

Ministère de l'Enseignement Supérieur et de la Recherche et de l'Innovation
Université de Lille
École Centrale de Lille



Programmation mobile et réalité augmentée

Sujet de rapport : **Développement de l'application android ToDoList version 3**

Rédigé Par : **Imen Kerkeni**
Wissem Chabchoub



Année universitaire : **2018 - 2019**

Table des matières

Contexte	1
1 Analyse	2
1.1 Activités	2
1.1.1 Main Activity	3
1.1.2 ChoixListActivity	4
1.1.3 ShowListActivity	5
1.2 RecyclerView	6
1.2.1 item_list.xml et item_item.xml	6
1.2.2 ItemAdapter.java :	6
1.3 Persistance des données	7
1.4 Diagramme UML	8
Conclusion	9

Contexte

N'oubliez plus jamais une tâche importante, Organisez votre vie puis profitez-en!!” La vie peut parfois sembler écrasante, mais ça n’a pas à être le cas. Avec “ToDo”, dans sa dernière version on vous garantie la tranquillité d’esprit et la sauvegarde de vos données dans notre API même si vous n’êtes pas connectés. Notre application android “ToDo” qui permet aux utilisateurs de lister,ajouter, sauvegarder et mettre à jour leurs "ListToDo" en toute simplicité.

La description détaillée de fonctionnalité de l’application en sa deuxième version se trouve dans les paragraphes suivantes.

Mots-clés : List, Activités, API,Requêtes, RecyclerView, Volley, ROOM, SQLite

Dans cette dernière version, nous avons amélioré notre application en proposant un mode offline. C'est à dire l'utilisateur peut récupérer ses listes et ses données à partir d'une base de donnée locale "cache" organisée sous la forme d'une base SQLite.

1.1 Activités

Dans cette version on ajouté un package DAO pour utiliser et gérer la base de donnée(insertion, mise à jour...). Ce package contient trois classes (itemDaO, ListDao, RequestDao).

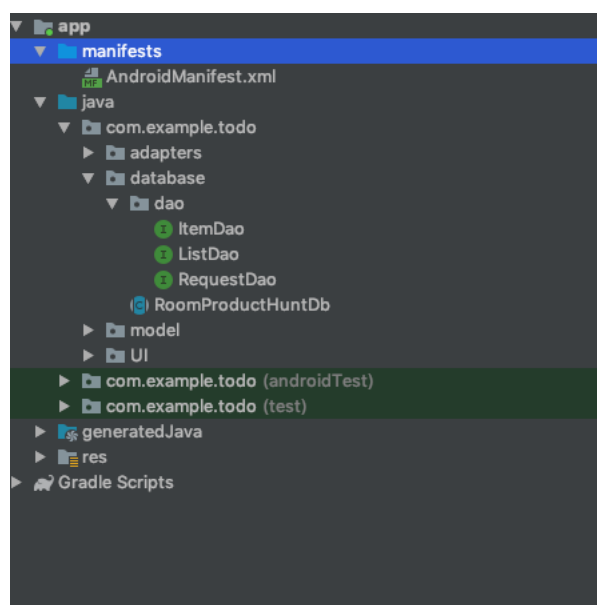


FIGURE 1.1 — Settings Activity

L'application se base sur trois activités principales qui consrtuit notre modèle (package model) :

- ◇ MainActivity : dans laquelle l'utilisateur saisit son login et son mot de passe.
- ◇ ChoixListActivity : dans laquelle on visualise une liste des ListToDo de l'utili-

sateur en question.

- ◇ ShowListActivity : dans laquelle on visualise les itemToDos de la liste sélectionnée.

Dans l'activité Settings on a ajouté l'url de base de l'API `http://tomnab.fr/todo-api/` et qu'on peut l'éditer.

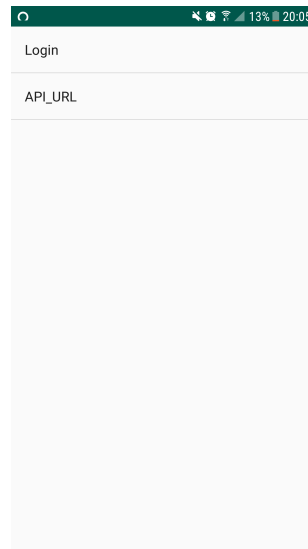


FIGURE 1.2 — Settings Activity

1.1.1 Main Activity

Une vérification de l'accès au réseau se lance lors de la démarrage de l'application et qui permet l'activation du bouton Ok si le terminal utilisé peut accéder au réseau. L'activité principale permet à l'utilisateur de saisir son login dans un champ Auto-Complete ainsi que son mot de passe. Cette connexion à l'api permet de récupérer le token d'identification et de le stocker dans les préférences de l'activité.

Dans cette version, l'utilisateur n'a pas besoin de connexion car s'il y a pas de réseau ses données sont générées à partir de la base de donnée qu'on a crée. En d'autres termes manipuler les données en cache. Après l'étape de l'identification l'activité ChoixListActivity se lance.

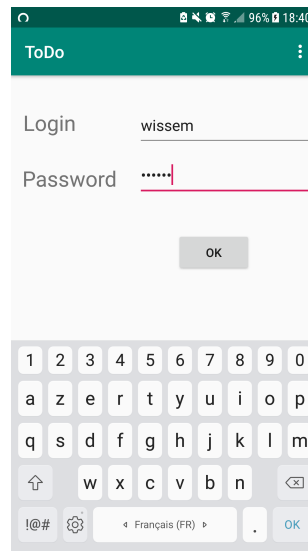


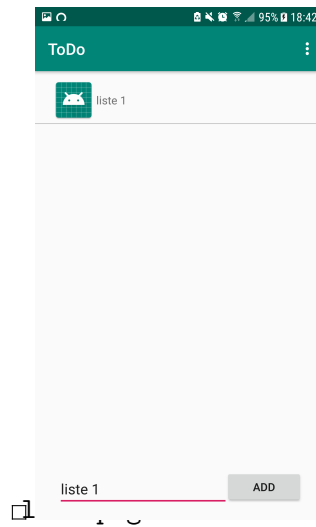
FIGURE 1.3 — Main Activity

```
} else {  
    Toast.makeText( context: this, text: "Accessing offline mode", Toast.LENGTH_LONG).show();  
    Intent toSecondAct = new Intent( packageContext: MainActivity.this, ChoixListActivity.class);  
    startActivity(toSecondAct);  
}  
break;
```

FIGURE 1.4 — Fonctionnement hors ligne

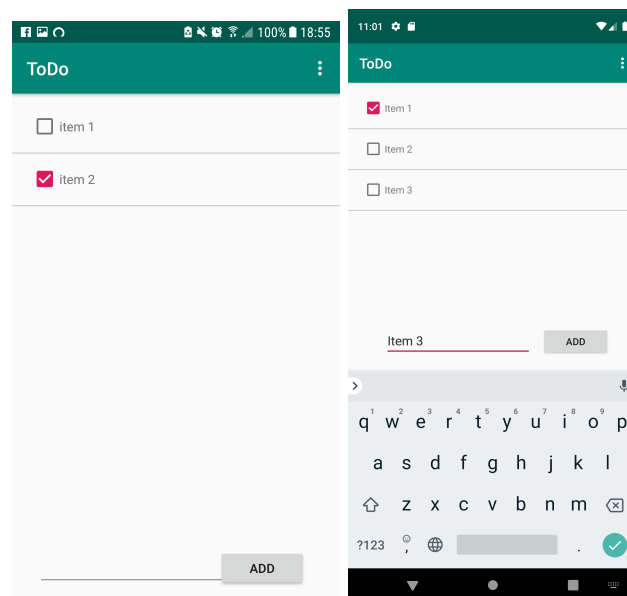
1.1.2 ChoixListActivity

Cette activités garde ses fonctionnalité comme les deux dernière versions Dans cette activité, on affiche les ListeToDos de l'utilisateur connecté en utilisant un RecyclerView. Le bouton add permet d'ajouter une nouvelle liste dont le nom est saisi dans le EditText. Les données dans l'API sont alors mises à jour. Le glissement d'un item permet de le supprimer. Un snack bar s'affiche dans ce cas et propose de remettre la liste supprimée. Le clique sur une liste permet de passer l'activité ShowListActivity en passant le id de la liste sélectionnée.

**FIGURE 1.5** — ChoixListActivity

1.1.3 ShowListActivity

Comme dans les deux dernières versions de l'application, cette activité, on affiche les ItemsToDos de la ListeToDo qu'on les a récupéré de l'API par des requêtes GET. Les actions de glissement et le bouton Add permettent respectivement de la suppression et l'ajout de la liste concernée. Les ItemsToDos sont équipés avec un CheckBox. ainsi on peut cocher/decocher chaque item. La 3ème version permette aussi de faire cocher et décocher les items et dès que la connexion s'établie l'api TODO se met à jour.

**FIGURE 1.6** — ShowListActivity

1.2 RecyclerView

Le recyclerView permet d'afficher des items en optimisant l'utilisation des ressources disponibles. Il se base sur un ItemAdapter (classe Java) et un design des items enregistré sous format xml.

1.2.1 item_list.xml et item_item.xml

Ce fichier xml implémente le design des items du recyclerView.

- i) item_list : Il concerne des items à afficher dans la liste des ListeTodos.
- ii) item_item : Il concerne des items à afficher dans la liste des ItemTodos.

1.2.2 ItemAdapter.java :

Dans ce projet, on a créé deux ItemAdapters en fonction des fichiers item.xml. Leur contenu est le même à l'exception du fichier xml qu'ils utilisent.

L'item adapter contient cinq attributs :

1. La liste des items à afficher
2. Une interface ActionListener qui contient des méthodes à implémenter par l'activité qui contient le RecyclerView.
3. Dernier item supprimé
4. Position du dernier item supprimé
5. L'activité parente.

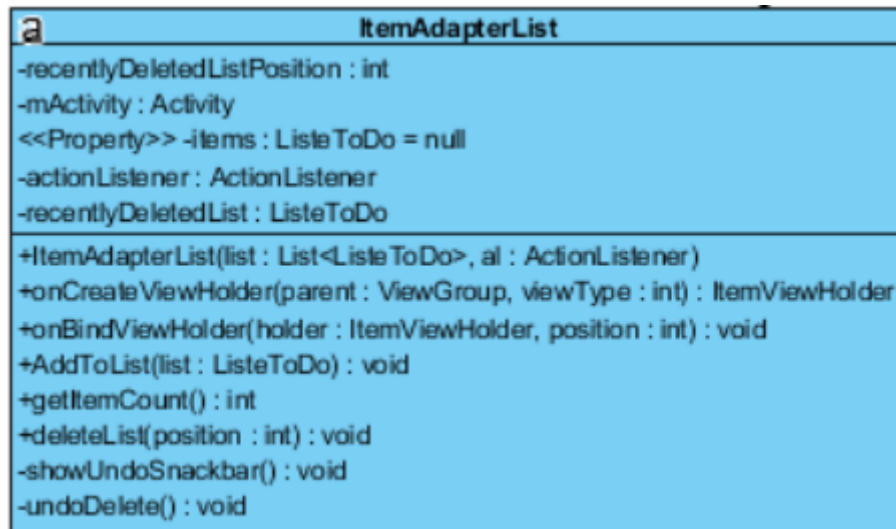
Le constructeur de l'adaptateur permet d'initialiser la liste des items et le ActionListener.

Dans la méthode bind, on attribue l'id de l'item au TextView ou au CheckBox pour identifier l'item en cas de clique ou de modification.

La méthode deleteListe permet de supprimer l'élément glissé et enregistrer cet élément et sa position. Si l'utilisateur choisit d'annuler la suppression, la méthode undoDelete, remet l'élément dans la liste des items dans sa position initiale.

Les méthodes deleteListe et undoDelete et le clique sur un item appellent les fonctions de l'interface ActionListener qui enregistrent les modifications dans les préférences

WaitingRequest DAO(itemDAO, ListDao, RequestDao) ProductHuntDb



```
class ItemAdapterList {
    -recentlyDeletedListPosition : int
    -mActivity : Activity
    <<Property>> -items : List<ToDo> = null
    -actionListener : ActionListener
    -recentlyDeletedList : List<ToDo>

    +ItemAdapterList(list : List<List<ToDo>>, al : ActionListener)
    +onCreateViewHolder(parent : ViewGroup, viewType : int) : ItemViewHolder
    +onBindViewHolder(holder : ItemViewHolder, position : int) : void
    +AddToList(list : List<ToDo>) : void
    +getItemCount() : int
    +deleteList(position : int) : void
    -showUndoSnackbar() : void
    -undoDelete() : void
}
```

1.3 Persistance des données

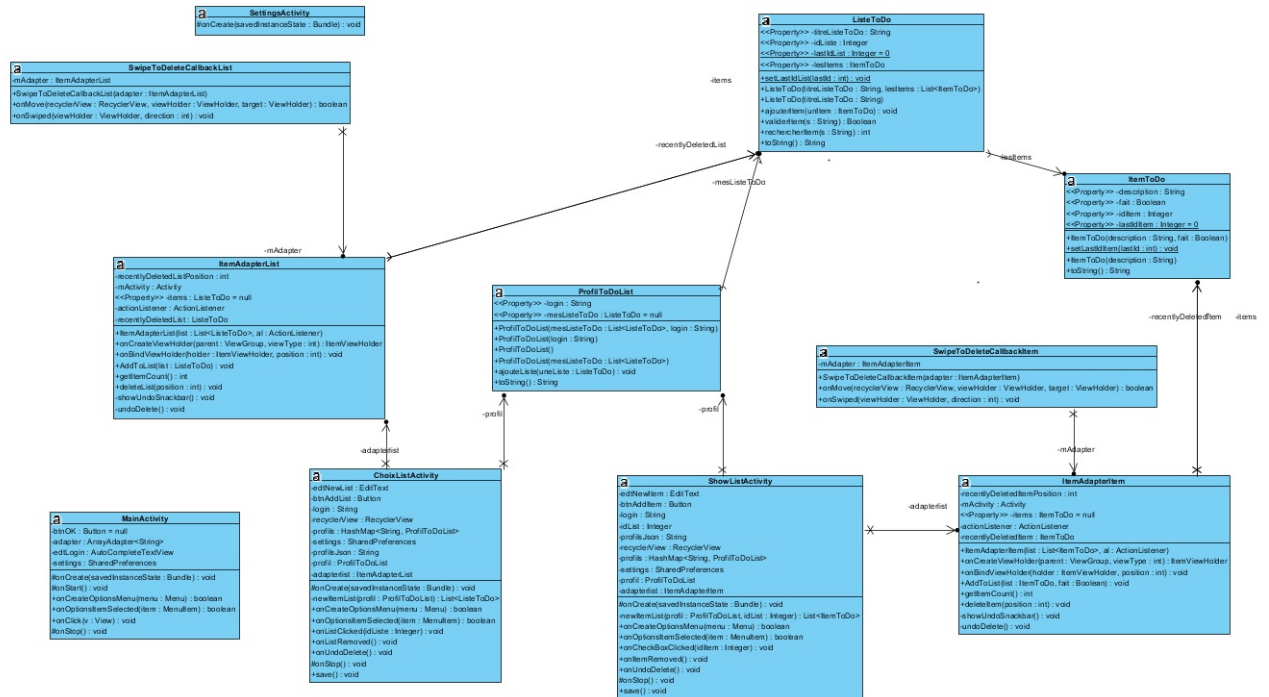
Dans notre trisieme version de notre application, nous avons utilisé un sauvegarde local des données de l'utilisateur pour lui permettre de les utiliser en hors ligne. Pour ce faire on a avons utilisé le coix entre SQLite et Room. En effet, SQLite est une base de données SQL, elle stocke les données de notre application dans un fichier texte. Il est très léger et aussi open source. La base de données SQLite supporte les fonctionnalités standard de la base de données relationnelle. Il prend en charge les types de texte, d'entier et de données réelles. Android.database.sqlite package a tout ce dont nous avons besoin pour sauvegarder les données de votre application. La classe SQLiteDatabase dispose de nombreuses méthodes différentes pour créer une base de données, exécuter des commandes SQL et supprimer une base de données.

Alors que Room fournit une couche d'abstraction sur SQLite pour permettre un accès fluide à la base de données tout en exploitant toute la puissance de SQLite. On a choisi d'utiliser Room et faire bénéficier notre application de la persistance de ces données localement(la mise en cache des données pertinentes). Tout changement de contenu initié par l'utilisateur est ensuite synchronisé sur le serveur une fois que l'appareil est de nouveau en ligne.

L'application utilise la base de données Room pour obtenir des DAO associés à cette base de données. L'application utilise ensuite chaque DAO pour récupérer les entités de la base de données et enregistrer tout changement apporté à ces entités dans la base de données. Enfin, l'application utilise une entité pour obtenir et définir les valeurs qui correspondent aux colonnes des tables de la base de données.

1.4 Diagramme UML

Le diagramme UML complet est dans ce lien : [Lien](#)



Conclusion

On a pu dans cette version d'améliorer notre application en proposant un mode hors ligne. C'était une bonne occasion de manipuler des nouvelles fonctionnalités et méthodes des bases de données locales. Durant l'étape de développement de notre application on a rencontré des différentes difficultés (bugs, blocage, connaissance..) mais on a pu dépasser ces situations grâce à des connaissances et des méthodes qu'on a pu voir durant les séminaires et des recherches et des lectures bibliographiques sur internet.

Bibliographie

- [1] <https://developer.android.com/training/data-storage/room>
- [2] <https://developer.android.com/reference/android/widget/AutoCompleteTextView.html?fbclid=IwAR1Usva0KqKKjPrl7qeZ7SFUeheiA8>
- [3] <https://openclassrooms.com/fr/courses/4568576-recuperez-et-affichez-des-donnees-distances/4893781-implementez-votre-premiere-recyclerview>
- [4] <https://developer.android.com/reference/android/preference/PreferenceActivity>
- [5] <https://docs.microsoft.com/fr-fr/dotnet/framework/wcf/feature-details/how-to-serialize-and-deserialize-json-data>
- [6] <https://www.lemagit.fr/conseil/Comment-connecter-une-API-JSON-a-un-projet-Android>