

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM

CONNECTIVE: Mentort és mentoráltat összekötő alkalmazás

**DIPLOMADOLGOZAT**

Témavezető:  
Dr. Antal Margit,  
Egyetemi docens

Végzős hallgató:  
Portik Andrea

**2023**

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA  
SAPIENTIA

CONNECTIVE: Aplicație care conectează mentorul și stagiarul

LUCRARE DE DIPLOMĂ

Coordonator științific:  
Dr. Antal Margit,  
Conferențiar universitar

Absolvent:  
Portik Andrea

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA  
FACULTY OF TECHNICAL AND HUMAN SCIENCES  
COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

CONNECTIVE: Application to connect mentor and mentee

**BACHELOR THESIS**

Scientific advisor:  
Dr. Antal Margit,  
Associate Professor

Student:  
Portik Andrea

**2023**

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș  
Programul de studii: Informatică

**Viza facultății:**

## LUCRARE DE DIPLOMĂ

Coordonator științific:  
**Conf. dr. ANTAL Margit**

Candidat: **Portik Andrea**  
Anul absolvirii: **2023**

**a) Tema lucrării de licență:**

CONNECTIVE: Aplicație care conectează mentorul și stagiarul

**b) Problemele principale tratate:**

- Studiu bibliografic privind importanța mentorilor la locul de muncă
- Prezentarea tehnologiilor alese pentru implementarea aplicației: Spring Boot, Android

**c) Desene obligatorii:**

- Schema bloc a aplicației
- Diagrame de proiectare și implementare pentru aplicația software realizată.

**d) Softuri obligatorii:**

- backend de tip REST Api implementat în Spring Boot
- o aplicație Android care comunică cu backend-ul printr-un REST Api

**e) Bibliografia recomandată:**

- <https://spring.io/>
- <https://spring.io/projects/spring-security>
- <https://developer.android.com/>

**f) Termene obligatorii de consultații: săptămânal**

**g) Locul și durata practicii:** Universitatea „Sapientia” din Cluj-Napoca,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, laboratorul 417

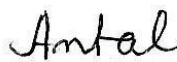
Primit tema la data de: 01.06.2022.

Termen de predare: 02.07.2023.

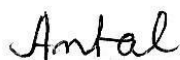
Semnătura Director Departament



Semnătura coordonatorului



Semnătura responsabilului  
programului de studiu



Semnătura candidatului



### Declarație

Subsemnatul/a .....**PORTIK ANDREA**....., absolvent(ă) al/a specializării  
.....**INFORMATICĂ**....., promoția.....**2023** cunoscând  
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a  
Universității Sapienția cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare  
de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea  
este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în  
mod corespunzător.

Localitatea, **TÂRGU MUREȘ**  
Data: **14.06.2023**

Absolvent

Semnătura.....**Portik**.....

# Kivonat

Dolgozatom témája egy telefonos alkalmazás, amely mentor és mentorált közötti kapcsolatot valósít meg.

Napjainkban mindent egyszerűen és gyorsan szeretnénk elérni, mind a magánéletben, mind a munkahelyen. Ehhez nagymértékben hozzájárulnak az okostelefonok, hiszen megkönnyítik mindennapi teendőinket. De egy új környezetben vagy helyen, ahhoz hogy gördülékenyen el tudjuk végezni a munkát, jó ha valaki utat mutat abban, hogy valójában mit is kell tennünk. Hatékonyabban tudunk dolgozni, ha megvannak a konkrét célok és a hozzájuk tartozó specifikációk, prioritások.

A projekt biztosítja a két felhasználó típus, mentor és mentorált közötti összekötést egy Androidos mobil applikáció keretein belül. Az alkalmazás lehetővé teszi, hogy bejelentkezés után a mentorok feladatokat tudjanak létrehozni egy speciális felhasználónak. Továbbá a mentoroknak lehetőségük van új tagok munkacsoporthoz való hozzáadására.

A mentoráltak pedig szintén a bejelentkezés után értesülnek az alkalmazáson belül a hozzájuk kapcsolódó tevékenységekről. Mind a mentoroknak, mind pedig a mentoráltaknak is lehetőségük van a feladataik módosítására és megtekintésére. A projektnek nem az a célkitűzése, hogy a gyakornokokat nyomon tudja követni a mentor, hanem csak az útmutatás a fő vezérelv.

A fejlesztett alkalmazás lehetővé teszi a felhasználók számára a saját személyes adataiknak, mint például mobiltelefon szám vagy email cím megjelenítését és azok módosítását. Mindezt azért, hogy a mentor és a mentorált között több elérési kapcsolat legyen, így megkönnyítve a hatékonyabb munkavégzést.

**Kulcsszavak:** mentor, mentorált, összekötés, feladatok, Android alkalmazás

# Rezumat

Tema lucrării mele este o aplicație mobilă care facilitează cooperarea între un mentor și un stagiar.

În zilele noastre, ne dorim să obținem totul ușor și rapid, atât în viața personală, cât și la locul de muncă. Smartphon-urile contribuie în mare măsură la aceasta, deoarece ne facilitează sarcinile zilnice. Dar într-un mediu sau loc nou, pentru a putea lucra în mod eficient, este bine să avem pe cineva care să ne arate cu adevărat ce trebuie să facem. Putem lucra mai eficient dacă avem obiective concrete și specificații și priorități asociate acestora.

Aplicația dezvoltată facilitează conectarea dintre cele două tipuri de utilizatori, mentor și stagiar. Aplicația permite mentorilor să creeze sarcini pentru un utilizator special după autentificare. De asemenea, mentorii au posibilitatea de a adăuga noi membri la echipa de lucru.

Stagiarii, de asemenea, sunt informați în aplicație despre activitățile lor asociate după autentificare. Atât mentorii, cât și stagiarii au posibilitatea de a modifica și vizualiza sarcinile lor. Scopul proiectului nu este de a urmări stagiarii de către mentor, ci doar de a oferi ghidajul ca principiu directiv.

Aplicația dezvoltată permite utilizatorilor să afișeze și să modifice propriile date personale, cum ar fi numărul de telefon mobil sau adresa de e-mail. Toate acestea pentru a crea mai multe căi de acces între mentor și mentorat, facilitând astfel eficiența în desfășurarea activităților.

**Cuvinte de cheie:** mentor, stagiar, conectează, aplicație Android.

# Abstract

The topic of my thesis is a mobile application that connects mentors and a mentees.

Nowadays, we want to reach everything easily and quickly both in our private life and at work, and smartphones also contribute to this, as they greatly facilitate our everyday tasks. But in a new environment or place, in order to be able to carry out the work smoothly, it is good if someone shows us what we actually have to do. We can perform more effectively if we have concrete goals and their corresponding specifications and priorities.

There are two types of users in the project, mentor and mentee, who are connected in an Android mobile application. The application allows mentors to create tasks for a user after logging in. In addition, mentors have the option to add new members to the group.

And the mentees are informed about the activities related to them after logging into the application. Just like the mentors, to mentees also have the opportunity to modify and view the tasks. The goal of the project is not for the mentor to be able to monitor the mentee, but only guidance is the main guiding principle.

The developed application enables users to display and modify their own personal data. All this in order to establish more connections between the mentor and the mentee, thus facilitating more efficient work.

**Keywords:** mentor, mentee, connect, Android application.



# Tartalomjegyzék

<b>1. Bevezető</b>	<b>11</b>
<b>2. Célkitűzések</b>	<b>12</b>
<b>3. Elméleti megalapozás és szakirodalmi tanulmány</b>	<b>13</b>
3.1. Mentorálás . . . . .	13
3.2. Hasonló szolgáltatások . . . . .	13
3.2.1. Together . . . . .	14
3.2.2. Microsoft To Do . . . . .	14
3.2.3. Slack . . . . .	14
3.2.4. Összehasonlítás a már létező eszközökkel . . . . .	14
3.3. Felhasznált technológiák . . . . .	15
3.3.1. PostgreSQL . . . . .	15
3.3.2. Spring Boot . . . . .	16
3.3.2.1. Tomcat . . . . .	16
3.3.2.2. Maven . . . . .	17
3.3.2.3. Inversion of Control . . . . .	17
3.3.2.4. Bean . . . . .	17
3.3.2.5. Proxy . . . . .	17
3.3.3. Android . . . . .	17
3.3.3.1. Android SDK . . . . .	17
3.3.3.2. Android alkalmazás . . . . .	17
3.3.3.3. Manifests . . . . .	18
3.3.3.4. Gradle . . . . .	19
<b>4. A rendszer specifikációja</b>	<b>20</b>
4.1. Felhasználói követelmények . . . . .	20
4.1.1. Fontosabb használati esetek . . . . .	20
4.2. Rendszerkövetelmények . . . . .	22
4.2.1. Funkcionális követelmények . . . . .	22
4.2.2. Nem funkcionális követelmények . . . . .	24
<b>5. Tervezés</b>	<b>26</b>
5.1. A rendszer architektúrája . . . . .	26
5.2. Modell-Nézet-NézetModell . . . . .	26
5.3. Háromrétegű architektúra . . . . .	27
5.4. Felhasználói interfész tervezése . . . . .	28

<b>6. Kivitelezés</b>	<b>30</b>
6.1. Szerveroldali megvalósítás	30
6.1.1. Adatelérési réteg	31
6.1.1.1. Adatmodellek	31
6.1.1.2. Konfigurációk	32
6.1.2. Üzleti logika réteg	33
6.1.2.1. Biztonsági intézkedések	33
6.1.3. Prezentációs réteg	34
6.1.3.1. API végpontok	34
6.2. Kliensoldali megvalósítás	35
6.2.1. Állapotmenedzsment	35
6.2.2. Navigáció	36
6.2.3. Szerveroldallal való kommunikáció	38
6.2.4. Korutinok	39
6.3. Az alkalmazás működése	40
6.3.1. Kezdőoldal	40
6.3.2. Aktivitások - Activities	41
6.3.3. Feladatok - My Tasks	41
6.3.4. Csoportok - My Groups	42
6.3.5. Beállítások - Settings	42
6.3.6. Hibák és visszajelzések	44
<b>7. Eszközök és munkamódszerek</b>	<b>45</b>
7.1. Integrált fejlesztői környezet (IDE)	45
7.2. Verziókövetés eszközök	45
7.3. API tesztelés	45
7.4. Design eszközök	45
<b>Összefoglaló</b>	<b>47</b>
7.5. Következtetések	47
7.6. Továbbfejlesztési lehetőségek	47
<b>Köszönetnyilvánítás</b>	<b>49</b>
<b>Ábrák jegyzéke</b>	<b>50</b>
<b>Irodalomjegyzék</b>	<b>53</b>

# 1. fejezet

## Bevezető

Ebben a fejezetben ismertetem a szakdolgozatom témáját, valamint megindokolom a témaválasztásomat befolyásoló okokat.

A technológiák fejlődésével egy olyan irányba haladunk, ahol két kattintás alatt elérünk bármilyen szolgáltatást az internet segítségével. A mindennapi életünk felgyorsulásával pedig még inkább nagy segítség ha a napi munkahelyi teendőinket megtekinthetjük. Ennek érdekében egyre több kis és nagy vállalat olyan alkalmazásokat vezetett be amelyek megkönnyítették a munkavégzést. De ezen alkalmazások egyáltalán vagy csak részben biztosítottak kapcsolatot a cégen belüli alkalmazott és az újonnan érkező gyakornok között.

A projekt célja, hogy ezen problémát megoldja, így hatékonyabbá tegye az alkalmazott vagy más megnevezéssel mentor és mentorált közötti munkavégzést. Ennél fogva a mentor bármikor új feladatot tűzhet ki mentoráltjának vagy más munkatársának. És ezen feladatokat könnyedén és egyszerűen megtekintheti a kezdő munkavállaló vagy gyakornok, ha bejelentkezik az alkalmazásba, amit a mentornak is meg kell tenni az alkalmazás használatához.

Az elkövetkezendőkben ismertetem nagyvonalakban a fejezetek tartalmát. Elsőként a bevezető után a felsorolom a projekt fő célkitűzéseit, hogy milyen funkcionálisokat kell biztosítson az alkalmazás. Ezt követően ismertetem a projekt fejlesztéséhez használt technológiákat, illetve bemutatok egy pár hasonló alkalmazást is. A harmadik fejezetben a rendszer specifikációit és követelményeit tárgyalom, mind a felhasználó, mind pedig a rendszer szemszögéből. Ezt követően a negyedik fejezetben részletesen bemutatom a rendszer tervezését, architektúráját. Az ötödik fejezetben az alkalmazás kivitelezését, a hatodikban pedig a munkamódszereket tárgyalom. A dokumentáció legvégén pedig egy összefoglaló található a projektről, amely során megemlítek néhány továbbfejlesztési lehetőséget.

## 2. fejezet

### Célkitűzések

A diplomadolgozatomban legfőbb célja, hogy egy olyan alkalmazást valósítson meg, amely mentorok és gyakornokok közötti összeköttetést biztosít a hatékonyabb munkavégzés érdekében.

Egy komplex alkalmazást fogok megtervezni és megvalósítani, amelynek két fő része van: egy Android kliensalkalmazás, és egy REST Api backend alkalmazás. Ennek alapján a következő célokat tűztem ki:

- szerveroldal felépítésének megtervezése,
- a szerveroldali végpontok biztonságos elérése és bizonyos adatok titkosított tárolása,
- a kliensoldal beléptető rendszerének biztosítása a felhasználó bejelentkezéséhez,
- a felhasználót érintő aktivitások megtekinthetősége,
- különböző műveletek elvégzése a felhasználók feladataival (feladatok megtekintése, szűrése vagy állapot frissítése),
- felhasználó specifikus műveletek kivitelezése (feladat létrehozása vagy szerkesztése, felhasználó meghívása egy adott csoportba),
- saját felhasználói profil szerkesztése.

## 3. fejezet

# Elméleti megalapozás és szakirodalmi tanulmány

### 3.1. Mentorálás

A mentorálásnak egy részéhez járul hozzá maga az alkalmazás is. A mentorálás nem más mint két személy közötti kapcsolat, ahol az egyik fél, a mentor egy tapasztaltabb, nagyobb tudással rendelkező személy és a másik személy pedig a mentorált. A mentor a mentorált fejlődését szem előtt tartva próbál irányt mutatni. A mentorálás a tradicionális értelemben az új munkavállaló vagy gyakornok szervezeti szocializációja és bevezetése bizonyos szakmai körökbe.[1]

A mentorálás történelmi szempontból már az ókori görögöknél jelen volt. A kifejezés Homérosz Odüsszeája művének Mentor szereplő nevéből származik. Mentór egy jó barátja volt Odüsszeusznak, aki otthon maradt a trójai háború alatt, hogy felügyelje Odüsszeusz fiának, Télémakhosznak a nevelését. Mentór jó tanácsokkal látta el Télémakhoszt a családi kötelezettségek felkészítésére. Majd a későbbiekben Athéné, a bölcsesség istennője is Mentór alakját veszi fel háromszor is, hogy tanácsot adjon az ifjúnak.[2]

Maga a szó az Egyesült Államokban az 1970-es években terjedt el a munkahelyi egyenlőség mozgalom során. Ebben az időben egy tanulmány kimutatta, hogy azon személyek, amelyek mentorálásban részesülnek hamarabb érnek el sikereket munkahelyükön mint a hasonló helyzetben lévő társaik, akik nem vettek részt a mentorálásban. A pályakezdő munkavállalók vagy gyakornok, jobban teljesítenek ha személyre szabott feladatokat kapnak ez által ösztönzően hat a munkavégzésükre is a sikerélmény.[3]

A mentorálásnak több technikája is van, amely a mentortól függ de mint mentor is többféle lehet a mentorálás során: több mentor, szakmai mentor, iparági mentor, szervezeti mentor, munkafolyamat mentor vagy technológiai mentor. Az projektben inkább a munkafolyamat mentor típus van technológiailag megvalósítva, mert a feladatok időbeli megoldására irefektál, így segítve a gyorsabb és hatékonyabb feladat befejezést.[3]

### 3.2. Hasonló szolgáltatások

Ebben az részben bemutatok néhány már létező, hasonló alkalmazást, amelyek már széles körű felhasználó csoporttal rendelkeznek.

### 3.2.1. Together

A together<sup>1</sup> (3.1 ábra) egy olyan platform, amely arra fekteti a hangsúlyt, hogy a felhasználónak a lehető legideálisabb mentort találja meg. Ezen felül biztosítja a mentor és a mentorált közötti kommunikációt és a munkafolyamatok napirendjét.

### 3.2.2. Microsoft To Do

A Microsoft To Do<sup>2</sup> (3.2 ábra) segít az alkalmazottaknak a napi feladatok kezelésében és azok megosztásában más kollégák körében.

### 3.2.3. Slack

A Slack<sup>3</sup> (3.3 ábra) egy ingyenes üzenetküldő alkalmazás, amely a szervezetek közötti kommunikációt valósítja meg, segítve az alkalmazottak rugalmas munkavégzését. Továbbá az alkalmazás képes egy üzenetet több felhasználónak elküldeni, és alkalmas a mentor és mentorált kapcsolat tartására is.



3.1. ábra. Together



Microsoft To Do

3.2. ábra. Microsoft To Do



3.3. ábra. Slack

### 3.2.4. Összehasonlítás a már létező eszközökkel

A fentebb említett alkalmazásokhoz, még hasonló rengeteg létezik a világon, amelyek közül nem mindegyik arra a célra összpontosít mint az általam megvalósított alkalmazás.

---

<sup>1</sup>Together: <https://www.togetherplatform.com/>, utolsó elérés dátuma: 2023. jún. 25.

<sup>2</sup>Microsoft To Do: <https://todo.microsoft.com/tasks/>, utolsó elérés dátuma: 2023. jún. 25.

<sup>3</sup>Slack: <https://slack.com/>, utolsó elérés dátuma: 2023. jún. 25.

A projektem csak egy mentor és mentorált közötti összekötést, feladat kiosztását oldja meg. Összeségében a projekt egy olyan lehetőséget nyújt, amely könnyen kezelhető és bárki számára elérhető.

### 3.3. Felhasznált technológiák

Ebben a részben ismertetném a dolgozat fő technológiai megvalósításait. A projektet három komponens alkotja, egy adatbázis, egy backend vagy más néven szerveroldal és végül egy kliensoldal, ami a frontendnek felel meg. Első sorban az alkalmazásban megjelenő adatokat PostgreSQL-ben tároltam. Az adatbázissal egy szerver kommunikál, amely Spring Boot keretrendszerben jött létre. Továbbá a felhasználói felület ahogy már említettem Androidban készült.

#### 3.3.1. PostgreSQL

A **PostgreSQL** [4] egy nyílt forráskódú relációs adatbázis-kezelő rendszer. Használata során biztosítja az adatok integrálását, a megbízhatóságot és egy néhány robosztus szolgáltatást. A PostgreSQL-t 1986-ban a Berkeley-i Kaliforniai Egyetemen POSTGRES projektje során jött létre, amit a későbbiekben minden nagyobb operációs rendszeren lehet már futtatni. Az adatbázis-kezelő rendszer 2001 óta biztosítja az ACID<sup>4</sup> alapelveket. A PostgreSQL megpróbál megfelelni az SQL-szabványnak, amelynek az SQL:2016 Core 179 megalkuvásból 170-et teljesített, ami eddig az egyik legnagyobb érték. Továbbá az adatbázis rendszer több jellemzőjét is megemlítem az következőkben:

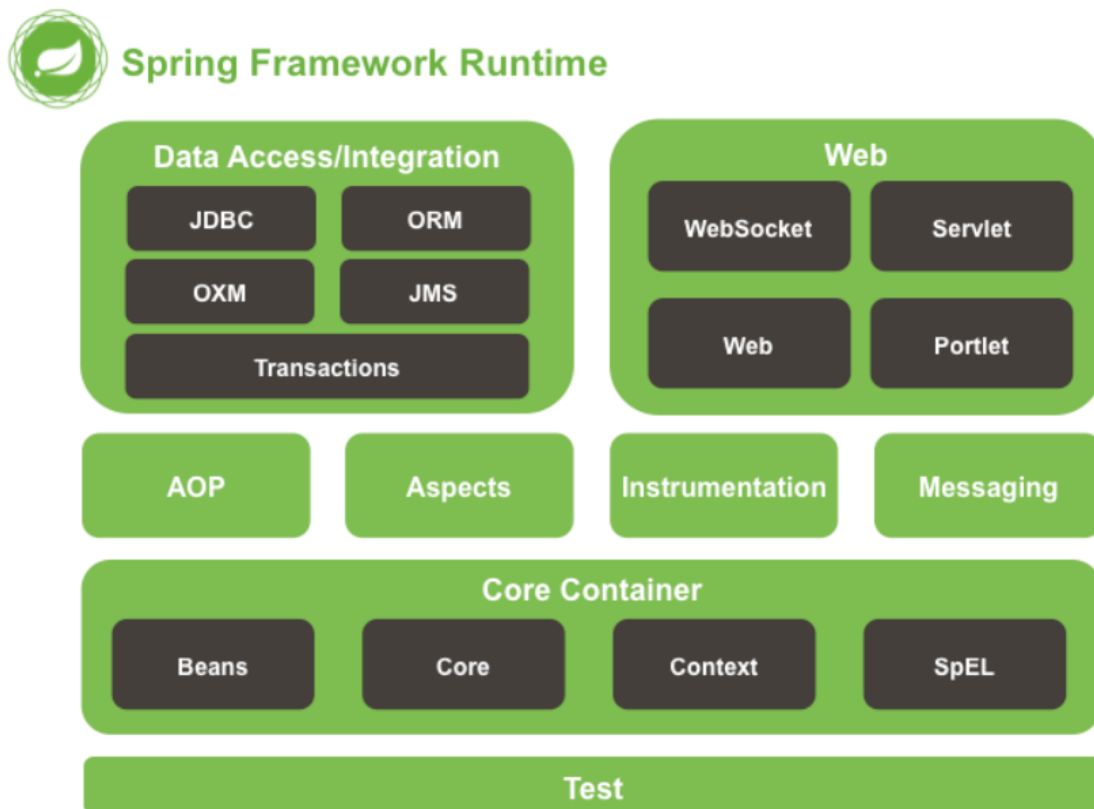
- **Adattípusok:** primitívek(numerikus, karakterlánc, logikai), strukturált(idő, tömb, tartomány, UUID), dokumentum(JSON, XML, kulcsérték), geometriai és testre szabható típusok.
- **Adatintegritás:** egyedi, nem null, elsődleges kulcsok, idegen kulcsok, kizárási korlátozások, explicit zárok és tanácsadó zárok.
- **Teljesítmény:** : indexelés, tranzakciók, olvasási lekérdezések, táblázat partícionálás, skálázhatóság.
- **Megbízhatóság:** naplózás, replikáció.
- **Biztonság:** hitelesítés, robosztus beléptető rendszer.
- **Rugalmasság:** tárolt eljárások és függvények(PL/pgSQL nyelv, Python, Java, JavaScript).
- **Nemzetközi karakterkészletek támogatása és szöveges keresés**

---

<sup>4</sup>ACID: Atomicity (atomiság), Consistency (konzisztencia), Isolation (izoláció), és Durability (tartósság)

### 3.3.2. Spring Boot

A **Spring Boot** egy Java alapú keretrendszer, ami sokkal egyszerűbbé és gyorsabbá teszi a programozást. A Spring Boot a Springből jött létre 2014-ben. A Springnek az alapköveit Rod Johnson tette le 2002-ben. A Spring Boot az egyik legtöbbször használt keretrendszer a gyakorlatban, mert biztosítja a POJO<sup>5</sup> konfigurációs környezetet és a POJOk tárolását. A keretrendszer abban különbözik az osztálykönyvtártól, hogy alkalmazza a vezérlés megfordítása vagy más néven az Inversion of Control (IoC) elvét. Tomcat szerveret használ és a hozzá tartozó függőségeket a Maven projektépítő eszköz menedzseli. A keretrendszer nagyon megkönnyíti a benne való fejlesztést kezdő fejlesztőknek is.[5]



3.4. ábra. Spring Framework Runtime [5]

#### 3.3.2.1. Tomcat

A Tomcat szervernek két fontos tulajdonsága van az egyik az, hogy kompatibilis a webfejlesztéssel a másik pedig, hogy automatikusan konfigurálható, egyszerűen telepíthetőek a war vagy jar fájlok. A spring-boot-starter-tomcat függőség hozzáadása során a következő lehetőségeket biztosítja a szerver: core, el, logging és a WebSocketek.[6]

---

<sup>5</sup>POJO: "Plain Old Java Object"



### 3.3.2.2. Maven

Segít a Spring Boot keretrendszernek a pom.xml<sup>6</sup> fájlban lévő függőségek megfelelő verziójának használatában, biztosítja az alkalmazás elindítását és a futtatható jar vagy war fájlok összeomagolását.

### 3.3.2.3. Inversion of Control

Az Inversion of Control (IoC) konténer egy folyamat, amely során meghatározza egy objektumnak a függőségeit de nem hozza létre azokat. Ezen konténerből az objektum bármikor lekérheti a függőségeit. Az IoC akkor a leghasznosabb, amikor nagyon sok osztályból áll az alkalmazás és ezek között függőségek vannak.<sup>[7]</sup>

### 3.3.2.4. Bean

A Spring alkalmazás alapjait alkotó objektumok, amelyeket a Spring IoC konténer kezel, komponensnek nevezik. A bean vagy más néven bab egy Spring IoC konténer által példányosított, összeállított és egyéb módon kezelt objektum.<sup>[7]</sup>

### 3.3.2.5. Proxy

A Proxy tervezési minta segít az aspektus-orientált programozás kivitelezésében. A következő funkciók ezzel a tervezési mintával vannak megvalósítva Spring Boot-ban: naplózás, teljesítményfigyelés, gyorsítótár és tranzakciókezelés.

## 3.3.3. Android

Az **Android** egy operációs rendszer, amely szintén nyílt forráskódú de az alapja egy Linux kernel. Elsősorban az érintőképernyős eszközökre fejlesztették ki, mint például az okostelefon és a táblagép. Az operációs rendszer felépítésének rétegeit a következő [3.5](#) ábra szemlélteti.

### 3.3.3.1. Android SDK

Az Android SDK <sup>[9]</sup> rövidítése nem más mint az Android Software Development Kit, amit a Google fejlesztett ki, hogy olyan könyvtárakat és szoftverfejlesztői eszközöket tartalmazzon, amelyek szükségesek az Android alkalmazások fejlesztéséhez. Az Android SDK több összetevőből áll: eszköz építők, emulátor, platform eszközök és eszközök.

### 3.3.3.2. Android alkalmazás

Az Android alkalmazások <sup>[10]</sup> fejlesztése lehetséges Java vagy Kotlin nyelvben is, ahogy a projektben is történt. Az imént említett Android SDK eszközök csomagolják össze a kódot az adat és az erőforrásfájlokkal együtt, a fájlnak .apk kiterjesztése lesz, amelyet az Android eszközökre telepítünk. Minden alkalmazás egy külön virtuális gépen (VM) fut, mivel hogy az Android rendszer a legkisebb jogosultság elven van. Egy Androidos alkalmazásnak négy összetevője lehet:

---

<sup>6</sup> pom.xml: Project Object Model



3.5. ábra. Android Architektúra [8]

- **Tevékenységek:** az a belépési pont ahol a felhasználó interakcióba kerül az alkalmazással.
- **Szolgáltatások:** ez egy olyan összetevő, amely a háttérben fut ha a műveletek hosszas ideig tartanak.
- **Broadcast:** lehetővé teszi, hogy az alkalmazás használatán kívül eseményt továbbítson a felhasználónak
- **Tartalomszolgáltatók:** az adatok szolgáltatását biztosítja akár egy adatbázisból, fájlból vagy egy webhelyről.

### 3.3.3.3. Manifests

Az Androidos rendszer az összetevőket deklarálnia kell az AndroidManifest.xml fájlban az elindítása előtt. Ez a fájl felelős az alkalmazás futtatásához szükséges felhasználói engedélyek megadásáért, mint például az internet elérés. Szintén ebben a fájlban kell megadni az alkalmazásnak az API szintet és a külső API-könyvtárakat, amelyeket majd

hozza kell kötni az alkalmazáshoz. De itt kell megadni azt is ha például a kamerát vagy a Bluetooth szolgáltatásokra van szükség. [10]

#### **3.3.3.4. Gradle**

Androidban a Gradle[11] projektépítő rendszert használtam, amely egy modern bővítményekre épülő rendszer. A Gradle kezeli az Android alkalmazásban a függőségeket és létrehozza a már fentebb is említett apk fájlokat.

## 4. fejezet

# A rendszer specifikációja

### 4.1. Felhasználói követelmények

Az alfejezetben megtalálható néhány fontos használati esetet és azoknak a részletes leírása. A 4.1 ábrán megtekinthető az összes használati eset, amelyeket az alkalmazás biztosít a felhasználónak. A felhasználó vagy más néven az aktor kétféle lehet, vagy mentor, vagy mentorált. Bizonyos funkciókat csak a mentor, vagy csak a mentorált érhet el melyeknek a leírásában ezt is feltüntettem majd.

#### 4.1.1. Fontosabb használati esetek

- **tevékenységek megtekintése**

- *előfeltételek:*

- \* PR1: a felhasználó be van jelentkezve
- \* PR2: a felhasználóhoz tartozik legalább egy feladat vagy csoport meghívás

- *follyamat:*

- \* F1: a felhasználó bejelentkezik
- \* F2: a felhasználó vár amíg betöltődnek az adatok
- \* F1: a felhasználó felfele görgetéssel megtekintheti a többi tevékenységet

- *alternatív folyamat:*

- \* AF1: ha a felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik

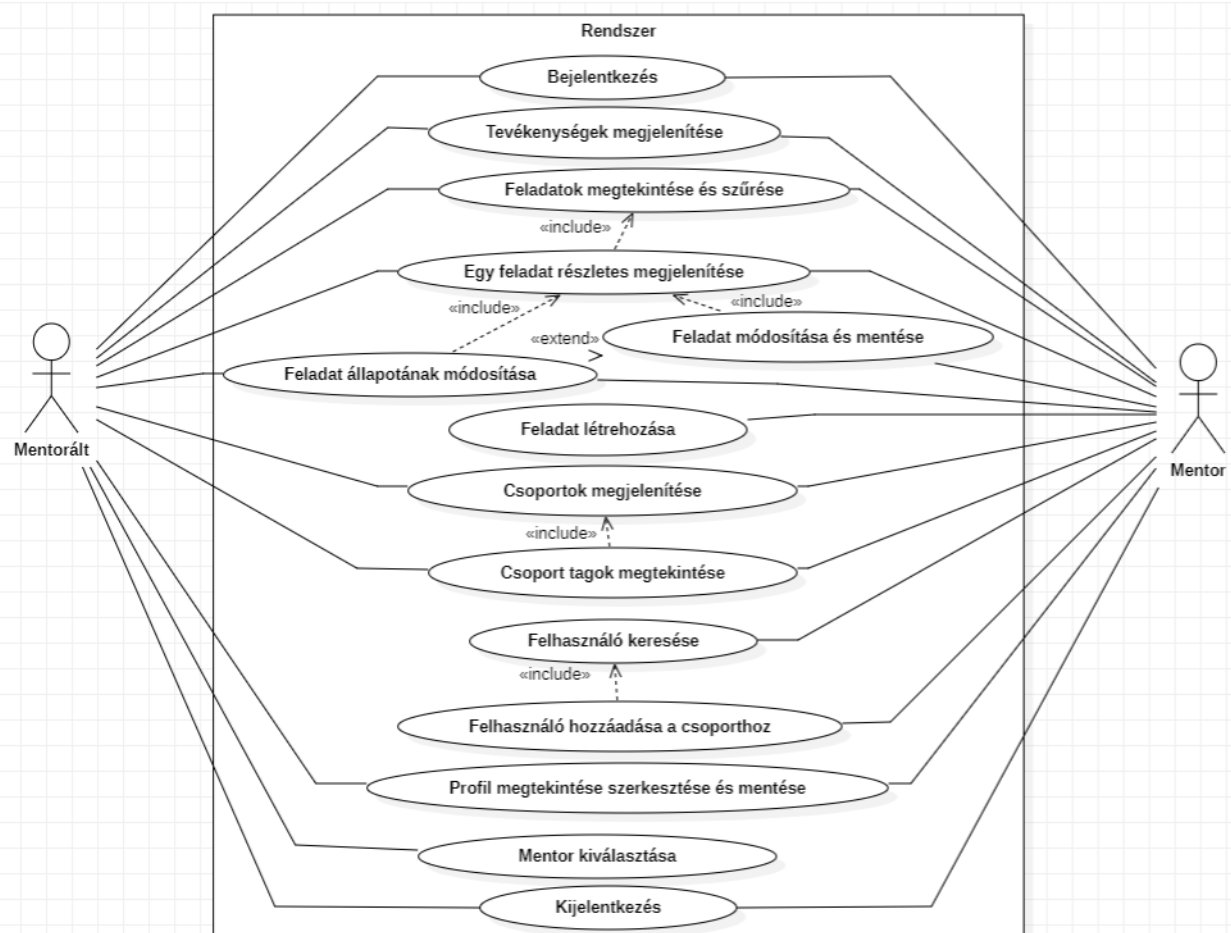
- **feladatok megtekintése**

- *előfeltételek:*

- \* PR1: a felhasználó be van jelentkezve
- \* PR2: a felhasználónak van legalább egy feladata

- *follyamat:*

- \* F1: a felhasználó bejelentkezik
- \* F2: a felhasználó kiválasztja az alsó menüben balról a második ikont



4.1. ábra. Használati eset diagram

- \* F3: a felhasználó görgetéssel tud megtekinteni további feladatokat
- \* F4: a felhasználó kiválasztja az feladatok fölött valamelyik címkét szűrhet bizonyos feladatokra
- *alternatív folyamat:*
  - \* AF1: ha a felhasználó már be van jelentkezve, akkor a folyamat az F2-től kezdődik
- *utófeltételek:*
  - \* PO1: megjelenik a feladatok listája
- **új feladat létrehozása**
  - *előfeltételek:*
    - \* PR1: a mentor be van jelentkezve
  - *folyamat:*
    - \* F1: a mentor bejelentkezik
    - \* F2: a mentor kiválasztja az alsó menüben balról a második ikont

- \* F3: mentor a jobb felső sarokban kiválasztja a plusz ikont
- \* F4: a mentor kiválasztja az új feladat projektjét
- \* F4: a mentor megadja az új feladat nevét
- \* F4: a mentor kiválasztja az új feladat végrehatóját
- \* F4: a mentor megadja az új feladat lejárati idejét
- \* F4: a mentor megadja az új feladat prioritását
- \* F4: a mentor megadja az új feladat leírását
- \* F5: a mentor megérinti a "Create" feliratú mentés gombot
- *alternatív folyamat:*
  - \* AF1: ha a mentor már be van jelentkezve, akkor a folyamat az F2-től kezdődik
  - \* AF2: ha a mentor már kiválasztotta az alsó menüben balról a második ikont, akkor a folyamat az F3-tól kezdődik
- *utófeltételek:*
  - \* PO1: a feladatot elküldi a szerveroldalnak majd bekerül az adatbázisba ha megfelelően vannak megadva az adatok
  - \* PO3: "The new task is successfully created!" toast megjelenítése
- *hibák:*
  - \* E1: "Task data is missing!" toast megjelenítése ha hiányosak az adatok
  - \* E2: szerverről visszakapott hiba üzenetek

- **mentor kiválasztása**

- *előfeltételek:*
  - \* PR1: a mentorált be van jelentkezve
  - \* PR2: a mentorálnak még nincs mentora
- *folyamat:*
  - \* F1: a felhasználó bejelentkezik
  - \* F2: a mentorált megnyomja a menüben jobb szélső ikont
  - \* F2: a mentorált megnyomja a "view profile" címkét
  - \* F3: a mentorált megnyomja a "Select mentor" gombot
  - \* F4: a mentorált kiválasztja a megjelenő nemtorok egyikét
- *utófeltételek:*
  - \* PO1: a mentorálnak megjelenik a mentor neve a profil oldalán
  - \* PO1: a mentorálnak többet nem lesz látható a "Select mentor" gomb

## 4.2. Rendszerkövetelmények

### 4.2.1. Funkcionális követelmények

A fejezet során rávilágítok néhány fontosabb funkcionalitásra, amelyeket biztosítani tud az alkalmazás.

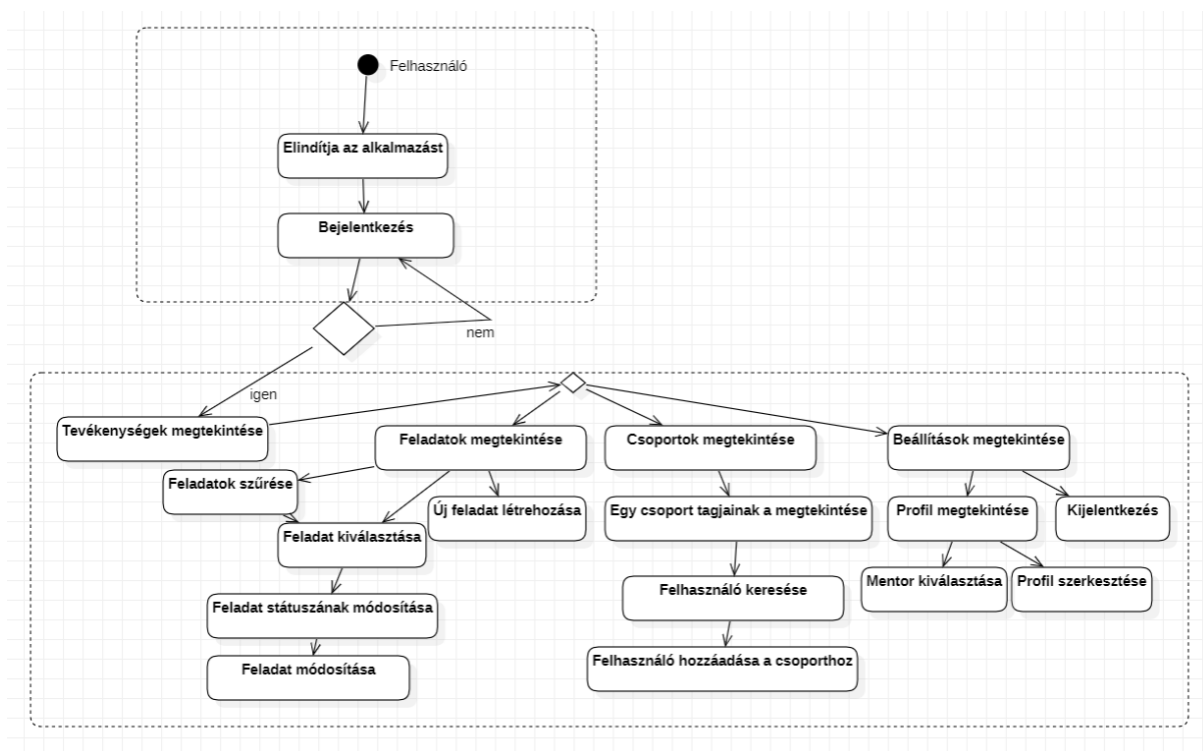
- **Bejelentkezés:** az applikáció elindítása során a felhasználót egy splash screen vagy is kezdőképernyő fogadja, amely után rögtön megjelenik a bejelentkezési oldal. Az oldalon a továbblépés érdekében szükséges megadnia egy email címet és a hozzátartozó jelszót. Ha mindkét mezőt kitöltötte és a “Login” gombot megérintette a felhasználó akkor az adatok helyességétől függően megjelenik az Activities vagy is a tevékenységek oldal és az alsó menü.
- **Tevékenységek megtekintése:** itt a felhasználó megtekintheti a hozzá kapcsolódó feladatokat és csoport meghívásokat.
- **Feladatok megtekintése:** az alsó menüpontból választhatja ki a balról a második ikont. Ezen az oldalon megjelenik kezdetben a felhasználó összes feladata prioritási sorrendben, amelyeket görgetéssel tekinthet meg. Továbbá lehetőség van a fejléc alatti címkék alapján a feladatok szűrésére attól függően, hogy milyen státusban vannak a feladatok, amelyekre kíváncsi a felhasználó.
- **Feladat részletes megtekintése:** a feladatok közül kiválaszthat egyet a felhasználó és részletesen megtekintheti annak adatait egy következő oldalon. A oldalon módosíthatja a feladat állapotát. Ha a felhasználó egy mentor akkor megjelenik a fejlécben egy ceruza ikon, amit ha megérint akkor lehetősége lesz a feladat szerkesztésére.
- **Feladat szerkesztése:** az oldalon módosítható a feladat címe, leírását, az hogy melyik projektbe tartozzon, hogy ki oldja meg a feladatot és megváltoztatható a határidő is. A változtatások végen az “Update” gombra kell nyomnia a mentornak, ahhoz hogy a módosításokat elküldje a szervernek.
- **Új feladat létrehozása:** a feladatok megtekintése oldalon a fejlécben megjelenik egy plusz ikon a jobb felső sarokban ha egy mentor van bejelentkezve. Az ikonra kattintva egy másik oldalra navigálja a mentort, ahol meg kell adni a feladat projekt besorolását, a címét a leírását azt, hogy ki oldja meg a feladatot és, hogy mikor van a lejárat dátum, milyen prioritású a feladat. Ha minden mezőt kitöltött vagy választott a mentor és rányom a “Create ” gombra akkor az új feladat adatait elküldi a szerveroldalnak. Ezek után a fejlécben egy nyíl segítségével vissza léphet az összes feladatra.
- **Csoportok megtekintése** Ha egy mentor van bejelentkezve akkor az összes csoport neve megjelenik ezen a menüpontban, de ha egy mentorált akkor csak azon csoportok nevei jelennek meg, amelyekhez hozzáadta már valaki. A csoportnevek mellett van egy-egy nyíl, amelyre ha a felhasználó rákattint akkor megtekintheti a csoporttagokat.
- **Felhasználó hozzáadása egy csoporthoz:** a csoporttagok oldalról tud átjutni erre az oldalra a mentor egy plusz ikon használatával. Itt rákereshet egy felhasználóra a neve alapján. A keresés eredményeként megjelenített felhasználók közül ha rányom az egyikre akkor hozzáadja a csoporthoz azt a felhasználót.
- **Beállítások megtekintése:** ezen a menüpontban a felhasználó kijelentkezhet a “Log out” gomb segítségével vagy megtekintheti a saját profilját a “view profile”-ra kattintva, amely átnavigálja egy következő oldalra.

- **Profil szerkesztése:** a profil oldalon a felhasználó kiválaszthat egy új képet a galériából a profilképének, módosíthatja a nevét, vagy megadhatja a lakhelyét és telefonszámát. De annak érdekében, hogy ezek mind elmentésre kerüljenek a “Save modifier” gombot kell megérintenie. Továbbá a mentorálnak ezen az oldalon megjelenik egy gomb, amely a “Select mentor” feliratot tartalmazza és arra szolgál, hogy egy következő oldalon kiválassza a mentorát, akinek a neve megjelenik a profil oldalon a visszalépés után.

#### 4.2.2. Nem funkcionális követelmények

- **Termék követelmények**
  - *Felhasználhatóság:* a termék bármilyen Android-os telefonon vagy táblagépen futtatható, amelyik rendelkezik minimum 21-es android SDK-val és legkisebb 12.1 Snow Cone v2 android verzióval
  - *Hatékonyság:* az alkalmazás megfelelő internetkapcsolattal bármikor kiszolgálja a felhasználót nagyon rövid időn belül
  - *Megbízhatóság:*
    - \* fejlesztés során a legtöbb hibalehetőséget igyekeztem kiszűrni.
    - \* a megfelelő használat alatt, nem jelenik meg hibaüzenet
  - *Hordozhatóság:* az applikáció előzetes telepítést igényel
  - *Biztonság:*
    - \* az alkalmazás token alapú bejelentkezést és azonosítást használ
- **Folyamat követelmények**
  - *Kódolási standard*
    - \* a változó- és függvénynevek esetén CamelCase kódolás
    - \* a tördelés tabulátorral való megoldása
  - *Verziókövetés*
    - \* GitHub [12] verziókövető rendszer használata minden fontossabb funkcionális befejezése után
- **Külső követelmények**
  - *Összeférhetőségi:* az alkalmazás semmilyen képpen nem tudja befolyásolni az eszközön lévő többi alkalmazást .
  - *Etikai:* az alkalmazás nem szolgáltatja ki másféle szolgáltatásnak az adatokat.
  - *Együttműködési:* A felhasználó a bejelentkezés során személyes adatokat kell megadnia, amelyeket a szerver megfelelően titkosít a tároláshoz.





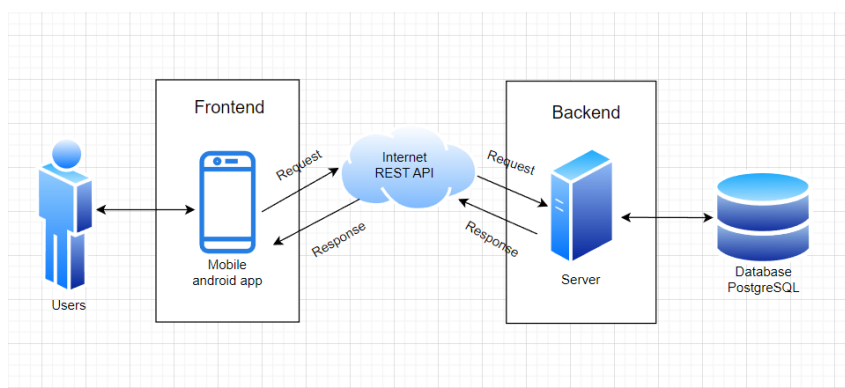
4.2. ábra. Aktivitás diagram

## 5. fejezet

# Tervezés

Ebben a fejezetben részletesen kifejttem a rendszer architektúráját és ismertetem a kliensoldali applikáció felépítését.

### 5.1. A rendszer architektúrája



5.1. ábra. Diagram a rendszer architektúrájáról

A dolgozatban kivitelezett rendszer három fő komponensből áll: egy kliensoldal vagyis angol megnevezésén frontend, szerver oldal a backend és adatbázis.

- *adatbázis*: célja az adatok tárolása és a szerver oldallal való kommunikáció, elhelyezkedésében pedig a legalsó rétegben található.
- *szerveroldal*: a REST API-k kiszolgálása, az üzleti logika megvalósítása és az adatok lekérése, beszúrása az adatbázisba a legfőbb feladatai.
- *kliensoldal*: mobilon applikáción keresztül grafikus felületet biztosít a felhasználónak és a szerverrel kommunikál

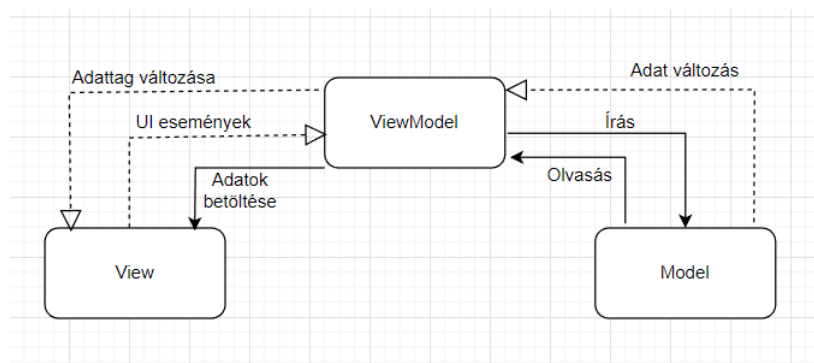
### 5.2. Modell-Nézet-NézetModell

A **Model-View-ViewModel (MVVM)** architektúra egy szoftvertervezési minta, ami szétválasztja a grafikus felhasználói felületet az alkalmazás üzleti logikájától. Erre

az architektúra mintára épül a projekt kliensoldali komponense. A minta alap elveit Ken Cooper és Ted Peters Microsoft szoftver tervezők fektették le. Az MVVM három kulcsfontosságú összetevőből áll:

- *Model*: az alkalmazás modelljét képviseli, valamint tartalmaz üzleti és érvényesítési logikát, csak a ViewModelmel kommunikál, a View-t nem ismeri.
- *View*: az alkalmazás felhasználói felületét és annak vizuális viselkedését valósítja meg, adatkötéseken keresztül kommunikál a ViewModellel.
- *ViewModel*: egy kapcsolat a Model és a View között, a beérkező bemenetet fogadja és továbbítja a Modellnek.

Az Modell-Nézet-NézetModell architektúra a három fő alkotóelem együttműködésében rejlik. Minden interakció a felhasználói felületen a View-n belül történik, így ezen interakciók (egér kattintás, gomb nyomás) során adatkötés történik ViewModel között. A ViewModel azokat az eseményeket valósítja meg amelyekhez a View hozzákötethető. A események meghatározzák, hogy a View milyen funkcionalitást tud adni a felhasználónak. A ViewModel egy másik feladata hogy a Model osztályokból származó adatokat átadja a Viewnek.[13].



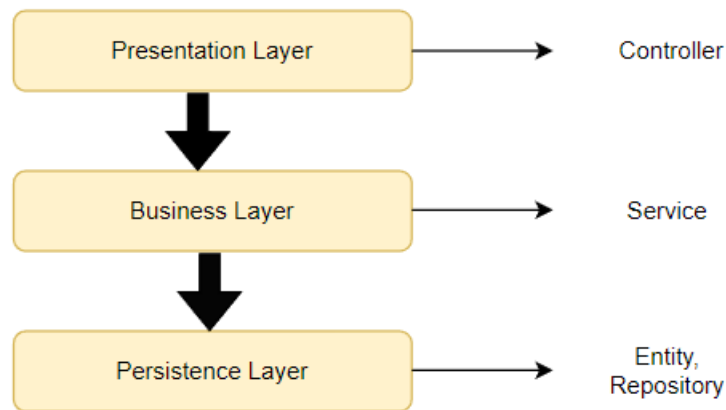
5.2. ábra. MVVM diagram

### 5.3. Háromrétegű architektúra

A háromrétegű architektúra[14] az egyik leggyakrabban használt szoftvertervezési architektúra minta. A háromrétegű architektúrában a rétegek vízszintesen helyezkednek el, minden egyes réteg jól meghatározott feladatot hajt végre és ami a legfontosabb egyik réteg sem függ a másiktól. Részletesebben, ha együtt vannak a hasonló komponensek akkor lehetővé válik a típusainak szétválasztása és egy helyen lesznek összegyűjtve a hasonló program kódok is. Ebben az architektúrában három réteg van, fentről lefele ezek a következők:

- **Prezentációs réteg (Presentation layer):** itt kezelődnek a HTTP kérések és a JSON formátumban kapott tartalmakat objektumokká való alakítása.

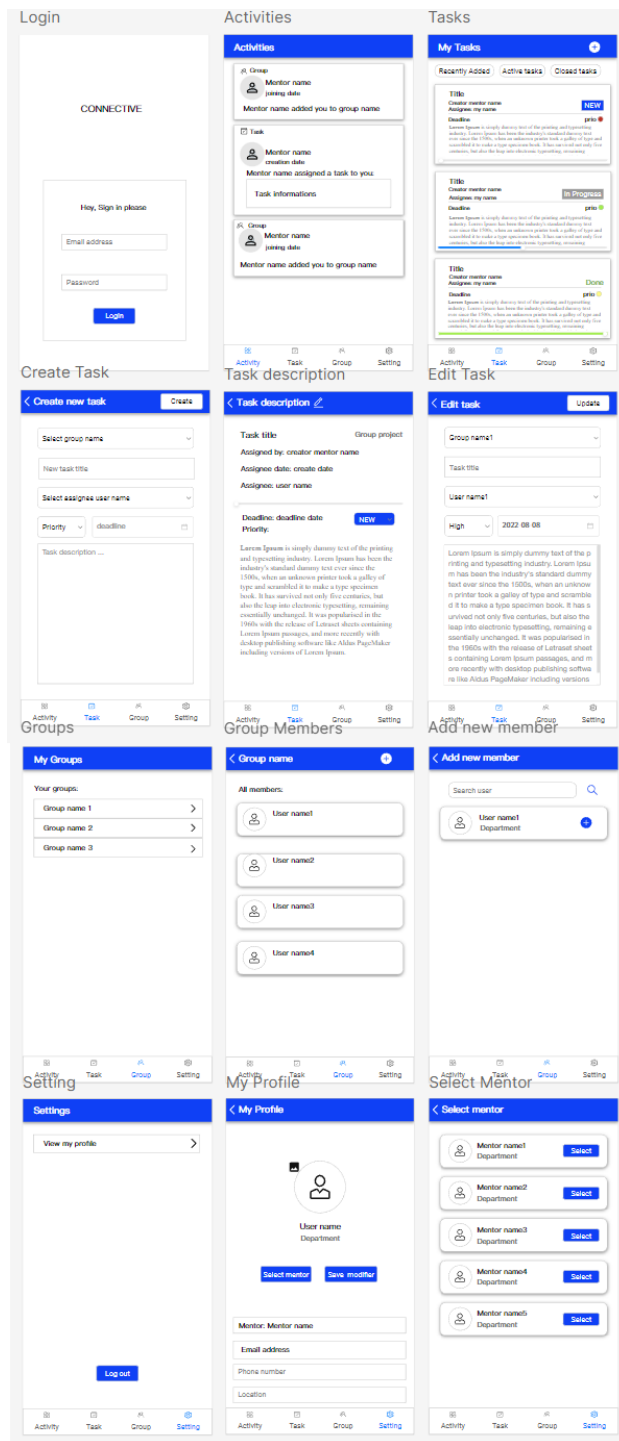
- **Üzleti réteg (Business Layer):** szolgáltatás (service) osztályokból tevődik össze, a rendszer üzleti logikáját tartalmazza.
- **Perzisztencia réteg (Persistence Layer):** az objektumokat alakítja adatbázis sorokra és fordítva adatbázis sorokat objektumokká.



**5.3. ábra.** Háromrétegű architektúra diagram

## 5.4. Felhasználói interfész tervezése

A felhasználói felület drótváza már adott volt mivel, hogy a projekt ötlete az Android tantárgy keretén belül született meg. De az idő elteltével a kisebb nagyobb változások miatt a projekt új drótvázát hoztam létre. A drótváz egy struktúra, amely egy vizuális képet mutat projekt grafikus felületéről, ezzel megkönnyítve a projekt megértését. (5.4 ábra).



5.4. ábra. Felhasználói felület drótváza

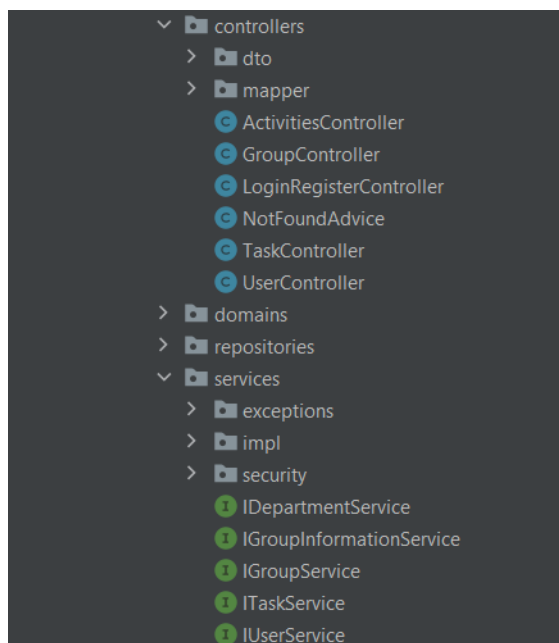
## 6. fejezet

# Kivitelezés

Ebben a fejezetben részletesen bemutatom a projekt kivitelezését. Elsősorban szeretném kihangsúlyozni, hogy az alkalmazás két felhasználói szerepkörrel rendelkezik, az egyik szerepkör a mentor és a másik a mentorált vagy másképpen a gyakornok. Ezek alapján ismertetve, hogy milyen funkciókat képes betölteni az alkalmazás attól függően, hogy melyik felhasználó típus van bejelentkezve.

### 6.1. Szerveroldali megvalósítás

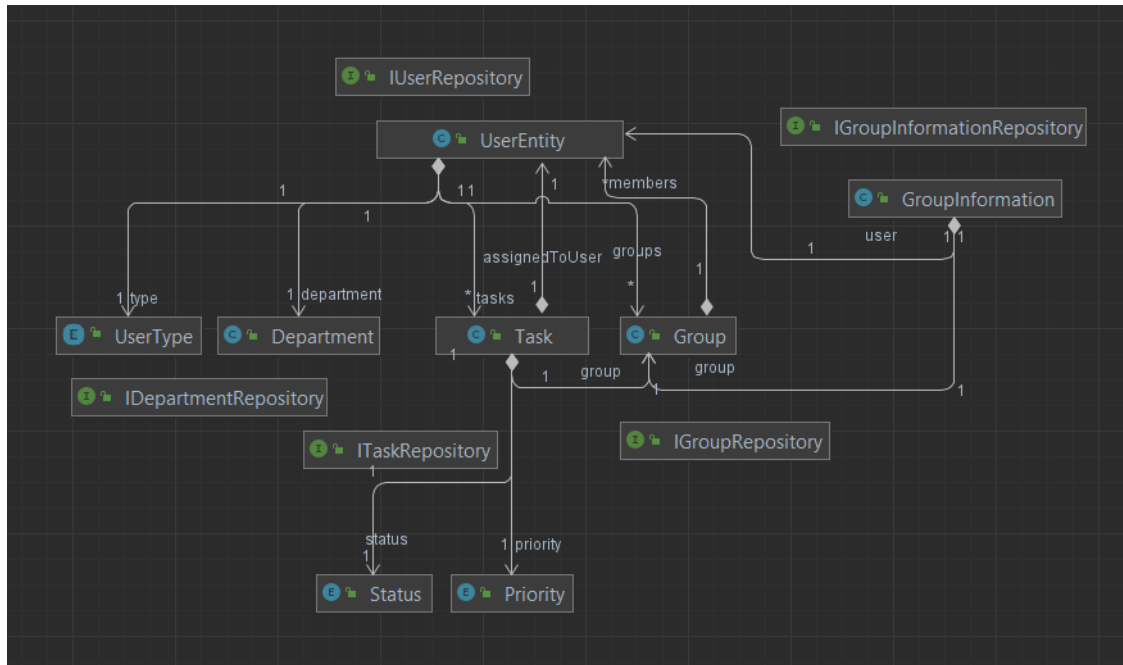
A projekt szerver részében található mappákat (6.1 ábra) a réteges architektúra szempontjai alapján alakítottam ki Spring Boot keretrendszerben. Minden hasonló funkcionális ellátó komponenst ugyanazon mappába helyeztem, ezáltal nem függenek egymástól a jövőbeni fejlesztés esetén és így könnyebbé teszi a tesztelést is.



6.1. ábra. A szerver mappaszerkezete

### 6.1.1. Adatelérési réteg

Ebben a részben kifejtem a serveroldalon megjelenő adatmodelleket és azok kapcsolatait, ahogy ezt a (6.2 ábra) osztály diagram is mutatja. A réteg továbbá azt a célt szolgálja, hogy az adatbázis és az adatmodellek kapcsolatát biztosítja, a megfelelő adat lekérésekért, ezért felelősek a repository interfészek.



6.2. ábra. Adatmodell osztály diagram

#### 6.1.1.1. Adatmodellek

A projekt alapvető adatmodelljei az 6.1 ábrában a domains mappában találhatóak. Az adatmodellek vagy más néven entitások[15] olyan objektumok, amelyek az adatbázis rekordjait képviselik a backenden ha Entity és Table annotációkkal látjuk el az osztályt. A projekt entitásai a következők:

- *UserEntity*: Tartalmazza a felhasználók személyes adatait, a feladataikat, azt hogy milyen csoportoknak a tagja és többek között azt is, hogy milyen típusú felhasználó, mentor vagy mentorált.
- *Task*: A adatmodell reprezentálja a feladat címét, leírását, azt hogy ki hozta létre és mikor, hogy kinek van szánva a feladat, meddig van a határidő, mi az állapota és a prioritása.
- *Group*: Minden csoport rendelkezik egy névvel és tagokkal.
- *GroupInformation*: Az adatmodell a csoportokhoz adott új felhasználót, a meghívási időt, meg a meghívó mentort foglalja magába.
- *Department*: A felhasználók munkaköreinek adatait tartalmazza.

Megjegyzendő, hogy minden entitásnak van egy azonosítója Id annotációval, amely a relációs adatbázisban a primary key lesz, természetesen minden azonosító egyedi a SequenceGenerator és a GeneratedValue annotációk használatával. Több entitás között van tartalmazási kapcsolat ezt pedig az adatbázisnak OneToOne, OneToMany, ManyToOne vagy ManyToMany kulcsszavak használatával jelezzük a JPA<sup>1</sup>-n keresztül. A következő 6.1 kódrészlet a csoport adatmodellt ábrázolja az említett annotációkkal:

```
@Entity
@Table(name = "groups")
public class Group {
    @Id
    @SequenceGenerator(name = "group_id_gen", sequenceName = "group_id_seq",
        initialValue = 1)
    @GeneratedValue(generator = "group_id_gen")
    private Long id;

    @Column(name = "group_name")
    @NotEmpty(message = "Group name is mandatory")
    private String groupName;

    @ManyToMany()
    @JoinTable(name = "group_user",
        joinColumns = {@JoinColumn(name = "group_id")},
        inverseJoinColumns = {@JoinColumn(name = "user_id")})
    private Set<UserEntity> members = new HashSet<>();
    //setter and getter
}
```

### 6.1. kódrészlet. Adatmodell osztály

#### 6.1.1.2. Konfigurációk

A projekt kezdetén code first elven hoztam létre a táblákat PostgreSQL-ben. A Spring Boot szerveren az application.properties fájlban, amely különféle tulajdonságok konfigurálására szolgál [16]. Ebben a fájlban adhatjuk meg az adatbázis elérését, a bejelentkezési adatokat és azt, hogy a szerver újra indulása során frissüljön vagy jöjjön ismét létre az adatbázis(6.2 kódrészlet).

```
spring.datasource.url=jdbc:postgresql://localhost:5432/mentormentee
spring.datasource.username=postgres
spring.datasource.password=12345password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

### 6.2. kódrészlet. Adatbázis konfigurációk

<sup>1</sup>JPA: Java Persistence API, az osztályok és az adatbázis közötti kommunikációt biztosítja.



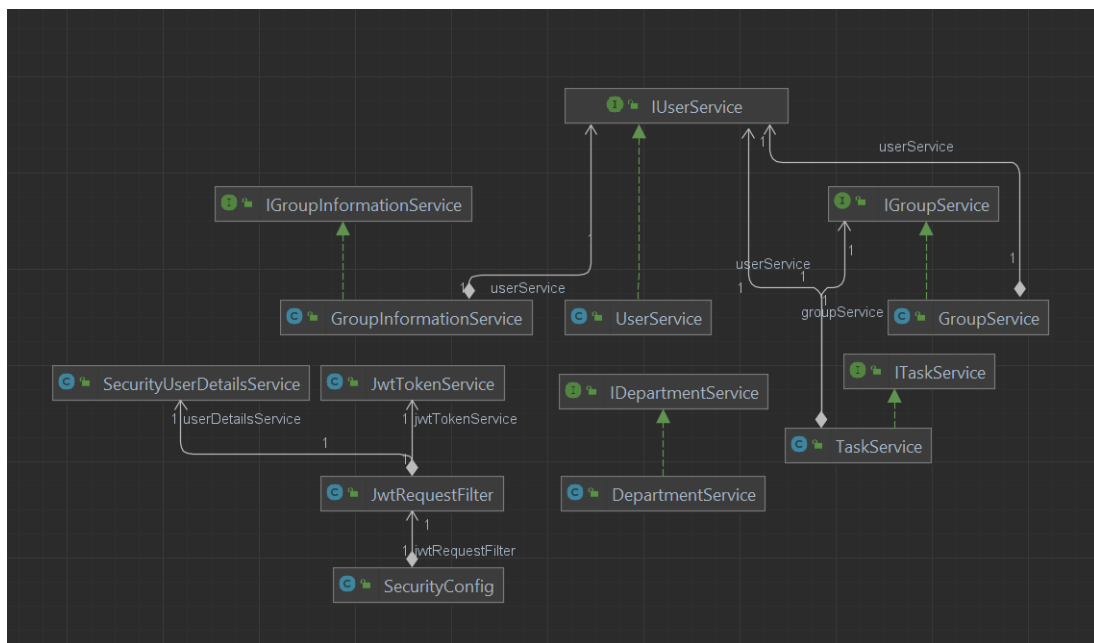
Ahogy már említettem az adatbázissal a JPA a kapcsolat az objektumok és a relációs adatbázis között. Az egyik legfőbb jellemzője, hogy dinamikusan generál lekérdezést a metódus alapján, amelyeket a Repositoryban használtam [17]. Erre egy példa a 6.3 kódrészlet, ahol a `findAllByAssignedToUserId` visszatéríti az összes olyan feladatot, amelyet a megadott id-val rendelkező felhasználóhoz rendeltek:

```
@Repository
public interface ITaskRepository extends JpaRepository<Task, Long> {
    List<Task> findAllByAssignedToUserId(Long assignedToUserId);
    List<Task> findAll();
}
```

### 6.3. kódrészlet. Dinamikus lekérdezések

#### 6.1.2. Üzleti logika réteg

A réteg legfőbb feladata, hogy a szolgáltatásokat egy úgynevezett service osztályokba csoportosítva megalapozzák az adatok megfelelő kezelését. Az adatok kezelése alatt a következők érthetők: az adat ellenőrzése, különböző műveletek és folyamatok vezérlése, adatelérés az adatbázisból, hiba és kivételkezelés. A 6.3 ábra reprezentálja a szervicek osztály diagramját.



6.3. ábra. Service osztály diagram

##### 6.1.2.1. Biztonsági intézkedések

A szerver végpontok és a kliensoldal között csak token hitelesítés alapján lehet biztonságosan információkat megosztani, ebben segít a JWT(JSON Web Token). A JWT

egy nyílt szabvány, amely egy megbízható HMAC algoritmussal írja alá digitálisan az adatot [18].

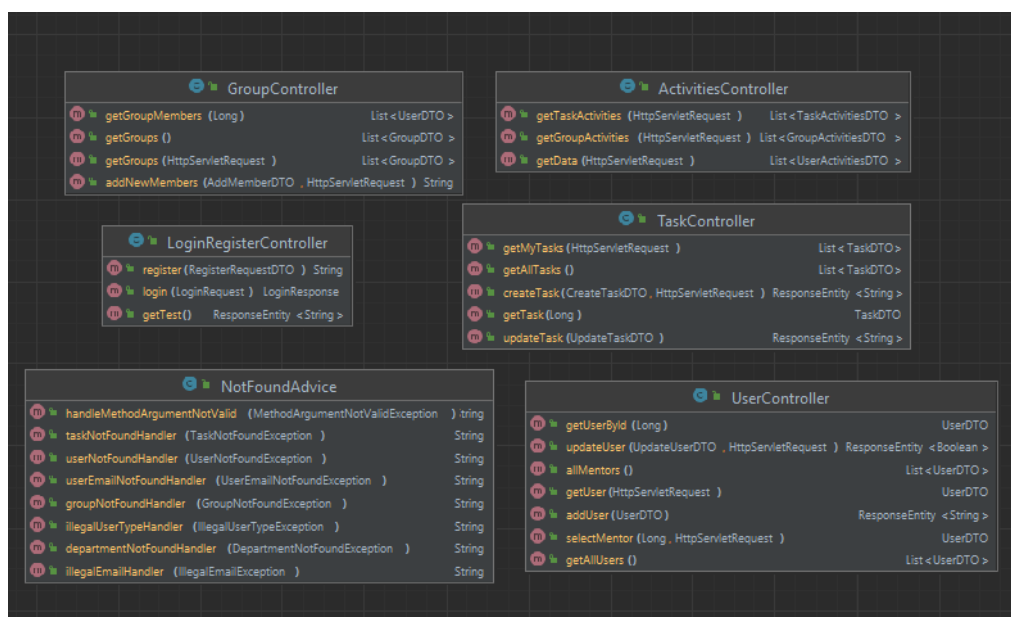
```
public String generateToken(final UserDetails userDetails) {
    return JWT.create()
        .withSubject(userDetails.getUsername())
        .withExpiresAt(new Date(System.currentTimeMillis() + jwtTokenValidity))
        .sign(this.hmac512);
}
```

## 6.4. kódrészlet. Token generálás

A bejelentkezés során egy egyedi e-mail cím és egy jelszó szükséges. A Spring Security biztosítja a PasswordEncoder [19] segítségével a jelszavak titkosítását, így mentve el az adatbázisba.

### 6.1.3. Prezentációs réteg

Ebben a rétegben a szerver végpontjait találhatók melyeket el tud érni a kliens oldal. Annak érdekében, hogy a szervertől átláthatóbb legyen a minden végpontnak egy-egy függvény felel meg, amelyeket controller osztályokba csoportosítottam. A controller osztályokat a következő (6.3 ábra) diagram alapján szemléltetem.



6.4. ábra. Controller osztály diagram

#### 6.1.3.1. API végpontok

A szervertől több API végpont is rendelkezésre áll a kliensoldali részére. A szervertől alap URL-e a következő **http://localhost:8080/api/v1**, amely két publikus, bárki számára elérhető hitelesítés nélküli API-t biztosít, az egyik a regisztráció, a másik pedig a bejelentkezés.

Ezt követően minden végpont eléréséhez szükséges a hitelesítés egy token alapján, amit a felhasználó sikeres bejelentkezés után kap meg válaszként. A továbbiakban (6.1, 6.2, 6.3 táblázatok) részletesen kifejtetem néhány fontos API-ra vonatkozó információt a paramétereikről és a válaszokról. Az API-k kontrollerekben vannak csoportosítva, annak megfelelően, hogy felhasználó, feladat, csoport, aktivitás vagy bejelentkezés specifikusak. Azt amint már fentebb is említettem, minden kérés és válasz tartalma JSON vagy egyszerű szöveg formátumban történik. A JSON alakban átadott adatokat a szerver egy-egy DTO(Data Transfer Objects)[20] formátumú objektummá alakítja annak érdekében, hogy több paramétert kössön össze egy API híváson során.

**6.1. táblázat.** Felhasználók bejelentkezése

<b>Végpont</b>	/public/login
<b>Metódus</b>	POST
<b>Leírás</b>	Ez a végpont bejelentkezést biztosít a felhasználók számára.
<b>Paraméterek</b>	"email": "felhasznalo@email.com", "password": "jelszo00"
<b>Válasz</b>	Egy token, amely szükséges majd a későbbi hitelesítéshez, és egy type, ami a felhasználó típusa lehet MENTOR vagy MENTEE

**6.2. táblázat.** Felhasználó feladatainak a lekérése

<b>Végpont</b>	/tasks/allTasks
<b>Metódus</b>	GET
<b>Leírás</b>	A végpont visszaadja a bejelentkezett felhasználó összes feladatát.
<b>Paraméterek</b>	A kérés során meg kell adni a HTTP fejlécben [21] a hitelesítési tokent.
<b>Válasz</b>	Egy olyan listát ad vissza, amely egy-egy feladatról a következő információkat tartalmazza: taskId, title, description, assignedToUserId, assignedToUserName, creatorUserId, creatorUserName, createTime, groupId, groupName, priority, deadline, status, progress

## 6.2. Kliensoldali megvalósítás

A kliensoldal vagy felhasználói felület egy Androidos alkalmazás keretein belül van kivitelezve, így a használata előtt telepíteni kell az applikációt. A kliensoldal felépítése az MVVM architektúrára alapszik és a szerverrel való kommunikáció aszinkron API hívásokkal van megoldva.

### 6.2.1. Állapotmenedzsment

Az alkalmazás két interaktív képernyőből vagy más néven aktivitásból tevődik össze. Az első aktivitás tartalmazza az animációval ellátott kezdőképernyőt. A második aktivitás pedig tartalmazza a fő funkciókat.

### 6.3. táblázat. Feladat létrehozás

Végpont	/tasks/createTask
Metódus	POST
Leírás	A feladat létrehozást csak egy mentor hajthatja végre de bármelyik felhasználót hozzárendelheti a feladathoz.
Paraméterek	A kérés során meg kell adni a HTTP fejlécben a hitelesítési tokent és a bodyban [22] JSON formában a következőket: title, description, assignedToUserId, assignedToUserName, creatorUserId, groupId, priority, deadline, status
Válasz	Sikeres válasz esetén: "Task creation is successful" Sikertelennél pedig: "Task creation is unsuccessful!" vagy egyéb hibák ha nem megfelelőek a paraméterek.

Az alkalmazás használatához mindenképp bejelentkezés szükséges. A belépés ideje alatt, hogy elérhetőek legyenek bizonyos felhasználó specifikus adatok *Preferences* API segítségével kulcs-érték páronként vannak elmentve a mobiltelefon memóriájába. Az adatok csakis kizárólag primitív típusúak<sup>2</sup> lehetnek így mentődnek el egy olyan XML<sup>3</sup> fájlban, ahol a fájl tartalmát nem képes más alkalmazás elérni. A következőkben azt reprezentálja a 6.5. kódrészlet, hogy a tokent, a felhasználó típusát és e-mail címét hogyan lehet privát módon kezelni:

```
val preferences = requireActivity().getPreferences(Context.MODE_PRIVATE)
val edit = preferences.edit()
edit.putString("token", MyApplication.token)
edit.putString("userType", MyApplication.userType)
edit.putString("email", emailEditText.text.toString())
edit.apply()
```

### 6.5. kódrészlet. Példa a preferences használatára

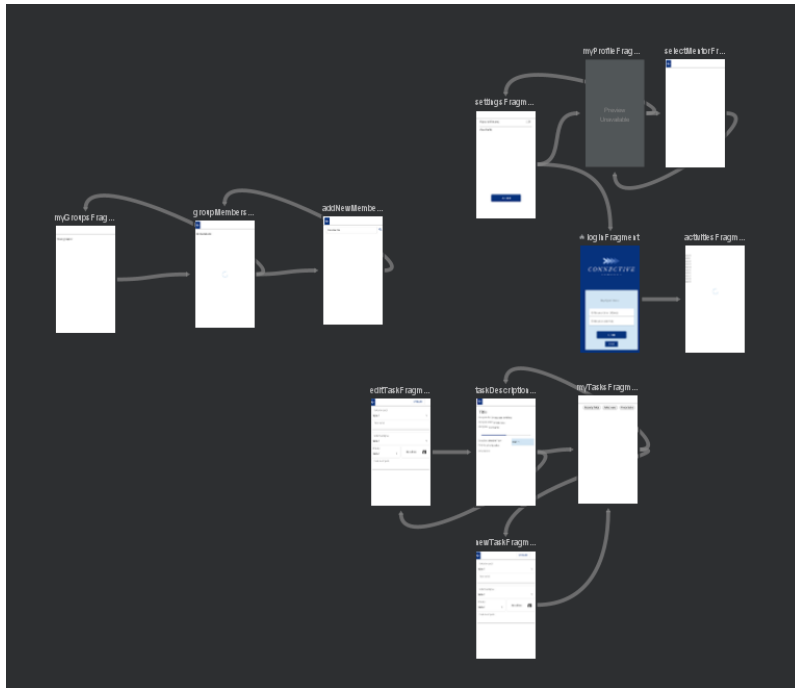
Az alkalmazásban a második aktivitás tartalmaz több oldalt vagy is fragmentet. Minden aktivitásnak és fragmentnek van egy-egy életciklusa, így annak érdekében, hogy az adatok ne vesszenek el egy fragment életciklusa során *ViewModel*Providert használtam. A *ViewModel*Provider ha nem létezik akkor létrehoz egy új *ViewModel*-t vagy visszaadja a már meglévőt.

#### 6.2.2. Navigáció

Az applikációban lévő fragmentek között a navigációt egy alul elhelyezkedésű menüvel és egy navigációs gráf (6.5 ábra). használatával valósítottam meg. A navigációs gráf nem más mint egy XML állomány, amely tartalmazza az összes navigációval kapcsolatos információt. Az állomány átöleli az összes fragmentet valamint a közöttük lévő lehetséges útvonalakat, amely által a felhasználó eljuthat alkalmazás különböző oldalaira.

<sup>2</sup>Primitív típusok: int, double, string, boolean

<sup>3</sup>A HTML-hez hasonló azzal a különbséggel, hogy nincsenek előre definiált címkék



6.5. ábra. Navigációs gráf

A menü használata (6.6. kódrészlet) lehetőséget biztosít olyan főoldalak közötti navigációra, amelyek amúgy nem feltétlenül kell rendelkezzenek a navigációs gráfban útvonallal.

```
//menu navigation
val navController = findNavController( R.id.myNavHostFragment)
val bottomNav =
    findViewById<BottomNavigationView>(R.id.bottomNavigation)
val toolbar = findViewById<Toolbar>(R.id.toolbar)
bottomNav.setOnNavigationItemSelectedListener{ it ->
    when (it.itemId) {
        R.id.activitiesFragment -> {
            Navigation.findNavController(this,R.id.myNavHostFragment)
                .navigate(R.id.activitiesFragment)
        }
        R.id.myTasksFragment -> {
            Navigation.findNavController(this,R.id.myNavHostFragment)
                .navigate(R.id.myTasksFragment)
        }
        R.id.myGroupsFragment -> {
            Navigation.findNavController(this,R.id.myNavHostFragment)
                .navigate(R.id.myGroupsFragment)
        }
        R.id.settingsFragment ->{
            Navigation.findNavController(this,R.id.myNavHostFragment)
                .navigate(R.id.settingsFragment)
        }
    }
}
```

```

        else -> super.onOptionsItemSelected(it)
    }
    it.isChecked=true
    true
}>

```

6.6. kódrészlet. Példa alul elhelyezkedő menüre

### 6.2.3. Szerveroldallal való kommunikáció

A kliensoldal REST API-k segítségével kommunikál a szerveroldallal, így a használt protokoll HTTP. A kliens és a szerver oldal közötti adatátvitel JSON<sup>4</sup> formában történik, maga az adatátvitel a Retrofit[23] 2 könyvtárcsomag segítségével történik. A Retrofit képes a kapcsolatok létrehozására, a HTTP API-n keresztül kapott válasz Kotlin által értelmezett kódra fordítani. Az alábbi 6.7 kódrészlet bemutatja a Retrofit példányosítást Singleton tervezési minta alkalmazásával:

```

object RetrofitInstance {
    var mHttpLoggingInterceptor = HttpLoggingInterceptor()
        .setLevel(HttpLoggingInterceptor.Level.BODY)

    private var mOkHttpClient = OkHttpClient
        .Builder()
        .addInterceptor(mHttpLoggingInterceptor)
        .addInterceptor{chain ->
            val originalRequest = chain.request()
            val requestBuilder = originalRequest.newBuilder()
                .header("Authorization", "Bearer ${MyApplication.token}")

            val newRequest = requestBuilder.build()
            chain.proceed(newRequest)}
        .build()

    var mRetrofit: Retrofit? = null

    var gson: Gson = GsonBuilder()
        .setLenient()
        .create()
    val client: Retrofit?
    get() {
        if(mRetrofit == null){
            mRetrofit = Retrofit.Builder()
                .baseUrl(Constants.BASE_URL)
                .client(mOkHttpClient)
                .addConverterFactory(ScalarsConverterFactory.create())
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build()
        }
    }
}

```

<sup>4</sup>JSON: JavaScript Object Notation

```

    }
    return mRetrofit
}
}

```

### 6.7. kódrészlet. Retrofit létrehozása

A retrofit példány által használt szolgáltatásokat egyetlen interfészben tároltam. Az interfészben *suspend* kulcs szóval vannak ellátva a függvények, ami arra utal, hogy aszinkron függvények tehát korutinkok vagy más suspend metódusokból hívhatóak meg. Az interfészben használt függvények előtt meg kell adni a HTTP metódust és a server elérési útvonalát. A szervertől visszaérkező válasz típusát a függvény paraméterei után lehet beállítani, hogy a JSON-t majd arra a típusra alakítja át a **Gson** [24] könyvtárcsomag. Erre a 6.8 kódrészletben látható példa.

```

interface TrackerApi {
    @POST(Constants.LOGIN_URL)
    suspend fun login(@Body request: LoginRequest): Response<LoginResponse>
    ...
}

```

### 6.8. kódrészlet. Suspend függvény

## 6.2.4. Korutinkok

Kotlinban az aszinkron programozást a korutinkok[25] segítségével lehet kivitelezni. A korutinkok biztosítják a jobb felhasználói élményt mivel, hogy a ViewModelben ha elküldünk egy hálózati kérést akkor a művelet egy háttérzálon fog futni és így nem fagy le az alkalmazás amíg megérkezik a válasz. A választ egy LiveData adattárolóba helyezem, ennél fogva már a fragmentben csak a adat változását kell figyelnie a fragment. A korutinkok létezését a CoroutineScope határozza meg, amely ha már nem létezik akkor a korutink is leáll.

```

class LoginViewModel(private val repository: UserRepository) : ViewModel() {
    var loginResult: MutableLiveData<LoginResult> = MutableLiveData()
    fun login(request: LoginRequest) {
        viewModelScope.launch {
            try {
                val response = repository.login(request)
                ...
            } catch (e: Exception) {
                ...
            }
        }
    }
}

```

### 6.9. kódrészlet. suspend típusú függvények hívása

A ViewModel példány a fragmentben a Factory tervezői minta alapján jön létre. A függvényeit pedig egy *Repository*-n<sup>5</sup> keresztül hívódnak meg azért, hogy a magasabb szintű modulok ne függjenek az alacsonyabbaktól, így megvalósítva a dependency inversion elvet.

## 6.3. Az alkalmazás működése

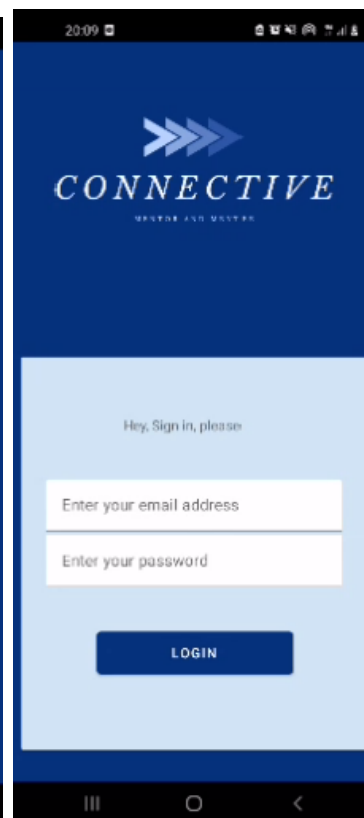
Ebben a fejezetben részletes bemutatásra kerül a rendszer, mindkét felhasználó szemszögéből.

### 6.3.1. Kezdőoldal

Az alkalmazás indításakor megjelenik egy kezdőoldal vagy más néven splash screen (6.6 ábra) és alkalmazás logóján egy animációs művelet megy végbe, majd ezt követően a bejelentkező oldal jön elő. A bejelentkezés során a felhasználónak meg kell adnia az e-mail címét és a jelszót (6.7 ábra). Akár sikeres volt a bejelentkezés, akár nem a felhasználó egy Toast<sup>6</sup> üzenetben értesül.



6.6. ábra Kezdőoldal



6.7. ábra Bejelentkezési oldal

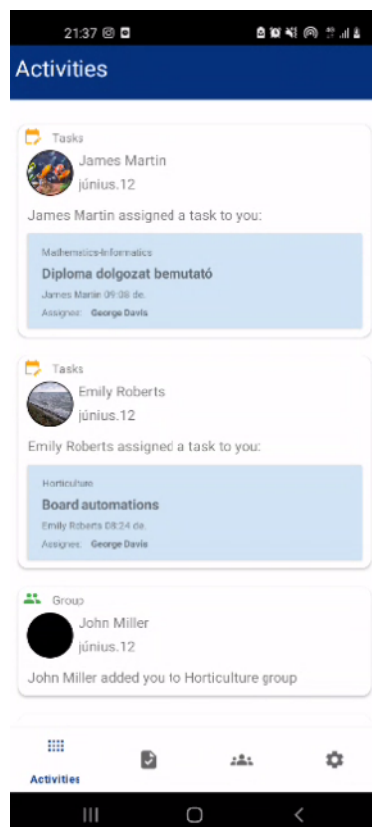
<sup>5</sup>Repository: fő szerepe az adatréteg absztrakciója és a tartományi objektumok kezelése

<sup>6</sup>Toast: egyszerű felugró szöveges visszajelzés.



### 6.3.2. Aktivitások - Activities

A sikeresen bejelentkezett felhasználókat erre az oldalra navigálja az alkalmazás, ahol már a teljes alkalmazásban elérhető egy alsó menü. Maga az oldalon minden felhasználó, tehát mentor vagy mentorált megtekintheti a saját aktivitásait vagy is a feladatait és a csoporthoz való hozzáadásokat (6.8 ábra). Az aktivitások betöltése alatt egy folyamatjelző úgynevezett progress bar jelenik meg az adatok betöltéséig. A tevékenységek fordított időrendi sorrendben vannak elhelyezve, ezzel is növelve a felhasználói élményt.



6.8. ábra. Aktivitások oldal

### 6.3.3. Feladatok - My Tasks

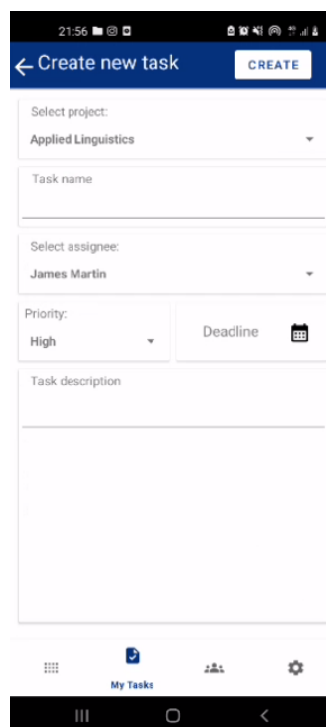
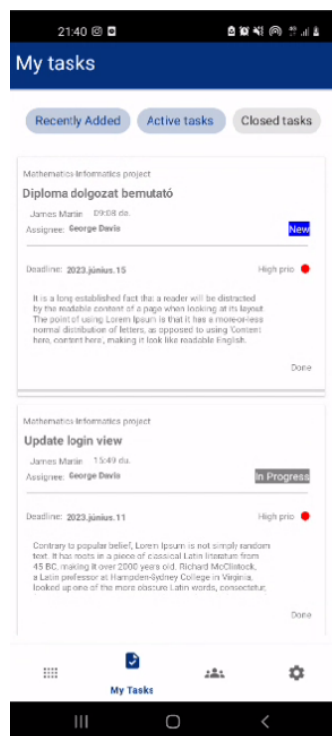
Ezen oldalon a mentor vagy mentorált megtekintheti az összes feladatát (6.9 ábra), amelyek még soron következnek, folyamatban vannak, befejeződtek vagy éppen blokkolt állapotban van. A feladatok állapota szerint lehet szűrőket beállítani. Ha a felhasználó részletesebben meg szeretné tekinteni a feladatát erre is lehetősége van, egy feladat kiválasztásával aminek hatására át navigál egy új oldalra (6.11 ábra).

A Task Description oldal keretén belül nem csak a feladat részletei tekinthetőek meg, hanem a felhasználó megváltoztathatja a feladat állapotát egy lenyíló lista használatával. Visszalépve az összes feladat oldalra a kiválasztott task állapota az új állapotra változott, mivel a módosítást elküldi az alkalmazás a szerveroldalnak.

A My Tasks cím mellett ha mentor van bejelentkezve megjelenik egy plusz ikon, az ikonra kattintva a mentor létrehozhat egy új feladatot (6.10 ábra). A feladatnak ki kell

választani, hogy melyik csoportba tartozik, szükséges beírni a címét és a leírását, meg kell adni, hogy ki végezze el a feladatot és mikor van a határidő, továbbá a feladat prioritását is be kell állítani. Mikor minden információt helyesen megadott a mentor akkor a Create gombra kattintva az alkalmazás elküldi az új feladatot a szervernek.

A feladat teljes szerkesztésére (6.12 ábra) ismét csak a mentornak van rá lehetősége, az oldal nagyon hasonló az új feladat létrehozás oldalra azzal a különbséggel, hogy az oldal betöltésekor a kiválasztott task adatait jeleníti meg a mezőkben.



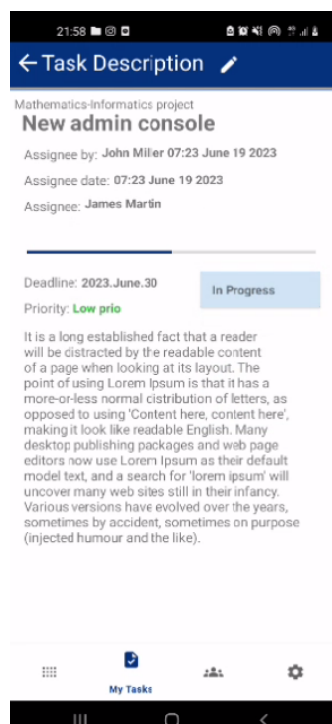
6.9. ábra Felhasználó feladatai 6.10. ábra Új feladat létrehozási oldala

#### 6.3.4. Csoportok - My Groups

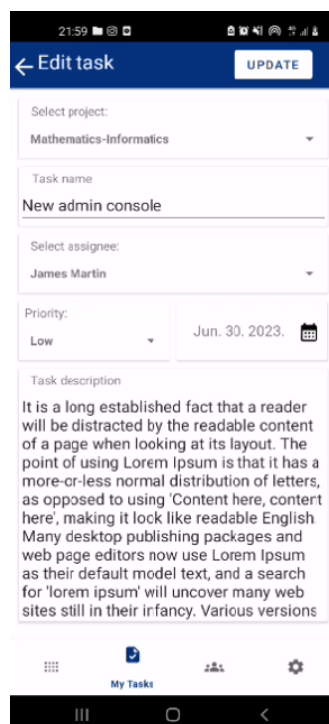
A csoportok menüpontban elérhető oldal (6.13 ábra) szintén annak függvényében jeleníti meg a csoportokat, hogy milyen felhasználó van bejelentkezve. Ha mentorált akkor megjelennek azon csoportok nevei, amelyeknek már tagja a mentorált. Ha pedig egy mentor típusú felhasználó jelentkezett be akkor az összes csoport neve megjelenik az oldalon. Mindkét esetben lehetőség van egy csoport tagjainak a megtekintésére ha a felhasználó rákattint a csoportnév utáni nyílra. A csoport tagjait megjelenítő oldalon (6.14 ábra) a mentornak láthatóvá válik új tag hozzáadására alkalmas ikon. Az ikon segítségével egy olyan oldalra (6.15 ábra) navigálja a felhasználót, ahol név alapján kereshet személyeket, akiket hozzá szeretne adni a csoporthoz.

#### 6.3.5. Beállítások - Settings

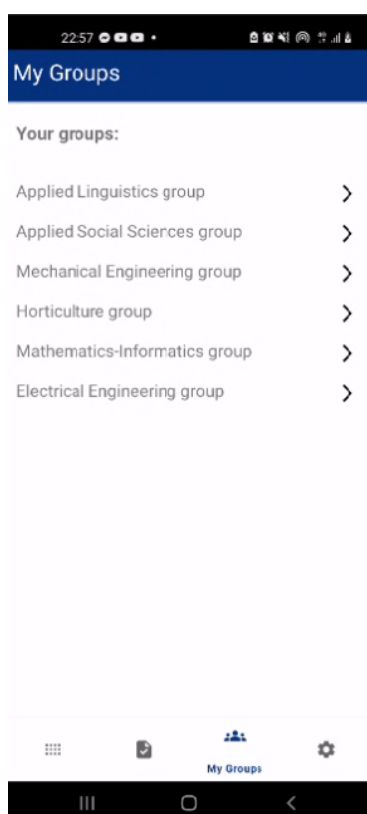
A beállítások oldalon (6.16 ábra) a felhasználó átnavigálhat a saját profil (6.17 ábra) oldalára, amelyen nem csak megtekintheti az adatait, hanem szerkesztheti is azokat. Ha



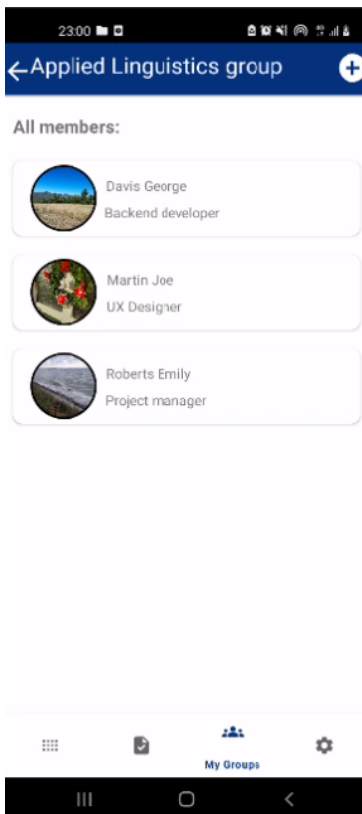
6.11. ábra Feladat részletei



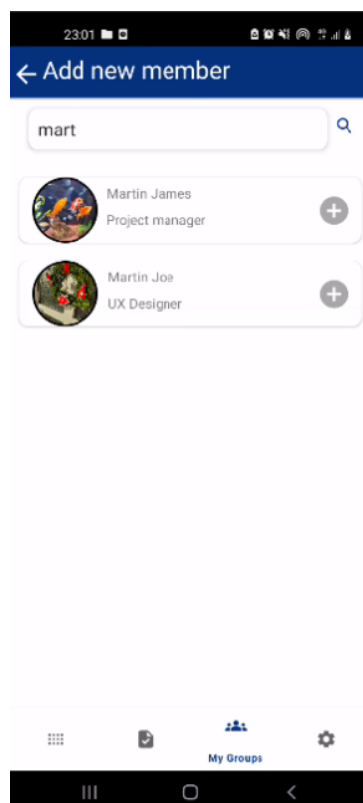
6.12. ábra Feladat szerkesztése



6.13. ábra Csoportok oldal



6.14. ábra Csoport tagok oldal

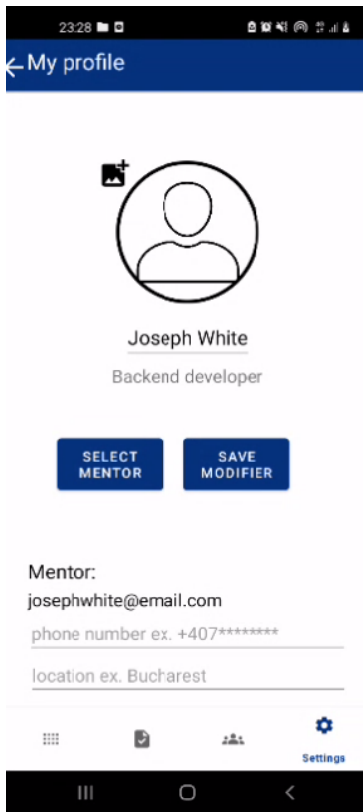


6.15. ábra Új tag hozzáadása a csoporthoz

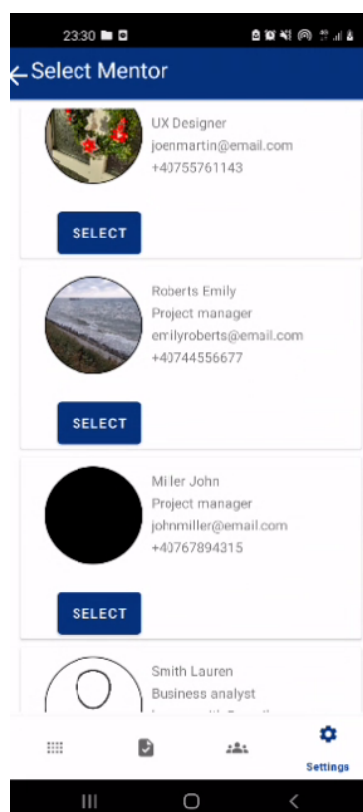
egy olyan felhasználó van először bejelentkezve, aki mentorált és még nincs mentora, akkor egy mentorokat tartalmazó oldalon (6.18 ábra) az összes mentor közül kiválaszthat egy személyt. A beállításoknál lehetősége van a felhasználónak a kijelentkezésre.



6.16. ábra Beállítások



6.17. ábra Saját profil



6.18. ábra Mentor kiválasztása

### 6.3.6. Hibák és visszajelzések

A felhasználó mindig kap egy Toast[26] visszajelzést a szerveroldali kommunikáció sikerességéről. A Toast egy lebegő nézet, amely egy kis üzenetet tartalmaz a felhasználó számára. Az alkalmazásban a beviteli mezők tartalmát már a kliensoldalon ellenőrizöm néhány reguláris kifejezés segítségével.

## 7. fejezet

# Eszközök és munkamódszerek

### 7.1. Integrált fejlesztői környezet (IDE)

Az projekt fejlesztése két külön álló integrált fejlesztői környezetben történt. A kliensoldal megvalósításához Android Studio-t [27] használtam, míg a szerveroldal fejlesztéséhez IntelliJ IDEA-t [28]. Mindkét fejlesztői környezetet a JetBrains fejlesztett ki.

Az IntelliJ IDEA-ban való fejlesztés a hatékonyabban kódolást segíti Java vagy akár Kotlin nyelvben. Továbbá biztosítja a Spring Boot keretrendszert és Mavent a projekt függőségeinek befecskendezéséhez, amelyeket alapkövei a projekt backend részének. IntelliJ IDEA-t használva lehetőség nyílik a kódkiegészítésre, a kódbázisban való navigációra, tesztelésre, probléma észlelésre és annak javításának opcióira is rávilágít.

Az Android Studio egy IntelliJ IDEA alapú fejlesztői környezet, amely az Android alkalmazások fejlesztésére specifikálódott. Ez a fejlesztői környezet Gradle projekt építő rendszert alkalmaz és az IntelliJ IDEA-hoz hasonlóan szintén támogatja az intelligens kódszerkesztést.

### 7.2. Verziókövetés eszközök

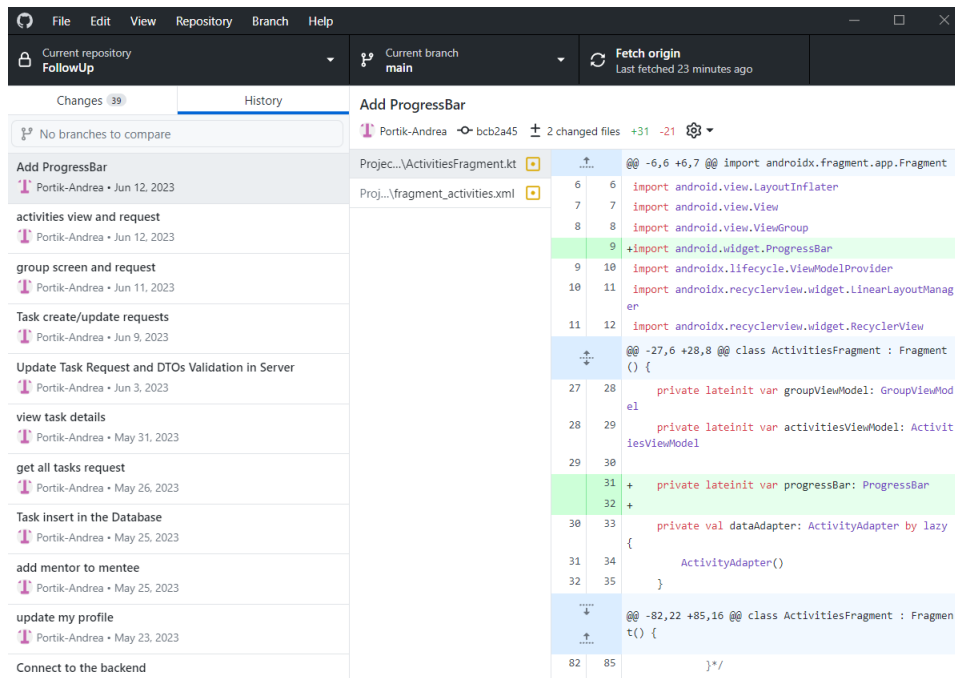
A verziókövetést megkönnyítő eszközként **Github Desktop** alkalmazást használtam (7.1 ábra). A Github segítségével biztosítottam a projekt biztonságos tárolását és a projekt esetleges verzió visszaállítását.

### 7.3. API tesztelés

A szerveroldali REST API-k tesztelésére **Postman**-t [29] alkalmaztam, amely egy API platform. A Postman használata leegyszerűsítette a HTTP kérések tárolását, a tesztesetek és eredmények megtekintését, így számos hibát sikerült kiküszöbölni a szerver oldalon.

### 7.4. Design eszközök

A projekt kezdetén adott volt egy Figmában készült design [30] az Android tárgy keretén belül, de az idő elteltével a kisebb nagyobb változások miatt egy új drótvá-



7.1. ábra. Github Desktop

zat hoztam létre. A drótváz megalkotásához Mockitt [31] használtam, amely egy online prototípus készítő. Az interaktív tervezés során bármilyen képernyőméretet és rengeteg animációt biztosít.

Publikus GitHub repository linkje: <https://github.com/Portik-Andrea/Connective>.

# Összefoglaló

## 7.5. Következtetések

Összeségében a projekt nagyvonalakban teljesíti a kitűzött feladatokat, így egy olyan felhasználói felület kommunikál az általam készített szerveroldallal, amely megkönnyíti a mentornak a feladatok kiosztását a mentoráltaknak. Az alkalmazás használatával a mentor bármikor létrehozhat egy új feladatot vagy csoporttag meghívást is elvégezhet, amelyekről az érintett felhasználó bejelentkezés után rögtön értesül. Minden felhasználó csak a saját feladatait látja és azok prioritásait, ennek eredményeképpen hatékonyabban kivitelezhető az időben elvégzett munka.

A szerveroldal fejlesztése során megvalósul az adatok megfelelő titkosítása tárolás előtt, itt legfőképpen a jelszavak titkosítására gondolok, ezzel is védve a kliens adatait. A szerver fejlesztésekor szem előtt tartottam a hibák megfelelő kezelését és a hozzájuk tartozó beszédes üzenetek visszajelzését a kliensoldalnak.

A bejelentkezést token használatával oldottam meg a kliensoldal és a szerveroldal közötti biztonságos adatcsere érdekében. A felhasználónak bármely művelet végrehajtása után megjelenik egy kis üzenetet, ha hiba történt akkor azt jeleníti meg röviden, ha pedig minden rendben történt akkor egy sikeres üzenete fog látni a felhasználó. Az üzenetek is biztosítják az alkalmazásnak jobb használhatóságát.

## 7.6. Továbbfejlesztési lehetőségek

Az projekt bizonyos körülmények között jól működik, de még nem tökéletes. Az alkalmazás alap funkciók befejezése során újabb továbbfejlesztési lehetőség került a látérbe, mint új funkciók vagy módosítások.

Az egyik ilyen funkció a regisztráció, amelyre már nem került sor az implementálás során. A regisztráció megvalósításával bárki könnyedén használhatja az alkalmazást egy cégen belül. Ennek tudatában egy másik továbbfejlesztési lehetőség is felmerül mégpedig az, hogy ne csak egy bizonyos munkacsoport tudja használni az alkalmazást, hanem bárki bármilyen cégen belül.

A későbbiekben szeretném megvalósítani azt, hogy a felhasználók a csoportban is tudják megtekinteni a csoporthoz kapcsolódó feladatokat. Ezzel növelni a felhasználók közötti információ megosztást, amihez akár egy chat is hozzá segíthet ha elakadása van a mentoráltaknak.

De a leghamarább kivitelezésre váró teendő az alkalmazás alapos tesztelése mind a kliensoldalon, mind pedig a szerveroldalon.

Publikus GitHub repository linkje: <https://github.com/Portik-Andrea/Connective>.



# Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Antal Margitnak a sok hasznos tanácsért, segítségért és a türelméért, amelyek nagyban hozzájárultak a diplomadolgozat létrejöttéhez. Külön köszönet, tanárainnak a sok jövedelmező tudásért, amit átadtak. És végül de nem utolsó sorban köszönöm szüleimnek a rengeteg támogatást.

# Ábrák jegyzéke

3.1. Together . . . . .	14
3.2. Microsoft To Do . . . . .	14
3.3. Slack . . . . .	14
3.4. Spring Framework Runtime [5] . . . . .	16
3.5. Android Architektúra [8] . . . . .	18
4.1. Használati eset diagram . . . . .	21
4.2. Aktivitás diagram . . . . .	25
5.1. Diagram a rendszer architektúrájáról . . . . .	26
5.2. MVVM diagram . . . . .	27
5.3. Háromrétegű architektúra diagram . . . . .	28
5.4. Felhasználói felület drótváza . . . . .	29
6.1. A szerver mappaszerkezete . . . . .	30
6.2. Adatmodell osztály diagram . . . . .	31
6.3. Service osztály diagram . . . . .	33
6.4. Controller osztály diagram . . . . .	34
6.5. Navigációs gráf . . . . .	37
6.6. Kezdőoldal . . . . .	40
6.7. Bejelentkezési oldal . . . . .	40
6.8. Aktivitások oldal . . . . .	41
6.9. Felhasználó feladatai . . . . .	42
6.10. Új feladat létrehozási oldala . . . . .	42
6.11. Feladat részletei . . . . .	43
6.12. Feladat szerkesztése . . . . .	43
6.13. Csoportok oldal . . . . .	43
6.14. Csoport tagok oldal . . . . .	43
6.15. Új tag hozzáadása a csoporthoz . . . . .	43
6.16. Beállítások . . . . .	44
6.17. Saját profil . . . . .	44
6.18. Mentor kiválasztása . . . . .	44
7.1. Github Desktop . . . . .	46

# Irodalomjegyzék

- [1] „Mentorálási alapfogalmak: Márton Gábor.” [https://eta.bibl.u-szeged.hu/5543/3/TM1\\_VI\\_Mentor%C3%A1ll%C3%A1s\\_A\\_M%C3%A1rton\\_G%C3%A1bor.pdf](https://eta.bibl.u-szeged.hu/5543/3/TM1_VI_Mentor%C3%A1ll%C3%A1s_A_M%C3%A1rton_G%C3%A1bor.pdf). (elérés dátuma: 2023. jún. 15.).
- [2] „Word “mentor” originates from homer *History Disclosure*.” <https://web.archive.org/web/20190620132531/https://www.historydisclosure.com/word-mentor-originates-homer/>. (elérés dátuma: 2023. jún. 15.).
- [3] „Mentorship *Wikipedia*.” [https://en.wikipedia.org/wiki/Mentorship#cite\\_note-11](https://en.wikipedia.org/wiki/Mentorship#cite_note-11). (elérés dátuma: 2023. jún. 15.).
- [4] „What is postgresql? *PostgreSQL*.” <https://www.postgresql.org/about/>. (elérés dátuma: 2023. jún. 15.).
- [5] „Overview of spring framework *Spring*.” <https://docs.spring.io/spring-framework/docs/5.0.0.RC3/spring-framework-reference/overview.html>. (elérés dátuma: 2023. jún. 16.).
- [6] „Difference between spring boot starter web and spring boot starter tomcat *GeeksforGeeks*.” <https://www.geeksforgeeks.org/difference-between-spring-boot-starter-web-and-spring-boot-starter-tomcat/>. (elérés dátuma: 2023. jún. 16.).
- [7] „What is a spring bean? *Baeldung*.” <https://www.baeldung.com/spring-bean>. (elérés dátuma: 2023. jún. 16.).
- [8] „Android-system-architecture.svg *Wikimedia*.” <https://commons.wikimedia.org/wiki/File:Android-System-Architecture.svg#>. (elérés dátuma: 2023. jún. 18.).
- [9] „Android sdk and it’s components *GeeksforGeeks*.” <https://www.geeksforgeeks.org/android-sdk-and-its-components/>. (elérés dátuma: 2023. jún. 18.).
- [10] „Application fundamentals *Android Developers*.” <https://developer.android.com/guide/components/fundamentals.html#Components>. (elérés dátuma: 2023. jún. 18.).
- [11] „What is gradle? *Gradle.org*.” [docs.gradle.org/current/userguide/what\\_is\\_gradle.html](https://docs.gradle.org/current/userguide/what_is_gradle.html). (elérés dátuma: 2023. jún. 21.).
- [12] „What is github? a beginner’s introduction to github.” <https://kinsta.com/knowledgebase/what-is-github/>. (elérés dátuma: 2023. jún. 25.).

- [13] „Mvvm (model view viewmodel) architecture pattern in android *GeeksforGeeks*.” <https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/>. (elérés dátuma: 2023. jún. 25.).
- [14] „Software architecture patterns — layered architecture.” <https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057>. (elérés dátuma: 2023. jún. 25.).
- [15] „Java persistence api architecture.” [https://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa\\_overview\\_arch.html](https://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa_overview_arch.html). (elérés dátuma: 2023. jún. 20.).
- [16] „Spring boot application properties *JavaTpoint*.” <https://www.javatpoint.com/spring-boot-properties>. (elérés dátuma: 2023. jún. 21.).
- [17] „Jpa features *JavaTpoint*.” <https://www.javatpoint.com/spring-boot-jpa>. (elérés dátuma: 2023. jún. 21.).
- [18] „Introduction to json web tokens *JWT*.” <https://jwt.io/introduction>. (elérés dátuma: 2023. jún. 21.).
- [19] „Spring security interview questions *InterviewBit*.” <https://www.interviewbit.com/spring-security-interview-questions/#passwordencoder>. (elérés dátuma: 2023. jún. 21.).
- [20] „The dto pattern (data transfer object) *Baeldung*.” <https://www.baeldung.com/java-dto-pattern>. (elérés dátuma: 2023. jún. 20.).
- [21] „Mdn web docs: Http headers. *Mozilla Developer*.” <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>. (elérés dátuma: 2023. jún. 20.).
- [22] „Http *Mozilla Developer*.” <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>. (elérés dátuma: 2023. jún. 20.).
- [23] „Introduction to retrofit *baeldung*.” <https://www.baeldung.com/retrofit>. (elérés dátuma: 2023. jún. 19.).
- [24] „Github - google/gson *Google*.” <https://github.com/google/gson>. (elérés dátuma: 2023. jún. 19.).
- [25] „Kotlin coroutines on android *Android Developers*.” <https://developer.android.com/kotlin/coroutines>. (elérés dátuma: 2023. jún. 19.).
- [26] „Toast *Android Developers*.” [developer.android.com/reference/android/widget/Toast](https://developer.android.com/reference/android/widget/Toast). (elérés dátuma: 2023. jún. 22.).
- [27] „Android studio: An ide built for android *Android Developers*.” <https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html>. (elérés dátuma: 2023. jún. 22.).

- [28] „What is intellij idea? *Jet Brains*.” <https://www.jetbrains.com/idea/features/>. (elérés dátuma: 2023. jún. 22.).
- [29] „What is postman?. *Postman*.” <https://www.postman.com/product/what-is-postman/>. (elérés dátuma: 2023. jún. 22.).
- [30] „Old design in figma. *Figma*.” <https://www.figma.com/file/o0FeSgcp0aCSR7HCbwpr8a/3Track?node-id=0%3A1>. (elérés dátuma: 2023. jún. 27.).
- [31] „Guide:product introduction *Mockitt*.” <https://mockitt.wondershare.com/guide/mockitt-product-introduction.html>. (elérés dátuma: 2023. jún. 23.).