Compte rendu Séquence 2:
Application ToDoList

Introduction

L'objectif de cette séquence 2 est de modifier l'application de ToDoList, afin qu'elle récupère les données (comme les utilisateurs, leurs listes, leurs items, etc) via une API Rest. Cela implique de modifier les activités principales, et de créer des classes nécessaires à la gestion de l'API.

Analyse

Description de la structure

La structure est similaire à celle de la séquence précédente, mais présente en plus un dossier **Retrofit** qui regroupe tous les types de réponse possibles de l'API, et implémente également une nouvelle interface **ToDoApiInterface** dans l'activité **GenericActivity**.

Soit dans le dossier **Activities** :

- **MainActivity**
- **CheckListActivity**
- **ShowListActivity**
- **SettingsActivity**
- **GenericActivity** (comprend l'interface ToDoApiInterface)

Dans le dossier **Modele** (inchangé à part l'utilisation de @SerializedName sur les attributs) :

- **ProfilListeToDo**
- **ListeToDo**
- **ItemToDo**

Dans le dossier **RecyclerView** (inchangé) :

- **ListAdapter** (comprend la classe ListViewHolder et l'interface ActionListenerListe)
- **ItemAdapter** (comprend la ItemViewHolder et l'interface ActionListenerItem)

Dans le dossier **Retrofit** :

- **ReponseDeBase** (possède les attributs *version, success, status*)
- **ReponseHash** (extend ReponseDeBase + attribut *String hash*)
- **ReponseList** (extend ReponseDeBase + attribut *ListeToDo list*)
- **ReponseLists** (extend ReponseDeBase + attribut *List<ListeToDo> lists*)
- **ReponseItem** (extend ReponseDeBase + attribut *ItemToDo item*)
- **ReponseItems** (extend ReponseDeBase + attribut *List<ItemToDo> items*)
- **ReponseUsers** (extend ReponseDeBase + attribut *List<ProfilListeToDo> users*)

Points particuliers

1. Vérification du réseau

Pour la vérification de l'accès au réseau, que ce soit mobile ou wifi, j'ai utilisé la fonction donnée sur Moodle. Cette fonction renvoie un booléen indiquant s'il on est effectivement connecté au réseau ou non.

```
/*
Verifie l'état du réseau internet, renvoie un booléen correspondant à une connexion reussie ou non.
 */
public boolean verifReseau()
{
    // On vérifie si le réseau est disponible,
    // si oui on change le statut du bouton de connexion
    ConnectivityManager cnMngr = (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cnMngr.getActiveNetworkInfo();

    String sType = "Aucun réseau détecté";
    Boolean bStatut = false;
    if (netInfo != null)
    {

        NetworkInfo.State netState = netInfo.getState();

        if (netState.compareTo(NetworkInfo.State.CONNECTED) == 0)
        {
            bStatut = true;
            int netType= netInfo.getType();
            switch (netType)
            {
                case ConnectivityManager.TYPE_MOBILE :
                    sType = "Réseau mobile détecté"; break;
                case ConnectivityManager.TYPE_WIFI :
                    sType = "Réseau wifi détecté"; break;
            }

        }
    }

    this.alerter(sType);
    return bStatut;
}
```

Dans la fonction onCreate(), à l'initialisation de l'activité, le bouton permettant d'accéder aux pages suivantes (CheckListActivity, ShowListActivity, etc) n'est pas cliquable si la connexion n'est pas confirmée.

```
btnOK.setEnabled(verifReseau());
```

Une amélioration pour être d'effectuer la fonction verifReseau() à intervalles de tant réguliers tant que le réseau n'est pas disponible, afin de ne pas obliger l'utilisateur à redémarrer l'application si son réseau n'était pas activé dès le lancement (ce qui est le cas ici).

2. Initialisation de Retrofit

Pour la gestion de l'API, j'ai utilisé la librairie Retrofit.
Tout d'abord il a fallu ajouter les dépendances dans le dossier build.gradle.

```
implementation 'com.squareup.retrofit2:retrofit:2.6.0'
implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
```

Ensuite, j'ai crée une fonction initRetrofit() dans GenericActivity, qui crée un objet Retrofit et lui associe l'URL de l'API, ainsi qu'un convertisseur pour assurer la compréhension de l'API et des classes déjà créées. Cet objet appelle ensuite l'interface ToDoApiInterface, également créée dans GenericActivity. La fonction initRetrofit() est appelée à chaque méthode onCreate() des activités de mon application.

```
public void initRetrofit() {

    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(urlAPI)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

    toDoInterface = retrofit.create(ToDoApiInterface.class);
}
```

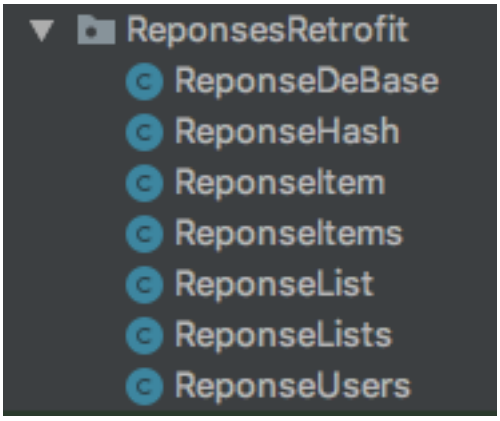
3. Interface ToDoApiInterface

L'objectif de l'interface est de définir les fonctions dont nous aurons besoin dans l'application. Ces fonctions doivent simuler un appel à l'API. Pour cela, on indique tout d'abord le type de méthode ainsi que la suite de l'URL (le début étant celle défini à la création de l'objet Retrofit). On indique à la fonction le type de réponse qu'elle recevra, et on place en paramètres les arguments qu'elle nécessite pour s'effectuer correctement. L'image suivante montre les fonctions présentes dans l'interface.

```
public interface ToDoApiInterface {
    @POST("authenticate")
    Call<ReponseHash> authenticate(@

    @GET("users")
    //Renvoie version, success, stat
    Call<ReponseUsers> recupUsers(@Q

    @GET("lists")
    //Renvoie version, success, stat
    Call<ReponseLists> recupLists(@Q

    @GET("lists/{idListe}/items")
    //Renvoie version, success, stat
    Call<ReponseItems> recupItems(@P
```

```
@POST("lists")
//Renvoie version, success, stat
Call<ReponseList> addList(@Query

@POST("lists/{idListe}/items")
//Renvoie version, succes, statu
Call<ReponseItem> addItem(@Path(

@PUT("lists/{idListe}/items/{idI
//Renvoie version, succes, statu
//Check = 1 veut dire oui
Call<ReponseItem> checkItem(@Pat

@DELETE("lists/{idList}")
//Renvoie version, success, stat
Call<ReponseDeBase> supressList(

@DELETE("lists/{idListe}/items/{
//Renvoie version, success, stat
Call<ReponseDeBase> supressItem(
```

Les réponses renvoyées par les fonctions de l'interface varient en fonction des demandes. Il a donc fallu créer une classe pour chaque type de réponse, afin de rendre le contenu de la réponse exploitable.



J'ai tout d'abord crée une classe ReponseDeBase, dont chaque autre classe de réponse hérite.

```
public class ReponseDeBase {

    @SerializedName("version")
    public String version;

    @SerializedName("success")
    public Boolean success;

    @SerializedName("status")
    public String status;
```

Pour que les données puissent être traitées correctement, j'ai ajouté des indications @SerializedName aux attributs des classes du dossier Modele (ProfilListeToDo, ListeToDo, ItemToDo), afin de pouvoir transformer les réponses reçues en des instances de ces classes. J'ai donc crée ensuite les autres types de réponses nécessaires, en y ajoutant leurs attributs spécifiques (exemple pour ReponseItem ci dessous).

```
public class ReponseItem extends ReponseDeBase {

    @SerializedName("item")
    public ItemToDo item;
}
```

Perspectives

Parmi les améliorations qui auraient pu être faites, la première est la création d'un nouvel utilisateur. Telle quel, l'application ne permet l'accès qu'aux utilisateurs possédant déjà un compte. Pour pouvoir ajouter un utilisateur, il aurait fallu s'authentifier avec un utilisateur connu (pour récupérer un hash), puis demander à l'API de créer un utilisateur.
Dans la même idée, il serait possible d'ajouter un onglet « Compte » où l'on retrouverait les informations de compte de l'utilisateur connecté (dernière connexion, nombre de listes, nombre d'items, mot de passe, etc). On pourrait alors permettre à l'utilisateur de modifier son mot de passe par exemple.

Une autre amélioration proposée dans l'énoncé était la connexion automatique de l'ancien utilisateur. Pour implémenter cela, la seule solution que j'ai trouvé était d'enregistrer le login et le mot de passe dans les SharedPreferences, et de tenter de connecter automatiquement si le hash enregistré était encore valable. Cependant, cette solution ne m'a pas semblé très sécurisée puisque cela impliquait de sauvegarder le mot de passe.

Bibliographie

- Documentation sur l'API utilisée : *https://documenter.getpostman.com/view/375205/S1TYVGTa?version=latest*
- Documentation sur Retrofit : *https://square.github.io/retrofit/*