



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Hajas András

**ANDROID ALAPÚ CSOPORTOS
FELADAT KEZELŐ RENDSZER
TERVEZÉSE ÉS
MEGVALÓSÍTÁSA JAVA SPRING
HÁTTÉRRENDSZERREL**

EKLER PÉTER

BUDAPEST, 2020

Tartalomjegyzék

Összefoglaló	6
Abstract.....	7
1 Bevezetés	8
1.1 Fő technológia választása	8
1.2 Az alapötlet bemutatása	9
1.3 A dolgozat szerkezete	11
2 Előkészület.....	12
2.1 Hasonló projektek	12
2.1.1 Trello.....	12
2.1.2 Apple Reminders	13
2.2 Korábbi tudás	13
2.2.1 Interjúk.....	14
2.3 Források	14
2.3.1 Baeldung	14
2.3.2 Android dokumentáció és Mitch.....	15
2.4 Technológiák és eszközök	15
2.4.1 Kotlin	15
2.4.2 dbdiagram.io	16
2.4.3 Postman.....	16
2.4.4 MockFlow	16
2.4.5 MySQL Database és Workbench.....	17
2.4.6 LiveData és MVVM	17
3 Tervezés	19
3.1 Követelmények	19
3.2 Dizájn.....	20
3.3 Adatmodell.....	21
3.4 Android architektúra	23
3.5 Webszolgáltatás architektúra	24
4 A megvalósítás részletes bemutatása.....	26
4.1 Bejelentkezés és regisztráció	26
4.1.1 Gyár tervezési minta	26

4.1.2 Bejelentkezést vezérlő adat.....	27
4.1.3 Bejelentkezett felhasználó mentése	27
4.1.4 Jelszó tárolása	28
4.1.5 Regisztrálás képpel	28
4.1.6 Dizájn.....	29
4.2 Fő nézet.....	31
4.3 Csoportok választása.....	31
4.3.1 MVVM.....	32
4.3.2 RecyclerView.....	33
4.3.3 Csoport létrehozása.....	33
4.4 Csoport részletes nézete.....	35
4.4.1 Tagok tárolása.....	36
4.4.2 Csatlakozás csoporthoz.....	37
4.4.3 Tag lista elem dizájnja	37
4.5 Feladat létrehozása.....	37
4.5.1 A nézet dizájnja és a feladat tárolása	39
4.5.2 Fontossági szint választás	40
4.5.3 Földrajzi hely választása.....	41
4.5.4 Háttérrendszerrel kapcsolatos fejlesztések	43
4.6 Feladatok böngészése.....	44
4.6.1 Feladatok csoportokba tagolása	45
4.6.2 Lista elem dizájnja	47
4.7 Feladatok részletes nézete.....	48
4.7.1 A tulajdonságok megjelenítése	49
4.7.2 Az feladat adatainak elérése	50
4.7.3 A nézet dizájnja	50
4.7.4 ISP megsértése.....	51
4.8 További gondolatok	52
4.8.1 Adatbázis tulajdonságok.....	52
4.8.2 Android erőforrások.....	53
5 A megvalósítás után	55
5.1 A tesztelés	55
5.1.1 Első eset	55
5.1.2 Második eset	56

5.1.3 Harmadik eset	56
5.1.4 Értékelés.....	57
5.2 A jövőbeni fejlesztések	57
5.2.1 Az új funkciók	57
5.2.2 Következő verzió érkezése	58
6 Az elvégzett munka összefoglalása	59
7 Irodalomjegyzék.....	60

HALLGATÓI NYILATKOZAT

Alulírott **Hajas András**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. 10. 20

.....
Hajas András

Összefoglaló

Mivel rengeteg dolgunk van a mindennapokban, nehéz számon tartani minden feladatunkat és sokszor elkésünk vagy elfelejtünk dolgokat emiatt. A szakdolgozatomban egy olyan rendszer megvalósítását tűztem ki célul, ami ezt a problémát hivatott megoldani. Egy *mobil alkalmazást* készítettem, amiben feladatokat tudunk kezelni más felhasználókkal együtt. Ezt az *alkalmazást* egy háttérrendszerrel akartam támogatni, ami egy *webszolgáltatásból* és egy *adatbázisból* áll. Ezen rendszer megvalósítása közben törekedtem olyan technológiák és módszerek használatára, amiknek manapság jó a megítélése és modern. Valamint általános célom volt a dolgozat közben olyan eszközöket használni, amiket még nem tudtam elsajátítani, hogy még jobban mélyíteni tudjam a tudásom.

Ez a dokumentum azon kívül, hogy a fél éves munkámat mutatja be, magyarázatként is szolgál a megvalósítás nem egyértelmű lépéseiben. A rendszer különböző részeihez használt eszközök bemutatását és a működés megtervezésének lépéseit is tartalmazza. Ezen kívül hosszabban elmagyaráztam a felmerülő problémákra a megoldásom. Lehet ez akár Android platformot és architektúrát érintő probléma, Java Spring keretrendszeres webszolgáltatásban felmerülő gond vagy egy MySQL adatbázisban beállított tulajdonságok is. Továbbá sok megjelenéssel és ergonómiával kapcsolatos elgondolást is tartalmaz és általános problémát, amibe bármilyen típusú alkalmazás fejlesztésekor belefuthat az ember. Végül tartalmazza egy alap szintű manuális tesztelésnek a leírását, ami alátámasztja az elért eredményeimet.

A munka során a kitűzött célokat sikerült megvalósítanom és számos új technológiával megismerkedtem. Mindenképpen ajánlanám a kezdő Android és egyéb fejlesztőknek, mert számos olyan megvalósítás és elgondolás van benne, ami egy pályakezdő programozónak jól jöhet. Valamint a használt eszközök nagy részét úgy választottam ki, hogy versenytársait is próbáltam, így egyszerűen használható és jól működő eszközöket is megismerhet az olvasó.

Abstract

Since our everyday life is getting busier and busier, it is hard to keep track of all our tasks. Sometimes we tend to be late or forget something because of this. In my thesis I was aiming to create a system which is capable of solving this problem. I developed a mobile application in which we can manage tasks while collaborating with other users. I wanted to support this application with a backend, which consists of a webservice and a database. While I was developing this system, I prioritized technologies and practices, which are modern and have a high reputation these days. Furthermore, my general goal while writing the thesis was to use tools that I am not completely familiar with, so I can develop a deeper knowledge.

This document next to introducing the work I have done in the semester, serves as an additional guideline for the steps which are not self-explanatory in my work. It also contains the introduction of the tools used and the desinging of the system. Moreover, there is a detailed explanation on how I solved various problems. Including problems with Android platform and architecture, Java Spring framework and properties of the MySQL database. Furthermore, it contains several thoughts on appearance and ergonomy along with general problems which can occur while developing any type of application. Finally it contains the description of a basic level manual testing, which supports my work.

I successfully achieved the goals I set and got to know several new technologies while developing my application. I would definietly recommend this to Android and other developers, since there are several solutions and thoughts which can be useful for a graduate engineer. Also, I already tried different tools, similar to the ones I ended up using. This way the reader can get to know tools which are efficient and easy to use.

1 Bevezetés

Az alábbi dokumentum lényege első sorban az, hogy bemutassa a félév során végzett munkámat. Azért választottam ezt a témát, mert az összes korábban használt technológia közül az Androidot szerettem meg a legjobban, ezért ebben jobban el akartam mélyíteni a tudásomat, mert eddig nem tudtam olyan mély szinten belenézni, ahogy szerettem volna. Ezen kívül úgy éreztem, hogy teljesen önállóan működő rendszert csak akkor tudok elérni, ha háttérrendszert is készítek hozzá valamilyen API használata helyett. Mivel a Springgel már találkoztam korábban, úgy döntöttem ezzel fogok újra dolgozni új technológia tanulása helyett.

A projekt alatt megterveztem egy feladatkezelő alkalmazást és a hozzá tartozó háttérrendszert. Ezekhez a tervekhez tartva magam implementáltam a rendszert. Végül elkészítettem egy *Android alkalmazást* és vele párhuzamosan egy *webszolgáltatást* és egy hozzá tartozó *adatbázist*. Ezután alap szinten teszteltem és terveztem továbbfejlesztési lehetőséget.

A projekt megvalósítása során több dolgot megismertem, amiből a legmérvadóbb az Androidos rendszer MVVM architektúrája. Egy ilyen architektúrát megvalósítani és gyakorolni volt a fő célom a félév a során. Ennek a keretein belül, valamivel jobban megismertem még a megfigyelhető tulajdonságok működését és a többszálú működést. Ezen kívül szívesen foglalkoztam a megjelenéssel és próbáltam a képzésem alatt tanult ergonómia ismereteimet alkalmazni a tervezésnél. Végül a tesztelésnek is megismerkedtem egy érdekes, de úgy éreztem meglepően hasznos formájával.

1.1 Fő technológia választása

Moore törvénye egy olyan fél századdal ezelőtti megfigyelésen alapszik, hogy az egy microchipre integrálható tranzisztorok száma körülbelül két évente duplázódott, miközben a chip ára a felére csökkent. Ez alapján már könnyen állíthattam azt, hogy a számítógépjeink minden évben egyre többet fognak tudni miközben egyre kevesebbe fognak kerülni. Ettől feltételezéstől manapság azonban már eltérünk. Jellemzően egyre drágábban jutunk hozzá az új készülékeinkhez, és minden évvel egyre többet költünk ezek fejlesztésére, korszerűsítésére is. Bár ezen költségek alapján azt gondolhatjuk, hogy nem volt igaza Moorenak, csak részben ez a helyzet. Ugyanis a technológia fejlődés egyes

ágainak sebessége továbbra is évről évre nagyobb léptéket vesz és még nem tudjuk, hogy mikor érhetjük el a képességünk maximumát. [1]

Moore törvénye alapján egyértelműen látszik, hogy nagyon könnyen le lehet maradni. Aki tartani akarja lépést, annak célszerű olyan technológiákkal foglalkozni, amik népszerűek és vezetik a fejlődést, hogy mindig a legújabb fejlesztésekkel, eszközökkel és módszerekkel dolgozzon és később tudjon ezekre építeni. Emiatt döntöttem úgy, hogy mobil fejlesztéssel szeretnék majd foglalkozni és ezért választottam ilyen témájú projektet szakdolgozatnak. Sokáig az asztali számítógépre történő fejlesztések vezették a piacot, viszont ahogy a technológia engedte a mobil telefonok szépen lassan elkezdtek felvenni egy számítógép funkcionalitását is és el is kezdték bővíteni azokat, kihasználva a mobilok sajátosságait. Ilyen az, hogy a mobilt az ember mindig magánál hordja, mivel elfér a zsebében, valamint emiatt tudunk következtetni például arra, hogy milyen tevékenységet végzünk és merre járunk. Továbbá egy mobil jellemzően rendelkezik kamerával és minden esetben van mikrofonja, ami egy asztali számítógép esetében nem feltétlenül igaz. Így a mobil mára az ember társává vált, és szinte az életünk minden részében segít nekünk. A gyerekek is mobilt kapnak először és legtöbbször az asztali gépet már csak munkavégzés miatt ismerik meg. Mára már több mobil internetező van, mint más platformokat használó és valószínűleg a mobil szépen lassan teljesen fel fogja váltani a számítógépet a közéletben. [2]

Ezen belül azért esett az Androidra a választásom, mivel ez a legnagyobb hatalom a mobil rendszer piacon. Továbbá ez egy nem csak mobilon használt, hanem egy széles körben elterjedt érintő kijelzős eszközökre optimalizált operációs rendszer. Így ennek a technológiának az elsajátításával nem csak a mobil fejlesztésről szerzek tudást, hanem könnyen tudok majd készíteni majd okos TV-re vagy az autóba is saját appot. Valamint én is Android felhasználó vagyok.

1.2 Az alapötlet bemutatása

Ahogy korábban említettem, a fejlődésünk egyre gyorsul. Emellett az életünk is egyre bonyolultabbá válik. Egyre nehezebb problémákkal nézünk szembe és emellett egyre több dolgot kell megtanulnunk használni. Bár ott van mellettünk a társ, aki nagyon sok területen és problémában tud nekünk segíteni, mégis nagyon sokan képtelenek kihasználni ezt. Sokszor úgy látom, hogy ez amiatt van, mert az adott személy nem tudja

úgy használni az mobilját, ahogyan szüksége lenne rá és emiatt sokszor nem is azon keresi a probléma megoldását.

Nem csak a problémák összetettsége miatt vagyunk bajban, de az életünk tempója is olyan mértékű, hogy nagyon sokan nem tudják követni. Sokszor egy határidő vagy egy apróság elmulasztásán bukjuk el a céljainkat. Továbbá a mindennapi életünkbe is felesleges feszültséget és negativitást hoz, ha elfelejtünk valamit, amire megkértek és nem is került volna nagy erőfeszítésbe. Ahogy egyre idősebbek leszünk egyre több feladat és felelősség hárul ránk míg egyszer csak észre nem vesszük, hogy teljesértékű emberek vagyunk már és egész nap problémákat vagy feladatokat kell megoldanunk otthon, munkában vagy valakinél a családban.

Ilyen sok és összetett feladatot nehezen tudnánk egyedül elvégezni, jellemzően mindenhol másokkal együtt dolgozunk. Legtöbbször csapatban dolgozunk a munkahelyünkön. Családként végezzük el a házimunkát vagy segítünk egymásnak. Továbbá nagyon sok több területre van az életünknek, ahol többen küzdünk egy közös cél eléréseért. Ilyen egy osztály találkozó megszervezése vagy egy közös program a barátokkal. Azonban még így is hajlamosak vagyunk elfelejteni a dolgokat, vagy előfordul, hogy valamit rosszul beszélünk meg és félreértéshez vezet, csak ilyenkor nem csak a saját helyzetünket keserítjük meg.

Ezeket a problémákat veszem észre a szüleimen és sok közeli ismerősömnél és sokszor magamon is, bár én már könnyen tanulok bele az új dolgokba. Nem tudjuk használni a segítséget nyújtó technológiát, aminek csak szenvedünk a megtanulásával.

Ezen problémák megoldására találtam ki a rendszert, amit megvalósítottam. Ezzel a feladatkezelő alkalmazással képesek vagyunk csoportokba szerveződni és meghatározni a teendőket és különböző kisegítő tulajdonsággal ellátni őket, mint például a feladat fontossága, földrajzi hely vagy a csoportból egy másik személy, akinek el kell majd ezt végeznie. Így minden közösség láthatja, hogy haladnak az elérni kívánt dolgokkal és nem tévesztenek szem elől semmit. Emellett törekedtem, hogy olyan felhasználói felületet valósítsak meg, ami nem jelent akadályt abban, hogy egy olyan személy használja, aki nehezen igazodik el a mobiljában vagy nincsen tapasztalata feladat menedzselésben.

1.3 A dolgozat szerkezete

A továbbiakban a dolgozat felépítése nagyjából a fejlesztési folyamatot fogja próbálni végigkísérni.

Először szó fog esni a projekt megvalósítására választott technológiákról és hasonló projektekről. Utána a feladat részletesebb elemzése és megvalósításának alap ötleteiről, valamint a tervezéséről írok röviden. Ezután jön a fő tartalmi rész, ahol a részletesen bemutatom a projektet és külön kitérek az érdekesebb problémákra, amik megoldása nem volt egyértelmű vagy nehézséget jelentett. Végül a kész projektnek a használhatóságáról, rajta végzett tesztek teljesítményéről és az utóéletéről esik szó.

2 Előkészület

Legelőször is, még a tervezés előtt kutatómunkát kellett végeznem, hogy mégis miként lenne célszerű megvalósítani a projektet. Ezalatt tulajdonképpen azt értem, hogy az ötletemhez hasonló alkalmazásokat kerestem és felmértem, hogy körülbelül milyen fő dolgokra kell majd összpontosítanom a fejlesztés során. Ezután mérlegeltem, hogy mire tudok támaszkodni a jelenlegi tudásomból és jól bevált megoldásokat kerestem azon problémákra, amikkel még nem találkoztam. Végezetül kiválasztottam az általam a legjobb ötletnek vélt technológiákat a különböző részek megoldására és készítésére.

2.1 Hasonló projektek

Ebben a fejezetben két projektet mutatok be, amit kiindulási alapul használtam az a saját *alkalmazásom* elkészítéséhez. Úgy éreztem, ezek a témában kiemelkedő eszközöknek számítanak. Tehát megpróbáltam megfigyelni mi miatt lehetnek jobbak, mint a versenytársaik és ezeket a tulajdonságokat figyelembe véve alakítottam tovább a saját elképzelésemet.

2.1.1 Trello

Bizonyára sokak által ismert alkalmazás a fejlesztői szakmában a Trello [3]. Ez egy együttműködésre tervezett alkalmazás, amivel a projektünkön végzett munkát táblákba szervezhetjük. Ezek a táblák olyan csoportokba foglalják az elvégzendő dolgot lépéseit, amiből láthatjuk például, hogy az adott funkció a fejlesztés melyik szakaszában van. Ezek a funkciók vagy lépések kártyaként jelennek meg az appban, amiket csak könnyedén dobálhatunk egyik csoportból a másik csoportba. Ezek a kártyák további információkat hordoznak a feladról, megtudhatjuk mikor kell végezni az adott feladattal, ki az, aki már dolgozik rajta vagy éppen azt, hogy a csapat melyik részének szánták a feladatot.

Alapját véve ez egy Kanban tábla, amit böngészőből tudunk használni. Egy ilyen táblát úgy lehet elképzelni, mintha jegyzetlapokat ragasztanánk a falra a problémákkal és az alapján tennénk őket valahova vagy szednénk le, hogy éppen hol tart a probléma megoldása.

Ránézésre láthatjuk, hogy a Trello szinte minden funkciót megvalósít, ami egy ilyen témában készülő applikációban előfordulhat, azonban ezzel már jóval túlmegy azon, amire nekem szükségem lesz. A részletessége miatt már inkább egy fejlesztőknek készülő alkalmazásról beszélünk, ugyanis egy projektmenedzsmentet nem ismerő személynek idegen ilyesmit látni a képernyőn, nem tudja mit kezdjen vele, nem beszélve arról, hogy a számítógépeket nehezen kezelő emberek elég nehezen tudnak csak kiigazodni a sok lehetőségben.

Mindezek ellenére a Trello nagyon sok jó ötletet adott, beszéljünk akár feladathoz rendelhető tulajdonságokról vagy a megjelenítéséről.

2.1.2 Apple Reminders

Habár Androidos projektet tervezek és magam is Android felhasználó vagyok, kevés rosszat tudok mondani az iOS használhatóságáról. Emiatt hasonló appok fejlesztésénél előszeretettel használom kiindulási alapul az Apple gyári alkalmazásait. Nem ritka, hogy olyan ötletet találok bennük, amit még korábban nem láttam és nagyban leegyszerűsítik a felhasználó dolgait. [4]

Ez egy jóval kisebb hangvételű alkalmazás, mint a Trello és nem igényel semmilyen projektmenedzsmentes előismeretet, egyértelműen kezelhető az első megnyitástól.

Megjelenésben nem hasonlít arra, amit elképzeltem az *alkalmazásomhoz*, de innen is sok elgondolást tartottam szem előtt a munkám közben. Megnyitva azt látom, hogy az app nagyobb csoportokba rendezi az egyes feladatokat, megnézhetjük egyszerre az összeset, vagy csak a fontosként megjelölteket és még néhány további kategóriát is. Ilyen elgondolás alapján próbáltam később én is csoportosítani a saját applikációmban a feladatokat. Továbbá minden funkció elérése teljesen letisztult és egyértelmű. Egy feladat felvételekor nem rak elénk felesleges információkat, mi döntjük el milyen részletekkel akarjuk létrehozni a feladatot és csak azt látjuk, amit kértünk, hogy láthassuk. Erre a fajta letisztultságra és egyszerűségre akartam, hogy hasonlítson a saját appom.

2.2 Korábbi tudás

A dolgozat elkezdésekor már ismerős volt az Android alkalmazások fejlesztése és Java Springes szolgáltatással is találkoztam. Előző féléves önlabor projektként is ezen technológiákat használva építettem alkalmazást. Azonban nagyban jellemző volt a

munkámra, hogy a problémák megoldására nem a legjobb, bevált módszereket próbáltam használni, hanem a cél csak a működőképesség elérése volt. Ezzel szemben ezen projekt megoldása során arra törekedtem, hogy a jelenleg legjobbnak ítélt sémák alapján dolgozzak és a fejlesztést jelenleg iparban is használt eszközökkel támogassam.

2.2.1 Interjúk

Szakmai gyakorlatként is próbáltam Android fejlesztőként elhelyezkedni, de erre sajnos végül nem került sor. Azonban ezen állások keresése közben mindig figyelmesen tanulmányoztam a munkához kellő követelményeket, vagy előnyt jelentő készségeket. Ezen felül volt, ahova otthoni feladatot készítettem. Ott is olyan naprakész technológiák használatát kérték a programhoz, ami segített abban, hogy felmérjem milyen területeken vagyok lemaradva és ezekkel is meg akartam ismerkedni, hogy fejlődni tudjak. Voltak kérdések a projekt megvalósítása során, amik megválaszolásában sokat segített az ilyen lehetőségeken szerzett tapasztalat.

2.3 Források

A projekt megvalósítása alatt sok helyről gyűjtöttem információkat, ötleteket és megoldásokat. Mégis úgy érzem, van pár forrás, amit meg szeretnék említeni, mert úgy gondolom nagyszerű segítséget nyújtottak nekem. Ezzel hozzájárultak, hogy könnyebben haladjak és valamilyen szinten konzisztenciát is vitt a megvalósításba, mivel egy oldalon jellemzően hasonló módon vannak elkészítve a problémák megoldásai. Emiatt, ha több dolgot tanultam meg vagy vettem át a programomba egy helyről, akkor ezen részek bonyolultsága vagy a kiemelt rész hossza hasonló a többihez.

2.3.1 Baeldung

Az oldal egy Eugen Paraschiv nevű szoftverfejlesztő mérnök által indított oldal kifejezetten Spring Boot applikációknak szentelve. [5] Itt fellelhetünk a teljesen kezdő szintű példákat, a legelső lépéseit is megtanulhatjuk egy Spring applikáció készítésének. Viszont, ezen kívül rengeteg haladó szintű probléma megoldása is meg van valósítva, mondhatni ki van merítve a téma, itt mindent meg lehet találni. Ezen felül, ami tetszett ezen az oldalon, hogy a témák nagy részletességgel szét vannak bontva és minden példához részletes magyarázat tartozik. Ez amiatt is jó, mert ha keresünk valamit, általában kizárólag a szorosan ahhoz a témához kapcsolódó dolgokat kaphatjuk meg az oldaltól, röviden és egyszerűen. Ebből kifolyólag egyszerű példákat használva mutatja be

az adott témát, jól el tudja magyarázni a megoldás mellett az adott kódrészletek mögött rejlő dolgokat, elősegíti az eddig ismeretlen lehetőségek megértését. Így könnyen kis lépésekben haladva meggyorsítja a megértését és megtanulását a Spring egyes lehetőségeinek.

2.3.2 Android dokumentáció és Mitch

Androiddal foglalkozni a segédanyagok szempontjából rendkívül jó. Bármilyen kérdést is kerestem, sok esetben nagyon jó oktató jellegű dokumentációt biztosított hozzá a Google. [6] Az általános szürke dokumentációk mellett, ami a metódusokat és tulajdonságokat listázza, további magyarázatokat lehet találni az oldalon. Szinte minden esetben egy élethű példán mutatják be az adott technológiát. Ábrákkal és olyan minta kódokkal vagy projekttel magyaráznak, ami általában azt a funkciót valósítja meg amire a megoldást keresed. Emellett még van, amihez videó is tartozik, ahol a Google fejlesztői magyaráznak az adott technológiáról, valamint minden kódrészlet egyaránt Java és Kotlin nyelven is megtekinthető.

Ezen kívül egy másik nagyon jó forrásom a CodingWithMitch nevű oldal anyagai voltak. [7] Gyorsan rájöttem, hogy Android témából nagyon sok videó van, amiben nehezen lehet érteni beszédet vagy egyáltalán nem is beszélnek, vagy csak a videó minősége rossz, de nagyon sok az elavult könyvtárakat használó videó is. Viszont itt ez a csatorna, amit egyből lementettem, miután megtaláltam. Aki csinálja ezt a csatornát nagyon jó minőségben rögzíti a hangját és a képet is, valamint nagyon tisztán és érthetően beszél angolul is. Tengernyi videót készített Android témából, szinte mindent meg lehet találni és sokszor újra feltölt egy témát, újabb technológiával, tehát az elavultságtól sem kellett itt félnem.

2.4 Technológiák és eszközök

Ebben a fejezetben a projekt készítése során használt fontosabb technológiákat és eszközöket mutatom be röviden. Mindegyikről írok valamilyen információt, hogy mire használják vagy valamilyen érdekességet róla. Néhánynál a használatról is írok valamilyen alap lépést.

2.4.1 Kotlin

Habár a Kotlin már Android fejlesztésben egy száz százaléig támogatott és preferált nyelv, még mindig nagyon sokan nem ismerik. Amióta megismertem az Android

platformot és elkezdtem a fejlesztésével foglalkozni, egyből nagyon tetszett, nagyon könnyedén lehet kódot írni ilyen nyelven.

Ez egy nyílt forráskódú nyelv, amit a JetBrains adott ki körülbelül tíz évvel korábban. Ez tökéletes Android fejlesztésre, a modern nyelvi elemei lehetővé teszik, hogy sokkal könnyebben fejezzünk ki dolgokat és nagyban csökkentik a felesleges kódsorok hosszát, egyszerű és egyértelmű szerkezete miatt a nyelv ismerete nélkül is jól olvasható a kód. Ezen kívül teljes mértékben együtt tud működni a Java nyelvvel. Egy projektben készülhetnek részek egyik és másik nyelven is egymás mellett, használhatunk vele Javas könyvtárakat. Fordításkor ebből is JVM bájtkód lesz.

2.4.2 dbdiagram.io

Ez egy egyszerű és gyorsan használható eszköz, amivel adatbázis kapcsolatokat ábrázoló diagrammokat lehet készíteni. Ingyenes használhatjuk és amellett, hogy vizualizálva is látjuk az adatszerkezetünket, párhuzamosan mellette nézhetjük és szerkeszthetjük kód formában. Egy minta projekttel indul az oldal, ami az oldal minden funkciójára tartalmaz egy rövid példát. Roppant egyszerű és egyértelmű a használata, mégis minden tervet el lehet vele készíteni. [8]

2.4.3 Postman

A Postman egy kifejezetten hasznos és népszerű alkalmazás. API-k létrehozására, tervezésére és tesztelésére használják. Ezt a programot amellett, hogy ezzel terveztem a *szolgáltatásom* végpontjait, végig teszteléshez is használtam a fejlesztés folyamata alatt. Rendkívül egyszerűen lehet benne tervezni és jól átlátható eredményeket kapunk belőle. Teszteléshez nagyon sok lehetőséget nyújt. Elmenthetünk benne gyakran ismételt kéréseket. A kérés minden paraméterét állíthatjuk, még hozzá számtalan módon. Szinte mindegyiknél kiválaszthatjuk milyen módon akarjuk megadni a kérés részeit és minden megadott mezőt egy kattintással félrerakhatunk későbbre. [9]

2.4.4 MockFlow

Ez egy online alkalmazás, amivel alkalmazás dizájnt tervezhetünk. Erősen támogatja az Androidra történő tervezést. Azért gondoltam úgy, hogy megemlítem, mert már próbáltam más lehetőségeket korábbi projektem készítése során, de egyik sem vált be igazán. Ezzel szembe ez az oldal nagyon jónak bizonyult. Sokkal gyorsabban tudtam elhelyezni a tervem elemeit és egyszerűen megtaláltam a funkciókat, amikre szükségem

volt, adta magát a használat. Kifejezetten sok olyan objektumot tartalmaz a minta elemei között, ami Android fejlesztéshez hasznos és mivel a képernyők oldalakra szervezhetők az appon belüli navigálás is átláthatóan tesztelhető. Mindezek mellett támogatja a közös munkát úgy, hogy valós időben tud több tervező dolgozni közösen egy adott projekten. [10]

2.4.5 MySQL Database és Workbench

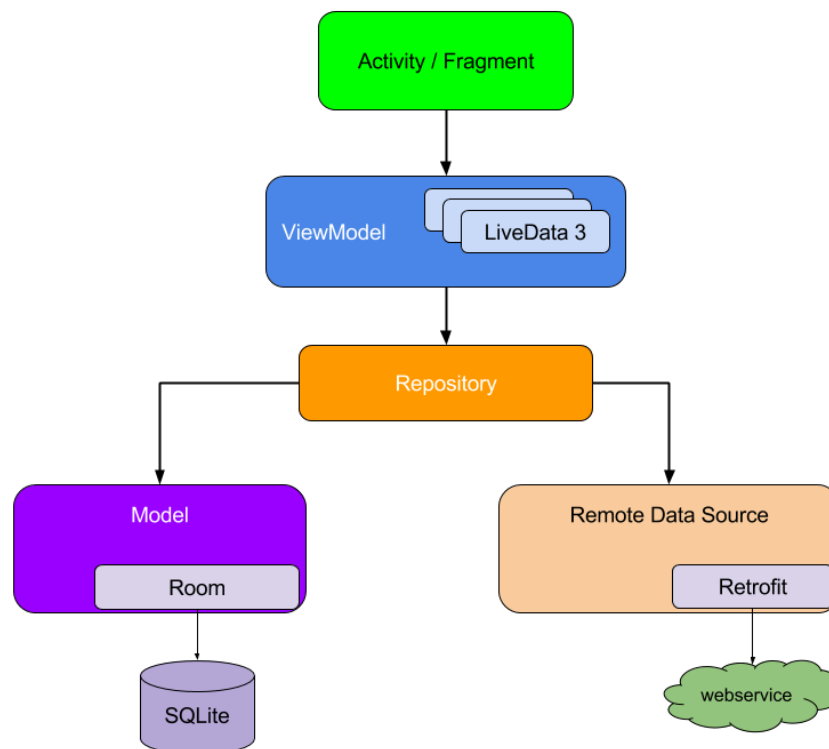
Adatbázisnak a MySQL adatbázist választottam. Korábban dolgoztam más eszközökkel is, előző félévben az Oracle adatbázisával, de a MySQL használata teljesen másnak bizonyult. A Workbench ennek a MySQL-nek az adatbázis kezelő eszköze. Ez lehetővé teszi, hogy az adatbázisunkról egy könnyen átlátható képet kapjunk és szinte mindent be tudunk rajta állítani a program által kínált menükben, amikben egyértelműen el lehet igazodni. Azért említettem ezt meg, mert személyes tapasztalataim alapján ez jobban használható eszköz ilyen méretű és összetettségű projektekhez, mint versenytársai. A végrehajtott műveleteknek gyorsan érkezik eredménye, két kattintással állíthatunk be különféle indexeket, kevés erőforrást igényel a működtetése és nagyon jól kialakított a felhasználói felülete. A gyakrabban használt és fontos funkciók könnyen elérhető helyen vannak és egyszerre több szemszögből láthatjuk az adatbázisunk állapotát. [11]

2.4.6 LiveData és MVVM

Az Android alkalmazásokhoz ajánlott architektúra sablon a *Model, View, View-Model*hez hasonlít a legjobban, kicsit ki van egészítve. [6] Az MVVM azért nagyon jó, mert a rétegek csak közvetlenül az alattuk lévő rétegtől függenek. A rendszer egyes részei kényelmesen fejleszthetők külön-külön, mert így nagyban csökken a csatolás közöttük. Az architektúra ezt úgy teszi lehetővé, hogy az alsóbb rétegek megfigyelhető adatokban tárolják a továbbítani kívánt információt, a felette lévő réteg pedig feliratkozik ezen adatok változására és az alapján tudja végrehajtani a saját feladatait.

Android fejlesztés esetén az erre a célra használt megfigyelhető adattípus a **LiveData**. Ez a típus szinte teljesen egyezik a hasonló megfigyelhető típusokkal, mint például Javában az **Observable**. Viszont külön Androidhoz készítve, rendelkezik egy további tulajdonsággal, tudatában van a hozzá kapcsolódó nézetnek az életciklusával és a vele történő változásokkal. Nagyban megkönnyíti a használatot, ugyanis kevésbé kell figyelni arra, hogy van-e valamilyen szivárgás a programban, vagy hogy ne szálljon el.

Az osztály magától kezeli a szükséges fel- és leiratkozásokat vagy szünetelteti a megfigyeléseket az alapján, hogy a hozzárendelt nézet milyen életciklusban van.



ábra 1: Modulok kapcsolata az ajánlott architektúrában [12]

Az alap elgondolás ki van egészítve még egy adatelérési réteggel, ez az egyetlen réteg több másik osztálytól függ. Lehetővé teszi, hogy egyszerre több adatforrásból nyerhessünk adatot és ezt elrejtí az alkalmazás többi része előtt. A tárgyalt architektúra rendkívül előnyös Androidra az adatkezelés miatt. A nézet állapotának változása alatt (például a képernyő elfordítása) is megőrizhetjük a releváns adatokat, annak ellenére, hogy több életciklusba is bekerül ezalatt a nézet. Ezen felül, az adatelérési réteggel a perzisztens adattárolást kihasználva akkor is egyből küldhetünk adatokat, ha nincs, vagy lassú a hálózati kapcsolatunk. A nézet frissíti magát az új adatokkal, ha helyreáll a kapcsolat és megérkeznek a friss adatok a szolgáltatástól.

Ennek az ajánlott architektúrának használatával nagyban nő az alkalmazást használó felhasználók elégedettsége, ugyanis mindegy, hogy egyből vagy pár perc múlva térnek vissza az alkalmazáshoz azonnal láthatják a legutóbbi állapotot, ami helyileg tárolva van, miközben elindul a háttérben az új adatok lekérése.

3 Tervezés

Miután összegyűjtöttem a technológiákat, amiket használni fogok, elkezdtem szakmai szemmel nézni a projektekre. Megpróbáltam megfogalmazni mire kell képesnek lennie a rendszernek és ezekből meghatároztam a használati eseteket. Elmagyarázok pár megjelenéssel kapcsolatos elgondolást, aminek nagy részét általános ergonómiai megfigyelésekből vettem. [13] Adatbázis sémának alkalmas alakra alakítottam a tulajdonságokat. Végül megterveztem az *Android applikáció* és a *webszolgalattas* alap architektúráját is.

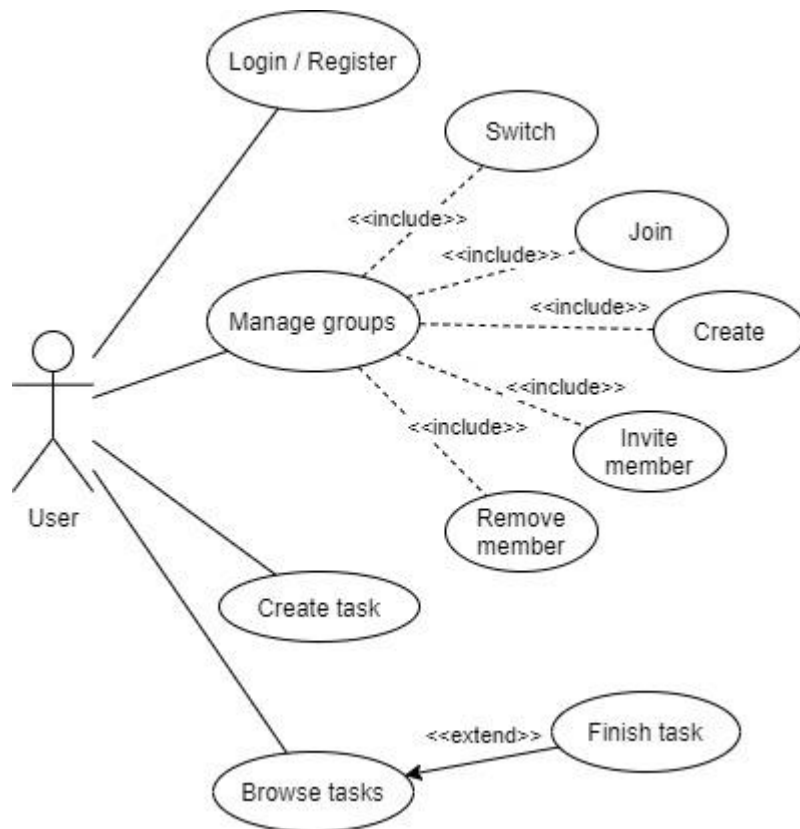
3.1 Követelmények

A munkát azzal kezdtem, hogy meghatároztam milyen funkciókat kell megvalósítania az *alkalmazásnak*. Az *alkalmazás* fő funkciója, hogy segítsen minket a teendőink szervezésében. Bár egyedül is lehetünk csoportban, a másik fontos része az *alkalmazásnak*, hogy együtt tudjuk kezelni a közös csoportban lévő feladatokat. Ezen kívül olyan másodlagos funkciókat próbáltam kitalálni hozzá, amik kiegészítik a működést és praktikussá és kényelmessé teszik az *alkalmazást*.

Ez alapján könnyen össze tudtam állítani a követelményeket. Egy felhasználónak képesnek kell lennie **böngésznie a feladatokat**. Egy feladatról meg kell tudnia **néznie részleteket**, majd tudnia is kell **elvégezni azt**. Továbbá tudnia kell **létrehozni feladatot**, miközben **hasznos tulajdonságokat** állít be rajtuk. Címet adhat neki és meghatározhatja a fontossági szintet, ami alapján majd böngészik köztük. Ezen kívül leírást és ellenőrző listát is beállíthat, továbbá a mobil lehetőségeit kihasználva praktikus volt földrajzi pozíciót is hozzáadni, amit térképen kezelhet és végül más felhasználókat is hozzárendelhet, akikkel egy csoportban van.

A felhasználóknak tudniuk kell **regisztrálni és bejelentkezni**, hogy az *alkalmazás* képes legyen megkülönböztetni őket és a felhasználó csak a neki releváns adatokat lássa.

Mivel több csoportba csatlakozhatunk, a felhasználónak képesnek kell lennie **létrehozni ilyen csoportot** és amikor szeretne tudnia kell **váltania közöttük**. Továbbá meg tudja **hívni másokat** és tudja **csatlakozni mások csoportjába**, képes legyen **eltávolítani a nem kívánt személyeket**.



ábra 2: Az alkalmazás használati esetei

Néhány nem funkcionális követelményt is meghatároztam. Az *alkalmazásnak* jó teljesítménnyel kell működnie, egyből adatokat kell mutatnia és a frissítéseknek gyorsan meg kell érkezniük, mert nem feltétlenül engedheti meg a felhasználó, hogy a töltésekre vár. Az *alkalmazásnak* egyszerűnek és egyértelműnek kell lennie, nem szabad, hogy bárkinek nehézséget jelentsen a használata, mert széles felhasználói körnek van szánva. A használhatóságot és a felhasználói élményt az elemek kinézete is támogassa, alkalmazzon ergonómiai módszereket és szabályokat.

3.2 Dizájn

Miután a funkciókat meghatároztam, a dizájn elkészítésével folytattam [10]. A kinézet tervezése a színválasztással kezdődött. Olyan színt kellett választanom háttérszínnek, ami jól társul, ha a teendőinket kell intéznünk vagy dolgozunk. A kékre esett a választásom, mert a kék megnyugtató érzetet kelt az emberben és még a produktivitást is növeli. [13] [14] A kék mellé a fehéret választottam a többi elem és a szövegek színének. A fehér színnel könnyen ki lehet emelni dolgokat és letisztultság érzetét kelti, ami miatt jobb minőségűnek érezzük az *alkalmazást*. A világos színeket

emellett rövid szövegek megjelenítésére ajánlják, ezért tökéletes választás, mert az *alkalmazásban* jellemzően nincsenek hosszú leírások.

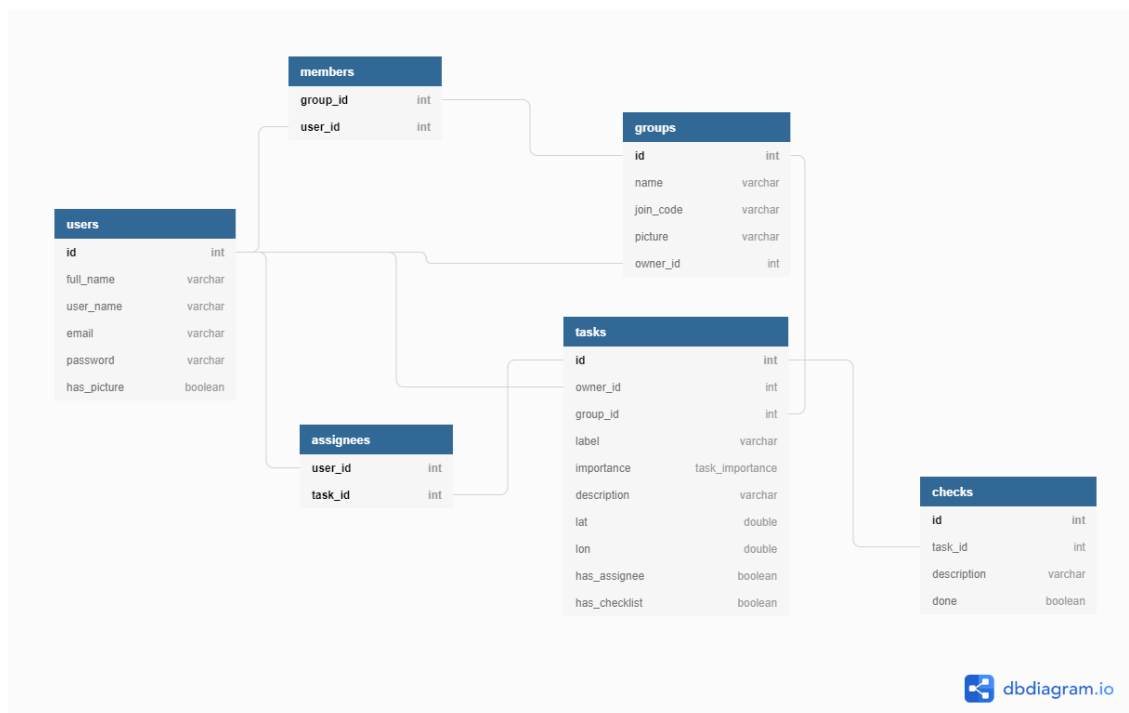
Ezen kívül a fontosságához is színeket rendeltem. Ez egy olyan tulajdonsága a feladatoknak, amit fontos tudnunk a kiválasztásakor és könnyen kódolhatunk színekben. Egy több helyen is együtt látott kombináció a piros, sárga, zöld mindenki által ismert. A pirosról veszélyre vagy figyelmeztetésre gondolunk, ez a sürgős feladat. A sárgának a piroshoz hasonlóan figyelemfelkeltő hatása van, de nem érzünk tőle veszélyt, ez a fontos feladat. A zöld a természet és a természetesség színe, ez az átlagos feladat színe.

A felhasználói felület lekerekített szélű elemeket használ, ez modernebb elgondolás, mint a négyzet alapú alakzatok. Az egymás mellett lévő lekerekített dolgok jobban elkülönülnek egymástól, ami jól jön a feladatok böngészése közben is. Ezen kívül a lekerekített szélű dolgokat szívesebben nézünk, mivel körülöttünk lévő tárgyak is nagyrészt le vannak kerekítve, túlnyomó többségben az éles és hegyes tárgyakkal szemben. Ezt az is fokozza, hogy egész gyerekkorunkban nem lehet nálunk éles vagy hegyes dolog, mert veszélyes ránk nézve és inkább labdákkal és hasonlókkal játszunk. Lekerekített dolgok mellett biztonságban érezzük magunkat.

A navigáció megtervezéséhez ki kellett találnom, hogy mik legyenek az egyből elérhető nézetek, amik az *alkalmazás* gyors és praktikus használhatósága miatt fontosak. Végül négy fő nézetet jelöltem ki, a feladat készítését és a böngészésüket, valamint a csoportok váltását és a csoport kezelését. Azt találtam, hogy négy fő vezérlési elemhez a mobilra ajánlott navigációs technika az alsó navigációs sáv, így ezzel is terveztem. Ez jól jön azért is, mert vertikálisan is két részre osztja az *alkalmazás* nézeteit. Így ki tudom használni azt, hogy ha egy oldalon a felső rész színe világos és az alsó sötét, az a biztonság és természetesség érzését kelti.

3.3 Adatmodell

Az adatbázis séma tervezését is próbáltam egyszerűen venni, hogy véletlenül se kerüljön az *adatbázisba* redundáns adat. Először három fő táblát határoztam meg. A felhasználók (**users**), a csoportok (**groups**) és a feladatok (**tasks**) adatstruktúráját. Amit lehetett hozzájuk rendeltem egyszerű tulajdonságként, ezen kívül három új segéd tábla létrehozására is szükség volt. A **checks** tábla a feladatokhoz tartozó ellenőrző lista elemeit tárolja. Az **assingees** tábla a feladatokhoz rendelt emberekét, a **members** tábla a csoportokhoz csatlakozott emberek azonosítóját tartalmazza.



ábra 3: Az adatmodell a kapcsolatokkal

A felhasználók tábla tulajdonságai közül a felhasználó névnek és az email címnek is egyedinek kell lennie, mindkettő kulcsként szolgálhat az alap azonosítón kívül. Ezen felül a jelszó mező a felhasználó jelszavát tartalmazza hashelve, hogy nehéz legyen visszafejteni. A csoportoknak lehet egy csatlakozó kódja, ami egyedi a csoport nevéhez hasonlóan. A felhasználókról és a csoportokról az is tárolva van, hogy van-e hozzá feltöltött kép, ez igaz vagy hamis értéket reprezentál számként.

A feladatokhoz tartozik egy tulajdonos és egy tulajdonos csoport azonosító, ami alapján rendeljük majd őket lekérdezésekhez. A fontosság mezője szöveggént tárolja az fontossági szintek enumerációjának értékeit, így praktikusabb tárolni sorszámozás helyett. A földrajzi pozíciót hosszúsági és szélességi koordinátával tudom pontosan meghatározni, ezeket egy-egy külön tulajdonságokként tárolom. Az ellenőrző listához és a beosztottakhoz csak egy logikai mező tartozik, ami arról ad információt, hogy kell-e keresni majd értékeket a segéd táblákban.

A beosztottak és a tagok tábla csupán kapcsolótáblák. Két tulajdonságuk a felhasználó azonosítója és vagy a csoport, vagy a feladat azonosítója. Mindkét tábla esetén ez a két tulajdonság alkot összetett kulcsot. Az ellenőrző elemek csak annyit tárol, hogy melyik feladathoz tartozik, egy rövid leírást és az állapotát, hogy megcsinálták-e már vagy sem. Azért csináltam ezeknek a jellemzőknek segéd táblát, mert mindegyikből

több tartozhat egy feladathoz vagy csoporthoz, így egy sima mezőben nem írható le a kapcsolatuk.

3.4 Android architektúra

Ahogy korábban említettem, MVVM architektúra mintájára próbáltam szervezni az forráskódot. Ezek az alapelemek ki vannak még egészítve egy adatelérési réteggel (**repository**). Rendkívül jól osztja el a felelősségeket ez a fajta megközelítés. Minden megjelenítéssel dolog és felhasználói interakcióval kapcsolatos esemény a *nézet* (**view**) részébe kerül az *alkalmazásnak*, mással nem foglalkozik. Ugyanígy a *modell* (**model**) kizárólag az adat tárolásával foglalkozik, nem kerül ide sem semmilyen üzleti logika. Ezt a kettő rétegek köti össze a *nézetmodell* (**view-model**), ami tartalmazza az üzleti logikát és gondoskodik arról, hogy a *nézet*hez megfelelő formátumban jusson az adat.

Ezen alap rétegek köze épül be az adatelérési réteg, a *nézetmodell* és a modell között foglalja el a helyét. Ennek az elemnek az a lényege, hogy egy tiszta interfészt biztosít a *nézetmodell*nek, aki ezen keresztül tudja hívni az adateléréseket. Viszont ez az adatot több helyről is nyerheti, de ezt nem mutatja a *nézetmodell* felé. Esetünkben az egyik forrás egy Android perzisztencia könyvtár, a *Room*. Ez lokálisan egy SQL alapú adatbázisban tárolja az adatokat, ezekre az adatokra támaszkodunk hálózati kapcsolat nélkül. A másik forrás távoli adatszolgáltatás, a saját *Spring szolgáltatásom*. Az ide intézett kéréseket *Retorfit* [15] könyvtár segítségével hajtja végre az *alkalmazás*.

A felsőbb rétegek nem ismerik az lejjebb lévőket. Mindegyik megfigyelhető kollekción vagy adatot tartalmaz, aminek a változásáról értesül az alatt lévő réteg és reagál rá a saját feladata szerint. Ezért az *alkalmazás* minden egységének az adatait egy ilyen megfigyelhető állapotban tárolom, ami alapján a *nézet* a teljes egészét meg tudja határozni.

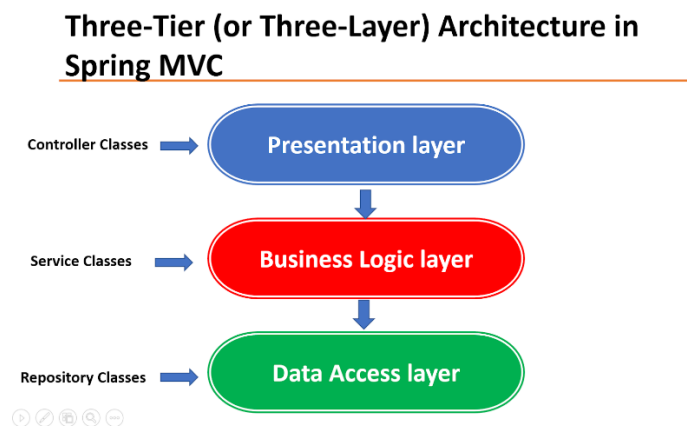
A forrásfájlok szervezése ennek az architektúrának megfelelően minden modul saját mappában foglal helyet. Ezek a nagyobb modulok még beljebb csoportosítva vannak aszerint, hogy a bejelentkezéshez vagy a fő részhez tartoznak, továbbá, hogy feladathoz vagy csoportokhoz kapcsolódó funkcióhoz tartoznak az *alkalmazásban*.

A *nézet* mappa *fragment*eket és *Activity*ket tartalmaz. A *nézetmodell* mappa az *adattárral* köti össze a *nézetet*, kezeli az eseményeket és megfigyelhető adatot biztosít a *nézet*nek. Az *adattár* perzisztensen tárolt adat használata esetén a *room* mappa,

webszolgáltatástól kért adat esetén pedig a retrofit mappa interfészeit és egységeit használja. A *model* mappában a szolgáltatott adatok struktúrája van leírva, valamint néhány segéd osztály és interfész. A *factory* mappa a gyár tervezési mintához szükséges *nézetmodell* gyárat tartalmazza. Végül az *adapter* mappában az Androidos listák vezérlőit találhatjuk.

3.5 Webszolgáltatás architektúra

A Spring applikációt három rétegű architektúrával próbáltam megvalósítani. A legfelső réteg a vezérlők (**controller**), amiben meghatározom az API végpontjait és a kérések paramétereit és válaszát, majd továbbítja a kérést a szolgáltatásnak. A szolgáltatás (**service**) a középső réteg, ez tartalmazza az üzleti logikát. Itt kerülnek összeállításra és megfelelő formátumba a válaszok és néhány esetben, amikor nem sikerült az *adattárral* megoldani hívásokat intéz az *adatbázis* felé. A legalsó réteg ez az *adattár* (**repository**), ami az *adatbázisban* tárolt adatok elérésével és manipulációjával foglalkozik és metódusokat biztosít a szolgáltatás felé, aki ezen a rétegen keresztül jut adathoz.



ábra 4: A webszolgáltatás architektúrájának mintája [16]

A forráskódban ezek a modulok még külön vannak tagolva a szerint, hogy a csoportokkal, a felhasználókkal vagy a feladatokkal kapcsolatos kéréseket szolgálnak ki. Ezeken kívül későbbi műveletek megkövetelték, hogy *adattárból* létrehozzak egy ellenőrző elemek táblához kapcsolódót is, amit a feladatok szolgáltatásából használok. Ezeket a csoportokat Postmanben [9] alakítottam ki miközben terveztem a végpontjait a *szolgáltatásnak*.

Ezek a részek még kiegészülnek néhány további egységgel. Az *entitások* (**entities**) mappa az *adatbázis* sémájának a mintájára határozza meg az osztályokat és tulajdonságaikat, annotációkkal az *adatbázis* mezőinek megfeleltethetők a tulajdonságaik vagy kezelhetjük velük például enumerációk tárolását. A **dto** mappa, az adat átviteli objektum rövidítése. Csupán annyi a feladata, hogy az *entitások* tulajdonságait és esetleg további számolt értékeket egy objektumba csomagoljon vagy elrejtse. Ilyen objektumokat adnak vissza azok a végpontok, amik válasz adatai nem tükrözik az *entitásokat*. Itt gondolkodtam azon, hogy bevezessek egy mapper osztályt az *entitások* és *dto*-k közötti átalakításhoz, de úgy mivel nincsen sok ilyen objektumom úgy döntöttem nem teszem meg és csupán 1-1 konstruktort írtam a típuskonverzióhoz. Ezen kívül a projekt még tartalmaz egy egyéb (**misc**) mappát, ahol olyan típusokat tárolok, amik a *szolgáltatás* működéséhez szükségesek, de nem jelennek meg az *adatbázisban*, így az *entitások* mappán kívülre akartam tenni.

4 A megvalósítás részletes bemutatása

Ebben a fejezetben már az implementálás folyamatával fogok foglalkozni. Megpróbálom minél jobban bemutatni a feladat sajátosságát és az elvégzett munkámat. Ezt úgy fogom csinálni, hogy megpróbálom megragadni a fejlesztésnek a nehezebb részeit, amikre sok időt szántam, vagy csak teljesen ismeretlen volt az alkalmazása. Ezen kívül a nem egyértelmű megoldásokról is elmagyarázom miért úgy valósítottam meg, és néhány helyen általános gondolatokat is elmondok, amit fontosnak tartottam az adott rész készítése alatt.

Ezeket a részeket olyan sorban mutatom be, ahogy a rendszeregyes funkcióit egymás után fejlesztettem, ezen belül a külön veszem az egyes akadályokat és gondolatokat, majd bemutatom a rendszer egyes részein a hozzátartozó fejlesztéseket.

4.1 Bejelentkezés és regisztráció

A bejelentkezéshez és a regisztrációhoz két külön nézet tartozik az *alkalmazásban* **LoginFragment** és a **RegisterFragment**. Ezek *fregmensek* a bejelentkezésre dedikált **LoginActivity**hez kapcsolódnak. Mindkét nézethez tartozik **ViewModel** és **Repository** is, de a bejelentkezéssel kapcsolatos fájlok még ezeken belül is egy mappában vannak, valamint az ide tartozó függőségek a **LoginContainer**ben vannak. Az ehhez kapcsolódó API végpontok a **LoginService**ben találhatók.

Az *adatbázisban* csak a **Users** táblát érintem ezalatt a rész alatt, részletesebben írok a jelszó és a kép tulajdonságáról. A *szolgáltatásban* **Controller**, **Service** és **Repository** is tartozik hozzá, valamint itt jelenik meg az **ApiToken** és a regisztrációhoz tartozó **RegisterStatus** is.

4.1.1 Gyár tervezési minta

Legelőször is a belépést kezdtem el megvalósítani, hiszen ennél a résznél kezdjük el használni az *alkalmazást*. Valamint gondoltam jó lenne olyan résszel kezdeni, amiben kevés az üzleti logika, inkább mindig megjelenő autentikációs dolgok vannak csak benne. Így ki tudom próbálni és bele tudok rázkódni kicsit a létrehozott rendszer architektúrájába és a fejlesztés menetébe.

A factory minta használata:

```
class RegisterViewModelFactory(private val registerRepository:
RegisterRepository) {
    fun create(): RegisterViewModel {
        return RegisterViewModel(registerRepository)
    }
}
```

Már a fejlesztés elején bevezettem a *Factory* minta használatát, mivel így pár funkciót és a *nézetmodellek* működését számomra átláthatóbban tudtam megvalósítani. A *nézetmodelleket* mindig a *Factory* mintával állítom elő. Ezek csak egy **create** metódust valósítanak meg, amikkel független *nézetmodell* példányokat állíthatunk elő.

4.1.2 Bejelentkezést vezérlő adat

Hogy tudjam magam tartani az MVVM architektúrához ki kellett találnom, hogy milyen adatok vezéreljék a működést, amire fel tud iratkozni a *nézet*. Ez úgy, hogy elsőre foglalkoztam ilyen architektúrával nem volt egyértelmű. Végül úgy döntöttem, hogy létrehozok egy **ApiToken** osztályt, aminek a feladata a felhasználó azonosítójának a tárolása. Ezen kívül tartalmaz egy **ApiKey** tulajdonságot, ez szolgálta volna az autorizációt az *alkalmazás* egyes részein, de sajnos az idő szűkében nem tudtam vele foglalkozni, későbbi fejlesztés lehetősége miatt azonban bent hagytam a projektben.

A *ViewModel*nek egy ilyen típusú tulajdonságára iratkozik fel a *nézet*, ami bejelentkezési kísérlet válaszára frissül. A *szolgáltatás* vagy a felhasználó azonosítóját és kulcsát adja vissza, majd belép ezeket az információkat továbbítva a fő *Activity*hez, vagy nullát azonosítóként és a kulcs részében egy hibaüzenetet, amit megjelenítek a felhasználónak.

4.1.3 Bejelentkezett felhasználó mentése

Úgy gondolom, hogy a bejelentkezett felhasználó mentése nagyon kényelmessé teszi az alkalmazások használatát, ezért mindenképpen meg akartam valósítani. Végül úgy döntöttem, hogy a *SharedPreferences*be fogom menteni, ahol primitív adat és kulcs párokat tudunk perzisztensen tárolni. Itt nem tárolok jelszót csupán a korábbi bejelentkezésből adott **ApiKey**-t.

Miután ezek megvoltak, már csak beállítottam a **repository**ban, hogy először próbálja meg a *SharedPreferences*ben mentett értéket betölteni, ami alapján a *nézet* frissíti is magát és tovább léphet. Amennyiben nem volt korábbi mentett bejelentkezés,

egyszerűen nullás azonosítóval és hibaüzenet nélkül létrehozásra kerül a megfigyelhető **ApiToken**, amire nem reagál semmit a *nézet*.

A nézet feliratkozása az ApiToken tulajdonságra:

```
loginViewModel.token.observe(viewLifecycleOwner){
    if(it.id != 0){
        val intent = Intent(activity, MainActivity::class.java).apply {
            putExtra("api-token", it)
        }
        startActivity(intent)
    }else {
        if(it.token.isNotEmpty()) Toast.makeText(activity, it.token,
        Toast.LENGTH_LONG).show()
    }
}
```

A *SharedPreferences*be az értékek sikeres bejelentkezéskor kerülnek mentésre és kijelentkezéskor pedig törlésre kerülnek.

4.1.4 Jelszó tárolása

Mivel jelszóval kell bejelentkezni és regisztrálni nyilván ezt is tárolnom kell az *adatbázisba*. Ezt egy egyszerű szöveges mezőként vettem fel az *adatbázisba* a felhasználókhöz. Hogy az adatok esetleges kiszivárgása esetén ne lehessen visszaélni a jelszavakkal, egy hashelő algoritmussal [17] kódolom, hogy nehéz legyen megkaparintani a jelszavakat. Ezt a kódolást már az *Android alkalmazásban* megteszem a *ViewModel* rétegben, hogy már a hálózaton is biztonságban utazzanak az adatok.

248B646537648C1FBDEB42B56771DBDB42129E8BAB527FF551A1F49CE499464F

ábra 5: A "jelszo" szó SHA256 hashe

4.1.5 Regisztrálás képpel

A regisztrálásnál egy **ApiTokenhez** hasonló osztállyal oldottam meg, a *nézet* vezérlését. Annyiban különbözik tőle, hogy azonosító helyett egy logikai értékben tárolja a regisztráció sikerességét, valamint a szöveg típusú mezőjében mindig a hibaüzenetek jeleníti meg. Ez a **RegistrationStatus**.

Meg akartam valósítani azt is, hogy a felhasználók képet tölthessenek fel magukról. Ehhez *Android alkalmazás* részről több dolgot is meg kellett tennem. Először Androidon minden bizalmas tartalom és veszélyes akció hozzáféréséhez engedélyt kell kapni a felhasználótól, különben letiltva marad az adott funkció. Ilyen veszélyes beviteli

eszköz a kamera is, amihez futásidőben is el kell kérnünk az engedélyt. A fényképet elkészítése után egy *képnézet* forrásaként beállítom és egy logikai változóval számontartom, hogy saját kép-e a forrás. Ezután adom majd tovább **Bitmap**ként, amit *ViewModel*ben bájtömbbé alakítok és így küldöm el a regisztráció kérésnek a testében, a kérés **hasPicture** paraméterével együtt. Ha nem adunk meg képet üres testtel megy a kérés és a *szolgáltatás* a hasPicture paramétert használva tudja eldönteni, hogy neki álljon-e a kérés testének feldolgozásához.

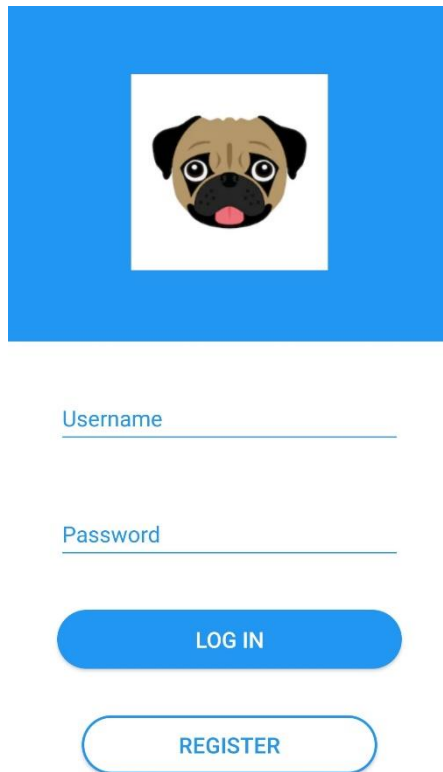
Részlet az engedélykérés folyamatából:

```
private fun askCameraPermission() {
    if(ContextCompat.checkSelfPermission(requireActivity(),baseContext,
        android.Manifest.permission.CAMERA) !=
        PackageManager.PERMISSION_GRANTED){
        ActivityCompat.requestPermissions(requireActivity(),
            arrayOf<String>(android.Manifest.permission.CAMERA),
            101)
    }
    else{
        openCamera()
    }
}
```

Korábban a képek tárolását úgy valósítottam meg, hogy az adatbázisban az adott kép elérési útját tároltam. Azonban, mivel például a felhasználónév is egyértelműen azonosítja az egyént, most az *adatbázisban* csak azt tárolom logikai értéként, hogy tartozik-e hozzá kép vagy nem, az elérési út vonal pedig a *szolgáltatásban* meghatározva, ami a képeket a felhasználó névvel megegyezően nevezi el.

4.1.6 Dizájn

A bejelentkező képernyő egyből ellent mond annak, amit eddig az *alkalmazás* színeiről és navigációjáról mondtam. Itt navigációs menüről még nyilván nincsen lehetőségünk beszélni, mert még nem azonosítottuk magunkat. Színeket tekintve pedig kék helyett a bejelentkezés nagy része fehér. Sokszor láttam ilyen stílusú bejelentkező felületet és mivel a fehér színről az új és tiszta dolgokra is gondolunk, meghozza a kedvet ahhoz, hogy regisztráljunk és belépünk az *alkalmazásba*.



ábra 6: A bejelentkezés egyéni megjelenése

Itt kezdtem el továbbá néhány dizájn elemnek a megtervezését. Ezen a ponton létre is hoztam a lekerekített gombok erőforrását és egyedi szöveg beviteli mezőt is használok, ami a szövegdoboz fölött jelzi a mező jelentését akkor is, ha van benne már adat. Ezeket úgy hoztam létre, hogy a resources **drawable** mappájába készítettem hozzájuk **erőforrást**, amiben alakzatokkal leírom az adott elem formáját, majd a **stílus** erőforrások között készítettem sajátot és beállítottam ezek hátterének a drawablet. Ezen kívül itt a stílusok között állítom be a nézet elemek többi alap tulajdonságát, mint például a betűszín vagy betűméret.

Egyedi stílus az elemekhez:

```
<style name="LightBlueOutlinedButton" parent="Widget.AppCompat.Button">
    <item
name="android:background">@drawable/button_outlined_blue_rounded</item>
    <item name="android:textSize">18sp</item>
    <item name="android:textColor">@color/white</item>
</style>
```

4.2 Főnézet

A főnézet felülete nem tartalmaz egyebet, csupán az alsó navigációs sávot és egy keretet, ahova az egyes fő funkciók nézetei lesznek betöltve. Feladata ezen nézetek megjelenítése és cserélgetésünk irányítása. Ez egy **Activity** és a betöltött nézetek **Fragmentek** lesznek, amiknek ez az *Activity* lesz a tulajdonosuk, ezért a korábban felosztott konténernek itt kérem el a **MainContainer** részét, ami tartalmazza a **ViewModelFactoryket** a funkcióknak. Ezt egy belső láthatóságú tulajdonságon keresztül éri el a többi nézet és kéri el tőle a *nézetmodelljeiket*.

Navigációs függvény a MainActivity-ban:

```
private fun toCreateFragment(): Boolean {
    supportFragmentManager.beginTransaction()
        .replace(R.id.container, CreateTaskFragment())
        .commitNow()
    return true
}
```

Mivel minden feladat csoportokhoz tartozik, először a csoport választásával kell találkozjon a felhasználó. Ezen a ponton, mivel a felhasználónak mindenképpen csoportot kell választania, meg kellett oldanom, hogy máshova ne tudjon navigálni. Ezt szimplán úgy csináltam meg, hogy az alsó navigációs sáv láthatóságát eltüntre cseréltem, így nem foglal helyet a felületen és a vezérlői is le vannak tiltva. Ezután a csoportválasztás után megjelenítem a sávot és az *alkalmazás* teljes egészében használhatóvá válik.

4.3 Csoportok választása

Az *alkalmazásba* való belépés után következő lépésként a csoport választását akartam megvalósítani. Ezekhez a csoportokhoz csatlakozhatunk és hozzájuk tartoznak majd a feladatok is, emiatt először a csoportokat akartam megcsinálni működőképesre.



ábra 7: Csoport választása nézet

Létrehoztam az *adatbázisban* a csoportok tábláját **todo_groups** néven, mert a *sima groups* kulcsszó volt az *adatbázisban*. A csoportokhoz is saját **Controller**, **Service** és **Repository** tartozik a *szolgáltatásban*. Ugyanígy az *Android* architektúrában is minden rétegében megjelenik a saját fájlja **ChooseGroup** előtaggal és külön **Servicebe** vannak az ide tartozó végpontok. Itt jelenik meg továbbá az *Androidos Room* adatbázis a csoportokhoz tartozó **GroupDao**val.

4.3.1 MVVM

Itt nagyon jól tudtam érvényesíteni az architektúra előnyeit, pont ilyen funkciókra tökéletes ez. Amint a *fregmensre* érkezünk az létrehozza a *ViewModeljét*, amit a tulajdonos *Activityből* ér el és feliratkozik a **groups** tulajdonságára, ami egy megfigyelhető, **Group** típusú lista. Eközben a *ViewModel* egyből értéket is kér a *Repositorytól* a tulajdonságához. Ezt a *Room* adatbázisban mentett csoportok listájából meg is kapjuk, így a képernyőn egyből láthatjuk a rendszer legutóbbi állapotát. Ezzel párhuzamosan a *Repository* indít külön szálon egy hálózati kérést a csoportok frissítésére és amikor ez megérkezik menti az adatbázisba az új adatokat. Ennek hatására a változás

végigfut a rendszeren a megfigyelhető tulajdonságoknak köszönhetően és a *nézet* frissíti a listáját az új adatokkal.

4.3.2 RecyclerView

A nézet nem tartalmaz egyebet egy listán kívül, amit Androidos ajánlások alapján *RecyclerView* segítségével valósítottam meg. Ehhez létre kell hoznom a **ChoosGroupAdaptert** ami arra szolgál, hogy kezelje a *RecyclerView* működését és tárolja az adatokat.

Az csoport kiválasztás elsütése a hallgatón:

```
init{
    view.setOnClickListener{
        group?.let { it1 -> onItemClick?.onGroupSelected(it1) }
    }
}
```

A csoportokba egy ilyen *RecyclerView* elem kiválasztásával tudunk belépni. Ehhez valahogy meg kellett oldanom, hogy tudjon reagálni az *fregmens* az elem kattintására. Ezt úgy csináltam meg, hogy az *adapterben* publikáltam egy **GroupItemClickListener** interfész egy **onGroupSelected** függvénnyel és meghatároztam egy publikus **itemClickListener** tulajdonságot, aminek meg kell valósítani ezt az interfészt. Az egy listaelem nézetének a létrehozásakor a hozzá tartozó **View** elemnek beállítok egy **onClickListenert**, ami nem csinál egyebet csupán meghívja a **itemClickListener** tulajdonság **onGroupSelected** függvényét, amennyiben az nem null. Így a *fregmensben* csak meg kellett valósítanom az interfészt, hogy kattintásra tovább lépünk az *alkalmazás* következő részére és megadhattam a *fregmenst* magát az *adapter* **itemClickListenerének**.

4.3.3 Csoport létrehozása

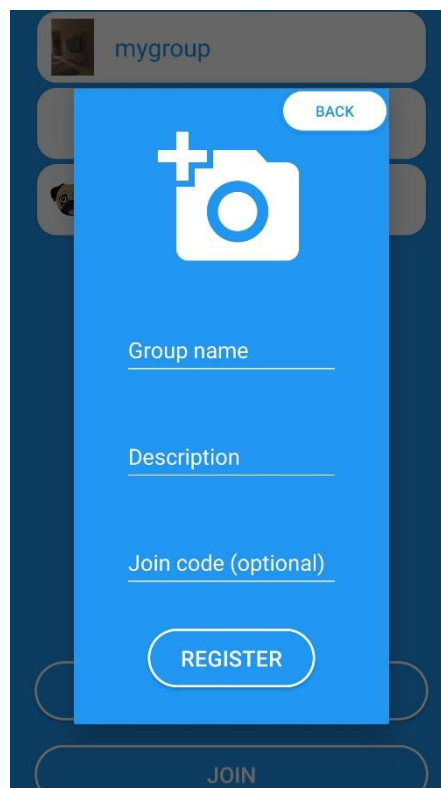
Most, hogy már a felvett teszt adataimmal működött a választás, lehetőséget kellett biztosítanom csoportokhoz való csatlakozáshoz és létrehozásukhoz is. Úgy gondoltam, hogy ezek viszonylag szorosan a választáshoz tartoznak, ezért ez a két funkció a választás alatt két gombbal érhető el. Ráadásul mivel a csoportnak nincsen sok tulajdonsága a létrehozásához csak egy dialógus jelenik meg a csoport választós *fregmens* felett. Úgy akartam megjeleníteni, hogy az oldal közepén jelenjen meg és arányaiban legyen hasonló a képernyőhöz, ezt úgy tudtam megoldani, hogy a dialógus

megjelenésekor kódban kiszámolom a képernyő méreteinek nyolcvan százalékát és ezt beállítom a dialógusnak.

A dialógus méreteinek kiszámolása futásidőben:

```
val width = (resources.displayMetrics.widthPixels * 0.8).toInt()
val height = (resources.displayMetrics.heightPixels * 0.8).toInt()
dialog?.window?.setLayout(width, height)
```

Mivel ez a **CreateGroupFragment** csak a csoport választásoknál jelenik meg és nem egy összetett funkció, úgy döntöttem egy **ViewModel** és **Repository** fog osztózni a csoport választással. Ez a *fregmens* egyébként hasonlóan működik a felhasználó regisztrációjához. A kép készítéséhez ugyanúgy engedélyt kérünk és egy logikai változóban nézem, hogy van-e készítve saját kép. *Szolgáltatás* oldalon is majdnem teljesen megegyezik a regisztrációhoz kapcsolódó kódrészekkel. Továbbá a csoportokhoz tartozik egy tulajdonos azonosító az *adatbázisban*, ezt a felhasználó nem adja meg készítéskor csak küldi magától az *alkalmazás*. Így külön jogokkal ruházhatom fel a tulajdonost, például emberek csoportból való eltávolításával vagy a csoport megszüntetésével.

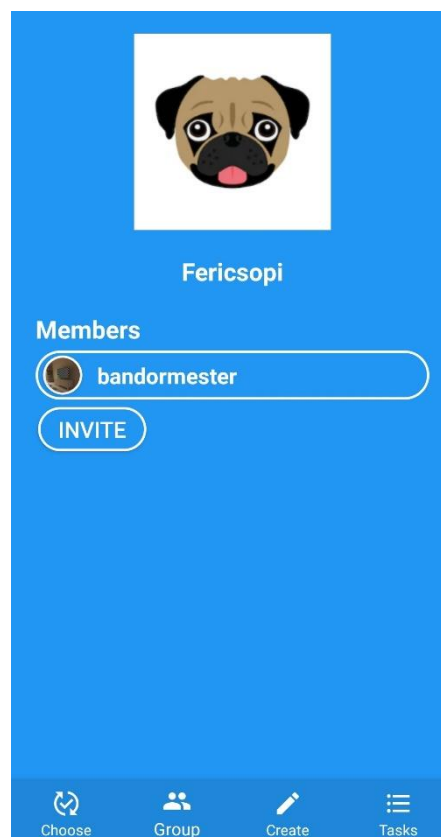


ábra 8: Csoport létrehozása dialógus

A csatlakozást meghívó kóddal valósítottam meg, mert ezt rendkívül egyszerű volt implementálni és elég gyakran használt megoldás. A regisztrációkor opcionálisan megadhatunk egy csatlakozási kódot, és később ezzel a kóddal tudunk majd másokat meghívni a saját csoportunkba. Ha nem adunk meg semmit a *szolgáltatás* majd magától generál egy egyedi kódot.

4.4 Csoport részletes nézete

A csoportoknak tulajdonképpen a feladatok és emberek összefogásán kívül nincsen sok szerepe. Létrehoztam egy részletes nézetet hozzá, amin áttekinthetjük a csoport adatait, néhány tulajdonságot és egy listát láthatunk a tagokról. *Android oldalon* a **ChooseGroup** előtéttel rendelkező fájlokban van a kódja a főbb modulokban, de van még a listához tartozó **GroupMemberAdapter** és a *Room* adatbázis felől is bele kellett nyúlni. A *szolgáltatásban* ugyanott foglalkozok ezzel, mint az előző csoportok választása nézettel. Ezen kívül az *adatbázisban* létrehoztam a **members** táblát, ami a csoportok tagságát hivatott tárolni.



ábra 9: Részletes csoport nézet

4.4.1 Tagok tárolása

A tagok tárolását *adatbázis* szinten egy külön kapcsolótáblába ki kellett szerveznem, hogy könnyen tudjam a több-több kapcsolatot jellemezni. Ez a tábla csupán két azonosítót tartalmaz, amik együtt alkotják az összetett kulcsot. Mivel az egy csoporthoz tartozó tagokat máshol is le kell majd kérni hasonlóan, úgy gondoltam jó lenne külön végpontba szervezni a *szolgáltatásban* is. Létrehoztam egyet a **getGroup** metódust, ami egy csoportot ad vissza az egyszerű tulajdonságaival és egy **getMembers** metódust, ami egy User listát ad vissza, azokkal, akik a csoport tagjai. Utóbbit több próbálkozás után nem sikerült megvalósítani a *Springes JpaRepository* lehetőségeivel, ezért közvetlenül a *szolgáltatásba* készítettem egy SQL lekérdezést.

Az *Android app* rész készítésekor ezt követve több fő megfigyelhető tulajdonságot vettem fel a *ViewModel*-nek, a csoportét és a tagok listáját. A **members** tulajdonság változására frissül a lista tartalma, a **group** tulajdonsággal pedig a *nézet* többi része. Ezen tulajdonságok frissítése majdnem teljesen ugyanolyan, mint a csoport választásnál, pár specifikus tulajdonságnál más. A csoportnál például lista helyett csak egy csoportot kapunk, a tagok lista *adapterében* pedig egy logikai tulajdonságban tárolom, hogy ebben az állapotban lehet-e eltávolítani tagokat a csoportból.

A lekérdezés a változtatott sorok számával tér vissza:

```
int count = em.createQuery("DELETE FROM Member m WHERE m.groupId = :groupId AND m.userId = :userId")
    .setParameter("groupId", groupId)
    .setParameter("userId", userId)
    .executeUpdate();
if(count == 1){
    return ResponseEntity.ok().body("Successfully removed member");
}
```

A tagok eltávolítását a mellettük lévő keresztre kattintva tehetjük meg. Ez meghívja a *ViewModel* megfelelő függvényét, majd tovább hív a *repositoryba*, ami elküldi a kérést a *szolgáltatásnak*. Ehhez is egy szöveg típusú megfigyelhető adat tartozik, hogy részletesebb hiba üzeneteket tudjak küldeni a *szolgáltatásból*. A *szolgáltatásban* itt is szintén egy saját lekérdezést készíték, mert több próbálkozás után nem sikerült kompozit kulcsos tábla kezelését megoldani a *Spring Repositoryjával*. Ennél a lekérdezésnél is és az előzőnél is a Java perzisztencia könyvtárával regisztrálom a lekérdezés paramétereit sima String összefűzés helyett, hogy megelőzzem az SQL injekciós támadásokat.

4.4.2 Csatlakozás csoporthoz

Ahogy korábban említettem, kóddal lehet csatlakozni a csoportokhoz, ehhez ezen a nézeten a meghívó gombot lenyomva a vágólapra másolhatjuk az adott csoport kódját. Erre az Andorid körülbelül egysoros megoldást biztosít.

4.4.3 Tag lista elem dizájnja

Szinte minden elem sarkát lekerítve hoztam létre, azonban itt nehéz volt a tartalmat formázni. Mivel itt nem olyan magas elemeket terveztem a tagoknak és képet is meg akartam jeleníteni, a képet kör alakúra akartam venni és egy keretet is terveztem rá. Magával az egész elem formájával nem volt gondom, újra tudtam használni a korábban a gomboknak megírt erőforrást. Ezt beállítottam az elemek hátterének és erre raktam rá a többi elemet. A képet már nehezebb volt megoldani kerettel. Kipróbáltam pár könyvtárat, ami elvileg megoldaná a problémát, de amiket találtam azok nagyon régen voltak frissítve és már nem működtek. Végül úgy oldottam meg, hogy vettem egy *CardView*-t, aminek van egy **cornerRadius** tulajdonsága, amivel lekerekíthettem. Ezt elhelyeztem a háttéren és fehérre színeztem, ez lett a keret. Ebbe belehelyeztem egy majdnem teljesen azonos *CardView*-t, amit kitöltöttem egy *ImageView*-val. Egyedül a sugara volt kicsit kisebb a második *CardView*nak, így úgy tűnik mintha fehér keretben lenne a kép.



ábra 10: Listaelemek a kereszt láthatóságtól függően

Az *alkalmazásban* nem mindenkinek van joga eltávolítani mást, továbbá máshol is használni akartam ezt az *Adaptert*, ezért meg kellett oldanom a piros kereszt eltüntetését. Ezt elég egyszerűen megoldottam úgy, hogy ha a korábban említett logikai érték hamis volt a keresztet tartalmazó *ImageView* láthatóságát eltűntre állítottam. Szinte minden *ConstraintLayout*ban valósítottam meg, beleértve ezt is. A nézet végét a kereszt elejéhez kötöttem, így a kereszt eltüntetésének hatására, simán kinyúlik a nézet elem többi része.

4.5 Feladat létrehozása

Most, hogy kész lettem a csoportok működésével is, megvolt minden alap, hogy a feladatokat teljes értékűen hozzá tudjam adni az *alkalmazáshoz*. Úgy gondoltam, hogy

először a feladatok létrehozását valósítom meg, mert a böngészésnél úgy is szükség lesz teszt adatokra. A feladat több és összetettebb tulajdonságok tartoznak, itt már nehezen tudtam volna úgy kézzel bevinni a teszt adatokat, mint a csoportoknál.

ábra 11: Feladat létrehozásának a nézetei

Létrehoztam az *adatbázisban* a **tasks** táblát és felvettem hozzá a korábban tervezett tulajdonságokat és két segédtáblát az **assignees** és a **checks** táblát. Ennek megfelelően létrehoztam a *szolgáltatásban* a hozzá tartozó **entitást**, valamint a **Controller**, **Service** és **Repository** modul. Itt is készült **Assignee** és **Check** *entitás* és megjelent az **Importance** enumeráció is. Az *Android app* részen is megjelentek ezek az új osztályok és az enumeráció. Létrehoztam az architektúra főbb részeinek az ehhez tartozó osztályait **CreateTask** előtaggal, valamint a **TaskService**t és egy segéd **CreateMapActivity**t. Ezt a létrehozás *fregmensével* összeraktam egy újabb mappába a *nézeteknél*, mert a feladat létrehozásánál érhető el és csak itt van szerepe. Ezen kívül készítettem egy **ChecklistAdaptert** az ellenőrzőlistának és egy **SelectMemberAdaptert** a tag feladathoz rendeléséhez. Az ezt a nézetet irányító megfigyelhető adatot ugyanúgy egy **RegistrationStatus** típusú tulajdonságként határoztam meg itt is, úgy, mint a regisztrációnál is.

4.5.1 A nézet dizájnja és a feladat tárolása

Úgy gondoltam jó lenne egy olyan létrehozó nézetet készíteni, ami egy oldalon tartalmazza minden felvehető tulajdonságot. Így bármelyik pillanatban egészében láthatjuk a feladatot, úgy együtt minden jellemzőjével, ahogy később is érdekelni fog. Viszont a sok tulajdonság miatt és a helyigényes állításuk miatt, nem akartam csak simán belerakni egy görgethető nézetbe, mert úgy éreztem így lehet, hogy rontana az átláthatóságon az egy oldalon készítés.

Úgy döntöttem, minden tulajdonsághoz felveszek egy kis lenyitható menüt. Ezek a menüpontok egyértelműen elhatárolják egymástól a tulajdonságokat. Ha valamire szükségünk van csak lehúzzuk, ha nem akarjuk a feladatunkban látni, összecsukjuk a menüt és nem kerülnek az új feladatba a tulajdonságok. Ezt úgy oldottam meg, hogy az egyes tulajdonságokat állító nézet elemek láthatóságát eltűntre állítottam. A leírás kivételével, ami egy egyszerű szövegdoboz, a többi tulajdonsághoz több elem tartozik. Ezeket az elemeket közös elrendezésbe raktam és ezeknek az elrendezéseknek változtattam a láthatóságát.

Hogy ezt a viselkedést megvalósítsam létrehoztam két segédfüggvényt. Az egyik a **setText** függvény, ami paraméterül kapja a szövegdobozt, egy logikai változót az állapotáról és a szövegét, majd választ, hogy az Add vagy a Remove szöveges erőforrásba helyettesítsen be. A másik **showInputView** függvény az összetett mezők nézetét várja tulajdonságul az állapotuk logikai változója mellett, e változó értéke alapján állítja be a megadott nézetelem láthatóságát.

A tulajdonságok fejlécét kezelő segéd metódus:

```
private fun setText(tv : TextView, added : Boolean, detail : String){
    if(!added){
        tv.text = getString(R.string.add_detail, detail)
    }else{
        tv.text = getString(R.string.remove_detail, detail)
    }
}
```

Ezután létrehoztam logikai tulajdonságokat a *ViewModel*ben, amik a mezők láthatóságát írják le, így tud dönteni, hogy melyik tulajdonságokat kell ténylegesen felvenni a feladathoz. Ez jó megoldásnak tűnt, ugyanis attól még, hogy eltűnik egy elem, nem veszíti el a tulajdonságait, nem baj, ha valamit véletlen lezárunk vagy hezitálunk, hogy vegyük-e hozzá a feladathoz, az értéke megmarad és a *ViewModel* dönt, hogy kezd-e majd vele valamit.

Sajnos ennyi nem biztosított teljes mértékben jó megoldást, például a tagok hozzárendelésénél, de a földrajzi pozíció miatt sem. Mivel utóbbinak a hozzáadását külön *nézet*en rendeztem, az onnan visszatérő értéket el kellett tárolnom valahol. Úgy gondoltam ennek nem a *nézet*ben lenne a helye, ezért visszaérkezéskor egyből átadom a *nézetmodell*nek és ott tároltam. Emiatt úgy döntöttem, hogy egységes legyen minden tulajdonság tárolása, hogy a *nézetmodell*ben felveszek egy egész **newTask** tulajdonságot, ami az új feladat állapotát tartalmazza és a nézet vezérlői ennek a változónak a tulajdonságait változtatják.

A feladat készítéshez használt tulajdonságok a nézet-modellben:

```
val creationStatus : LiveData<RegistrationStatus> =
createTaskRepository.subscribeStatus()
lateinit var members : LiveData<List<User>>

var descriptionAdded = false
var placeAdded = false
var personAdded = false
var checklistAdded = false

var newTask : Task? = null
```

4.5.2 Fontossági szint választás

Három szintet határoztam meg az **Importance** enumerációban és mindegyikhez jellegzetes színt választottam. Emiatt gondoltam, hogy jó lenne színek alapján dönteni a fontossági szintről. Egyértelműnek látszik, mivel egyet választhatunk csupán három lehetőségből, rádiógombokkal kéne kezelni a beállítását. Sajnos nem sikerült ezzel megoldani. Találtam egy lehetséges megoldást, hogy kép erőforrásokat lehetne rendelni a gombok állapotához, de ezt nem is sikerült egyből életre kelteni és közben elárastottam vele az erőforrásaim mappáját, szóval letettem róla.



ábra 12: Fontos feladat kiválasztva a gombokon

A korábban a tagok listájánál használt képes megoldáshoz hasonlóan használtam ebben a helyzetben is. A *nézet*re egymásba ágyazott *CardView*kat vettem fel, majd a külsőnek a **contentPadding** tulajdonságát állítva keretet csináltam a belső színes *CardView* köré. Ez legalább azért jó, mert így nem csak pár sűrűséghez közelíték a több felbontású képpel, hanem minden képernyősűrűséghez számolok keret vastagságot. Ez

igazából olyan, mintha sűrűség független pixelben adnám meg, de a `contentPadding` tulajdonságot pixelben lehet csak megadni.

A fontossági szint váltás kezelése:

```
private fun handleImportanceSwitch(view : CardView){
    resetImportance()
    view.setContentPadding(border, border, border, border)
    when(view){
        cvCrucial -> createTaskViewModel.setImportance(Importance.CRUCIAL)
        cvRegular -> createTaskViewModel.setImportance(Importance.REGULAR)
        cvImportant ->
        createTaskViewModel.setImportance(Importance.IMPORTANT)
    }
}
```

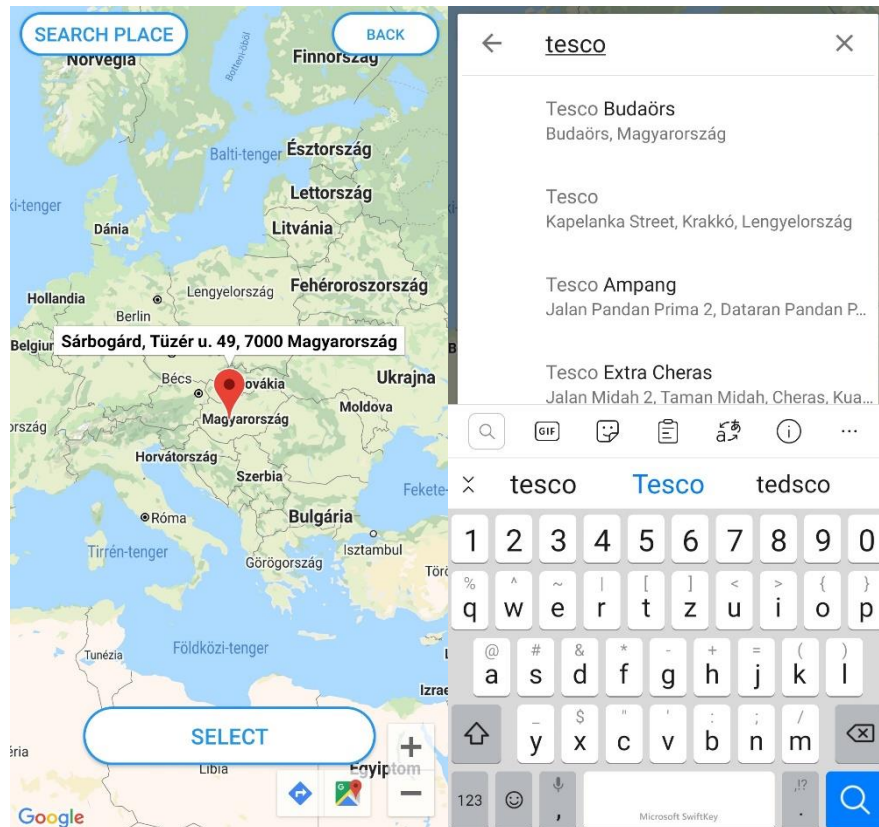
Ezután megvalósítottam kódban egy a rádiógombokéval azonos működést. Ehhez csupán egyet, a **handleImportanceSwitch** metódust írtam meg. Ezt a metódust meghívtam mindhárom nézet elem kattintás eseményében, ez egy *CardView* paramétert vár el, ami mindig a kiválasztott szinthez tartozó *CardView*. Ez a függvény három egyszerű lépésből áll. Először egy segédfüggvényben leveszem a *contentPaddinget* mindegyik nézetről, így biztos nem lehet egyszerre több kiválasztva. Ezután a függvény paraméterében kapott *CardView*nak visszaállítottam ezt a tulajdonságát, így megjelenik a keret a kiválasztott körül. Ezután egy *when* feltételeként vizsgáltam a paramétert, és minden ágán meghívtam a *viewModel*nek a fontossági szint állító függvényét a megfelelő szinttel. A *when* működése azonos a Javas switch működésével, tehát mindig egyet választ ki a lehetőségek közül.

4.5.3 Földrajzi hely választása

A földrajzi hely tulajdonság beállítása volt az egyetlen tulajdonság, aminek külön nézetet hoztam létre a beállításához. Ehhez szükség volt egy Google térképre is és a keresett helyek kiegészítéséhez is szükség volt egyéb beállításokra, úgy gondoltam ezekhez már túl sok elem tartozik és inkább külön nézetbe szerveztem a **CreateMapActivity**be. Itt a munka nem igényelt olyan sok egyedi kódolást, vagyis nem tartalmazott csak nagyon kevés logikát, a kód nagyrésze majdnem ugyanilyen minden más térképes nézetnél is. Ezzel szemben sok kutatómunkát igényelt és sok időt töltöttem el a Google fejlesztői oldalán.

Két lehetőségünk van helyet választani. Az egyik, hogy a térképünkön megérintünk egy pozíciót és ennek a hatására egy jelző jelenik meg rajta, pontosan azon a koordinátán, ahol megérintettük a térképet. Ehhez a megoldáshoz csupán a térkép

onMapClickListenerjét kellett beállítanom úgy, hogy ennek megfelelően működjön. A másik lehetőség, hogy a nézet tetején elhelyezkedő szövegdobozba kattintva elkezdhetünk beírni bármilyen címet, amiből az *alkalmazás* próbál következtetni a pontos címre és egy legördülő listából kiválaszthatjuk azt amelyikre gondoltunk. Miután ezt megtettük a térképen ugyanúgy megjelenik egy jelölő a kiválasztott cím koordinátáin.



ábra 13: A helyválasztás kétféle lehetősége

Ehhez a funkcióhoz keresnem kellett valamilyen könyvtárat, amit tudok itt használni. Szerencsére a Googlenek szinte minden hasonló funkcióra van saját könyvtára és szolgáltatása, amit használhatunk. Így találtam rá a Googlenél a *Place könyvtárra* és a *Maps API*-ra. Ezek használata kódban nem túl bonyolult, viszont sok előkészület előzi meg, csupán egy *Activity*hez hasonlóan elindítjuk az **Autocomplete** nézetet, ami a választott hely azon tulajdonságait adja vissza válaszban, amit az indítás előtt beállítunk. Ehhez szükség volt azonban egy API kulcsra, amivel a Google szolgáltatásait használhatom.

Ilyen API kulcsokat a Google fejlesztői konzoljáról lehet szerezni. Itt regisztrálnom kellett egy projektet, ami a szakdolgozathoz tartozik. Ehhez a projekthez lehet beállítani olyan tulajdonságokat, hogy például csak Android alkalmazásból lehet

használni és akár ezen belül is a saját alkalmazásunk ujjlenyomatát hozzárendelni. Ezen kívül a projektben aktiválhatjuk az API-kat amiket használni szeretnénk, így is tettem a szükséges *Maps* foglalkozó szolgáltatásokkal, amikor megtaláltam, hogy a cím kiegészítésért fizetni kell. Ekkor felfedeztem, hogy a Google a fizetős API-k teszteléséhez biztosít egy háromszáz dolláros keretet, meg kellett adnom a számlázási adataimat és már használatba is vehettem. Ez a pénzüsszeg bőven tartalmaz annyi kérés lehetőséget, ami elég a teszteléshez, így tulajdonképpen fejlesztés közben ingyenesen tudjuk használni az fizetős szolgáltatásokat is, mielőtt elérhetővé tesszük mások számára.

4.5.4 Háttérrendszerrel kapcsolatos fejlesztések

Nem nagyon tudtam az előző fejeztek alá hova beilleszteni pár dolgot, amit a *szolgáltatás* és az *adatbázis* oldalán végeztem, így egy két érdekességhez létrehoztam ezt az új fejezetet.

Ahogy mondtam az egy-több, több-több kapcsolatokat segéd táblákkal valósítottam meg, így az *adatbázisban* a feladat objektum sémája ezen tulajdonságok helyett logikai értékeket tárol arról, hogy tartozik-e ilyen tulajdonság az adott feladathoz. A *szolgáltatás* majd ez alapján a tulajdonság alapján fogja tudni megmondani, hogy nekiálljon-e a segéd táblákban való keresésnek vagy elég-e az az információ, amit a fő táblából szerez. Így legalább pár lekérdezés-kiértékelésnyi időt és erőforrást is spóroltam. Ezen kívül az *adatbázis* tárol még egy **TaskState** nevű állapotot is a feladatokhoz, ezt a feladatokat böngészése részénél bővebben kifejtem.

A létrehozás végpont meghatározása a TaskControllerben:

```
@PostMapping()  
public ResponseEntity<RegisterStatus> createTask(  
    @RequestBody(required = true) TaskDto task,  
    @RequestParam Boolean hasAssignee,  
    @RequestParam Boolean hasChecklist  
) {  
    return taskService.createTask(task, hasAssignee, hasChecklist);  
}
```

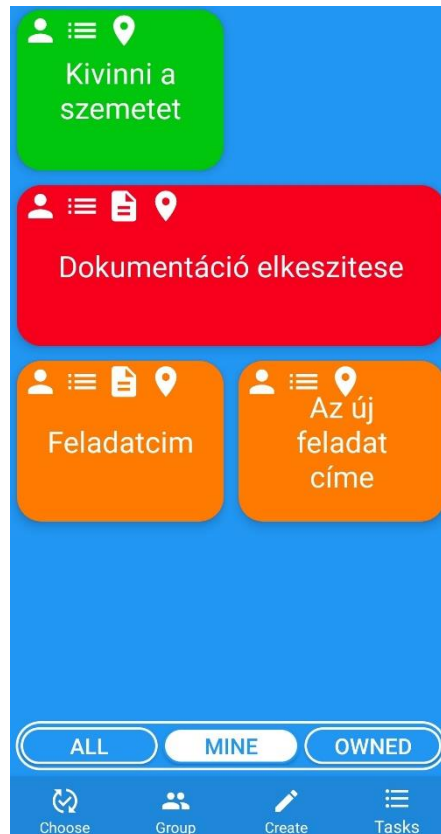
A *szolgáltatásban* az itt történő fejlesztés kicsit eltért az *alkalmazás* többi részének a fejlesztésétől. Eddig az *Android alkalmazás* egy része csak a *szolgáltatás* egy főbb osztályához tartozó kéréseket hívta, azonban itt megjelent például az, hogy a feladat létrehozása végpont mellett a csoport végpontjai közül is használtam a tagok lekérését. Ezen kívül az adatok tárolási formája miatt itt létrehoztam az első adat továbbítási objektumomat, a **TaskDto**-t. Ez az osztály tulajdonságait tekintve teljesen ugyanolyan,

mint amit az *Android alkalmazásnál* is használok, azaz az objektum lista tulajdonságokban tárolja a hozzá tartozó nem egyszerű tulajdonságokat is.

A feladat létrehozásakor egy ilyen osztályban kell átadni a létrehozott feladat paramétereit, továbbá két logikai változó és szükséges hozzá, amik azt jelölik van-e ellenőrzőlista és feladathoz rendelt személy. Ezen logikai értékek alapján dönt a *szolgáltatás*, hogy megkezdje-e a lista mezők feldolgozását. Ezen kívül mivel a létrehozás során egymás után több táblát is szerkesztek, a kérést az *adatbázis* felé egy tranzakcióba kellett fognom, hogy ne vehessen fel inkonzisztens állapotot az *adatbázis*. Erre szerencsére biztosított egy elég egyszerű megoldást a Spring keretrendszer. A létrehozó függvényt elláttam egy **Transactional** annotációval, ami az ebben történő több írást egy atomi műveletként fog össze.

4.6 Feladatok böngészése

A létrehozás megvalósítása miatt, már könnyen tudtam mindenféle különböző feladatot felvenni a rendszerbe, így nekiálltam a feladatok böngészésének megvalósításához. Ennek a nézetnek a funkciója az, hogy segítsen választani a feljegyzett feladataink közül. Ehhez különböző csoportosításokat kínál és egyéb jellemzőket is megjelenít a feladatokról.



ábra 14: Feladat böngészés nézete

Az adatbázisban bevezettem ehhez a **TaskState** tulajdonságot. A szolgáltatásban a forráskódok ugyanazokra a helyekre kerültek, ahova az előző fejezetben tárgyalt feladat létrehozása funkciónak is kerültek a kódjai. Ezeket csak a feladat csoportokba tartozó három különböző lekérdezéssel egészítettem ki, valamint létrehoztam egy **TaskBrowseDto** nevű adat továbbítási osztályt is. *Android oldalon* hasonlóan az eddigi főbb nézetekhez, minden modulban saját fájlba szerveztem a kódját, amiket **BrowseTask** előtaggal láttam el. Ezen kívül készült hozzá egy **BrowseTaskAdapter**, ami a majdnem az egész nézetet elfoglaló listát kezeli. Ezután egy **TypeConverterre** is szükségem volt a *Room* adatbázishoz és később itt is megjelent a **TaskCard**, a **TaskBrowseDto**-val egyidőben. Ezt azért vezettem be, mert a részletes nézet megnyitásakor úgy is újra elkérek minden adatot, így itt csak elég volt a legalapvetőbb tulajdonságokat elkérni és pár logikai tulajdonságot, hogy mit tartalmaz a feladat. Ez azért is jó, mert nem árasztjuk el a csatornát olyan adatokkal, amiket később nem használunk.

4.6.1 Feladatok csoportokba tagolása

Biztos voltam benne, hogy a feladatokat a böngészésnél még további csoportokba akarom osztani valamilyen állapotuk alapján, hogy könnyebben kereshetőek legyen. Úgy

döntöttem, az **összes** feladat halmazán kívül még két másikat fogok készíteni. Az egyik a **saját** feladatok csoportja lett, ahol azokat a feladatokat láthatjuk, amikhez hozzá lettünk rendelve, hogy végezzük el. Valamint a másik a **birtokolt** feladatok csoportja, ahol azokat láthatjuk, amiket mi hoztunk létre. Amikor a csoportokat terveztem olyan szempontokat is figyelembe vettem, amiket nem vettem bele a szakdolgozat készítésébe, ám jó ötleteknek tartottam jövőbeli fejlesztésnek. Ilyen például, hogy a saját feladatoknál kaphassunk egy olyan funkciót, amivel vissza tudjuk dobni a nekünk osztottakat, valamint a birtokolt feladatoknál egy olyan funkció, ami a kiválasztott lezárt birtokolt feladatot elmenti, a lista végére tolja és ezek sablonként szolgálnak, hogy újra ki tudjuk írni ugyanezt a feladatot.

A saját, nyitott feladatok szűrése:

```
List<Integer> taskIds =
    em.createQuery("SELECT a.taskId FROM Assignee a WHERE a.userId = :id",
        Integer.class).setParameter("id",userId).getResultList();

for(Task task : groupTasks){
    if(task.getTaskState() == TaskState.OPEN){
        if(taskIds.contains(task.getId())){
            tasks.add(new TaskBrowseDto(task));
        }else if(task.getOwnerId().equals(userId)){
            ownedTasks.add(new TaskBrowseDto(task));
        }
    }
}
```

Ehhez a működéshez hoztam létre a háttérrendszerben a **TaskState** tulajdonságot. Mivel *Android oldalon* nem válogatok az egyes feladatok közül, hanem ezt a *szolgáltatás* elintézi és mindig csak azokat adja vissza, amit kértünk, így itt felesleges volt megjelentetni ezt. Ez az állapot lehet nyitott, kész vagy mentett értékű, azonban a mentett értékkel még nem csinállok semmit, ezt is azért hagytam benne, mert a szakdolgozat elvégzése után is folytatni szeretném a feladatot. Így az összes feladatnál szimplán visszakapjuk mindegyik, még nyitott feladatot. A saját feladatoknál azokat a nyitott feladatokat, amiket nekünk el kell végeznünk. Valamit a birtokolt feladatoknál mindent véget ért feladatot, ami hozzánk tartozik.

Ezután *Android oldalon* már csak annyi maradt, hogy a nézet alján lévő gombok kattintás eseményére a *szolgáltatás* megfelelő végpontjáról frissítsem a feladatokat. Ezt egy a korábban a nehézségi szint választásnál használt megoldáshoz hasonlóval valósítottam meg. Írtam egy **switchButton** függvényt, amit, meghívok minden gomb eseménykezelőjében és ez rádiógomb szerű nézetelemet csinál a három gombból.

4.6.2 Lista elem dizájnya

Egymás mellett akartam két oszlopban megjeleníteni az elemeket és négyzetes formát akartam neki, így kicsit olyan hatása van mintha, cetlik lennének egy táblára tűzködve, természetesnek tűnik. Itt is ki akartam emelni színekkel a fontossági szintek megkülönböztetését, ezen kívül azt találtam ki, hogy a sürgős feladatok mindkét oszlopot elfoglalják egy sorban, még a méretével is felkelti a figyelmet.

A két oszlopos rendszer megvalósításához *GridLayoutotManagert* állítottam be a feladatkártyákat megjelenítő *RecyclerView*nek. Ebben, hogy két oszlop vastagra állíthassam a sürgős feladatokat, felülírtam a **getSpanSize** függvényét. Ez simán ellenőrzi az *adapterben* a megadott indexű elemet és ha sürgős a fontossága, akkor duplát ad vissza oszlop méretnek.

Bár részleteket itt nem jelenítek meg, minden feladatról jelezni akartam, hogy érdemes-e megnyitni a részletes nézetét. Ezt képekkel akartam jelezni a feladatkártyák felső sarkában. Ezeket a piktogramokat az **AndroidDrawableImporter** minta ikonjai közül választottam ki, ahol nagyon jó piktogram minták voltak. Ezeknek a tartásához létrehoztam egy *LinearLayout*ot a feladatkártyák nézetének a bal felső sarkában, majd az Android *LayoutInflater*ét használva *ImageView*kat fűjtam fel bele, amikor szükség volt. Mindegyek jelen lévő tulajdonsághoz felfűjtam egy kép nézetet és a megfelelő piktogramot állítottam be forrásaként, ezt az lista *adapterben* tettem meg.

A piktogramok felfűzése a feladatkártyára:

```
fun inflateDetail(drawableId : Int){
    val pictogram =
        activity.layoutInflater.inflate(R.layout.inflate_pictogram, pictoHolder,
            false)

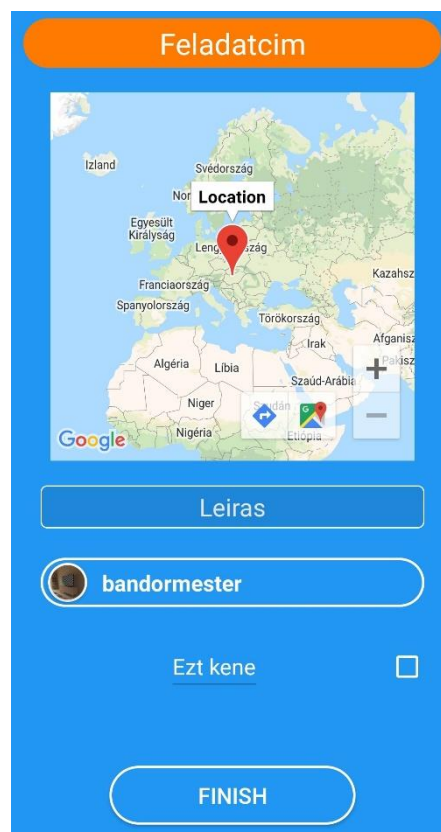
    pictogram.ivPictogram.setImageDrawable(ResourcesCompat.getDrawable(activity.resources,
        drawableId, null))
    pictoHolder.addView(pictogram)
}
```

Az ehhez szükséges tulajdonságokat legjobban úgy tudtam leírni, hogy lemásoltam magának a **Task** osztálynak a tulajdonságait. Ezek közül a tulajdonságok közül a fontosabbakat, amik a kártyán valós értékükkel megjelennek, megtartottam. A többi tulajdonság típusát logikaira cseréltem, mert itt elég azt megmondani belőle, hogy

szükség van-e a piktogramra vagy nincsen. Ez lett a **TaskCard** osztály az *Android oldalon*, és **TaskBrowseDto** a *szolgáltatásban*. A *szolgáltatás* mindhárom feladatcsoportot visszaadó végpontja ilyen **TaskBrowseDto** típusú választ küld.

4.7 Feladatok részletes nézete

Tulajdonképpen ez már az utolsó eleme az *alkalmazásnak*, amit megvalósítottam. A böngészésről tudunk ide navigálni egy kártyát kiválasztva. A *nézetnek* az a szerepe, hogy a feladat részleteit segítségként használva meg tudjuk oldani a feladatot. Ennek megfelelően a feladat tulajdonságait látjuk rajta, valamint egy gombbal lezárhatjuk a feladatot.



ábra 15: Részletes nézet térképpel, leírással és személyekkel

A fejlesztés ezen pontján ennek az új nézetnek a bevezetése már nem igényelt semmilyen műveletet az *adatbázis* szintjén, már minden elkészült valamilyen korábbi funkcióhoz. A *szolgáltatásban* a most szükséges új végpontokat az eddigi főbb modulokban a feladatokhoz tartozó fájlokba raktam, ez pár új függvényt jelentett csak, valamint létrehoztam egy **TodoCheckRepositoryt**, amit a feladatok **Service** rétegéből fogok hívni. *Android oldalon* hasonlóan a többi nézethez elkészült ehhez is minden főbb

modul az architektúrából **TaskDetails** előtaggal, valamint a *Room* adatbázis **TaskDao**jában is megfogalmaztam egyedi lekérdezéseket. Ezen kívül a **TaskService** interfésze kiegészült még pár végponttal, valamint az előző, böngészés funkcióhoz létrehozott **TypeConverter**t is ki kellett egészítenem pár metódussal. Végül itt fedtem fel egy-két hibát, amit a korábban megvalósított részek okoztak, ezeket is javítottam.

4.7.1 A tulajdonságok megjelenítése

Ahogy a dolgozat korábbi részeiből kiderül, nem feltétlenül jelenik meg minden tulajdonság egy adott feladatnál, ezért valahogy meg kellett oldanom, hogy csak a jelen lévő tulajdonságokhoz tartozó nézet részeket jelenítsem meg. Ehhez részben a feladatok létrehozásánál használt megoldást is implementáltam újra, de a térkép megjelenítésénél inkább a *layoutInflater*t használtam. Először a térkép nézeténél a *layoutInflater*es megoldással kezdtem, viszont ezt valamivel hosszabb volt elkészíteni. Azt gondoltam, mivel a többi tulajdonsághoz tartozó nézet amúgy is egyszerű elem, alig számít valamit a létrehozásuk. Ezért nem foglalkoztam a felfújásukkal, csak áttértem a láthatóság állítására.

Erre az *Activity*re a böngészésről kerülünk úgy, hogy a kiválasztott kártya átadja a saját tulajdonságait az oldalnak. Így miután a *nézet*re kerülünk és elindul a feladat részleteinek a lekérdezése, a feladat címét és fontosságát be tudom állítani. Ezen kívül a kártya logikai tulajdonságai alapján megjelenítem az egyes nézet elemeket. Így már akkor is meg tudok jeleníteni valamit a *nézet*en, ha nem tároltam semmilyen előzményét lokálisan a feladatnak. Majd miután megérkeznek az adatok a *nézet* frissíti minden megjelt tulajdonság értékét.

Jelölő mozgatása a megadott helyre:

```
private fun moveMap(lat: Double, lng: Double){
    mMap?.moveCamera(CameraUpdateFactory.newLatLng(LatLng(lat, lng)))
    val markerOptions =
        MarkerOptions().title("Location").position(LatLng(lat, lng))
    mMap?.addMarker(markerOptions)
}
```

Ezen belül is a térkép egy összetettebb elem, amit nem csak egyszerű tulajdonságok alapján állítunk. A térkép kezeléséhez fel kell venni egy **GoogleMap** típusú objektumot és lekérni egy **getMapAsync** metódussal. Miután ezt az objektumot kezelem. Földrajzi hely érkezésekor új jelölőt hozok létre a térképen és a térkép közepét ehhez a ponthoz igazítom, ezt mind a térkép szolgáltatás segéd függvényeivel teszem.

4.7.2 Az feladat adatainak elérése

Itt kicsivel bonyolultabb volt az adatok elérése, mint a többi funkciónál. Ez azért volt, mert ahogy mondtam a *szolgáltatásban* nem tudtam teljesen jól bánni a kompozit kulcsos *entitásokkal*, ezért saját lekérdezést kellett írnom a feladathoz rendelt emberek megkapásához. Először lekértem a feladat *entitást* és ha volt hozzá személy vagy ellenőrző lista, azokat is lekértem. Ezután összeállítom a **TaskD**-toba, aminek van egy **Task** osztályos konstruktora.

Ez mikor megérkezik az *Android alkalmazáshoz* egyből mentésre kerül a lokális adatbázisba, de itt már nem került szóba a segéd táblás tárolás. Az összetett mezők tárolásához létrehoztam két újabb metódust az adatbázishoz tartozó **TypeConverter**-be. A hozzárendelt felhasználók és az ellenőrzőlista tulajdonságot is azért nem tudta menteni magától, mert listák voltak. Azt találtam, hogy leginkább JSON formában szokás elmenteni, szöveg típusú tulajdonságként. Ehhez találtam egy remek könyvtárat, ami majdnem pontosan ezt csinálja. Annyi kiegészítést kellett hozzátennem, hogy a tömböt még átkonvertáltam egy listába. Bár a szöveg irányába tudtam listát is átalakítani, visszafelé valamiért nem működött és a tömbös megoldás megtartottam.

Típusátalakító az ellenőrzőlistához:

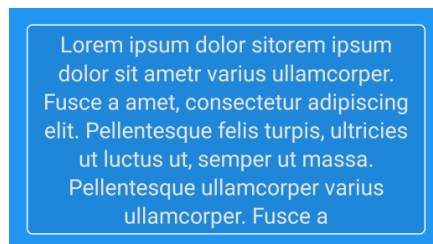
```
@TypeConverter
fun checklistToJson(value: List<Check>?): String = Gson().toJson(value)

@TypeConverter
fun jsonToChecklist(value: String) : List<Check> {
    val array = Gson().fromJson(value,
        Array<Check>::class.java)?::emptyArray()
    return array.toList()
}
```

4.7.3 A nézet dizájnja

Ahogy említettem a feladat létrehozásánál, úgy akartam elkészíteni a részletes és a létrehozó *nézetet*, hogy hasonlítsanak egymásra. A tulajdonságokat olyan sorrendben próbáltam elhelyezni, ahogy a létrehozás *nézeténél* is fentről lefele haladunk a beállításuknál. Annyi különbséggel, hogy itt fejléce sincsen a nem megjelenített tulajdonságnak.

Ezen felül kitaláltam egy nagyon jó kis animációt jövőbeli megvalósításra, ahogyan a böngészés *nézetéből* választott kártya egész egyszerűen felcsúszik a *nézet* a tetejére, ahogy látható és megjelennek alatta az adatok.



ábra 16: Leírás a részletes nézetben

Ezen kívül, ahogy a fehér szövegről is mondtam a dolgozat elején, hogy rövid szövegrészek megjelenítésére szokás használni a fehér színt. Erre egyszerűen ez a magyarázat, hogy kellemetlen és a szem is gyorsabban fárad, ha hosszú szöveg színe világos. Emiatt úgy döntöttem, hogy berakom a leírás szövegét egy keretbe, aminek változtatni tudom a háttérszínét, majd ezt állítottam egy kicsit. A fehér színű szöveget egy sötétebb szürkés sárgás színre cseréltem és a keret háttérszínét is enyhén sötétebb kékre állítottam. Ez annyira nem feltűnően észrevehető és mivel a sárga elűt a kéktől és a szöveg színes valamivel sötétebb lett, remélhetően javult az olvashatóság és kevésbé káprázik a szöveg.

4.7.4 ISP megsértése

Azért emeltem ki ezt a kis hibát, mert a korábbi munkáimra jellemző volt, hogy a hanyag tervezés miatt komolyabb gondokba futottam. Itt viszont ez volt az egyetlen komolyabb probléma, amit nem vettem észre elsőre és később futottam bele, hogy rossz. Ráadásul az objektum orientáltság főbb tervezési elvét szegtem meg benne, az *interfész szétválasztási elvet* (ISP) [18], vagyis, hogy ne függjenek osztályok olyan metódusoktól, amit nem használnak.

Ahogy korábban említettem, a tagok lista *adapterjét* újra akartam használni. Ehhez úgy is készítettem el a nézetet, hogy a törlés gomb eltüntethető a személy eleme mellől. Úgy valósítottam itt meg a kattintás esemény kezelését, hogy felvettem az *adapterben* egy **Listener** tulajdonságot, ami megvalósította az *adapter* által publikált **ItemClick** interfészt. Ennek a **Listenernek** mindig az adott nézetet adtam értékül, amin használtam az *adaptert*. Ennek az interfésznek a metódusai közé vettem a sima elem kattintás és a törlés gomb kattintás eseménykezelőjét is. Végül akkor jöttem csak rá, hogy ez így nem jó, amikor itt implementáltam az interfészt és nem tudtam mit írni a törlés gomb metódusába. Egyből javítottam és két külön interfészre bontottam a működést két **Listener** tulajdonsággal.





4.8 További gondolatok

Miután végeztem idáig a dolgozattal rájöttem még, hogy van pár általános dolog, amit semelyik fejezetben sem tudtam igazán elhelyezni, vagy csak elfelejtettem hozzávenni valamelyik fejezethez és már nem akartam utólag megbontani a fejezetek szerkezetét.

4.8.1 Adatbázis tulajdonságok

Az *adatbázis* adta lehetőségeket is ki akartam használni. Azon kívül, hogy egy egyszerű adattárolónak használok, megadtam itt tulajdonságokat, amik segítenek konzisztens állapotban tartani a rendszer és javítanak a teljesítményén.

A lekérdezések gyorsításához számos indexet állítottam be az egyes táblákra. A felhasználóknak a felhasználónév mezőjére, mert a bejelentkezéskor felhasználónév alapján keresek a táblában. A csoportoknál csak a csatlakozási kódra, mert ilyenkor ebben a kód mezőben keresem a csoportot. A beosztottak táblájára felesleges volt saját indexet készítenem, a kapcsolótábla két mezője alkotja az összetett elsődleges kulcsot, az elsődleges kulcsra pedig alapértelmezetten van definiálva index. A feladatoknál két indexet is létrehoztam, egyet csak a csoport azonosítójára, a másikhoz pedig e mellé hozzávettem a tulajdonos azonosítóját. Kétféleképpen keresek feladatot. Egyszer a csoportban a lévő összes feladatot keresem, ehhez csak a csoport azonosítóját használok. A másik esetben a saját feladataimat keresem csoporton belül, így itt a csoport azonosítója mellett a tulajdonos azonosítóján keresek. Ezen kívül az ellenőrző lista elemeinél a feladat azonosítójára is állítottam be indexet.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI
 id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 full_name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 user_name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 password	VARCHAR(200)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ábra 17: Részlet a felhasználók tábla jellemzőiről

A további, táblákat jellemző tulajdonságokat az egyértelműség és konzisztencia fenntartásához próbáltam beállítani. A kapcsolótáblákkal nem tudtam semmit csinálni, hiszen két mezőjük van, amik alkotják a kulcsot. A felhasználók tulajdonságai közül mindegyiket nem nullára állítottam, hiszen a regisztrációhoz minden adatra szükség van, ha valamelyik hiányzik, akkor valami rossz. Ezen kívül egyedivé tettem a felhasználónevet, hiszen ez alapján különböztetem meg a felhasználókat

bejelentkezéskor. A csoportoknak is nem nullára állítottam a leírásán kívül minden tulajdonságát, mivel azokat kötelezően meg kell adni. Továbbá egyedivé tettem a csatlakozási kódot, mert ennek kulcsnak kell lennie, hogy helyesen működjön a csatlakozás. A feladatoknál is nem nulla tulajdonságokat állítottam be. Itt már több tulajdonság kimaradt, de a logikai értékeknek, hogy van-e beosztott vagy ellenőrzőlista mindenképpen kellett értéket tartalmazni, mert ezek alapján adok vissza feladatot a *szolgáltatásban*.

4.8.2 Android erőforrások

A megvalósítás során elég sok erőforrást kellett létrehoznom, már csak az elrendezések önmagában is elég sok fájlt adnak a projekthez. Rövid keresés után azt találtam, hogy nem lehet mélyebb szinteken mappákba csomagolni az erőforrásokat, ezért nehezen tudtam fenntarthatóan szervezni a fájlokat. Emiatt elkezdtem egy olyan elnevezési módszert használni, amivel könnyebben átláthatóvá vált az egész. Először hasonlóan neveztem el őket, mint a forrásfájlokat is például **LoginViewModel**. Azaz előre írtam, hogy mivel foglalkozik és mögé, hogy a rendszer milyen eleme, de ezt a mappákra osztás miatt lehetett megtenni. Ezért itt felcseréltem a kettőt és nagyot javult számomra a helyzet. Így könnyebben is lehetett olvasni, mert sokszor ugyanaz a szó állt egymás alatt és a rendezés miatt olyan, mintha típusok alapján csoportosítva lennének az erőforrások.

Részlet a strings.xml erőforrásból:

```
<resources>
  <string name="app_name">ToDoApp</string>
  <string name="username">Username</string>
  <string name="password">Password</string>
  <string name="register">Register</string>
  <string name="log_in">Log in</string>
  ...
```

Próbáltam a lehető legnagyobb mértékben használni a technológiához biztosított erőforrás állományokat, a színeket és a szövegeket. Így az *alkalmazás* színe központosítva állítható és a szöveges erőforrások átírásával könnyen fordítható több nyelvre az *alkalmazás*. Ezen felül próbáltam minél praktikusabban kihasználni a stílusokat is, hogy minél kevesebb megjelenési tulajdonság jelenjen meg az elrendezésekben. Ezt még nem sikerült teljesen jól kihasználni, már a fejlesztés elején nehezen átláthatóvá tettem és nem is tudtam rajta javítani a végéhez közeledve sem.

Az elrendezéseket létrehozásakor próbáltam a *ConstraintLayout*-ot erőltetni. Noha nem mindenhol az optimális vagy legegyszerűbb megoldás, mindent le lehet írni benne. Ezen felül számomra sokszor ezzel volt a legkönnyebb sűrűségfüggetlenre megcsinálni egy nézetet, vagy egyből működőképesre csinálni a nézetet álló és fekvő nézetben is. Nem utolsó sorban sok helyen olvastam róla ajánlást, mert sok lehetőség van benne, valamint nagyon jól tervezővel is be lehet állítani az elemeket.

5 A megvalósítás után

Ebben a fejezetben a fejlesztés utáni tesztelésekről és a jövőbeli tervekről írok. Tesztelésből csak manuális tesztelést végeztem, viszont bevontam ismerőseimet is. A projektet pedig mindenképpen folytatni fogom a jövőben.

5.1 A tesztelés

A rendszer fejlesztése közben fokozatosan végeztem tesztelést is. Az egyes *nézetek* lefejlesztése után manuális teszteltem és közben a figyeltem a rendszer állapotára is. Minden teszteléskor megpróbáltam minél több lehetséges bemenettel dolgozni. Miután végeztem az egész rendszerrel, még újra végig teszteltem egyszerre az összes részt.

Miután úgy tűnt, hogy minden hiba nélkül működik a használhatóságát is megpróbáltam tesztelni. Ehhez arra gondoltam megkeresek másokat, hogy próbálkozzanak élesben a használatával. Megkértem az adott személyt, hogy a használati eseteket sorban végezze el, miközben én figyelmet mit csinál. A belépés és regisztrációkor a bejelentkezési képernyőről indulva, majd a többi esetben a csoport választásról, ahova érkezünk bejelentkezés után. A tesztesetek közben próbáltam mindenhol követni a viselkedésüket és a hibákat vagy furcsa eseteket külön feljegyeztem. Létrehozáskor mindenki olyan feladatot készített, amelyet szeretett volna. Viszont a térképes nézetet meg kellett nyitniuk akkor is, ha nem akartak helyet adni neki.

5.1.1 Első eset

Először egy volt osztálytársamat kértem fel a tesztesetek végrehajtására. A bejelentkezési képernyővel nem volt semmilyen gondja. Ellenben a regisztráció közben elsőre nem töltött ki minden mezőt, de ezt pótolta egyből a hiányjelző üzenet után. Folytattuk a csoport választás képről a többi tesztesettel. Csoportot gyorsan létre tudott hozni hiba nélkül, de lassabban a regisztrációhoz képest. A csoporthoz való csatlakozást és a kiválasztását is zökkenőmentesen csinálta. Majd első csoport választás után a csoport részletes *nézetén* a feladatok böngészéséhez navigálás helyett, hezitálva megnyomta az egyik csoporttagok, majd miután megjelent a tag teljes neve lassan megtalálta az alsó navigációs sávot. Ezután egyből felismerte a böngészéshez tartozó menüt és gyorsan rájött a feladatcsoportok közti váltásra is. A részletes *nézetén* azt jelezte, hogy néhány pillanatig nem tudta átlátni. A feladat befejezéséhez is várt egy kicsit a gomb

megnyomása előtt. Végül a feladat létrehozását kellett megcsinálnia, aminél két dolog is előjött. Először is elsőre nem állított be címet a feladatnak, mert a mezőjét nem vette észre miután görgetett egy kicsit. Ezen kívül a hely hozzárendelésekor nem találta meg a cím kiegészítő lehetőségét, csak a térképet érintős opciót. Ezt leszámítva gyorsan kezelte minden tulajdonság előhozását és állítását.

5.1.2 Második eset

Másodikként a testvéremet kértem fel tesztelni. Erről nem vonhattam le semmilyen elhamarkodott következtetést, ugyanis a testvéremnek többször megmutattam a projekt állapotát fejlesztés közben. Elmagyaráztam neki néhány funkciót, vagy megkérdeztem tőle, hogy neki mi tetszene jobban. De a fejlesztés utolsó fázisában már nem tájékoztattam őt és osztottam meg vele semmit. A belépéses és regisztrációs részt hiba nélkül vette, ezt már korábbról ismerte. A csoportok kezelését is tökéletesen tudta működtetni azonnal és a navigációval a feladat egész része alatt nem volt gondja. A feladatok böngészésénél és részletes *nézet*énél is jól eligazodott és átlátta a rendszert, valamint egyszerűen lezárt feladatot. A feladatok létrehozása részénél ő is kétszer került kétes szituációba. Ő se találta meg elsőre a feladat címét és nem tudta hirtelen, hogy keresse amikor eszébe jutott. A másik probléma az volt, hogy véletlenül a térkép mező kinyitása helyett a leírás szövegdozarába kattintott. Összességében a két figyelmetlenségi hibán kívül a végén mindent tudott.

5.1.3 Harmadik eset

Harmadik esetben anyámnak adtam oda a készüléket és elindítottam a használati esetekkel, amit végre kellett hajtania. Ez sokkal nehezebben ment, mint az előző két esetben. Már a regisztrációnál nem akart írni a legtöbb mezőbe és nehezen jött rá, hogyan tud képet adni a felhasználójához. Belépés után a csoportok létrehozását végre tudta hajtani fennakadás nélkül. A csoportba való csatlakozáskor a csoportok részletes *nézet*énél csak idővel jött rá a csatlakozó kód másolására, de ezután tudta, hogyan kell használni a csatlakozás funkciót. Amikor a feladatok böngészését kellett volna végezni, hosszas értetlenkedés után rávezettem az alsó navigációs sáv használatára, így a böngészésig eljutott. Hamar váltott a feladatcsoportok között, de a részletes *nézet*re váltáskor kis idő után nyomott csak rá magára a feladatkártya elemre. A részletes *nézet*en viszonylag hamar le tudta zárni a feladatot és értette. A feladat készítésekor jól teljesített az előző eredményeihez képest. A tulajdonságokat hamar meg tudta jeleníteni és kis

hezitálás után el is tudta kezdeni állítani őket. Ő volt az egyetlen, aki a cím mezőt nem hagyta ki a kezdéskor. A földrajzi hely meghatározásánál ő sem találta meg a cím kiegészítő gombot és ő nem is akart magától helyet adni a feladathoz.

5.1.4 Értékelés

Működésbeli problémát a tesztelések közben nem tapasztaltunk. Ebből arra következtettem, hogy használható élesben a rendszer. Amilyen eredményekre pedig szükségem volt, azt megkaptam ezekből a tesztekben. Ez alapján használhatóbbá tudtam tenni a programot. Felhívta arra a figyelmemet, hogy a nézet elemek túl közel helyezkednek el egymáshoz a feladat készítésénél, ezért itt kicsit növeltem a margót közöttük. Ugyanitt még egy probléma lett abból, hogy többször nem tudtak címet állítani egyből a feladatnak, ezért ezt a mezőt is jobban ki kellett emelnem. Valamint abból, hogy a térképes nézeten nem találják a cím kiegészítés lehetőségét, tehát ezt is ki kellett emelnem valamivel jobban.

5.2 A jövőbeni fejlesztések

Bár most a félév végével kész feladatként adom be a projektet, még szeretnék rajta később is dolgozni. Főleg a dolgozat végéhez közeledve zavart, több funkcionál elérhetőek még elegánsabb megoldások. Ezen kívül nem sikerült annyit megvalósítani, amivel megelégedtem volna, még sok funkciót szívesen fejlesztenék bele.

5.2.1 Az új funkciók

Pár új fejlesztési ötletet már írtam a dolgozat korábbi részei alatt is. Ilyen volt, hogy lehessen a lezárt feladatokat sablonként használni új létrehozásához, a feladatokat miután hozzárendeltek le tudjuk adni és a részletes *nézet* megnyitásához vezető animáció is.

Továbbá úgy gondolom jó lenne megvalósítani, hogy a felhasználók tudják szerkeszteni a profiljukat és jutalmakat kapni hozzá. Valamint függetlenül a csoporttól interakcióba léphessenek egymással. Amikor folytatom a rendszer fejlesztését, ez lesz a következő funkció, amit megvalósítok. Sokkal interaktívabbá teszi alkalmazást és egy alkalmazáson belüli chat felület is nagyon hasznos funkció, főleg mivel csoportban dolgozunk.

Ezután kiegészíteném további tulajdonságokkal a feladatokat. Felvennék például határidővel idővel kapcsolatos jelzéseket vagy telefonszámokat, amire egyből az alkalmazáson belül indíthatunk hívást és akár egy olyan titkos mezőt is, amit csak a beosztottak látnak.

Ezen kívül a tesztelés után úgy gondoltam, egy nagyon rövid tájékoztatót kéne csinálni az alkalmazáshoz. Ez első indítás után rámutatna az alkalmazás fontosabb részeire és megnevezné őket. Ezt mindig újra meg lehetne jeleníteni, amikor új funkció érkezik.

További mobilos lehetőségeket kihasználó funkciók is érdekelnek. Ha van rá mód, kiegészítenem például Bluetooth-on vagy NFC-n keresztüli csoportba való csatlakozással. Vagy automata csoport váltást lehet készíteni földrajzi hely alapján, esetleg az alapján, hogy vagyunk-e csoporttagunkkal közös wifin.

Esetleg tovább lehetne fejleszteni az *alkalmazást* egy közösségi feladat megosztóvá vagy alkalmi munka keresővé. Földrajzi hely alapján folyamatosan váltogatna egy csoportot. Ez akár város vagy egy kisebb körzet szintjén is lehet. Az itt lévő emberek tudnának feladatok kirakni és aki segíteni akar hozzárendeli magát. Ehhez még az alkalmazáson belüli fizetéseket is fel lehetni venni, hogy még kényelmesebb legyen használni.

5.2.2 Következő verzió érkezése

Amint lehetőségem lesz rá folytatni szeretném a projektet. Azt tervezem, hogy a mesterképzés alatt, akár az egyetemen fejlesztem tovább. Ha valamilyen tárgy keretein belül folytatni lehet saját rendszert vagy valamilyen szempontból elemezni vagy fejlesztéseket végezni, akkor ezt a projektet viszem majd. Azt hiszem erre legkésőbb egy év múlva a diplomamunkához van lehetőség.

Ezen felül néha hobbiból is szoktam egyszerűbb alkalmazásokat tervezni a volt osztálytársaimmal és már hosszú ideje célom, hogy valami olyat fejlesszek, amit fel tudok tölteni valamilyen alkalmazás üzletbe. Azt hiszem így amikor tanuláson és munkán kívül foglalkozok fejlesztéssel, azt az időt ebbe a projektbe fogom fektetni. Viszont, legkésőbb a diplomamunka előtt tervezek egy ráncfelvarrást a rendszernek. Így amikor legközelebb komolyabb időbefektetéssel folytatom a projektet már tudok a fontos és új fejlesztésekre koncentrálni.

6 Az elvégzett munka összefoglalása

A félév során elég sok különböző feladatot végeztem a rendszer fejlesztéséhez. Először ki kellett találnom, mit készítek, amihez végül a valós életből merítettem ötletet. Hasonló alkalmazásokat és megoldásokat kerestem, hogy megismerjem mi a piacon az elvárás és ötletet szerezzek a saját megvalósításomhoz is. Kiválasztottam, hogy milyen technológiákat fogok használni, amik közül a legtöbb jó választásnak bizonyult és úgy éreztem jól tudom alkalmazni rajta a képzés alatt tanultakat.

Az valódi munkát a tervezésével kezdtem. Meghatároztam az *alkalmazás* használati eseteit és ezeket a funkciókat szétszítottam nézetekre. Ezután megjelenést terveztem a *nézetek*nek, amivel szívesen foglalkoztam a projekt ezen része alatt. Ezeket segítségül használva megterveztem az *adatbázis* sémáját és a *szolgáltatás* architektúráját. Ezután megpróbáltam minél többet megtanulni az MVVM architektúráról és ez alapján megterveztem az *Android alkalmazást* is. Ez az architektúra tervezést és tanulást nagyon hasznosnak tartom, sokkal könnyebbé tette az alkalmazásfejlesztést.

Létrehoztam egy *adatbázist*, amin tulajdonságok és indexek beállításával támogattam a rendszert többi részét. Továbbá egy erre illeszkedő *szolgáltatást* is készítettem Spring keretrendszerrel, amiben törekedtem a keretrendszer által biztosított megoldásokat használni, de sajnos még mindig előfordult, hogy natív SQL lekérdezést hajtottam végre, ezen is fejlesztenék a jövőben. Nem utolsó sorban egy ezt a *szolgáltatás* használó mobil *alkalmazást* készítettem, amiben próbáltam újabb számomra még ismeretlen eszközöket használni. Így megalkottam egy önállóan működő rendszert.

Végül a tesztelést idő szűkében egy ilyen rendhagyó módszerrel végeztem, de talán ez volt a legmegnyugtatóbb és legszórakoztatóbb része egyaránt a dolgozatnak. Ezalatt úgy érzem hasznos információkat gyűjtöttem a mindennapi felhasználók szemszögéből.

Összességében az volt a célom, hogy az Android tudásomat még jobban elmélyítsem és megpróbáljak olyan technológiákat is használni, amikkel eddig nem tudtam foglalkozni a hiányos gyakorlatom miatt. A háttérrendszerrel az volt a fő célom, hogy az is jobban megértsem mi megy a mobil platform mögött. Úgy érzem a ezeket a magam szintjén sikerült megvalósítani, de még mindig van hova fejlődni és szívesen tanulok is tovább.

7 Irodalomjegyzék

- [1 Intel Corporation, „Over 50 Years of Moore's Law,” Intel, [Online]. Available:
] <https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html>. [Hozzáférés dátuma: 9. december 2020.].
- [2 StatCounter, „StatCounter Global Stats,” StatCounter, november 2020. [Online].
] Available: <https://gs.statcounter.com/>. [Hozzáférés dátuma: 9. december 2020.].
- [3 Atlassian, „Trello,” Atlassian Corporation Plc, [Online]. Available:
] <https://trello.com/home>. [Hozzáférés dátuma: 9. december 2020.].
- [4 Apple, „Use reminders on your iPhone, iPad or iPod touch,” Apple Inc., 16.
] szeptember 2020.. [Online]. Available: <https://support.apple.com/en-us/HT205890>. [Hozzáférés dátuma: 9. december 2020.].
- [5 Baeldung, „Baeldung: Java, Spring and Web development tutorials,” Baeldung
] SRL., [Online]. Available: <https://www.baeldung.com/>. [Hozzáférés dátuma: 9. december 2020.].
- [6 Google, „Google Developers,” Google LLC, [Online]. Available:
] <https://developers.google.com/>. [Hozzáférés dátuma: 9. december 2020.].
- [7 CodingWithMitch, „CodingWithMitch.com,” CODINGWITHMITCH, [Online].
] Available: <https://codingwithmitch.com/>. [Hozzáférés dátuma: 9. december 2020.].
- [8 Holistic Software, „dbdiagram.io - Database Relationship Diagrams Desing Tool,”
] Holistic Software, [Online]. Available: <https://dbdiagram.io/home>. [Hozzáférés dátuma: 10. december 2020.].
- [9 Postman, „Postman | The Collaboration Platform for API Development,” Postman
] Inc., [Online]. Available: <https://www.postman.com/>. [Hozzáférés dátuma: 10. december 2020.].

- [1 MockFlow, „MockFlow - Online Wireframe Tools, Prototyping Tools,” MockFlow, 0] [Online]. Available: <https://mockflow.com/>. [Hozzáférés dátuma: 10. december 2020.].
- [1 Oracle, „MySQL :: MySQL Workbench,” Oracle Corporation, [Online]. Available: 1] <https://www.mysql.com/products/workbench/>. [Hozzáférés dátuma: 10. december 2020.].
- [1 Google, „final-architecture.png,” [Online]. Available: 2] <https://developer.android.com/topic/libraries/architecture/images/final-architecture.png>. [Hozzáférés dátuma: 10. december 2020.].
- [1 H. Károly és L. Izsó, Ergonómia, 2008. 3]
- [1 VerywellMind, „Color Psychology: Does It Affect How You Feel,” DotDash Inc., 4] [Online]. Available: <https://www.verywellmind.com/color-psychology-2795824#the-psychological-effects-of-color>. [Hozzáférés dátuma: 10. december 2020.].
- [1 Square, „Retrofit,” Square Inc., [Online]. Available: <https://square.github.io/retrofit/>. 5] [Hozzáférés dátuma: 10. december 2020.].
- [1 Java Guides, [Online]. Available: https://1.bp.blogspot.com/-PIbPpPIzkIA/Xwfy3JShvQI/AAAAAAAAAH_M/O8N2xbXfqtsGH7jQdfpC9f3GvtKYvT_CACLCBGAsYHQ/s1600/Screenshot%2B06-27-2020%2B22.27.59.png. 6] [Hozzáférés dátuma: 10. december 2020.].
- [1 „Descriptions of SHA-256, SHA-384, and SHA-512,” [Online]. Available: 7] <http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf>. [Hozzáférés dátuma: 11. december 2020.].
- [1 U. Thelma, „The S.O.L.I.D. Principles in Pictures,” Medium, 18. május 2020.. 8] [Online]. Available: <https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898>. [Hozzáférés dátuma: 10. december 2020.].