CR TP2_TodoList

FANG Zhengjie
YANG Liu

1. Introduction

Ce projet a pour objectif d'établir une application TodoList qui peut satisfaire les besoins de faire une liste de chose à faire pour de différents utilisateurs avec différents pseudos, et nous pouvons gérer les pseudos avec le menu setting.

Pour construire une Todolist ayant les fonctions nécessaires, nous avons besoin de quatre activités en total, qui s'en chargent respectivement les quatre écrans qu'on utilise ici. On a réalisé avec fonctionnalités comme ci-dessous parmis les fonctionnalités proposé(Ce qui est en jaune sont des fonctionnalités ajoutées dans séquence 2):

MainActivity

- Permet de saisir son *pseudo* et *password*.
- Les informations pour le dernier utilisateur sont automatiquement renseignées dans le champ de saisie.
- Vérifier l'accès au réseau, et activer le bouton OK si le terminal utilisé peut accéder au réseau
- Lors du clic sur le bouton OK, demander une identification auprès de l'API en lui envoyant pseudo/mot de passe
- Récupérer le token d'identification sous forme d'un hash et le stocker dans les préférences de l'application
- Lors du clic sur le bouton setting dans le menu, une activité **SettingsActivity** s'affiche.
- Lors du clic sur le bouton OK, les informations sont sauvegardées dans les préférences partagées de l'application et une activité **ChoixListActivity** s'affiche.

SettingsActivity

- Permet d'afficher toutes les informations des utilisateurs
- D'autres champs utiles peuvent être ajoutés, pour débogage par exemple
- sauvegarder l'url de base de l'API

ChoixListActivity

- Récupérer les listes d'items associés à l'utilisateur connecté, et les afficher dans l'activité.
- Lors du clic sur le bouton NOUVELLE, une nouvelle liste est ajoutée dans la liste des listes de chose à faire.
- Lors du clic sur une des listes, passer à l'activité suivante en lui fournissant l'identifiant de la liste sélectionnée, et une activité **ShowListActivity** s'affiche.

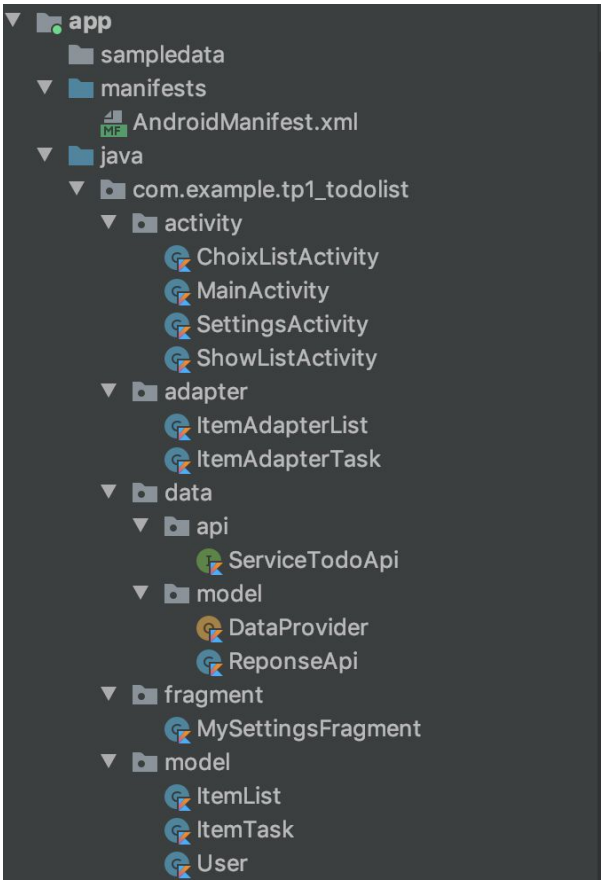
ShowListActivity

- Récupérer les items associés à la liste sélectionnée et les afficher dans l'activité.
- Permet de cocher et décocher un item.
- Lors du clic sur le bouton NOUVELLE, un nouvel item est ajouté dans la liste des items.

Améliorations facultatives :

- Utiliser des fragments à la place d'une PreferenceActivity
- Connecter automatiquement l'utilisateur sans passer par l'activité de connexion en stockant pseudo/passe dans les préférences; ajouter un menu "déconnexion" dans toutes les activités, qui fera revenir à l'activité de connexion
-

2.Implémentation Globale



Globalement, nous avons 5 packages, "activity" pour les activités, "adapter" pour les adaptateurs, "model" pour les modèles, "data" pour tout ce qui concerne les requêtes auprès de API et "fragments" prépare les fragments pour le "SettingActivity". Le package activity correspond aux activités mentionnées ci-dessus, qui sont en effet les différents écrans de l'utilisation. Le package adapter contient les adaptateurs pour l'utilisation du

RecyclerView , qui a besoin de l'adaptateur de faire les mises à jours sur les *Views* portés par les *ViewHolders.*

3.Analyse

Donc pour analyser tout le processus de la réalisation de cette application, les détails vont être précisés à la suite avec la démonstration du code. Nous allons tout d'abord nous concentrer sur les objets qu'on a besoin ici, c'est que les classes *ItemList, ItemTask* et *User*.

ItemTask

```
package com.example.tp1_todolist.model

import com.google.gson.annotations.SerializedName

data class ItemTask( @SerializedName( value: "label") var label:String?)  {
    @SerializedName( value: "id")
    val id:Int?=null
    @SerializedName( value: "checked")
    var checked :Int = 0 //private属性没有getter、setter

}
```

Nous devrions commencer par la classe *ItemTask*, car c'est l'objet le plus basique dans cette application. Dans cette classe, il y a 3 attributs auxquels on va faire attention, où l'attribut *check* est pour illustrer l'état d'être coché ou pas, et l'attribut *label* est la description des travails à faire en forme de *String*, *id* est l'identifiant de ce ItemTask dans base de données

ItemList

```
package com.example.tp1_todolist.model

import com.google.gson.annotations.SerializedName

data class ItemList( @SerializedName( value: "label") var title: String?)  {
    @SerializedName( value: "id")
    val id:Int? =null

    var myTask= mutableListOf<ItemTask>()

}
```

Cette classe a pour objectif de construire un modèle d'une liste des tâches à faire, où l'attribut *myTask* contient tous les items de type *ItemTask*, et l'attribut *title* est le nom de cette liste qu'on a créée auparavant, *id* est l'identifiant de ce ItemTask dans base de données

User

```
package com.example.tp1_todolist.model

import com.google.gson.annotations.SerializedName

data class User(@SerializedName( value: "pseudo") private var pseudoName:String?, private var password :String?) {
    @SerializedName( value: "id")
    val id:Int?=null

    var myList= mutableListOf<ItemList>()

}
```

Étant donné que l'on voudrait un objet qui peut contenir tous les informations de l'utilisateur, la classe *User* est créée pour générer ce type d'objet, dont l'attribut *myList* contient de différents listes de tâches créées par l'utilisateur spécifié. De plus, en tant qu'un utilisateur, il faudrait taper son *pseudoName* et *password.*

Ayant les bases sur les modèles, on peut ensuite commencer à analyser les fonctionnalités de différents activités concernantes dans cette application.

MainActivity

Dans cette classe, on a d'abord Vérifier l'accès au réseau, et activer le bouton OK si le terminal utilisé peut accéder au réseau. Lors du clic sur le bouton OK, on récupère pseudo et password qui sont saisis par l'utilisateur. A partir de password qui sont saisis , on a fait une requête "authenticate" auprès de l'API en lui envoyant pseudo/mot de passe pour vérifier que le pseudo et password sont correcte. Si c'est correcte, on récupérer le token d'identification sous forme d'un hash ainsi que le pseudo et mot de pass et les stocker dans les préférences de l'application. Puis une activité **ChoixListActivity** s'affiche.

ChoixListActivity

Dans cette activité, on récupérer d'abord le hash dans les préférences. En utilisant ce hash et fait une requête pour récupérer les listes de lists associés à l'utilisateur connecté et les afficher.
Lors du clic sur le bouton NOUVELLE,on va faire une requête créer une liste pour l'utilisateur connecté, une nouvelle liste est ajoutée dans la liste de list. Lors du clic sur une des listes, passer à l'activité suivante en lui fournissant
l'identifiant de la liste sélectionnée, et une activité **ShowListActivity** s'affiche.

ShowListActivity

Dans cette activité, on récupérer d'abord le hash dans les préférences. En utilisant ce hash et fait une requête pour récupérer les listes de tasks associés à l'utilisateur connecté et les afficher.
Lors du clic sur le bouton NOUVELLE,on va faire une requête créer une tâche pour

l'utilisateur connecté, une nouvelle task est ajoutée dans la liste de tâche. Lors du clic sur une des tâche, on peut cocher et décocher une tâche.

SettingsActivity

Dans les trois autres activités, il y a une actionbar dans lequel il y a une menu “Setting”. En cliquang sur “Settitng”, on dirige l’utilisateur à la activité “**SettingsActivity**”
Dans la class SettingActivity, on a utilisé un fragment qui est implémenté dans la class MyMySettingsFragment dans la module fragment. Pour l’instant il y a pas trop de fonctionnalités dans cette activité. Par contre, des fonctions plus complexe seront ajouté dans le “**SettingsActivity**” très facilement grâce aux Fragments

ItemAdapterList et ItemAdapterTask

On a utilisé notamment le RecyclerView dans les activites **ChoixListActivity** et **ShowListActivity**, qui nous oblige d'utiliser des adaptateurs correspondants.

4. Package “data”

Dans cette package, l’interface **ServiceTodoApi** définit tous les requêtes possibles auprès de l’API en utilisant bibliothèque **Retrofit.** Dans l’objet DataProvider , on a créer une instance de service de Retrofit. Cette objet prend en charge de récupérer les paramètres depuis les activités et fait des requêtes en utilisant les interfaces de **ServiceTodoApi.** La class **ReponseAPI** correspond aux les reponse de requêtes Retrofit qui est sous forme JSON.

Conclusion

Nous avons réussi à construire une application TodoList qui peut ajouter des listes de choses, nous pouvons également générer les informations de plusieurs utilisateurs. Pendant cette période de travail, on a pu bien réviser les cours sur le TP qui nous a appris comment utiliser les Views, les Adapters, les Layouts, le corotine de Kortlin et Retrofit, et nous avons aussi acquis pas mal de bonnes pratiques dans le domaine de développement Android.

Perspectives

- Donner un attribut de temps pour lier les choses au calendrier.
- Ajouter une fonction chercher pour rapidement localiser les items ce qu’on veut trouver.
- Completer l’activity **SettingActivity** et les fragments
- Ajouter des longClickedListener sur les List et Item pour les supprimer.