

Séquence 3

CR TP 3

Élèves: Cristopher Pérez
Asunción Gómez Colomer
Enseignant: Thomas Bourdeaud'huy

Programmation mobile et réalité augmentée
Date: 19 de junio de 2021
Villeneuve d'Ascq, 59650

Table des Matières

1. Introduction	1
2. Activities	1
3. Data	1
4. ViewModel	2
5. Analyse	3
6. Conclusion	3

Liste des figures

1	Storyboard	1
---	----------------------	---

Liste des Codes

1.	Classe View Model	2
2.	Recuperation des donnees de ViewModel dans Activity	2

1. Introduction

Nous souhaitons améliorer l'application développée à la séquence 2 en proposant un mode offline. Pour cela, à chaque récupération de listes ou d'items auprès de l' todo, l'application doit stocker les informations collectées dans un “cache” organisé sous la forme d'une base SQLite.

Objectifs:

- Mise en place d'une persistance de données sur le téléphone à l'aide d'une base SQLite
- Utilisation de l'ORM Room
- Gestion d'un mode offline

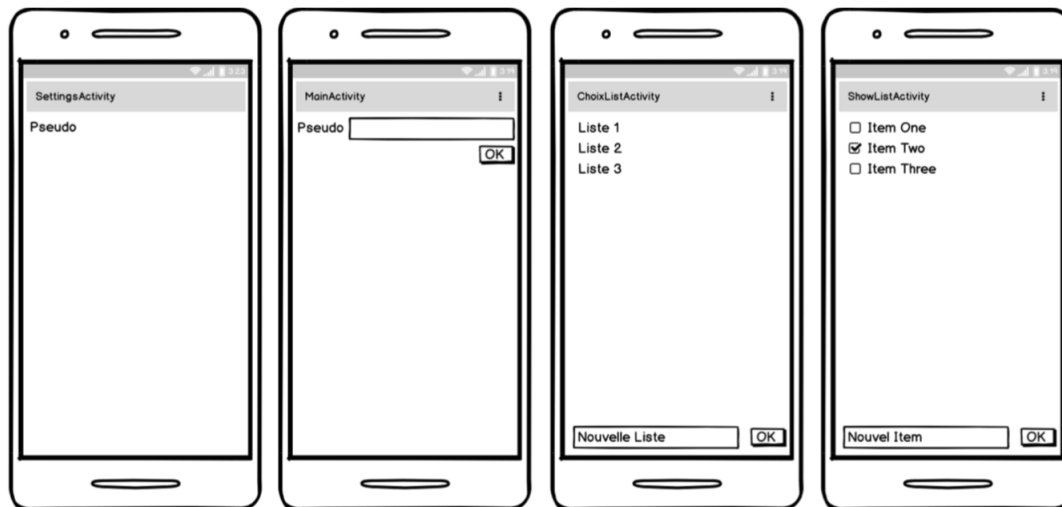


Figure 1: Storyboard

2. Activities

Pour les activités on a toujours les mêmes qu'on a utilisé dans les séquences précédentes mais cette fois avec l'implémentation de ViewModel. Alors comme on a observé dans le code 2 il faut appeler les fonctions faites dans ViewModel depuis l'activity mais en utilisant l'observe et comme ça récupérer les données ou faire les opérations. Aussi grâce à cette implémentation on a ajouté une ProgressBar pour pouvoir visualiser le temps de chargement entre les fonctions de nos activités

3. Data

Pour la manipulation des données, on a créé trois repositories, où on peut obtenir les données soit en faisant une requête à l'API, soit en faisant une requête SQL à la base de données local.

Pour cela, on a ajouté un ‘Local Data Provider’ qui nous permet d’obtenir les données à partir d’une instance d’un DAO (Data Access Objects) qui fournit les méthodes pour faire des queries, insertions, update et de supprimer des données de la base de données. Dans cette partie, on a utilisé la librairie Room.

4. ViewModel

On utilise la classe ViewModel pour pouvoir garder et administrer des données de l’UI d’une manière plus optimisée, avec cette classe on peut conserver des données après de faire des changements comme la rotation d’écran.

Pour l’utiliser il faut diviser les tâches et faire l’acquisition des données avec le ViewModel comme on peut voir dans le code 1.

Ainsi, on a créé une classe ViewModel par objet, donc on a un ViewModel pour une liste, pour un item et aussi pour un utilisateur.

Code 1: Classe View Model

```
1 class ListViewModel(application: Application): AndroidViewModel(application) {
2     private val listRepository by lazy { ListRepository.newInstance(application) }
3
4     val lists = MutableLiveData<ListViewModel.ViewState>()
5
6     fun getList(id: Int, hash: String){
7         viewModelScope.launch {
8             lists.value = ListViewModel.ViewState.Loading
9             try {
10                 lists.value = ListViewModel.ViewState.Content(lists =
listOf(listRepository.getList(id, hash)))
11             } catch (e: Exception) {
12                 lists.value = ListViewModel.ViewState.Error(e.message.orEmpty())
13             }
14         }
15     }
16 }
17 }
```

Pour après récupérer les données dans notre Activity avec le code 2

Code 2: Récupération des données de ViewModel dans Activity

```
1 class MyActivity : AppCompatActivity() {
2
3     override fun onCreate(savedInstanceState: Bundle?) {
4         // Create a ViewModel the first time the system calls an activity's onCreate()
method.
5         // Re-created activities receive the same MyViewModel instance created by the
first activity.
6
7         // Use the 'by viewModels()' Kotlin property delegate
8         // from the activity-ktx artifact
9         val model: MyViewModel by viewModels()
```

```
10     model.getUsers().observe(this, Observer<List<User>>{ users ->
11         // update UI
12     })
13 }
14 }
```

Alors pour notre programme on a utilisé 2 ViewModel, ItemViewModel et ListViewModel pour pouvoir diviser les tâches qu'on avait déjà dans la séquence 2 comme la connexion des utilisateurs et la manipulation des listes. En utilisant les exemples de code montrés précédemment on a fait toutes les fonctions nécessaires et l'adaptation à notre code

5. Analyse

L'utilisation de 'lazy' nous a permis de créer une meilleure application qui n'a pas besoin de re-crée des objets dont on en a déjà créé. Alors comme ça l'implantation de ViewModel on l'a utilisé seulement pour charger les données et toutes les autres fonctions restent un peu pareils, à l'exception que cette fois pour récupérer et ajouter des données on a utilisé les Repositories qu'on a mentionné avant

6. Conclusion

On a réussi à implémenter toutes les fonctions qui interagissent avec l'API et avec la base de données locale. Cependant, on n'a pas pu utiliser ces fonctions dans les activités.

La principale difficulté rencontrée c'était la bonne implémentation de ViewModel avec les Repositories, mais en utilisant la documentation d'Android et de SQL on a réussi

Références

- [1] ViewModel https://developer.android.com/topic/libraries/architecture/viewmodel?gclid=CjwKCAJ...JQDYmr5OfZy4BH00y8tBRHQIp1QcxoCJoQAvD_BwEg&src=aw.ds