RAPPORT
TP3 Bis

Kotlin

BOURDIER Loann
G4B

Table des matières

I. Exercice Livre

A. Objectif

L'objectif de ce TP est de se familiariser un avec une base de donnée en Kotlin, ici SQLite. Pour ce faire, nous devons gérer des livres avec possibilité d'ajout, de suppression, de modification et de consultation.

Ces derniers ont :

- Un id
  - Type : Int et unique
- Un numéro Isbn
  - Type : String
- Un titre
  - Type : String

B. Fonctionnement de SQLite

Tout d'abord SQLite est une base de données open source qui supporte les fonctionnalités standards des bases de données relationnelles. Il est donc possible pour nous d'utiliser la syntaxe SQL.

De plus, SQLite demande très peu de mémoire à l'exécution. Cela la rend assez pratique en ce qui concerne les systèmes embarqués. SQLite prend en charge les types de données TEXT (String), INT (Long) ainsi que REAL (Double), cependant, tous les autres types doivent être convertis en l'un de ces types avant d'être enregistré dans la base. Dans autre exercice, il n'y a pas besoin de convertir un type car on utilise 1 Int et 2 String. SQLite ne vérifie pas non plus si les types de données insérées dans les colonnes correspondent au type défini c'est-à-dire qu'il est possible d'insérer un Int dans une colonne de type String. Un autre avantage de SQLite, c'est qu'il est intégré à chaque appareil Android et ne nécessite pas d'administration de la base.

La fonctionnalité permettant d'utiliser la base de données SQLite est fournit par la classe SQLiteOpenHelper. Cette classe est utilisée pour la création de bases de données et la gestion des versions.

C. Explication des fichiers

1. Fichier MainActivity.kt

Nous retrouvons dans ce fichier faisant le lien entre les différents différentes classes contenues dans les packages adapter, classes, handler plusieurs méthodes que l'ont retrouvent ci-dessous.

A) Méthode saveRecord

Cette méthode s'occupe d'enregistrer les valeurs saisies par l'utilisateur. Dans cette méthode on vérifie d'abord si les valeurs saisies dans les champs id, isbn et titre ne sont pas vide, si c'est le cas on les enregistres sinon on affiche un message d'erreur.

```
//méthode pour sauvegarder les valeurs saisies dans une base de données
fun saveRecord(view: View){
    val id = u_id.text.toString()
    val isbn = u_isbn.text.toString()
    val titre = u_titre.text.toString()
    val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
    if(id.trim()!="" && isbn.trim()!="" && titre.trim()!=""){
        val status = databaseHandler.addLivre(livModelClass(Integer.parseInt(id),isbn, titre))
        if(status > -1){
            Toast.makeText(applicationContext, text: "valeurs enregistrées",
                    Toast.LENGTH_LONG).show()
            u_id.text.clear()
            u_isbn.text.clear()
            u_titre.text.clear()
        }
    } else {
        Toast.makeText(applicationContext, text: "id ou isbn ou titre ne peut être vide",
                Toast.LENGTH_LONG).show()
    }

}
```

B) *Méthode viewRecord*

Cette méthode s'occupe de lire les valeurs enregistrées et s'occupe également des les afficher dans un ListView. Dans cette méthode créer d'abord une variable pour avoir accès à la classe DatabaseHandler, ensuite on appelle la méthode viewLivre se trouvant dans la classe DtabaseHandler afin de lire les valeurs précédemment enregistrées.

```
//Méthode pour lire les valeurs enregistrés de la base de données dans le ListView
fun viewRecord(view: View){

    //création de l'instance de la classe DatabaseHandler
    val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
    //appel de la méthode viewLivre contenu dans la classe DatabaseHandler pour lire les valeurs
    val liv: List<livModelClass> = databaseHandler.viewLivre()
    val livArrayId = Array<String>(liv.size){"0"}
    val livArrayIsbn = Array<String>(liv.size){"null"}
    val livArrayTitre = Array<String>(liv.size){"null"}
    var index = 0
    for(e in liv){
        livArrayId[index] = e.livreId.toString()
        livArrayIsbn[index] = e.livreIsbn
        livArrayTitre[index] = e.livreTitre
        index++
    }
    //création d'un ArrayAdapter personnalisé
    val myListAdapter = MyListAdapter( context: this,livArrayId,livArrayIsbn,livArrayTitre)
    listView.adapter = myListAdapter
}
```

C) *Méthode updateRecord*

Cette méthode s'occupe de mettre à jour les valeurs voulues par l'utilisateur et déjà enregistrées. Il choisit le livre qu'il veut modifier grâce à l'id de ce dernier car il est unique et ça peut y avoir plusieurs livres avec le même titre ou le même isbn. Si lors de la modification un champ au minimum est vide un message d'erreur sera afficher. Il est également possible d'annuler une modification.

```
//méthode de mise à jour des valeurs enregistrées
fun updateRecord(view: View){
    val dialogBuilder = AlertDialog.Builder( context: this)
    val inflater = this.layoutInflater
    val dialogView = inflater.inflate(R.layout.update_dialog,  root: null)
    dialogBuilder.setView(dialogView)

    val edtId = dialogView.findViewById(R.id.updateId) as EditText
    val edtIsbn = dialogView.findViewById(R.id.updateIsbn) as EditText
    val edtTitre = dialogView.findViewById(R.id.updateTitre) as EditText

    dialogBuilder.setTitle("Mise à jour des valeurs")
    dialogBuilder.setMessage("Entrez les données ci-dessous")
    dialogBuilder.setPositiveButton( text: "Mise à jour", DialogInterface.OnClickListener { _, _
        ->
        val updateId = edtId.text.toString()
        val updateIsbn = edtIsbn.text.toString()
        val updateTitre = edtTitre.text.toString()
        //création de l'instance de la classe DatabaseHandler
        val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
        if(updateId.trim()!="" && updateIsbn.trim()!="" && updateTitre.trim()!=""){
            //appel de la méthode updateLivre contenu dans la classe DatabaseHandler
                // pour lire les valeurs
            val status = databaseHandler.updateLivre(livModelClass(Integer.parseInt(updateId),
                    updateIsbn, updateTitre))
            if(status > -1){
                Toast.makeText(applicationContext, text: "valeurs mises à jour",
                        Toast.LENGTH_LONG).show()
            }
        }else{
            Toast.makeText(applicationContext, text: "id ou isbn ou titre ne peut pas être vide",
                    Toast.LENGTH_LONG).show()
        }

    })
    dialogBuilder.setNegativeButton( text: "Annuler", DialogInterface.OnClickListener { dialog,
                                                                                         which ->
        //pass
    })
    val b = dialogBuilder.create()
    b.show()
}
```

D) Méthode deleteRecord

Cette méthode s'occupe de supprimer un livre. Il choisit le livre qu'il veut suppprimer en renseignant seulement l'id de ce dernier car il est unique. Lorsque la suppression a correctement été réalisée un message pour prévenir l'utilisateur s'affiche et si l'utilisateur ne renseigne pas l'id, un message s'affiche. Tout comme la modification il est possible d'annuler la suppression.

```
//méthode de suppression d'enregistrements basée sur l'identification
fun deleteRecord(view: View){
    //création d'un AlertDialog pour prendre l'identifiant de l'utilisateur
    val dialogBuilder = AlertDialog.Builder( context: this)
    val inflater = this.layoutInflater
    val dialogView = inflater.inflate(R.layout.delete_dialog, root: null)
    dialogBuilder.setView(dialogView)

    val dltId = dialogView.findViewById(R.id.deleteId) as EditText
    dialogBuilder.setTitle("Supression du livre Record")
    dialogBuilder.setMessage("Entrez l'id ci-dessous")
    dialogBuilder.setPositiveButton( text: "Delete", DialogInterface.OnClickListener { _, _ ->

        val deleteId = dltId.text.toString()
        //creating the instance of DatabaseHandler class
        val databaseHandler: DatabaseHandler= DatabaseHandler( context: this)
        if(deleteId.trim()!=""){
            //calling the deleteLivre method of DatabaseHandler class to delete record
            val status = databaseHandler.deleteLivre(livModelClass(Integer.parseInt(deleteId),
                    livreIsbn: "", livreTitre: ""))
            if(status > -1){
                Toast.makeText(applicationContext, text: "Livre supprimé",
                        Toast.LENGTH_LONG).show()
            }
```

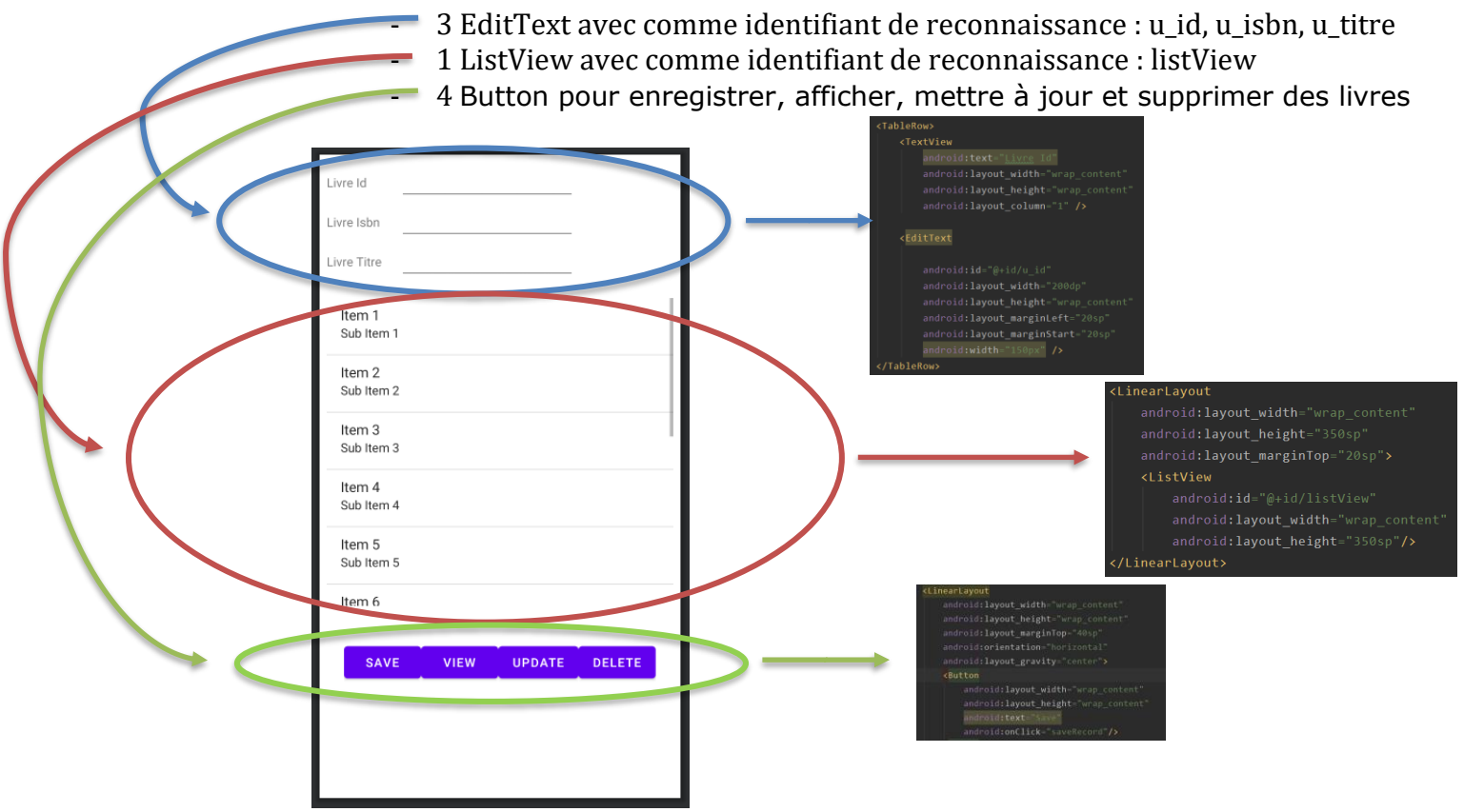
```
        }else{
            Toast.makeText(applicationContext, text: "id ou isbn ou titre ne doit pas être vide",
                    Toast.LENGTH_LONG).show()
        }

    })
    dialogBuilder.setNegativeButton( text: "Annuler", DialogInterface.OnClickListener { _, _ ->
        //pass
    })
    val b = dialogBuilder.create()
    b.show()
}
```

2. Fichier activity_main.xml

Dans ce fichier nous retrouvons l'interface graphique « d'accueil » sur le téléphone (quand l'utilisateur ouvre l'application). Ce fichier est l'interface graphique du fichier ActivityMain.kt.

Nous retrouvons dans ce fichier :

- 3 EditText avec comme identifiant de reconnaissance : u_id, u_isbn, u_titre
- 1 ListView avec comme identifiant de reconnaissance : listView
- 4 Button pour enregistrer, afficher, mettre à jour et supprimer des livres




```
<TableRow>
    <TextView
        android:text="Livre Id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1" />

    <EditText

        android:id="@+id/u_id"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20sp"
        android:layout_marginStart="20sp"
        android:width="150px" />
</TableRow>
```

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="350sp"
    android:layout_marginTop="20sp">
    <ListView
        android:id="@+id/listView"
        android:layout_width="wrap_content"
        android:layout_height="350sp"/>
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="400sp"
    android:orientation="horizontal"
    android:layout_gravity="center">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Save"
        android:onClick="saveRecord"/>
```

3. Fichier livModelClass.kt

Ce fichier contient seulement la déclaration de la classe livModelClass avec en paramètre l'id, l'isbn et le titre du livre. Les getters, setters et le constructeur sont implémentés automatiquement.

```
class livModelClass(var livreId: Int, val livreIsbn:String , val livreTitre: String)

}
```

4. Fichier MyListAdapter.kt

Ce fichier permet d'afficher la liste des livres une fois que le bouton « view » a été appuyé.

```
class MyListAdapter(private val context: Activity, private val id: Array<String>, private val isbn: Array<String>, private
    : ArrayAdapter<String>(context, R.layout.custom_list, isbn) {

    override fun getView(position: Int, view: View?, parent: ViewGroup): View {
        val inflater = context.layoutInflater
        val rowView = inflater.inflate(R.layout.custom_list, root: null, attachToRoot: true)

        val idText = rowView.findViewById(R.id.textViewId) as TextView
        val isbnText = rowView.findViewById(R.id.textViewIsbn) as TextView
        val titreText = rowView.findViewById(R.id.textViewTitre) as TextView

        idText.text = "Id: ${id[position]}"
        isbnText.text = "Isbn: ${isbn[position]}"
        titreText.text = "Titre: ${titre[position]}"
        return rowView
    }
}
```

5. Fichier DatabaseHandler.kt

a) *Companion object*

Un companion object est une instance unique pour une classe donnée, il est partagé par toutes les instances de la classe que vous allez créer par la suite et pour toute la durée de vie des instances de la classe.

Ici on donne le nom de la base de donnée, le nom de la table, la version ainsi qui que le nom des colonnes de la table.

```
companion object {
    private val DATABASE_VERSION = 1
    private val DATABASE_NAME = "LivreDatabase"
    private val TABLE_LIVRES = "LivreTable"
    private val KEY_ID = "id"
    private val KEY_ISBN = "isbn"
    private val KEY_TITRE = "titre"
}
```

b) *Fonction onCreate*

Cette fonction s'occupe de créer la table « LivreTable » avec différents champs

```
override fun onCreate(db: SQLiteDatabase?) {
    // TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
    //création d'une tables avec des champs
    val CREATE_LIVRES_TABLE = ("CREATE TABLE " + TABLE_LIVRES + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_ISBN + " TEXT,"
            + KEY_TITRE + " TEXT" + ")")
    db?.execSQL(CREATE_LIVRES_TABLE)
}
```

c) *Fonction onUpgrade*

Cette fonction va s'occuper de supprimer la table déjà en registrer avec les anciennes valeurs et va la remplacer avec la table contenant les valeurs insérées les plus récente

```
override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    //  TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
    db!!.execSQL( sql: "DROP TABLE IF EXISTS $TABLE_LIVRES")
    onCreate(db)
}
```

d) *Fonction addLivre*

Cette fonction permet d'insérer des données dans la table, ici les données sont l'id, l'isbn et le titre.

```
//méthode pour insérer des données
fun addLivre(liv: livModelClass):Long{
    val db = this.writableDatabase
    val contentValues = ContentValues()
    contentValues.put(KEY_ID, liv.livreId)
    contentValues.put(KEY_ISBN, liv.livreIsbn) // livModelClass Isbn
    contentValues.put(KEY_TITRE,liv.livreTitre) // livModelClass Titre
    // Insertion d'une rangée
    val success = db.insert(TABLE_LIVRES,  nullColumnHack: null, contentValues)
    //Le second argument est une chaîne de caractères contenant nullColumnHack.
    db.close() // Fermeture de la connexion à la base de données
    return success
}
```

e) *Fonction viewLivre*

Cette fonction permet de lire les données contenues dans la table

```
//méthode de lecture des données
fun viewLivre():List<livModelClass>{
    val livList:ArrayList<livModelClass> = ArrayList<livModelClass>()
    val selectQuery = "SELECT  * FROM $TABLE_LIVRES"
    val db = this.readableDatabase
    var cursor: Cursor? = null
    try{
        cursor = db.rawQuery(selectQuery,  selectionArgs: null)
    }catch (e: SQLiteException) {
        db.execSQL(selectQuery)
        return ArrayList()
    }
    var livreId: Int
    var livreIsbn: String
    var livreTitre: String
    if (cursor.moveToFirst()) {
        do {
            livreId = cursor.getInt(cursor.getColumnIndex( columnName: "id"))
            livreIsbn = cursor.getString(cursor.getColumnIndex( columnName: "isbn"))
            livreTitre = cursor.getString(cursor.getColumnIndex( columnName: "titre"))
            val liv= livModelClass(livreId = livreId, livreIsbn = livreIsbn, livreTitre = livreTitre)
            livList.add(liv)
        } while (cursor.moveToNext())
    }
    return livList
}
```

f) *UpdateLivre*

Cette fonction permet de mettre à jour les données contenues dans la table

```
//méthode de mise à jour des données
fun updateLivre(liv: livModelClass):Int{
    val db = this.writableDatabase
    val contentValues = ContentValues()
    contentValues.put(KEY_ID, liv.livreId)
    contentValues.put(KEY_ISBN, liv.livreIsbn) // livModelClass Isbn
    contentValues.put(KEY_TITRE,liv.livreTitre ) // livModelClass Titre

    // Mise à jour de la rangée
    val success = db.update(TABLE_LIVRES, contentValues, whereClause: "id="+liv.livreId, whereArgs: null)
    //Le second argument est une chaîne de caractères contenant nullColumnHack.
    db.close() // Fermeture de la connexion à la base de données
    return success
}
```

g) deleteLivre

Cette fonction permet de supprimer les données contenues dans la table

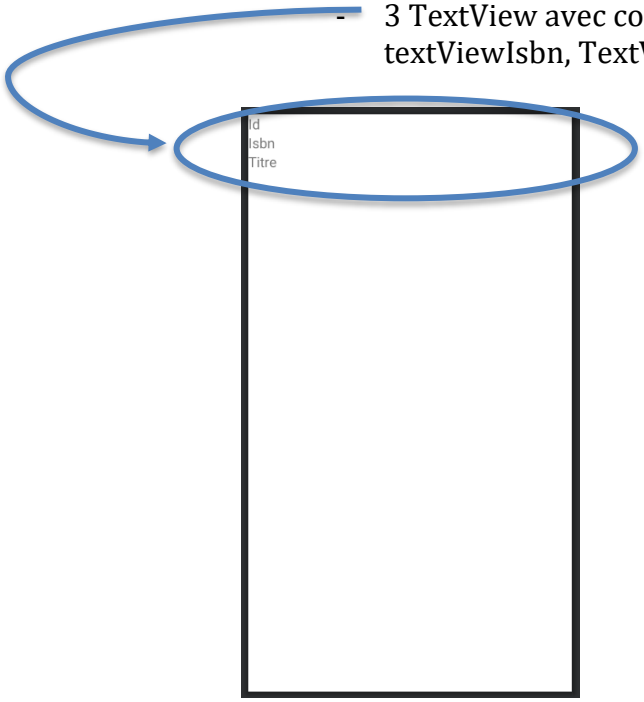
```
//méthode pour supprimer les données
fun deleteLivre(liv: livModelClass):Int{
    val db = this.writableDatabase
    val contentValues = ContentValues()
    contentValues.put(KEY_ID, liv.livreId) // livModelClass livreId
    // Suppression d'une rangée
    val success = db.delete(TABLE_LIVRES, whereClause: "id="+liv.livreId, whereArgs: null)
    //Le second argument est une chaîne de caractères contenant nullColumnHack.
    db.close() // Fermeture de la connexion à la base de données
    return success
}
```

6. Fichier custom_list.xml

Ce fichier permet de créer une disposition de ligne personnalisée afin d'afficher les éléments de liste dans la ListView.

Il comprend :

- 3 TextView avec comme identifiant de reconnaissance : textViewId, textViewIsbn, TextViewTitre



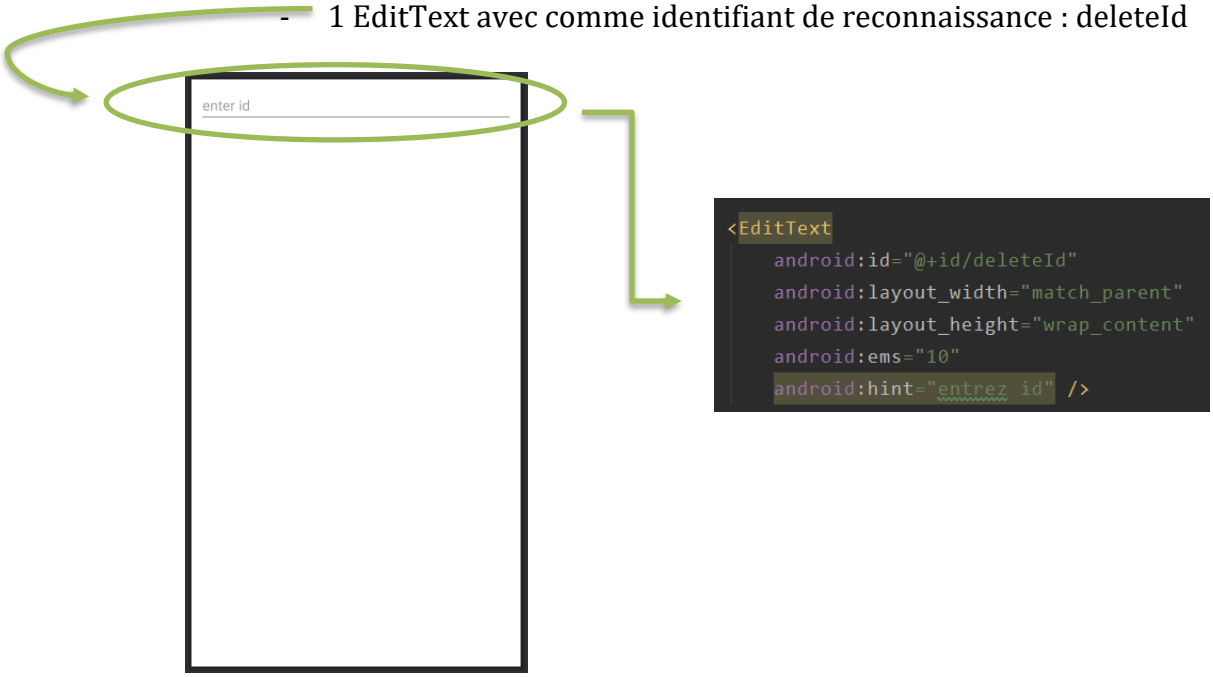

```
<TextView
    android:id="@+id/textViewId"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Id"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"/>

<TextView
    android:id="@+id/textViewIsbn"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Isbn"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium" />

<TextView
    android:id="@+id/textViewTitre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Titre"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium" />
```

7. Fichier delete_dialog.xml

Ce fichier permet de créer une mise en page pour afficher AlertDialog afin de supprimer le livre grâce à son identifiant.

Il comprend :

- 1 EditText avec comme identifiant de reconnaissance : deleteId

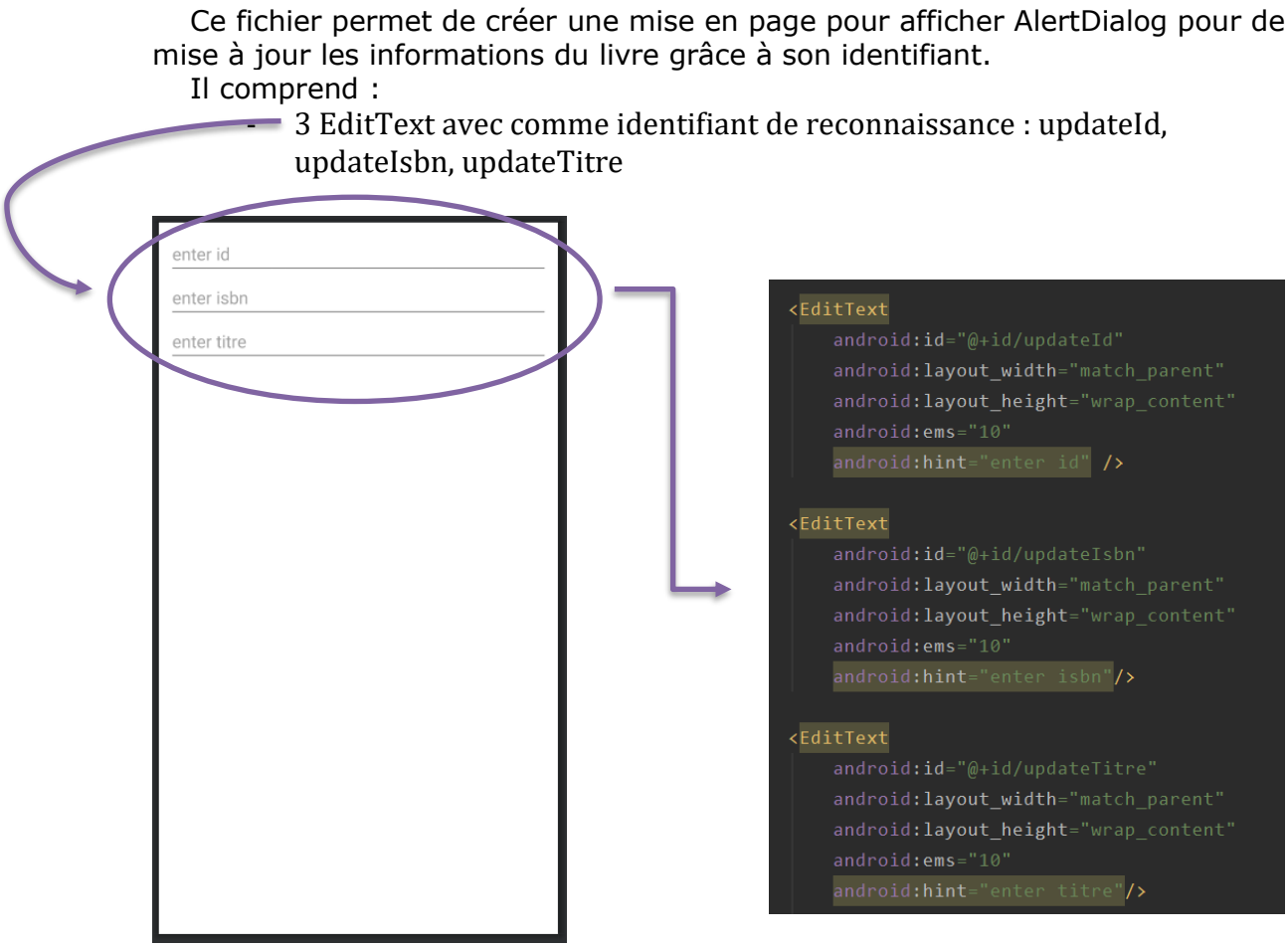


8. Fichier update_dialog.xml

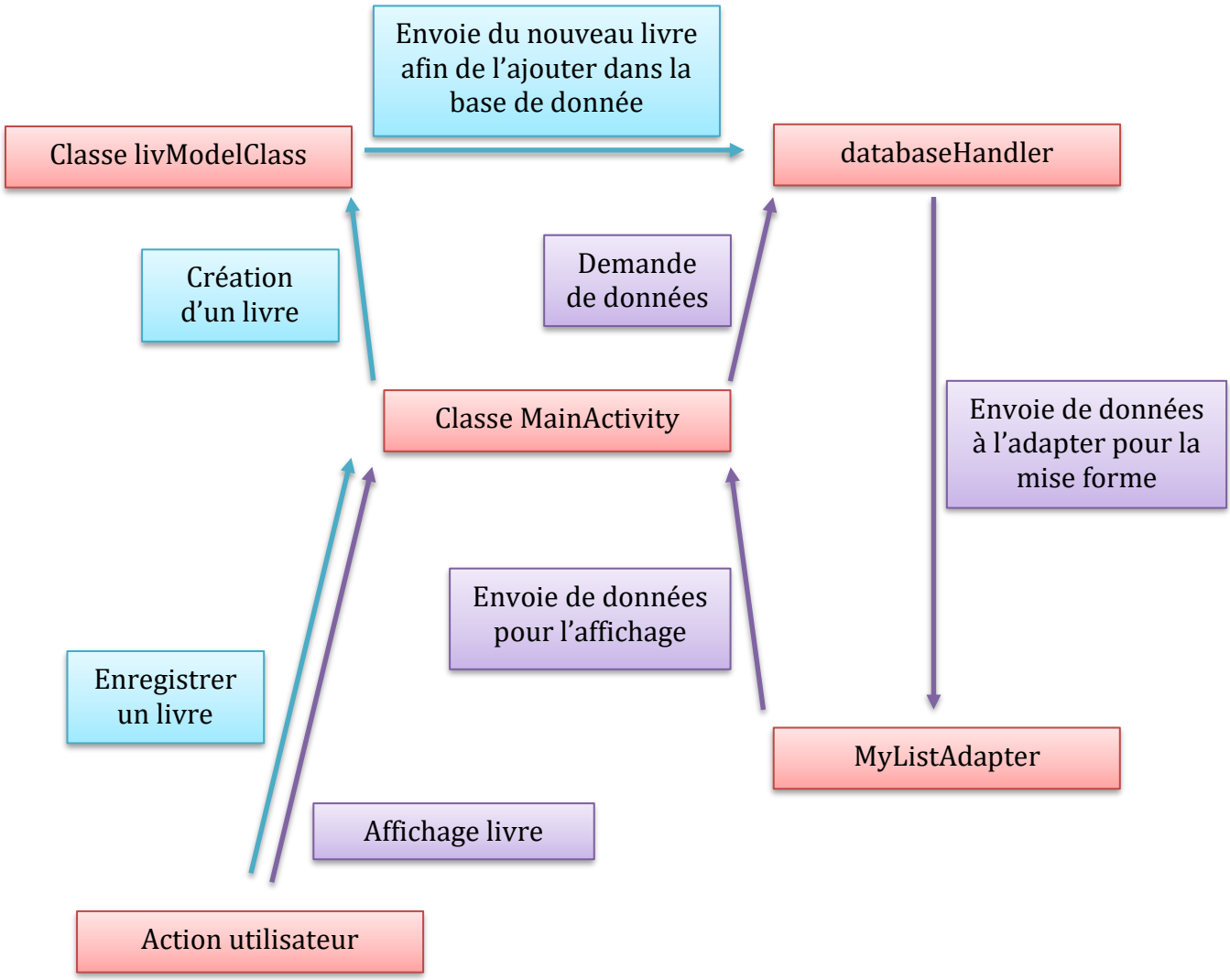
Ce fichier permet de créer une mise en page pour afficher AlertDialog pour de mise à jour les informations du livre grâce à son identifiant.

Il comprend :

- 3 EditText avec comme identifiant de reconnaissance : updateId, updateIsbn, updateTitre



D. Schéma


Envoie du nouveau livre afin de l'ajouter dans la base de donnée
Classe livModelClass
databaseHandler
Création d'un livre
Demande de données
Classe MainActivity
Envoie de données à l'adapter pour la mise forme
Envoie de données pour l'affichage
Enregistrer un livre
MyListAdapter
Affichage livre
Action utilisateur