Mobil megoldások

Kocsis Gergely

2019.01.04.

Room

A perzisztens adattárolás legkifinomultabb formája az adatok adatbázisba történő mentése. Az adatbázisok kezelése természetesen minden egyébként is elérhető eszközzel lehetséges Android alkalmazások alól is.

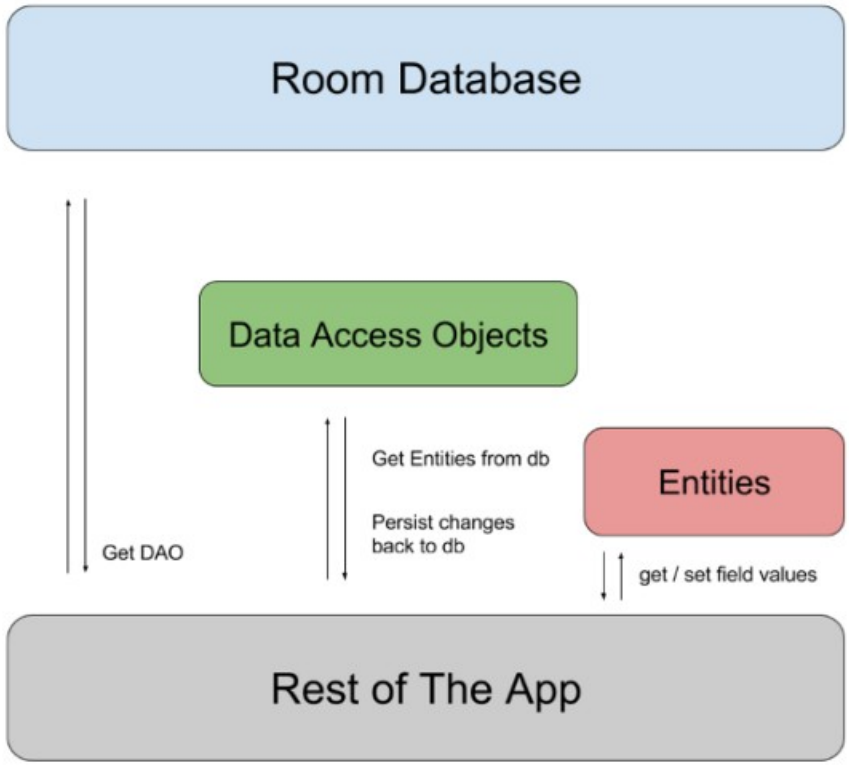
Az SDK azonban natív lehetőséget is kínál erre. Ezt hívják Room-nak.

Készítsünk olyan alkalmazást, mely a korábbiakban elkészítetthez hasonlóan egy bevásárlólistát implementál, a listát viszont adatbázisban tárolja.

Room

Egy minimális Room alkalmazás három komponenst kell, hogy tartalmazzon:

- Entity: Egy osztály, mely az adatbázis táblát reprezentálja
- DAO (Data Access Object): Interfész, mely az adatbázis elérését írja le. Az interfészt a Room automatikusan implementálja
- Adatbázis: Egy olyan absztrakt osztály, mely a RoomDatabase osztályt terjeszti ki. Feladata az adatbázis kapcsolat kezeélse



https://developer.android.com/training/data-storage/room/index.html

Room

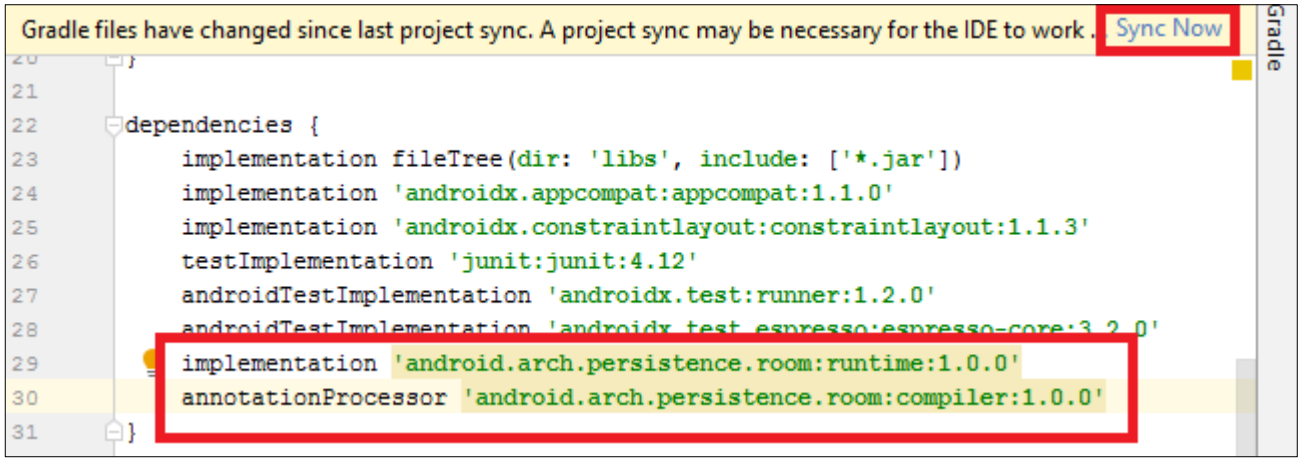
Hozzunk létre egy üres alkalmazást és benne az Entity osztályt.

Az osztály neve legyen ShoppingListItem. Ez egy egyszerű Bean osztály. Különlegességét a JPA-ban is ismerthez hasonló annotációk adják.

Ezeket az annotációkat alapértelmezés szerint nem tudja feldolgozni az alkalmazásunk. Hozzá kell adnunk a megfelelő függőségeket a gradle.build fájlhoz a dependencies tag-be. Használhatjuk a legfrissebb elérhető verziót. (https://developer.android.com/jetpack/androidx/releases/room)

```
implementation 'android.arch.persistence.room:runtime:2.2.1'
annotationProcessor 'android.arch.persistence.room:compiler:2.2.1'
```

Szinkronizáljuk a projektet újra, mielőtt továbbhaladunk.



Room

Hozzunk létre egy üres alkalmazást és benne az Entity osztályt.

Az osztály neve legyen ShoppingListItem. Ez egy egyszerű Bean osztály. Különlegességét a JPA-ban is ismerthez hasonló annotációk adják.

```
import androidx.annotation.NonNull;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "ShoppingList")
public class ShoppingListItem {
    @PrimaryKey
    @NonNull
    private int itemID;

    private String itemName;

    //... getters and setters

    @Override
    public String toString() {
        return itemID + " - " + itemName;
    }
```

Room

A második feladat a DAO interfész létrehozása. Az interfész metódus szignatútári adják meg az adatbázisban végzendő műveleteket. Itt implementálhatjuk az alapvetű CRUD műveleteket.

Egyelőre készítsünk egy egyszerű hozzáadó és egy listázó műveletet.

```
@Dao
public interface ShoppingListDAO {
    @Insert
    void insertListItem(ShoppingListItem sli);

    @Query("SELECT * FROM ShoppingList")
    List<ShoppingListItem> getAllItems();
}
```

A tábla neve

Létezik még @Update és @Delete annotáció is. Ezekkel a későbbiekben bővítjük a projektet.

Room

A harmadik feladat az adatbázis létrehozása. Az adatbázis osztály egy a RoomDatabase osztályt kiterjesztő absztrakt osztály.
Az osztályban minden kezelendő entitáshoz adjunk “getter” metódust.

```
@Database(entities = {ShoppingListItem.class}, version = 1, exportSchema = false)
public abstract class ShoppingListDatabase extends RoomDatabase {
    public abstract ShoppingListDAO shoppingListDAO();
}
```

A @Database annotációban minden kezelendő entitást leíró osztályt fel kell sorolni. A verziót a séma változásakor kell növelni.

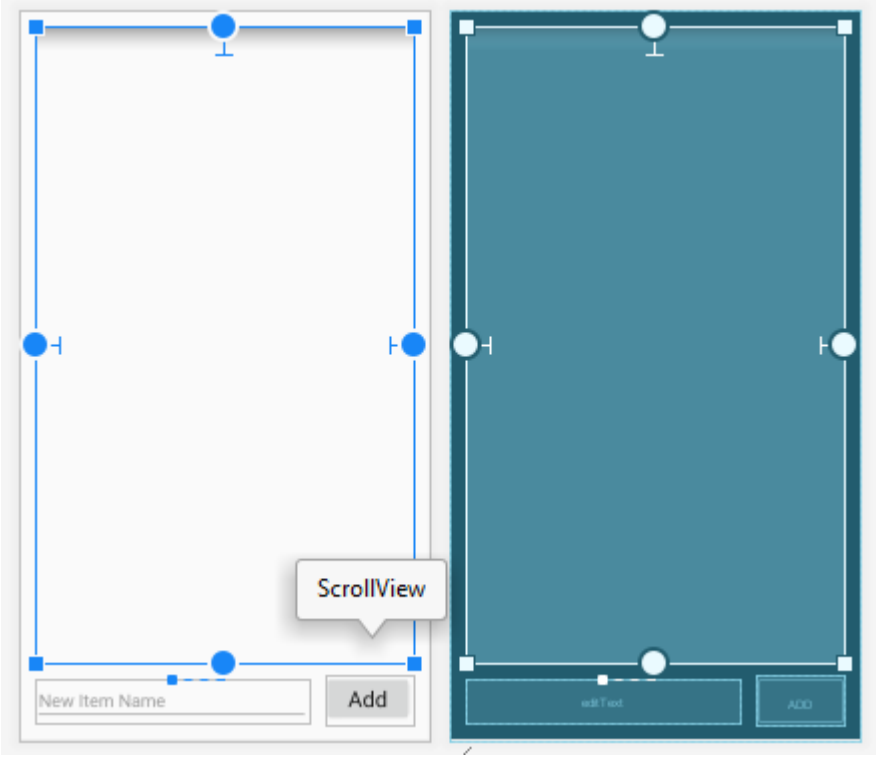
Az adatbázis létrehozása költséges művelet, ezért a későbbiekben figyeljünk oda, hogy az csak egyszer történjen meg.

Room

Készítsük el az alkalmazás felhasználói felületét. A felületen egy gomb segítségével lehet új elemet a listához adni. A teljes lista egy ScrollWiew-ban látszik.

Az Add gomb megnyomására a MainActivity addItem metódusa hívódik meg.

Az új elem megadására szolgáló EditText hint attribútuma: “New item name”.



Room

Teszteljük az adatbázisunkat. Ehhez szükség van egy referenciára a MainActivity-ben, amit az onCreate-ben példányosítunk is. (Már most jegyezzük meg, hogy ezzel antimintátkövetünk!).

```
private ShoppingListDatabase shoppingListDatabase = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    shoppingListDatabase = Room.databaseBuilder(
            this, ShoppingListDatabase.class, "shoppingList_db")
            .fallbackToDestructiveMigration()
            .build();
```

A probléma, hogy így valahányszor az onCreate lefut, mindig újrapéldányosodik az adatbázis is.

Room

Teszteljük az adatbázisunkat.

Végezzünk el két egyszerű adatbázis műveletet is. Figyeljünk oda, hogy ezek a műveletek ne a fő folyamatban fussanak. Ennek megoldására két lehetőség kínálkozik.

Egyrészt alkalmazható az Android natív megoldása, az AsyncTask:

```
\\onCtreate
new AsyncTask<String, Void, Void>() {
    @Override
    protected Void doInBackground(String... itemData) {
        ShoppingListItem sli = new ShoppingListItem();
        sli.setItemID(Integer.parseInt(itemData[0]));
        sli.setItemName(itemData[1]);
        shoppingListDatabase.shoppingListDAO().insertListItem(sli);
        return null;
    }
}.execute("1", "Apple");
```

Room

Teszteljük az adatbázisunkat.

Végezzünk el két egyszerű adatbázis műveletet is. Figyeljünk oda, hogy ezek a műveletek ne a fő folyamatban fussanak. Ennek megoldására két lehetőség kínálkozik.

Másrészt használhatjuk a klasszikus Java megoldást a Thread segítségével:

```
\\onCtreate
new Thread(new Runnable() {
    @Override
    public void run() {
        Log.d("CHECKdb",
          shoppingListDatabase.shoppingListDAO().getAllItems().toString());
    }
}).start();
```

Room

A minta lekérdezés alapján adjunk a MainActivityhez és hívjunk meg az onCreate-ből egy refreshScrollView metódust, mely a listát megjeleníti egy a ScrollView-ban lévő TextView-ban.

Szintén a minta alapján implementáljuk az addItem metódust is. Hogy ne kelljen mindig kézzel állítani az új elemek azonosítóját, annotáljuk a ShoppingListItem Entitás ID mezőjét az alábbiak szerint.

```
@Entity(tableName = "ShoppingList")
public class ShoppingListItem {
    @PrimaryKey(autoGenerate = true)
    @NonNull
    private int itemID;
```

Így az ID-t meg sem kall adnunk egy új elem hozzáadásakor.
Mivel ezzel módosult a séma, módosítsuk az adatbázis verziószámát!

```
@Database(entities = {ShoppingListItem.class}, version = 2, exportSchema = false)
```

.

Room

Jó tudni, hogy a UI elemeit csak a UI Thread módosíthatja. Ellenkező esetben egy nehezen reprodukálható hibára futunk.

```
private void refreshScrollView() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            List<ShoppingListItem> shoppingList;
            final String listText =
shoppingListDatabase.shoppingListDAO().getAllItems().toString();
            runOnUiThread(new Runnable(){
                @Override
                public void run() {
                    itemsView.setText(listText);
                }
            });
        }
    }).start();
}
```

Room

Oldjuk meg az adatbázis példányosításakor elkövetett hibát.

Alakítsuk át a Database osztályunkat úgy, hogy csak Egykeként (Singleton) lehessen példányosítani.