TP4 - Stockage

Télécharger PDF

[../tp3.pdf]

Objectifs du TP

Familiariser les étudiants avec les différentes techniques de stockage dans Android : Shared Preferences et base de données interne Realm.

Outils et Versions

- Android Studio [https://developer.android.com/studio/index.html#tos-header] Version: 3.0.1
- Android : Version 5.0.1 (API 21)
- Realm : Version 5.0.0

I. Stockage dans Android

Il existe plusieurs options de stockage des données persistantes dans Android.
Le choix de la solution idéale dépendra des besoins spécifiques des utilisateurs :
- Les données sont-elles privées ou accessibles à d'autres applications ?
- Combien d'espace disponible est-il requis ?
Les données doivent-elles être structurées, semi-structurées ou pas structurées ?
Les options de stockage que nous allons montrer dans ce TP sont les suivantes :

1. Shared Preferences :

Framework général qui permet de sauvegarder et extraire des paires clef-valeur persistantes de type primitif. Il permet de stocker des données de types booléen, réel, entier, long et string.
Typiquement, les Shared Preferences sont utilisées pour sauvegarder les préférences utilisateur, tel que la sonnerie choisie, par exemple.

2. Stockage interne

Il possible d'utiliser le stockage interne du téléphone pour sauvegarder des fichiers. Par défaut, ces fichiers sont privés (inaccessibles à partir d'autres applications). Quand l'application est supprimée, ces fichiers le sont aussi.

3. Stockage externe

Tous les appareils compatibles Android supportent un espace de stockage externe, qui peut être une SD card, ou un espace interne non-amovible.
Les fichiers sauvegardés dans un espace de stockage externe sont accessibles à toutes les applications en lecture.

Ils peuvent être modifiés par l'utilisateur si le « USB mass storage » est activé pour transférer les fichiers sur un ordinateur.

4. Base de données SQLite

Android fournit un support total du SGBD SQLite. SQLite est une bibliothèque logicielle qui implémente un moteur de base de données SQL avec zéro-configuration, léger et sans dépendances externes.

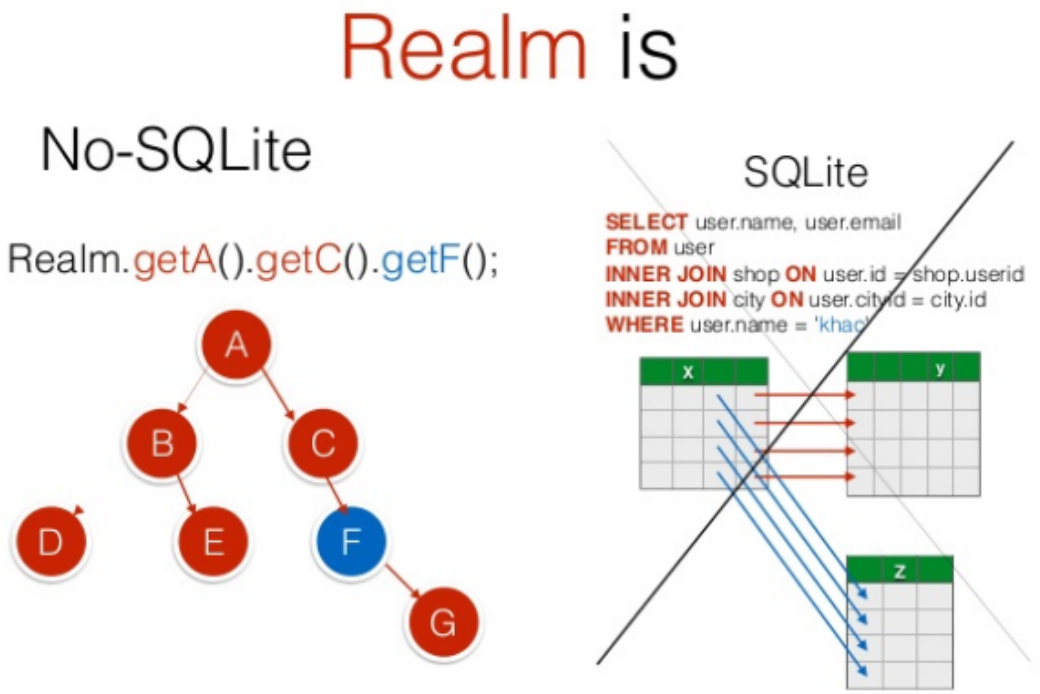
Toutes les bases de données créées dans une application seront accessibles par nom depuis tout emplacement de cette application, mais pas de l'extérieur.
La méthode recommandée pour la création d'une base de données SQLite d'utiliser une sous-classe de SQLiteOpenHelper, une classe d'assistance qui aide l'utilisateur à créer et manipuler sa base de données de manière simple.

5 . Base de données Realm

Realm est une solution de base de données multiplateforme qui peut être utilisée comme alternative à SQLite. Bien que Realm est une base de données orienté objet, elle a quelques différences avec les autres bases de données. Realm est plus facile à configurer et à utiliser. Pour effectuer la même opération dans Realm, vous pouvez écrire moins de code qu'avec SQLite. En ce qui concerne les performances, Realm est dit être plus rapide et offre également d'autres fonctionnalités modernes telles que le cryptage, le support JSON et les notifications de changement de données.

- Avantages du Realm par rapport à SQLite :
    - plus rapide que SQLite (jusqu'à 10 fois plus rapide que SQLite pour les opérations usuelles (Read, Insert, Delete...)
    - Facile à utiliser : Pas besoin d'écrire des requêtes SQL
    - Gère la conversion d'objet
    - " Safe Threading" afin que vous puissiez accéder aux mêmes données en même temps à partir de plusieurs threads
    - Sécurise vos données avec un cryptage et un décryptage transparents
    - Ajoute les objets json directement.

- Possibilité de créer votre base de données locale, puis de la connecter à Realm Mobile Platform pour synchroniser vos données sur tous les appareils connectés



Dans Ce TP, nous allons voir plus en détails la base de données Realm.

II. Exercice 1 : Shared Preferences

1. Objectif

Durée estimée de l'exercice : au plus 0h15
L'objectif de cette première partie est de réaliser une application simple pour stocker quelques données dans les Shared Preferences. L'interface sera alors comme suit :

2. Réalisation

Pour lire un objet SharedPreferences, utiliser l'une de ces méthodes:
getSharedPreferences: pour utiliser plusieurs fichiers de préférences identifiés par nom

getPreferences: pour utiliser un seul fichier de préférences, donc sans définir un nom de fichier
Pour ajouter des valeurs :
- Appeler *edit()* pour obtenir un objet SharedPreferences.Editor
- Ajouter des valeurs avec des méthodes, tel que *putBoolean()* et *putString()*
- Valider les nouvelles valeurs avec *commit()*
Pour lire des valeurs, utiliser les méthodes de SharedPreferences tel que *getBoolean()* ou *getString()*…

Le code obtenu devra ressembler à ce qui suit :

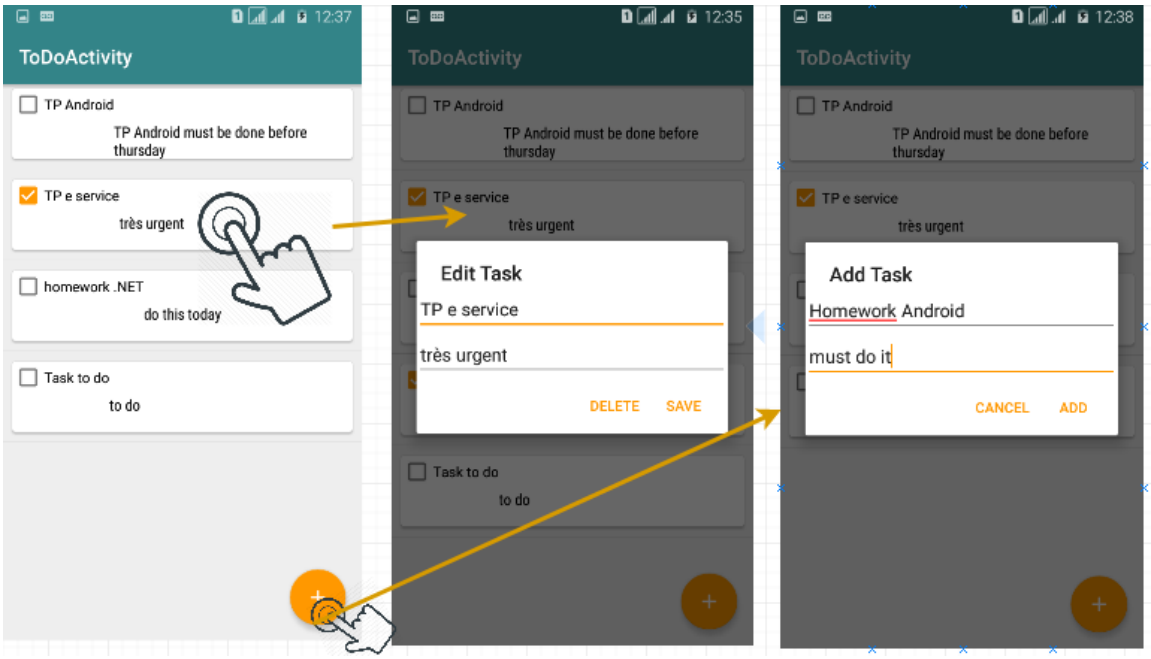
```
public void save (View view)
   {
       SharedPreferences.Editor editor = getSharedPreferences(My_PREF
       editor.putString("name",editName.getText().toString());
       editor.putInt("age", Integer.valueOf(editAge.getText().toStrir
       editor.commit() ;


       SharedPreferences preferences = getSharedPreferences(My_PREF_N
       String name = preferences.getString("name","no name defined")
       int age = preferences.getInt("age",0) ;
       textName.setText(name);
       textAge.setText(age+"");


   }
```

Activité 1
Créer un projet Android TP4-P1, permettant d'enregistrer et de lire des données dans les Shared Preferences.

II. Exercice 2 : base de données Realm

1. Objectif

l'objectif de cette partie est d'apprendre à configurer une base de données Realm sous Android et de réaliser une application simple permettant la manipulation des données.
Le résultat de notre application sera comme suit :



2. Réalisation

1. Configuration de Realm

Pour configurer Realm dans votre projet suivez les étapes suivantes :

1- Pour ajouter la bibliothèque Realm au projet, ajoutez d'abord la dépendance classpath au fichier build.gradle au niveau du projet.

```
classpath "io.realm:realm-gradle-plugin:5.0.0"
```

2- Ensuite, appliquez le plugin realm-android en haut du fichier build.gradle au niveau de l'application.

```
apply plugin: 'realm-android'
```

Synchronisez ensuite les fichiers Gradle du projet.

3- Avant de pouvoir utiliser Realm dans votre application, vous devez l'initialiser. Realms sont l'équivalent d'une base de données. Ils correspondent à un fichier sur le disque et contiennent différents types d'objets. L'initialisation du Realm est effectuée une fois dans le cycle de vie de l'application. Un bon endroit pour le faire est dans une sous-classe de la classe Application.

Créez une classe nommée App qui étend de la classe Application et modifiez-la comme indiqué.

```
@Override
  public void onCreate() {
      super.onCreate();
      Realm.init(this);
      RealmConfiguration realmConfig = new RealmConfiguration.
               Builder()
              .schemaVersion(0)
              .build();
      Realm.setDefaultConfiguration(realmConfig);
  }
```

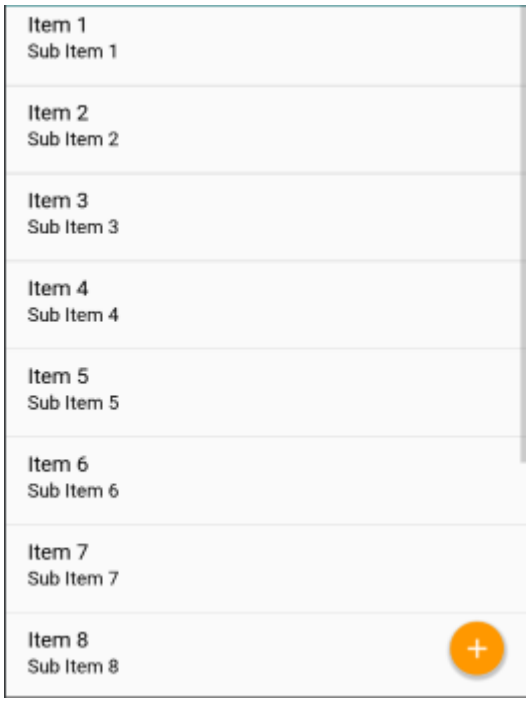
Dans ce qui précède, nous initialisons d'abord Realm, puis le configurons avec un objet RealmConfiguration. RealmConfiguration contrôle tous les aspects de la création de Realm. La configuration minimale utilisable par Realm est *RealmConfiguration config = new RealmConfiguration.Builder (). Build ();* qui créera un fichier appelé *default.realm* situé dans *Context.getFilesDir ()*.
Dans notre configurationn, nous définissons également un numéro de version.

4- 4. Dans le fichier Manifest, définissez cette classe comme le nom de la balise application.

```
   android:name="App"
```

2.Création et affichage de la ListView

1- Créez une Activity nommée TaskToDoActivity et y définnez une listView et un FloatingActionButton comme illustré dans la figure suivant :



2- Ajoutez un fichier de disposition nommé task_row.xml au dossier layout qui spécifiera le format de chaque ligne du ListView précédemment ajouté. Modifiez-le pour avoir le résultat suivante :

3- Dans votre package principal, ajoutez un package nommé models et créez une classe nommée Task dans ce package. Ajoutez ce qui suit au fichier.

```
public class Task extends RealmObject {

    @Required
    @PrimaryKey
    private String id;
    @Required
    private String name;
    private boolean done;

    private String description ;

    public String getDescription() {
        return description;
    }
```

```
    public void setDescription(String description) {
        this.description = description;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }
}
```

Dans ce qui précède, nous créons notre modèle Realm en étendant la classe de base RealmObject. Realm supporte les types de champs suivants: booleen, int, long, float, double, String, Date. Ainsi que les types Boolean, Byte, Short, Integer, Long, Float et Double. Sous-classes de RealmObject et RealmList <? extends RealmObject> sont utilisés pour modéliser les relations. (relation 1-1, relation 1-n, etc)

Chaque tâche dans notre application aura un nom, une description et un champ booléen qui indiquera si la tâche a été complétée ou non et un identifiant unique qui sera utilisé pour identifier la tâche.

Les champs id et name sont marqués d'une annotation **@Required**. Ceci est utilisé pour indiquer à Realm d'appliquer des contrôles sur ces champs et

d'interdire les valeurs nulles. Le champ id est annoté avec **@PrimaryKey** celà veut dire que ce champs est un clé primaire.

4- POur mettre la listview et la lier aux données on a besoin d'un adaptateur. Lorsque vous stockez vos données dans Realm, **RealmBaseAdapter** et **RealmRecyclerViewAdapter** peuvent être utilisés pour configurer rapidement votre liste et lier automatiquement vos données à votre vue. Pour se faire, vous devez ajouter la dépendance suivante dans le fichier gradle.

```
“ io.realm:android-adapters:2.0.0 “
```

Créez ensuite votre adaptateur et modifiez le comme suit :

```
public class ToDoAdapter extends RealmBaseAdapter<Task> implements Li

    ToDoActivity activity ;
    List<Task> tasks ;

    public ToDoAdapter(ToDoActivity activity,@Nullable OrderedRealmCo
        super(data);
        this.activity = activity ;

    }



    private static class ViewHolder {
        TextView taskName, taskDescription ;
        CheckBox isTaskDone;
    }


    @Override
    public View getView(int position, View convertView, ViewGroup par
        final ViewHolder viewHolder;
        if (convertView == null) {
            convertView = LayoutInflater.from(parent.getContext())
                    .inflate(R.layout.task_row, parent, false);
            viewHolder = new ViewHolder();
            viewHolder.taskName = (TextView) convertView.findViewByIc
            viewHolder.taskDescription = convertView.findViewById(R.i
            viewHolder.isTaskDone = (CheckBox) convertView.findViewBy
            viewHolder.isTaskDone.setOnClickListener(listener);
            convertView.setTag(viewHolder);
        } else {
```

```
            viewHolder = (ViewHolder) convertView.getTag();
        }


        if (adapterData != null) {
            Task task = adapterData.get(position);
            viewHolder.taskName.setText(task.getName());
            viewHolder.taskDescription.setText(task.getDescription())
            viewHolder.isTaskDone.setChecked(task.isDone());
            viewHolder.isTaskDone.setTag(position);
        }

        return convertView;
    }

    private View.OnClickListener listener = new View.OnClickListener(
        @Override
        public void onClick(View view) {
            int position = (Integer) view.getTag();
            if (adapterData != null) {
                Task task = adapterData.get(position);
                activity.changeTaskDone(task.getId());
            }
        }
    };


}
```

5- Pour afficher les données dans notre listView, on doit les restaurer d'abord de notre base de données, pour se faire on doit utiliser *RealmResults* :
Pour restaurer la liste des Tasks de la base de données on utilise :
realm.where(Task.class).findAll()

Ajoutez ces lignes dans la méthode onCreate() de votre ACtivity :

```
Realm realm = Realm.getDefaultInstance();
RealmResults<Task> tasks = realm.where(Task.class).findAll();
```

Après avoir obtenu l'instance de Realm , nous interrogeons la base de données avec **realm.where (Task.class) .findAll()** pour enregistrer tous les objets Task et les affecter à un objet RealmResults. RealmResults (et RealmObject) sont des objets directs (live objects) qui sont automatiquement mis à jour lorsque des

changements surviennent dans leurs données. Le RealmBaseAdapter assure également le suivi automatique des modifications apportées à son modèle de données et des mises à jour lorsqu'une modification est détectée.
Finalement, appelez votre Adapter et le lier à la listeView . (Vous savez comment le faire).

Vous pouvez maintenant executer votre application et afficher les taches dans la listview. Mais à ce niveau là votre base de données est encore vide, nous allons la manipuler dans ce qui suit.

3. Manipulation de la base de données

Toutes les opérations d'écriture sur Realm (créer, mettre à jour et supprimer) doivent être encapsulées dans des transactions d'écriture. Une transaction d'écriture peut être validée ou annulée. Lors d'une validation, toutes les modifications sont persistées sur le disque, et une validation n'est réussie que si toutes les modifications sont conservées. En annulant une transaction d'écriture, toutes les modifications seront supprimées. Les opérations d'écriture peuvent être effectuées en utilisant le format suivant:

```
realm.beginTransaction();

//... add or update objects here ...

realm.commitTransaction();
```

Les méthodes

```
realm.executeTransaction();
```

ou

```
realm.executeTransactionAsync()
```

Par défaut, les transactions d'écriture se bloquent les unes les autres. Il est recommandé d'utiliser des transactions asynchrones sur le thread d'interface utilisateur (main Thread)qui s'exécutera sur un thread d'arrière-plan et d'éviter

de bloquer l'interface utilisateur. C'est pourquoi nous utilisons executeTransactionAsync () par opposition à l'autre fonction. Vous pouvez passer une fonction de rappel(callBack) à executeTransactionAsync () qui sera appelée lorsque la transaction sera terminée.

Créer une classe nommée RealmData, ou nous allons définir tous nos méthodes de manipulation de la base de données:

1. AJOUT

Nous créons une tâche avec **realm.createObject()**, définissons une valeur UUID aléatoire comme identifiant,définissons le texte entré par l'utilisateur comme son nom et une déscription.
Pour se faite ajouter la méthode suivante dans la classe RealmData.

```
public void addTask (Realm realm, String name, String description)
    {
        realm.executeTransactionAsync(new Realm.Transaction() {
            @Override
            public void execute(Realm realm) {
                Task task = realm.createObject(Task.class, UUID.randc
                task.setName(name);
                task.setDescription(description);


            }
        });

    }
```

2. MISE À JOUR

Pour la modification d'un élement, nous devons interroger la base de données pour une tâche avec un ID donné puis nous définissons son nom et sa déscription avec le texte transmis.

```
public  void editTask(Realm realm, final String taskId, final String


      realm.executeTransactionAsync(new Realm.Transaction() {
          @Override
          public void execute(Realm realm) {
              Task task = realm.where(Task.class).equalTo("id", taskI
              task.setName(name);
```

```
                task.setDescription(description);
            }
        });
    }
```

3. SUPRESSION

pour supprimer un élement, il suffit d'appeler la méthode **deleteFromRealm()** aprés avoir trouver l'élément à supprimer.

```
public  void deleteTask(Realm realm,final String taskId) {
    realm.executeTransactionAsync(new Realm.Transaction() {
        @Override
        public void execute(Realm realm) {
            realm.where(Task.class).equalTo("id", taskId)
                    .findFirst()
                    .deleteFromRealm();
        }
    });
}
```

4. QUELQUES MÉTHODES PRÉDÉFINIES :

La méthode "where" fait un RealmQuery en spécifiant un modèle. Le critère de filtre est spécifié avec des méthodes de prédéfinies comme par exemple "equalTo()"). Une méthode prédéfinie prend toujours un nom de champ comme premier argument.
Pour les champs numériques on peut citer les méthodes suivantes :
- between
- greaterThan
- lessThan
- greaterThanOrEqualTo
- lessThanOrEqualTo

Pour les chaînes de caractères :
- contains
- beginsWith
- endsWith
- like

Remarque

Consultez la documentation officielle pour plus de détails : ici [https://realm.io/docs/java/latest/#queries].

Les opérations sont tous définits, il nous reste qu'à implementer les méthodes onClickLIstener sur le boutton pour ajouter une nouvelle tache et sur l'element de la listeView pour modifier ou supprimer une tache.

Afin de faire une boîte de dialogue d'alerte, vous devez faire un objet de **AlertDialogBuilder** qui est une classe interne de **AlertDialog**. Sa syntaxe est donnée ci-dessous :

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this
```

y ajouter :
- La vue que vous voulez par la méthode : **setView (View view)**, dans notre cas nous allons ajouter deux EditText pour le nom et la désciption.
- Un titre : par la méthode **setTitle(String title)**
- Les bouttons positives et négatives par les méthodes : **alertDialogBuilder.setPositiveButton(CharSequence text, DialogInterface.OnClickListener listener)** et **alertDialogBuilder.setNegativeButton(CharSequence text, DialogInterface.OnClickListener listener)** .

Activité 2
Vous savez ce que vous devez faire !

4. Les migrations

Nous voulons que la liste conserve un ordre précis. Pour cela, nous allons ajouter un nouvel attribut appelé priority à chaque élément et trier la liste en utilisant ce champs. Cela signifie que nous allons changer le modèle de domaine et donc le schéma de la base de données.

Si vous avez enregistré des données sur le disque, vous ne pouvez pas simplement modifier le schéma de base de données et le faire fonctionner avec le fichier .realm enregistré. Vous devez effectuer une migration de l'ancien schéma vers la nouvelle définition. S'il n'y a pas de fichier sur le disque lors du lancement de Realm, aucune migration n'est nécessaire. Realm créera

simplement un nouveau fichier .realm et un schéma basé sur les derniers modèles définis dans votre code.

Si votre application est en cours de développement et que vous acceptez de perdre des données enregistrées, vous pouvez simplement supprimer le fichier .realm sur le disque au lieu d'avoir à écrire une migration. Vous pouvez supprimer le fichier Realm avec le code suivant dans la classe App. N'oubliez pas de supprimer les instructions Realm.deleteRealm (realmConfig) lors des prochaines exécutions, sinon votre base de données sera supprimée à chaque lancement de l'application.

```
RealmConfiguration realmConfig = new RealmConfiguration.Builder()
        .schemaVersion(0)
        .build();
Realm.deleteRealm(realmConfig);
Realm.setDefaultConfiguration(realmConfig);
```

Cette solution n'est pas idéales que pendant la phase de développement. Si vous avez une application en production et que vous souhaitez publier une mise à jour qui utilise un schéma différent, vous ne voulez pas que vos utilisateurs perdent leurs données lorsqu'ils mettent à jour l'application. Pour cela, vous devrez créer une migration.

Tout d'abord, ajoutez un champ priority au modèle Task et ajoutez des fonctions setter et getter pour celui-ci.
Trier ensuite la liste des taches suivant ce champ :

```
RealmResults<Task> result = realm.where(Task.class).sort("priority").
```

Créez ensuite une classe nommée Migration modifiez-la comme indiqué ci-dessous.

```
public class Migration implements RealmMigration {
    @Override
    public void migrate(DynamicRealm realm, long oldVersion, long new
        RealmSchema schema = realm.getSchema();
        if (oldVersion == 0) {
            RealmObjectSchema taskSchema = schema.get("Task");
```

```
            taskSchema.addField("priority", int.class)
                    .transform(new RealmObjectSchema.Function() {
                        @Override
                        public void apply(DynamicRealmObject obj) {
                            obj.set("priority", 0);
                        }
                    });
            oldVersion++;
        }
    }


}
```

Dans Ce qui précède nous spécifions une migration du schéma de la version 0 (qui est le numéro de version que nous avons défini lors de sa création) à 1.
Nous utilisons *addField ()* pour ajouter un nouveau champ à la classe de modèle RealmObject. Le bloc transform () n'est pas nécessaire, nous l'ajoutons ici pour montrer comment vous allez définir une valeur pour le champ dans des objets déjà existants. transform () exécute une fonction de transformation sur chaque instance RealmObject de la classe en cours. Dans ce qui précède, les objets existants auront une valeur de priorité égale à 0.
Nous incrémentons ensuite la valeur de oldVersion.

Enfin, dans votre classe App, remplacez le schemaVersion par la version à laquelle vous effectuez la mise à jour et ajoutez la migration au code de configuration avec migration ().

```
RealmConfiguration realmConfig = new RealmConfiguration.Builder()
        .schemaVersion(1)
        .migration(new Migration())
        .build();
Realm.setDefaultConfiguration(realmConfig);
```


Attention
Si vous changez le shéma de la base de données sans faire une migrations vous aurez l'exception :
io.realm.exceptions.RealmMigrationNeededException

Homework

Toute application consistante ne peut pas de contenter d'un stockage en interne dans le device. Il faut donc penser à d'autres modes de stockage distants. L'un de ces modes de stockage est le stockage dans une base de données hébergée sur le cloud.
Les systèmes Android fonctionnent parfaitement avec l'une de ces bases de données, Firebase. Créer une base de données avec Firebase, et un client Android permettant d'afficher son contenu dans des Cards (en utilisant le RecyclerView), de supprimer et d'ajouter de nouveaux éléments.
Remarquez la réactivité de ces opérations en lançant à la fois votre émulateur, un appareil Android et un navigateur montrant le contenu de la base.
Activer également la fonction permettant de manipuler les éléments de la base en Offline.