les méthodes convertFromBdde permettent de créer, à partir des objets spécifiques aux classes de la base de données de Room, des objets issus des classes de notre package models. Ainsi, l'implémentation de convertToBDD dans ChoixListActivty permet de créer une liste d'objet bddListTodo à partir d'une liste d'objet ListeToDo, et convertFromBdd permet de créer une liste de ListeToDo à partir d'une liste de bddListToDo.

La méthode AsyncTaskGetListFromBdd permet de faire un appel asynchrone à la base de donnée pour importer les données de la base de données. Après cet appel asynchrone, on met à jour le RecyclerView, mais dans le thread principale, car il n'est pas possible d'accéder à l'UI depuis un thread secondaire.

Dans le OnDestroy on désactive le NetworkCallback pour éviter les fuites mémoire.

v. ShowListActivity

Depuis cette activité, on peut accéder aux tâches déjà créées et en ajouter de nouvelles si on dispose d'Internet. Les tâches sont affichées grâce à un RecyclerView dont l'adaptateur est défini dans RecyclerViewAdapterItem. L'état des checkList peut être mis à jour même en hors-ligne.

On retrouve le même système de gestion de la disponibilité du réseau que dans ChoixListActivity avec le ConnectivityManger, checkConnectivity, connectivityCallback et le booléen isConnected. Ici le callback met à jour isConnected, (dés)active le bouton d'ajout, et appelle l'API pour récupérer la dernière version des éléments de la liste. Dans le OnDestroy, on désactive le NetworkCallback pour éviter les fuites mémoire.

Un deuxième mécanisme nous permet de gérer le (dé)cochage des items en hors-ligne : on utilise le WorkManager. Ceci est observable dans la définition du onClickListener appliqué sur les items du RecyclerView. Celui-ci est dans la fonction onItemClick. On peut y voir que si l'on est connecté, on fait appel à l'API pour modifier un Item :

```
appelAPIUpdateItem(itemId, (fait ? 0 : 1));
```

En revanche si on ne dispose pas de connexion internet, on utilise le WorkManager. On va définir une tâche, les conditions pour l'effectuer et la passer au WorkMAnager.

Pour ceci on va tout d'abord définir une classe UpdateItemWorker qui étend la classe Worker. Celle-ci doit disposer des informations sur l'item à mettre à jour :

```
private Integer idItem;
private Integer idList;
private Integer check;
```

On définir le travail à effectuer en overidant la fonction doWork(). On y récupère les données passées lors de la définition de la tâche pour initialiser les attributs idItem, idList, check et on appel l'API pour mettre à jour l'item.

Dans l'activité on définit la tâche qu'on va passer au workManager.

```
Constraints constraints = new Constraints.Builder().setRequiredNetworkType(NetworkType.CONNECTED).build();
```

Notre tâche nécessite une connexion internet. On définit donc cette contrainte :
On définit les données à passer au Worker :

```
Data itemData = new Data.Builder()
        .putInt("idList", id)
        .putInt("idItem", itemId)
        .putInt("check", fait ? 0 : 1)
        .build();
```

On définit ensuite la requête qui devra s'exécuter une fois à l'aide de la classe définie précédemment, de la contrainte sur le réseau et des données. On la passe ensuite au WorkManager qui l'exécutera dès que possible.

```
OneTimeWorkRequest updateItemRequest = new OneTimeWorkRequest.Builder(UpdateItemWorker.class)
        .setConstraints(constraints)
        .setInputData(itemData)
        .addTag("Todo-app")
        .build();
WorkManager.getInstance().enqueue(updateItemRequest);
```

On met alors à jour la base de donnée locale.

Les méthodes convertToBdd et convertFromBdd sont similaires à celles qu'on retrouve dans ChoixListActivty. Ainsi, l'implémentation de convertToBDD dans ShowListActivty permet de créer une liste d'objet bddItemTodo à partir d'un objet ListeToDo, et convertFromBdd permet de créer une ListeToDo à partir d'une liste de bddItemToDo.

La méthode AsyncTaskGetItemFromBdd permet de faire un appel asynchrone à la base de donnée pour importer les données de la base de données. Après cet appel asynchrone, on met à jour le RecyclerView, mais dans le thread principale, car il n'est pas possible d'accéder à l'UI depuis un thread secondaire.

La méthode AsyncTaskUpdateItemBdd permet de faire un appel asynchrone à la base de donnée pour mettre à jour les données de la base de données après création d'un itemToDo en local. Après cet appel asynchrone, on met à jour le RecyclerView.

vi. SettingsActivity

Cette activité n'a pas changé. Elle permet d'afficher les préférences de l'utilisateur, ici limitées à son pseudonyme, à son hash et à l'URL de base de l'API. On peut accéder à cette activité depuis toutes les autres activité de l'application.

Elle utilise un fragment pour les préférences qui hérite de PreferenceFragment-Compat. Cette dernière classe est spécialisée pour la gestion des préférences à l'aide de fragments. Le fragment est ajouté statiquement dans le layout activity_settings.xml.

vii. RecyclerViewAdapterList

Cette classe n'a pas été modifiée par rapport à la séquence 1.

vii.bis RecyclerViewAdapterItem

Cette classe n'a pas été modifiée par rapport à la séquence 2.

viii. Internationalisation

Ces éléments n'ont pas été modifiés par rapport à la séquence 1.

iv. models package

Ce package n'a pas changé depuis la séquence 2. On rassemble dans ce package les models nous servant à représenter nos objets et servant à Retrofit à parser les réponses dans l'API. On y retrouve :

- AuthenticateResponse

Cette classe est utilisée lors de l'appel à l'API permettant l'authentification, elle possède comme attribut les éléments de la réponse de l'API, ce qui permet à Retrofit de parser la réponse sous forme d'un objet utilisable.

- ItemToDo

Classe servant à représenter les items. On notera l'ajout d'un attribut url et la conversion du booléen fait en un entier par rapport à la version précédente pour se conformer aux données renvoyées par l'API. IsFait a donc été repensé pour renvoyer un booléen à partir de l'entier récupérer par Retrofit.

- ListeToDo

Classe représentant les listes. On a fait attention à ce que getLesItems ne renvoie pas de null même si Retrofit n'a pas initialisé lesItems.

- ProfilToDo

Classe représentant un profil utilisateur avec une liste de ListToDo.
Classe servant également à parser les réponses renvoyant la liste de ListToDo d'un utilisateur.

x. API Client

Cette classe permet d'instancier le service qui va contacter l'API utilisant la bibliothèque Retrofit. Elle implémente la méthode permettant de créer le service.

xi. API Interface

Cette interface permet de déclarer des méthodes d'appel à l'API, selon une écriture propre à Retrofit. On y déclare les routes de l'API et leurs paramètres.

xi. bdd package

On rassemble dans ce package les models et classes nous servant à représenter nos objets en base de donnée et servant à Room à parser les réponses de celle-ci. On y retrouve :

- AppDatabase
Cette classe étend Roomdatabase, et permet d'obtenir accès aux différentes interface DAO que nous utiliserons pour discuter avec la base de données.

- ItemToDoDao
Interface DAO contenant les méthodes permettant d'accéder à la base de données pour les objets itemToDo.

- ListToDoDao
Interface DAO contenant les méthodes permettant d'accéder à la base de données pour les objets listeToDo.

- ProfilListeToDoDao
Interface DAO contenant les méthodes permettant d'accéder à la base de données pour les objets profilToDo.

- BddItemToDo
Classe représentant les items dans la base de données.

- BddListTo
Classe représentant les listes dans la base de données.

- BddProfilToDo
Classe représentant un profil utilisateur avec une liste de ListToDo dans la base de données.

Conclusion

Cet exercice nous a donné l'occasion de manipuler Room et les bases de données, le WorkManager et les workers, et le ConnectivityManager et les callbacks. Nous avons ainsi pu prendre le temps de comprendre, d'approfondir et de manipuler ces notions ainsi que celle de thread. Il est satisfaisant d'avoir réussi à mener ce fil rouge de bout en bout et d'avoir une application fonctionnelle à la fin de celui-ci.

Perspectives

Il subsiste encore de nombreuses perspectives d'amélioration. Comme on l'avait mentionné dans les compte-rendus précédents, on peut ajouter une classe intermédiaire pour factoriser les méthodes onCreate des activités ShowListActivty et ChoixListActivity, ainsi réduire encore le code, sans surcharger MainActivity. On peut aussi penser à pouvoir masquer et supprimer les tâches déjà faites. L'historique des pseudo n'a pas été mis en place. On avait également mentionné l'amélioration de la connexion automatique et la vérification qu'une application permettant de répondre à l'intent implicit permettant d'ouvrir une page web existe à l'aide de resolveActivity(). On pourrait éventuellement améliorer l'interface utilisateur en utilisant un bouton d'action flottant (cf bibliographie) plutôt que l'interface actuelle. On pourrait aussi pouvoir mettre en place un système d'inscription, pour créer de nouveaux utilisateurs car une nouvelle personne ne peut pas s'inscrire.

Les nouvelles améliorations possible qui ont émergé avec cette version de l'application sont l'ajout d'item et de listes en mode hors ligne ce qui nécessiterai de faire une différence en bdd entre ceux déjà en ligne et les nouveaux. Lors de notre réflexion sur la faisabilité de cette amélioration, nous avions remarqué notamment le problème de l'id des nouveaux éléments. Comme ceux-ci servent d'identifiant et sont normalement fournis par l'API il est délicat de les intégrer dans la même base de données que ceux déjà en ligne si les identifiants ne sont pas définitifs. En effet ceux-ci servent de primary key. Il faudrait donc mettre en place une deuxième bdd et ajouter tout au long du code cette nouvelle source d'informations. Les appels à l'API pour mettre en ligne les nouveaux items des nouvelles listes seraient les plus problématiques car il nécessite d'envoyer la nouvelle liste, récupérer son id et l'utiliser pour ajouter les nouveaux items. Au vu du temps disponible cette semaine, nous avons préférez nous concentrer sur le mini-projet.

Bibliographie

Connectivity Manager
https://developer.android.com/reference/android/net/ConnectivityManager

Room BDD :
https://developer.android.com/topic/libraries/architecture/room

WorkManager :
https://developer.android.com/topic/libraries/architecture/workmanager