Compte rendu PMR séquence 3

Introduction :

L'application ToDoListe permet à plusieurs utilisateurs de stocker des listes personnelles de tâches à effectuer. Dans le TP précédent, nous faisions appel à une base de données distantes via une API. Si le réseau vient à manquer, l'application devient inutilisable. Nous allons donc implémenter un cache qui permet de sauvegarder temporairement les données en local. J'ai fait le choix de faire appel à la dépendance Room qui fait partie d'Android Jetpack. Cette dépendance permet de sauvegarder des données dans le cache sous forme de base de données (BdD).

Analyse :

Chaque partie est à lire en complément avec le code commenté.

Détection du réseau :

Pour savoir si l'on travaille sur le cache, on a besoin de vérifier constamment si le réseau est disponible. Si cela n'est plus le cas, on travaille sur le cache.

J'ai choisi d'implémenter un Broadcast Receiver dans chaque Activité pour cela.

Gestion des Threads :

Les appels à la BdD demandant un certain temps d'exécution, il est recommandé d'éviter de les faire dans le MainThread. Pour cela, j'ai choisi d'utiliser un executor pour éviter de faire les appels à la BdD sur le MainThread. Les appels à l'API se font déjà sur un autre Thread par la fonction .enqueue().

Gestion de la persistance de données :

Puisque lors du dernier TP, j'avais choisi de ne pas créer de classe spécifique pour l'API, j'ai été forcé d'en recréer pour la BdD. En effet il n'est pas possible de placer des List<> dans la BdD. Dans l'idéal, il aurait fallu créer 3 classes distinctes : une pour le modèle de l'application, une pour l'API et une pour la BdD.

C'est ainsi que j'ai créée les classes ListeToDoDb (table "listes") et ItemToDoDb (table "items"), simples restranscriptions de ListeToDo ItemToDo.

ListeToDoDAO() :

Cette interface permet d'effectuer les requêtes vers la BdD qui concerne la table "listes". Deux requêtes sont écrites : pour récupérer toutes les listes, et pour sauvegarder les listes dans la base de donnée.

ItemToDoDAB() :

Cette interface permet d'effectuer les requêtes vers la BdD qui concerne la table "items".

RoomListeToDoDb

La classe de la base de données. Si cela n'a pas été déjà effectuée, la base de donnée est créée. On le détecte au moyen d'un singleton. De plus, elle instancie les interfaces citées précédemment.

Au sein des activités :

SyncFromAPI() : permet de récupérer les données de l'API.

SyncFromBDD() : permet de récupérer les données de la BdD.

SyncToBDD() : permet de sauvegarder les données dans le cache. À chaque fois que SyncFromApi() est appelée, cette méthode est appelée.

Conclusion :

L'application n'est pas complètement fonctionnelle. En effet, il y a un bug au niveau de l'authentification et l'utilisateur doit redémarrer l'application pour que les requêtes à l'API fonctionnent de nouveau. De plus, la fonction de cochage et de modification de la BdD ne fonctionne pas complètement. Lors du retour du réseau, on récupère d'abord les données de l'API avant de sauvegarder les modifications du cache. Je n'ai pas réussi à identifier le code.

Je n'ai pas eu le temps de modifier voire d'améliorer mon code entre chaque séance. Je n'ai donc pas solutionné de la bonne manière ou utilisé les bonnes pratiques. Cela est dommage car cela m'a beaucoup ralenti. En 3 TP, le code de mon application est devenu très peu lisible, et ressemble de plus en plus à un sac de nœud. Je n'ai ainsi pas réussi à terminer dans les temps la séquence 3. Cependant, ce TP m'aura montré l'importance des bonnes pratiques.

Perspectives :

Refactor le code dans le but de le simplifier/segmenter.

Résoudre le bug lors de l'authentification.

Finir l'implémentation du cache avec le cochage des Items (synchronisation mal effectuée).