**Szegedi Tudományegyetem Informatikai Intézet**

SZAKDOLGOZAT

**Ocztos Károly Levente**

**2024**

**Szegedi Tudományegyetem Informatikai Intézet**

**Magyarországi Madarak mobilalkalmazás**

Szakdolgozat

|  |  |
| --- | --- |
| Készítette: | Témavezető: |
| **Ocztos Károly Levente** | **Dr. Bilicki Vilmos** |
| programtervező informatikus szakos  hallgató | egyetemi adjunktus |

Szeged 2024

**Feladatkiírás**

A szakdolgozat keretein belül a cél egy olyan mobilalkalmazás elkészítése, ami Magyarországon előforduló madarakról nyújt információkat, és a felhasználók számára madárhatározó funkciót kínál. Emellett alkalmas megfigyelések létrehozására és szerkesztésére.

Az alkalmazásban a felhasználók madarakat kereshetnek, amelyekről részletes információkat tekinthetnek meg, valamint madárhatározóval lehetőség van előre meghatározott jellemzők alapján keresni madarakat.

Az alkalmazás a felhasználók számára offline módot is biztosít, hogy a felhasználók akkor is információkhoz juthassanak a madarakról, ha nem csatlakoznak az internethez vagy éppen nincs térerő.

A fajok megfigyelései közvetlenül rögzíthetők az alkalmazásban, ezek szerkesztésére van lehetőség. Felhasználói fiókot is létre lehet hozni, ami segítségével a felhőben tárolhatja el a felhasználó a megfigyeléseit.

Az alkalmazás grafikus felhasználói felülete könnyű kezelhetőséget biztosít, és a reszponzív oldalak széleskörű eszközökön való elérhetőséget tesznek lehetővé.

**Tartalmi összefoglaló**

**A téma megnevezése**

Egy android mobilalkalmazás, ami a magyarországi madarakról nyújt információkat és madárhatározó funkciókkal rendelkezik.

**A megadott feladat megfogalmazása**

Az alkalmazás madarakat listáz, mely listát kereséssel, vagy madárhatározóval lehet szűkíteni. Megfigyelések létrehozására is lehetőséget nyújt, melyeket módosítani, törölni lehet. Az alkalmazás célja megismertetni a madarakat és segítséget nyújtani a beazonosításukhoz.

**A megoldási mód**

A Magyarországi Madarak mobilalkalmazás specifikációja után megterveztem az alkalmazást. Ez a fázis magába foglalja a piackutatást és a használt technológiák alaposabb megismerését, diagramok elkészítését. A fejlesztést xml és natív java nyelveken valósítottam meg, az adatok tárolásához pedig Google Firebase és RoomDatabase szolgáltatásokat használtam. A folyamatot az alkalmazás ellenőrzésével és tesztelésével zártam le.

**Alkalmazott eszközök, módszerek**

A diagramokat a draw.io weboldalon készítettem. Az Android Studio fejlesztői környezetben fejlesztettem az android mobilalkalmazást. A megjelenítésért felelős részeket XML, a mögöttes logikát natív Java nyelven programoztam le. A GitHub platformot használtam egyaránt verziókövetéshez és a fejlesztés feladatainak kezeléséhez. A helyi adatokat RoomDatabase, az online adatokat Google Firebase segítségével tároltam.

A fejlesztési folyamathoz agilis fejlesztési modellt választottam Scrum-szerű módszertannal: miután megterveztem az alkalmazást, úgy elkezdtem a fejlesztését. A fejlesztés során az igényekhez és a szoros határidőkhöz igazítottam a terveket. A módszertan lehetővé tette, hogy a folyamat során felfedezett technikákat és technológiákat könnyedén felhasználhassam a végeredmény kialakításához.

**Elért eredmények**

A kezdeti tervek során kitűzött célokat sikeresen megvalósítottam, a fejlesztés során kialakult új célokkal együtt. Emellett mélyebb tudást szereztem az elkészítés során használt eszközökről.

**Kulcsszavak**

Android Studio, Google Firebase, adatkezelés

**Tartalomjegyzék**

[FELADATKIÍRÁS 3](#_bookmark0)

[TARTALMI ÖSSZEFOGLALÓ 4](#_bookmark1)

[TARTALOMJEGYZÉK 5](#_bookmark2)

[MOTIVÁCIÓ 8](#_bookmark3)

1. [TERÜLETI ÁTTEKINTÉS 9](#_bookmark4)
   1. [Alkalmazások nyugták tárolásához 9](#_bookmark5)
      1. [Zoho Expense 10](#_bookmark6)
      2. [Verify 10](#_bookmark7)
      3. [Expensify 10](#_bookmark8)
      4. [Shoeboxed 10](#_bookmark9)
      5. [Dext Prepare 10](#_bookmark10)
      6. [Invoice Maker 11](#_bookmark11)
   2. [Alkalmazások nyugták felosztásához 11](#_bookmark12)
      1. [EveryoneAllin 11](#_bookmark13)
      2. [Best Bill Splitter 12](#_bookmark14)
      3. [Kittysplit 12](#_bookmark15)
2. [FUNKCIONÁLIS SPECIFIKÁCIÓ 13](#_bookmark16)
   1. [Autentikáció 14](#_bookmark17)
      1. [Regisztráció 14](#_bookmark18)
      2. [Bejelentkezés 14](#_bookmark19)
   2. [Profil 15](#_bookmark20)
   3. [Résztvevők 16](#_bookmark21)
      1. [Résztvevők listázása 16](#_bookmark22)
      2. [Résztvevők létrehozása 16](#_bookmark23)
      3. [Résztvevők részletes oldala 17](#_bookmark24)
   4. [Csoportok 17](#_bookmark25)
      1. [Csoportok listázása 18](#_bookmark26)
      2. [Csoportok létrehozása 18](#_bookmark27)
      3. [Csoportok részletes oldala 18](#_bookmark28)
   5. [Nyugták 19](#_bookmark29)
      1. [Nyugták listázása 19](#_bookmark30)
   6. [Nyugták létrehozása 19](#_bookmark31)
      1. [Képfeltöltés 19](#_bookmark32)
      2. [Új nyugta 20](#_bookmark33)
      3. [Csoportok, résztvevők megadása 20](#_bookmark34)
      4. [Termékek hozzáadása 21](#_bookmark35)
   7. [Nyugták részletes oldala 21](#_bookmark36)
      1. [Fejenkénti végösszeg 21](#_bookmark37)
      2. [Kiexportálás 22](#_bookmark38)
      3. [Nyugták szerkesztése, törlése 22](#_bookmark39)
   8. [Nyugták vendég felhasználóknak 23](#_bookmark40)
3. [FELHASZNÁLT TECHNOLÓGIÁK 24](#_bookmark41)
   1. [Angular 24](#_bookmark42)
      1. [Angular Material 24](#_bookmark43)
      2. [AngularFire 24](#_bookmark44)
      3. [Ngx Image Cropper 24](#_bookmark45)
   2. [Google Firebase 25](#_bookmark46)
      1. [Authentication 25](#_bookmark47)
      2. [Cloud Firestore 25](#_bookmark48)
      3. [Cloud Storage 25](#_bookmark49)
      4. [Hosting 25](#_bookmark50)
   3. [Tesseract.js 25](#_bookmark51)
   4. [Sass 26](#_bookmark52)
   5. [Egyebek 26](#_bookmark53)
4. [ARCHITEKTÚRA 27](#_bookmark54)
   1. [Frontend 27](#_bookmark55)
   2. [Backend 27](#_bookmark56)
5. [BELSŐ FELÉPÍTÉS 28](#_bookmark57)
   1. [Modulok 28](#_bookmark58)
   2. [Interfészek 29](#_bookmark59)
   3. [Szolgáltatások 30](#_bookmark60)
   4. [Routing 31](#_bookmark61)
6. [BIZTONSÁG 32](#_bookmark62)
7. [ADATMODELL 34](#_bookmark63)
   1. [Pénznem 34](#_bookmark64)
   2. [Felhasználó 35](#_bookmark65)
   3. [Profilkép 35](#_bookmark66)
   4. [Résztvevő 35](#_bookmark67)
   5. [Csoport 35](#_bookmark68)
   6. [Termék 36](#_bookmark69)
   7. [Nyugta 36](#_bookmark70)
8. [A RENDSZER MAGASSZINTŰ FOLYAMATAI, MŰKÖDÉSE 37](#_bookmark71)
9. [FONTOSABB KÓDRÉSZLETEK ISMERTETÉSE 40](#_bookmark72)
   1. [Regisztráció 40](#_bookmark73)
   2. [Csoportok létrehozása 41](#_bookmark74)
   3. [Nyugta beolvasása képről 42](#_bookmark75)
   4. [Nyugta szerkesztése 44](#_bookmark76)
   5. [Nyugta részletes oldala 46](#_bookmark77)
10. [TAPASZTALATOK, TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK 49](#_bookmark78)
    1. [Alkalmazás megtervezése 49](#_bookmark79)
    2. [Ütemterv, feladatok 49](#_bookmark80)
    3. [Szakmai ismeretek 49](#_bookmark81)
    4. [Továbbfejlesztési lehetőségek 50](#_bookmark82)
    5. [Összegzés 50](#_bookmark83)

[IRODALOMJEGYZÉK 51](#_bookmark84)

[NYILATKOZAT 52](#_bookmark106)

[KÖSZÖNETNYILVÁNÍTÁS 53](#_bookmark107)

[ELEKTRONIKUS MELLÉKLET 54](#_bookmark108)

**MOTIVÁCIÓ**

Manapság egyre több ember dönt úgy, hogy túrázik a természetben, ami remek lehetőség a kikapcsolódásra és a friss levegőn való feltöltődésre. Egy-egy kirándulás vagy séta során gyakran találkozhatunk érdekes madarakkal. Van, hogy eltűnődünk azon, hogy milyen madár lehet, milyen különleges tulajdonságai vannak, vagy hogy honnan érkezett. Az ilyen pillanatok azonban gyakran elszállnak, mert nem mindig áll rendelkezésünkre a megfelelő eszköz vagy tudás, hogy választ kapjunk a kérdéseinkre.

A Magyarországi Madarak mobilalkalmazás pontosan ezt a célt szolgálja. Ez az alkalmazás lehetővé teszi, hogy bárhol és bármikor információt szerezzünk a minket körülvevő madarakról. Legyen szó egy hétvégi túráról, egy délutáni sétáról a parkban, vagy akár arról, hogy az otthonunk közelében látunk meg egy különleges madarat – az applikáció segítségével azonosíthatjuk, és bővebb információkat tudhatunk meg róla.

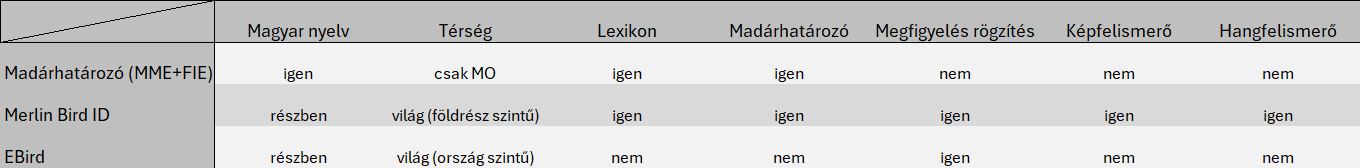
Azonban az alkalmazás nemcsak az azonosítást könnyíti meg, hanem lehetőséget ad arra is, hogy saját megfigyeléseinket rögzítsük. Így személyes madárfigyelési naplót vezethetünk, amelyben visszanézhetjük, hogy milyen madarakkal találkoztunk az utazásaink során. Ez nemcsak hasznos és szórakoztató, hanem segíthet jobban megérteni a természet világát és közelebb hoz minket a környezetünkhöz.

**1. Területi áttekintés**

A piackutatás során a célom olyan alkalmazások megtalálása volt, amelyek madárhatározó szerepet töltenek be. Az elsődleges szempont az volt, hogy magyarországi madarakról tároljon információkat és mobilalkalmazás legyen, de a magyar nyelvűség is kiemelkedő szerepet játszott, mivel elsősorban magyaroknak szánt alkalmazás elkészítése volt a cél.

Az alkalmazások keresése közben azt vettem észre, hogy nem áll túl sok ilyen alkalmazás rendelkezésünkre, magyarok számára többnyire automatikusan fordított, félmagyar mondatokkal tarkítottak elérhetőek. Az alkalmazások hiánya miatt a szempontok kiválasztása főként az általam elvárt funkciók voltak nagyrészt, és ezek egészültek ki a kutatás során talált egyéb funkciókkal.

A kutatás során felmért alkalmazások teljes listáját és funkcióit az 1.1. ábra mutatja be. Ezeknek az alkalmazásoknak köszönhetően sikerült felmérnem, hogy a Magyarországi Madarak mobilalkalmazásnak milyen felhasználói igényeket kell kielégítenie. Ötleteket adtak és példát mutattak a tervezett funkciók megvalósításához. Arra is rávilágítottak, hogy milyen hiányosságokat tudok orvosolni a fejlesztés során, és teljesebb képet kaptam az elérendő céljaimhoz.



* 1. ábra – Piackutatás

**1.1. Madárhatározó**

A Madárhatározó Magyarország első számú madárhatározó alkalmazása, a Magyar Madártani és Természetvédelmi Egyesület (MME) és a Farkaskölykök Ifjúsági Egyesület közös munkája. Ezt a mobilalkalmazás vettem alapul a Magyarországi Madarak mobilalkalmazás megtervezése közben, hiszen ez az egyetlen, ami kifejezetten magyar nyelven készült.

Az alkalmazásban található egy „Ez mi lehet?” modul, ami madárhatározó jelleggel működik. Itt kiválasztható a határozni kívánt madár alakja, színei és élőhelye, és ezek alapján jelenít meg találatokat. Rendelkezik egy „Lexikon” modullal is, ahol az alkalmazás és egyben Magyarországon előforduló összes madár megtalálható. Egy madarat kiválasztva megnyílik egy részletező oldal, ahol sok érdekes információt tudhatunk meg a madárról. Maga az alkalmazás rendelkezik egy „Játék” modullal is, ami a madárfelismerő tudásunkat teszteli.

**1.2. Merlin Bird ID**

A Merlin Bird ID mobilalkalmazás a Cornell Lab of Ornithology világhírű kutatóintézet, amely a madarak és más vadon élő állatok tanulmányozásával foglalkozik. Az ő mobilalkalmazásuk igen fejlett, azonban alapvetően angol nyelvű, magyar csupán gépi fordítással érhető el és csak részlegesen.

Rendelkezik „Step by Step”, lépésről lépésre, madárhatározóval, ami pár előzetes szempont alapján jelenít meg találatokat. Ilyen szempontok a madár tollainak színei, mérete, milyen időszakban volt megfigyelve a madár és hol, vagy éppen mit csinált. A rengeteg nemzetközi adatuknak és méréseiknek köszönhetően rendelkezik az alkalmazás hangfelismerővel is, ami a madár hangjáról azonosítja be a fajt. Található még benne egy képfelismerő is, de a sok zavaró tényező miatt sok esetben nem tudja megfelelően felismerni a megfigyelt madarat. Lehetőségünk van madárhatározás után elmenteni a találkozásunkat, így megörökíthetjük az eseményt.

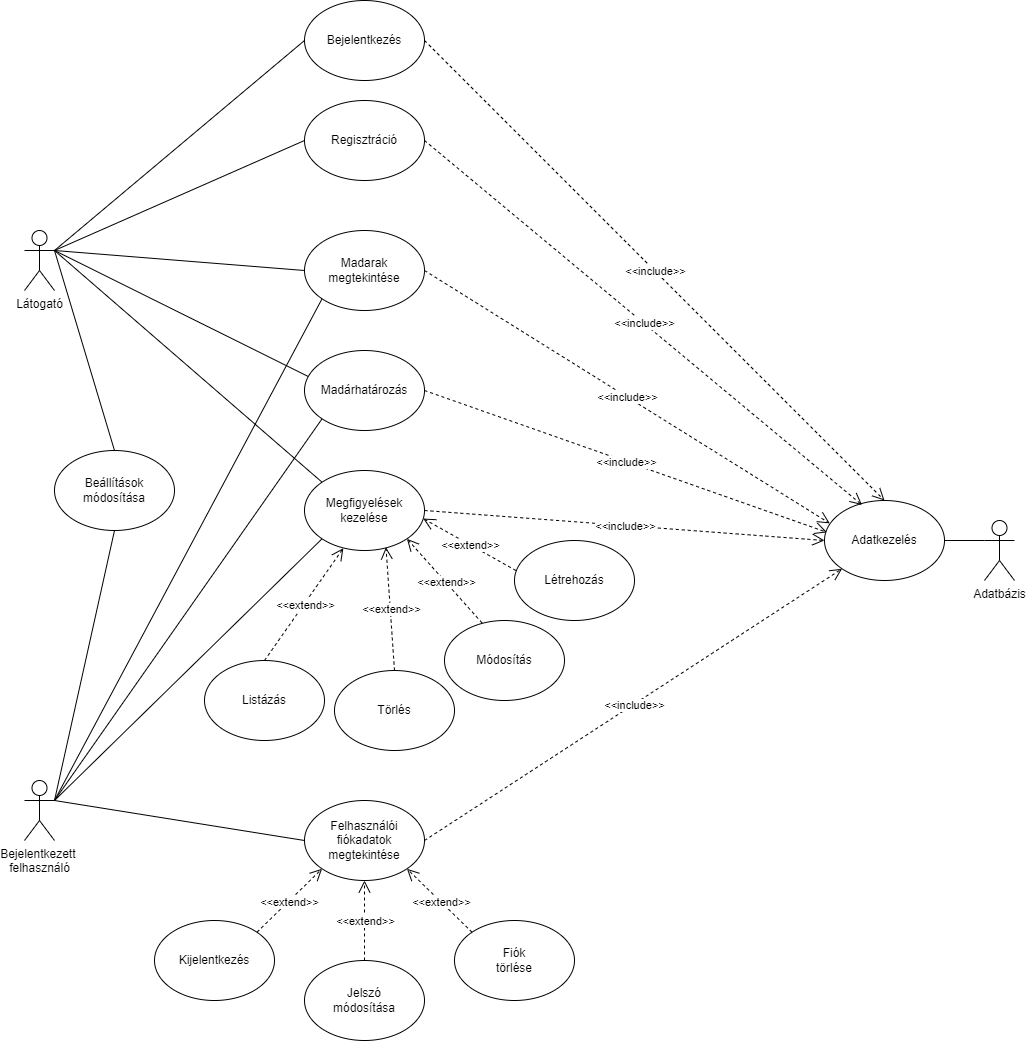
**1.3. eBird**

Az eBird mobilalkalmazása egy globális megfigyelésrögzítő alkalmazás. Ami szintén a Cornell Lab of Ornithology által készült, és így a Merlin Bird ID-val szorosan együttműködhet.

Az alkalmazás leginkább megfigyelések rögzítésére szolgál, mintsem tájékozódásra a madarakról. Rendelkezik egy hatalmas globális adatbázissal, ami menti a megfigyeléseket. Idő és hely megadásával, majd a megfigyelt madár kiválasztásával elmenthető a madár megfigyelése és ezzel hozzájárulhatunk a madarak szokásainak megfigyeléséhez és elemzéséhez. Amennyiben szeretnénk, a Merlin Bird ID-val készített megfigyelésünket meg tudjuk osztani az eBird-del is, ahol így bekerül a globális adatbázisba is. Az eBird alkalmazásban lehetőségünk van arra is, hogy megtekintsük a közelben jelentett megfigyeléseket, így ha szerencsések vagyunk, mi magunk is megláthatjuk az adott faj egyedeit.

**2. Funkcionális specifikáció**

A 2.1. ábrán látható az alkalmazás Use Case (Használati eset) diagramja. A felhasználók két csoportba tartoznak: látogató és bejelentkezett felhasználó. Ezek a csoportok többnyire ugyanazokat a funkciókat érik el, különbözőségük mögöttes üzleti logika szintű.



2.1. ábra – Use Case diagram

Ahogy az a diagramon is látható, a Látogatók számára elérhető minden funkció a Felhasználói adatok megtekintésén kívül, viszont a megfigyelések előzetes regisztráció és bejelentkezés nélkül csupán helyileg tárolódnak el.

A bejelentkezett felhasználók hozzáférnek a saját fiókjukhoz. Emellett az ő megfigyeléseik már a felhőben is eltárolódnak, nem csak az eszközön, így más eszközökre is magunkkal vihetjük adatainkat.

**2.1. Autentikáció**

Az alkalmazás használatához nem kötelező bejelentkezni, és enélkül is elérhető az összes funkció. Azonban, ha szeretnénk, hogy megfigyeléseink ne vesszenek el, bejelentkezés szükséges. Elsődlegesen a beállítások menüpontból érhető el a bejelentkezés és regisztráció felülete. Az autentikálást a Google Firebase beépített szolgáltatásával valósítottam meg.

**2.1.1. Regisztráció**

Fiók létrehozása két módon is lehetséges. Az egyik egy alap e-mail cím és jelszó, valamint megerősítő jelszó megadásával történik. A jelszónak legalább 12 karakter hosszúnak kell lennie. E-mail cím esetében ellenőrzésre kerül a megfelelő formátum, és a regisztrálás megpróbálása után a felhasználó arról is tájékoztatást kap, ha a megadott cím már foglalt. Sikeres regisztrációt követően a felhasználó jelzést kap arról, hogy erősítse meg e-mail címét a kiküldött e-mail-ben.

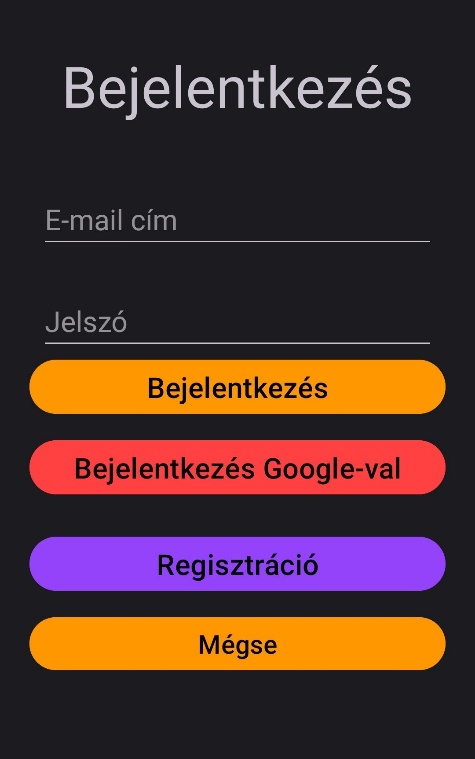
A másik módszer során lehetőség van regisztrálni Google fiókkal. A hitelességről a Google gondoskodik. Amennyiben tudjuk igazolni magunkat a Google-nél, úgy a regisztráció jelszó nélkül és e-mail megerősítése nélkül végbe megy.

**2.1.2. Bejelentkezés**

A bejelentkezéshez előzetes regisztráció és az e-mail cím visszaigazolása szükséges, ha valamelyik hiányzik, vagy a megadott jelszó helytelen, arról egy hibaüzenet jelenik meg. Google fiókkal való bejelentkezés során ugyanaz az eljárás történik, mint regisztrációnál.

Az alkalmazás mindaddig bejelentkezve marad, amíg ki nem jelentkezünk, így ezt a folyamatot nem kell minden alkalommal megismételni.

Bejelentkezés után az alkalmazás a főoldalra navigál, megváltozik a fejléc, és újabb funkciók válnak elérhetővé.



2.1.2.1. ábra – bejelentkezési felület

**2.1.3. Kijelentkezés**

Kijelentkezni a felhasználói fiókunkból minden esetben a Beállításokban tehetjük meg. Itt mutatja a fiókot, amelyikbe be vagyunk jelentkezve. A gombra kattintva automatikusan, azonnal kijelentkezünk és megszűnik az autentikációnk. A fiókhoz ezután nem férünk hozzé, csak akkor, amikor újra bejelentkezünk. Az oldal automatikusan frissül és tájékoztatja a felhasználót a művelet sikerességéről. A megfigyelések az eszközön maradnak, mivel az az eszközhöz tartoznak, de erről részletesebben a 2.5. Megfigyelések pont alatt lehet olvasni. Ha nem vagyunk bejelentkezve, a kijelentkezés opció nem látszik.

**2.2. Beállítások**

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

2.2.1. ábra – beállítások felület bejelentkezett felhasználónak

**2.2.1. Értesítések**

Beállításokban lehet kezelni az értesítéseket. Ha nem szeretnénk kapni üzeneteket az alkalmazástól, akkor nem kell letiltani, hanem itt ki lehet kapcsolni. A változtatás mentése automatikusan végbemegy, amiről tájékoztatást is ad az alkalmazás.

**2.2.2. Fiókbeállítások**

A fiókbeállításokban találhatóak a bejelentkezéshez és regisztrációhoz vezető gombok is, amik átvisznek a megfelelő oldalakra.

Bejelentkezett felhasználóknak lehetőségük van a saját profiljuk megtekintésére is. Ezen az oldalon megjelenik az e-mail címük, és itt lehet kijelentkezni vagy akár törölni a felhasználói fiókot. Egyszerű e-mail és jelszavas fiók esetében a fiók törléséhez meg kell adni a jelszót. Jelszó módosításához meg kell adni az új jelszót, valamint meg is kell erősíteni azt, emellett a régi jelszót is igényli a folyamat. A jelszó ellenőrzése itt sem marad el, legalább 12 karakteresnek kell lennie, valamint a jelszónak és megerősítő jelszónak egyezniük kell.

A felhasználó fiókjának törléséhez itt is szükséges megadni a felhasználó jelszavát. A törlés automatikusan végbemegy, sikerességéről tájékoztatja a felhasználót. Az online tárolt adatok, amik a felhasználóhoz köthetők, törlődnek kivétel nélkül, így a megfigyelések is. Azonban a megfigyeléseink elérhetőek maradnak az eszközünkön, mivel csupán a felhőszinkronizálás szűnt meg. Erről bővebben a 2.5. Megfigyelések pont alatt lehet olvasni.

**2.3. Madarak megtekintése**

A Tudásbázis oldalon kerül listázásra az alkalmazásban található összes madárfaj alfabetikus sorrendben. A listában keresésre is van lehetőség. Ez a madarak nevére szűr, és nem érzékeny a kis- vagy nagybetűkre.

Ha a felhasználó egy listaelemre kattint, azzal megnyílik a madár részletes oldala, ahol a madár összes adata megtekinthető, mint a mérete, természetvédelmi értéke összefoglaló leírása, és még sok más.

TODO: Kép beszúrása

2.3.1. ábra – tudástár oldal

**2.4. Madárhatározó**

Ez a modul azt a célt szolgálja, hogy egy látott madár jellemzőinek megadásával ki tudja találni a felhasználó, milyen faj lehetett a megjelenített találatok alapján. A megadható szempontok között vannak a madár színei, alakja, esetleg élőhelye. Az oldalak között gombok segítségével lehet lapozni előre és vissza. Az utolsó oldalra érve elénk tárul az eredmények oldal, ahol a találatok vannak listázva.

**2.4.1. Jellemzők megadása**

A jellemzők megadása három fő lépésből, ezáltal három oldalból tevődik össze. Ezek a madár tollainak színei, alakjai és élőhelyei. A lapok között következő és előző gombok segítségével tudunk váltani.

Egy-egy oldalon az összes elérhető jellemző fel van sorolva az adott kategóriában. Ezeket ki tudjuk választani egyesével, de akár több megjelölése is lehetséges egyszerre. A válaszainkat a rendszer automatikusan megjegyzi, ha később visszatérünk, akkor a korábban bejelölt szempontjaink továbbra is ki lesznek választva.

Első lapon a madarak színei vannak felsorolva, az alkalmazásban található összes madár minden színe össze lett gyűjtve egy halmazba. Ezáltal bármelyik szín kiválasztásával érintünk valamilyen madarat. Ugyanígy készült a második oldal, ahol a madarakat jellemző madáralakok vannak felsorolva. Mivel egy fajt van, hogy kettő, három alak is jellemez, így több alak megadását is engedélyezzük. Az összes madár összes alakja ki lett gyűjtve egy halmazba. A módszer a harmadik oldalon sem változott, a madaraknak az élőhelyei kiválasztásánál. Az összes madár minden élőhelye össze lett gyűjtve egy halmazba, majd meg lett jelenítve a felhasználó számára.

**2.4.2. Jellemzők megadása**

Az utolsó oldalon találhatók az eredmények. A megjelenítésnél figyelembe vannak véve a megadott jellemzők. A találatok között csak olyan fajok szerepelnek, amik a megadott jellemzők mindegyikével rendelkeznek. A megadottakon felül is lehet még jellemzője a madárnak, de a megadottakkal mindenképpen rendelkeznie kell. A végeredmény egy tudástárhoz hasonlító oldal a listázást és felépítést tekintve. A madarak részletező oldala itt is megnyitható.

**2.5. Megfigyelések**

Ez a modul azt a célt szolgálja, hogy a felhasználók rögzíteni tudják megfigyeléseiket, követni tudják a látott madarakat és rögzítsék élményeiket. Meglévő megfigyeléseikhez könnyen hozzáférhetnek, módosítani, vagy akár törölni is tudják. Felhasználói fiók regisztrálásával a felhőbe is elmenthetik feljegyzéseiket. Az oldal felső részén státuszüzenet van, és ez hogyan befolyásolja a mentéseket.

TODO: Kép beszúrása

2.5.1. ábra – megfigyelések listázó oldala

**2.5.1. Megfigyelés listázó oldal**

Az oldalra navigálva listázódnak az eszközön található megfigyelések. A listanézetben látható a létrehozás dátuma tájékoztató jelleggel, valamint a megfigyelésnek adott név.

Az elemek sorrendjét a legutolsó módosítás dátuma határozza meg., tehát a legutoljára módosított elem fog szerepelni legfelül.

**2.5.2. Megfigyelés létrehozása**

Az oldalon található egy Plusz ikon, amire kattintva megnyílik egy megfigyelés létrehozó ablak.

Itt ki kell tölteni a cím mezőt, ahova a megfigyelésünk nevét kell beírni, amivel könnyen be tudjuk azonosítani, miről szól a megfigyelésünk. A megfigyelés időpontját a naptáras választó rendszer és az óra időválasztó segítségével tehetjük meg. Legalul található a megfigyelés leírására szolgáló mező. Bár a kitöltése opcionális, érdemes leírni az élményünket, vagy a megfigyelés körülményeit és a madarat.

Végül van egy törlés gomb, amivel el tudjuk vetni a létrehozást, valamint egy mentés gomb, amivel létre tudjuk hozni a megfigyelésünket. A létrehozás dátumát és legutolsó módosítás dátumát a rendszer automatikusan a mentés pillanatára állítja. Mentést követően bezárja az ablakot és visszavisz a megfigyelések listázó oldalára. A listát automatikusan frissíti.

**2.5.3. Megfigyelés szerkesztése**

Meglévő megfigyelésekre kattintva megnyílik a szerkesztő oldal, hasonló felépítésben, mint a létrehozó oldal, annyi különbséggel, hogy a mezők ki vannak töltve.

Itt tudjuk módosítani a megfigyelésnek adott elnevezést, a címet, lehet módosítani még a megfigyelés időpontját is és leírását is.

Alul itt is megtalálható a törlés gomb. Erre kattintva azonnal törli a megfigyelést, aminek a sikerességéről értesíti a felhasználót. A mentés gombra kattintva módosítja a módosított értékekre a megfigyelést. A legutolsó módosítás dátumát a mentés gombra kattintáskori időpontra állítja be, és így a megfigyelések lista tetejére kerül. Mindkét gomb esetében visszavisz a megfigyeléseket listázó oldalra. A listát automatikusan frissíti.

**2.5.4. Megfigyelés mentési rendszer**

Regisztrált és bejelentkezett felhasználóknak a megfigyeléseit automatikusan menti a rendszer a felhőbe.

Bejelentkezést követően minden olyan megfigyelést, ami az eszközön található a bejelentkezett felhasználóhoz társít, és menti a felhőbe azokat. Minden olyan megfigyelést, ami más felhasználóhoz tartozik töröl az eszközről. Ezekhez nincsen hozzáférésünk. Ezt követően minden felhőben található megfigyelést, ami a felhasználóhoz köthető letölt, ha nem található meg az eszközön.

Kijelentkezéskor a megfigyelések ugyanúgy megtalálhatóak lesznek az eszközön, hisz a felhőbe mentés csupán kiegészítő szolgáltatás az adatok mentésére.

Előfordulhat olyan eset, amikor törlünk olyan megfigyeléseket az eszközről, amikre nincsen szükségünk. Azonban azokat, ha már mentettük a felhőbe, akkor onnan nem törlődnek, hisz nincs jogosultságunk hozzá. Ismételt belépéskor letölti, mert ott megtalálhatók.

**2.5.5. Felső státuszjelző üzenet**

Ez a felső üzenet három állapotot különböztet meg, amik nagy hatással vannak a mentési formára. Üzenet jellegétől függően van, hogy kattintható az elem, amely tulajdonságot a szöveg magában is tartalmaz.

Az egyik szöveg a *„Nem vagy bejelentkezve, a megfigyeléseid nem mentődnek! Kattints ide a bejelentkezéshez!”*. Ez az alap állapot, amikor a felhasználó megfigyelései csak az eszközön elérhetők, a felhőben tárolt adatokhoz nem fér hozzá, az ottani módosításukhoz be kell jelentkezni. Ahogy azt az üzenet szövege is mutatja, a szöveg kattintható, amely hatására megnyílik a bejelentkező ablak. Sikeres bejelentkezés után a megfigyelések oldalára visz vissza, ahol a szöveg megváltozik automatikusan.

Van egy bejelentkezett és autentikált státuszt jelző szöveg is, ami a *„Be vagy jelentkezve (felhasználó e-mail címe).”*. A felhasználó e-mail címe helyén megjelenik az e-mail címe a bejelentkezett felhasználónak. Ennél az üzenetnél a felhasználó adatai mentődnek a felhőbe, szinkronizálva vannak az ottani adatokkal az eszközön lévők. Ez minden látható megfigyelésre vonatkozik. Ekkor a módosításaink azonnal bekerülnek a felhőbe is.

Az utolsó üzenet a *„Jelenleg offline vagy. Csatlakozz hálózathoz!”*. Ez azt jelzi, hogy nem érhető el internet, így a módosításaink nem mentődhetnek a felhőbe. Ez olyankor is megjelenik, amikor nem vagyunk bejelentkezve, mivel ilyenkor nem elérhető a bejelentkezés sem és erre akarja felhívni a figyelmünket.

**3. Felhasznált technológiák**

**3.1. Natív XML**

Az XML egy mobilalkalmazás-fejlesztésben elterjedt nyelv, amely a felhasználói felületek leírására szolgál. Elsődleges célja, hogy strukturált, könnyen olvasható formában határozza meg az alkalmazás vizuális elemeit, valamint azok elrendezését. Az Android fejlesztés során az XML a felhasználói felület kialakításának alapvető eszköze, amely minden vizuális komponenst, például gombokat, szövegeket vagy listákat tartalmaz. Stílusok testreszabására, valamint egységes stílusminták kialakítására is alkalmas.

Az XML fájlok kizárólag a felhasználói felület statikus elemeinek definiálására szolgálnak, így a működés leírására egy másik nyelvet is igénybe kell venni. Ez a tagoltság segít jobban elkülöníteni a részeket, így karbantarthatóbb lesz a kód és szervezettebb, valamint a hibák detektálásában is sokat segít.

Az XML-t használva a vizuális elemek újra felhasználhatóak és testreszabhatóak. Például az alkalmazásban használt stílusokat és szöveges erőforrásokat külön fájlokban lehet tárolni, ami elősegíti a rugalmasságot és a következetességet, elkerülve a kódduplikációt.

Az Android fejlesztés során az XML fájlok moduláris szervezése lehetővé teszi, hogy az egyes képernyők elrendezései külön fájlokban helyezkedjenek el, ezáltal a projektek könnyebben skálázhatóak.

Fejlesztés során natív XML fájlokat használtam, amelyek segítségével részletesen meghatároztam az alkalmazás vizuális elemeit és azok elrendezését, sablonokat készítettem a hasonló elemekhez.

**3.2. Natív Java**

A Java egy platformfüggetlen, objektumorientált programozási nyelv, ami a mobilalkalmazás-fejlesztés egyik alapeleme. Elsődleges célja, hogy robusztus, biztonságos és skálázható alkalmazásokat hozzon létre, melyek különböző eszközökön és platformokon egyaránt működnek. Az Android fejlesztés során a Java a funkcionalitás, a működés leírásának biztosítására szolgál, legyen szó adatkezelésről, felhasználói interakciók kezeléséről vagy hálózati műveletekről.

A Java nyelv struktúrája lehetővé teszi a könnyen érthető és karbantartható kód írását. Az osztályokra és objektumokra épülő felépítés támogatja az újra felhasználhatóságot és a moduláris tervezést. Az Android alkalmazásokban a Java segítségével valósíthatóak meg a különböző tevékenységek és komponensek működése, például gombokra való kattintás kezelése, kitöltött szövegbeviteli mezők tartalmának feldolgozása vagy adatok lekérdezése egy adatbázisból.

A Java kiterjedt könyvtárai és keretrendszerei támogatják a különféle fejlesztési igényeket, például a hálózati kapcsolatok, az adatok feldolgozása vagy a felhasználói értesítések kezelése terén. Az Android fejlesztéshez biztosított API-k segítségével a Java lehetővé teszi az alkalmazások eszközspecifikus funkcióinak használatát. A Java nyelv stabilitása és széles körű támogatottsága és népszerűsége biztosítja, hogy az elkészült alkalmazások hosszú távon is fenntarthatók legyenek.

Fejlesztés során a natív Java nyelvet használtam az Android alkalmazásom működésének biztosítására. Ez magában foglalta a felhasználói interakciók kezelését, az adatbázis-műveletek megvalósítását és az alkalmazás logikájának kialakítását, miközben szorosan együttműködött az XML-ben definiált vizuális elemekkel. A Java nyújtotta rugalmasság és teljesítmény lehetővé tette, hogy az alkalmazás gyorsan és hatékonyan tudjon működni.

**3.3. Firebase**

A Firebase egy felhőalapú platform, amely a modern alkalmazásfejlesztéshez nyújt eszközöket és szolgáltatásokat. Fő célja, hogy megkönnyítse a fejlesztők számára az alkalmazások backend-funkcióinak megvalósítását, mint például az adatok tárolása, a felhasználók hitelesítése vagy értesítések küldése. Alapvető funkciói ingyenesek, azonban egy komplexebb alkalmazáshoz tudjuk bővíteni csomagunkat extra funkciók eléréséhez.

Fejlesztés során a Firebase szolgáltatásait használtam az adatok tárolására és szinkronizálására, valamint a felhasználók hitelesítésére. A Firebase rugalmassága és integrált megoldásai nagyban megkönnyítették az alkalmazásom fejlesztését és skálázhatóságának biztosítását.

**3.3.1. Cloud Firestore**

A Firebase egyik alapvető eleme a Firestore, egy NoSQL alapú adatbázis, amely valós idejű adatkezelést és szinkronizációt tesz lehetővé a felhasználói eszközök között. Támogatja az adatok összetett lekérdezését és a tranzakciókat, így könnyen kezelhetők bonyolult adatkapcsolatok is. Egyszerű szabályokkal lehet korlátozni az adatokhoz való hozzáférést, ezzel növelve a biztonságot különösebb erőfeszítés és időráfordítás nélkül.

**3.3.2. Firebase Authentication**

A Firebase Authentication a Firebase által nyújtott hitelesítési szolgáltatás, amely egyszerű és biztonságos megoldást kínál felhasználók azonosítására különböző platformokon. A célja, hogy minimalizálja a hitelesítéshez szükséges fejlesztési időt, miközben erős biztonságot nyújt.

A szolgáltatás szorosan integrálható más Firebase megoldásokkal, például a Firestore-ral, amely lehetővé teszi, hogy a hitelesített felhasználókhoz kapcsolódó adatok gyorsan és biztonságosan tárolhatók legyenek.

**3.3.3. Cloud Storage**

A Firebase Cloud Storage egy megbízható, skálázható felhőtárhely, amely fájlok, például képek vagy videók tárolására és elérésére szolgál. Egyszerűen integrálható alkalmazásokba. Én a képek tárolására használtam.

**3.4. Room Database**

A Room Database egy Android fejlesztésben használt könyvtár, amely leegyszerűsíti a helyi SQLite adatbázisok használatát. Elsődleges célja, hogy könnyen kezelhető, biztonságos és hatékony adatbázis-műveleteket biztosítson az alkalmazások számára, miközben automatikusan kezeli a komplex adatkapcsolatokat.

A Room három fő összetevőre épül: az entitások az adatbázis tábláit reprezentálják, a DAO-k (Data Access Objects) az adatbázis-műveleteket definiálják, a Database osztály pedig összefogja az adatbázis konfigurációját és hozzáférést biztosít. A Room támogatja az adatok típusbiztonságát, a reaktív programozást LiveData használatával, és automatikusan kezeli a migrációkat, ha az adatbázis struktúrája megváltozik.

A Room a háttérmunkák kezeléséhez optimalizált, és lehetővé teszi az adatok gyors, hatékony és valós idejű lekérdezését, módosítását. Fejlesztés során a Room Database-t használtam az alkalmazásom helyi, offline adatkezelésének biztosítására, amely stabil és jól strukturált módon tárolta és kezelte az adatokat.

**3.5. Egyebek**

A diagramokat, mátrixokat a draw.io oldal segítségével készítettem. Itt több szoftverfejlesztéshez kapcsolódó diagram sablonja elérhető, valamint előre elkészített elemeket is biztosít.

A fejlesztést az Android Studio fejlesztői környezetben végeztem, ami az Android alkalmazások fejlesztésére szolgáló hivatalos integrált fejlesztőkörnyezet. A JetBrains IntelliJ IDEA platformjára épül. Intelligens kódszerkesztés, vizuális elrendezésszerkesztő, beépített emulátor, hibakereső és még sok más funkciója könnyíti a fejlesztést. Az Android Studio segítségével a fejlesztők gyorsan készíthetnek, tesztelhetnek és optimalizálhatnak alkalmazásokat különböző eszközökre és képernyőméretekre.

A GitHub-ot egyaránt használtam verziókövetésre és a feladatok kezelésére is. A különböző feladatrészekhez Issue-kat vettem fel. Amikor elkészültem egy feladattal, zároltam, így láttam milyen teendőim vannak még hátra és mikkel vagyok kész.

A Microsoft Excel egy népszerű táblázatkezelő szoftver, amit adatok rendszerezésére, elemzésére és vizualizálására használnak. Az alkalmazás elkészítése során táblázatok készítéséhez lett alkalmazva, például a piackutatás eredményének összegyűjtésére.

**4. Architektúra**

A Magyarországi Madarak mobilalkalmazás architektúrája két részből, a frontendből és a backendből áll. A frontendet XML segítségével valósítottam meg, míg a backendhez Java-t, Room Database-t és Firebase-t használtam.

**4.1. Frontend**

A frontend az a része az alkalmazásnak, amivel a felhasználó interakcióba lép. Ide tartoznak az olyan elemek, mint az alsó navigációs sáv, a madarak listázása, megfigyelések létrehozása, az autentikációs felület és még sok más is.

Az XML határozza meg a struktúrát és az oldal elemeit, elrendezésüket és stílusukat. Hasonló felépítésű és célú elemekhez sablonok készíthetőek, így jelentősen csökkentve a kódduplikációt, valamint egységes kinézetet ad az alkalmazásnak. A mögöttes logika nem része az XML-nek, így a gombok és beviteli mezők kezelése sem tartozik a feladatai közé.

**4.2. Backend**

A backend az a része az alkalmazásnak, ami a működést, a mögöttes logikát határozza meg. Ezen szerepkörbe tartozik az események kezelése, gombok és beviteli mezők működésének leírása, valamint az adatok kezelése és tárolása.

A fő része a natív Java, ami a jelentős részét kezeli az eseményeknek. Feldolgozza a felhasználói interakciókból keletkező adatokat, így a gombok nyomását, beviteli mezők tartalmának kezelését, és a megfelelő adatok küldését a frontendre.

A helyi adattároláshoz Room Database lett igénybe véve, ami nem tekinthető hagyományos backendnek. A könyvtár egyedi annotációkat tartalmaz, ami extra logikával egészíti ki az osztályokat és változókat, valamint a backend ezen részét nem kell implementálni, helyette a Room Database által biztosított szolgáltatások és metódusok alkalmazhatóak. A könyvtár egy biztonságos adatbázist is biztosít, így könnyítve a fejlesztők munkáját, növelve az alkalmazás értékét.

A Firebase sem tekinthető hagyományos backendnek, erősen hasonlít a Room Database-hez. Ez azt jelenti, hogy a hagyományos megoldással ellentétben itt sem szükséges külön backendet implementálni, helyette a Firebase által biztosított szolgáltatások és metódusok használhatók az autentikációk kezelésére, adatok tárolására és lekérésére. A Firebase is egy nagyon megbízható, biztonságos, könnyen kezelhető adatbázist nyújt.

**5. Belső felépítés**

**5.1. Activity-k**

Egy Android alkalmazás több activity-ből épül fel, amelyek az alkalmazás szorosan összetartozó részeit alkotják. Az activity-k egy adott képernyőhöz vagy funkcióhoz kapcsolódó feladatokat kezelnek, például egy bejelentkezési folyamatot vagy egy lista megjelenítését. Az egyes activity-k között navigációval biztosítható az alkalmazás logikai folyamata.

Az activity-k más komponensek (például fragmentek, service-ek) funkcióit is használhatják, és szorosan együttműködnek a vizuális elemekkel, amelyeket XML-ben definiálnak. Ez a felépítés lehetővé teszi az alkalmazás moduláris kialakítását és átláthatóbb fejlesztését. Minden activity osztályhoz tartozik egy megfelelő layout XML fájl is.

A Magyarországi Madarak mobilalkalmazásban az alábbi activity-k valósulnak meg:

* *KnowledgeBaseActivity*:

Ez a launcher activity az alkalmazásban, annyit jelent, hogy indításkor ez az oldal jelenik meg. Ez az úgynevezett főoldal, mely *Tudásbázis* néven szerepel a navigációban. Itt listázódnak a madarak és lehet keresni közöttük.

* *BirdPageActivity*:

Ez az oldal egy madárnak a részletező oldala. Itt kerül megjelenítésre egy faj minden jellemzője és a leírása. Megnyitásakor egy új activity-ként jelenik meg, tehát az activity, ahonnan megnyitottuk ezt az oldalt, még létezik, nem zárult be.

* *BirdIdentificationActivity*:

Ez a *Madárhatározó* néven szereplő oldal. Itt három nagy szempont (tollazat színei, madár alakja, madár élőhelye) alapján kereshetünk madarakat. Fő célja, hogy egy látott madarat tudjunk könnyedén beazonosítani. Négy fragmentből épül fel, amik között gombok segítségével tudunk váltani. A fragmentek együtt osztozkodnak az activity életciklusán, nagyobb lehetőséget adva a moduláris felépítésre. Négy fő fragmentből épül fel a *Madárhatározó*:

* *BirdColorFragment:* A tollazat színeinek megadásához.
* *BirdShapeFragment:* A madár alakjának megadásához.
* *BirdHabitatFragment:* A madár élőhelyeinek megadásához.
* *BirdIdentificationResultsFragment:* Az eredmények megjelenítéséhez.
* *ObservationsActivity*:

Itt található a *Megfigyelések* oldal. Itt kerülnek listázásra a felhasználó megfigyelései, itt lehet létrehozni, valamint innen érhetőek el a megfigyeléseket részletező oldal.

* *ObservationPageActivity*:

Ez egy megfigyelés részletező oldala, mely külön indított activity-ként jelenik meg. Ezt a sablont használja fel az alkalmazás megfigyelések készítésére, de már meglévő megfigyelések adatai is ide töltődnek be. Ezáltal itt lehet létrehozni, módosítani, törölni megfigyeléseket.

* *SettingsActivity*:

Ez az activity tartalmazza az alkalmazás beállításait, valamint a felhasználói beállításokat is, ha a felhasználó be van jelentkezve. Ha nincsen autentikált felhasználó, úgy innen érhető el a bejelentkezés és a regisztráció egy-egy gomb segítségével.

* *LoginActivity*:

A *Bejelentkezés* oldalon lehet bejelentkezni e-mail cím és jelszó megadásával vagy a Google bejelentkezés is választható. Innen érhető el a *Regisztráció* oldal is. Ez az oldal egy külön activity-ként indul el, amin a regisztráció oldalával osztozkodik.

* *RegisterActivity*:

A *Regisztráció* oldalon lehet regisztrálni e-mail cím és jelszó, valamint jelszó megerősítése beviteli mezők helyes kitöltésével. Regisztráláskor egy megerősítő e-amil kerül elküldésre. Ezen az activity-n elérhető még a Google bejelentkezés is, valamint a *Bejelentkezés* oldalra navigálás. Az az activity egy külön activity-ként jelenik meg, amin a *Bejelentkezés* oldalával osztozkodik.

**5.2. Adapterek**

Az adapterek az Android alkalmazásokban olyan osztályok, amelyek a különböző adatkészleteket és a felhasználói felület megjelenítéséért felelős elemeket kapcsolják össze. Az adapterek segítségével az adatok strukturáltan és dinamikusan jeleníthetők meg a képernyőn, miközben az egyes elemek megjelenését is testreszabhatjuk.

Az adapterek különféle forrásokból származó adatokat, például tömböket, listákat vagy adatbázisok lekérdezéseit, képesek kezelni, majd ezeket vizuálisan reprezentálják a megfelelő nézetekben, például RecyclerView-kban. Ez a felépítés lehetővé teszi a felhasználói felület és az adatok elkülönítését, így egyszerűbbé válik az alkalmazás fejlesztése és karbantartása.

A Magyarországi Madarak mobilalkalmazásban az alábbi adapterek szerepelnek:

* *ListBirdsAdapter*:

Az az adapter felelős azért, hogy lekérje a madarakat az adatbázisból, majd megjelenítse azokat egy egyedi dizájnú listaszerkezetben. A lekérés és megjelenítés úgy van kialakítva, hogy ne akadályozza az UI szálat, így a felhasználó tudja használni az alkalmazást akkor is, ha még nincsen teljesen betöltve az összes elem. Az adatbázisváltozásokat is követi, így ha valami változik, akkor a megjelenítésben is változás történik. Ezt az adaptert a *Tudásbázis* és a *Madárhatározó* is használja.

* *BirdIdentificationAdapter*:

Ez az adapter egy egyedi kinézetű jelölőnégyzeteket felhasználva listáz adatokat. A *BirdIdentificationAdapter* a madár tollainak színeinek, madár alakjainak és madár élőhelyeinek listázásához lett felhasználva a *Madárhatározó* oldalon.

* *ObservationsAdapter*:

Ez az adapter kéri le az adatbázisból a felhasználó megfigyeléseit, majd megjeleníti azokat egy egyedi listanézetben. Ez az adapter a *Megfigyelések* oldalon került felhasználásra.

**5.3. Adatok**

A Magyarországi Madarak mobilalkalmazás követi a **követi az** **MVVM (Model-View-ViewModel)** architektúrát. Ez az androidfejlesztésben egy elterjedt mintázat, ami tiszta rétegződést biztosít, és különválasztja a különböző felelősségeket az alkalmazásban. Az adatok így Model, DAO, Repository, Database és ViewModel osztályokra és interface-kre lett felosztva. Ezek elsősorban a Room Database-hez lettek készítve és tervezve.

A Magyarországi Madarak mobilalkalmazásban az alábbi adatok készültek el:

* *Bird*:

Egy madár minden adata elérhető ebből az entitásból. Legtöbb adata kapcsolótáblák mentén tárolódnak, amit a Room Database automatikusan kezel. Emellett van egy *BirdEntity* típusú adattagja is, ami már ténylegesen is tartalmazza az olyan adatokat, amik nem referenciakánt vannak megadva. A *BirdDAO* interface kezeli a keresztkapcsolattáblák mentén való összekötést, valamint a *Bird* és *BirdEntity* megfelelő létrehozását a lekérésekhez. A lekéréseket és szinkronizálást *BirdRepository* kezeli, ami a madarakhoz tartozó konstansok Firestore szinkronizálását is kezeli. A megjelenítésért a *BirdViewModel* felelős.

* *BirdEntity*:

Lényegében a Firestore-ban szereplő nem referenciaalapú adatokat tárolja, valamint a madár id-ját is, ami a Firestore-ban a dokumentum azonosítójaként van jelen.

* Konstansok:

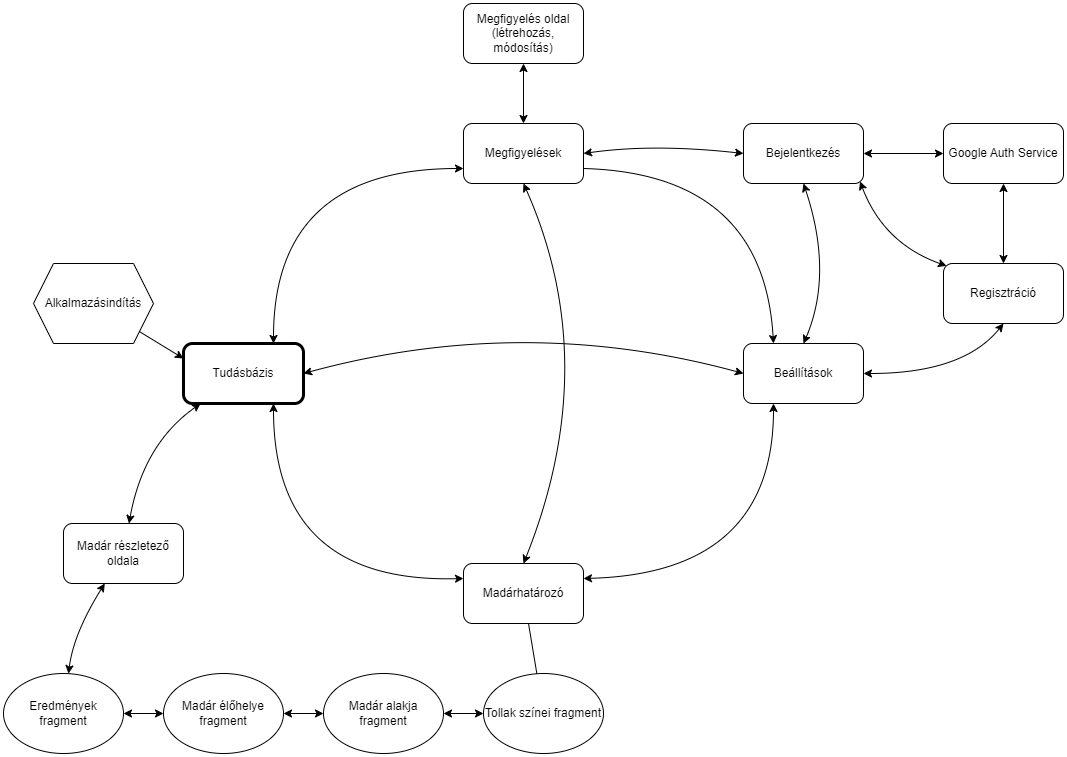
Főként madarak adataihoz lettek létrehozva a redundancia elkerülése végett. Kapcsolótáblák segítségével kerülnek összeköttetésbe a madarakkal. Csupán egy id-ból, ami a Firestore-ban a dokumentum azonosítójaként szerepel, és egy értékből áll. Dao csupán a Firestore-ral való szinkronizáláshoz szükséges. A konstansok közé tartoznak:

* *ConservationValue:* A madarak természetvédelmi értékeinek meghatározásához. Egész számértéket tárol. Firestore szinkronizáláshoz *ConservationValueDAO* metódusai lettek használva.
* *Diet:* Madarak főbb táplálékait tárolja. Tartozik hozzá egy *DietCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *DietDAO* metódusai lettek használva.
* *Color:* Egy madár tollainak színét tárolja. Tartozik hozzá egy *ColorCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *ColorDAO* metódusai lettek használva.
* *Shape:* Egy madár lehetséges alaklját tárolja. Tartozik hozzá egy *ShapeCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *ShapeDAO* metódusai lettek használva.
* *Habitat:* Madarakfőbb élőhelyét tárolja. Tartozik hozzá egy *HabitatCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *HabitatDAO* metódusai lettek használva.
* *Observation*:

Egy megfigyeléshez tartozó összes adat tárolására szolgáló entitás. Firestore-ban tárolt adatoktól az observationId-ban tér el, ami a dokumentum azonosítójaként szerepel. A Room Database az *ObservationDAO* interface segítségével hajtja végre az adatbázisműveleteket. A felhővel való szinkronizálásért és az adatbázisműveletek megfelelő használatáért az *ObservationRepository* felelős. A megjelenítés és az UI-val való kommunikáció az *ObservationViewModel* feladata.

* *User*:

Mivel végül nem kerül tárolásra külön felhasználói adat, így végül nem lett használva.

**5.4. Navigáció**

5.4.1. ábra – Navigációs gráf

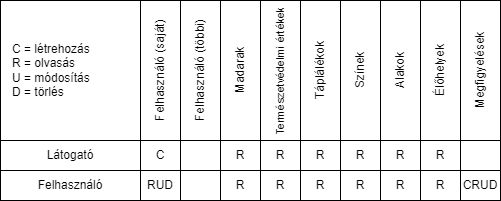
A Magyarországi Madarak mobilalkalmazásban a navigációt egy alsó navigációs sáv kezeli a főbb modulok eléréséhez. A főbb modulok közötti váltás során az activity megszűnik, majd elindul az az activity, ahova navigáltunk.

Az 5.4.1.-es ábrán látható, hogy az alkalmazás induló activity-je a *Tudásbázis*. A madarak részletes leíró oldalát megnyitva új activity indul a korábbi bezárása nélkül. Itt már nem érhető el a navigációs menü.

Érdekesség lehet, hogy a *Madárhatározó* oldal fragmenteket használ a tartalom megjelenítéséhez. Itt az alsó navigációs menü látható minden esetben. A fragmentek között előre és hátra gombok segítségével lehet váltani szekvenciálisan haladva. A legvégső eredmények fragmentben megjelenített madár találatokra kattintva megtekinthető a leíró oldaluk, ami egy új ablakban jelenik meg.

Végül kiemelném az autentikációt. Ha ki vagyunk jelentkezve, úgy a *Megfigyelések* oldal felajánlja a bejelentkezés lehetőségét, így elérhető lesz onnan is. Alapértelmezetten viszont a *Beállítások* oldalról érhető el a *Bejelentkezés* és *Regisztráció*. Mindkettő lehetőséget biztosít Google-os bejelentkezésre, ahol a Google külső bejelentkeztető oldala jelenik meg.

**6. Biztonság**

A Firestore lehetőséget biztosít biztonsági szabályok írására, ami korlátozhatja az adatbázisban tárolt adatokhoz való hozzáférést. A Security Rules szolgáltatás használatával megadtam, hogy egyes kollekciókban milyen szabályok legyenek érvényesek az adatok kezelése során külön olvasás, létrehozás, módosítás és törlés szinten.

6.1. ábra – Hozzáférési mátrix

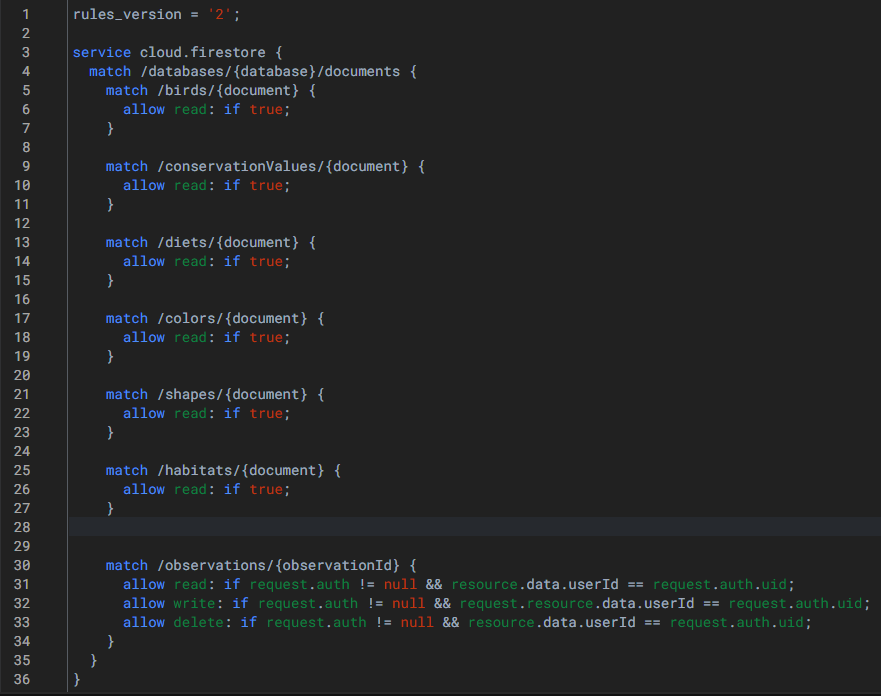
A 6.1. ábrán láthatóak a látogatók és a bejelentkezett felhasználók jogai az egyes objektumokhoz vagy erőforrásokhoz való hozzáférés során. Ez a mátrix a Firestore adatbázisra vonatkozik.

Felhasználói fiók létrehozására csak látogatóknak van lehetőségük, valamint csak ők tudnak bejelentkezni.

Bejelentkezés után lehetőségünk van a fiókunkhoz tartozó adatok megtekintésére. Csak saját fiókot tudunk törölni és módosítani. Fontos kiemelni, hogy az alkalmazásban jelenleg nincs lehetőség kapcsolatba lépni, interaktálni más felhasználókkal, ezért más felhasználói adatokhoz nem tudunk hozzáférni semmilyen formában.

A természetvédelmi értékek, táplálékok, színek, alakok és élőhelyek amolyan konstans szereppel rendelkeznek, redundancia elkerülése végett lettek kiszervezve. Ezek olyan adatok, melyek nem változnak, és nem is bővülnek, így csak lekérni lehet. Ami érdekes lehet még az az, hogy a madarak is konstans jelleggel léteznek. Róluk is csak tájékozódni lehet, módosításra nincs lehetőség.

A megfigyelések jelölése is érdekes lehet. Ugyan a látogatók tudnak létrehozni, majd teljeskörűen rendelkezni felettük, de ez csak a helyi tárolásra vonatkozik. Csak a bejelentkezett felhasználók megfigyelései kerülnek be a felhőbe is, amolyan biztonsági mentésképpen. És nekik van lehetőségük így átvinni másik eszközre is megfigyeléseiket.



6.2. ábra – Hozzáférési szabályok

A 6.2. ábra megmutatja a teljes szabálykészletet, ami a Firebase-ra vonatkozik. Az *allow read: if true;* sorok engedélyezik a lekéréseket, olvasásra lekérhetőek ezek az elemek. Írásra azonban nem soroltam fel a szabályokat (*allow read: if false;*), mivel ez az alapértelmezett beállítás, ha valamire nem vonatkozik megszorítás. Ezek a szabályok biztosítják, hogy a madarakat, természetvédelmi értékeket, étrendeket, színeket, alakokat és élőhelyeket csakis kizárólag megtekinteni lehet, módosítani, létrehozni és törölni nem lehet.

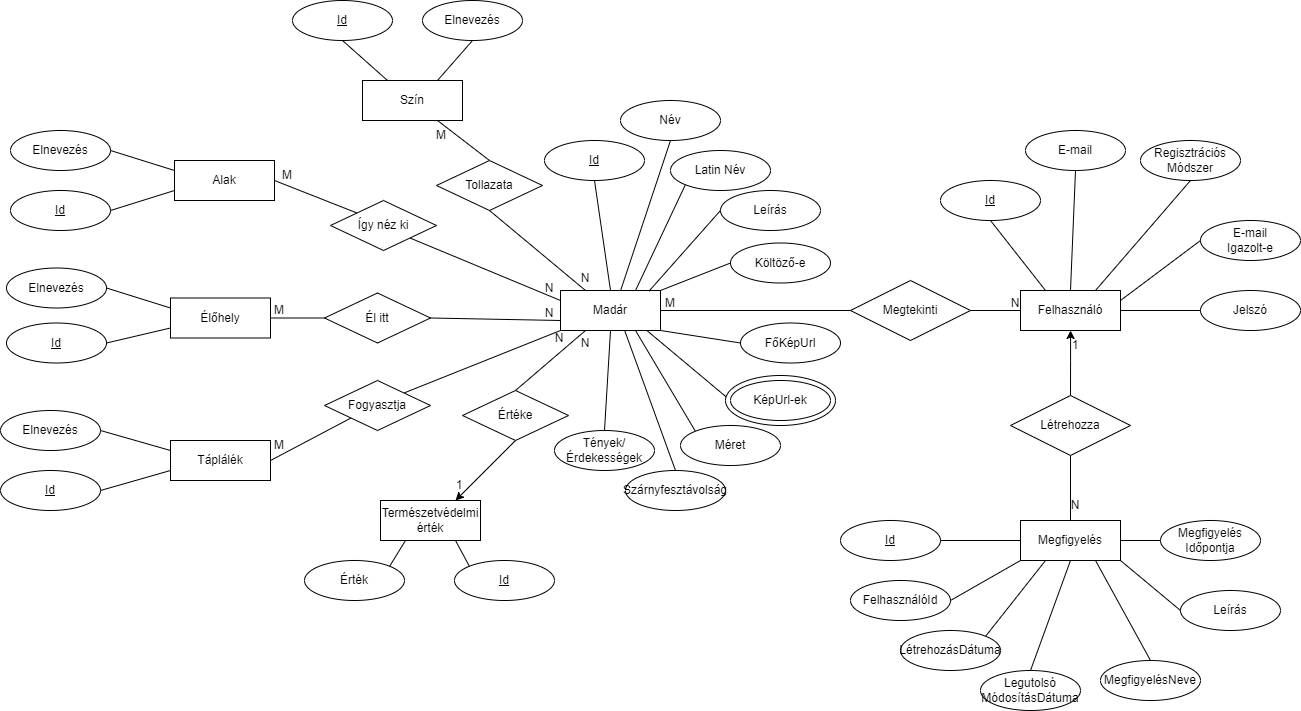
Az *if request.auth != null* részek felelnek azért, hogy a megfigyelésekhez csak akkor kérhessük le, hozhassunk létre vagy módosíthassuk, valamint törölhessük, ha bejelentkezett, autentikált felhasználók vagyunk. A második fele a lekérdezésnek, tehát a *resource.data.userId == request.auth.uid*, biztosítja azt, hogy csak akkor férünk hozzá a műveletekhez, ha a megfigyelés ténylegesen hozzánk, az autentikációban szereplő felhasználóhoz tartozik. Ez biztosítja a felhőadatokhoz való hozzáférések korlátozását a hozzáférési mátrix (6.1. ábra) alapján.

A helyi adattároláshoz Room Database-t használtam. Az Android alapértelmezett fájl-hozzáférési modellje, miszerint az adatokhoz alapvetően csak őket létrehozó alkalmazások férhetnek hozzá közvetlenül, elegendő biztonságot nyújthat.

A típusok ellenőrzését és konverziót az alkalmazás ellenőrzi.

**7. Adatmodell**

A Firestore szolgáltatását és a Room Database-t vettem igénybe az adatok tárolásához. A Firestore egy NoSQL adatbázis, ahol az objektumokat dokumentumok képviselik, amik kollekciókban tárolódnak. A Room Database pedig egy Androidon használt ORM (Object-Relational Mapping), amely az SQLite adatbázist egyszerűsíti, relációs struktúrát és típusbiztonságot biztosítva.

A két adatbázis hasonló felépítésű, de vannak apróbb eltérések. A Firestore lehetőséget nyújt referencia típusú adattagok tárolására. Olyan is előfordulhat, hogy egy listában vannak ezek a referenciák eltárolva, így ezekhez nem kell külön kapcsolótáblát létrehozni. A Room Database viszont csak egyszerű típusokat tárol, így kellett hozzá létrehozni kapcsolótáblákat is. További eltérés, hogy a Firestore képes dátumot is tárolni, amire a Room Database szintén nem képes. Ezekhez külön TypeConverter-eket kellett létrehozni, ami egyszerű szöveggé alakítja megadott szabály szerint a komplexebb típusokat. Eltérés van a felhasználói adatok tárolása terén is. Például helyileg nem tárolunk felhasználóról információkat, az kizárólag a felhőben tárolódnak a Firestore és Firebase Authentication jóvoltából.

7.1. ábra – Egyed-kapcsolat diagram

A 7.1. ábrán látható a rendszer egyed-kapcsolat diagramja. Kiemelném, hogy az ábrán feltüntetett kép url-ekhez tartozó képek a felhőben a Firebase Storage-ban lettek eltárolva. Offline is elérhetőek, mivel néhány képet az eszközön is eltárol.

**7.1. Felhasználó**

A felhasználók regisztrálást követően jönnek létre. A hozzájuk tartozó adatok többségét csupán a Firebase Authentication tárolja, biztonsági okokból nem kerülnek külön tárolásra az eszközön. A diagramra (7.1. ábra) a számunkra fontosabb adatok kerültek fel.

Érdekesebb adatok között szerepelhet a jelszó, amit alapvetően titkosít a Firebase saját magának.

A regisztrációs módszer alapján lehet beazonosítani, hogy valaki e-mail és jelszó segítségével regisztrált-e vagy Google-val. Ez alapján jelenít meg bizonyos mezőket. Például a beállításokban jelszó módosítására csak akkor van lehetőség, ha azzal regisztráltál. Valamint a fiók törléséhez is csak akkor kell megadni a régi jelszót.

Szerepel a diagramon az *e-mail igazolt-e*. Ez azt a célt szolgálja, hogy ha e-mail és jelszóval regisztrálsz, csak akkor vált igazra, ha a kiküldött megerősítő e-mailben lévő linket meglátogatja a felhasználó. Amíg nincs megerősítve az e-mail, nem enged bejelentkezni sem.

**7.2. Megfigyelés**

A felhasználók tudnak megfigyeléseket létrehozni. Ilyenkor a megadott adatok, mint a megfigyelés neve, időpontja és leírása kerülnek bírásra, a többi adatot automatikusan állítja be az alkalmazás.

A felhasználóId mező ellenőrzése biztosítja, hogy csak a saját megfigyelésünkhöz tudjunk hozzáférni. Ezért is szerepel 1:N kapcsolat a felhasználó és megfigyelés között, mivel egy felhasználó bármennyi megfigyelést létrehozhat, de csak egy felhasználóhoz tartozhatnak.

A legutolsó módosítás dátuma alapján kerül rendezésre a megfigyelések listázása. A létrehozás dátuma alapján pedig be tudjuk azonosítani, hogy mikor készült a megfigyelésünk.

**7.3. Madár**

Madarakat is egyszerűen id alapján azonosíthatunk be, ami a Firestore-ban a dokumentum azonosítója. A különféle fajokról rengeteg információt eltárolunk.

A természetvédelmi érték egy külső kulcsként szerepel a madárban és a kapcsolat mentén azonosítható be.

A táplálékok, színek, alakok, élőhelyek N:M kapcsolata miatt eltér a Firebase és Room Database tárolás szempontjából. A Firebase enged tömbben, listában tárolni hivatkozásokat, így ott egyszerű ezt kivitelezni. A Room Database-ban nincs ilyesmire lehetőség. A több-a-többhöz kapcsolatot kapcsolótáblával kell feloldani, ahova bekerül a madár azonosítója, valamint a listában szereplő értékek egyenként, és így alkotnak egy rekordot.

**7.4. Természetvédelmi érték**

A madarak természetvédelmi értéke sokszor megegyezik, így a redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz (dokumentum esetében ez a dokumentum azonosító), valamint egy egész értéket, ami az értékét jelöli. A nulla számérték azt jelöli, hogy a madár egyáltalán nem védett.

**7.5. Táplálék**

A madarak táplálékai redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett táplálék.

**7.5. Szín**

A madarak tollainak színei redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett szín.

**7.5. Alak**

A madarak lehetséges alakjai redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett alak.

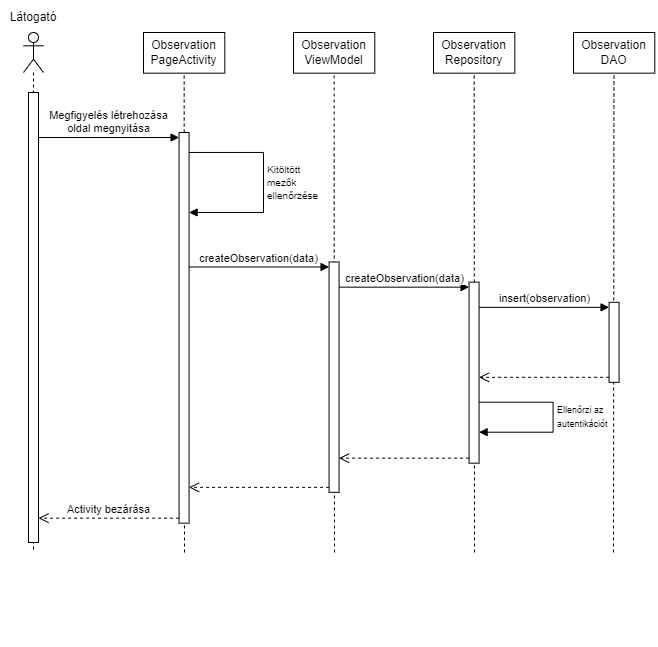
**7.5. Élőhely**

A madarak élőhelyei redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett élőhely.

**8. A rendszer magasszintű folyamatai, működése**

A Magyarországi Madarak mobilalkalmazás sok apró folyamatra épül. Mivel egy **MVVM (Model-View-ViewModel)** architektúrát követ, ami tiszta rétegződést biztosít, és különválasztja a különböző felelősségeket az alkalmazásban, a folyamatok tiszták, könnyen érthetőek és követhetőek.

Az alkalmazás egyik magasszintű folyamata a megfigyelés létrehozása, ahogy ez az egyik fő funkciója is. Ebben a fejezetben ezt a folyamatot fogom részletezni és ezen keresztül fog látszódni az alkalmazás rétegződése is.



8.1. ábra – Szekvencia diagram megfigyelés létrehozásáról

A 8.1. ábra mutatja be egy megfigyelés létrehozásának szekvencia diagramját. Ez a megfigyelések oldal jobb alsó sarkában található plusz ikonra kattintva nyitható meg.

Miután az alkalmazás betöltötte az oldalt, el lehet kezdeni az oldal kitöltését. Az első beviteli mező a megfigyelés neve, második helyen a megfigyelés időpontja található és legvégül a megfigyelés leírása. Az időpont alapvetően a megnyitáskori időpillanatra van beállítva, és a leírás csupán opcionális.

Alul a mentés gombra kattintva a rendszer ellenőrzi a megadott adatokat és jelez, ha a felhasználónak, ha valami probléma merült fel a mezőkben megadott adatokat illetően.

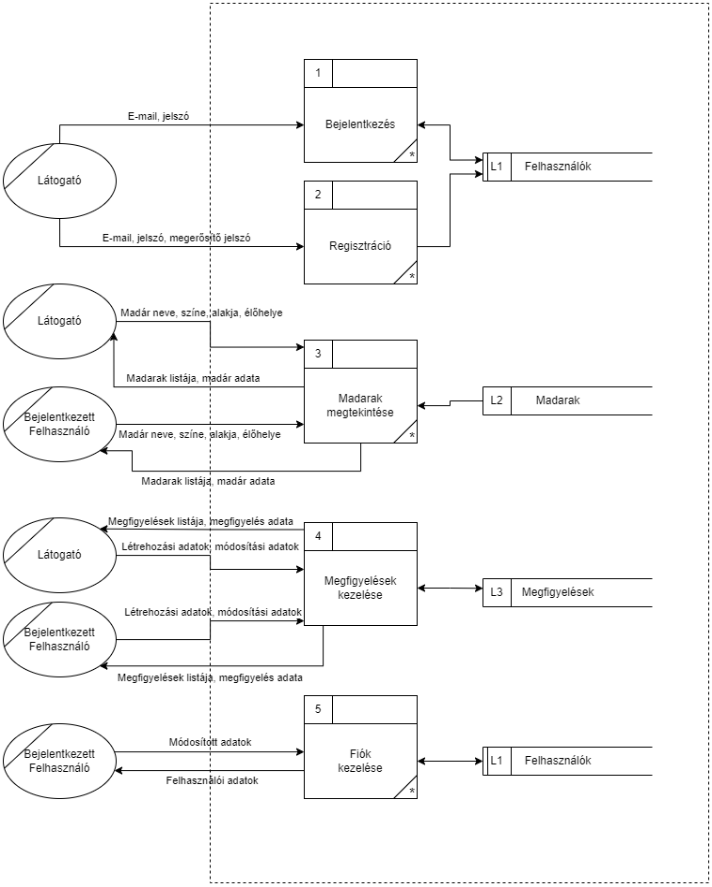
Ha a rendszer megfelelőnek találta a bevitt adatokat, úgy elkezdődik a létrehozásának folyamata. A rendszer meghívja az ObservationViewModel-nek a *createObservation()* metódusát, paraméterben átadva a felhasználó által megadott adatokkal.

Az ObservationViewModel köti össze az UI-t és a Repository-t. A meghívott metódusban csupán annyi történik, hogy meghívja az ObservationRepository-nak a *createObservation()* metódusát, aminek átadja paraméterben a kapott adatokat, valamint autentikációtól függően a felhasználó azonosítóját, vagy egy *”local”* kulcsszót, ami jelzi, hogy ez csak helyileg létezik.

Az ObservationRepository-nak van hozzáférése az observationDAO-hoz. Itt a művelethez indít az alkalmazás egy külön szálat, hogy ne foglalja le az alkalmazás fő szálát. Ezt követően létrehoz egy megfigyelés példányt. A létrehozás időpontjának és a legutolsó módosítás dátumának is beállítja a jelenlegi időpillanatot. A paraméterben kapott adatokat átadja az új objektumnak, valamint hozzárendel egy egyedi azonosítót is. Ezt követően meghívja az observationDAO *insert()* metódusát, átadva a létrehozott metódust paraméterben.

A DAO végrehajtja a műveletet, beszúrva a megfigyelést a helyi adatbázisba.

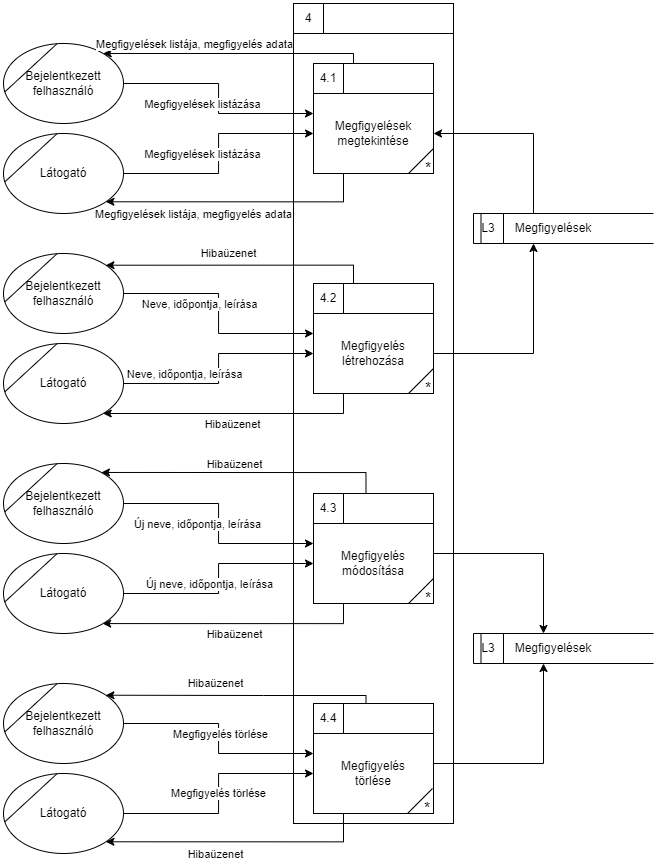
A Repository ezt követően leellenőrzi, hogy autentikált felhasználók vagyunk-e, hogy beszúrja-e a felhőbe is, de ez egy összetettebb folyamat része.

Végül véget ér a Repository metódusa, majd a ViewModel is és visszaérünk az ObservationPageActivity-re. Itt meghívódik a *finish()* metódus és véget ér az activity. Visszatérünk az ObservationActivity-re.

8.2. ábra – Logikai adatfolyam diagram, első réteg

A 8.2. ábrán látható egy logikai adatfolyam diagram első rétege, ami ezúttal a teljes rendszert írja le. Az átláthatóság miatt az egyes elemeket többször is felvettük. Az ábrán látható, hogy a folyamatok többsége egyszerű, a legösszetettebb közülük a *Megfigyelések kezelése*.

A diagram fontos tényezője, hogy látható, a folyamatok mely adattárakkal kommunikálnak és milyen módon. Egyes helyeken csak olvassuk azokat, van olyan is, ahol csak írni kell az adattárakba.

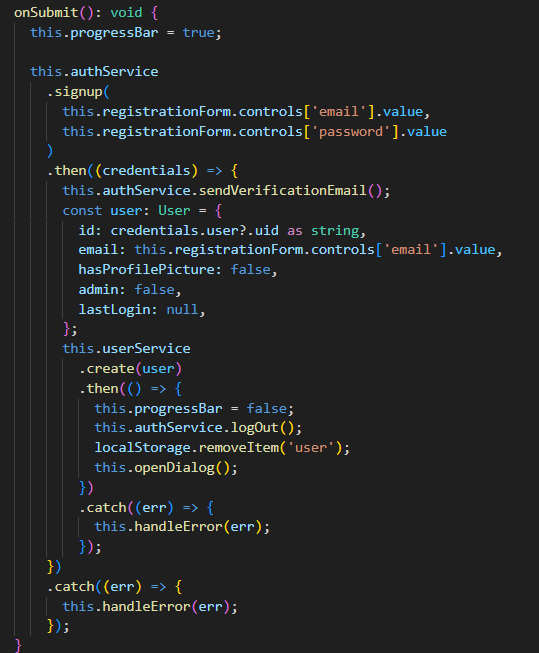
Egy megfigyelés kezelése összetett folyamat, ezért részfolyamokra bontható. Erről egy második szintű adatfolyam diagramot is készítettem, ami megtekinthető a 8.3. ábrán. Az összetett folyamat közé tartozik a megfigyelések megtekintése, megfigyelés létrehozása, módosítása és törlése. Ezekre mindegyik felhasználó képes, azonban itt is ki kell emelni, hogy a látogatók megfigyelései nem szinkronizálódnak a felhőbe.

8.3. ábra – Logikai adatfolyam diagram, második réteg

**9. Fontosabb kódrészletek ismertetése**

**9.1 Regisztráció**

A *RegisterComponenten* belül található regisztrációs űrlap kitöltésével kezdődik a folyamat. A mezők értékei az Angular[[1]](#_bookmark85) *Validators* osztályának metódusaival kerülnek ellenőrzésre, a hibaüzenetek *mat-error* tagokként jelennek meg. Helyes kitöltés esetén a *Regisztráció* gomb elérhetővé válik, ami kattintásra meghívja a 9.1.1. ábrán látható *onSubmit* függvényt.



* + 1. ábra – *onSubmit* függvény

A függvény először igazra állítja a *progressBar* értékét, ami egy töltődést jelző csíkot eredményez a fejléc alatt. Az *AuthService signup* metódusa megkapja az űrlapból származó e- mail és jelszó értéket, ami az *AngularFireAuth*-on belüli *createUserWithEmailAndPassword* segítségével hoz létre egy új felhasználót, és egy *Promise*-szal tér vissza.

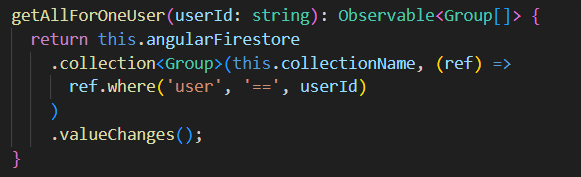
Ezután a létrejött felhasználó e-mail címére elküldi a visszaigazoló üzenetet, és egy *User* objektumot is létrehoz, amit a *UserService create* metódusának használatával ment el. A *createUserWithEmailAndPassword* lefutása után a felhasználó automatikusan bejelentkezik, de mivel ennek megerősített e-mail cím nélkül nem szabad megtörténnie, ezért a rendszer egyből ki is jelentkezteti őt az *logOut* függvénnyel. Végül az *openDialog* hatására megnyílik a *DialogComponent*, ami *data* tulajdonságában kapott adatokat jeleníti meg dialógus formában.

Ha a felhasználó létrehozása nem sikerülne, és közben hiba dobódik, akkor azt a rendszer elkapja, és meghívja a *handleError* függvényt. Itt a hibakód ellenőrzése történik, ami alapján a megfelelő hibaüzenet jelenik meg.

## Csoportok létrehozása

Csoportok létrehozása a *GropuModule*-on belüli *ListComponent* felületén történik. A lista megnyitása után a *GroupService getAllForOneUser* metódusával a felhasználó összes csoportja listázásra kerül. Ez az *AngularFirestore collection* függvényének segítségével történik, aminek átadásra kerül a kollekció neve, majd szűrés történik a felhasználó ID-jára, mint ahogy az a

* + 1. ábrán is látható.



9.2.1. ábra – *getAllForOneUser* függvény csoportok lekérésére

A megkapott csoportokból két *MatTableDataSource* osztálypéldány jön létre, az egyik a teljes listát tartalmazza, míg a másik annak egy másolatát, amin szűrések történhetnek. Az adatok táblázatba kerülése után arra egy név alapján csökkenő rendezést alkalmaz a rendszer.

A lista fölött látható *Új* gombra nyomás által hívódik meg az *openEdit* függvény. Ez az *editComponentet* nyitja meg dialógus formában, ahol a csoportok listája mellé a felhasználó által létrehozott összes résztvevő is lekérésre kerül.

A megjelenő űrlap név mezőjét a *Validators* osztály segítségével ellenőrzi a rendszer, emellett saját validátor függvényeket is létrehoztam. Például az *existenceValidator* felel azért, hogy ne létezzen egy felhasználónak két egyforma nevű csoportja – ehhez a lekért csoportokon végigiterál, és ha egyezést talál, egy hibaüzenetet jelenít meg.

A csoport tagjainak kiválasztásához az Angular Material[[19]](#_bookmark103) Chips és Autocomplete komponensét használtam. Ezek rendelkeznek saját tulajdonságokkal, amik meghatározzák, hogy melyik függvény hívódjon meg, ha például a felhasználó belekattint a mezőbe, esetleg kiválaszt vagy eltávolít egy elemet. Szöveges érték megadása esetén a *filter* függvény szűri a résztvevők nevét a lenyíló listában. Résztvevők hozzáadására két függvény, az *addMember* és a *selectedMember* áll rendelkezésre, mindkettő hozzáad egy új elemet a kiválasztott résztvevők listájához. Az előbbi az Autocomplete (szöveg beírása, majd enter vagy vessző billentyű lenyomása), az utóbbi a Chips (a lenyíló listából kattintással egy elem kiválasztása) komponenssel történt interakció hatására fut le.

A *Mentés* gomb megnyomása az *onSubmit* függvényt hívja meg. Itt létrejön egy új tömb, amibe a mezőben szereplő tagok ID-i kerülnek. Ezt a rendszer átalakítja *Set* adattípusra, aminek tulajdonsága, hogy minden értéket csak egyszer tárol, így kerülnek eltávolításra a többször megadott résztvevők. Utána ezt visszaalakítja tömbbé, amit átad egy új *Group* típusú objektumnak. A csoport többi mezőjének is értéket ad, a nevet az űrlap *Név* mezőnek értéke alapján, a létrehozó felhasználót pedig a local storage-ban tárolt ID alapján állítja be.

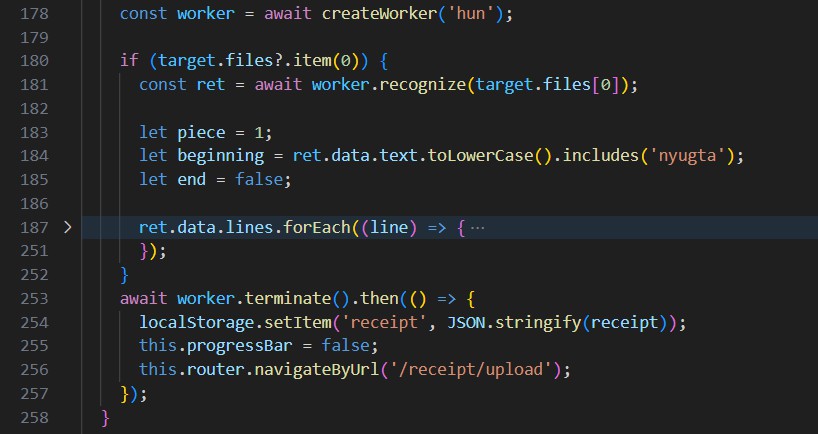
Végezetül a *GroupService create* metódusának segítségével a létrehozott csoportot a rendszer az adatbázisba is elmenti, a dialógus ablakát pedig bezárja. A *ListComponent* a *MatDialogRef* osztály *afterClosed* metódusának köszönhetően értesül az eseményről, és az új csoportot figyelembe véve rendezi a táblázatot.

## Nyugta beolvasása képről

A képfeltöltés a *ReceiptModule ListComponent* oldalán történik a *Képfeltöltés* gombra kattintással. Ekkor az *openDialog* függvény fut le, ami megnyitja a *DialogComponentet*. Ez a komponens rendelkezik egy *submitEvent* nevű *EventEmitterrel*, ami akkor bocsát ki eseményt, ha a felhasználó a dialóguson is a *Képfeltöltés* gombot választja. Ennek hatására a rendszer a listán belüli fájlfeltöltő HTML input elemre nyom, ami a felhasználó elől rejtve van.

Ezután az *uploadImage* függvény meghívása következik. Itt ellenőrzésre kerül, hogy a felhasználó ténylegesen töltött-e fel fájlt, és annak megfelelő-e a fájlformátuma; ha nem, akkor egy dialógus nyílik meg, ami tájékoztatja őt a hibáról.

Létrejön egy üres nyugta, aminek a felhasználó tulajdonsága a local storage *user* értéke, a többi attribútumba pedig a típusnak megfelelően üres szöveg vagy lista, esetleg 0 kerül.



* + 1. ábra – Tesseract.js[[7]](#_bookmark91) *Worker* használata

Mint ahogy az a 9.3.1. ábrán is látható, a Tesseract.js[[7]](#_bookmark91) *Worker* osztályát a *createWorker* metódussal példányosítottam, *’hun’* paraméter átadásával, aminek köszönhetően az magyar nyelvet képes felismerni. A *target.files[0]* tartalmazza a feltöltött képfájlt, ami – miután a létezése ellenőrzésre került – átadódik a *Worker recognize* metódusának. Ez felel a képen való szöveg felismeréséért, ami utána a *ret* változóban tárolódik.

A *ret.data* tartalmazza a képfelismeréshez kapcsolódó adatokat. Ezen belül több tulajdonság is található, például a *confidence*, ami egy százalékos érték a felismert szöveg helyességének biztosságáról. A szöveg többféleképpen is elérhető, a *text* attribútum egyetlen sztringkén, a *paragraphs* bekezdésenként, a *lines* soronként, a *words* szavanként a *symbols* pedig betűnként egy-egy tömbben tárolja azt.

Én a nyugták felépítése miatt a *lines* attribútumot választottam, aminek elemein a program egy *for each* ciklussal iterál végig. A 9.3.1. ábrán a cikluson belüli kódrészlet helytakarékosság miatt nem szerepel. Itt kapnak értéket a nyugta tulajdonságai.

A bolt neve az első nem üres sor tartalma, hiszen egy hagyományos nyugtán is itt helyezkedik el.

Ha a sor tartalmaz pontot, akkor annak szavait a rendszer egyenként megpróbálja dátummá alakítani. Amint sikerrel jár, az értéket elmenti a vásárlás dátumaként.

A végösszeget általában az *összesen*, esetleg az *összeg* vagy *fizetendő* szavak előzik meg, így az ezt követő szám és szöveg lesz a nyugta végösszege, valamint pénzneme. Ehhez a *converToSumAndCurrency* metódust használja, ami eltávolítja a sorból a felesleges karaktereket, majd végigiterál a szavakon. Ha számot talál, akkor azt hozzáadja a szöveg típusú *number* változóhoz. Azért szükséges, hogy szövegként tárolja, mivel előfordulhat, hogy a nyugtán a végösszeg ezresenként szóközökkel van tagolva, és így külön szavakban találhatók az összeg számjegyei. A pénznemet esetében minden talált szöveget megpróbál megkeresni a pénznemek szimbólumai és nevei között – ha sikerül megtalálnia, azt az értéket menti el.

A nyugtában található bizonyos szavakat, mint a *cikkszám*, *sorszám* és *részösszeg* a rendszer figyelmen kívül hagy. A termékek általában a *nyugta* szó után helyezkednek el, így ha ez a szöveg megjelenik, a *beginning* érték hamisra áll, ami jelzi, hogy a feldolgozás elhagyta a nyugta elejét. A végösszeg és a pénznem beállítása a termékek végét jelzi, így ekkor az *end* változó kap igaz értéket, mert ettől kezdve már a nyugta vége következik.

Termékek beolvasása akkor történik, ha elhagytuk a *nyugta* szót, de még nem értük el a végösszeget, tehát a *beginning* és az *end* is hamis. Megvizsgálásra kerül, hogy a sor számmal kezdődik-e, és ha igen, ez lesz a következő termék darabszáma. Egyéb esetben termékről van szó, ami a *convertToProduct* funkció segítségével kerül feldolgozásra. Ha az előtte lévő sorban nem darabszám szerepelt, akkor az automatikusan 1 lesz.

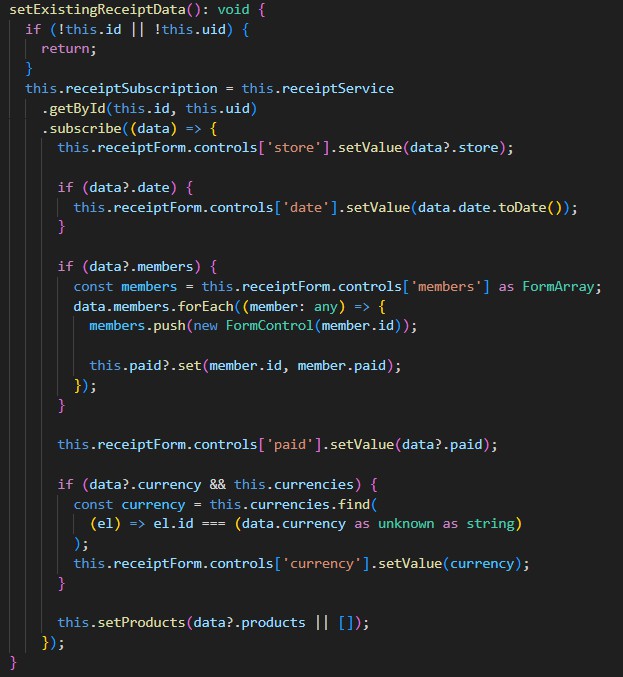
A *convertToProduct* először a termék árát olvassa be, a soron visszafelé haladva, hiszen az ár a termékek jobb oldalán található. A végösszegnél említett indok miatt a számok itt is szövegként kerülnek először tárolásra. A beolvasás addig halad, amíg el nem ér egy olyan szöveget, amit nem lehet számmá konvertálni, és a szöveg nem mínusz jel, hiszen negatív számok is szerepelhetnek a nyugtán. Emellett a beolvasás folytatódik, ha még nem sikerült egy számot sem beolvasni, mivel előfordulhat, hogy az ár mellett a pénznem is szerepel. A termék neve az árat megelőző szavak összekapcsolásából jön létre. Ha mindkét értéket sikerült kiolvasni, a rendszer egy *Product* objektumba menti őket, a metódus ezzel tér vissza. Az így felismert termékek a nyugta *product* adattagjában tárolódnak.

A beolvasás végét a *Worker terminate* metódusa által visszaadott *Promise* teljesülése jelzi. Ezt követően a létrejött nyugta objektum ideiglenesen tárolásra kerül a *local storage*-ban *receipt* néven, majd megnyílik a nyugta *EditComponent* komponense a */receipt/upload* URL használatával.

## Nyugta szerkesztése

Ez a folyamat a *ReceiptModule EditComponent* komponensén belül történik. Ez az oldal több esetben nyílhat meg: nyugta létrehozásakor (bejelentkezett vagy vendég felhasználó), képfeltöltéskor és szerkesztéskor. Ekkor más az URL utolsó tagja is (a fenti sorrendben: *new*, *guest*, *upload*, *edit:id*), valamint a komponens viselkedésében is vannak kisebb eltérések.

A pénznemek listája minden esetben lekérésre kerül a *CurrencyService getAll* metódusa által, de résztvevők és csoportok listája csak bejelentkezett felhasználó esetén kap értéket. Szerkesztés (*edit:id*) és képfeltöltés (*upload*) esetén már létezik egy nyugta objektum, aminek értékei előbbi esetén a *setExistingReceiptData*, utóbbinál pedig a *setUploadedReceiptData* meghívásával kerülnek az űrlap mezőibe. A kettő közötti fő különbség, hogy a *setExistingReceiptData* a *ReceiptService getById* metódusa segítségével kéri le a nyugtát, míg a *setUploadedReceiptData* a local storage *receipt* értéke alapján dolgozik. Ezen kívül az adatokat hasonlóan állítják be, és a *setProducts* függvényt mindkettő meghívja, ahol minden termék számára külön *FormGroup* jön létre a *formBuilder.group* futtatásával. A *setExistingReceiptData* függvény megtekinthető a 9.4.1. ábrán.



* + 1. ábra – *setExistingReceiptData* függvény

Mivel ezen az oldalon is szerepelnek az Angular Material[[19]](#_bookmark103) Chips és Autocomplete komponensei, ezért itt is definiáltam csak ezekhez kapcsolódó funkciókat. A *filter* a pénznemek listáját szűri a mezőbe írt adatok alapján – ehhez hasonló a *filetPayer*, a *filterMember* és a *filterMembersAndGroups*, amik a *Fizetett*, a *Résztvevők* és a termékek *Fizet* mezőjének lenyíló listáit szűrik. Emellett minden Chips komponenshez tartozik egy-egy függvény, ami lekéri a megjelenítendő adatokat, és ami hozzáad vagy eltávolít egy értéket.

A *Mentés* gomb megnyomása hatására az *onSubmit* funkció fut le, ami a nyugta objektum létrehozásáért és adatbázisba való feltöltéséért felelős.

A nyugta tagjai egy *Setbe* kerülnek, hogy a duplán szereplő résztvevők törlődjenek.

Bejelentkezett felhasználó esetén a résztvevők ID-ja, vendég esetén a mező értéke tárolódik.

Az űrlapban szereplő termékeken a rendszer *for each* segítségével megy végig, ezen belül a termék fizetőin is végigiterál. Bejelentkezett felhasználó esetén ellenőrzésre kerül, hogy csoportról van-e szó, ebben az esetben a csoportot a rendszer felbontja, és a résztvevőit egyesével a termék fizetői közé adja, ellenkező esetben a résztvevő ID-ját használja. Vendég felhasználónál a *Fizet* mező értékei lesznek a fizetők. A termékekből ezután egy-egy *Product* objektum készül. Emellett minden termék ára hozzáadódik a nyugta teljes végösszegéhez.

Miután a termékek listája értéket kapott, a résztvevők tömb formátumra alakítása is megtörténik. Ezt követően a duplikátumokat már nem tartalmazó résztvevőkön is végigmegy a rendszer. Egy-egy objektum kerül hozzáadásra a nyugta résztvevői közé, ami bejelentkezett

felhasználó esetében az ID-t és az eddig kifizetett összegét tartalmazza, vendég esetén pedig egy ID-t és egy nevet, amik megegyeznek a mező értékével.

A résztvevők kifizetett összege új nyugta esetén 0, kivéve azét, aki a nyugtáért fizetett, neki nincs tartozása. Már létező nyugtánál ez az érték nem változik a szerkesztésnél.

A *Receipt* objektum a nyugta fent részletezett adataiból, valamint a további mezők értékeiből jön létre. Vendég esetén a local storage-ban egy *receipt* objektum jön létre, míg bejelentkezett felhasználónál a *ReceiptService create* (új nyugta) vagy *update* (létező nyugta frissítése) metódusa segítségével az adatok az adatbázisban is rögzítésre kerülnek.

A mentés befejeztével navigáció történik a nyugta részletes oldalára.

## Nyugta részletes oldala

A *ReceiptModule PreviewComponent* komponense reprezentálja a nyugta részletes oldalát. A felület megnyitása után bejelentkezett felhasználó esetén a *ReceiptService getById* metódusával kéri le a rendszer a megjelenítendő nyugtát, aminek azonosítóját az URL-ből olvassa ki. Ha a keresett nyugta nem létezik, a nyugták listájára történik átirányítás. Vendég felhasználó esetén a local storage-ban szereplő nyugta objektum kerül átkonvertálásra. Amennyiben a vendég úgy próbálná megnyitni a felületet, hogy a local storage-ban nem szerepel nyugta, vagy a nem megfelelő URL-t próbálta használni, őt a nyugta létrehozásának oldalára irányítja át a rendszer.

A nyugta egy *Receipt* objektumként tárolódik az oldalon. A *currency* mezőjében található pénznem ID-ját használva a rendszer a pénznem objektumot is lekéri a *CurrencyService getById* metódusának használatával.

A *calculatePrices* függvény fut le ezek után, ami a tagok fizetendő összegét állítja be. Ehhez először a *needToPay Map*nek kulcs-érték párokat állít be, a kulcs a nyugta egyes résztvevőinek ID-ja, az ehhez tartozó értékek pedig kezdetben mind 0-k. Ezt követően végigiterál a termékeken, azokon belül pedig az egyes fizető résztvevőkön. Minden egyes résztvevő esetében a termékek árát elosztja annak fizetőinek számával, és ezt hozzáadja a *needToPay* adott résztvevőhöz tartozó, eddig számolt értékhez.

Következőként a *getMembersFromReceipt* függvény kerül meghívásra. Itt is végigmegy a nyugta résztvevőin, ahol vendégek esetén a pays attribútumuknak értékül adja a *needToPay* megfelelő értékét, kerekítve. Bejelentkezett felhasználók esetén a résztvevők itt kerülnek lekérésre, a *MemberService getById* metódusának használatával. A résztvevők *paid* attribútuma is értéket kap a nyugtában tárolt azonos nevű adattag alapján. Előfordulhat, hogy adott ID-jú résztvevő nem létezik, ha az törlésre került, ebben az esetben egy új *Member*

objektum jön létre, aminek neve *\*Törölt résztvevő\**. Az így beállított résztvevők egy listához lesznek adva, ami az oldalon egy táblázatként jelenik meg.

A felhasználónak lehetősége van szerkeszteni a nyugtát, ekkor az *editReceipt* függvény megnyitja a szerkesztési oldalt. A törlés ikonra kattintva a *deleteReceipt* hívódik meg, ami egy *DialogComponent* dialógust nyit meg figyelmeztető üzenettel. Ha a felhasználó itt az *Ok* gombot választja, a *submitEvent* hatására kitörlődik a nyugta a *ReceiptService delete* függvényének használatával, majd navigáció történik a nyugták listájára.



* + 1. ábra – Nyugta letöltéséért felelős kód

A nyugtát ki is lehet exportálni, a letöltés ikonra kattintva a felhasználó egy lenyíló menüből választhatja ki a CSV vagy JSON fájlformátumot. A kiterjesztés átadásra kerül a *downloadReceipt* metódusnak. Ez először kiszűri a nyugta objektumból a kimentendő adatokat, ami az egyes résztvevők neve, fizetett és fizetendő összege, valamint a nyugta boltja, formázott dátuma, pénzneme, végösszege és a fizető neve. Ezután készít egy URL-t, ami az adattípust, a karakterkészlet, és az *encodeURIComponent* metódussal kódolt adatokat tartalmazza.

CSV választása esetén a kódolás a *createCsv* metódus által visszaadott szövegen történik. Itt először definiálásra kerül a CSV fájl fejléce, ami a nyugta kulcsait tartalmazza, majd ezek pontosvesszővel összefűzve a *rows* tömbbe kerülnek. A többi érték hasonló módon pontosvesszővel elválasztva lesz hozzáadva a tömbhöz. Ezután egy üres szöveg is a tömbbe kerül, ezzel lesz egy sor hely a nyugta alapadatai és a résztvevők táblázata között. A résztvevők esetében ismét pontosvesszővel lesznek összefűzve a kulcsok, majd az értékek. Végül a *rows* tömb elemei új sor karakterekkel elválasztva lesznek visszaadva egyetlen szövegként.

A HTML kódban található egy elrejtett link elem *download* ID-val, a letöltendő fájl URL-jét a rendszer ennek a *href* attribútumának adja értékül. Az URL létrehozása, a HTML link elem attribútumainak megadása és az arra való kattintással a letöltés megkezdése látható a

9.5.1. ábrán, ahol a *downloadData* a nyugta kimentendő adatait tartalmazza.

A nyugta részletes oldalán lehetőség van a résztvevők által kifizetett összeg módosítására is. A résztvevők táblázatában minden sorban található egy jelölőnégyzet, aminek

megnyomása a *payForMember* funkciót hívja meg. Itt a nyugta résztvevői közül megkeresi a megfelelőt, akinek a *paid* attribútumát a *needToPay Mapben* hozzá tartozó értékre állítja. A nyugtán ezután átalakítást végez a *transformReceipt* függvénnyel, ami eltávolítja a nyugtából a csak a frontenden használt értékeket. Az így átalakított nyugtát a *ReceiptService update* metódusával tölti fel a szerverre. Végül újraszámolja a *paidSum* értéket.

Ha egy résztvevő nem a teljes összeget fizette ki, hanem annak csak egy részét, a felhasználó ezt is be tudja állítani a résztvevők táblázatában látható szerkesztés gombra kattintva. Ekkor az *openPaidDialog* hívódik meg, ami a *PaidComponentet* nyitja meg dialógus formában. A komponensnek átadódnak a nyugta tagjai és a pénznem szimbóluma.

A *PaidComponenten* belül először minden résztvevőnek létrejön egy űrlapmező, aminek kezdeti értéke az általa eddig fizetett összeg. Az űrlap mentésekor az *onSubmit* függvény hívódik meg, ami a résztvevőknek beállítja az új fizetett értékét, elküldi a *PreviewComponentnek* a *submitEvent* eseményt a résztvevőkkel, és bezárja a dialógust.

A *PreviewComponenten* belül a *submitEvent* hatására a rendszer végigiterál a nyugta résztvevőin, és beállítja nekik az új fizetett értéküket, valamint az összesen fizetett mennyiséget is újraszámolja. Ezt követően a korábban már részletezett *transformReceipt* használatával átalakítja a nyugtát, amit a *receiptService update* metódusával küld el az adatbázisba.

A *PaidComponent* megnyitása és az eredmény hatására a résztvevők fizetett értékének módosítása látható a 9.5.2. ábrán.



* + 1. ábra – *openPaidDialog* függvény

**10. Tapasztalatok, továbbfejlesztési lehetőségek**

## Alkalmazás megtervezése

Az Nyugtafelosztó fejlesztését a tervezési fázissal kezdtem, amikor még egészen más elképzeléseim voltak az alkalmazásról. Elsőként számos diagramot elkészítettem, amik akkor útmutatást adtak a kód alapjának elkészítéséhez, de ahogy haladtam az implementálással, egyre kevesebb részletet tartottam meg belőlük.

Bár a kezdeti tervek nem mindenhol egyeznek a végeredménnyel, ez nem jelent problémát az elkészült alkalmazás tekintetében. A korai ötletek felülvizsgálásával végezetül sikerült olyan megoldásokat kitalálnom, amik jobban illeszkednek az alkalmazás motivációjához és céljához.

## Ütemterv, feladatok

A fejlesztési fázis elején készítettem egy ütemtervet, amit sajnos nem mindig sikerült tartanom. Ennek egyik fő oka az volt, hogy az implementálás során további új funkciók jutottak eszembe, amik az alkalmazás javára válhatnak, és ezek megtervezésével, valamint megvalósítása nem számoltam a tervezés során.

Ami az időm megtervezésében sokat segített, az a GitLab[[6]](#_bookmark90) Issue board funkciója volt. Ide vettem fel a feladatokat, amiket el kellett végeznem, amiket próbáltam kisebb részekre, esetleg részfeladatokra osztani. Ez által mindig tudtam, hogy nagyjából mennyi van még hátra a fejlesztésből, emellett egy-két hetes időszakokban mindig előre el tudtam tervezni, hogy min fogok dolgozni.

## Szakmai ismeretek

A szakdolgozatomhoz azért választottam az Angular[[1]](file://localhost/C:/Users/hanna/OneDrive/Dokumentumok/Szakdoga/nyugtafeloszto/szakdolgozat/SomogyváriHanna_GLXGEU_szakdolgozat.docx%23Angular) keretrendszert Google Firebase[[2]](file://localhost/C:/Users/hanna/OneDrive/Dokumentumok/Szakdoga/nyugtafeloszto/szakdolgozat/SomogyváriHanna_GLXGEU_szakdolgozat.docx%23GoogleFirebase) backenddel, mivel már rendelkeztem tapasztalattal ezek terén, emellett a tudásom továbbfejlesztésére is lehetőség nyílt.

A fejlesztésben nehézséget okozott bizonyos Angular Material[[19]](#_bookmark103) komponensek használata. Ennek oka az volt, hogy a komponensek dokumentációja általában csak az alap használati eseteket fedi le, én viszont bonyolultabb megoldásokat szerettem volna megvalósítani, mint az Autocomplete és Chip komponensek egymásba ágyazása, és ezekben adatbázisból lekért elemek megjelenítése.

A Google Firebase[[2]](file://localhost/C:/Users/hanna/OneDrive/Dokumentumok/Szakdoga/nyugtafeloszto/szakdolgozat/SomogyváriHanna_GLXGEU_szakdolgozat.docx%23GoogleFirebase) esetén a felhasználók hozzáférését korlátozó biztonsági szabályok implementálása volt újdonság számomra. Itt kezdetben nehezen ment ezek meghatározása, hiszen nem voltam tisztában a lehetőségekkel és a megfelelő módszerekkel. Végül sikerült elsajátítanom a szabályok írását, és örülök, hogy ezzel a tudással is gazdagodhattam.

## Továbbfejlesztési lehetőségek

Számos ötletem van, amivel az alkalmazást tovább lehetne fejleszteni, és ami jobb felhasználói élményt biztosítana.

Az egyik a fordítások bevezetése lenne, hogy a Nyugtafelosztó szélesebb közönség körében elérhetővé váljon. Ezzel kapcsolatban az egyik probléma a nyugták formátuma, ami országonként más-más lehet, ami megnehezítené a nyugtákról a megfelelő adatok kinyerését.

A képfelismeréssel kapcsolatban is van lehetőség a továbbfejlesztésre. Jelenleg ezt a Tesseract.js[[7]](#_bookmark91) végzi, ami az elképzelés bemutatásához megfelelő, de a korábban említett hibái miatt nem tekinthető teljeskörű megoldásnak. A probléma kiküszöbölhető lenne saját OCR motor fejlesztésével, ami kifejezetten nyugták felismerésére fókuszál, és nemcsak azok szövegét lenne képes felismerni, hanem a struktúra alapján azt is ki tudja következtetni, hogy melyik karakterlánc mit reprezentál.

Az alkalmazás azzal is továbbfejleszthető lenne, ha a felhasználók interakcióba léphetnének egymással. Így nyugtákhoz nem csak résztvevő objektumokat lehetne hozzáadni, hanem más felhasználókat is, akik értesülnének róla, ha tartozásuk van. Emellett az összeg kifizetését nem csak egy jelölőnégyzettel lehetne jelezni, hanem az alkalmazásban egy fizetési szolgáltatás igénybe vételével akár a tartozás átutalására is lehetne lehetőség.

## 10.5. Összegzés

Összességében úgy érzem, végül sikerült egy olyan alkalmazást fejlesztenem, aminek funkciói kielégítik az elvárásokat, és egyszerűen, gyorsan képes egy nyugtát felosztani.

Bár a fejlesztés során voltak nehézségeim, de ezek hatására sikerült tovább bővítenem a tudásom a használt technológiák körében. Remélem, hogy a megszerzett tapasztalatokat a későbbiekben is kamatoztatni tudom, és ezek által tovább fejleszthetem a Nyugtafelosztó alkalmazást.

**Irodalomjegyzék**

1. Angular: <https://angular.io/> (Utolsó megtekintés: 2024. 05. 22.)
2. Google Firebase: <https://firebase.google.com/> (Utolsó megtekintés: 2024. 05. 22.)
3. draw.io: <https://app.diagrams.net/> (Utolsó megtekintés: 2024. 05. 22.)
4. Visual Studio Code: <https://code.visualstudio.com/> (Utolsó megtekintés: 2024. 01. 18.)
5. Sass, SCSS: <https://sass-lang.com/> (Utolsó megtekintés: 2024. 01. 18.)
6. GitLab: <https://about.gitlab.com/> (Utolsó megtekintés: 2024. 05. 22.)
7. Tesseract.js: <https://tesseract.projectnaptha.com/> (Utolsó megtekintés: 2024. 01. 18.)
8. Zoho Expense: <https://www.zoho.com/expense/> (Utolsó megtekintés: 2024. 01. 21.)
9. Verify: <https://www.veryfi.com/> (Utolsó megtekintés: 2024. 01. 21.)
10. Expensify: <https://www.expensify.com/> (Utolsó megtekintés: 2024. 01. 21.)
11. Shoeboxed: <https://www.shoeboxed.com/> (Utolsó megtekintés: 2024. 01. 21.)
12. Dext Prepare: <https://dext.com/en/products/prepare> (Utolsó megtekintés: 2024. 01. 21.)
13. Invoice Maker, Invoice Calculator, Invoice Generator: <https://saldoinvoice.com/invoice-calculator/> (Utolsó megtekintés: 2024. 01. 21.)
14. EveryoneAllin: <https://everyoneallin.com/bill-splitter/online-bill-splitter/> (Utolsó megtekintés: 2024. 04.

25.)

1. Best Bill Splitter: <https://bestbillsplitter.com/> (Utolsó megtekintés: 2024. 01. 21.)
2. Kittysplit: <https://www.kittysplit.com/en/> (Utolsó megtekintés: 2024. 01. 21.)
3. IRIS KashFlow: <https://www.kashflow.com/> (Utolsó megtekintés: 2024. 01. 21.)
4. xero: <https://www.xero.com/> (Utolsó megtekintés: 2024. 01. 21.)
5. Angular Material: <https://material.angular.io/> (Utolsó megtekintés: 2024. 04. 30.)
6. AngularFire: <https://firebaseopensource.com/projects/firebase/angularfire/> (Utolsó megtekintés: 2024.

01. 22.)

1. Flaticon – alapértelmezett profilkép, készítette Freepik : <https://www.flaticon.com/free-icons/user> (Utolsó megtekintés: 2024. 02. 27.)
2. Ngx Image Cropper: <https://www.npmjs.com/package/ngx-image-cropper> (Utolsó megtekintés: 2024. 04. 25.)
3. Wikipédia – Forgalomban lévő pénznemek listája: <https://hu.wikipedia.org/wiki/Forgalomban_l%C3%A9v%C5%91_p%C3%A9nznemek_list%C3%A1ja> (Utolsó megtekintés: 2024. 04. 28.)

**Nyilatkozat**

Alulírott Somogyvári Hanna programtervező informatikus szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2022. 05. 22.

Aláírás

**Köszönetnyilvánítás**

Szeretnék köszönetet mondani a témavezetőmnek, Dr. Bilicki Vilmos egyetemi adjunktusnak, aki útmutatást adott, felügyelte a haladásom, és tanácsokkal látott el.

Emellett szeretnék köszönetet mondani a családomnak a támogatásukért, megértésükért és az ötletemért való lelkesedésért.

Végezetül hálával tartozok a páromnak is, aki mellettem volt, ha nehézségekbe ütköztem, és velem együtt örült, amikor sikereket értem el.

**Elektronikus melléklet**

1. Az alkalmazás GitLab[[6]](#_bookmark90) elérhetősége: <https://gitlab.com/Hanna723/nyugtafeloszto> (Utolsó megtekintés: 2024. 05. 22.)
2. A hosztolt alkalmazás linkje: <https://nyugtafeloszto.web.app/> (Utolsó megtekintés: 2024. 05. 22.)