Rapport TP2

Jean Bouzereau - Jean Le Goff

Introduction

Ce deuxième TP propose une amélioration du TP précédent. Il s'agit d'utiliser une API REST depuis l'application. En gardant le contexte de ToDo Lists, nous devons utiliser les services proposés par l'API et la base de données qui lui est liée pour améliorer notre application de base.

Analyse

La consommation d'une API présente plusieurs problématiques. Tout d'abord la gestion des threads. En effet les appels à l'API prennent trop de temps pour être exécutés dans le Main thread. Le Main thread étant responsable de l'interface utilisateur, l'utiliser diminuerait le nombre de rafraîchissements de l'écran et rendrait l'expérience utilisateur bien moins bonne. C'est pourquoi nous avons utilisé des coroutines qui permettent de gérer efficacement les threads en évitant les callbacks.

Une fois la question de la gestion des threads mise de côté, nous avons pu commencer à nous occuper de la manière de consommer l'API. La manière assez naturelle proposée par M. Boukadir est d'utiliser un DataProvider. Dans cette classe DataProvider, différentes méthodes permettent de générer chacune des requète gérées par l'API et documentées ici.

Pour cela nous avons dû commencer par comprendre les différentes requêtes c'est-à-dire comprendre leur but ainsi que la manière dont il faut les utiliser. Cela fut relativement simple à l'aide de la documentation et de quelques tests avec Postman. La gestion des headers fut un des points délicats.

L'API retourne une réponse sous forme de Json. Pour rendre cette réponse exploitable, nous avons utilisé la librairie Gson. Cette librairie permet de convertir des objets java en Json et réciproquement. Elle permet de réduire drastiquement la quantité de code. Il suffit d'établir les correspondances entre les propriétés des objets et les différentes parties du Json, et ensuite, tout est automatique.

Finalement nous nous sommes rendu compte de la facilité que représentait Retrofit pour la consommation des API. Nous avons donc décidé de reprendre notre travail afin d'utiliser Retrofit.

Pour implémenter l'utilisation de Retrofit nous avons tout d'abord dû créer des data class modélisant les objets manipulés, les trois data class de base (Items, Lists, Users) sont le modèle des tables de la base de donnée, il a ensuite fallu aussi créer des classes pour modéliser les objets renvoyés par l'API en réponse à une requête (AuthenticateResponse,

ItemResponse…) sans ces classes il n'est pas possible de récupérer directement par une simple deserialisation l'objet renvoyé par l'API, ces classes permettent aussi de savoir si la requête a été fructueuse grâce au paramètre success.

Ensuite, nous avons créé la classe ToDoApiService pour y écrire les fonctions requêtes avec les mots clés Retrofit (@GET, @POST…) et @Path ou encore @Query pour gérer les paramètres de ces fonctions et l'intégration des hashcode.

Enfin un objet DataProvider a été créé pour faire appel à ces requêtes et mettre les données à disposition des activités.

Toutes les fonction de ToDoApiService et DataProvider sont initialisées avec le mot clé suspend. Elles sont ensuite appelée dans les activités grâce à des Coroutines via des ActivityScope.

Description du parcours de l'utilisateur

L'utilisateur arrive en premier sur Main Activity. Il doit rentrer son pseudo et son mot de passe pour acceder à l'activite lui presentant ses listes. Si il s'est déjà connecté précédemment, il rentre dans l'application sans problème, si c'est sa première connexion, un nouvel utilisateur est créé (en utilisant un hash d'administrateur récupéré via un utilisateur rentré dans le code source.), il arrive alors sur l'activité lui permettant de créer des listes et de les afficher. Dans ces deux cas, le hashcode de l'utilisateur (nouveau ou non) est stocké dans les SharedPreferences pour pouvoir être ensuite facilement récupéré dans les activités suivantes. Si l'utilisateur se trompe de mot de passe ou si il veut s'inscrire avec un pseudo déjà utilisé, un toast s'affiche pour lui demander de réessayer en changeant de pseudo ou en utilisant un mot de passe correct. La gestion de cette possibilité nous a posé problème car dans le cas où le mot de passe est incorrect lors de l'identification ou dans le cas ou le pseudo est déjà utilisé, l'API ne renvoit pas un objet JSON comme dans le cas ou tout va bien mais une erreur HTTP. Nous avons mis du temps à identifier et nous avons finalement géré ces exceptions avec des try{} catch(){}.

Finalement l'utilisateur arrive sur l'activité ChoixListActivity ou on récupère ses listes grâce à son hash stocké dans les SharedPreferences et on les affiche grâce à un RecyclerView. L'utilisateur a la possibilité de créer un nouvelle liste via le champ au bas de la page, lorsqu'il clique sur le bouton Ok, une requête publie cette nouvelle liste dans l'API et la page est rechargée via un Intent pour que cette liste soit tout de suite affichée. Un autre solution aurait été de relancer l'Adapter du RecyclerView via la méthode .showdata pour éviter d'avoir à recharger toute l'activité. Enfin, en cliquant sur une des listes affichées, l'utilisateur accède à l'activité lui permettant de visualiser les items de cette liste. Lors du clic sur une liste dans cette activité, l'ID de la liste choisie est stockée dans les SharedPreferences pour etre récupérée dans l'activité suivante.

L'activité affichant les items est ShowListActivity. On commence par y récupérer l'ID de la liste à afficher stocké dans les SharedPreferences, et via une requête on récupère les item de cette liste, on les affiche grâce à un RecyclerView. Pour chaque item on affiche son label ainsi qu'un checkbox indiquant son état. L'implementation du OnclickListener pour les checkbox nous a demandé un peu de temps, nous avons finalement utilisé la même technique que pour le OnItemListener mis en place dans le tp1 pour gérer les click sur les

éléments du RecyclerView. Enfin, l'utilisateur peut créer un nouvel item grâce au champ en bas de l'activité, Lors du clic sur le bouton Ok, un nouvel item est inséré dans la liste de l'API via une requête, la page est aussi rechargé via un intent pour afficher directement cet item sur l'activité.

L'utilisation d'intent ramenant à la meme activité pour recharger les élèments pose un problème quand à l'utilisation du bouton retour du Smartphone. Pour palier à ce problème nous avons fait en sorte que ce bouton retour ramène tout le temps à la MainActivity lorsque l'utilisateur est sur la ChoixListActivity et qu'il ramène à ChoixListActivity lorsque l'utilisateur est sur ShowListActivity.

Bibliographie

- Documentation Android : https://developer.android.com/docs (pour les SharedPreferences…)
- Cours Open Classroom pour certaines precisions : https://openclassrooms.com/fr/courses/3499366-developpez-une-application-pour-android
- Baeldung https://www.baeldung.com/java-adapter-pattern (pour les Adapter Patterns)
- CodePath https://guides.codepath.com/android/using-the-recyclerview (pour les RecyclerView)
- Documentation de l'API utilisée https://documenter.getpostman.com/view/375205/S1TYVGTa?version=latest#a387d2aa-35cf-4739-bee1-157a25a7e09d
- StackOverflow, GitHub