École Centrale de Lille
MIN
Mathématiques -Informatique

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

Liste des figures

Liste des Codes

1. Introduction

Nous souhaitons améliorer l'application développée à la séquence 2 en proposant un mode offligne. 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

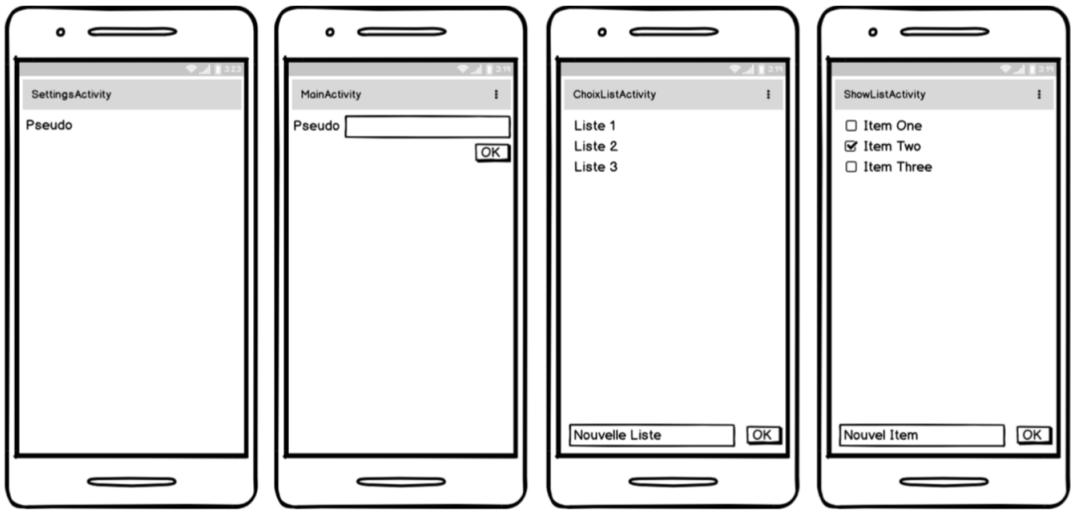

Figure 1: Storyboard

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 donnes 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 crée 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 methodes pour faire des query, 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 donnes de l'UI d'une manière plus optimise, avec cette classe on peut conserver des donnes après de faire changes comme la rotation d'écran

Pour l'utiliser il faut diviser les taches et faire l'acquisition des donnes avec le ViewModel comme on peut voir dans le code 1

Ainsi, on a crée une class view model par object, donc on a un view model pour une liste, pour un item et aussi pour un utilisateur.

Code 1: Classe View Model

```
class ListViewModel(application: Application): AndroidViewModel(application) {
    private val listRepository by lazy { ListRepository.newInstance(application) }

    val lists = MutableLiveData<ListViewModel.ViewState>()

    fun getList(id: Int, hash: String){
        viewModelScope.launch {
            lists.value = ListViewModel.ViewState.Loading
            try {
                lists.value = ListViewModel.ViewState.Content(lists =
    listOf(listRepository.getList(id, hash)))

            } catch (e: Exception) {
                lists.value = ListViewModel.ViewState.Error(e.message.orEmpty())
            }
        }
    }
}
```

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

Code 2: Recuperation des donnes de ViewModel dans Activity

```
class MyActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        // Create a ViewModel the first time the system calls an activity's onCreate()
    method.
        // Re-created activities receive the same MyViewModel instance created by the
    first activity.

        // Use the 'by viewModels()' Kotlin property delegate
        // from the activity-ktx artifact
        val model: MyViewModel by viewModels()
```

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

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

5. Analyse

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

6. Conclusion

On a réussi à implémenter toutes les fonctions qui interagissent avec l'API et avec la base de données local. On a aussi pu implémenter ces fonctions dans les différentes activités grâce au view model.

La principale difficulté rencontre c'était la bonne implémentation de ViewModel avec les Repositories, mais cela a été possible grâce à l'utilisation de la documentation d'Android et de SQL.

Références

[1] ViewModel `https://developer.android.com/topic/libraries/architecture/viewmodel?gclid=CjwKCAj`
$JQDYmr5OfZy4BH00y8_tBRHQIp1QcxoC_JoQAvD_BwEgclsrc = aw.ds$