DAA - Laboratoire 04

Développement d'Applications Android

Martins Alexis, Saez Pablo

HEIG VD

5 décembre 2023

Table des matières

1 Introduction

Dans le cadre de ce laboratoire, nous avons développé une application Android basée sur l'architecture MVVM (Modèle-Vue-VueModèle) et exploitant la puissante base de données Room. Notre objectif était de concevoir une application adaptable à divers facteurs de forme, allant des téléphones portables aux tablettes. Cette expérience nous a permis d'explorer les principes fondamentaux du développement Android moderne, tout en mettant en évidence les avantages de l'architecture MVVM et de la persistance des données avec Room. Ce rapport offre un aperçu succinct de notre démarche et de nos réalisations, fournissant des informations utiles pour les développeurs intéressés par la création d'applications Android robustes et polyvalentes.

2 Questions complémentaires

2.1 Quelle est la meilleure approche pour sauver, même après la fermeture de l'app, le choix de l'option de tri de la liste des notes ? Vous justifierez votre réponse et l'illustrez en présentant le code mettant en œuvre votre approche.

Durant l'exécution de l'application, la meilleure option est d'utiliser les ViewModels avec leur LiveData. Ainsi, s'il devait survenir un quelconque changement de l'état de la donnée, la vue serait immédiatement avertie.

Par contre lorsque l'application se ferme, les ViewModels et LiveData ne sont plus suffisants. Une des solutions que l'on peut utiliser serait de se servir des SharedPreferences. Elles permettent de stockes des données (de configuration) de manière persistante.

Pour faire cela, on a fait une classe qui est un manager de préférences partagées.

```kotlin
package com.example.daa_lab04

import android.content.Context
import android.content.SharedPreferences
import com.example.daa_lab04.viewModels.SortType

class SortPreferencesManager(context: Context) {

    private val sharedPreferences: SharedPreferences = context.
        getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE)

    fun saveSortType(sortType: SortType) {
        sharedPreferences.edit().putString(KEY_SORT_TYPE, sortType.name).
            apply()
    }

    fun getSortType(): SortType {
        val savedSortType = sharedPreferences.getString(KEY_SORT_TYPE,
            SortType.ETA.name)
        return SortType.valueOf(savedSortType ?: SortType.ETA.name)
    }

    companion object {
        private const val PREFS_NAME = "SortPreferences"
        private const val KEY_SORT_TYPE = "SortType"
    }
}
```

Il propose simplement des méthodes pour stocker et récupérer des préférences. Ensuite, on a modifié l'activité principale pour qu'à sa création, elle aille récupérer les préférences et que pour qu'à chaque changement de préférences, elle sauvegarde cette préférence.

```
class MainActivity : AppCompatActivity() {

    ...

    private val sortPreferencesManager by lazy {
        SortPreferencesManager(this)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        myViewModel.selectSortType(sortPreferencesManager.getSortType())
    }

    ...

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when(item.itemId) {

            ...

            R.id.eta -> {
                myViewModel.selectSortType(SortType.ETA)
                sortPreferencesManager.saveSortType(SortType.ETA)
                true
            }
            R.id.creation -> {
                myViewModel.selectSortType(SortType.CREATED)
                sortPreferencesManager.saveSortType(SortType.CREATED)
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }
}
```

2.2 L'accès à la liste des notes issues de la base de données Room se fait avec une LiveData. Est-ce que cette solution présente des limites ? Si oui, quelles sont-elles ? Voyez-vous une autre approche plus adaptée ?

L'utilisation de LiveData pour accéder à la liste des notes depuis la base de données Room peut entraîner des inefficacités. Les requêtes LiveData s'exécutent à chaque modification de la table correspondante, même lorsque les changements ne sont pas pertinents pour l'interface utilisateur, entraînant une surcharge. Afin de résoudre ces problèmes, des alternatives telles que l'utilisation de Flow, qui offre une gestion plus flexible des opérations asynchrones, et la bibliothèque Paging de Jetpack, spécialement conçue pour optimiser les performances avec de grandes quantités de données, sont préconisées. Ces solutions visent à améliorer l'efficacité des requêtes et à offrir une expérience utilisateur plus performante.

Il est également important de noter que la bibliothèque Paging peut charger dynamiquement des blocs de données, optimisant ainsi l'utilisation des ressources et améliorant la réactivité de l'application. En adoptant

ces approches alternatives, on peut surmonter les limitations liées à l'utilisation de LiveData dans le contexte spécifique des listes de notes de l'application.

2.3 Les notes affichées dans la RecyclerView ne sont pas sélectionnables ni cliquables. Comment procéderiez-vous si vous souhaitiez proposer une interface permettant de sélectionner une note pour l'éditer ?

On pourrait par exemple implémenter le mécanisme en modifiant le ViewHolder de la RecyclerView en y ajoutant un `SetOnClickListener`. Ainsi, il est possible de détecter quel élément a été sélectionné.

```
1     inner class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
2
3         init {
4             itemView.setOnClickListener {
5                 val position = adapterPosition
6                 if (position != RecyclerView.NO_POSITION) {
7                     Toast.makeText(itemView.context, "You clicked on item #$
                          {position + 1}", Toast.LENGTH_SHORT).show()
8                 }
9             }
10        }
11
12        ...
13    }
```

3 Choix d'implémentation

A priori, on a surtout suivi ce qui nous était indiqué dans la consigne. Il n'y avait pas beaucoup de marge de manoeuvre pour faire des folies en termes d'implémentation.

On a surtout mis un point d'honneur sur l'encapsulation en faisant bien attention que chaque classe fasse exactement et uniquement ce qu'elle devait faire. Typiquement en faisant bien la gestion des notes dans le fragment prévu à cet effet que ça soit pour le tri de sa liste avec la mise à jour des éléments de la liste, etc…

On a aussi essayé d'avoir une arborescence la plus propre possible en séparant bien dans des sous-packages les fichiers qui jouaient le même rôle au sein de l'application.

4 Conclusion

En conclusion, ce laboratoire a été l'occasion d'explorer en profondeur l'architecture MVVM et l'utilisation de Room dans le développement d'applications Android. Nous avons réussi à créer une application adaptable à divers facteurs de forme, démontrant ainsi notre compréhension des concepts fondamentaux du développement mobile moderne. Cette expérience renforce notre capacité à concevoir des applications Android performantes et flexibles.