Szakképesítés megnevezése: **Szoftverfejlesztő**

OKJ száma: **54 213 05**

ZÁRÓDOLGOZAT

Light Airlines – Android applikáció

Merényi Miklós

témavezetőNyírő Levente Gyula

14.S

Budapest, 2020

*Nyilatkozat*

*Alulírott Nyírő Levente kijelentem, hogy ez a záródolgozat saját tudásom, önálló munkám terméke.*

*Kijelentem, hogy a záródolgozat beköttetett és elektronikus formában leadott példányai mind formátumban, mind tartalomban egyezőek, eltérést nem tartalmaznak.*

*Dátum:*

*hallgató aláírása*

Tartalom

[1. Bevezetés 4](#_Toc31453033)

[2. Témaválasztás 6](#_Toc31453034)

[3. Fejlesztői dokumentáció 8](#_Toc31453035)

[3.1 Az alkalmazott fejlesztői eszközök 8](#_Toc31453036)

[3.2 Adatmodell leírása 9](#_Toc31453037)

[3.3 Részletes feladatspecifikáció 11](#_Toc31453038)

# 1. Bevezetés

A 2 éves szoftverfejlesztői képzésem keretében, az utolsó évben be kell mutatnom egy általam szabadon választott témát, ezt megtervezni és megvalósítani a rendelkezésre álló eszközökkel. Ennek a produktumát tartja most a kezében, ahol leírom részletesen a témaválasztásomat, fejlesztésem kitűzött céljait, a megvalósított programom jellemzőit, a felhasználóknak szóló dokumentációt, ahol a belső funkcionalitásokra reflektálok, illetve a fejlesztőknek szánt részletes bemutatást, amelyben kiderül, hogyan is készült az applikáció minden egyes funkciója, ezen belül bemutatva a metódusokat és a programkód részleteit.

A választásom a mobilapplikáció fejlesztésre esett a többi lehetőség közül, ugyanis kíváncsi vagyok erre a technológiára, szeretném magam minél jobban kiismerni, illetve ezen az irányvonalon kívánok tovább haladni a jövőben. Ezekkel a feltételekkel adott volt a kérdés, hogy Android platformra kívánom készíteni a záródolgozatom.

Vessünk egy pillantást a mobilapplikáció-fejlesztés hasznosságára. A mai felgyorsult világunkban a mobiltelefon egyre inkább, sőt már szerves részünké vált. Ott van velünk a legrövidebb útjaink során, de útitársnak is kiváló. Az ember nem minden környezetben tudja használni a számítógépét, hogy egy szimpla adatra rátaláljon az interneten, ezért volt szükség egy olyan hordozható számítógépre, ami viszonylag kicsi, kompakt, elfér a zsebünkben és a lehetőségek végtelen tárházát tudjuk vele végrehajtani. Ezért is látok nagy potenciált a mobilapplikációkban, mert ezek a legtöbbet használt programok, amik mindig velünk vannak.

A szakdolgozati mobilapplikációm a fentieket figyelembe véve készült. Ez az egész projekt programnyelvet tekintve Java és XML ötvözete, mindez kialakítva az Android Studio IDE segítségével. A design is fontos része az applikációnak, ami főképp az Adobe termékek, azaz Illustrator, Photoshop és XD felhasználásával valósult meg. Az adatbázis részénél az SQLite-ot választottam. Ennek a kezelésére, a kapcsolatok vázolására és a benne szereplő egyes adatok ellenőrzésére a JetBrains által fejlesztett DataGrip nevű programra támaszkodtam, illetve a PhpMyAdmin is sokat segített a munkámban.

A programom lényege egy fiktív légitársaság menetrend keresési, illetve helyfoglalási felület, amelyben lehetőség nyílik a saját profil kezelésére, belépés előtt, valamint azután is. Járatok között lehet böngészni és információkat megtekinteni, majd az információk alapján foglalást tenni, a felhasználó által kiszemelt ülőhelyre. Az előző lépések után elkészül a „Boarding pass”, azaz a beszállókártya. Ezen a felületen lehet az adatokat megtekinteni, illetve a foglalást lemondani.

Rengeteg kitűzött cél volt a fejlesztésem megkezdése előtt. Mindenképp szerettem volna bizonyítani magamnak, hogy képes vagyok felépíteni egy olyan mobilapplikációt, amelyben nagy hangsúly van a kinézetbe, illetve a funkcionalitásba fektetve a legkifinomultabb módon, és ehhez a jelen szakdolgozat egy kiváló lehetőség volt. Célomnak tekintettem az Android Studio rutinos használatát, amelyet a fejlesztés során sikerült elérnem, valamint a Java programnyelvben is sok tapasztalatra tehettem szert. A fejlesztés során nagyon sok problémára bukkantam, majd ezekre sikerült megoldást találnom, amivel folyamatosan nőtt az önbizalmam a programozással kapcsolatban. Mivel egy általam frissen tanult programozási környezethez volt szerencsém, ezért nagyon sok sikerélmény és érdekes kutatás tudott engem érni a fejlesztés során.

# 2. Témaválasztás

Sok gondolkodás és ötletelés során már októberben előkerült egy tervem, hogy egy légitársasággal kapcsolatos applikáció lenne a hozzám illő projekt. Persze volt nagyon sok más ötlet is. Példákat nem említve biztos voltam abban, hogy valamilyen menetrenddel kapcsolatos applikáció megvalósítása volna a célom.

Mivel közel áll hozzám a repülés és az utazás világa, ezért sokszor van szerencsém repülőjegy foglalásokkal foglalkozni, így megihletett a terv, hogy lemodellezzem egy légitársaság applikációjának belső rendszerét. -Érdekel egy ilyen cégen belüli szervezések és adatáramlások. Az a cél is szintén vezérelt, hogy ez az applikáció minél átláthatóbb legyen a rendelkezésemre álló UI design és backend segítségével. El is vállaltam a feladatot, miután benyújtottam az ötletemet.

1. ábra Light Airlines logó

Mérvadó volt számomra a döntés során, hogy minél jobban ki tudjam fejteni mire is vagyok képes az arculattervezés világában. Nagyon sokat kutattam és vizsgálódtam ebben a témában. Megvalósítottam a program arculatát, az Activity-kben a háttereket és hozzá az építőelemet, azaz View-ok kinézetét, hogy minél könnyebben átlátható és használatra is letisztult alkalmazást sikerüljön készítenem. A szakdolgozatomnál lényeges volt, hogy minden elem a saját elképzelésem és alkotásom alapján készüljön el. Így lett az általam kreált fiktív légitársaság neve Light Airlines. Az arculathoz szorosan hozzátartozik egy logó, aminek az végleges verziója az ábrán látható.

Fontos kihangsúlyozni egy ilyen project keretében az applikáció funkcionalitását is, ugyanis ez fontos összetevője a programnak, ami nagy mértékben befolyásolja a minőséget is. Figyelmet kell szentelni az alkalmazás logikai felépítésére és az activity-k közötti kohézióra is. Ezzel az elhatározással készítettem el az oldalakat és a kapcsolatokat, hogy minél átláthatóbb és egyszerűbb legyen használni az applikációt egy kezdő felhasználó számára is. Ez is közrejátszott a témaválasztásomban, ugyanis célom volt megtanulni, miképp lehet egy Android alkalmazás backend funkcióit megvalósítani a legkifinomultabb eszközökkel.

Az eddigi tanulmányaimban webprogramozásban volt szerencsém találkozni a frontend, adatbázis és háttérbeli funkciók megvalósításával, így egy kicsit szkeptikusan álltam ehhez a feladathoz Androidon. Az egy biztos tény, hogy az elhivatottságom sokat segített a szakdolgozatnál, illetve az ehhez kapcsolódó tudás elsajátításához.

# 3. Fejlesztői dokumentáció

## 3.1 Az alkalmazott fejlesztői eszközök

A program fejlesztésénél az egyik alapja mindennek, a rendelkezésre álló eszközök meghatározása. Tudnunk kell milyen fejlesztői környezet, programnyelv, illetve program kell. Ez a fontos tényező már a programom megtervezése elején eldöntött kérdésnek minősült.

Programnyelv tekintetében a Java-ra támaszkodtam, ugyanis ez volt az a programnyelv, amit a mobilapplikáció fejlesztésben tanultam, valamint a tudásomat is biztosnak érzem ebben a programnyelvben. Az Android programozáshoz tartozik egy leírónyelv, amely főként a frontend elemekre összpontosul, ez ebben az esetben XML.

Fejlesztői környezetnek Android Studio legfrissebb, 3.5-ös verzióját használtam, ugyanis a tanulmányaim során ez volt az egyetlen meghatározó IDE, amivel mobilapplikációkat készítettem.

Az applikáció adatbázis típusa SQLite, ami lokális szerverként funkcionál az eszközön. Magát az adatbázist Android Studio-ban alakítottam ki, azonban támaszkodtam más programokra is az elkészítéskor és a kapcsolatok meghatározásakor. Egész pontosan 2 szoftverrel éltem: az egyik a JetBrains által fejlesztett DataGrip, a másik a PhpMyAdmin nevezetű program. Itt készítettem el a táblákat, a kapcsolatokat, beillesztéseket, illetve a lekérdezéseket is, majd az SQL kódokat bemásoltam az Android Studio-ba, amivel ki tudtam alakítani a lokális adatbázist, valamint a listázásokat is, ahol erre szükség volt.

Ezeken kívül nagy szükség volt a dizájn elkészítésére, amihez kép-, illetve vektorgrafikus szerkesztőkre kellett támaszkodnom. Ezekhez az Adobe termékeket használtam, így a Photoshop, Illustrator és az Xd segítségével sikerült kialakítanom az elvárt eredményt. Az előbb felsorolt programok egy részét tanulmányaim során, egy részét pedig önálló módon sajátítottam el.

## 3.2 Adatmodell leírása

2. ábra Az adatbázis szerkezete

Ebben az alfejezetben kifejezetten az adatbázisról és a köztük lévő kapcsolatokról lesz szó, valamint arról, hogyan lehetett mindezt minél jobban normalizálni.

Két nagy csoportja volt az adatbázisnak a fejlesztés elején: az egyik a *user* tábla, ahol a felhasználók adatai tárolódnak, a másik fő tábla a járatok adatait tartalmazó „járat” tábla. A két tábla közül az utóbbinak a normalizálására volt szükség az adatredundanciák kizárására. Ebből a célból hoztam létre további 2 táblát *utvonal* és *airport* néven. Ezeken kívül létezik még egy tábla, amely a foglalásokat tartalmazza, benne a felhasználó és a járat hivatkozásával. Minden tábla azonosítására egy integer típusú *id* nevű primary key tulajdonságú elem szolgál.

A két alaptábla közül a felhasználók adataival foglalkozó táblát a *user* nevet adtam. Ez a tábla rendkívül fontos a regisztrációnál, belépésnél, azonosításnál, illetve személyre szabásnál. Ide tartozik két unique tulajdonságú elem, és ennek köszönhető, hogy az adott elemből kizárólag csak egy szerepelhet a táblán. Ez a két adatmező *username* és *email* nevet kapta. Ezek az adatok a belépésnél kardinális kérdésnek számítanak, ugyanis a felhasználó nevet, vagy az E-mail címet feltétlenül tudni kell a belépéshez. A tábla rendelkezik továbbá a *firstname* és a *lastname* nevű adatmezőkkel, amelyeknek célja, hogy az applikáció személyesebbé legyen téve. A keresztnév például visszaköszön a bejelentkezés utáni activityben, ahol az „Üdvözöljük!” üzenet jelenik meg. Az utóbb felsorolt négy adatmező mindegyike varchar típusú és 200-as hosszal rendelkezik. Létezik egy születési dátumot is letároló adattag *birthdate* néven, amelynek azért van fontos tulajdonsága, mert a regisztrációnál ellenőrzésre kerül, hogy a felhasználó idősebb-e a meghatározott korhatárnál, azaz 13 évnél. Erre az adattagra date tulajdonságot állítottam be, ezzel csak a dátumot eltárolva YYYY-MM-DD formátumban. Az utolsó adatmező ebben a táblában a jelszó eltárolására szolgál *password* név alatt. Erre egy 255 hosszú varchar tulajdonságot szántam. Fontos tudni, hogy a jelszó nem eredeti formájában tárolódik el ebben a mezőben, hanem titkosított formában a salting-al összevegyítve, így megakadályozva a jelszavak eltulajdonítását.

A másik alap tábla a *jarat* tábla, amely minden egyes járat adatait hívatott leképezni és eltárolni. A normalizálás következménye miatt ez egy viszonylag kevés adatmezővel rendelkező tábla, azonban közvetlenül és közvetetten is kapcsolódik hozzá tábla. Mit is nevezünk járatnak az applikációban? Egy repülőjárat az, amely egy adott útvonalat teljesít egy előre meghatározott időpontban adott számú férőhellyel. Így lett hát az első adattag az *utvonal\_id*, ami az előre meghatározott repülési útvonalat határozza meg „A” pontból „B” pontba. Ez az adatmező kapcsolódva van az *utvonal* tábla *id* azonosítójához, ezzel importálva az adott rekord adatait. Ezen kívül van egy másik integer típusú elem, amely a *helyek\_szama* nevet kapta. Ez mutatja meg, hogy az adott járaton indított repülőgépen, mennyi a maximálisan elfoglalható helyek száma (ez a szám alapesetben 120). Ez akkor jelenik meg az applikációban, amikor kivonjuk ebből a számból az eddig foglalt helyek összegét és visszaadjuk, hogy mennyi foglalható hely létezik még az adott járat repülőgépén. Az utolsó adattag ebben a táblában egy datetime típusú *idopont*, ami megmutatja, hogy a gép pontosan mikor indul. Az adatbázisban ez YYYY-MM-DD HH:MM:SS formátumban van eltárolva, azonban az applikációban való megjelenéskor le van splitelve róla a másodperc, ugyanis nincs szükségünk ekkora pontosságra.

Az előző táblát követi két normalizált tábla, amelyből az első az *utvonal* nevet kapta. Egy útvonal megmutatja, hogy honnan-hova tarthat egy járat és ezt az időt mennyi idő alatt teljesíti egy előzetes becslés szerint. Itt két integer tulajdonsággal rendelkező adatmező is hivatkozik ugyanarra az *airport* tábla azonosítójára. Ez a két adatmező az *indulas\_id*, ami az indulási repülőteret, és a *celallomas\_id*, ami az érkezési repülőteret azonosítja és határozza meg. Ez egy alapinformációja egy járatnak. Létezik még egy time típusú adatmező *idotartam* névvel, amely megmutatja az applikációban, hogy mennyi idő alatt teljesíti az utat a repülőgép. A formátuma HH:MM:SS, azonban az applikációban eltűnik a másodperc, mert itt sem egy mérvadó információról van szó.

A másik normalizált tábla *airport* névvel szerepel. Ez a tábla csak azt mutatja meg, hogy egy repülőtérnek mi a neve és a kódja. Mind a két adatmező unique tulajdonsággal rendelkezik és varchar. Az első adatmező *nev* alatt fut 100-as hosszúsággal, míg a másik *rovidites* névvel szerepel, amelynek 3 karaktert kell tartalmaznia.

Az adatbázisban szereplő utolsó tábla nagyon fontos kérdéseket lát el. Ez foglalas névvel rendelkezik és a feladata, hogy megmutassa, melyik felhasználó melyik járatra rendelkezik ülőhellyel. Az első adatmező a *jarat\_id*, amely hivatkozik a *jaratok* tábla azonosítójára ezzel megtudva, hogy melyik járatra érvényes a foglalás. Ez akkor lesz fontos, amikor kilistázzuk, hogy hol vannak foglalt ülőhelyek a repülőgépen egy adott járaton. A másik adatmező a *user\_id*, amely a *user* tábla adataira hivatkozik annak *id*-ján keresztül. Ennek segítségével kilistázhatjuk egy adott felhasználó jegyeit a járatokra. Az admin felhasználónak is nagy segítség, hogy ki vett jegyet egy adott járaton, bizonyos ülőhelyekre, ezáltal összeállítva egy utaslistát, ami a légitársaságoknak egy rendkívül fontos dokumentum. Az utóbb felsorolt két adatmező mindegyike integer típusú. A harmadik és egyben utolsó adatmező az ules, amely megmutatja, hogy az adott foglalás melyik kiválasztott ülőhelyre szól. Ez egy varchar típusú és maximum három karakterrel rendelkező adatmező, ugyanis az adat lehet „8D”, de akár „15B” is. Az utóbbi példa azt jelenti, hogy a 15. sor 2. üléséről van szó.

Minden előbb felsorolt tábla nagyon fontos komponense az applikációnak. Nincs olyan tábla az adatbázisban, amelyre nem lenne még minimálisan se szükség. Mindez maximálisan normalizált, így az alkalmazás is bökkenőmentesen tudja az adattranzakciókat kezelni.

## 3.3 Részletes feladatspecifikáció

Ebben az alfejezetben nagyrészt az applikációban található osztályokról, valamint activitykről szól, amik ténylegesen meghatározzák, hogyan is nézzen ki a szakdolgozat egésze, ebben szót ejtve mind a backend, mind a frontend témájáról. Mindezt érdemes olyan sorrendben nézni, ahogy haladunk végig az applikációban felhasználóként.

Az osztályok több packagebe, azaz csomagba vannak szervezve. Ennek célja az átláthatóság programozás közben. Öt package van: adminActivity, ahol az admin felület osztályai tárolódnak, globalActivity, itt az az osztály található, amely több felületen is használatos, kezdoActivity, ahol a regisztráció és a bejelentkezés működik, segedOsztaly, amelyben azok az osztályok találhatóak, amelyek kiszolgálják az activityket, illetve a userActivity, ahol az átlagfelhasználói felület activityjei szerepelnek..

Elsősorban a segédfüggvényeket és a felépítésüket nézzük át. Ezek mind egy packagebe vannak helyezve, amelynek a neve *segedOsztaly*. Három osztály található itt: *Database*, *Metodus* és *PasswordUtils* néven.

### 3.3.1 Database

A tervezésnél és a fejlesztésnél szükség volt egy olyan osztályra, amely a kommunikációt és az adattranzakciót végzi az adatbázis és az applikáció között. Erre szolgál a *Database* osztály, amely SQLiteOpenHelper osztályt hívja meg, hiszen az applikáció SQLite adatbázist használ minden tekintetben. Itt található egy konstruktor ami egy Context típusú paramétert vár, majd létrehozza az adatbázist lightairlines.db néven. A meghívott osztály metódusai az onCreate, ahol az adatbázis tábláit, a köztük lévő kapcsolatokat definiálom és alapadatokat illesztem be, illetve a másik metódus az onUpgrade, ahol a táblák létezését ellenőrzi, és csak akkor illeszt be új táblát, ha ilyen még nem létezik. Ezeken kívül található még metódus minden eset lekezelésére, ahol select, insert, delete, valamint update utasításra van szükség. Minden ilyen metódus esetében szükség van az alábbi programkódra:

SQLiteDatabase db = this.getWriteableDatebase();

Az SQLiteDatabase osztálynak köszönhetően képesek vagyunk adattranzakciókat végrehajtani. Select esetén az előbb említett osztály rawQuery metódusával tudunk lekérdezést készíteni, amit azonnal vissza tudunk adni Cursor típusú változóként. Insert függvény esetén boolean lesz a visszatérési érték. A ContentValues osztályt kell meghívni, majd a put metódus segítségével megadni, hogy melyik adatmezőbe, milyen adatot szeretnénk behelyezni. Az adatok nagy részét paraméterek segítségével illeszti be az alkalmazás. Inicializálni kell egy *eredmeny* nevű változót long típusban az SQLiteDatabase insert metódusának meghívásával, amelyben megadom, hogy melyik táblába szeretnénk beilleszteni, a második paramétert null értékként inicializáljuk, valamint meg kell adni a ContentValuest, amiben a beillesztendő elemek szerepelnek. A kód lefutása után az *eredmeny* változó -1, 0 vagy 1 értékkel térhet vissza. Az 1 jelenti azt, hogy a beszúrás sikeresen megtörtént. Minden más esetben sikertelen az eset. Így visszaadjuk azt a logikai értéket, hogy az eredmény egyenlő-e egyel, vagyis sikeres volt-e a művelet. Update esetén szintén a ContentValueshoz kell nyúlni, azonban ebben a metódusban az update függvényt hívom meg. A paraméterei a tábla neve, a ContentValues, egy where feltétel, hogy melyik azonosítójú elemet kívánom törölni, amit a metódus paraméteréből kértem le, illetve egy negyedik paramétert, amelyet null értékként inicializáltam. A visszatérési érték típusa szintén boolean lesz és abban az esetben igaz, ha az előbb említett függvény visszatérési értéke 1 lesz. Az utolsó lehetőség a delete metódus. Ebben az esetben is logikai visszatérési értékről van szó. Az SQLiteDatabase osztály delete metódusát meghívva meg kell adnunk, hogy melyik táblából kívánok adatot törölni, ezután a where feltételt kell begépelni, kérdőjellel helyettesítve a változót, amit a harmadik paraméterben határozok meg. Ebben a metódusomban paraméterből kértem le az id-t, ugyanis azonosító alapján töröl a program.

A második segédosztály a *Metodus* class. Ennek az elsődleges célja, hogy globálisan kezeljen minden olyan függvényt és eljárást, amelyek több helyen szerepelnek, ezáltal megóvva a programot az kódredundanciától. Az elején két változót deklarálok: Database és Context osztálybelit, amit a konstruktorban inicializálok, az Context bekérésével paraméterként, ugyanis erre van szükség, ha egy adott metódust össze van kötve a Database osztállyal, amelynek a konstruktora szintén a Context osztályú változót várja. Az első metódusa az osztálynak a dpToPx, ami egy integer típusú változót vár dp néven (dot per inch), valamint egy Resources osztályú változót, amelyet abban az activityben találunk, ahol meghívtuk ezt a metódust. Szintén egy int típusú változót fog visszaadni, mégpedig a dp mértékegységet váltja át pixelre, amelyre szükség van néhány activitynél, ahol programkódban hozzuk létre a Viewokat és azoknak a tulajdonságait, ugyanis az px-ben várja a méretet és a reszponzivitás érdekében fontos ezt meghatározni inkább dp-ben. Található még ebben az osztályban sok más metódus is: usernameHosszEllenorzes, usernameWhiteSpace