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](#_Toc33034368)

[2. Témaválasztás 6](#_Toc33034369)

[3. Fejlesztői dokumentáció 8](#_Toc33034370)

[3.1 Az alkalmazott fejlesztői eszközök 8](#_Toc33034371)

[3.2 Adatmodell leírása 9](#_Toc33034372)

[3.3 Részletes feladatspecifikáció 11](#_Toc33034373)

[3.3.1 segedOsztaly package 12](#_Toc33034374)

[3.3.2 kezdoActivity package 13](#_Toc33034375)

[3.3.3 userActivity package 14](#_Toc33034376)

[3.3.4 adminActivity package 15](#_Toc33034377)

[3.3.5 globalActivity package 16](#_Toc33034378)

[3.4 Tesztelési dokumentáció 16](#_Toc33034379)

[3.5 Továbbfejlesztési lehetőségek 17](#_Toc33034380)

[4. Felhasználói dokumentáció 19](#_Toc33034381)

# 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..

### 3.3.1 segedOsztaly package

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. Konstruktora egy Context tiípusú paramétert vár, amely létrehozza az adatbázist lightairlines.db néven. A meghívott osztály metódusai az onCreate, ahol a táblák adatai valósulnak meg, illetve a másik az onUpgrade: itt a táblák létezése kerül ellenőrzésre. A többi metódus select, insert, update és delete parancsokra vonatkozik, amelyek az activitykben kerülnek meghívásra.

A második segédosztály a Metodus class. Ennek az elsődleges célja, hogy magába foglaljon, minden olyan függvényt és eljárást, amelyek több helyen is szerepelnek, ezzel is megóvva a programot a kódismétlődéstől. A konstruktorban két osztály kerül példányosításra: Database és Context, azonban csak a Context paraméterét kéri be a program. Sok metódus van ebbe az osztályba belefoglalva, ilyen a dpToPx, ami az osztály által várt pixeles értéket ad vissza a dp megadásával. Ezen kívül még található felhasználóhoz kapcsolódó ellenőrzés, a hosszával, valamint, hogy nem található-e benne whitespace, azaz üres karakter. E-mail címmel kapcsolatban a helyessége kerül ellenőrzésre. Jelszónál itt található maga a jelszóellenőrzés folyamata, illetve a regisztrációnál található feladat, amely a jelszó erősségét hívatott vizsgálni. A keresztnév, vezetéknévvel kapcsolatban az elsoNagybetu() metódus szabja formára a helytelenül írt neveket. Található itt még idő átalakítás, valamint üléssel és foglalással kapcsolatos metódus is.

Az utolsó osztály ebben a packageben a PasswordUtils osztály. Itt történik a jelszó titkosítás a getSalt, hash és generateSecurePassword segítségével, illetve belépésnél a visszafejtés, ezzel validálva a jelszó érvényességét a verifyUserPassword felhasználásával.

### 3.3.2 kezdoActivity package

Az activityk csoportosítása úgy történt, hogy bizonyos funkciócsoportok, el legyenek egymástól szeparálva. Így meg kellett határozni, hogy a belépéskor megjelenő activityk hova kerüljenek besorolásra, illetve meddig tart ez a felületcsoport. Így lett végül pontosítva, hogy a kezdő funkciócsoport a betöltő képernyőtől egészen a bejelentkezés véglegesítéséig tart. Ezekhez az activitykhez már XML fájl is tartozik, amelyek a UI-t szolgáltatják.

Sorrendben az első, amivel találkozunk az a betöltő képernyő, ami a LoadingActivity nevet kapta. Ezzel az activityvel kezd maga az applikáció, majd 1,3 másodperc elteltével, amit a Thread osztály késletlet, Intent osztály segítségével átléptet a LoginActivitybe. Az átléptetésig egy animációt látunk a képernyőn, ami a LottieAnimation felhasználásával történt meg.

Ezután a LoginActivityben találjuk magunkat. Itt 4 fontosabb komponens található: a két EditText, valamint két Button. Az egyik gomb segítségével elindítjuk a bejelentkezési procedúrát, ellenőrzéseket, ha minden adat egyezik átkerülünk vagy az adminActivity packagebe, vagy a WelcomeActivitybe, ami az átlagfelhasználók felülete. A másik gombbal a regisztrációs részben találjuk magunkat.

Három RegActivity osztály létezik. Itt lehet lépésenként az adatokat megadni a sikeres regisztrációhoz. Minden ilyen felületnél 3 lehetőség létezik: a továbblépés, a visszalépés, illetve a van már profilom, aminek a segítségével visszatérhetünk a LoginActivitybe. Az elsőnél a felhasználónevet és az e-mail címet kell megadni. A tovább gombbal a metódusok ellenőrzik, hogy biztos nem létezik még egy ilyen adat, valamint, a Metodus osztály segítségével biztos elég hosszú-e a felhasználónév és helyes-e az e-mail cím. Csak igaz esetekben léptet tovább a program a Reg2Activitybe.

A második regisztrációs felületen a keresztnév, vezetéknév, illetve születési dátum megadása kell. A program figyel arra, hogy 13 éven aluliak ne tudjanak regisztrációt tenni. A nevek átalakulnak úgy, hogy az első betű nagy legyen.

Az utolsó regisztrációs osztályban a jelszó kerül bekérésre, mégpedig kétszer. A metódusok ellenőrzik, hogy egyeznek-e, valamint a hosszúság és a benne lévő karakterek is fontos kontroll szempontjából.

A legutolsó osztály a packageben egy WelcomeActivity, ami akkor kerül elindításra, ha a felhasználó sikeres azonosítást tett. Itt megjelenik egy üdvözlő üzenet a felhasználó keresztnevével, amely kapott egy fade animációt, majd 1 másodperc elteltével átkerülünk a userActivity packagebe.

### 3.3.3 userActivity package

Valaki akkor kerül be ezekbe a felületekbe, ha az azonosítás alapján átlagfelhasználónak minősül. Itt található rengeteg olyan felület, amelynek célja, hogy a felhasználó elérje a célját az applikáció használatával. Tulajdonképpen ez a csoport adja a mobilalkalmazás lényegi részét.

Ebben a packageben az első felület az InnerActivity, azonban ez önmagában csak egy üres felület, erre ugyanis rájönnek a fragmentek, mivel ez egy alsó navigációs területtel (bottom navigation) rendelkező felület. Az InnerActivity szolgál a kilépés megerősítésére, amire a választ a felhasználótól várja, valamint ez kezeli, hogy éppen melyik fragment váljon láthatóvá. Három fragment van: Jaratok-, Jegyek-, BeallitasokFragment. Az elsőben az olyan járatok láthatók, amelyre a felhasználónak még nincs jegye és a jövőben fog indulni a járat (a múltbéli járatok nem kerülnek kilistázásra). Érdekesség, hogy itt a kártyákat az osztályban hozza létre a program és nem az XML fájlban, erre azért volt szükség, mert nem tudjuk pontosan hány járatot fog kilistázni az adatbázis. Átlépve a JegyekFragmentekbe láthatók az eddig foglalt helyek adott járatokon. Ez is szintén kártyás stílusban lett elkészítve az osztályban. Az utolsó fragment, azaz a beállításoknál személyre szabhatók a felhasználó adatok. Egy gomb megnyomásával szerkeszteni lehet az EditTexteket, majd a megfelelő ellenőrzések mellett menteni lehet, ha minden helyes. A „Mégse” gombra kattintva elvetjük a változtatásokat. Egy metódus segítségével meg kellett oldani, hogy a felhasználónév és az e-mail menthető legyen, ha nem létezik, illetve, ha eddig a felhasználó birtokában állt. Itt található még egy gomb, ami a PasswordUpdate osztályba vezet át, azonban ez egy másik packageben található, így erről később lesz szó.

A fragmenteken kívül léteznek még bőven felületek, amelyek adatokat képesek közvetíteni. Az első ilyen a JaratActivity, ami akkor jelenik meg, ha a JaratFragmentben kiválasztunk egy adott járatot. Itt megjelennek részletesebb információk a járatról, valamint a foglalás is elérhetővé válik. Innen tovább lépve a FoglalasActivity található, ahol az üléseket az osztályban volt célszerű létrehozni. Egy ülést ki tudunk jelölni, ha az nincs lefoglalva. Amennyiben nem választottunk ülést az ellenőrzés gondoskodik róla, hogy figyelmeztetést tegyen. Ez az osztály főként a Metodus osztály elemeit használja, azaz lekéri a lefoglalt helyeket és listában visszaadja azokat, illetve fordítja is a helyek számát, ugyanis a program int-ként kezeli, míg az adatbázisban más formátumban, azaz Stringként jelenik meg (8-as hely: 2B – második sor, második ülése). A foglalás véglegesítésével az insertFoglalas metódus rögzíti az adatokat az adatbázisba.

A JegyActivity segítségével megtudhatjuk hova és mikorra szól a járatunk, illetve egy QR kód is található ott, ami funkciókat nem szolgál, jelenleg ez még esztétikai elem. Amikor ez megnyitásra kerül a képernyő felvilágosodik és így a későbbi leolvasást is megkönnyíti ez a folyamat. A foglalás lemondására is képesek vagyunk: az erre hivatkozó gomb megnyomásával az applikáció áthelyez a MegerositesActivitybe, ami azt a célt szolgálja, hogy jelszóval megerősítsük a jegylemondási szándékunkat. Ez megvéd attól, hogy jogosulatlan felhasználó foglalást töröljön, illetve a figyelmetlen felhasználót is segíti. Ezután kerül sor a jegytörlésre, majd a felhasználó visszakerül az InnerActivitybe.

### 3.3.4 adminActivity package

Ha az azonosítás után kiderül, hogy a felhasználó igazából admin jogokkal rendelkezik, akkor megjelenik előtte az AdminActivity, ami hasonlóan épül fel, mint a userActivity package, így ez is bottom navigation tulajdonságokkal bíró activity, amelyben fragmentek találhatók. A AdminJaratokFragmentben a járatok listázódnak ki, hasonlóan a felhasználóknál megjelenő fragmentnél. Itt viszont egy járat kiválasztásakor a MegtekintesActivityt nyitjuk meg, amiben a foglalt helyek és rányomáskor a foglaló adatai jelennek meg, ezzel elkészítve egy utaslistát a légitársaságnak. Innen már csak vissza tudunk menni az előző fragmentre. A másik fragment neve AdminJaratInsertActivity, ahol megjelennek a lehetséges útvonalak, amelyre a légitársaság járatot szervez. Ennek kiválasztásával tovább lépünk a JaratInsertActivitybe, ahol ki tudjuk tölteni, hogy a leendő járat melyik nap, melyik időpontban induljon. Ezt a timeToString metódus átalakítja, majd az insertJarat eljárás rögzíti. Az utolsó itt található fragment az AdminBeallitasokFragment, ahol csak a jelszót lehetséges változtatni, a „Jelszómódosítás” gomb megnyomásával. A másik lehetőség a „Kijelentkezés” gomb, ami előhívja az AdminActivity AlertDialogját, hogy az admin biztos-e a kilépésében.

### 3.3.5 globalActivity package

Ebben a packageben szerepel az az activity, amely több helyről is meg lett hívva. Innen bármelyik package bármelyik osztálya meg tudja hívni ezt a felületet. Itt csak egy activity szerepel, mégpedig a PasswordUpdateActivity, amely a jelszó megváltoztatására szolgál. Több helyen is meghívásra került: a userActivity és az adminActivity package beállítási részéről is elérhető. A régi jelszó ellenőrzésének a folyamata a Metodus osztályban történik. Itt valójában, csak az új jelszó titkosítására és rögzítésére kerül sor. Majd visszalépünk arra a felületre, ahonnan jöttünk, ugyanis az admin felhasználónak is itt módosul a jelszava.

## 3.4 Tesztelési dokumentáció

A program írása közben és után is rengeteg tesztelés történt, illetve az esetlegesen felmerülő hibákat azonnal orvosolni is kellett. A végső tesztelésnél a program megfelelt mindennek a kezdeti specifikációban foglaltak szerint. Azonban egy hiba szerepelt az applikációban, amit a fejezet végére hagytam. Ezen kívül már csak az volt hátra, hogy mindezt egy fejezetbe foglaljam.

Az első fontos funkció, amit tesztelni kell az a regisztrációnak a lépései. Először a felhasználó nevet és az e-mail címet szükséges megadni. A rendszer kiszűri az 5 karakternél rövidebb felhasználó neveket, valamint ellenőrzi a felhasználónév eddigi létezését. E-mail cím esetében a maszkot ellenőrzi: „@” előtt és mögött szöveg, egy pont karakter követi mindezt, ami után egy domain következik. Természetesen ebben az esetben is ellenőrzésre kerül az egyedi adat elérhetősége az adatbázisban. Szélsőséges vagy hibás esetben a program vörössé változtatja a hibás mezőt és kiírja mit vett hibának. Tovább lépve a következő mezőkre a keresztnevet, vezetéknevet, illetve a születési dátumot kell megadni. A hibalehetőségek kizárása érdekében bármilyen nevet beírhat a felhasználó, azonban a lezárást követően az első betű kapitálissá válik. Mivel 13 éven aluliak nem regisztrálhatnak, így értelemszerűen erre hibát fog dobni a program és a következő regisztrációs felületre való lépés semmissé válik. A harmadik, egyben utolsó activityben a jelszó megadására van szükség. Ahhoz, hogy a felhasználó biztos legyen a jelszavában, így ezt kétszer kell bekérni. Ha valami nem egyezik, vagy a jelszó túl rövid, illetve nem tartalmaz minimum egy nagybetűt és egy számot, a rendszer hibát dob, a jelszó eltűnik a mezőből, maga a mező vörössé válik, illetve a regisztráció véglegesítése lehetetlen lesz mindaddig, míg a két jelszó a feltételeknek meg nem felelnek.

A belépésnél is meg van oldva, hogy bármilyen hiba lépjen fel. Az első mezőnél meg lehet adni felhasználónevet, illetve e-mail címet egyaránt. Ha ezek az adatok az adatbázisban nem léteznek, teljesen mindegy, hogy milyen jelszót adott meg a felhasználó, az azonosítás sikertelenné válik. A jelszó helytelen megadása is ugyanazt a hibát dobja fel, mintha az első mező lenne helytelen. Mind a két hiba esetében a két mező vörössé válik, a jelszó mezőben a szöveg kitörlődik, illetve a hibaüzenet csak a sikertelen belépésről ad tájékoztatást. Ennek egy célja van: ne lehessen tudni, hogy milyen felhasználónévvel, illetve e-mail címmel vannak regisztrálva felhasználók egy esetleges kibertámadás esetén. Maga a jelszó is egy sajátos titkosítással van ellátva és így eltárolva az adatbázisban, belépésnél ez visszafejthetetlen, csupán a sózás és a titkosított forma ad utalást arra, hogy a jelszó egyezik a mezőben megadott stringgel.

A belső felhasználói felületeken a helyfoglalásnál kellett megoldani, hogy egy foglalható járat csak abban az esetben jelenjen meg, amennyiben az még nem múltbeli, illetve a felhasználó nem tett rá még foglalást (ugyanis minden felhasználó egy helyre foglalhat). Ennek a lekezelése megtörtént és a program a kívánságnak megfelel. A keresés is kiválóan visszaadja azokat az eredményeket, amikre a felhasználó számíthat. A helyfoglalásnál a felhasználónak kötelező kiválasztani egy ülőhelyet és oda nem ülhet, ahol már más megtette a foglalását.

A tesztelés során sikerült egy problémát is kijavítanom. Ezt a hibát a honnan-hova szűrésnél vettem észre. A lényeg annyiból áll, hogy a lekérdezés ezen szakaszába az adatbázis műveletet összezavarja a félidézőjel, így ezt ki kellett szűrni a gépelés közben. Miközben a program folyamatosan figyelte, hogy a felhasználó mit ír be, innentől kezdve tiltva lesz számára ennek a karakternek a begépelése, ezzel megoldva a problémát.

A jegyekhez térve a felhasználó hiba nélkül megtalálja az összes lefoglalt jegyét, amikre rányomva megtekinthet minden részletet hibátlanul a legmegfelelőbb felbontásban.

A beállításoknál a felhasználó az adatmódosításnál minden mezőt átírhat. Azonban kritériumokra figyelnie kell, amit a program hibaüzenetekkel azonnal jelez egy mentés során. A felhasználónévre és az e-mail címre megvannak a fentebb említett protokollok, azonban arra ügyelni kellett, hogy ugyanazt a felhasználónevet és/vagy e-mail címet megadhassa, amivel eddig rendelkezett. A jelszómódosításba lépve pedig elsősorban a régi jelszónak kell érvényt szerezni, majd az újat kétszer megadni, hogy minden hibalehetőség kizárásra kerüljön.

Egy-két helyen a program a megerősítés felületre lép. Ennek azért van értelme, hogyha a készülék feloldva marad, illetve a felhasználó bejelentkezve maradt, például egy esetleges jegytörlésnél a jelszavával véglegesítse a döntését. Ennek köszönhetően kiszűrhetők az akaratlan módosítások és műveletek.

A program készítésénél ügyelni kellett, hogy minden elem az alkalmazásban reszponzív legyen, azaz bármilyen eszközön minden adat ugyanúgy megtalálható legyen. A tesztelésnél főként kétféle eszközt használtam. Az egyik egy Samsung A50 volt, amely 19.5:9-es felbontással rendelkezik. Miközben fejlesztettem erre a mobiltelefonra, fontos volt odafigyelni, hogy mindez egy 16:9-es készüléken is tökéletesen jelenjen meg. Ezen a készüléken minden tökéletesen működött, főleg erre a teljes képernyős verzióra készítettem az applikációt és ez adott nekem egyfajta iránymutatást, hogy a dizájnt tekintve hogyan is haladjak tovább.

A tesztelés egy Samsung Note 10 táblagépen is megtörtént. Azt tudni kell, hogy ez a készülék egy alacsonyabb API szinttel rendelkezik. Alapból a program minimum elvárt szintje a 17-es, de a lényeges elemek inkább az új készülékekre vannak szánva. A hiba annyiból áll, hogy az elemek kiválóan megjelennek, azonban megjelent az applikáció egyetlen hibája. Maga a Bottom Navigation Bar egy ismeretlen hiba miatt használhatatlan. A sikeres belépés után megjelennek a kártyák, azonban az alsó navigációs panel a kártyák alá kerül. Amikor ez a panel előbukkan a fragmentek közötti váltás lehetetlenné válik, ugyanis az ikonok semmit nem reagálnak a megnyomásukra. Próbáltam kijavítani ezt a hibát, de sem önerőből, sem pedig az internet használatával nem sikerült mindezt megoldani, így elkönyveltem ezt az alkalmazásom egyetlen súlyos hibájaként.

A harmadik tesztelési felület egy emulátor volt, amely Android Nougat-al volt ellátva és 16:9-es képernyővel rendelkezett. Ez nagy segítség volt abban, hogy biztos legyek abban, hogy erre a felületre is kiválóan alkalmazkodik a megvalósított alkalmazásom. Részletesen teszteltem minden lehetőségét a programnak ezzel a régebbi Android verzióval és az előbb említett táblagéphez képest még az alsó navigációs panel is tökéletesen működött.

Ezekkel a módszerekkel sikerült kijavítani a hibalehetőségek számát, bár a program még így se 100%-os, de ez a része az applikációnak már a karbantartási szakaszra vonatkozik.

## 3.5 Továbbfejlesztési lehetőségek

Az elkészített program önmagában elégnek minősülhet, de valójában rászorul még egy-két tovább fejlesztés, hogy a mai világban egy releváns és modern applikációnak minősüljön. Értem itt például egy globális adatbázis felállítását a jelenlegi, lokális SQLite-al szemben.

A globális adatbázisra több koncepció is volt. Az első egy MySql csatlakozás, azonban ehhez API hívások kellenek, amik részben bonyolultabbnak számítanak, mint a másik lehetőség, amit FireBase-nek hívnak. Ez egy Google által üzemeltetett online adatbázis, amiben lehet kezelni az autentikációkat (felhasználók azonosítását), illetve az azonnali idejű adatbázist. A program specifikációban szereplő biztos verziója után, elkezdtem GitHub-on egy külön branch készítésével a FireBase kiépítését, az SQLite leendő leváltására. Ez valameddig sikerült is. Az autentikáció és a bejelentkeztetés készen is áll, azonban a probléma a járatok listázásánál kezdődött. A lekérdezés tökéletesen megy, kereséssel együttműködve, azonban, ha több kereséshez fűződő feltételt nem lehet már megadni, ami a nehézséget okozta ebben a feladatban. A határidő közeledte miatt így ezt a feladatot félretettem a későbbi alkalmakra.

Egy másik lehetőség, amit a későbbiekben be lehetne illeszteni az az új ülésrend hozzáadása repülőgéptől függően, ugyanis jelenleg csak egy 120 ülőhellyel rendelkező jármű van forgalomban. Azért készült csak ez a típus, mert más esetben újra kell tervezni az osztályt, hogyha felületről tápláljuk be, hogy az adott járat mennyi eladható hellyel rendelkezik.

Ott volt továbbá az első- és a turistaosztály lehetősége, azonban, hogy ez szembetűnő lehessen egy bizonyos pénzért kell árusítani a helyeket. A rendszernek ehhez ki kell építeni egy olyan lehetőséget, hogy pénzt szerezhessen és azt később a saját keretéből megvásárolhassa.

Volt egy vízió, a több helyre való foglalás. Ez egy azért nehéz dolog, mert ha több helyet szeretne egy felhasználó foglalni, abban az esetben meg kell adnia a többi felhasználó adatait. A többi felhasználó adatainak a valóságát viszont vissza kell igazolni és ez is egy plusz fejlesztés lenne ebben a projektben.

Az utolsó lehetséges továbbfejlesztési lehetőség az az volna, ha az admin felületen, nem csak előre meghatározott útvonalon lehetne járatokat meghatározni, hanem új útvonalakat, ezzel együtt új desztinációkat létrehozva. Ehhez egy nagyon komoly adatbázis kell minden repülőtérről, illetve az ellenőrzést is meg kell határozni a programban, hogy ne lehessen két ugyanazon repülőteret megadni.

A fent említett fejlesztések nem telnének sok időbe, azonban nem volna elég ahhoz, hogy a programot határidőn belül tudjam teljesíteni. Az egyértelmű azonban, hogy a szoftver tele van végtelen továbbfejlesztési lehetőségekkel és valószínű, hogy érdemes is lenne a jövőben vele foglalkoznom, ugyanis ez is egy kulcsa a fejlődésemnek ebben a programozási ágban.

# 4. Felhasználói dokumentáció