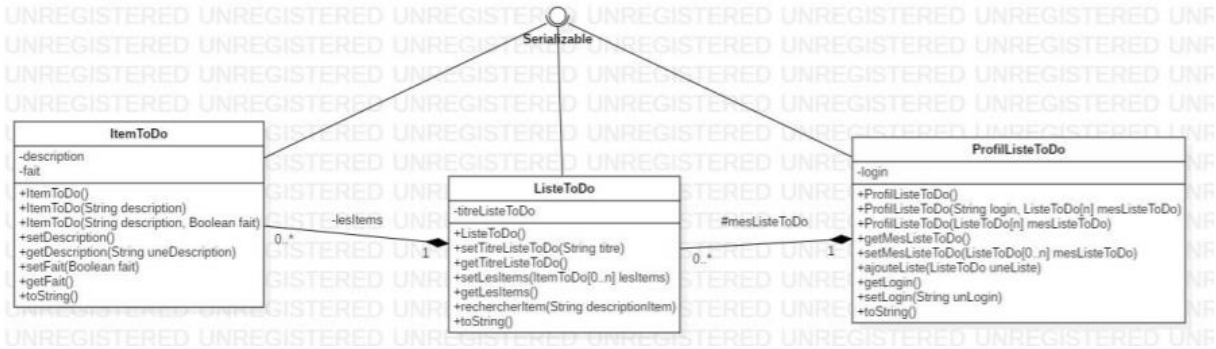
Compte rendu application ToDoList séquence 3

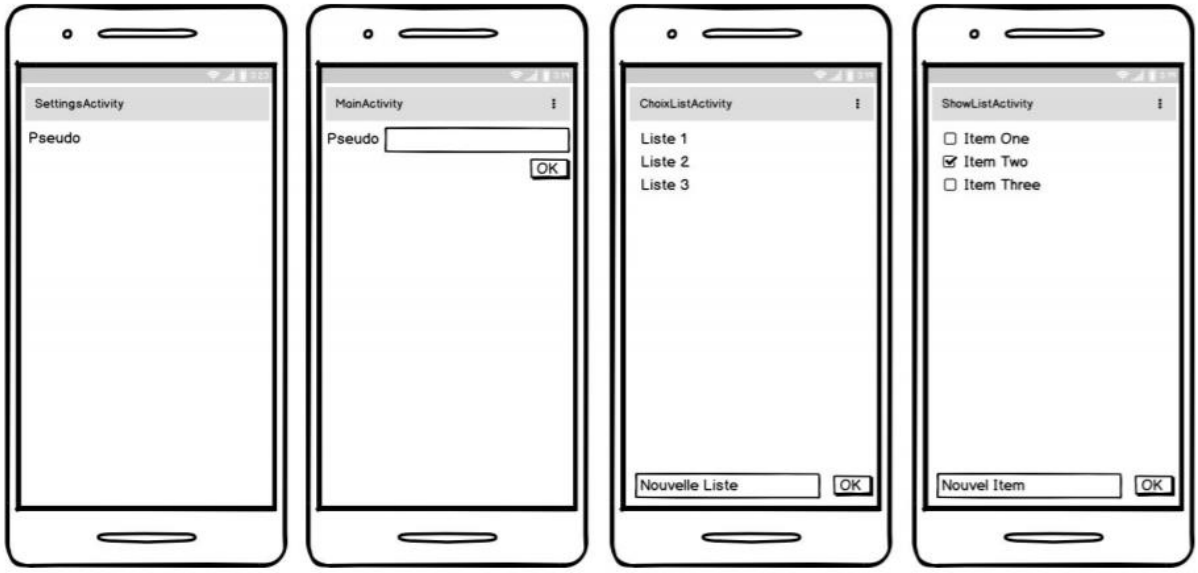
Table des matières

Introduction

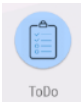
Ce projet a pour objectif l'élaboration d'une application ToDoList sur Android avec Java. **Les données sont stockées sur un serveur et sur une base de données locale. On fera les download et upload via une API, et les sauvegardes locales (en cache) dans une base de données SQL.**

Pour rappel, voici le diagramme de classe de l'application et les mockups des pages à afficher :





Par rapport aux consignes énoncées, voici les choix techniques :

- Les profiles des utilisateurs sont stockés dans des fichiers au format json.
- Un bouton Suppr. pour supprimer des items a été ajouté à la ShowListActivity
- Un bouton Suppr. Pour supprimer des listes a été ajouté à ChoixListActivity
- L'icone de l'application est celle-ci : 

- Pour les deux recyclerView de choixListActivity et ShowListActivity, on a utlisé deux RecyclerViewAdapter différents (pas d'heritage).
- **Pour les appels à l'API, on utilise la librairie Retrofit.**
- **Pour les modifications de la base de données locale, on utilise Room.**

Analyse

Entre la séquence 2 et 3, il y a eu une refonte de l'organisation des classes java.

Différents packages ont été créés :

- Activities
- API_models : package avec les classes permettant d'interagir avec l'API.
- Database : package avec les classes permettant d'interagir avec la BDD.
- Models
- Ui : package de classes pour l'interface utilisateur, ici les recycler view adapter.
-

Détail des classes

Package « models »

ItemToDo représente un item d'une liste.

ListeToDo représente une liste.

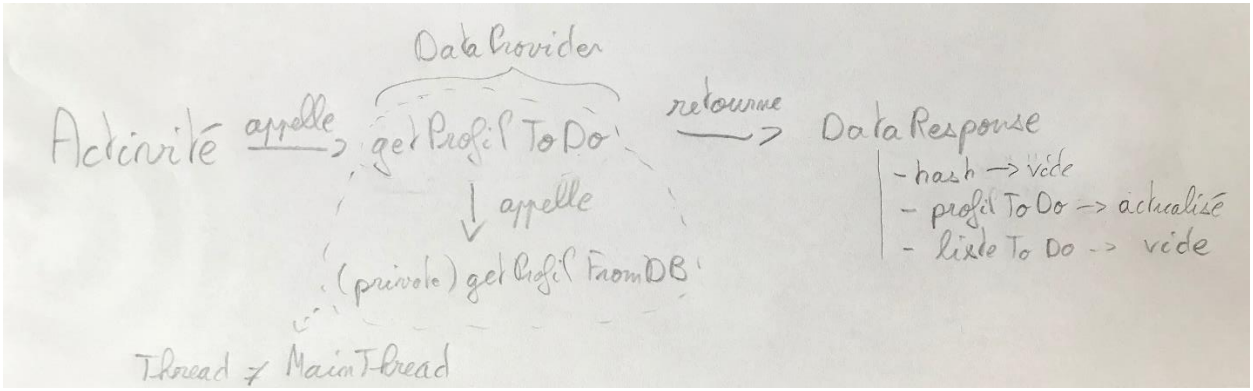
ProfileListeToDo représente un utilisateur.

InternetCheck : Cette classe créer un socket et tente de se connecter à google.com (ip 8.8.8.8). Si la connexion réussie, c'est que l'on est connecté à internet. Pour utiliser des sockets, on doit créer un nouveau thread. C'est pourquoi pour utiliser cette classe, l'objet qui instance InternetCheck doit contenir la méthode isConnectedToInternet(Boolean internet) qui sera appelé après la tentative de connexion à google.com. Dans ce projet, isConnectedToInternet est utilisé dans MainActivity qui définit isConnectedToInternet comme suit :

```
@Override
public void isConnectedToInternet(Boolean internet) {
    if (!internet){
        btnOk.setEnabled(false);
        Toast.makeText( context: this,  text: "Aucune connexion internet.", Toast.LENGTH_SHORT).show();
    } else {
        btnOk.setEnabled(true);
        Toast.makeText( context: this,  text: "Connexion internet établie.", Toast.LENGTH_SHORT).show();
    }
}
```

DataProvider : Classe principal d'accès aux données. La plupart des fonctions sont commentées. Au début de chaque fonction, on vérifie l'accès à internet. DataProvider contient des méthodes publiques que l'on appelle dans les activités, et des méthodes privés permettant l'accès soit à l'API soit à la BDD. Les méthodes publiques sont lancées dans un nouveau thread comme vu en cours (avec des executor), tandis que les méthodes privées sont lancées dans le thread actif. Puisque seules les méthodes publiques de dataprovider appellent les méthodes privées, ces dernières sont de facto lancés dans un thread autre que le MainThread. Les résultats des méthodes publiques (getProfilToDo, getListeToDo…) sont transmis aux activités avec un handler (cf. uiHandler, **Postlistener** de M. Boukadir). Ces résultats sont contenus dans une classe **DataResponse**, semblable à retromain pour l'api, qui peut contenir un profilToDo, une listeToDo ou encore un hash, et en fonction de la méthode appelé, certains de ces attributs sont initialisés et l'objet DataResponse crée est transmis à l'activité.

Schéma récapitulatif :



Voici le détail de quelques méthodes publiques importantes :

- Authenticate : si on a une connexion internet, on essai l'authentification auprès de l'api, et si elle réussit, on renvoi le hash. Si il n'y a aucune connexion internet, on essai de s'authentifier sur la BDD (profil en cache ?).
- getProfilToDo, utilisé dans MainActivity, retourne le profil depuis la base de données.
- getListeToDo, utilisé dans ChoixListActivity, retourne la liste depuis la base de données.
- AddListeToDo, DelListeToDo etc, ajoute supprime ou met à jour l'élément concerné dans la base de données et sur l'api si l'utilisateur est connecté dessus.
- updateAPIfromDB ajoute tous les éléments de la base de données (listes et items) que ne contient pas l'API, dans l'API : si MaListe est présente dans la BDD mais pas dans l'API, MaListe est ajouté dans l'API.
- updateDBfromAPI, fait l'inverse et ajoute tous les éléments de l'API dans la BDD. Puis elle supprime tous les éléments de la BDD qui ne sont pas présent dans l'API.

Pour voir la séquence d'appelles à ces fonctions, voir timeline.

Package « database »

MyDatabase : hérite de RoomDatabase, permet l'accès à la BDD. Elle contient une méthode permettant d'autoriser qu'un seul accès simultané à la BDD : getInstance.

Package « dao » : package contenant tous les « database access object » c'est-à-dire les interfaces definissant les fonctions SQL. Une classe dao correspond à une table dans la BDD.

Package « API_models »

TodoInterface : Postman a beaucoup été utilisé pour écrire cette classe. On crée une fonction pour chaque appel @Get ou @POST etc. Les retours de ces fonctions sont des instances de la classe RetroMain. Retromain est une classe pouvant contenir tous les objets du json reçu. De plus si l'on a une liste contenant d'autres objets dans le json reçu par l'API, par exemple lorsque l'on reçoit la liste des utilisateurs, l'instance de RetroMain reçu crée une liste d'objets RetroMain également représentant la liste des utilisateurs. Cela permet de n'avoir qu'un modèle de donnée, mais demande

à connaitre parfaitement le résultat de la requête http pour savoir quels attributso nt été initialisés, d'où l'utilisation de Postman.

RetroMain : Définit les variables retournées par les requêtes http. Voir TodoInterface ci-dessus pour plus de détails. Les variables définies doivent être sérialisable d'où le @SerializedName(« nomvariable »).

Package « activities »

- MainActivity

Est l'activité de démarrage de l'application. Elle permet de saisir un pseudo et un mot de passe pour se connecter à l'API. Si la connexion réussie, on passe à ChoixListActivity.

Au lancement de l'application, s'il n'y a pas de connexion internet, le bouton ok est désactivé, rendant les tentatives de connexion impossibles.

- SettingActivity

Préférences de l'application. Elle comporte trois préférences : le dernier pseudo saisi par l'utilisateur, le hash permettant d'effectuer des appels à l'API en tant qu'utilisateur connecté, et isConenctedToInternet qui informe sur l'état de la connexion lorsque l'activité actuelle a été lancée. Le hash est initialisé à la fin d'une authentification réussie dans MainActivity.

- ChoixListActivity

On affiche les listes du profil dans un RecyclerView. Un clic sur une liste permet d'accéder à ShowListActivity.

- ShowListActivity

Cette activité charge les items de la liste préalablement sélectionné grâce au même fichier utilisé précédemment. On analyse ensuite chaque item pour savoir si oui ou non il a été coché précédemment et on le coche si oui.

Package « ui »

RecyclerViewAdapterList : adpatateur pour le recyclerview qui affiche les listeToDo.

RecyclerViewAdapterItem : atateur pour le recyclerview qui affiche les itemToDo.

Package principal

Utils : classe avec une méthode statique définissant les executor utilisés dans dataProvider. Ici, seuls 2 autres threads que le thread principal ont été autorisés.

Timeline

Cette section montre comment fonctionne l'application au niveau temporel : comment on passe d'une activité à une autre, quand met-on à jour l'API depuis la BDD ou inversement etc.

Règle : tous les éléments ajoutés s'ajouteront nécessairement sur l'API quand une connexion sera retrouvée (ce qui peut arriver à un prochain lancement de l'application).

Choix arbitraire : Les suppressions locales sans internet ne supprimeront pas les items sur l'API une fois la connexion retrouvée (sinon la BDD doit être agrandie pour sauvegarder temporairement les éléments supprimés).

1. Lancement de l'application : MainActivity.
2. Vérification de la connexion internet.
    a. Si internet : démarche d'authentification autorisée.
    b. Sinon : demande à l'utilisateur s'il veut modifier les données locales (qui modifieront l'API ensuite). S'il ne veut pas, l'application est inutilisable.
3. Authentification
    a. Réussite avec internet : updateAPIfromDB ajoute les éléments locaux qui ne sont pas sur l'API. Puis updateDBfromAPI ajoute les éléments de l'API (toutes les listes et items du profil) qui ne sont pas présents localement.
    b. Reussite sans internet : chargement du profil local.
    c. Echec : retour à MainActivity avec un message d'erreur.
4. Ajout et suppression de liste/item :
    a. Si internet : ajout ou suppression de l'élément sur l'api et en local.
    b. Sinon : ajout ou suppression de l'élément en local. Une fois la connexion retrouvée, seuls les ajouts se feront sur l'API via la fonction updateAPIfromDB.

Conclusion

L'application ToDo est terminée. Cette séquence a été la plus lourde en travail avec environ 25h de travail. La principale difficulté a été d'écrire la classe DataProvider. Une fois écrite, le reste du travail était beaucoup plus simple : plus besoin de voir comment accéder aux données.

Bibliographie

- RecyclerView : https://www.youtube.com/watch?v=Vyqz_-sJGFk&t=499s
- Ecriture/Lecture de fichiers sur android : https://support.google.com/docs/answer/49114?co=GENIE.Platform%3DAndroid&hl=fr
- Debug : stackoverflow.com

- Utilisation de retrofit : https://medium.com/@prakash_pun/retrofit-a-simple-android-tutorial-48437e4e5a23
- Utilisation d'un executor : https://stackoverflow.com/questions/52164957/usage-of-executor-in-android-architecture-components
- Guide sur Room : https://guides.codepath.com/android/Room-Guide