Compte rendu Séquence 3:
Application ToDoList

Introduction

L'objectif de cette séquence 3 est de modifier l'application de ToDoList, afin qu'elle affiche les données même lorsque la connexion au réseau n'est pas accessible, à partir d'informations stockées dans une base de données. Cela implique de modifier les activités principales, les classes Modèle pour qu'elles soient comprises par la base de données et de créer des classes nécessaires à la gestion de Room.

Analyse

Description de la structure

La structure est similaire à celle de la séquence précédente, mais présente en plus un dossier **Room** qui regroupe les deux interfaces **ItemDao** et **ListeDao**, permettant de définir les requêtes aux différentes tables, ainsi qu'une classe abstraite **AppDatabase**.

Soit dans le dossier **Activities** :

- **MainActivity**
- **CheckListActivity**
- **ShowListActivity**
- **SettingsActivity**
- **GenericActivity** (comprend l'interface ToDoApiInterface)

Dans le dossier **Modele** :

- **ProfilListeToDo**
- **ListeToDo** (ajout de @Entity et de la clé primaire)
- **ItemToDo** (ajout de @Entity et de la clé primaire)

Dans le dossier **RecyclerView** (inchangé) :

- **ListAdapter** (comprend la classe ListViewHolder et l'interface ActionListenerListe)
- **ItemAdapter** (comprend la ItemViewHolder et l'interface ActionListenerItem)

Dans le dossier **Retrofit** (inchangé) :

- **ReponseDeBase**
- **ReponseHash**
- **ReponseList**
- **ReponseLists**
- **ReponseItem**
- **ReponseItems**

- **ReponseUsers**

Dans le dossier **Room** :

- **ItemDao**
- **ListeDao**
- **AppDatabase** (extends RoomDatabase)

Points particuliers

1. Modification de la classe Modèle

Afin que les classes ListeToDo et ItemToDo contenues dans le package Modele soient lisibles par une base de données, on ajoute une annotation @Entity au début de chaque classe. On indique également la clé primaire de la table à l'aide de l'annotation @PrimaryKey. Enfin on nomme les autres colonnes du tableau à l'aide de l'annotation @ColumnInfo.

Exemple pour ItemToDo :

```
@Entity
public class ItemToDo implements Serializable {
```

```
@SerializedName("label")
@ColumnInfo(name = "label")
private String description;

@SerializedName("checked")
@ColumnInfo(name = "checked")
private int fait;

@SerializedName("idItem")
@PrimaryKey
private int idItem;

@ColumnInfo(name = "listeId")
private int listeId;
```

2. Enregistrement de la base de données

A chaque fois que l'application s'ouvre avec le réseau activé, elle met à jour la base de données à partir des informations contenues dans l'API.
Au moment où on ajoute la liste trouvée au RecyclerView, on ajoute également la liste à la base de données dans un nouveau Thread.

```
public void affichageListes(String hash) {
    new Thread(){
        public void run() {
            database.listeDao().clean();
        }
    }.start();
    toDoInterface.recupLists(hash).enqueue(new Callback<ReponseLists>() {
        @Override
        public void onResponse(Call<ReponseLists> call, Response<ReponseLists> response) {
            if (response.isSuccessful() && response.body().success) {
                List<ListeToDo> allLists = response.body().lists;
                for (int i=0 ; i<allLists.size() ; i++) {
                    final ListeToDo list_i = allLists.get(i);
                    listesUserCourant.add(allLists.get(i));
                    new Thread(){
                        public void run() {
                            database.listeDao().createListe(list_i);
                        }
                    }.start();
                }
                RecyclerView recyclerView = findViewById(R.id.recyclerView);
                recyclerView.setLayoutManager(new LinearLayoutManager( context: CheckListActivity.this));
                adapterEnCours = new ListAdapter(listesUserCourant,  actionListener: CheckListActivity.this);
                recyclerView.setAdapter(adapterEnCours);
                recyclerView.addItemDecoration(new DividerItemDecoration( context: CheckListActivity.this, LinearLayout.VERTICAL));

            }
        }
        @Override
        public void onFailure(Call<ReponseLists> call, Throwable t) {
            alerter( s: "Il y a un problème : affichage listes");
        }
    });
}
```

3. Récupération de la base de données

Si le réseau n'est pas activé lors de l'ouverture de l'application, l'utilisateur peut se connecter via un bouton spécial « Connexion Hors Wifi ». Dans ce cas, les listes et les items affichés sont ceux récupérés via la base de données. Dans ce cas, la fonctionnalité d'ajout et de suppression des listes / items n'est pas activée.

```
if (reseau) {
    affichageListes(hash);

    // Lors du clic sur le bouton d'ajout
    btn_ajout.setOnClickListener((v) → {
            String label = edt_liste.getText().toString();
            ajouterListe(hash, label);
    });
}

else {
    affichageListesViaSQL();
    btn_ajout.setEnabled(false);
}
```

```
public void affichageListesViaSQL() {
    new Thread(){
        public void run() {
            List<ListeToDo> listeUser = database.listeDao().getAll();
            for (ListeToDo list : listeUser) {
                listesUserCourant.add(list);
            }
        }
    }.start();
    RecyclerView recyclerView = findViewById(R.id.recyclerView);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: CheckListActivity.this));
    adapterEnCours = new ListAdapter(listesUserCourant,  actionListener: CheckListActivity.this);
    recyclerView.setAdapter(adapterEnCours);
    recyclerView.addItemDecoration(new DividerItemDecoration( context: CheckListActivity.this, LinearLayout.VERTICAL));
}
```

Bibliographie

Room :

- *https://developer.android.com/training/data-storage/room?fbclid=IwAR3NIwWvdhsADbs_iwWurP-m3K_yeHjM1d9SUdF8jkFApDCDu7Y2WdCIodY#java*
- *https://developer.android.com/jetpack/androidx/releases/room*