PMR - Compte rendu Séquence 2

Johan Manuel
Thibault Ayanides

Remarque : Pour tester l'application rendue, il suffit d'ouvrir le projet dans Android Studio et de lancer l'application. Par défaut, l'application se connecte à l'API située à `tomnab.fr`, ce qui peut être modifié dans les préférences.

Objectifs

Dans cette séquence, nous devions modifier notre application de todo list pour qu'elle fasse usage de l'API REST mise à notre disposition. Les listes ne seront plus stockées localement mais obtenues depuis cette API, et les listes crées seront également envoyées à travers celle-ci. Pour cela, nous devrons utiliser des coroutines, permettant d'exécuter du code de manière asynchrone.

Développement

Nous nous sommes basés sur le modèle de client REST donné par M. Boukadir en cours. Celui-ci est centré autour d'un singleton, *DataProvider*, qui sert de source de donnée et d'une manière générale, de moyen de communication avec l'API.

Cette communication se fait à travers la bibliothèque *Retrofit*. Elle permet de spécifier la forme de notre API (noms des endpoints, paramètres, etc...) directement par la définition d'une interface kotlin annotée par les verbes HTTP, *@GET*, *@POST* etc... Cette interface est un *Service* Retrofit et est ici nommée *TodoApiService*. Le singleton *DataProvider* utilise ensuite un constructeur de Retrofit pour obtenir une implémentation de cette interface, utilisable pour communiquer avec l'API, et il l'expose au reste du code par des fonctions plus ergonomiques que les réponses d'API. On a permis le changement d'URL de l'API dans les paramètres, en reconstruisant le service Retrofit avec la nouvelle URL de base quand celle-ci est modifiée.

Ces réponses d'API guident d'ailleurs la forme des données dans cette version de l'appli todo list. On retrouve dans *TodoApiService.kt* plusieurs *data classes* servant à désérialiser les réponses d'API (*ListsResponse, ItemsResponse*) qui elles contiennent les *data classes* utilisées dans le reste du code (*ListProperties, ItemProperties*, anciennement *ListToDo* et *ItemToDo*). Ceci est spécifique à l'API utilisée, il est possible qu'avec une autre API on ait pu utiliser directement les réponses.

La communication avec l'API se fait de manière asynchrone, ce qui est géré par les coroutines de kotlin. Toutes nos méthodes dans *TodoApiService* et *DataProvider* sont déclarées *suspend* et renvoient directement des types comme *List<ItemProperties>* Les classes d'affichage de ces listes, *ChoixListActivity* et *ShowListActivity* notamment, appellent ensuite ces méthodes dans des coroutines. Ces *Activities* ont leur *CoroutineScope* et appellent les méthodes de *DataProvider* pour mettre à jour leur interface, par exemple comme ici :

```
private fun refreshLists() {
    activityScope.launch {
        val lists = withContext(Dispatchers.IO) {
            DataProvider.getLists(getHash())
        }

        adapter.updateData(lists)
    }
```

```
}
```

On voit ici un appel à une fonction *getHash* : celle-ci récupère le hash identifiant l'utilisateur courant dans les préférences de l'application. Il est récupéré depuis l'API une seule et unique fois au moment du login. On notera que l'on a essayé de supporter la création de nouveaux utilisateurs par l'endpoint `POST users?pseudo=a&pass=b` comme indiqué dans la documentation, mais celui renvoyait toujours `success=false`.

On a pensé à vérifier l'accès à internet au lancement de l'activité de connexion, comme demandé dans le sujet. On peut noter cependant que cette vérification n'est pas parfaite : avec les données mobiles activées et sans réception, elle indique que l'on a accès à internet.

En plus des fonctionnalité strictement requises, nous avons également implémenté la possibilité d'ajouter une nouvelle liste pour l'utilisateur connecté.

Pour ce qui est de l'interface, on a simplement réutilisé le code du TP 1 qui utilisait les *RecyclerViews* présentées précédemment par M. Boukadir.

Conclusion

Ce sujet nous a permis de bien comprendre l'usage des coroutines en kotlin, les avantages qu'elles présentent mais aussi les restrictions qu'elles imposent. Par l'usage de la bibliothèque Retrofit, nous avons également pu voir dans un cas pratique ce qu'était une API REST et comment en faire usage.

Perspectives d'amélioration

Dans son état actuel, cette application est très sensible à la connexion à internet : suite à une perte de connexion, l'utilisateur perd tout accès à ses listes, et l'application elle même n'a pas la garantie de fonctionner correctement. Il faudrait idéalement garder une copie des listes localement, qui pourrait toujours être consultée et éditée.

D'un point de vue utilisation, il serait très intéressant de pouvoir modifier les items de liste et de les réordonner, et de même pour les listes de l'utilisateur. Retenir les identifiants de l'utilisateur précédent serait également très pratique, tout comme permettre de créer un nouveau compte, ce que nous n'avons pas réussi à faire fonctionner.