TP2 : Persistence des données dans une base SQLite

Hiver 2023

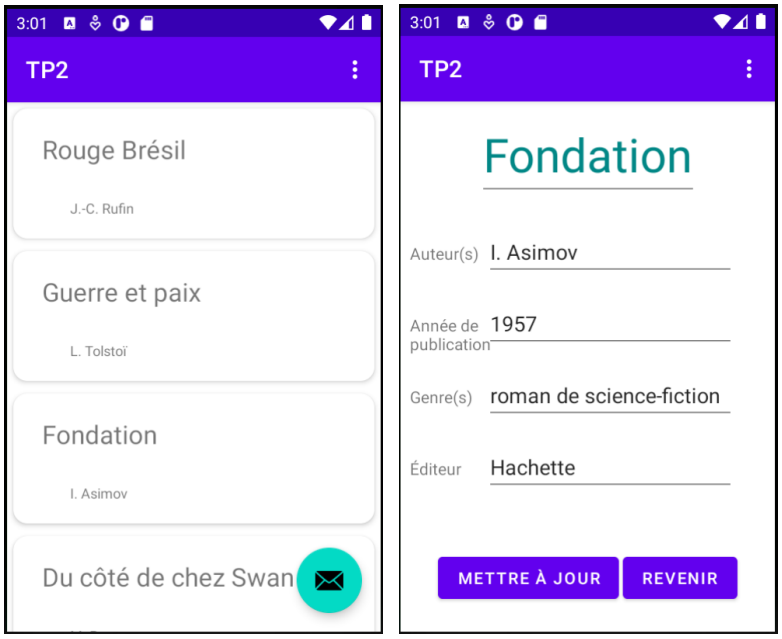
Objectifs

Le but est de développer une application Android permettant de sauvegarder les informations saisies dans une base de données. Dans ce sujet, prévu pour 6h en séances, il est demandé de partir d'une application similaire au sujet précédent et d'y inclure une base de données pour y sauvegarder de manière persistente les informations saisies.

L'application demandée pour ce TP permet de gérer des informations sur des livres d'une bibliothèque personnelle. Pour accélérer le développement, le code gérant l'affichage est fourni. Dans cette version préliminaire, les informations sont stockées dans un tableau statique.

1 Utilisation de la bibliothèque Room

Récupérez l'archive *TP2.zip* sur le site du cours et ouvrez le projet avec Android Studio. Il est probable que vous ayez à mettre à jour les scripts Gradle en fonction de votre environnement de développement. Une fois compilée, l'application doit afficher les deux écrans de la figure 1. Le premier écran montre la liste des livres gérée avec une RecyclerView, le second affiche les informations détaillées sur un livre sélectionné, aucune action n'étant encore associée au bouton « Mettre à jour ».



(a) Écran 1 (b) Écran 2

FIGURE 1 – Les deux écrans de l'application

Dans cette première partie, l'objectif est de remplacer le tableau statique books contenue dans la classe Book par une base de données SQLite gérée par la bibliothèque Room.

Ajoutez les lignes suivantes dans le script Gradle de l'app :

```
def room_version = "2.2.6"
implementation "androidx.room:room-runtime:$room_version"
annotationProcessor "androidx.room:room-compiler:$room_version"
```

Déclarez la classe Book comme étant une entité de la table « books ». Ajoutez un attribut id de type long et déclarez-le comme clé primaire de la table générée dans la base de données.

Créez une interface Java DAO BookDao. Définissez-y les cinq méthodes suivantes :

— updateBook(Book book) pour mettre à jour les informations d'un livre existant déjà dans la base de données,

— insertBook(Book book) pour insérer un nouveau livre dans la base de données,

— getBook(long id) pour obtenir un livre suivant son identifiant,

— deleteBook(long id) pour supprimer un livre,

— getAllBooks() retournant la liste de toutes les livres contenus dans la base.

Créez une classe `BookRoomDatabase` permettant de créer une instance de la base données gérée par la bibliothèque Room en utilisant le patron de conception singleton. Vous pourrez également définir dans cette classe l'ExecutorService permettant d'effectuer les opérations sur la base de données dans des *threads* séparés de celui gérant l'interface utilisateur.

Créez un dépôt BookRepository pour faire l'interface entre le modèle de vue et la base de données. Dans cette classe définissez une méthode d'accès pour chaque méthode définie au niveau du DAO. Vous aurez également besoin de définir deux attributs :

— Une liste de livres allBooks de type LiveData correspondant à la liste affichée,

— Un livre selectedBook de type MutableLiveData associé aux informations affichées sur le second écran.

Pour tester, vous pourrez charger dans la base les livres de du tableau statique de Book. Dans la classe BookRoomDatabase, ajoutez le code suivant :

```
1  private static RoomDatabase.Callback sRoomDatabaseCallback = new RoomDatabase.Callback() {
2    @Override
3    public void onCreate(@NonNull SupportSQLiteDatabase db) {
4      super.onCreate(db);
5
6       databaseWriteExecutor.execute(() -> {
7        // Populate the database in the background.
8        BookDao dao = INSTANCE.bookDao();
9        dao.deleteAllBooks();
10
11       for(Book newBook : Book.books)
12         dao.insertBook(newBook);
13
14     });
15   }
16 };
```

Vous pouvez faire appel à ce *thread*, juste avant l'appel à build :

```
1    INSTANCE = Room.databaseBuilder(...)
2                    .addCallback(sRoomDatabaseCallback)
3                    .build();
```

2 Définition des modèles de vue

Maintenant que les bases de données et les méthodes d'accès ont été définis, l'objectif dorénavant est de faire le lien entre ces données et l'affichage, en recourant à l'architecture MVVM.

Créez ListViewModel, le modèle de vue du premier fragment ListFragment. Dans cette classe, définissez deux attributs : un permettant de garder le lien vers le dépôt de la base de données et un autre de type LiveData<List<Book> > pour la liste des livres à afficher.

Dans ListFragment, au niveau d'onViewCreated, associez le fragment à son modèle de vue. Fixez l'adaptateur de la RecyclerView comme observateur de l'attribut LiveData<List<Book> > du modèle de vue.

Dans RecyclerAdapter, définissez une méthode setBookList pour que l'adaptateur conserve un lien vers la liste des livres et soit notifié des modifications. Modifiez onBindViewModel et getItemCount de façon à ce que l'adaptateur utilise la base données et non plus le tableau statique défini dans Book. Au niveau de la classe interne ViewHolder, modifiez le setOnClickListener afin que ce soit l'ID du livre sélectionné et non la position dans la liste qui soit envoyé au second fragment :

```
1  long id = RecyclerAdapter.this.bookList.get((int)getAdapterPosition()).getId();
```

Définissez un modèle de vue DetailViewModel pour DetailFragment. De manière similaire à ce qui a été fait pour le premier fragment, définissez deux attributs : un permettant de garder le lien vers le dépôt de la base de données et un autre de type MutableLiveData<Book> pour le livre à afficher.

Au niveau d'onViewCreated de DetailFragment, associez le fragment au modèle de vue. Fixez les champs du formulaire comme observateurs de l'objet Book sélectionné. Définissez le traitement nécessaire pour les clics sur le bouton « Mettre à jour ».

3 Ajout d'un nouveau livre

Jusqu'à présent, seule la modification d'un livre existant est faisable, il n'est pas encore possible d'en créer un nouveau.

Afin de remplacer l'enveloppe qui apparait dans le bouton flottant en bas à droite du premier écran, par un « + », il va valloir changer l'icône affichée dans le fichier d'agencement fragment_list.xml. Pour ce faire, remplacez le champ `@android:drawable/ic_dialog_email` par `@android:drawable/ic_input_add` au niveau du `FloatingActionButton`.

L'appui sur ce bouton doit faire apparaitre un écran très similaire au second écran montrant les informations sur un livre de la base de données, la seule différence étant qu'aucun champ textuel n'est encore rempli. Pour ne pas dupliquer le code, il est demandé de réutiliser DetailFragment. Ajoutez une action sur le bouton, au niveau de la classe ListFragment, pour que le navigateur affiche le second fragment. Vous pourrez utiliser la constante -1 comme ID du livre encore vide, à passer comme argument.

Au niveau du modèle de vue DetailViewModel renommez la méthode updateBook en insertOrUpdateBook. Suivant que l'attribut MutableLiveData<Book> soit nul ou non, ou bien ait un ID qui soit différent ou non de -1, vous pourrez faire appel à la méthode d'insertion ou de mise à jour du dépôt.

Faites une vérification sur le remplissage des champs lors de l'appui sur le bouton « Mettre à jour » pour l'utilisateur, un nouveau livre ne devant être ajouté que lorsqu'au moins le titre et les auteurs ont été précisés. Le cas échéant, vous pourrez afficher un message d'erreur au moyen d'une SnackBar.

4 Suppression d'un livre

Pour pouvoir supprimer un livre, un menu contextuel similaire à celui affiché à la figure 2 devra être affiché lors de la pression prolongée sur un item de la liste. Il vous faudra créer un fichier d'agencement (context_menu.xml) de type menu pour afficher l'item « Supprimer ».

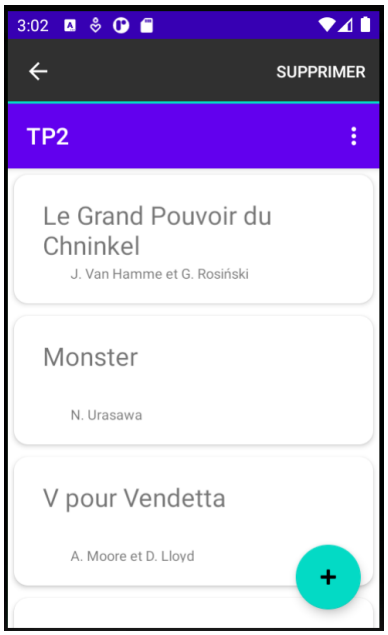


FIGURE 2 – Menu contextuel pour la suppression d'un livre, apparaissant en haut de l'écran

L'affichage du menu contextuel et le traitement de l'action associée devront être réalisés dans le ViewHolder de l'adaptateur de la RecyclerView.

5 Rendu

Ce TP est à faire de manière individuelle. Il vous est demandé de déposer votre projet Android sur Github. Pour cela, veuillez faire un `fork` du projet suivant `https://github.com/CERI-L3-20222023-ApplicationsMobiles/TP2` en respectant votre groupe. Sur ce dépôt, vous déposerez les sources de votre projet, ainsi qu'un `apk` prêt à être déployé sur un smartphone et un rapport au format pdf montrant les écrans de votre application et expliquant ce qui est fonctionnel ou non dans votre application. Assurez-vous de faire apparaître votre nom et prénom dans le fichier README afin d'être facilement identifiable par votre enseignant ou enseignante.

Remarque : Ce TP est noté. Vous devez le rendre dans un délai de 6 jours après la deuxième séance qui lui est consacrée.