Webinaire Android

Mettez en place votre architecture MVVM

Bien comprendre l'architecture proposée par Google sur Android : le MVVM (et ses outils : ViewModel, LiveData)

Description du webinaire : Initiation au MVVM Android, LiveData, repositories, ViewModel. Partage de données entre différents fragments

Exercice 1

🎯**Compétences visées :**

- **Architecture MVVM**
- **Utiliser un ViewModel**
- **Partager des données entre différents fragments**
- **Utiliser les livedatas**
- **Utiliser un repository**

Projets et paths concernés :

- DA Android projets 4, 5, 7, 9

Énoncé

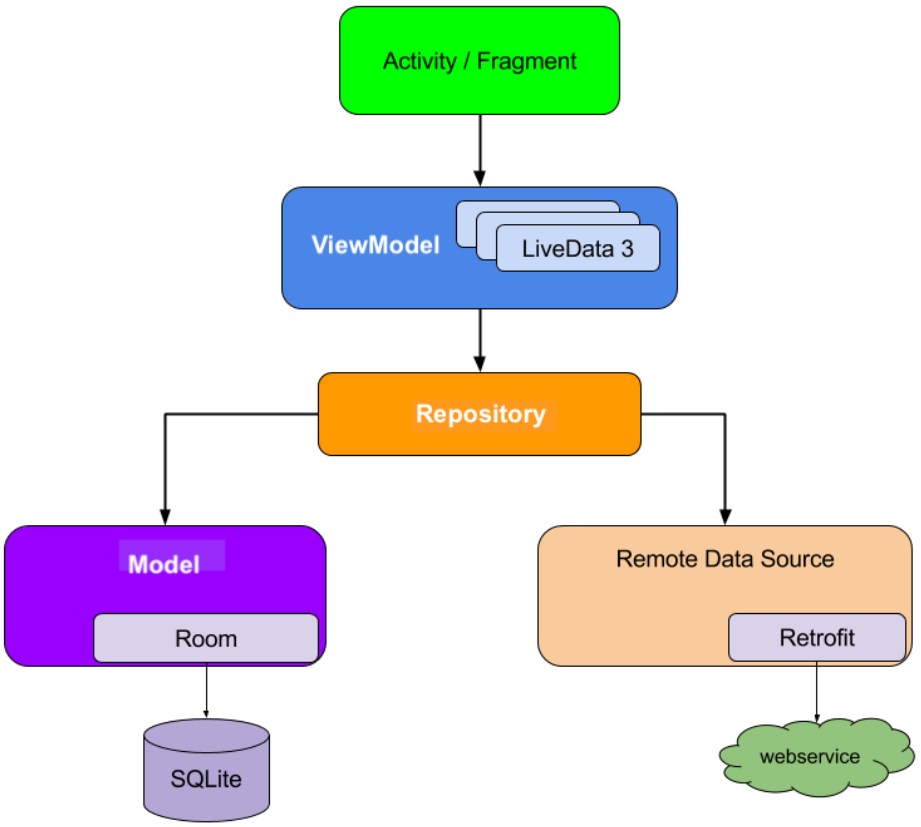
Vous êtes employé en tant que développeur d'application Android au sein de l'entreprise Objectivisor.
Suite à la dernière conférence de Google sur l'architecture MVVM, votre chargé de projet vous demande de refactoriser l'application existante en suivant les nouvelles pratiques proposées par Google.
Chaque appel API de l'application est facturé, et la nouvelle architecture permettrait de diviser par 5 le coût actuel de l'API en la sollicitant 5 fois moins.
Cela représenterait un gain d'argent et de temps indéniable.

Télécharger le code ici

L'architecture actuellement en place est un MVC classique, avec EventBus qui s'occupe de la communication entre les différents fragments. L'application comporte 1 activité et 3 fragments qui partagent une même liste de mails.

Chaque fragment appelle une ou plusieurs fois l'API pour récupérer les données, le but ici va donc être de refactoriser ces appels, afin que l'API soit appelée une seule et unique fois dans la MainActivity, que le résultat soit mis dans un livedata dans un ViewModel, et que les fragments observent ce livedata pour récupérer les données et se mettre à jour.

L'architecture à mettre en place ressemblera à ça :



(image prise sur le site officiel de Google)

Le repository est déjà (partiellement) codé mais pas utilisé, vous pouvez le modifier pour l'adapter aux besoins.

1 - Créez le ViewModel

Vous devez créer le fichier ViewModel et y placer une MutableLiveData de type ArrayList de mails, afin de stocker les données dont vous aurez besoin. Il faut ensuite ajouter le setter et le getter. Si vous voulez, vous pouvez passer par une factory, et/ou des injections, peu importe pour cet exercice.

2 - Faites appel à l'API depuis la MainActivity

Vous devez faire le lien entre votre ViewModel et votre activité, afin que celle-ci puisse appeler les méthodes du ViewModel.
Une fois ceci fait, il faut créer une méthode dans le ViewModel, qui demandera au repository (qui, lui, demandera à l'API) les données des mails, pour les stocker dans le MutableLiveData.

3 - Observez les données depuis les fragments

Une fois ceci fait, vous allez devoir lier le ViewModel aux fragments. L'owner du ViewModel reste l'activité, il faut faire attention de bien rester sur la même instance du ViewModel.
Les fragments peuvent donc ensuite utiliser le get mis en place dans le 1) afin de pouvoir créer un observable sur le MutableLiveData.

4 - Refactor les subscribes EventBus

EventBus était utilisé pour mettre à jour la UI dans les subscribes, vous devez donc adapter ce code à la nouvelle architecture.

5 - Terminez le refactor des fonctionnalités

Il faut que l'application fonctionne comme avant mais avec la nouvelle architecture ; il faut donc revoir l'ajout et la suppression de mail. EventBus peut être totalement supprimé du projet, bien que ce ne soit pas obligatoire.

Prêt à faire économiser de l'argent à Objectivisor grâce à l'architecture MVVM et au partage des données de vos fragments ? À vous de jouer !

Livrable

- Le code de votre application refactorisé.

Corrigé type

Le code final en architecture MVVM est disponible sur ce repo (branche finale du repo).

Pour aller plus loin

- Utilisation d'une ViewModel Factory : cours *Gérez vos données localement pour avoir une application 100 % hors ligne*
- Utilisation de Dagger pour injecter vos dépendances
- Accéder à des données externes grâce à Retrofit
- Utiliser le modèle MVVM avec Room (utilisez le cours *Gérez vos données localement pour avoir une application 100 % hors ligne*)
- Voir l'architecture MVI (MVVM un peu plus poussé avec gestion des states)
- Voir le data binding