Développement Android
Laboratoire n°4

Architecture *MVVM*, utilisation d'une base de données *Room* et d'un *RecyclerView*

15.11.2023

Introduction

Ce laboratoire consiste au développement d'une application *Android* basée sur une architecture *MVVM* et disposant d'une base de données *Room*. La réalisation de ce laboratoire illustrera également le design d'une application mobile s'adaptant à plusieurs *form factors*.

Manipulations

1. Mise en place

Pour ce laboratoire vous allez partir d'une application par défaut « empty view activity » à laquelle vous ajouterez et implémenterez les différents composants architecturaux permettant la réalisation de l'application décrite dans ce document.

Nous vous fournissons les éléments suivants :

- Le code des entités qui seront gérées par la base de données *Android Room*. Il s'agit de *Note* qui peuvent être associées (One-to-One facultatif) à un *Schedule* (la date à laquelle la tâche décrite dans la note doit être réalisée). Les notes sont composées de différents champs, en particulier un *Type* de note (todo, shopping, travail, famille, aucun) et un *State* (à faire, fait) qui sont représentés sous la forme d'énumérations, également fournies. La relation One-to-One facultative est décrite par la classe *NoteAndSchedule*. La classe Note fournie met à disposition des méthodes statiques permettant de générer des *Notes* et des *Schedules* avec des données aléatoires.
- Des icones, au format vectoriel, que vous pouvez utiliser pour illustrer les différents types de *Notes*.

2. Conception du squelette de l'*Activité* et des *Fragments*

Cette première partie concerne la conception et la mise en place du squelette de l'Activité et du/des Fragment(s) qu'elle hébergera. Il s'agit dans cette étape de mettre en place :

- La *MainActivity* et ses *layouts* (smartphone et tablette) ;
- Le *Menu* associé à l'*Activité* (tri par date de création, tri par date de réalisation prévue, création d'une *Note* (aléatoire) et suppression de toutes les *Notes* ;
- Le *Fragment* et son *layout* affichant la liste des *Notes* ;
- Le *Fragment* et son *layout* qui contiendra le compteur et deux boutons de contrôles (sur tablette uniquement, remplaçant les entrées du *Menu* permettant la création aléatoire d'une *Note* et la suppression de toutes les *Notes*).

Hint : les slides de présentation accompagnant le laboratoire présentent un exemple d'interface.

3. Mise en place de la base de données *Room*

Votre deuxième tâche sera de réaliser la mise en place des différents composants nécessaires pour la mise en œuvre d'une base de données basée sur la librairie *Android Room*. Il s'agit en particulier du *DAO*, des *Convertisseurs* (en particulier pour les dates), de la *Base de Données* et du *Repository*. Vous pouvez mettre en place une méthode populate() permettant de générer des données de tests lors de la création de la base de données, cela vous simplifiera le développement des étapes suivantes.

Vous pouvez surcharger l'*Application* afin qu'elle puisse créer et stocker une instance du repository. Vous n'oublierez pas d'ajouter le plugin *ksp* dans le fichier *Gradle* du projet :

```
plugins {
        id("org.jetbrains.kotlin.android") version "1.9.20" apply false
        id("com.google.devtools.ksp") version "1.9.20-1.0.14" apply false
}
```

Et de l'activer, ainsi que d'ajouter les dépendances nécessaires dans le fichier *Gradle* du module app :

```
plugins {
        […]
        id("com.google.devtools.ksp")
}

dependencies {
        // Room components
        implementation("androidx.room:room-runtime:2.6.0")
        implementation("androidx.room:room-ktx:2.6.0")
        ksp("androidx.room:room-compiler:2.6.0")
        androidTestImplementation("androidx.room:room-testing:2.6.0")
}
```

4. Conception du *ViewModel* et intégration à l'*Activité* et aux *Fragments*

Veuillez créer l'implémentation du *ViewModel* qui fera le lien entre le *Repository* de la *Base de données*, l'*Activité* et les deux *Fragments*. Votre *ViewModel* mettra à disposition au minimum les valeurs (*LiveData*) et méthodes suivantes :

- `val allNotes = repository.allNotes` *//: LiveData<List<NoteAndSchedule>>*
- `val countNotes = repository.countNotes` *//: LiveData<Int>*
- `fun generateANote() { /* création d'une Note aléatoire et insertion dans base de données */ }`
- `fun deleteAllNote() { /* suppression de toutes les Notes de la base de données */ }`

Vous ajouterez ensuite le code à votre *Activité* et à vos *Fragments* leur permettant de les lier à une instance du *ViewModel*. Et vous mettrez en place le code nécessaire permettant de relier les éléments de l'interface aux valeurs et méthodes du *ViewModel*.

Pour rappel, un *ViewModel* ne doit comporter aucune référence vers des éléments du cycle de vie d'Android : *Activités*, *Fragments*, *Vues*, *Application*, etc.

N'oubliez pas d'ajouter les dépendances nécessaires :

```
dependencies {
      // Lifecycle components
      implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")
      implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.6.2")
      implementation("androidx.lifecycle:lifecycle-common-java8:2.6.2")

      // ViewModels
      implementation("androidx.activity:activity-ktx:1.8.0")
      implementation("androidx.fragment:fragment-ktx:1.6.2")
}
```

5. Conception et implémentation de la *RecyclerView*

Le *Fragment* responsable de l'affichage de la liste des *Notes* accueillera une *RecyclerView* associée à un *RecyclerView.Adapter* permettant d'afficher les *Notes* présentes dans la base de données. Nous différencions les *Notes* « simples » et les *Notes* « associées à un *Schedule* », et votre *RecyclerView* utilisera deux *layouts* différents adaptés à ces deux cas possibles. Votre Adapter devra également implémenter le tri des *Notes* lors du choix correspondant dans le *Menu* de l'*Activité*.

Nous n'allons pas accorder une grande importance à l'interface de présentation d'une Note, mais nous souhaitons tout de même que tous les champs soient visibles. Vous pouvez par exemple utiliser des icônes différents en fonction du type, ou d'appliquer une teinte colorée à certains textes ou images en fonction de l'état.

Hint : les slides de présentation accompagnant le laboratoire présentent des exemples d'interface.

6. Questions complémentaires

Veuillez répondre aux 3 questions suivantes. Pour chacune d'entre elles, vous développerez votre réponse et l'illustrerez, si demandé, par des extraits de code.

6.1 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.

6.2 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 ?

6.3 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 ?

Durée / Evaluation

- 6 périodes
- A rendre le mardi **05.12.2023** à **23h55** au plus tard.
- Pour rendre votre code, nous vous demandons de bien vouloir zipper votre projet Android Studio en veillant à bien supprimer les dossiers build (à la racine et dans app/) pour limiter la taille du rendu. Vous remettrez également un document **pdf** comportant les explications sur l'implémentation de votre solution ainsi que les réponses aux questions posées.
- Merci de rendre votre travail sur *CyberLearn* dans un zip unique. N'oubliez pas d'indiquer vos noms dans le code, sur vos réponses et de commenter vos solutions.