

Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék

Mobilprogramozás alapok

GEIAL335-BL

Jegyzőkönyv

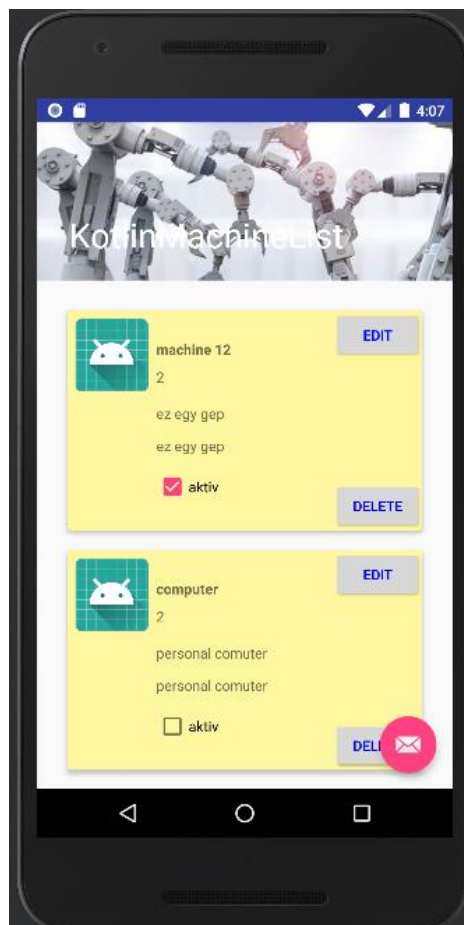
Készítette: Salamon István

Neptun: FA6VDV

Kurzusvezető: Dr. Agárdi Anita

1. Bevezetés

A KotlinMachine egy android alkalmazás, amelyet ipari vagy karbantartási környezetekben használt gépek adatainak nyilvántartására és kezelésére lehet alkalmazni. Az alkalmazás lehetővé teszi gépek egyszerű azonosítását, felvételét, és az állapotuk követését. Minden géphez kapcsolódóan rögzíthetők alapvető információk, mint például a gép neve, mennyisége, állapota aktív/nem aktív, rövid leírása és az esetleges hibajelzések. Az alkalmazás célja, hogy segítse a gépek rendszerezését és állapotuk naprakész követését.



2. Adatbázis Struktúra és Kialakítás

Az adatbázis kialakításának alapját a Room ORM biztosítja, amely egy egyszerű, de hatékony megoldás az Android alkalmazások adatkezelésére. A Room segítségével egy objektumorientált hozzáférést kapunk az adatbázishoz, megkönnyítve ezzel az SQL lekérdezések kezelését és karbantartását.

- **Adatbázis neve:** machine.db
- **Táblanév:** machineitem
- **Entitás:** MachineItem

Az adatbázis a machineitem táblát tartalmazza, amely egyedi MachineItem elemeket tárol, beleértve az elem nevét, mennyiségét, aktív állapotát, leírását, valamint a hibák leírását.

MachineItem Entitás

A *MachineItem* osztály definiálja a tábla oszlopait és az adatstruktúrát, amelyben az egyes elemek adatai tárolódnak.

```
@Entity(tableName = "machineitem")
data class MachineItem(@PrimaryKey(autoGenerate = true) var itemId: Long?,
    @ColumnInfo(name = "name") var name: String,
    @ColumnInfo(name = "quantity") var quantity: Int,
    @ColumnInfo(name = "aktiv") var aktiv: Boolean,
    @ColumnInfo(name = "description") var descript: String,
    @ColumnInfo(name = "fault") var fault: String
) : Serializable
```

- **itemId:** Elsődleges kulcs, automatikusan generált.
- **name:** Az elem neve, String típus.
- **quantity:** Mennyiség Int típusban.
- **aktiv:** Boolean érték az aktív állapot jelzésére.
- **descript:** Az elem rövid leírása.
- **fault:** Az elemhez tartozó hibajelzés.

3. DAO Interfész

A DAO (Data Access Object) interfész biztosítja az adatbázissal való műveletek elvégzését, beleértve az elemek beszúrását, lekérdezését, frissítését és törlését.

```
@Dao
interface MachineItemDAO {

    //Az összes listázása
    @Query( value: "SELECT * FROM machineitem")
    fun findAllItems(): List<MachineItem>

    //Egy elem beszúrása
    @Insert
    fun insertItem(item: MachineItem): Long

    //Egy törlése
    @Delete
    fun deleteItem(item: MachineItem)

    //Egy módosítása
    @Update
    fun updateItem(item: MachineItem)

}
```

MachineItemDAO Metódusok

- **findAllItems():** Visszaadja az összes MachineItem elemet a machineitem táblából.
- **insertItem(item: MachineItem):** Új MachineItem elemet szúr be az adatbázisba, és visszaadja az újonnan beszúrt elem azonosítóját.
- **updateItem(item: MachineItem):** Frissíti a meglévő MachineItem elemet az adatbázisban.
- **deleteItem(item: MachineItem):** Törli a megadott MachineItem elemet az adatbázisból.

4. Adatbázis Kezelés és Singleton Implementáció

Az adatbáziskezelést a *AppDatabase* osztály biztosítja, amely egy singleton adatbázis-példányt hoz létre, hogy az egész alkalmazásban hozzáférhető legyen. A singleton minta biztosítja, hogy az adatbázishoz való hozzáférés hatékony és egyszerű legyen, egyetlen példány használatával.

```
abstract class AppDatabase : RoomDatabase() {  
  
    abstract fun machineItemDAO(): MachineItemDAO  
  
    companion object {  
        private var INSTANCE: AppDatabase? = null  
  
        fun getInstance(context: Context): AppDatabase {  
            if (INSTANCE == null) {  
                INSTANCE = Room.databaseBuilder(context.applicationContext,  
                    AppDatabase::class.java, name: "machine.db")  
                    .build()  
            }  
            return INSTANCE!!  
        }  
  
        fun destroyInstance() {  
            INSTANCE = null  
        }  
    }  
}
```

- **getInstance():** Létrehozza és visszaadja az adatbázis példányát. Ha az adatbázis már létezik, visszaadja a meglévő példányt.
- **destroyInstance():** Az INSTANCE értékét null-ra állítja, ezzel lehetőséget adva egy új adatbázispéldány létrehozására.

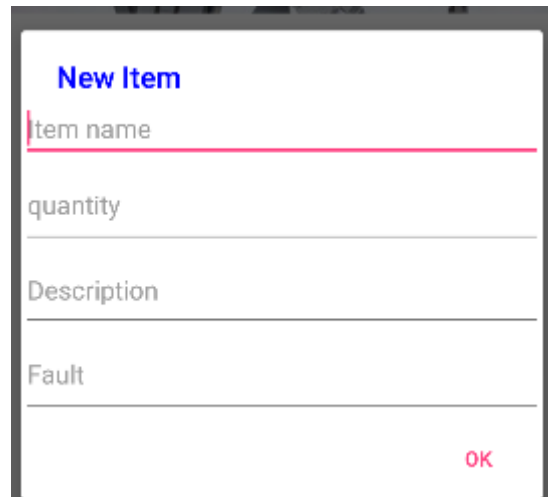
Implementációk

A *MachineAdapter* osztály kezeli az adatbázisban tárolt elemek megjelenítését és interakcióját a felhasználói felületen. A *RecyclerView* Adapter bővítményeként megjeleníti az adatokat, és különböző eseménykezelőket biztosít az elemek törlésére, szerkesztésére és állapotának frissítésére.

Eseménykezelők

- **Elem hozzáadása:** Az *addItem()* metódus felelős az új elemek listához való hozzáadásáért, és értesíti a *RecyclerView*-t a frissítésről.

```
fun addItem(item: MachineItem) {  
    items.add(item)  
    notifyItemInserted(items.lastIndex)  
}
```



- **Elem törlése:** A *deleteItem()* metódus eltávolít egy elemet a listából és az adatbázisból.

```
fun deleteItem(position: Int) {  
    val dbThread = Thread {  
        AppDatabase.getInstance(context).machineItemDAO().deleteItem(  
            items[position])  
        (context as MainActivity).runOnUiThread{  
            items.removeAt(position)  
            notifyItemRemoved(position)  
        }  
    }  
    dbThread.start()  
}
```

- **Elem frissítése:** Az *updateItem()* metódus egy meglévő elem adatait frissíti a listában és az adatbázisban.

```
fun updateItem(item: MachineItem) {
    val idx = items.indexOf(item)
    items[idx] = item
    notifyItemChanged(idx)
}
```

onBindViewHolder() metódus – Elemek megjelenítése és események kezelése

A metódus kezeli az elemek adatainak megjelenítését a felületen, és eseménykezelőket biztosítja a törlés, szerkesztés és állapotváltoztatás funkciókhoz.

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.tvName.text = items[position].name
    holder.tvQuantity.text = items[position].quantity.toString()
    holder.cbAktiv.isChecked = items[position].aktiv
    holder.tvDescript.text = items[position].descript
    holder.tvFault.text = items[position].descript

    holder.btnDelete.setOnClickListener { it: View!
        deleteItem(holder.adapterPosition)
    }
    holder.btnEdit.setOnClickListener { it: View!
        (holder.itemView.context as MainActivity).showEditItemDialog(
            items[holder.adapterPosition])
    }
    holder.cbAktiv.setOnClickListener { it: View!
        items[position].aktiv = holder.cbAktiv.isChecked
        val dbThread = Thread {
            AppDatabase.getInstance(context).machineItemDAO().updateItem(items[position])
        }
        dbThread.start()
    }
}
```

6. Activity és Adapter közötti kapcsolat

A *MainActivity* és a *MachineAdapter* közötti kapcsolat biztosítja, hogy az adatbázisban tárolt elemek megjelenjenek a felhasználói felületen, és lehetővé teszi a felhasználó számára az elemek interaktív kezelését (pl. törlés, szerkesztés).

- **RecyclerView inicializálás:** A *MainActivity* meghívja az *initRecyclerView()* metódust, amely betölti az adatokat az adatbázisból és beállítja a *MachineAdapter*-t a *RecyclerView*-hoz.
- **Eseménykezelők:** Az adapter biztosítja az egyes elemek eseménykezelőit, amelyek az *onBindViewHolder()* metódusban kerülnek beállításra.
- **Dialógus kezelés:** A *MainActivity* a *showEditItemDialog()* metóduson keresztül jeleníti meg az elemek szerkesztési felületét.

7. Összegzés

Az alkalmazás korántsem teljes, de az alapvető adatbázis műveletekkel már egész jó alapot ad a további fejlesztésekhez. Jelenlegi formájában is alkalmazható a gépek adatainak nyilvántartására, és hibajelzések kezelésére.

Az alkalmazás bővíthető további funkciókkal és tulajdonságokkal, hogy még jobban támogassa a felhasználók igényeit:

- További tulajdonságok a gépekhez: Lehetőség van extra mezők hozzáadására, mint például gyártási év, üzemóra, karbantartási dátum, garanciális információk.
- QR kódos vagy vonalkódos azonosítás: A gépek egyedi azonosításához beépíthető egy scanner funkció, amely QR-kód vagy vonalkód beolvasásával gyorsan azonosítja a gépeket. Ez a bővítés lehetővé tenné a gépek gyorsabb beazonosítását és nyilvántartását.
- Automatikus állapotfrissítés: Integrálható olyan funkció, amely meghatározott időközönként automatikusan frissíti a gépek állapotát (például üzemidő alapján), és jelzi, ha egy gép karbantartásra szorul.
- Exportálási és riportálási lehetőségek: Az alkalmazás bővíthető olyan lehetőségekkel, amelyekkel riportok készíthetők és exportálhatók, például PDF vagy Excel formátumban, a gépek adatainak könnyebb áttekinthetősége érdekében.