Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék

**Szakdolgozat**

A képen szimbólum, Betűtípus, embléma, Grafika látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**Webes alkalmazás fejlesztése szakmai gyakorlati adatok nyilvántartásához**

**Készítette:**

Tóth Dávid Olivér, YIF399

Mérnökinformatikus BSc

Korszerű WEB-technológiák szakirány

**Tervezésvezető:**

Dr. Tompa Tamás

Miskolc

2025

**Tartalomjegyzék**

[1. Bevezetés 3](#_Toc198794114)

[1.1 Miért választottam ezt a feladatot? 3](#_Toc198794115)

[1.2 Az alkalmazás rövid bemutatása 3](#_Toc198794116)

[1.3. Problémafelvetés 4](#_Toc198794117)

[1.4. Felhasználók köre 5](#_Toc198794118)

[1.5. Elvárások a szoftverrel szemben 5](#_Toc198794119)

[2. A fejlesztési követelmények meghatározása 6](#_Toc198794120)

[2.1. Az alkalmazás helye és feladata 6](#_Toc198794121)

[2.2. Architektúra és struktúra modellek 6](#_Toc198794122)

[2.2.1. Monolitikus vagy mikroszolgáltatásos architektúra 6](#_Toc198794123)

[2.2.2. Three-tier (háromrétegű) alkalmazás 7](#_Toc198794124)

[2.3. Relációs adatbázis 7](#_Toc198794125)

[2.4. Back-end technológiák 8](#_Toc198794126)

[2.4.1. Nyelv és keretrendszer 8](#_Toc198794127)

[2.4.2. REST API és autentikációs lehetőségek 9](#_Toc198794128)

[2.5. Front-end technológiák 10](#_Toc198794129)

[3. Megvalósítási terv, szoftverkövetelmények meghatározása 12](#_Toc198794130)

[3.1. A felhasználói követelmények 12](#_Toc198794131)

[3.2. Megvalósítandó funkciók 12](#_Toc198794132)

[3. 2. 1. Bejelentkezési folyamat 13](#_Toc198794133)

[3.2.2. Szerepkörök és jogosultságok 13](#_Toc198794134)

[3.2.3. Use-case diagram 14](#_Toc198794135)

[3.3. Modell, tervezett struktúra 14](#_Toc198794136)

[3.3.1. Adatbázis modell 14](#_Toc198794137)

[3.3.2. Relációs modell 16](#_Toc198794138)

[3.3.3. Back-end struktúra 17](#_Toc198794139)

[3.3.4. Front-end struktúra 17](#_Toc198794140)

[4. Felhasznált technológiák és keretrendszerek részletes bemutatása 19](#_Toc198794141)

[4.1. MySQL Adatbázis 19](#_Toc198794142)

[4.2. Wampserver 19](#_Toc198794143)

[4.3. Spring Boot back-end API 19](#_Toc198794144)

[4.4. Gradle 20](#_Toc198794145)

[4.5. IntelliJ IDEA 21](#_Toc198794146)

[4.6. Könyvtárak, keretrendszerek 21](#_Toc198794147)

[4.6.1. Spring JPA 21](#_Toc198794148)

[4.6.2. Spring Session 21](#_Toc198794149)

[4.6.3. Lombok 21](#_Toc198794150)

[4.6.4. MapStruct 22](#_Toc198794151)

[4.7. Angular front-end 22](#_Toc198794152)

[4.7.1. Komponensek 22](#_Toc198794153)

[4.7.2. Modulok 22](#_Toc198794154)

[4.7.3. Direktívák 23](#_Toc198794155)

[4.7.4. Függőséginjektálás (Dependency Injection) 23](#_Toc198794156)

[4.7.5. Routing 23](#_Toc198794157)

[4.7.6. Űrlapok és validáció (Forms & Validation) 23](#_Toc198794158)

[4.7.7. HTTP kliens (HTTP Client) 23](#_Toc198794159)

[4.7.8. Állapotkezelés (State Management) – RxJS 24](#_Toc198794160)

[4.8. Bootstrap 24](#_Toc198794161)

[4.9. Visual Studio Code 24](#_Toc198794162)

[5. Megvalósítás 25](#_Toc198794163)

[5.1. Adatbázis 25](#_Toc198794164)

[5.2. Back-end 25](#_Toc198794165)

[5.2.1. Függőségek 25](#_Toc198794166)

[5.2.2. Spring környezet beállítások 26](#_Toc198794167)

[5.2.3. Struktúra 28](#_Toc198794168)

[5.3. Front-end 33](#_Toc198794169)

[5.3.1. Struktúra 33](#_Toc198794170)

[5.4. Az alkalmazás funkciói 35](#_Toc198794171)

[5.4.1. Bejelentkezés 35](#_Toc198794172)

[5.4.2. Navigációs sáv 36](#_Toc198794173)

[5.4.3. Saját adatok megtekintése és frissítése 37](#_Toc198794174)

[5.4.4. Felhasználók kezelése 38](#_Toc198794175)

[5.4.5. Hallgatók kezelése 39](#_Toc198794176)

[5.4.4. Cégek kezelése 40](#_Toc198794177)

[5.4.4. Szakmai gyakorlatok kezelése 40](#_Toc198794178)

[5.4.5 Hallgató saját szakmai gyakorlatának lekérdezése 42](#_Toc198794179)

[5.5 Az alkalmazás funkcionális tesztelése 43](#_Toc198794180)

[5.6. Felhasználói élmény tesztelés 43](#_Toc198794181)

[6. Összegzés tapasztalatok 44](#_Toc198794182)

[6.1. Fejlesztés során felmerülő nehézségek 44](#_Toc198794183)

[6.2. Továbbfejlesztési lehetőségek 44](#_Toc198794184)

[7. Summary 45](#_Toc198794185)

[7.1. Difficulties encountered during developement 45](#_Toc198794186)

[7.2. Further development opportunities 45](#_Toc198794187)

[8. Irodalomjegyzék 46](#_Toc198794188)

[9. Ábrajegyzék 48](#_Toc198794189)

# 1. Bevezetés

Szakdolgozatom célja egy olyan webes alkalmazás készítése, amely hatékonyan támogatja a Miskolci Egyetem Általános Informatikai Intézeti Tanszékét a hallgatók szakmai gyakorlatainak nyilvántartásában és adminisztrációjában.

Ezzel együtt szakdolgozatomban szeretném bemutatni a fejlesztési folyamat lépéseit, amelybe beletartoznak a tervezési feladatok, mint az alkalmazás feladatainak meghatározása, a megvalósításához használt technológiák és architektúra kiválasztása és azok ismertetetése, valamint az alkalmazás strukturális megtervezése és szemléltetése diagrammokkal. Végül pedig szeretném bemutatni a fejlesztési terv implementálásának menetét, az alkalmazáslogika és a funkciók kivitelezését és a tesztelési eredményeket,

## 1.1 Miért választottam ezt a feladatot?

Úgy gondolom, a szakdolgozati téma kiválasztásakor fontos szempont, hogy a választott feladat ne csupán elméleti kihívást jelentsen, hanem gyakorlati relevanciával is bírjon. A szakmai gyakorlatok nyilvántartására szolgáló web alapú rendszer fejlesztése ideálisnak bizonyult ebből a szempontból, hiszen egyrészt valós adminisztratív problémára kínál megoldást, másrészt kiváló lehetőség a szakmai fejlődésre a korszerű web-technológiák területén.

## 1.2 Az alkalmazás rövid bemutatása

Az alkalmazás fő feladata a hallgatók adatainak, gyakorlati helyeinek, valamint a gyakorlatokhoz kapcsolódó dokumentumok és egyéb információknak a tárolása és kezelése.

A rendszer webes felületen keresztül érhető el, így a felhasználók – hallgatók, tanszéki adminisztrátorok és gyakorlati koordinátorok – bárhonnan, egyszerűen hozzáférhetnek a szükséges információkhoz, és kezelhetik az összes hallgató gyakorlati adatait. Ehhez mindössze egy korszerű böngészőre van szükség.

## 1.3. Problémafelvetés

Jelenleg a szakmai gyakorlatok nyilvántartása Microsoft Excel munkafüzetekben történik, és habár az Excel program széleskörűen használható és kiváló bármely általános célra, a szakdolgozati dokumentumok ily módon való tárolása felvet néhány problémát.

Elsősorban a munkafüzetekben tárolt adatok rendszerezése kérdéses, és nem, vagy csak nehezen változtatható meg, például, ha minden évre új munkafüzet hozunk létre az aktuális szakmai gyakorlatok tárolására. Továbbá a formátum érvényesítésére nincsen lehetőség, manuálisan kell minden munkafüzetet a megfelelő formátumban kialakítani, vagy másolni az előzőt.

A gyakorlati adatok megosztása és kooperatív kezelése sem ideális, talán a legjobb megoldás, ami elérhető az Excel megosztott használatára, a Microsoft OneDrive által kínált osztott munkafüzet megoldás, viszont az is inkább projekt kategóriájú feladatok kezelésére ajánlott. Az előbbi megoldás kivitelezhetetlen egy sokfelhasználós rendszerben, amelyben bizonyos felhasználói szinteket kell alkalmazni, az adatvesztés elkerülése és az információvédelem érdekben. Ezen kívül lehetőség van bonyolult Excel makrókkal is a kívánt eredmény elérésére, azonban ehhez többnyire nincsen egységes modell a működést és kialakítást illetően, és gyakran jelentősen bonyolultabbak lehetnek, mint egy megfelelően a célra kialakított alkalmazás.

Mindezeken kívül pedig a szakdolgozathoz szükséges dokumentumok nyilvántartása is fontos, amelyet sokkal átláthatóbbá tesz egy előre megtervezett és dinamikusan változtatható fájlrendszer létrehozása. Excel használatakor ez is csak manuálisan kialakított fájlrendszer és hivatkozások segítségével kivitelezhető.

A megfogalmazódott számos probléma és hiányosság alapján egyértelműen megállapítható tehát, hogy egy egységes, web alapú rendszer bevezetése jelentősen növelheti a folyamat hatékonyságát és átláthatóságát, továbbá számos hasznos funkcióval rendelkezhet, és szükség esetén a továbbfejlesztés lehetősége is megvan.

## 1.4. Felhasználók köre

Az alkalmazás felhasználói két nagyobb csoportra oszthatóak. Tanszéki adminisztrátorok vagy gyakorlati koordinátorok, akik az intézet munkatársai, ők a hallgatók felvételéért és a gyakorlati jegyek beírásáért felelősek. Minden gyakorlati adat kezelésére jogosultak, szükség esetén módosíthatnak. Valamint hallgatók, akik a képzésükben kötelező szakmai gyakorlatukat végzik, ők a saját szakmai gyakorlati adataik és dokumentumaik feltöltéséért felelősek. Ezen kívül megemlíthető még az alkalmazás felelősök – ADMIN-ok szerepe is, akik új felhasználókat hozhatnak létre, szükség esetén a felhasználónevet, szerepkört, és jelszót módosíthatnak.

## 1.5. Elvárások a szoftverrel szemben

Az alkalmazással szemben néhány alapvető elvárás fogalmazható meg:

* **Felhasználóbarát felület:** Az alkalmazás könnyen kezelhető és intuitív legyen minden felhasználói csoport számára.
* **Hatékony adminisztráció:** A rendszer biztosítsa az egyszerű adatfeltöltést, és tegye lehetővé az adminisztrációs feladatok megosztását.
* **Átláthatóság:** Könnyen követhető legyen a hallgatók szakmai gyakorlatainak státusza, legyenek átlátható formátumban az adatok, és mellette elérhetőek a feltöltött dokumentumok is.
* **Rendezhetőség:** Az adatok lekéréséhez és listázásához legyen elérhető a szűrő, és rendező.
* **Biztonság:** A rendszer megfelelően védje a személyes adatokat, jogosultságkezeléssel biztosítsa, hogy minden felhasználó csak a számára releváns információkhoz férjen hozzá.
* **Bővíthetőség:** A szoftver architektúrája tegye lehetővé a későbbi fejlesztéseket, új funkciók egyszerű integrálását.

# 2. A fejlesztési követelmények meghatározása

## 2.1. Az alkalmazás helye és feladata

Mint a bevezetőben is említettem, a Tanszéken jelenleg a hallgatók szakmai gyakorlati adatait Excel táblázatokban tartják nyilván. Ez a megoldás nem skálázható, nehezen átlátható, és manuális adatbevitel miatt hajlamos hibákra. Az igény egy olyan adminisztratív webes alkalmazásra irányul, amely a szakmai gyakorlatok adatait strukturáltan, biztonságosan és könnyen kezelhetően tárolja. Az alkalmazásnak támogatnia kell a hallgatók, adminisztrátorok és gyakorlatvezetők munkáját, lehetővé téve a gyakorlatok nyomon követését, dokumentumok feltöltését, valamint az adatok gyors keresését és módosítását.

Az ideális választás az alkalmazás egyetemi szerveren való üzemeltetése, böngészőből elérhető klasszikus webes alkalmazásként. Így biztosítható a platformfüggetlenség, könnyű hozzáférés az egyetemi dolgozók és hallgatók számára, valamint a központi karbantartás lehetőségét. A továbbiakban olyan technológiákat vizsgálok meg, amelyek megfelelnek ezeknek a követelményeknek.

## 2.2. Architektúra és struktúra modellek

### 2.2.1. Monolitikus vagy mikroszolgáltatásos architektúra

Két klasszikus architektúramegvalósítást érdemes megemlíteni. Ezek a monolitikus architektúra és a mikroszolgáltatásos architektúra. A monolitikus architektúra esetén az egész alkalmazás egy egységként működik. Előnye, hogy egyszerűbb fejleszteni, üzemeltetni és tesztelni, valamint gyorsabb belső kommunikációt tesz lehetővé, mivel minden egy logikai folyamaton belül történik. Használata kisebb, kevésbé komplex alkalmazások esetében ajánlott. Gyakran monolitikus architektúrában megvalósított felépítés például a klasszikus háromrétegű alkalmazás struktúra.

Ezzel szemben mikroszolgáltatásos architektúra esetén az alkalmazás kisebb, önállóan működő szolgáltatásokból áll, amelyek API-kon keresztül kommunikálnak. Előnyei közé tartozik, hogy jobb a skálázhatósága és hibatűrése, mivel egy szolgáltatás meghibásodása nem érinti az egész rendszert. Az osztott rendszer megkönnyíti a fejlesztői csapatok közötti együttműködést is, mivel minden fejlesztői csapat a megfelelő szolgáltatáson dolgozhat. Hátrányai a nagyobb komplexitás a telepítésben és az adatkezelésben, valamint a szolgáltatások közötti kommunikáció miatt teljesítményhátrány léphet fel a monolitikus kialakításhoz képest. [1]

Tekintve, hogy a szakmai gyakorlat nyilvántartó program egyszerű, kis méretű alkalmazás, így ebben az esetben a monolitikus architektúra számos előnnyel rendelkezik, és sokkal ideálisabb az alkalmazás megvalósítására.

### 2.2.2. Three-tier (háromrétegű) alkalmazás

A háromrétegű architektúra klasszikus modell, a réteges architektúrákhoz tartozik. A kliens-szerver alkalmazásokhoz legyakrabban használt alkalmazás modell volt évtizedekig, viszont jelenleg már inkább leváltották a felhő alapú technológiák. Három elkülönülő rétegre bontja az alkalmazást: [2]

* Adatkezelési réteg (adatbázis): az adatok tartós tárolása.
* Alkalmazáslogikai réteg (back-end): az üzleti logika megvalósítása, adatfeldolgozás.
* Prezentációs réteg (front-end): felhasználói felület, amely böngészőből elérhető.

Ez a felépítés jól skálázható, karbantartható, és lehetővé teszi, hogy minden réteg önállóan fejleszthető, cserélhető legyen. A háromrétegű architektúra támogatja a biztonságosabb adatkezelést és a rétegek közötti világos felelősségmegosztást. [2]

A háromrétegű architektúra megfelelő kialakítás lenne a készítendő alkalmazásnak, tekintve, hogy megbízható, valamint a legelterjedtebb kliens-szerver kialakítás. Mivel egyetemi szerveren fog futni, így felhő alapú technológiák használatára nincs szükség.

## 2.3. Relációs adatbázis

A szakmai gyakorlat nyilvántartó alkalmazás strukturált, kapcsolatokkal rendelkező adatokat kezel, ezért egy megbízható relációs adatbázis-kezelő rendszer a legjobb választás.

A relációs adatbázisok (RDBMS – Relational Database Management System) az adatok táblázatos (sorok és oszlopok) formában történő tárolását és kezelését teszik lehetővé. Az adatok közötti kapcsolatok (relációk) kulcsok (elsődleges és idegen kulcsok) segítségével valósulnak meg. Ez a modell különösen alkalmas strukturált, egymással összefüggő adatok kezelésére. Az SQL lehetővé teszi összetett lekérdezések, adatelemzések és aggregációk végrehajtását, akár több táblán keresztül is. [3]

Az adatokon SQL parancsokkal minden fontos műveletet elvégezhetünk:

* Lekérhetjük az adatokat az adatbázisból (SELECT).
* Új rekordot illeszthetünk be az adatbázisba (INSERT).
* Módosíthatjuk az egyes rekordok tartalmát (UPDATE).
* Törölhetünk adatokat (DELETE).
* Több adaton való műveletvégzést a szűrés tesz lehetővé (WHERE).
* A lekért adatok rendezhetőek és csoportosíthatóak (ORDER BY, GROUP BY). [3]

Számos megfelelő relációs adatbázis-kezelő rendszer érhető el, többek között a MySQL amely nyílt forráskódú, könnyen kezelhető és nagy mennyiségű dokumentáció érhető el hozzá. Vagy a PostgreSQL, amely szintén nyílt forráskódú, robosztus, szorosan megfelel az SQL szabványoknak, és komplexebb lekérdezéseket is lehetővé tesz. Emellett megemlíthetjük a Microsoft SQL server, amely Microsoft alapú rendszerekbe könnyen integrálható. [3]

A szakmai gyakorlat nyilvántartó alkalmazáshoz a MySQL adatbázis egyszerűsége, népszerűsége és széles körben elterjedt használata miatt megfelelő lesz.

## 2.4. Back-end technológiák

### 2.4.1. Nyelv és keretrendszer

A back-end fejlesztéséhez többféle programozási nyelv és keretrendszer közül választhatunk, például Java-alapú framework-ök, a .NET (C#) vagy a Node.js (JavaScript). Mivel a Java nyelvben rendelkezem a legnagyobb tapasztalattal, ezért a back-end megvalósítására Java framework alkalmazását tervezem. A Java ökoszisztémán belül számos lehetőség kínálkozik, ezek közül a legelterjedtebb és legnépszerűbb a Spring Boot keretrendszer, amelyet én is választok, hiszen ebben van a legnagyobb tapasztalatom. A Spring Boot framework építéséhez a korszerű Gradle automatizációs build-eszközt fogom használni, amely jelentős segítséget nyújt az alkalmazás beállításában és a megfelelő függőségek (külső könyvtárak, egyéb keretrendszerek) csatolásában. Továbbá használni fogom még a JPA (Java Persistance API) framework-öt, amely egy ORM (Object-Relational Mapping) keretrendszer, és az adatbázis kommunikáció megvalósításáért felelős. Mivel a Spring Boot elterjedt technológia a vállalti alkalmazások fejlesztésében, ezért számtalan dokumentáció érhető el hozzá, nagyban segítve a fejlesztési folyamatot.

### 2.4.2. REST API és autentikációs lehetőségek

Az alkalmazásban a back-end front-end kommunikációhoz a back-end server RESTful megvalósítása szükséges.

A REST (Representational State Transfer) API (Application Programmin Interface) az egyik legelterjedtebb webes interfész. A reprezentációs állapotátvitel alapelveit alkalmazza, így platformfüggetlen: HTTP protokollon keresztül szolgáltat adatokat a front-end számára, továbbá támogatja a szabványos adatcsere formátumokat, mint a JSON vagy az XML. Az alapelveknek megfelelve laza összekapcsolást biztosít a kliens és a szerver között, így azok egymástól külön fejlődhetnek. [4][5]

Egységes erőforrás-azonosítókat, URI-ket (Uniform Resource Locator) alkalmaz, amely azt jelenti, hogy minden erőforrás egyedi URL címen érhető el. Az erőforrások-on végrehajtott műveleteket a szabványos HTTP parancsok teszik lehetővé, a kliens a GET, POST, PUT, PATCH és DELETE kérések valamelyikét küldheti a szervernek. A HTTP kérések funkciói a következők:

* GET: adatok lekérése,
* POST: új erőforrás létrehozása (új adatok felvitele),
* PUT: Meglévő erőforrások frissítése (adatmódosítás),
* PATCH: Részleges módosítások végrehajtása az erőforráson,
* erőforrás törlése (adat törlés) [4][5]

Ezen kívül a HTTP-nek köszönhetően a kommunikáció stateless, vagyis állapotmentes. Ez azt jelenti, hogy minden egyes request-nek (HTTP kérés) tartalmaznia kell az összes szükséges információt, például authentikációs adatokat is, hogy a kérés végbemehessen. Az állapotmentes megkötés miatt a kommunikáció megbízható, és jól átlátható, mivel a szervernek nincs szüksége másra a kérés adatain kívül, hogy meghatározza annak természetét [4][5]

Bár a REST API állapotmentes kommunikációt valósít meg, az autentikáció nem feltétlenül kell, hogy az legyen. A stateful autentikáció lényege, hogy a szerver tárolja a felhasználói munkamenetet egy session-ben, és minden kéréskor ellenőrzi azt. Így a hitelesítési adatokat nem kell újra elküldeni. A front-end ezután egy session cookie-val azonosítja magát, a szerver ebből tudja a felhasználó személyét. Előnyei, hogy a munkamenet bármikor visszavonható, könnyen kialakítható és kezelhető egy szerveres megvalósítás esetében, hátránya, hogy nagy terhelés esetén a szerver erőforrásai gyorsan kimerülhetnek, valamint nehéz skálázni több szerveres megvalósítás esetén. [6]

Stateless autentikáció esetén a felhasználói munkamenet adatai a kliens oldalon tárolódnak, például token formájában. Az adatokat minden egyes kérésnél újra kell küldeni, viszont a szerver generál kriptográfiával tikosított aláírt token-t a kliens számára. A kérést a titkosított aláírás segítségével hitelesíti a szerver. Előnye, hogy kisebb a szerverterhelés, könnyebben skálázható, gond nélkül használható sokszerveres megvalósításnál is, valamint egyszerűbb az integráció harmadik felekkel. Hátránya, hogy megvalósítása bonyolultabb, a jogosultságokat nem lehet azonnal visszavonni, a felhasználói adatok csak a token lejárata után módosíthatóak, ezért a token rövid élettartamú kell, hogy legyen. Gyakori megvalósítása például a JSON Web Token (JWT). [6]

Bár a stateless autentikáció jelenleg a legnépszerűbb a skálázhatósága miatt, a nyilvántartó projekt megvalósításához elegendő lesz a session alapú autentikáció.

## 2.5. Front-end technológiák

A front-end megvalósításához szintén számos framework érhető el, melyek többnyire HTML, CSS és JavaScrip használatát biztosítják. Ezek közül a legnépszerűbbek az Angular, a React és a Vue.js. Az Angular egy Google által támogatott komplett TypeScript alapú keretrendszer, amely kedvelt technológia a nagyvállalati webalkalmazások fejlesztésére, ezáltal nagy mennyiségű dokumentáció és segédlet érhető el hozzá. A TypeScript egy JavaScripten alapuló programozási nyelv, amelyet a Microsoft fejlesztett, és az adattípusok kezelését teszi lesz lehetővé. Az Angular előnyei a jól strukturált architektúra, valamint a számos beépített funkció és könyvtár. Hátránya a meredekebb tanulási görbe. [7]

Elérhető még a React, ami egy JavaScript alapú komponens-orientált könyvtár. Előnye, hogy nagyon rugalmas, gyors fejlesztést tesz lehetővé, és szintén nagy fejlesztő közösséggel rendelkezik, kiváló dinamikus interaktív alkalmazásokhoz. Hátránya szintén a meredek tanulási görbe, valamint a gyengébb strukturáltság. [7]

A Vue.js egy könnyen tanulható, kis méretű, komponens alapú JavaScript könyvtár. Előnye, hogy könnyen tanulható és gyors fejlesztési ciklust tesz lehetővé. Hátránya, hogy kisebb közösség van mögötte, és emiatt kevesebb eszköz, könyvtár érhető el hozzá. Továbbá egyre népszerűbbek még a Svelte és a SolidJS is, melyek érdekes új megoldásokkal dolgoznak, azonban még túlságosan új technológiák, kevesen ismerik ezeket, ezért nem ajánlottak például egy egyszerű nyilvántartó rendszerhez. [7]

A lehetőségeket és a személyes képességeket összevetve, az Angular az ideális eszköz a front-end megvalósítására. Azért döntöttem az Angular mellett, mert már van tapasztalatom a keretrendszerrel, ezáltal a tanulási görbe nem annyira meredek, emellett pedig jól dokumentált, sok bővítmény érhető el hozzá. TypeScript nyelvet használ, tehát az adatok típusossága, típusvédelme megmarad a front-end-en is.

# 3. Megvalósítási terv, szoftverkövetelmények meghatározása

## 3.1. A felhasználói követelmények

A fejlesztendő alkalmazás célja, hogy kiváltsa a jelenleg használt, Excel-alapú szakmai gyakorlat nyilvántartási rendszert, és egy korszerű, webes platformot biztosítson a Miskolci Egyetem Általános Informatikai Intézeti Tanszékének. Az alkalmazásnak alkalmasnak kell lennie a hallgatók személyes adatainak (például név, neptunkód), szakmai gyakorlatuk részleteinek (időtartam, eltöltött hetek száma, fogadó cég, céges kapcsolattartó, igazolás dátuma, kapott jegy), valamint a cégek adatainak (cégnév, cím) strukturált kezelésére. Emellett biztosítania kell a gyakorlat teljesítéséhez szükséges dokumentumok feltöltését és tárolását is.

A rendszer egyik fő újítása, hogy a hallgatók saját maguk tölthetik fel a szakmai gyakorlattal kapcsolatos adataikat és dokumentumaikat, ezzel jelentősen csökkentve az egyetemi adminisztrátorok terheit, akiknek főként csak módosítási, ellenőrzési és jegybeírási feladataik maradnak. Ez a megközelítés nemcsak hatékonyabbá teszi a folyamatot, hanem növeli az adatpontosságot és az átláthatóságot is, mivel a hallgatók aktívan részt vesznek saját adataik kezelésében.

A leírt felhasználói követelmények alapján a következtetés az, hogy az új webes alkalmazásnak egy korszerű, átlátható és felhasználóbarát rendszert kell biztosítania a szakmai gyakorlatok adminisztrációjára, amely megfelel a felsőoktatási szabályozásban és intézményi gyakorlatban elvárt követelményeknek.

## 3.2. Megvalósítandó funkciók

Az alkalmazás funkciói a bevezetésben már említett szerepkörök szerint kerülnek kialakításra, és a hozzáférés jogosultságokhoz kötött. A rendszer három fő felhasználói csoportot különböztet meg: hallgató, koordinátor és admin (administrator, jelentése is az alkalmazás felelőse).

Az admin felhasználók minden koordinátori jogosultsággal rendelkeznek, míg a hallgatók kizárólag saját adataikhoz férhetnek hozzá.

### 3. 2. 1. Bejelentkezési folyamat

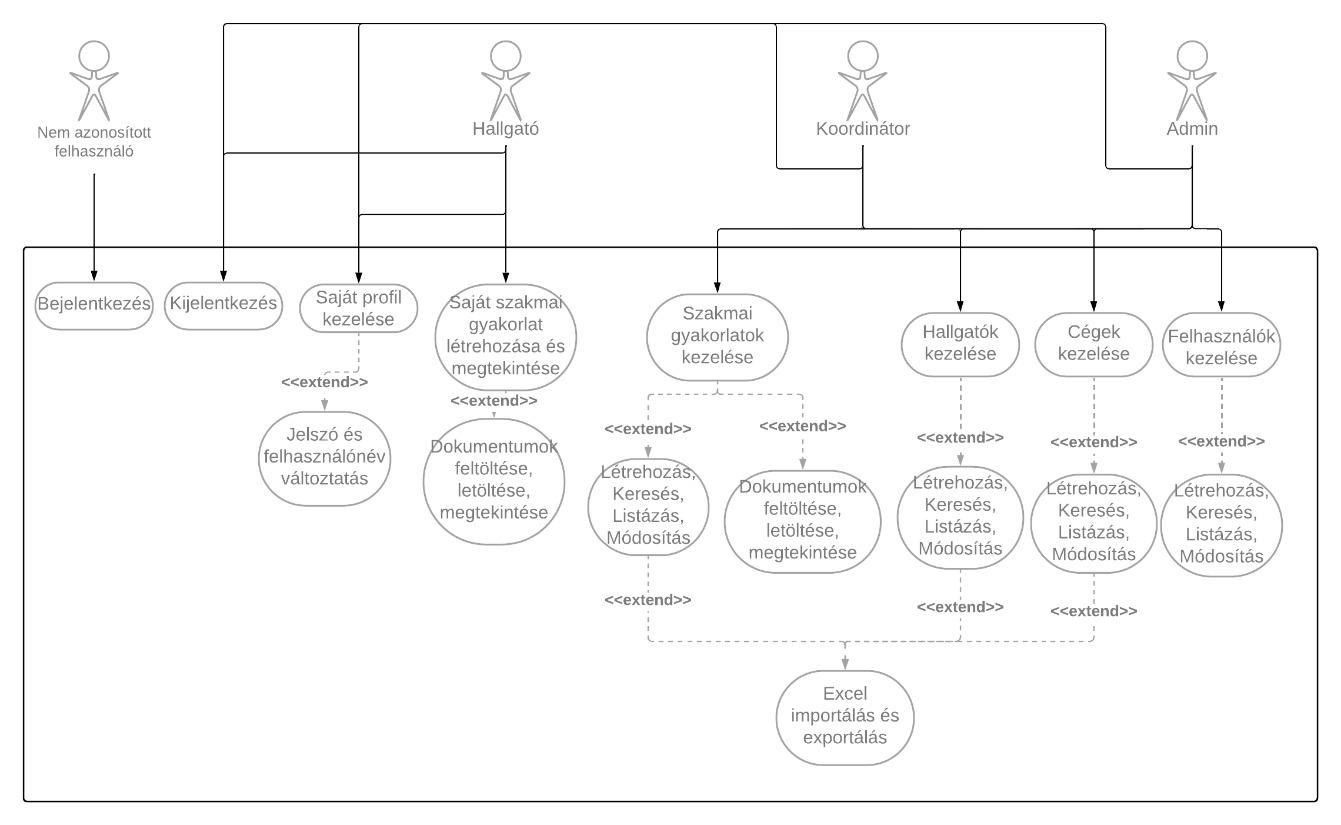
Az alkalmazás használata regisztrációhoz és hitelesítéshez kötött. A nem hitelesített felhasználók kizárólag a bejelentkezési felületet érhetik el.

### 3.2.2. Szerepkörök és jogosultságok

* **Hallgató:**
  + Saját szakmai gyakorlat adatainak létrehozása, megtekintése és szerkesztése.
  + Szükséges dokumentumok feltöltése vagy törlés szükség esetén.
* **Koordinátor:**
  + Hallgatók és cégek kezelése (létrehozás, listázás, módosítás, szűrés).
  + Gyakorlatok adatainak kezelése (létrehozás, listázás, módosítás, szűrés, dokumentumok feltöltése vagy törlése).
  + Adatok importálása meghatározott formátumú fájlból.
  + Adatok exportálása új Excel fájlba.
* **Admin:**
  + Minden koordinátori funkció elérése.
  + Felhasználók kezelése (jelszó módosítása, jogosultságok beállítása).
* **Minden hitelesített felhasználó számára elérhető funkciók:**
  + Saját profil megtekintése.
  + Felhasználónév módosítása.
  + Jelszó megváltoztatása.
  + Kijelentkezés.

### 3.2.3. Use-case diagram

A következő use-case diagram szemlélteti az alkalmazásban elérhető fő funkciókat és azt, hogy ezekhez melyik felhasználói szerepkör fér hozzá.



*1. ábra: Use-case diagram*

## 3.3. Modell, tervezett struktúra

### 3.3.1. Adatbázis modell

Az alkalmazás kialakítását először ER (Entity-Relationship) diagram segítségével szemléltetem. Ezen látható milyen szükséges adatbázistáblákat kell létrehozni, és milyen kapcsolat van közöttük.

A képen kör, képernyőkép, fekete-fehér, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*2. ábra: Adatbázis ER modell*

A diagramon látható, hogy melyik entitásnak (később adatbázistábla) milyen attribútumai (később adatbázis mezők) vannak. Az Internships (szakmai gyakorlatok) entitásnál az’(O)’-val jelölt mezők nem kötelező mezők, mert lehet ezeket még nem tudják, vagy nem írhatják be a hallgatók a szakmai gyakorlat létrehozásakor.

Az ER modellben látható kapcsolatok értelmezése a következő:

* **Felhasználók és hallgatók:** A felhasználók és a hallgatók között öröklődés van. A hallgató egy speciális felhasználó, aki a felhasználó adatai mellett a hallgatótáblában definiált mezőkkel is rendelkezik, ezért a hallgatótábla nem rendelkezik saját ID mezővel.
* **Hallgató és szakmai gyakorlat (egy az egyhez kapcsolat):** Egy hallgató egyetlen szakmai gyakorlaton vehet részt.
* **Szakmai gyakorlat és cég (több az egyhez kapcsolat):** Egy szakmai gyakorlat csak egy cégnél történhet, ugyanakkor egy céghez több szakmai gyakorlat is tartozhat.
* **Szakmai gyakorlat és dokumentumok (egy a többhöz kapcsolat):** Egy szakmai gyakorlathoz több dokumentum is kapcsolódhat.

### 3.3.2. Relációs modell

Az ER modell alapján a relációs adatbázis séma a következő táblákból áll:

* **User** (id, username, password, role),
* **Student** (id, name, neptuncode, specializaton),
* **Company** (id, name, address, active),
* **Internship** (id, start\_date, end\_date, weeks, completed, company\_instructor, grade, certificate\_date, completed),
* **Document** (id, internship\_id, title, file\_extension, file\_path)

Az átalakítás során a kapcsolatok idegen kulcsok (foreign key) segítségével kerülnek megvalósításra, tehát a kapcsolat két táblája közül az egyik táblában létrehozunk egy mezőt, amely hivatkozik a második tábla ID mezőjére. Öröklődésnél pedig osztott ID-t használunk, így a Studnet tábla megkapja a User tábla elsődleges kulcsát, és saját elsődleges kulcsaként használja.

Alul látható a ER modell alapján konvertált relációs modell.

A képen szöveg, képernyőkép, Betűtípus, fekete látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*3. ábra: Adatbázis relációs modell*

### 3.3.3. Back-end struktúra

A back-end oldalon a klasszikus Spring Boot architektúra szerint épül fel a rendszer, amely a következő fő komponensekből áll:

* **Entity osztályok:** Minden adatbázistábla mögött egy Java osztály (pl. User, Student, Company, Internship, Document) áll, amely az adatok leképezéséért felelős, adataik megegyznek a megfelelő adatbázistábla mezőivel.
* **Repository réteg:** Az adatbázis-műveletek (CRUD) kezelésére szolgál, például UserRepository, StudentRepository stb.
* **Service réteg:** Az üzleti logika megvalósítását tartalmazza, például StudentService, InternshipService.
* **Controllel réteg:** A REST API végpontokat valósítja meg, kezeli a kliens oldali kéréseket (pl. StudentController, InternshipController).
* **DTO-k és Mapper-ek:** A DTO-knak adatvédelmi szerepe van, biztosítja, hogy az Entity objektumból csak a szükséges mezők kerüljenek elküldésre, érzékeny adatok például jelszó semmiképpen sem.

Összefoglalva az Entity-k felelősek az adatbázis táblák lemodellezéséért. Az adatbázisból kapott adatok Entity formátumba kerülnek először. A Repository-k felelősek az adatbázis műveletek végrehajtásáért. Ezt a Service-ek fogják igénybe venni, akik üzleti logika megvalósításával a Repository rétegből összeillesztik az adatokat, DTO-kba konvertálják őket. A konvertálásért a Mapperek felelősek, majd a Service-ek továbbítják a kért adatokat a Controller-eknek, azok pedig HTTP kliensen keresztül küldik a front-end-nek. Adatok beillesztésekor az adatbázisba ugyanez történik meg, csak fordított sorrendben, ahol a végén a Repository-k beillesztik a kért adatokat az adatbázisba.

### 3.3.4. Front-end struktúra

Az Angular frontend oldalon a klasszikus komponens-alapú struktúrát követem:

* **Component-ek:** Minden fő funkcióhoz (például bejelentkezés, hallgatók lista, új hallgató létrehozása, hallgatók módosítása) külön komponens tartozik.
* **Service-ek:** Az adatok lekérését és küldését végzik a back-end API felé (pl. StudentService, InternshipService).
* **Modellek:** A back-end DTO-knak megfelelő TypeScript osztályok vagy interfészek, amelyek az adatok típusosságát biztosítják.
* **Routing és Auth Guard-ok:** Az útvonalak (oldalak) és a jogosultság-alapú hozzáférés kezelése.

Összegezve az Angular struktúrában a Komponensek felelnek az adatok megjelenítéséért a felhasználóknak. A Komponensek a Szervizektől kérik be a megjelenítendő adatokat. A service közvetlenül HTTP kéréseken keresztül kérik meg az adatokat a back-end-től, majd a kapott adatokat Modell formátumba konvertálják, és azokat továbbítják a Szervizeknek. Az oldalak (Komponensek) közötti navigálásáért a Routing felel, az Auth Guard-ok pedig biztosítják, hogy minden Komponenst csak az arra jogosult felhasználók érhessenek el. Ugyanez a folyamat fordítva valósul meg adatok küldésekor a back-end felé.

# 4. Felhasznált technológiák és keretrendszerek részletes bemutatása

## 4.1. MySQL Adatbázis

A rendszer adatainak tárolására relációs adatbázist használunk, amely lehetővé teszi a strukturált, kapcsolatokkal rendelkező információk biztonságos és hatékony kezelését. A MySQL egy széles körben elterjedt, nyílt forráskódú relációs adatbázis-kezelő rendszer, amely kiválóan alkalmas egy nyilvántartó alkalmazás kiszolgálására. Előnyei közé tartozik a gyorsaság, a megbízhatóság, a könnyű telepítés és karbantartás, valamint a széleskörű támogatottság.

Számos nagy és gyorsan növekvő vállalat, például a Facebook, a Twitter, a Booking.com és a Verizon is a MySQL adatbázis-kezelő rendszert használja, hogy időt és pénzt takarítson meg a nagy forgalmú weboldalak, üzletileg kritikus rendszerek és szoftverimplementációk működtetése során. [8]

## 4.2. Wampserver

A fejlesztési környezetben a MySQL-t a Wampserver-en keresztül használom, amely egy egyszerűen kezelhető, Windows-alapú webszerver csomag, integrált Apache, PHP és MySQL szolgáltatásokkal.

## 4.3. Spring Boot back-end API

A back-end fejlesztéséhez a Java nyelvet és a Spring Boot keretrendszert választottam. A Spring Boot nagy előnye, a széles választék rugalmas és átfogó harmadik féltől származó könyvtáraiból. Gyors indítást, leállítást és alapértelmezett optimalizált végrehajtást kínál. Egy hatékonyan izolált futtatókörnyezetet biztosít, amely független webszerver vagy alkalmazáskiszolgáló nélkül is működtethető. Emellett támogatja a visszafelé kompatibilitást, és megkönnyíti a projektek tesztelését [9]

A Spring MVC és a Spring Boot modern, reaktív és felhőalapú technológiákat biztosít a keretrendszernek nagy teljesítményű, komplex webes alkalmazások építéséhez, amelyeket számos technológiai óriás, köztük a Netflix, az Amazon, a Google, a Microsoft stb. használ. [9]

A Spring Boot egy stabil Java web framework, amelyet jelenleg is folyamatosan fejlesztenek, így biztosítva a korszerű technológiákat. A Spring megkönnyíti RESTful API-ok gyors és strukturált fejlesztését. Előnyei közé tartozik a magas szintű modularitás, a beépített biztonsági és adatkezelési megoldások, valamint a széles körű közösségi támogatás.

Az Spring tartalmazza a Spring MVC keretrendszert, amely az MVC (Model-View-Controller) tervezési mintát, egy szoftvertervezési mintát, amely az alkalmazást három fő komponensre osztja: Model , View és Controller. A modell az adatok és az üzleti logika kezeléséért felel, a view (nézet) az adatok megjelenítését teszi lehetővé a felhasználó számára, a controller (vezérlő) pedig a felhasználói interakciók alapján végre hajtja a megfelelő műveleteket. Ez a szétválasztás segít a kód strukturálásában, megkönnyíti a karbantartást és a tesztelést, valamint támogatja a skálázhatóságot. Az MVC széles körben alkalmazott webes és asztali alkalmazások fejlesztésében. [10]

A Spring megvalósítja a Dependency Injectiont, amely segít kezelni az alkalmazás összetevői közötti függőségeket. Ahelyett, hogy egy osztály saját függőségeket hozna létre, azokat egy külső forrásból injektálja, így a kód modulárisabbá, tesztelhetőbbé és karbantarthatóbbá válik. Az injection a ’@Autowired’ annotációval használható, osztály és konstruktor szinten. Az annotációk ’@’ jellel kezdődő sorok, a Java-ban olyan metaadat szolgáltatók, amelyeknek a program futásában nincs szerepük, viszont a fordítónak adnak addicionális információt a konstruktorok metódusok osztályok kezeléséhez. [10]

## 4.4. Gradle

A projekt menedzselésére a Gradle-t használom, amely egy open-source, hatékony, deklaratív build-eszköz, Java-hoz, Kotlin-hoz és Arndroid-hoz, amely Groovy szintaxist használ. Lehetővé teszi az építési folyamatok automatizálását, optimalizálását és gyorsítását, valamint egyszerűvé teszi a függőségek kezelését. Új Spring projekten belül a build.gradle fájlba kell a Gradle natív szintaxisán megadni a dependency elérhetőségét, és használható is az adott könyvtár a projektben. [11]

## 4.5. IntelliJ IDEA

A fejlesztéshez az IntelliJ IDEA fejlesztői környezetet alkalmazom, amely elsőosztályú integrációt nyújt a Spring, Spring Boot, JPA és más további Java keretrendszerekhez. Az IntelliJ lehetővé teszi az adatbázisokkal való közvetlen interakciót, SQL-lekérdezések szerkesztését és élő adatbázis-kapcsolatokat. Fejlett kódolási, hibakeresési és refaktorálási eszközöket kínál. Továbbá intelligens kód kiegészítéssel rendelkezik, ezzel jelentősen növelve a fejlesztés hatékonyságát. Mindezeken felül lehetővé teszi a kódolást könnyű helyi klienseken, miközben a nehéz feldolgozást távoli szerverekre terheli. [12]

## 4.6. Könyvtárak, keretrendszerek

### 4.6.1. Spring JPA

A projektben használt függőségek közül szeretnék kiemelni néhányat. Elsősorban a Spring Data JPA-t, amely szinte teljes egészben kezeli az adatbázis feladatokat. A Java objektumokból képes automatikusan létrehozni vagy frissíteni az adatbázis sémát. Számos adatbázis-kezelő rendszerrel kompatibilis, és nem csak relációs adatbázist tud kezelni. Elvégzi az alapvető CRUD (Create, Read, Update, Delete) adatbázis műveleteket, és teljes egészében elvégzi az SQL utasítások megírását. [11]

### 4.6.2. Spring Session

Kiemelném még a Spring Session-JDBC könyvtárat, amely a JDBC adatbázis kapcsolaton keresztül, automatikusan létrehozza a session kezeléshez szükséges adattbázis sémát, és a MySQL adatbázisban tárolja, onnan olvassa ki a felhasználók session adatait. Ez azért fontos, mert így az adatok perzisztens környezetben vannak, és a session adatok nem vesznek el egy újraindítás után. [11]

### 4.6.3. Lombok

A Lombok nagyon hasznos könyvtár a boilerplate kód eltávolítására. A boilerplate olyan újrahasználható kód, amit a programban több helyen is le kell írni minimális változtatással, például getter setter metódusok. A Lombok az ilyen kódok kiváltására biztosít annotációkat. [11]

### 4.6.4. MapStruct

Végül pedig a MapStruct egy könyvtár, amely a Java-ban létrehozott Entity objektumok automatikus konvertálását végzi el DTO-kra (Data Transfer Object) és DTO-ról vissza enity-re. [11]

## 4.7. Angular front-end

A felhasználói felület fejlesztéséhez az Angular keretrendszert választottam, amely egy nagy teljesítményű front-end keretrendszer, melyet a Google fejlesztett ki, és dinamikus, egyoldalas webes alkalmazások – SPA-k (Single Page Application) építésére alkalmas. TypeScript programozási nyelvet használ, amely egy JavaScripten alapuló, erősen típusos nyelv, használata növeli a kód biztonságát és olvashatóságát. [13]

Az Angular előnye, hogy strukturált, moduláris fejlesztést tesz lehetővé, támogatja a kétirányú adat-kötést, és komponens-alapú architektúrát követ, így jól skálázható nagyobb alkalmazások esetén is. [13]

Alább megemlítem az Angular keretrendszer lényeges elemeit és tulajdonságait.

### 4.7.1. Komponensek

A komponensek az Angular alkalmazások alapvető, újra felhasználható UI (felhasználói felület) elemei. Minden komponens tartalmaz egy HTML sablont, stílusokat, logikát és adatokat. Minden Angular alkalmazás egy gyökérkomponenssel indul, amelybe további komponensek ágyazhatók, így építve fel az alkalmazás szerkezetét. A komponensek egymásba ágyazhatók, kommunikálhatnak egymással, és lehetővé teszik a tiszta, moduláris fejlesztést. [13]

### 4.7.2. Modulok

Az Angular modulok (NgModules) segítenek a kód logikai egységekre bontásában, ami egyszerűbbé és átláthatóbbá teszi a nagyobb alkalmazások karbantartását. A modulok tartalmazhatnak komponenseket, direktívákat, szolgáltatásokat és egyéb Angular elemeket. Minden alkalmazásnak van egy gyökérmodulja (AppModule), amely elindítja az alkalmazást, de további funkcionális vagy feature modulokat is létrehozhatunk a kód szervezésére. [13]

### 4.7.3. Direktívák

A direktívák lehetővé teszik a HTML funkcionalitásának kibővítését egyedi viselkedésekkel. Ezek speciális utasítások, amelyeket a HTML elemekhez adhatunk, például feltételes megjelenítés (\*ngIf), ciklusok (\*ngFor) vagy saját, egyedi viselkedések hozzáadása. A direktívák révén dinamikusan módosíthatjuk a DOM-ot és az alkalmazás viselkedését. [13]

### 4.7.4. Függőséginjektálás (Dependency Injection)

Az Angular beépített függőséginjektálási rendszere (DI) hatékonyan kezeli a szolgáltatások és egyéb függőségek elérhetőségét a komponensekben. A szolgáltatások (@Injectable dekorátorral ellátott osztályok) például adatkezelésre, hitelesítésre vagy naplózásra használhatók, és egyszerűen injektálhatók bármely komponensbe, így elősegítve a laza csatolást és az újrafelhasználhatóságot. [13]

### 4.7.5. Routing

Az Angular router lehetővé teszi, hogy az alkalmazás különböző nézetei (oldalai) között navigáljunk anélkül, hogy az egész oldalt újratöltenénk. A router útvonalakat (routes) definiál, amelyekhez komponenseket rendelhetünk, így megvalósítva az egyoldalas alkalmazások (SPA) dinamikus navigációját. [13]

### 4.7.6. Űrlapok és validáció (Forms & Validation)

Az Angular kétféle űrlapkezelési megközelítést támogat: a template-driven (sablonvezérelt) és a reactive (reaktív) űrlapokat. Mindkettő lehetővé teszi összetett űrlapok létrehozását, adatkötést, valamint beépített és egyedi validációk alkalmazását, így biztosítva az adatok helyességét még a beküldés előtt. [13]

### 4.7.7. HTTP kliens (HTTP Client)

Az Angular HttpClient modulja leegyszerűsíti a szerverrel történő kommunikációt, például API-k hívását, adatok lekérését vagy küldését. Az HttpClient az RxJS Observable-okat használja, így támogatja az aszinkron adatkezelést, a hibakezelést, valamint a válaszok és hibák kezelésének testreszabását is. [13]

### 4.7.8. Állapotkezelés (State Management) – RxJS

Az Angularban az állapotkezeléshez gyakran használják az RxJS könyvtárat, amely lehetővé teszi az aszinkron adatfolyamok kezelését. Az Observable-ok segítségével könnyen kezelhetők az események, adatok, API válaszok és egyéb aszinkron műveletek, ami különösen fontos nagyobb, összetett alkalmazásoknál. [13]

## 4.8. Bootstrap

A felhasználói élmény és a reszponzív dizájn érdekében Bootstrap-et integrálok az alkalmazásba, amely előre elkészített stílusokat és komponenseket kínál. Installálása az Angular projektben történik, VS Code terminálba a következő parancsot beírva: npm i bootstrap@latest (vagy az aktuális verzió szám, amelyből a legfrisebb az 5.3.6-os verzió.

A BootStrap egy népszerű CSS keretrendszer, amely számos előnyt kínál a webfejlesztésben. A BootStrap rendkívül reszponzív, a beépített grid system és stílusai segítenek abban, hogy a weboldalak minden eszközön jól jelenjenek meg. Előre elkészített komponensek, például gombok, űrlapok és navigációs elemek gyorsítják a fejlesztési folyamatot. Biztosítja, hogy a weboldalak minden modern böngészőben megfelelően működjenek. Az elemeik megjelenése és stílusa Sass és CSS változók segítségével könnyen módosíthatók. Számos dokumentáció és nagy fejlesztői közösség segíti a használatát. [14]

## 4.9. Visual Studio Code

A fejlesztéshez a Visual Studio Code szerkesztőt használom, amely könnyen testreszabható, gyors és számos hasznos bővítménnyel támogatja a modern webfejlesztést A Visual Studio Code egy könnyű és nagy teljesítményű forráskód-szerkesztő, amely támogatja a különböző programozási nyelveket, például a JavaScriptet, a TypeScriptet, a C++-t, a C#-t, a Pythont, a PHP-t és a Go-t. Bővítmények gazdag ökoszisztémájával rendelkezik, amelyek bővítik a képességeit. A Visual Studio Code néhány hasznos funkciója és felhasználási módja az IntelliSense intelligens kódkiegészítés, a paraméterinformációk, gyorsinformációk és taglisták biztosítása. A beépített hibakeresési támogatás lehetővé teszi az alkalmazások futtatását és hibakeresést közvetlenül a szerkesztőből. Számos bővítmény elérhető, amely lehetőséget kínál a funkciók testreszabására és bővítésére. Beépített terminállal rendelkezik a parancsok és szkriptek közvetlenül a szerkesztőben történő futtatásához. A VS Code fejlesztési feladatok széles skálájára alkalmas, a webfejlesztéstől az alkalmazásfejlesztésig, így sokoldalú eszköz a fejlesztők számára. [15]

# 5. Megvalósítás

## 5.1. Adatbázis

Elindítottam a Wampserver-t, bejelentkeztem, és a localhost:3306-os porton létrehoztam egy új üres adatbázist internship\_registry néven. Az adatbázissal további teendőm nem volt, a sémát a Spring JPA alakítja ki.

## 5.2. Back-end

Készítettem egy új Spring Boot projectet az IntelliJ beépített Spring Initializr funkciójával, és megadtam a projektben használni kívánt dependenciákat (függőségeket) az alkalmazás gyökérkönyvtárában lévő build.gradle fájlban. A dependenciák többnyire olyan keretrendszerek és könyvtárak, amelyek boilerplate kódot, vagy egy nagyobb részfeladat megvalósítását váltják ki, és automatizálják.

### 5.2.1. Függőségek

* **Spring-boot-starter-web:** REST API-k és webalkalmazások gyors fejlesztéséhez szükséges alapcsomag (Spring MVC, beépített Tomcat szerver).
* **Spring-boot-starter-security:** Biztonsági funkciók (pl.: hitelesítés, jogosultságkezelés) egyszerű integrálása.
* **Spring-boot-starter-actuator**: Alkalmazás monitorozása, egészségügyi állapot, statisztikák, metrikák elérése.
* **Spring-boot-starter-validation**: Adatbeviteli validáció (Bean Validation API, pl.: @NotNull, @NotBlank, @Size validációk). A DTO modellek adatainak validálására használom, hogy az adatbázisba csak megfelelő értékek kerüljenek.
* **Spring-boot-starter-jdbc:** Egyszerű JDBC-alapú adatbázis-kezeléshez szükséges függőségek.
* **Spring-boot-starter-data-jpa:** Objektum-relációs leképezés (ORM) és adatbázis-műveletek JPA/Hibernate segítségével. A teljes adatbázis kezelésért felel, a Java műveleteket SQL utasításokká alakítja.
* **Spring-session-jdbc**: Felhasználói munkamenetek (session) kezelését valósítja meg. Sémát alakít ki a session-ök és lehetséges attribútumaik tárolásra az adatbázisban JDBC-n keresztül.
* **Spring-boot-test:** Spring Boot alkalmazások integrációs teszteléséhez szükséges eszközök.
* **Spring-security test:** Biztonsági funkciók (pl. jogosultságok) tesztelésének támogatása.
* **MySQL-connector:** Java alkalmazás és MySQL adatbázis közötti kommunikációt biztosító JDBC meghajtó.
* **Lombok:** Kódgeneráló eszköz, amely annotációkkal (pl. @Getter, @Setter) csökkenti a boilerplate kódot. Csak fordítási időben szükséges.
* **MapStruct**: Gyors és típusbiztos DTO (Data Transfer Object) és entitás objektumok közötti automatikus leképezés (mapping).
* **Apache Tika:** Fájlok típusának felismerése és tartalmának kinyerése (PDF, Word, Excel, és további népszerű fájltípusok).
* **Apache POI:** Microsoft Excel (.XLS, .XLSX) dokumentumok olvasása és írása Java-ban.

### 5.2.2. Spring környezet beállítások

A resources mappában található application.properties fájlban beállítottam minden, a projekthez hasznos és szükséges beállítást. Ezek a következők:

**Alkalmazás neve és portja**:

* spring.application.name=InternshipRegistryBackend,
* server.port=8080

**Hibakezelés**:

* server.error.include-message=always (mindig tartalmazza a hibaüzenetet),
* server.error.include-binding-errors=always (mindig tartalmazza a binding hibákat)

**Session beállítások**:

* server.servlet.session.cookie.secure=true (a session cookie biztonságos, HTTPS esetén),
* server.servlet.session.cookie.same-site=lax (SameSite cookie attribútum beállítása)

**Fájlfeltöltés**:

* spring.servlet.multipart.enabled=true (fájlfeltöltés engedélyezve),
* spring.servlet.multipart.max-file-size=5MB (maximális fájlméret: 5MB),
* spring.servlet.multipart.max-request-size=10MB (maximális kérésméret: 10MB)

**Spring Session JDBC beállítások**:

* spring.session.jdbc.initialize-schema=always (az adatbázis séma inicializálása mindig megtörténik),
* spring.session.jdbc.table-name=SPRING\_SESSION (a session adatok tárolására használt tábla neve),
* spring.session.cleanup-cron=0 \* \* \* \* \* (a session takarítási időzítése),
* spring.session.timeout=60m (session időtúllépés: 60 perc)

**Adatbázis konfiguráció (MySQL)**:

* spring.datasource.url=jdbc:mysql://localhost:3306/internship\_registry (adatbázis URL),
* spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver (JDBC driver),
* spring.datasource.username=root (felhasználónév),
* spring.datasource.password= (jelszó - üres)

**JPA/Hibernate beállítások**:

* spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect (hibernate dialektus),
* spring.jpa.properties.hibernate.format\_sql=true (formázott SQL kimenet),
* spring.jpa.show-sql=true (SQL lekérdezések megjelenítése a logban),
* spring.jpa.hibernate.ddl-auto=update (adatbázis séma frissítése indításkor),

**Logolási szintek**:

* logging.level.root=INFO (gyökér szintű logolás: INFO),
* logging.level.org.springframework.web=DEBUG (Spring Web logolás: DEBUG),
* logging.level.hu.me.iit=TRACE (egyedi csomag logolása: TRACE)

**Actuator beállítások**:

* management.endpoints.web.exposure.include=info (az info endpoint elérhetővé tétele a weben)

### 5.2.3. Struktúra

Az back-end API fejlesztése során a klasszikus web API struktúrával alakítottam ki az alkalmazás logikai felépítését. A következő fájlok készültek el funkció szerint az ’src/main/java/hu.me.iit.internshipregistrybackend/’ mappában, további szerepük szerinti mappákba csoportosítva:

**Congiuration osztályok:** A konfigurációs osztályok felelősek az alkalmazás indításakor szükséges beállításokért és a Spring kontextus konfigurálásáért. A védelmi és hitelesítési feladatokat látják el.

* **WebConfig.java:** Ez az osztály a webes konfigurációért felelős, különösen a CORS (Cross-Origin Resource Sharing) beállításaiért.
  + Engedélyezi a kéréseket a http://localhost:4200 címről, amely az angular frontend portja.
  + Engedélyezi a GET, POST, PUT, DELETE HTTP metódusokat.
  + Engedélyezi az összes fejlécet (\*).
  + Engedélyezi a credential-öket (pl. cookie-k, HTTP hitelesítés) küldését a kérésekkel.
* **SecurityConfig.java:** Ez az osztály a Spring Security konfigurációjáért felelős, amely az alkalmazás biztonságát kezeli.
  + **CORS konfiguráció:** Hasonlóan a WebConfig-hoz, itt is beállítja a CORS szabályokat.
  + **CSRF (Cross-Site Request Forgery) védelem:** Jelenleg a fejlesztési folyamatban a CSRF védelem ideiglenesen ki van kapcsolva.   
    A .CookieCsrfTokenRepository.withHttpOnlyFalse() parancs használatával, felállíthatjuk a CSRF védelmet, amelyet az Angular front-end automatikus kezel. A front-end ilyenkor egy CSRF token cookie-t tárol és küld minden HTTP kérés során.
  + **Engedélyezési szabályok (authorizeHttpRequests):** 
    - A "/login" útvonal minden felhasználó számára elérhető (.permitAll()).
    - Az "/admin/\*\*" útvonalak csak az ADMIN szereppel rendelkezők számára engedélyezettek (.hasRole("ADMIN")).
    - Minden más kéréshez hitelesítés szükséges (.anyRequest().authenticated()).
    - Bejelentkezés (formLogin): a bejelentkezési kérések feldolgozási URL-je "/login".
    - Sikeres bejelentkezés kezelése: Ha a bejelentkezés sikeres, a válasz HTTP 200 OK lesz, application/json tartalomtípussal. A válasz tartalmazza a status: "ok", a felhasználónevet (username) és a felhasználó szerepeit (roles).
    - Sikertelen bejelentkezés kezelése: Ha a bejelentkezés sikertelen, a válasz HTTP 401 UNAUTHORIZED lesz, application/json tartalomtípussal. A válasz tartalmazza a status: "unauthorized", és a username, roles mezők null értékűek lesznek.
  + **Kijelentkezés (logout):** 
    - A kijelentkezési URL "/logout".
    - A JSESSIONID, XSRF-TOKEN, SESSION cookie-kat törli.
    - Érvényteleníti a HTTP session-t.
    - Sikeres kijelentkezés esetén HTTP 200 OK státuszkódot ad vissza.
  + **Kivételkezelés (exceptionHandling):** Hitelesítési hiba (authenticationEntryPoint) esetén HTTP 401 UNAUTHORIZED hibát küldvissza "Unauthorized" üzenettel.
  + **Session kezelés (sessionManagement):** Maximálisan egy aktív session-t engedélyez felhasználónként (.maximumSessions(1).
  + **Jelszókódoló (passwordEncoder):** A BCryptPasswordEncoder típusú jelszókódolót definiálja, amely a jelszavak biztonságos tárolására szolgál.

**AdminIntializer:** Létrehoztam egy osztályt a fejlesztési ciklushoz, amely egy admin felhasználót kreál az adatbázisban, ’admin’ felhasználónévvel és ’admin123’ jelszóval, amennyiben még nincs létrehozva ilyen nevű felhasználó.

**Enitity osztályok:** Az adatbázistáblák leképzése a céljuk, tehát minden adatbázistáblára készül egy Entitiy class. Az adatbázistábla mezőit, a relációs kulcs mezőket tartalmazzák, @Entity annotációval rendelkeznek. A spring JPA automatikusan elvégzi az adatbázis séma kialakítását az Entitiy osztályok alapján. Az Entity mappában helyzkednek el:

* **User**: A felhasználók tábla leképzéséért felelős.
* **Student:** A hallgatók tábla leképzéséért felelős.
* **Company**: A cégtábla leképzéséért felelős.
* **Internship**: A szakmai gyakorlatok tábla leképzéséért felelős.
* **Document:** A dokumentumok tábla leképzéséért felelős.

A user Role-ok kezelésére létrehoztam egy **Role enum** osztályt egy külön enum mappában.

**Repository réteg:** Az itt jelenlévő osztályok az adatbázis műveletek végrehajtásáért felelősek, közvetlenül írnak az adatbázisba. CRUD műveleteket alkalmaznak, vagyis funkcióik a rekordok létrehozása, olvasása, frissítése és törlése. A Java utasításokat SQL kódra fordítják és továbbítják az adatbázisnak. A JPA Repository leszármazottjai, így az mezőkre vonatkozó egyszerű adatbázis kezelő metódusokat (pl.: find, save, delete) már nem kell explicit módon létrehozni a programban.

Mindegyik Entityhez tartozik egy Repository, amely a megfelelő adatbázistábla adatbázis műveleteiért felelős. Az adatbázisból lekért adatok Entitiy osztály példányokban kerülnek tárolásra.

A InternshipRepository-ban létrehoztam néhány egyedi @Query annotációval ellátott SQL-ben megírt lekérést, hogy a szakmai gyakorlat ideje és a befejezett státusz alapján is lehessen keresni.

**Service réteg:** Az itt jelen lévő osztályok az üzleti logikát valósítják meg. A Service réteg metódusait a Controller réteg hívja meg felhasználói interakció hatására, a Service pedig a Repositpory osztályokat hívja meg az adatbázis műveletek elvégzésére. A kapott Entity-ket DTO-kká konvertálva küldi tovább a Controllerek felé. A service réteg rendkívül fontos, mivel itt valósul meg az üzleti logika, és az információ rejtés is itt történik.

A projektemben az egy adatbázistáblához tartozó összes logikát, a táblával megegyező nevű entitás hajtja végre, így itt az osztályok funkciók szerinti elkülönítésére nem került sor.

Speciális service osztály a MyUserDetailsService, amely a bejelentkezéshez szükséges metódusokat valósítja meg. A bejelentkezéshez először felhasználónév alapján meg kell keresni a felhasználót majd, UserDetailsService objektummá konvertálni az adatbázisból lekért felhasználót. Az osztályban jogosultság kezelési logikákat is implementálhatunk, itt adtam meg, hogy az admin felhasználók automatikusan kapják meg, koordinátor role-t is.

**DTO osztályok**: A Data Transfer Objektumok felelnek azért, hogy csak a szükséges adatok legyenek továbbítva a Controllereknek, és majd a front-end-nek. Több különböző típusú DTO osztály van a projektben, két nagy csoportra osztva: az olvasási, létrehozási, frissítési DTO-k. Az írási DTO-k, mindig ugyanazokat az adatokat küldik, és azok az adatok vannak kihagyva belőle, amelyek elküldése a kliens felé nem célszerű, vagy nem ajánlott (pl.: jelszó). Az olvasási és frissítési DTO-knál eltérő lehet a követelmény, így azokat felsorolom:

* **CreateUpdateUserDto:** A felhasználó minden adatát meg kell adni a létrehozáshoz.
* **UpdateUserDto:** A felhasználó adatainak frissítése jelszó nélkül, mivel az külön valósul meg.
* **UpdatePasswordDto:** A felhasználó jelszavának validációjához és frissítéséhez.
* **CreateStudentDto:** A hallgató és a szülője, a felhasználó minden adatát meg kell adni a létrehozáshoz, kivéve a jogosultsági kört, mert az automatikusan a hallgató.
* **UpdateStudentDto:** A hallgató frissítése, csak a hallgató adatait kell megadni, a felhasználói adatok frissítésére rendelkezésre állnak az UpdateUser és UpdatePassword DTO-k.
* **CreateUpdateCompanyDto:** A cég létrehozásában és frissítésében nincs lényeges eltérés.
* **CreatUpdateInternshipDto:** A szakmai gyakorlat létrehozásában és frissítésében nincs lényeges eltérés.
* A dokumentumok létrehozása a feltöltött fájl alapján történik a Service osztályban így arra nincs külön DTO.

**Mapper osztályok:** MapStructtal készült osztályok @Mapper annotációval rendelkeznek. Csak az olvasási DTO-khoz készültek Mapper osztályok, mivel az írási és frissítési DTO-kat builder metódussal építem fel a Service osztályokban, mivel így transzparensebb és jobban kezelhető a létrehozási/frissítési folyamat.

**Controller réteg:** A Controller rétegben található osztályok a HTTP kérések fogadásáért és válaszok küldéséért felelősek. DTO objektumokat továbbítanak a front-end felé (amely JSON formátumra lesz konvertálva HTTP-n keresztül), és adatokat fogadnak a front-end-től, amelyeket JSON formátumból DTO-kba illesztenek. Metódusaik egyedi URI címeken elérhetők a front-end számára.

A Controller osztályok logikailag jobban elkülönülnek egymástól a projektben, az osztályok és feladataik a következők:

* **UserController:** Admin jogosultsággal rendelkező felhasználók az ebben az osztályban található címeken kezelhetik a felhasználókat.
* **StudentController:** Az adminok és koordinátorok az ebben az osztályban található címeken kezelhetik a hallgatók adatait.
* **CompanyController:** Az adminok és koordinátorok az ebben az osztályban található címeken kezelhetik az cégek adatait.
* **InternshipController:** Az adminok és koordinátorok az ebben az osztályban található címeken kezelhetik a szakmai gyakorlatokat.
* **DocumentController:** Az az ebben az osztályban található címeken keresztül a fájlfeltöltés és letöltés valósul meg az adminok és koordinátorok által elérhető szakmai gyakorlati címekhez.
* **UserSelfController:** Az ebben az osztályban található címeken keresztül az összes felhasználó megtekintheti a saját adatait, és frissítheti azokat, a jogosultsági körét kivéve.
* **StudentSelfController:** Ebben az osztályban a hallgatóknak kialakított metódusokhoz tartozó címek találhatóak, amelyen keresztül a hallgatók létrehozhatják és megnézhetik saját szakmai gyakorlatukat, és dokumentumokat tölthetnek fel az adott gyakorlathoz.

**Exception osztályok:** Ezek az osztályok az alkalmazásban előforduló specifikus hibák kezelésére szolgálnak. Segítenek az üzleti logikában felmerülő problémák vagy az adatbázis műveletek során jelentkező kivételek egységes kezelésében. Két osztály tartozik ide:

* **AppException:** Ez egy újrahasználható Exception osztály, amely a RunTimeException leszármazottja. HTTP státuszt, valamint hibaüzenetet fogad be, és minden kivételnél meghívható, amikor az üzleti logika szerint valamilyen kérésre hiba választ kell adni.
* **GlobalExceptionhandler:** Globálisan kezeli az alkalmazásban keletkező kivételeket. @ControllerAdvice annotációval rendelkezik, amely lehetővé teszi a kivételek központi kezelését a teljes Spring alkalmazásban. AppException típusú kivételeket keres, és mikor kivétel képződik, biztosítja, hogy a kivétel megfelelően továbbításra kerüljön a front-end felé.

**Dependency Injection:** A rétegek közötti megfelelő adatátvitelt a Spring Depency Injection valósítja meg. Ennek lényege, hogy a Spring konténer kezeli az alkalmazásban lévő objektumok (Bean-ek) életciklusát és függőségeit. Amikor egy osztálynak szüksége van egy másik osztályra, azt az @Autowired annotációval kell jelezni, a Spring pedig megtalálja és injektálja a megfelelő példányt. Az újabb Spring verzióval rendelkező projektekben, mint amilyen ez is, már a @Autowired annotáció explicit kiírása nélkül automatikusan is megtörténik a Dependency Injection.

## 5.3. Front-end

Az alkalmazás front-end rétegének fejlesztéséhez az Angular keretrendszert használtam. Először új projektet hoztam létre: a parancssorban elnavigáltam a célkönyvtárba, majd az ng new InternshipRegistryFrontend paranccsal új angular fájlt hoztam létre, ezután pedig a code. paranccsal megnyitottam Visual Studio Code környezetben. Itt telepítettem a BootStrap könyvtárat az npm install BootStrap @latest paranccsal, és a lokálisan létrejött BootStrap könyvtár helyét behivatkoztam az angular.json Styles listájába. Legeneráltattam az enviroment fájlokat, ahova behivatkoztam a back-end szerver elérhetőségét az ApiURL változóba. Ezt követően már minden készen állt a front-end elkészítéséhez.

### 5.3.1. Struktúra

A klasszikus Angular struktúrát követve, 3 rétegben hoztam létre a front-end megvalósítást, azonban egy külön mappában gyűjtöttem össze az authentikációhoz használt Service és Guard fájlokat.

**AuthService:** A felhasználó autentikációját végzi, a LoginComponent hívja meg és a back-end-del kommunikálva hitelesíti a felhasználót és session cookieval tér vissza, amelyet a böngésző tárol, és automatikusan küld minden HTTP request-hez. Az AuthService kezeli a kijelentkezést is, localRepository-ban tárolja a felhasználó jelszavát és jogosultságait (a jogosultságokat egy tömbben tárolja a Service, mert az admin felhasználóknak két jogosultsága is van), amelyeket kijelentkezéskor töröl. Az AuthService segítségére létrehoztam az autentikációs role-ok kezeléséhez egy enum fájlt, valamint a back-end autentikációs válasszal megegyező felépítésű ResponseEntity interfészt.

**Guard fájlok:** ezek a fájlok a felhasználói jogosultság alapján védik az front-end erőforrásokat. A routes.ts fájlban kell megadni minden címhez, hogy milyen védelemmel rendelkezik.

* **AuthGuard:** Azt ellenőrzi, hogy a felhasználó be van-e jelentkezve, ha nincs visszairányítja a bejelentkezési felületre.
* **AdminGuard:** Az admin jogosultságot igénylő Component-eket védi.
* **CoordinatorGuard:** A koordinátorok által használt felületeket védi.
* **StudnetGuard:** Mivel csak a hallgatóknak van saját szakmai gyakorlata, ezért biztosítja, hogy a félreértések és hibaüzenetek elkerülése végett csak a felhasználók érjék el a nekik szánt felületeket.

**Routing:** Az Angular-ban minden Component-et be kell illeszteni a Routing.ts fájlba egy elérési úttal együtt például:

{ path: 'home', component: HomeComponent, canActivate: [AuthGuard] },

Az adott Component-et ez után lehet elérni a front-end kiszolgálón. Az AuthGuard hozzáadásával biztosítjuk a jogosultsági korlátozást, itt például a HomeComponent-et csak az autentikált felhasználók érhetik el.

**Service-ek:** a back-end-del való kommunikációt valósítják meg HTTP protokollon keresztül. Kialakításuk a back-end Controllerek struktúrájával egyezik meg.

* **UserService:** Admin számára a felhasználó kezeléséhez szükséges kérésekért felel.
* **StudentService:** Koordinátor számára a hallgatók kezeléséhez szükséges kérésekért felel.
* **CompanyService:** Koordinátor számára a cégek kezeléséhez szükséges kérésekért felel.
* **InternshipService:** Koordinátor számára a szakmai gyakorlatok kezeléséhez szükséges kérésekért felel.
* **DocumentService:** Dokumentumok elküldéséhez és fogadásához szükséges kéréseket kezeli.
* **UserSelfService:** A felhasználók saját adataik megtekintéséhez és módosításához szükségeskéréseket kezeli.
* **StudentSelfService:** A hallgató saját szakmai gyakorlatához kapcsolódó kéréseket kezeli.
* **DateConversionService:** A TypeScript-ben definiált Date változó formátuma eltér az ISO8601dátumformátumtól, így a JSON formátumban megkapott dátumértékeket előbb át kell konvertálni, hogy a front-end dolgozni tudjon vele. Küldés előtt vissza kell konvertálni ISO formátumra a back-end-nek

**Model objektumok:** Teljes egészében megegyeznek a DTO-k kialakításával a front-end környezetben, az egyetlen különbség, hogy TypeScript típusok felelnek meg a Java adattípusoknak. Feladatuk a Component-ek és Service-ek közötti adatátvitel.

**Component objektumok:** Az alkalmazás funkciói alapján vannak orientálva, minden felülethez, vagy funkcióhoz tartozik egy Component. A Componentek egy HTML és egy TypeScript fájlt tartalmaznak (esetleg hozzájuk adható CSS, de ez BootStrap-et használva kevésbé releváns). A Component-ek az index.html fájlba töltődnek be a felhasználói interakció és a Routing.ts alapján, így egy SPA-t (Single Page Application) megvalósítva. Speciális Component az AppComponent, amely a gyökér szintű Component, és az Entry Point a Compent-ekbe. A Component-ek alapvető szükségletek az Angular-ban (és a modern webfejlesztésben általánosan), mivel dinamikus adatmegjelenítést valósítanak meg a HTML fájlokban (pl.: \*ngIF, \*ngFor), miközben a TypeScript fájlokban alakíthatóak a kezelt adatok. Az alkalmazás Component fájljairól részletesen a következő részben írok, amelyben kifejtem az alkalmazás funkcióit.

## 5.4. Az alkalmazás funkciói

Végül a program felépítése után bemutatnám az alkalmazás által megvalósított funkciókat.

### 5.4.1. Bejelentkezés

Minden felhasználónak be kell jelentkeznie az alkalmazás használatához. Regisztrációs felület nincs és nem is szükséges mivel ez egy zárt egyetemi nyilvántartó alkalmazás terve, így kezdetben elegendő egy admin jogosultságú felhasználó, aki hozzáadja a koordinátorfelhasználókat, és ők pedig a hallgatókat a rendszerhez.

A bejelentkezési felületen jelszó és felhasználónév beírása után be lehet jelentkezni. Amikor a felhasználók először navigálnak a frontend oldalára, bármely linkről automatikusan a bejelentkezési felületre lesznek irányítva.

Ezért az funkcióért a LoginComponent felelős, amely meghívja az AuthService-t a bejelentkezés gombra kattintáskor. Hibás bejelentkezési adatok esetén az oldal kiírja, hogy: Hibás felhasználónév vagy jelszó.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*4. ábra: Alkalmazás bejelentkezési felület*

### 5.4.2. Navigációs sáv

Bejelentkezés után a felhasználók az üdvözlő nyitólapra lesznek irányítva, innen egy fejlécen keresztül érhetik el a profiladataikat és a nyilvántartó alkalmazás további funkcióit. A fejléc minden jogosultság esetén más, és csak a felhasználó által elérhető funkciókat jeleníti meg. Ez az admin-nak a funkciók teljes listája, míg a hallgatóknak csak a saját szakmai gyakorlatuk megtekintése elérhető. Az AuthService jogosultság lekérő metódusai teszik ezt lehetővé. A fejléc jobb szélén azonban minden jogosultsági körnél elérhető a kijelentkezés gomb.

A fejléc minden lapon megjelenik, mivel bele került az alábbi sor az AppComponent fájlba: <app-header \*ngIf="showHeader"></app-header>

A showHeader feltétel a fejléc bejelentkezési oldalon való elrejtéséért felelős. A fejlécet HeaderComponent valósítja meg, amely routerlinkeket használ a hivatkozások megjelenítéséhez, és injektálja a Router és AuthService fájlokat.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.*5. ábra: Alkalmazás - Admin navigációs fejléce*

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*6. ábra:Alkalmazás - Hallgató navigációs fejléce*

### 5.4.3. Saját adatok megtekintése és frissítése

Az alkalmazásban a bal felső sarokban a saját nevükre kattintva a felhasználóknak lehetősége van a saját adataik megtekintésére. Itt megjelennek a fehasználó adatai, és lehetőség van a jelszó, valamint a felhasználónév megváltoztatására. Ezért a felületért a ProfileComponent felel, amely a UserSelfService-t használja az adatok megjelenítésére. A kérés back-end oldalon pedig felhasználó session adatai alapján módosítja a megfelelő felhasználót az adatbázisban.

### 5.4.4. Felhasználók kezelése

A felhasználókezelés admin funkció. Csak az adminok láthatják és módosíthatják a felhasználók adatait, elfelejtett jelszó esetén is az adminok tudnak új jelszót beállítani a felhasználónak. A 3 komponens tartozik hozzá, amelyek mind a UserSevice-t használják a kérések továbbítására a back-end felé. Felhasználókezelésen belül a következő 3 komponens valósít meg 3 funkciót:

* **UserListComponent:** A felhasználók megjelenítéséért felelős. Egy listában csökkenő sorrendbe rendezi a felhasználókat az ID mező alapján. Minden felhasználó mellett található egy módosítás gomb, alul pedig egy új felhasználó létrehozása gomb.
* **UserCreateComponent:** A felhasználók létrehozásáért felelős. Csak admin és koordinátor felhasználókat lehet létrehozni, mivel a hallgatókat egyben kell létrehozni a hallgató adatokkal. Éppen ezért külön komponens menedzseli a hallgatók létrehozását. Egy Form-on keresztül kell a felhasználók adatait feltölteni, és a végén, alul a létrehozás gombra kattintani. Ezután egy üzenet jelenik meg a képernyő alján, amely tájékoztat a sikeres vagy sikertelen létrehozásról. A művelet megszakítható a vissza gomb megnyomásával az oldal alján, ilyenkor visszakerülünk a felhasználók lista nézetére.
* **UserComponent:** Egy felhasználó adatainak megjelenítéséért és frissítéséért felelős. Az elérési útjának tartalmaznia kell egy felhasználónevet, különben nem elérhető a lap. Egy Form-on keresztül betöltődnek a felhasználó adatai és bármelyik mező szabadon frissíthet. Amennyiben nem felelne meg a követelményeknek a beírt adat, abban az esetben Form adott mezője alatt azonnal ki is írja, hogy mi nem felel, meg és mik az elvárások. A felületen a felhasználó frissítése és vissza a listára gombok elérhetők, itt is tájékoztatást kapunk frissítés esetén, hogy sikeres volt-e a művelet.

A képen szöveg, képernyőkép, szám, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*7. ábra: Alkalmazás – felhasználók listázása*

### 5.4.5. Hallgatók kezelése

A hallgatók kezelését mind az admin, mind a koordinátor felhasználók végezhetik. Itt is 3 komponens érhető el, amelyek a StudentServicet-t használják az adatok lekérésére, a back-end-től. Felépítése szinte megegyezik az felhasználók felületével, kis dizájnbeli különbséggel, hogy meg lehessen különböztetni az admin funkciókat a többitől.

* **StudentListComponent:** A hallgatók listázásáért felelős. Kialakítása megegyezik a felhasználók kezelése felület azonos funkciót megvalósító felületével.
* **StudentCreateComponent:** A hallgatók létrehozásáért felelős. Kialakítása megegyezik a felhasználók kezelése felület azonos funkciót megvalósító felületével, új hallgató létrehozásakor csak a hallgató táblában feltüntett adatokat szükséges megadni, mert a felhasználónév és a jelszó automatikusan generálódik, felhasználónév: neptunkód, jelszó: vezetéknév + neptunkód, valamint a felhasználói role automatikusan hallgató lesz.
* **StudentListComponent:** A hallgatók létrehozásáért felelős. Az elérési útnak tartalmaznia kell egy létező neptunkódot, különben hibaüzenet kapunk. Kialakítása megegyezik a felhasználók kezelése felület azonos funkciót megvalósító felületével.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*8. ábra: Alkalmazás – hallgatók létrehozása*

### 5.4.4. Cégek kezelése

A cégek kezelését szintén admin és koordinátor felhasználók végezhetik, 3 komponens alkotja, és a CompanyService-t használja az adatok lekérésére a back-end-től.

* **CompanyListComponent:** A cégek listájának megjelenítéséért felelős. A kialakítása megyezik az előző listázó felületekével.
* **CompanyCreateComponent:** Egy cég létrehozásáért felelős. Kialakítása megegyezik az előző létrehozó felületekével.
* **CompanyComponent:** A cégek megjelenítéséért felelős, az elérési útjának tartalmaznia kell egy létező cég ID-t, kialakítása szintén megegyezik, az előző frissítő felületekével.

### 5.4.4. Szakmai gyakorlatok kezelése

A szakmai gyakorlatokat kezelő funkció némileg elér az előző funkcióktól, mivel ez a leglényegesebb funkció az alkalmazásban. Nem csak az InternshipService-t használják a funkció komponensei, hanem a DocumentService-t is, így minden gyakorlatnál a részletek megjelenítése után kezelhetők a dokumentumok is. A funkció itt 4 komponensből áll, a listázás, létrehozás, és frissítés felület ugyanúgy létezik, de kiegészül egy szűrés funkcióval, amivel lehetséges a gyakorlati lista szűrése év szerint, vagy aszerint, hogy elvégezték-e már a gyakorlatot.

* **InternshipSelectComponent:** Egy egyszerű oldal két legördülő listával. Az egyik az éveket tartalmazza, a másik pedig a befejezett vagy nem befejezett lehetőséget, ezen kívül mindkét lista tartalmaz egy ’összes’ opciót, amellyel csak az egyik paraméter alapján, vagy akár az összes gyakorlatra is rá lehet keresni. Alapértelmezett értékei az aktuális év, és nem teljesített gyakorlatok.
* **InternshipListComponent:** A szakmai gyakorlati lista megjelenítéséért felelős. Elérési útjában két paramétert vár: az évet, és hogy milyen státuszban van a gyakorlat. Amennyiben a navigációs fejlécről próbáljuk elérni a funkciót, úgy alapértelmezetten az összes gyakorlatot listázza ID mező alapján csökkenő sorrendben. Minden rekord jobb szélén van egy módosítás gomb, alul pedig egy új gyakorlat létrehozása gomb.
* **InternshipCreateComponent:** A szakmai gyakorlatok létrehozásáért felelő, megegyezik az előző oldalak létrehozási felületével.
* **InternshipComponent:** Egy szakmai gyakorlat megtekintését és módosítását teszi lehetővé. Az elérési útjában vár egy létező szakmai gyakorlat id-t. Egy Form-ban megjeleníti a szakmai gyakorlat adatait, amelyeket módosítani lehet. Egyedül a hallgató nem módosítható, akihez a szakmai gyakorlat rendelve van. Frissítés gombbal menthetőek a módosítások. Amennyiben jegy lett beírva a gyakorlatra, úgy a befejezett státusz automatikus igen-re vált. A Form alatt láthatóak a feltöltött dokumentumok. Ammenyiben még nem lett feltöltve egy dokumentum sem, akkor a nincs még dokumentum felirat olvasható. A listázott fájlok mellett egy letöltés és egy törlés gomb található. A dokumentumok listázása alatt egy fájlválasztó ablak látható, mellette egy felöltés gombbal. A dokumentumok csak pdf és docx formátumúak lehetnek, ezt a back-end logikában az Apache Tika ellenőrzi.

A képen szöveg, képernyőkép, szám, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*9. ábra: Alkalmazás – fájlok kezelése egy szakmai gyakorlatnál*

### 5.4.5 Hallgató saját szakmai gyakorlatának lekérdezése

A hallgatóknak a navigációs sávon egy felület érhető el, amely 2 komponenst tartalmaz. Amennyiben van szakmai gyakorlatuk, abban az esetben, amikor a hallagató rákattint a linkre, a szakmai gyakorlati adatait tudja megtekinteni. A szakmai gyakorlatokat módosítani nem tudja, viszont dokumentumokat tud feltölteni a gyakorlathoz, valamint a meglévő dokumentumokat letölteni és törölni. Amennyiben még nem rendelkezik a hallgató szakmai gyakorlattal, úgy a felület egy linket fog megjeleníteni, amelyre kattintva egy létrehozási felületre kerül a hallgató. Itt létre tudja hozni a szakmai gyakorlatát, azonban néhány adatot, mint a jegy, az igazolás megszerzésének dátuma, vagy a befejezett-e mező, nincs jogosultsága beállítani.

* **StudentInternshipComponent:** A UserSelfService-t használja. A back-end-től az autentikációs adatai alapján tudja elérni a saját szakmai gyakorlatát, amelyet aztán megtekinthet, és a hozzá tartozó dokumentumokat kezelheti.
* **StudentInternshipCreateComponent:** Szintén a UserSelfService-szel áll kapcsolatban. A szakmai gyakorlat létrehozásának egy korlátozott változatát éri el, amely a back-end logikában egyszerűen nem veszi át a beérkező adatok közül azokat, amelyeket a hallgató nem jogosult módosítani, így megőrizve a gyakorlati adatok biztonságát.

## 5.5 Az alkalmazás funkcionális tesztelése

Bár automata végpont-végpont (end-to-end) teszteket nem implementáltam a szakdolgozat keretein belül, manuális funkcionális teszteléssel ellenőriztem az alkalmazás felhasználó felőli viselkedését. Ez a tesztelési fázis a felhasználói elvárások és specifikációk alapján történt, és az alábbi területekre terjedt ki:

* **Felhasználói szerepkörök:** A rendszerben a különböző felhasználói szerepkörök jogosultságainak és hozzáférési szintjeinek ellenőrzése: minden szerepkör csak a megengedett funkciókat érje el.
* **CRUD műveletek:** Az összes entitás létrehozásának, lekérdezésének, módosításának és törlésének validálása a felhasználói felületen keresztül.
* **Navigáció és űrlapok:** Az Angular alkalmazás navigációjának, a különböző oldalak közötti átmeneteknek, valamint az űrlapok validációjának és beküldésének tesztelése.
* **Hibakezelés:** A GlobalExceptionHandler által kezelt, alkalmazásszintű hibák megfelelő megjelenésének ellenőrzése a felhasználói felületen. Például, ha egy adatbázis művelet során hiba történik, a felhasználó megfelelő, értelmezhető üzenetet kapjon.

## 5.6. Felhasználói élmény tesztelés

A szakmai gyakorlat nyilvántartó programot egy kollégám segítségével leteszteltem, aki még nem látta előtte a programot. Adtam hallgatói egy bejelentkezéshez, szükséges jelszót és felhasználónevet, és megkértem, hogy hozzon létre saját szakmai gyakorlatot, majd tekintse azt meg, töltsön fel hozzá fájlokat. Majd egy admin fiók bejelentkezési adatait adtam meg neki, és megkértem, hogy hozzon létre új hallgatót, új céget, és egy új szakmai gyakorlatot a hallgatónak a céggel. Visszajelzései alapján az alkalmazás intuitív és egyszerűen használható, viszont a dizájnt illetőleg lehetne még fejleszteni. Ez teljes mértékben igaz, csak egy alapvető, egyszerű BootStrap készítettem, amely még tovább finomítható, de a funkcionalitás, fontosabb része a programnak. Előfordultak még helyenként kimaradt alkalmazás üzenetek a feltöltésről és frissítésről, ami valószínűleg programozói figyelmetlenség miatt történt.

Összességében még találhatunk fejlesztendő területeket az alkalmazásban, de a teszt felhasználó képes volt az alkalmazás használatára probléma nélkül.

# 6. Összegzés tapasztalatok

A szakmai gyakorlati nyilvántartó rendszer egy adminisztratív szoftver, melynek elsődleges célja a szakmai gyakorlati adatok hatékony tárolása és kezelése. A fejlesztés során a program sikeresen teljesítette ezt az alapvető elvárást, megalapozva a rendszer funkcionalitását. Azonban a jövőbeli fejlesztések elengedhetetlenek a felhasználói élmény javításához és az adminisztratív folyamatok optimalizálásához.

## 6.1. Fejlesztés során felmerülő nehézségek

A fejlesztés során jelentős kihívást jelenthetett a háromrétegű architektúra (frontend, backend, adatbázis) összehangolása, a Spring Boot és Angular keretrendszerek specifikus konfigurációja, különösen a biztonsági (Spring Security) és kommunikációs (CORS) beállítások terén. A legnagyobb kihívást az autentikációs folyamat összehangolása jelentette a back-end és front-end között. Az Angular tanulási görbéje viszonylag meredek ezért, habár volt már tapasztalatom a keretrendszerrel, nem volt egyszerű az alkalmazás megvalósítása.

## 6.2. Továbbfejlesztési lehetőségek

A rendszer további fejlesztése a következő területeket érintheti:

Az Excel export és import funkció hiánya jelenleg a legfontosabb továbbfejlesztendő terület. Ennek a funkciónak az implementálása jelentősen megkönnyítené a nagy mennyiségű adat bevitelét és kinyerését, csökkentve a manuális adatkezelési terhet és a hibalehetőségeket. Ez az adminisztratív szoftverek elengedhetetlen része.

Az adatszűrési és -rendezési lehetőségek kiterjesztése nagyban javíthat az alkalmazás használatának egyszerűségén. A fejlettebb szűrési és rendezési funkciók lehetőséget adnak összetett lekérdezésekre is, így biztosítva a keresett adatok gyorsabb megtalálását nagyobb adatlisták esetén.

A felhasználói felület modernizálása és a felhasználói élmény (UX) javítása is segítheti a rendszer megítélését. Egy intuitívabb, esztétikusabb és reszponzív dizájn növelné a rendszer használhatóságát és elfogadottságát a felhasználók körében. Ez magában foglalja a vizuális elemek frissítését, és a különböző eszközökön való egységes megjelenés biztosítását.

# 7. Summary

The internship registry system is an administrative software, and its primary purpose is to efficiently store and manage internship data. During its development, the software has successfully met this basic requirement, establishing the functionality of the system. However, future enhancements are essential to improve the user experience and to optimize the administrative processes.

## 7.1. Difficulties encountered during developement

During development, a major challenge was the alignment of the three-layer architecture (front-end, back-end, database), the specific configuration of the Spring Boot and Angular frameworks, especially in the security (Spring Security) and communication (CORS) settings. The biggest challenge was the alignment of the authentication process between the back-end and front-end. The learning curve of Angular is relatively steep so, although I had some experience with the framework, it was not easy to implement the application.

## 7.2. Further development opportunities

Further development of the system could address the following areas:

The lack of Excel export and import functionality is currently the most important area for further development. The implementation of this function would significantly facilitate the input and extraction of large amounts of data, reducing the manual data management burden and the potential for errors. This is an essential part of administrative software.

Extending the datafiltering and sorting capabilities of the application can greatly improve the ease of use. The advanced filtering and sorting functions also allow for complex queries, ensuring thatdata can be found quickly in larege data sets..

Modernisation of the user interface and improvement of the user experience (UX) could also help to improve the perception of the system. A more intuitive, aesthetic and responsive design would increase the usability and acceptance of the system by users. This includes updating visual elements and ensuring a consistent look and feel across different devices.

# 8. Irodalomjegyzék

[1] Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE access*, *10*, 20357-20374.

[2] IBM. What is three-tier architecture?

<https://www.ibm.com/think/topics/three-tier-architecture>

[3] Intersystems. Mi az a relációs adatbázis és miért van rá szükség?

<https://www.intersystems.com/hu/resources/mi-az-a-relacios-adatbazis-es-miert-van-ra-szukseg/>

[4] Representational State Transfer (REST)

<https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>

[5] Microsoft. Ajánlott eljárások a RESTful webes API-tervezéséhez

<https://learn.microsoft.com/hu-hu/azure/architecture/best-practices/api-design>

[6] Geeks for Geeks. Session vs Token Based Authentication

<https://www.geeksforgeeks.org/session-vs-token-based-authentication/>

[7] Mege. What is the best front end framework in 2025? (Expert breakdown)

<https://merge.rocks/blog/what-is-the-best-front-end-framework-in-2025-expert-breakdown>

[8] MySQL. Why MySQL?

<https://www.mysql.com/why-mysql/>

[9] Geeks for Geeks. Top 10 Most Popular Java Frameworks for Web Development

<https://www.geeksforgeeks.org/top-most-popular-java-frameworks-for-web-development/#1-spring>

[10] Parsons, D. (2020). Java Web Servers and the HttpClient. In *Foundational Java: Key Elements and Practical Programming* (pp. 565-585). Cham: Springer International Publishing.

[11] Gradle. Gradle Build Tool accelerates developer productivity

<https://gradle.org/>

[12] JetBrains. IntelliJ IDEA The IDE for Pro Development

<https://lp.jetbrains.com/intellij-idea-promo/>

[13] Adam Freeman. (2022). Pro Angular, Build Powerful and Dynamic Web Apps. APress Media LLC

[14] BootStrap. Build fast, responsive sites with Bootstrap

<https://getbootstrap.com/>

[15] Visual Studio Code. Your code editor. Redefined with AI.

<https://code.visualstudio.com/>

Hivatkozások ellenőrzésének utolsó dátuma: 2025.05.15.

# 9. Ábrajegyzék

1. ábra: Use-case diagram 15

2. ábra: Adatbázis ER modell 16

3. ábra: 3. ábra: Adatbázis relációs modell 17

4. ábra: Alkalmazás bejelentkezési felület 38

5. ábra: Alkalmazás - Admin navigációs fejléce 39

6. ábra: Alkalmazás - Hallgató navigációs fejléce 39

7. ábra: Alkalmazás – felhasználók listázása 41

8. ábra: Alkalmazás – hallgatók létrehozása 42

9. ábra: Alkalmazás – fájlok kezelése egy szakmai gyakorlatnál 43