Listes et fragments - Introduction à l'architecture MVVM

L'architecture MVVM (Model-View-ViewModel) est une approche recommandée pour construire des applications Android robustes et maintenables. Elle sépare les responsabilités en trois principales composantes:

- Le *Repository*, qui fournit et gère les données. Il représente souvent la source de données, qu'elle soit locale ou distante.
- Le *ViewModel*, qui sert d'interface entre le Repository et l'UI. Il prépare et fournit les données pour l'affichage.
- La *Vue*, qui est responsable de l'affichage de l'interface utilisateur et de la collecte des interactions utilisateur.

Dans cette architecture, nous utilisons également la classe *LiveData*, une enveloppe de données qui est conçue pour informer la Vue de tout changement de données, garantissant ainsi que l'UI est toujours à jour.

Pour cet exercice, nous construirons une petite application de chat. L'interface utilisateur sera composée de deux éléments principaux:

- La liste des messages enregistrés, montrant l'auteur et le début du message.
- Le détail d'un message, présentant le contenu complet d'un message sélectionné.

1. Récupérer le squelette d'app dans l'archive *ChatM1.zip*. Il contient un ensemble de classes à compléter dans le cadre de cet exercice.
2. La classe Message est destinée à représenter un message. Elle doit comporter des méthodes publiques permettant d'accéder à son contenu. Terminer son implémentation s'il y a lieu.
3. La classe MessagesRepo est un singleton représentant le dépôt (ou "repository") de données pour notre application. Le but de ce dépôt est de centraliser la gestion des données et de fournir une source unique pour les messages (principe de la "Single source of truth"). Implémenter les méthodes marquées TODO :
   - ajouteMessage(...) créant un nouveau message et l'ajoutant en fin de liste ;
   - get(int) renvoyant un message à partir de sa position ;
   - deleteMessageFromID(int) permettant d'effacer le message dont l'ID est passé en paramètre, et retournant l'index de la position qu'occupait ce message ;
   - size() retournant le nombre de messages stockés.

   NB : le constructeur de cette classe contient déjà quelques lignes permettant d'ajouter automatiquement quelques exemples de messages dans tout ensemble que l'on créé. Ne pas retirer ces lignes pour l'instant (elles serviront de test).
4. La classe MessageViewModel doit présenter les données prêtes à être affichées à l'UI. C'est elle qui a la notion de message sélectionné (c'est une considération qui ne concerne pas le repository, qui se contente de stocker les messages).
   - Implémenter les méthodes ajouteMessage et supprimeMessage, qui auront un effet sur le repository ;
   - Implémenter la méthode selectMessage définissant le message sélectionné. Cette méthode ne devrait pas toucher au repository.

5. exécuter le programme. Les messages de tests devraient s'afficher dans la partie haute de l'écran (leur contenu étant cependant tronqué dans cette liste)

NB : le squelette d'app fourni comprend deux fragments : celui affiché en haut s'occupe de l'affichage des messages, tandis que celui du bas aura pour fonction de gérer l'affichage complet d'un message sélectionné. Il sera câblé juste après.

Affichage complet d'un message.

L'appui sur un élément de la liste ne provoque pour le moment qu'un affichage complet du message dans le Logcat et la mise à jour de l'attribut selectedMessage du ViewModel. On veut pouvoir afficher le message sélectionné à l'écran dans le fragment correspondant.

1. Dans MessageFragmentList, implémenter la méthode update(Message) de manière à afficher le message passé en paramètre.

2. Compléter la méthode onViewCreated de manière à observer l'attribut selectedMessage du ViewModel. On appellera update à chaque mise à jour du message sélectionné.

Ajout de message

1. Ajouter dans MainActivity un bouton "Ajouter" qui ajoute un message (pour le moment défini en dur dans l'app) via le ViewModel. L'affichage devrait se mettre directement à jour.

2. Créer une activity permettant à l'utilisateur de saisir un nouveau message. Cette activité comprendra deux zones EditText (une pour le titre, l'autre pour le contenu) et un bouton Valider. L'appui sur ce bouton devra déclencher l'ajout d message (NB : cette activity aura besoin d'un MessageViewModel également. On ne touche pas au repository directement depuis l'Activity !

   Faire en sorte que l'appui sur le bouton *Creer* défini dans MessagesFragment appelle désormais cette activité.

Finitions

1. Ajouter dans MainActivity un bouton "Supprimer" qui supprime le message sélectionné (toujours en passant par le ViewModel).

2. faire en sorte que l'affichage des deux fragments se fasse cote à cote (et non plus l'un en dessous de l'autre) lorsque l'appareil est tourné en mode paysage. (c'est possible en ne touchant qu'au répertoire *res*)