Alban RAHIER

Compte-rendu application Android ToDoListe

Introduction :

L'application ToDoListe permet à plusieurs utilisateurs de stocker des listes personnelles de tâches à effectuer. Le stockage se fait cette fois-ci de manière distante sur une base de donnée. Pour cela, on doit faire des requêtes HTTP vers une API. On reprend l'architecture qui a été pensée pour la séquence 1. J'ai fait le choix d'utiliser la dépendance Retrofit pour faire les requêtes.

Analyse :

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

Fonctionnement des requêtes API :

Pour ce TP, il aurait été possible de sauvegarder une configuration temporaire en local lors de modifications sur le profil utilisateur (ajout d'éléments, cochage ou décochage d'objet). Ces modifications seraient envoyées toutes d'un coup lors d'un changement d'activité. Cette méthode présente l'intérêt de ne pas recréer à chaque fois une instance de servicefactory.

J'ai choisi plutôt de faire une requête à chaque modification. Cette méthode permet d'éviter de surcharger le téléphone de plusieurs requêtes d'un coup. En effet, chaque requête prend du temps et cela aurait bloqué le téléphone pendant au moins une seconde lors du changement d'activité. De plus, cette méthode est plus simple à implémenter.

Alban RAHIER

Utilisation de la dépendance Retrofit :

J'ai choisi d'utiliser la dépendance Retrofit car elle me paraissait être la plus simple d'utilisation.

Dans un premier temps, on créée la ListeToDoServiceFactory qui permettra de créer une instance Retrofit.

```
public class ListeToDoServiceFactory {

    // L'Url de base.
    public static String Url = "http://tomnab.fr/todo-api/";

    public static Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(Url)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

    // Cette méthode est appelée lors d'un changement
    // d'Url de base.
    public static void changeUrl(String url){
        Url = url;
        // On a besoin de recréer l'instance retrofit.
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(url)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
    }

    public static <T> T createService(Class<T> type) { return retrofit.create(type); }
}
```

Ensuite, il faut créer l'interface qui fera les requêtes HTTP.

```
public interface ApiInterface {

    // Demander un nouveau hash
    @POST("authenticate")
    public Call<Hash>  getHash(@Header("hash") String hash,
                               @Query("user") String user, @Query("password") String password);

    // Lister les listes
    @GET("lists")
    public Call<ProfilListeToDo> getLists(@Header("hash") String hash);

    // Ajouter une liste
    @POST("lists")
    public Call<ProfilListeToDo> addLists(@Header("hash") String hash, @Query("label") String label);

    // Lister les items
    @GET("lists/{id}/items")
    public Call<ListeToDo> getItems(@Header("hash") String hash, @Path("id") Integer id);

    // Cocher/décocher un item
    @PUT("lists/{idListe}/items/{idItem}")
    public Call<ListeToDo> cocherItems(@Header("hash") String hash,
                                       @Path("idListe") int idListe, @Path("idItem") int idItem,
                                       @Query("check") String check);

    // Ajouter un item
    @POST("lists/{idListe}/items")
    public Call<ListeToDo> addItem(@Header("hash") String hash,
                                     @Path("idListe") int idListe,
                                     @Query("label") String label, @Query("url") String url);

}
```

Enfin, on fait appel à l'interface ApiInterface à chaque changement dans les données. La capture ci-après montre un exemple générique de cet appel.

```
// On récupère le hash dans les préférences.
final SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
String hash = settings.getString( key: "hash", defValue: "44692ee5175c131da83acad6f80edb12");

// On crée le service en utilisant la ListeToDoServiceFactory,
// à partir de l'ApiInterface.
ApiInterface Interface = ListeToDoServiceFactory.createService(ApiInterface.class);
// On fait la requête permettant de récupérer
// la liste des listes de l'utilisateur connecté.
call = Interface.getLists(hash);
// On rajoute l'appel à la liste des tâches
call.enqueue(new Callback<ProfilListeToDo>() {

    // Si l'on réussit à envoyer la requête
    @Override
    public void onResponse(Call<ProfilListeToDo> call, Response<ProfilListeToDo> response) {
        // Dans le cas où la réponse indique un succès :
        if(response.isSuccessful()){

        }
        // Dans le cas où la réponse indique un échec :
        else {

        }
    }

    // Si l'on ne réussit pas à envoyer la requête
    @Override public void onFailure(Call<ProfilListeToDo> call, Throwable t) {

    }
});
```

Classes Objets :

Chaque classe doit implémenter les mêmes informations que la base de données. Ainsi, j'ai changé la plupart de chaque classe.

```
public class ItemToDo implements Serializable {

    @SerializedName("label")
    private String description;

    @SerializedName("checked")
    private int fait;

    @SerializedName("id")
    private int id;
```

```
public class ListeToDo implements Serializable {
    @SerializedName("label")
    private String titreListeToDo;

    @SerializedName("items")
    private List<ItemToDo> lesItems;

    @SerializedName("id")
    private String mId;
```

```
public class ProfilListeToDo implements Serializable {

    @SerializedName("lists")
    private List<ListeToDo> mesListeToDo;

    @SerializedName("pseudo")
    private String login;
```

MainActivity :

L'activité de démarrage propose de se connecter avec son pseudo et son mot de passe. Lors de l'appui sur «OK», on récupère l'utilisateur et le mot de passe rentrés par l'utilisateur. On envoie une requête de type PUT pour demander un nouveau hash. (Remarque : cette requête est faite avec un hash de base. Ce hash est stocké en clair ce qui ne se fait pas) Ce hash sera stocké dans les

préférences pour être réutilisé par la suite. Si l'utilisateur n'existe pas ou que le mot de passe ne correspond pas, un Toast s'affiche pour indiquer l'erreur.

Le bouton «OK» est désactivé si le réseau n'est pas accessible. Comme j'ai implémenté cette fonctionnalité dans le onStart() de l'activité, il faut sortir de l'activité pour mettre à jour l'état du bouton. L'écoute ne se fait pas en continu. Pour cela, il aurait fallut implémenter un Listener sur l'état du réseau.

PreferenceActivity :

J'ai gardé l'activité de la séquence 1. Je me suis contenté de changer les arguments de l'EditText à savoir que l'on stocke une URL et non un pseudo.

ChoixListActivity et RecyclerViewAdapter1 :

Cette activité permet de lister et d'ajouter des listes correspondant à l'utilisateur connecté.

J'ai conservé la méthode du bouton flottant de la séquence 1 pour ajouter une nouvelle liste.

J'ai conservé le code de la séquence 1. Excepté que désormais, on liste les listes à partir de l'API. On fait une requête de type GET dans le onCreate() de l'activité (méthode sync()). Lors de l'ajout d'une liste, on fait une requête de type POST (méthode add()).

De plus, la manière dont j'ai développé le RecyclerViewAdapter1 impose de renvoyer une List<ItemToDo>. Or avant de faire la requête, on ne connaît pas cette liste. Donc je dois envoyer une liste vide lors de la création du RecyclerViewAdapter1. Je pense qu'il y avait une autre manière de faire, car cela ne me semble pas être une bonne pratique.

Lors du clique sur une liste, on passe à l'activité ShowListActivity, en envoyant l'id de la liste qui a été cliquée.

ShowListActivity et RecyclerViewAdapter2 :

Cette activité permet de lister et d'ajouter des objets correspondant à l'utilisateur connecté, à la liste demandée.

J'ai conservé la méthode du bouton flottant de la séquence 1 pour ajouter un nouvel objet.

J'ai conservé le code de la séquence 1. Le traitement se fait toujours dans le onBindViewHolder du RecyclerViewAdapter2 ce qui n'est pas une bonne pratique.

Conclusion :

Ce deuxième TP m'a permis de renforcer mes compétences en Programmation Orientée Objet. J'ai perdu beaucoup moins de temps sur la compréhension du fonctionnement de la programmation sur Java. De plus, j'ai beaucoup mieux utilisé les outils de debuggage d'Android Studio. L'utilisation de raccourcis comme «logt» ou «logd» y est pour beaucoup. Enfin, j'ai appris à lire les requêtes envoyé par le portable au sein d'Android Studio, ce qui permet d'accélérer le débuggage.

Alban RAHIER

Perspectives :

Ajout de la possibilité de création d’un utilisateur dans la MainActivity.

Changer le mode de raffraîchissement lors d’ajout d’éléments.

Fusionner les 3 RecyclerViewAdapter en un seul.

Ajouter une option de suppression des listes/objet lors d’un appui long.

Bibliographie :

Méthode verifReseau fournie par M. BOURDEAUD’HUY.

Tutoriel d’implémentation de Retrofit : https://www.androidhive.info/2016/05/android-working-with-retrofit-http-library/

Code de M. BOUKADIR : https://github.com/pmr2019/sequence2/blob/livecoding-retrofit-end/app/