Dokumentáció

1. Felhasználói dokumentáció

Az alkalmazás, egy egyszerű teendő lista (todo) app. Használata egyszerű, mégis sokat segíthet a napi feladatok észben tartásánál. A felső szövegmezőben egy leírást lehet készíteni a teendőről, majd a Hozzáad gomb megnyomásával bekerül a listába.

Amennyiben valamelyiket teljesítettük, úgy törölni lehet a szöveg melletti Törlés gomb megnyomásával, így átláthatóbb lesz a lista.

Az alkalmazás csak portré módban, azaz álló helyzetben használható.

2. Fejlesztői dokumentáció

Öt osztály, és két felületet leíró xml található a forráskódban.

2.1. MainActivity.java

2.1.1. onCreate()

Az onCreate() metódusban példányosítom az adatbázist kezelő osztályt, illetve a ListView-katt, melyekről a későbbiekben bővebben is írok. Ezek után meghívásra kerül az updateUI() metódus. Végül pedig a dátumválasztó eseményére íratkozok fel, ami visszaadja a kiválasztott évet, hónapot és napot.

2.1.2. updateUI()

A list privát változóba lekérem az adatbázisban lévő összes feladatot, majd ezt a listát átadom a TodoAdapter osztálynak, amely feldolgozza, és végül megjeleníti.

2.1.3. addTaskNow()

Ez a függvény a Hozzáad gomb megnyomására megkeresi a beviteli szövegmezőt, és kinyeri belőle a beírt szöveget. Amennyiben ez a szöveg üres, akkor egy Toast üzenetet küld, hogy írja be a felhasználó a teendő leírását. Ha beírta, és megnyomta a gombot, akkor példányosít egy Task osztályt, és átadja neki a beírt szöveget, majd elmenti az adatbázisba. Végül egy Log bejegyzést készít, kiüríti a beviteli mezőt és frissíti a felületet.

Emellett a beállított dátum is átadásra kerül az adott Tasknak. Amire figyelni kell, az hogy ha kezdetben nincs beállítva dátum, akkor mindig az adott napot fogja alapértelmezetten beállítani, viszont ha már be lett állítva egy, akkor amíg ez nem lesz változtatva, addig minden új feladathoz az kerül be.

2.1.4. deleteTask()

Bármely törlés gomb megnyomására megkeresi az adott gomhoz tartozó task_id-t mely egy egyedi kulcs az adatbázisban. Ezt az értéket átadja az adatbázist kezelő osztály deleteTask metódusának, mely törli a táblából. Végül frissíti a felületet, így látható, hogy kikerült a táblából is az elem.

2.1.5. getDate()

Ebben a metódusban a felületre kihelyezett dátum választó gomb megnyomásakor megjelenő felugró ablak létrehozását és megjelenítését kezelem. Gyakorlatilag annyi történik, hogy az adott napi (mai) dátumot jelenítem meg.

2.1.6. getTodayTasks()

Átadásra kerül az összes teendőt tartalmazó lista, amit ebben a metódusban átnézek. Azaz kiválogatom azokat az elemeket, melyek beállított dátuma a mai napra szól.

2.1.7. getTomorrowTasks()

Ebben a metódusban a holnapi dátumokra szűrök az átadott listából.

2.1.8. getOtherTasks()

Ebben a metódusban pedig azt vizsgálom, hogy a beállított dátum megegyezik-e a maival vagy a holnapival. Ha egyikkel sem, akkor kerül kiválasztásra az elem.

2.2. Task.java

Ez az osztály reprezentálja a Task (feladat) objektumot. Két privát adattagja van, ezek a taskName (feladat neve/leírása) és az id (egyedi azonosító).

Néhány egyszerű metódust találunk benne, melyekkel kinyerhetők az adattagok értékei, illetve beállíthatók azok (get, set).

2.3. TodoAdapter.java

Ez az osztály azért került létrehozásra, mert a felületre valahogy ki kellett helyezni a feladat leírását, és egy láthatatlan mezőbe a feladat egyedi azonosítóját. Erre azért volt szükség, mert ha két teljesen azonos leírás kerül a listába, és valamelyiket törölni szeretnénk, akkor azt a legkönnyebben úgy tudjuk megtenni, hogy az egyedi azonosító alapján távolítjuk el a táblából. Ezt a getView() metódusban készítettem el, ahol egyesével kerülnek kiírásra a listában szereplő elemek.

2.4. DbHelper.java

Ebben az osztályban valósítottam meg az SQLite adatbázishoz való kapcsolódást, illetve a CRUD (Create, Read, Update, Delete) műveleteket.

Az osztály elején létrehoztam néhány statikus változót, melyek az adatbázis főbb információit tartalmazzák. Erre azért volt szükség, hogy ne kelljen minden esetben leírni a tábla és mezők nevét, így minimalizálva a félre gépelésből adódó hibákat. Ezek a mezők az alábbiak:

- DATABASE_VERSION melynek értéke 1, ezt a példányosításkor kell használni
- DATABASE_NAME, ami az adatbázis nevét tartalmazza
- TABLE_TASK, ami a tábla nevét tartalmazza
- KEY_ID, az egyedi kulcs oszlopának nevét tartalmazza
- KEY_TASKNAME, a feladat nevét tartalmazó oszlop neve

2.4.1. onCreate()

Ez a függvény minden példányosításkor lefut, és amennyiben nem létezik még a tábla, akkor létrehozza azt, a fenti két mezővel. Fontos kiemelni, hogy az ID mező értéke autoincrement tulajdonsággal bír, így nem kell azzal törődni, hogy milyen érték kerül be, mert az adatbázis motor ezt automatikus elvégzi.

2.4.2. onUpgrade()

Ez is minden esetben lefut, és amennyiben verzióváltás történik, akkor törli a régi táblát, és meghívja az onCreate() függvényt, mely létrehozza az újat.

2.4.3. addTask()

Lekérdezi az írható táblákat, majd egy ContentValues típusú változóba beírja, hogy melyik mezőbe szeretnénk eltárolni a megadott feladat leírást, ezt is eltárolja a változóban, végül az insert metódus meghívásával beírja a táblába.

2.4.4. getAlltasks()

Egy feladatokat tartalmazó listával tér vissza. Gyakorlatilag lekérdezi az összes, tasks táblában lévő sort, majd ezeket egyesével 1-1 Task osztályba rendezi, és ezeket az osztályokat adja át a visszatérési listának.

2.4.5. updateTask()

Ezt a függvényt csak azért hoztam létre, hogy ha később tovább fejlesztem az alkalmazást egy olyan funkcióval, melyben módosíthatom az adott feladat tulajdonságait, akkor már csak meghívni kelljen ezt a metódust.

Itt annyi történik, hogy megkeresi ID alapján, hogy melyik Taskról van szó, és módosítja a megfelelő mezőket.

2.4.6. deleteTask()

ID alapján megkeresi a megfelelő sort a táblában, és törli azt.

2.5. NoScrollListView.java

Ezt az osztályt azért hoztam létre, mert a felületen 3 ListView található, amik egy NestedScrollView-ban vannak és ha több elem volt egy ListView-ban, akkor is csak egy jelent meg, mert a többit csak scrollozás után lehetett látni. Ez az megoldja, hogy ne lehessen scrollozni, így fixen egymás alá kerülnek az elemek.

Ahhoz, hogy meg tudjam írni ezt az osztályt, az alábbi segítséget használtam: https://stackoverflow.com/questions/18813296/non-scrollable-listview-inside-scrollview

2.6. activity_main.xml

Ez a főoldal felületének leírója. Az egész felület egy RelativeLayout, melyben 4 fő elem van. Az első egy EditText, amiben megadhatjuk a feladat leírását. A második a dátum beviteli gomb, a harmadik pedig a Hozzáad gomb. Ezekről bővebben nem írnék, egyértelmű a feladatuk, illetve korábban már említésre kerültek. A negyedik fő elem egy NestedScrollView, mely egy LinearLayout-ot tartalmaz. Ebben a LinearLayoutban pedig 3 TextView, és 3 NoScrollListView található. A TextView-k tartalma statikus szüveg. A NoScrollListView-k tartalma pedig a todo_list.xml több példánya, amennyiben több feladat is el van tárolva az adatbázisban.

2.7. todo_list.xml

Ennek a felépítése egyszerűbb. Egy RelativeLayout-ban található 2 TextView és egy gomb. Az első TextView-ba kerül az adott feladat szövege, a második pedig egy láthatatlan elem, melybe a feladat egyedi azonosítója kerül. Ennek a lényege, hogy mikor a Törlés gombra kattint a felhasználó, akkor pontosan be lehesssen azonosítani, hogy melyik sort kell törölni a táblából.

2.8. Egyéb módosítások

A drawable mappába bekerült egy calendar.png fájl, ami a dátumválasztó ImageButton-on jelenik meg.

3. Források

https://androstock.com/tutorials/create-a-todo-task-app-on-android-android-studio.html

https://guides.codepath.com/android/Basic-Todo-App-Tutorial

https://stackoverflow.com/questions/18813296/non-scrollable-listview-inside-scrollview

https://developer.android.com/docs/