Mobil Programozás Beadandó Feladat – Jegyzőkönyv

Készítette: Lakatos Tamás

Neptun: S45FXF

Dátum: 2026. január 3.

1. Bevezetés

Az alkalmazás célja: Munkavállalók adatainak (név, pozíció, fizetés, tapasztalat, email) nyilvántartása egy modern, átlátható felületen. Főbb funkciók:

Listázás: Az összes rögzített munkavállaló megjelenítése.

Új felvétel: Új dolgozó hozzáadása az adatbázishoz.

Szerkesztés: Meglévő adatok módosítása.

Törlés: Munkavállaló eltávolítása a listából (elhúzással).

Keresés: Gyorskeresés név alapján.

Felhasznált technológiák:

Nyelv: Kotlin

Platform: Android SDK

Adatbázis: Room ORM (SQLite absztrakció)

UI: RecyclerView, CardView, FloatingActionButton, DialogFragment, Material Design.

2. Adatbázis Réteg (Room ORM)

Az alkalmazás lelkét a Room adatbázis adja, amely lehetővé teszi az adatok perzisztens tárolását a készüléken.

2.1. Adatmodell (Employee.kt)

Létrehoztam egy Employee nevű data class-t, amely az adatbázis employee tábláját reprezentálja. Az osztály implementálja a Serializable interfészt, hogy az objektumokat könnyen átadhassam a komponensek (pl. Activity és Dialog) között.

```
@Entity(tableName = "employee")
data class Employee(
   @PrimaryKey(autoGenerate = true) var employeeId: Long?,
```

```
    @ColumnInfo(name = "name") var name: String,
    @ColumnInfo(name = "position") var position: String,
    @ColumnInfo(name = "salary") var salary: Int,
    @ColumnInfo(name = "experience") var experience: Int,
    @ColumnInfo(name = "email") var email: String
) : Serializable
```

2.2. Adat hozzáférés (EmployeeDAO.kt)

A DAO (Data Access Object) interfészben definiáltam az SQL műveleteket. Itt határoztam meg a lekérdezéseket, beszúrást, törlést és frissítést.

```
@Dao
interface EmployeeDAO {
    @Query("SELECT * FROM employee")
    fun findAllEmployees(): List<Employee>

    @Query("SELECT * FROM employee WHERE name LIKE '%' || :search || '%'")
    fun findEmployeesByName(search: String): List<Employee>

    @Insert
    fun insertEmployee(employee: Employee): Long

    @Delete
    fun deleteEmployee(employee: Employee)

    @Update
    fun updateEmployee(employee: Employee)
}
```

2.3. Adatbázis Példány (AppDatabase.kt)

Ez az absztrakt osztály kapcsolja össze az entitásokat és a DAO-t. Singleton mintát alkalmaztam, hogy az alkalmazás futása során csak egyetlen adatbázis-kapcsolat éljen. Fontos megjegyezni, hogy az exportSchema = false beállítást használtam a build figyelmeztetések elkerülése érdekében.

3. Felhasználói Felület és Megjelenítés

3.1. Fő Activity (MainActivity.kt)

Ez az alkalmazás belépési pontja. Itt inicializálom a RecyclerView-t, és itt kezelem a menüt (keresés) valamint a Floating Action Button-t (új hozzáadás). A initRecyclerView metódusban gondoskodtam arról, hogy ha az adatbázis üres, automatikusan feltöltődjön mintaadatokkal (John Doe, Jane Smith, stb.), így az alkalmazás első indításkor sem üres. 3.2. Lista Adapter (EmployeeAdapter.kt) Az adapter felelős az adatok összekötéséért a nézettel. A ViewHolder osztályban referenciákat tárolok a row_item.xml-ben definiált UI elemekre (TextView-k, Gombok). Itt implementáltam a törlés és szerkesztés gombok eseménykezelőit is.

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val employee = items[position]
    holder.tvName.text = employee.name
    // ... többi mező beállítása ...

    holder.btnDelete.setOnClickListener {
        deleteItem(holder.adapterPosition)
    }
    holder.btnEdit.setOnClickListener {
        (holder.itemView.context as MainActivity).showEditEmployeeDialog(employee)
    }
}
```

4. Funkciók Megvalósítása

4.1. Új felvétel és Szerkesztés (EmployeeDialog.kt)

Egy DialogFragment-et használtam az űrlap megjelenítésére. Ez az osztály kezeli mind az új felvételt, mind a szerkesztést. Új felvétel: Üres mezőkkel nyílik meg. Szerkesztés: A Bundle-ben kapott Employee objektum adataival tölti fel a mezőket. A getSerializable metódusnál figyeltem a visszafelé kompatibilitásra, így régebbi Android verziókon is stabilan fut.

```
// Adatok betöltése szerkesztésnél
val arguments = this.arguments
if (arguments != null && arguments.containsKey(MainActivity.KEY_ITEM_TO_EDIT)) {
    val employee = arguments.getSerializable(MainActivity.KEY_ITEM_TO_EDIT) as
Employee
    etName.setText(employee.name)
    // ...
}
```

4.2. Törlés és Szálkezelés

A törlés funkciót a deleteItem metódusban valósítottam meg. Mivel az adatbázis műveletek időigényesek lehetnek, ezeket külön szálon (Thread) végzem, hogy ne akasszam meg a felhasználói felületet (UI Thread). A sikeres törlés után a runOnUiThread segítségével frissítem a listát.

```
fun deleteItem(position: Int) {
   val employeeToDelete = items[position]
   val dbThread = Thread {
      // Adatbázis művelet a háttérben

AppDatabase.getInstance(context).employeeDao().deleteEmployee(employeeToDelete)

      // UI frissítés a főszálon
      (context as MainActivity).runOnUiThread {
         items.removeAt(position)
         notifyItemRemoved(position)
      }
   }
   dbThread.start()
}
```

4.3. Keresés

A MainActivity-ben implementáltam a SearchView.OnQueryTextListener-t. Amikor a felhasználó gépel, a filterItems metódus fut le, amely egy SQL LIKE lekérdezést hajt végre a háttérszálon, majd frissíti az adaptert az eredményekkel.

5. Összegzés

Az elkészült alkalmazás teljesíti a kiírt követelményeket. Stabilan kezeli az adatbázis kapcsolatot, a felhasználói felület reszponzív a megfelelő szálkezelésnek köszönhetően, és modern Android komponenseket használ. A fejlesztés során külön figyelmet fordítottam a hibatűrésre (pl. üres adatbázis kezelése) és a kód karbantarthatóságára.