Rapport n°2

I. Introduction

Le but de ce TEA 2 était de connecter l'application développée lors du TEA 1 à une API REST. Ainsi, l'application peut être utilisée depuis plusieurs appareils et avoir accès aux mêmes informations

II. Analyse

Tout d'abord, il convient d'analyser et de comprendre le fonctionnement de l'API. Pour cela, on peut utiliser la documentation Postman proposée, ou encore aller soit-même sur le logiciel Postman pour voir les structures des réponses aux différentes requêtes.

Après avoir compris le fonctionnement de l'API, on peut commencer à définir les objets de réponse dans lesquelles on cast les réponses obtenues en JSON :

- HashResponse : sert à récupérer le hash token de l'utilisateur lors de la connexion. Ce token est essentiel pour toutes les requêtes.
- ListsResponse : sert à récupérer les listes lorsque l'utilisateur accède à la ChoixListActivity
- ItemsResponse : sert à récupérer les items d'une liste donnée lorsque l'utilisateur accède à la ShowListActivity

A chacune de ces data classes, on associe une interface qui va définir les requêtes pour chaque type d'objet :

- UserService
    - Permet de s'authentifier auprès de l'API et de récupérer le hash token
    - Implémente une fonction permettant de créer un nouveau compte. Cependant, cette fonction n'est pas implémentée dans l'application car la création de compte requiert d'être identifié.
- ListsService
    - Permet de récupérer les listes de l'utilisateur connecté
    - Permet de récupérer le nom d'une liste en fonction de son id (pour l'afficher dans la ShowListActivity)
    - Permet d'ajouter une nouvelle liste pour l'utilisateur connecté
    - Permet de supprimer une liste de l'utilisateur connecté
- ItemsService
    - Permet de récupérer les items d'une liste donnée
    - Permet d'ajouter un item à une liste donnée
    - Permet de mettre à jour l'état "checked" d'un item donné
    - Permet de changer le nom d'un item donné
    - Permet de supprimer un item donné d'une liste

Les requêtes désormais définies, l'application doit être capable de les exécuter sur les différentes activités.

On va mettre toutes les fonctions qui permettent de recueillir les données dans une classe DataProvider. Pour envoyer les requêtes à l'API, on utilise la librairie Retrofit qui permet directement de manipuler des API REST.
Cependant, comme vu en cours, on ne peut pas exécuter de requêtes HTTP depuis le main thread (ou UI thread) par risque de bloquer l'application. Ainsi, on utilise la notion de coroutines proposée par Kotlin. Les fonctions définies sont alors des "suspend fun" et s'exécutent dans le Dispatchers.Default.
Chaque méthode du DataProvider renvoie l'échec ou le succès de la requête, et le cas échéant le contenu de la réponse.

Il reste donc seulement à adapter les différentes activités pour afficher toutes les informations.
On remplace toutes les anciennes méthodes de récupération de données par les nouvelles. Pour pouvoir exécuter les fonctions mises en place, il faut sortir du main thread, pour cela on utilise lifecycleScope.launch pour y mettre les appels réseau.
On adapte aussi la MainActivity pour inclure un champ "Mot de passe" et bloquer le bouton de connexion si l'appareil n'est pas connecté à internet. Pour vérifier la connexion, on fait une requête vers une page de test proposée par Google qui renvoie une réponse HTTP 204 (sans contenu) (https://clients3.google.com/generate_204) qui est assez rapide.

Pour terminer, il faut autoriser l'application à accéder à internet : il suffit de définir l'autorisation dans le AndroidManifest.

III. Conclusion

L'application permet désormais de se connecter à une API distante afin d'avoir une notion de compte utilisateur et d'application synchronisée sur plusieurs appareils.

IV. Perspectives

Parmi les requêtes implémentées dans l'application, certaines ne sont pas utilisées comme la suppression d'item dans une liste ou encore la création de compte. Ce sont des fonctions intéressantes à implémenter par la suite.
Enfin, si l'utilisateur n'a pas d'accès à internet, l'application est inutilisable. Il serait donc intéressant d'implémenter une fonction hors ligne (objet du TEA 3).