



EÖTVÖS LORÁND TUDOMÁNYEGYETEM  
INFORMATIKAI KAR  
INFORMATIKATUDOMÁNYI INTÉZET  
INFORMÁCIÓS RENDSZEREK TANSZÉK

## Munkaerő-felvételt menedzselő webalkalmazás

*Témavezető:*  
Varga Dániel  
egyetemi adjunktus

*Szerző:*  
Solti Martin  
programtervező informatikus BSc

Budapest, 2024

# EÖTVÖS LORÁND TUDOMÁNYEGYETEM

## INFORMATIKAI KAR

## SZAKDOLGOZAT TÉMABEJELENTŐ

### Hallgató adatai:

Név: Solti Martin

Neptun kód: BESZN1

### Képzési adatak:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Belső témavezetővel rendelkezem

Témavezető neve: Varga Dániel

munkahelyének neve, tanszéke: ELTE IK, Információs rendszerek Tanszék

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.

beosztás és iskolai végzettsége: Adjunktus, doktori fokozat

A szakdolgozat címe: Munkaerő-felvételt menedzselő webalkalmazás

### A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

A szakdolgozat célja egy olyan webalkalmazás elkészítése React és ASP.NET keretrendszer felhasználásával, amely lehetővé teszi  
- az alkalmazást felhasználó vállalat számára nyitott pozíciók meghirdetését, jelentkezések kezelését, az interjük videohívásban történő  
lebonyolítását, valamint  
- külsős felhasználók számára - a regisztrációt követően - a meghirdetett pozícióra történő jelentkezést.

A vállalat regisztrálni tudna különböző jogosultságokkal rendelkező, recruiter és menedzser felhasználókat. A menedzser felhasználók a  
vállalon belüli projektek élén állnak, akiknek joguk lenne kérવényezni egyes pozíciók meghirdetését, valamint a projektükhez tartozó  
bemutatkozó, egy regisztráció nélkül megtekinthető wiki oldalt szerkeszteni. A recruiter felhasználók hirdethetnék meg az egyes, pontosan  
egy projekt alá tartozó pozíciókat, illetve továbbíthatnák a jelentkezéseket a projekt menedzser felé. Emellett jogosultak lennének  
különböző kulcsszavak regisztrálására, amelyeket meghirdetett pozícióhoz is rendelhetnek (az adott pozícióra és annak betöltéséhez  
szükséges képességeket leíró szavak, egy IT vállalatnál pl. C#, Agilis, Scrum stb.).

Külsős felhasználóknak regisztrációt követően lehetőségük lenne jelentkezni a nyitott pozíciókra, valamint saját profiljuk szerkesztésére.  
Ez utóbbi funkcionálitás lehetőséget biztosítana önéletrajz feltöltésére is, amelyből a rendszer megpróbálja a regisztrált kulcsszavakat  
automatikusan detektálni, majd ezeket a profilhoz rendelné. A felhasználó saját magához is tud rendelni további kulcsszavakat. Ezen  
kulcsszavakat a rendszer felhasználhatja javaslattételre. A jelentkezést követően, amennyiben a vállalat úgy dönt, a felhasználó kaphat  
videointerjúra történő meghívást, melybe a megadott időpontban be tudna csatlakozni.

Budapest, 2023. 11. 21.

# **SZAKDOLGOZAT / DIPLOMAMUNKA**

## **EREDETISÉG NYILATKOZAT**

Alulírott .....SOLTI MARTIN..... Neptun-kód: BESZN1.....

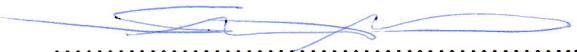
ezennel kijelentem és aláírásommal megerősítem, hogy az Eötvös Loránd Tudományegyetem Informatikai Karának, INFORMÁCIÓS RENDSZEREK..... Tanszékén írt, MUNKAKERÉD - FELVETELETT MÉNIEDZSELŐ VERBALKALMAZÁS.....

című szakdolgozatom/diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.

Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Budapest, 2024. 05. 12.

  
.....  
*hallgató aláírása*

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>4</b>
<b>2. Felhasználói dokumentáció</b>	<b>5</b>
2.1. Az alkalmazás rövid ismertetése . . . . .	5
2.2. Rendszerkövetelmények . . . . .	6
2.3. Telepítés . . . . .	6
2.3.1. Előzetes lépések . . . . .	6
2.3.2. Az alkalmazás kihelyezése . . . . .	8
2.4. Az alkalmazás általános használata . . . . .	8
2.4.1. A félület . . . . .	8
2.4.2. Felsoroló kártyák . . . . .	9
2.4.3. Visszajelzések . . . . .	10
2.5. Vállalati entitások regisztrálása . . . . .	13
2.6. Pozíciók meghirdetésének folyamata . . . . .	15
2.7. Candidate felhasználó regisztrálása . . . . .	20
2.8. Jelentkezés és elbírálásának folyamata . . . . .	22
2.8.1. Jelentkezés nyitott pozícióra . . . . .	22
2.8.2. Jelentkezés kezelése a vezérlőpulton . . . . .	24
2.8.3. Meghívás videóinterjúra . . . . .	25
2.9. Videóinterjú . . . . .	27
<b>3. Fejlesztői dokumentáció</b>	<b>31</b>
3.1. A megoldandó feladat . . . . .	31
3.1.1. A felhasználói felületre vonatkozó elvárások . . . . .	31
3.1.2. A szerveroldali komponensre vonatkozó elvárások . . . . .	32
3.2. Fejlesztői környezet . . . . .	32
3.3. Az alkalmazás rétegei és felelősségi köreik . . . . .	33
3.3.1. A modell réteg . . . . .	34

3.3.2. Az applikáció réteg . . . . .	35
3.3.3. Az infrastruktúra réteg és annak kapcsolata az applikáció réteggel . . . . .	35
3.3.4. A prezentáció: a web- és kliens rétegek . . . . .	36
3.4. Perzisztencia . . . . .	36
3.5. Autentikáció . . . . .	40
3.6. Parancsok és lekérdezések . . . . .	41
3.6.1. CQRS-minta . . . . .	41
3.6.2. Hibakezelés, utasítások visszatérési értéke . . . . .	42
3.6.3. Entitások sémái . . . . .	43
3.6.4. Absztrakt végrehajtó ősosztályok . . . . .	45
3.6.5. Pipelineok . . . . .	47
3.7. A szerveroldal belépési pontja . . . . .	48
3.7.1. Vezérlők . . . . .	48
3.7.2. HTTP kérések és válaszok felépítése . . . . .	49
3.8. Videóhívás . . . . .	51
3.8.1. Megvalósítás kliensoldalon . . . . .	51
3.8.2. Megvalósítás szerveroldalon . . . . .	51
3.9. Kliens . . . . .	53
3.9.1. A felület tervezése . . . . .	53
3.9.2. Állapotkezelés . . . . .	57
3.9.3. Oldalak . . . . .	57
3.9.4. Közös komponensek . . . . .	58
3.9.5. HTTP kérések kezelése . . . . .	59
3.10. Tesztelés . . . . .	60
3.10.1. Automatikus egységtesztek . . . . .	60
3.10.2. Alkalmazás manuális tesztelése . . . . .	63
3.10.3. Tanulságok . . . . .	69
3.11. Továbbfejlesztési lehetőségek . . . . .	69
<b>4. Összegzés</b>	<b>71</b>
<b>Köszönetnyilvánítás</b>	<b>72</b>
<b>Irodalomjegyzék</b>	<b>72</b>



## 1. fejezet

# Bevezetés

Egy szoftver fejlesztése során a tervezési fázis a legmeghatározóbb. Élettartamának egy jelentős részét karbantartása, továbbfejlesztése teszi ki, így kiemelten fontos, hogy a tervezés során ügyeljünk a kód bővíthetőségre, modularizáltságára. Nagyobb alkalmazások esetén, esetleg több fejlesztő közös munkája során ezek a szempontok még lényegesebbé válnak. Elengedhetetlen tehát, hogy egy olyan architektúrát válasszunk, amely ezeknek a feltételeknek eleget tesz.

A clean architecture irányelvei mentén történő webalkalmazás fejlesztés az utóbbi időben egyre népszerűbb, így az én figyelmemet is felkeltette. Ez az architektúra arra vonatkozóan fogalmaz meg ajánlásokat, hogy egy nagyobb alkalmazást milyen komponensekre, milyen felelősségi körök mentén bontsunk fel, biztosítva a kód tesztelhetőségét, bővíthetőséget és újrafelhasználhatóságát. Ezt az architektúrát gyakran alkalmazzák ASP.NET alapú API-k fejlesztése során. Az ASP.NET az egyik legnépszerűbb webfejlesztési keretrendszer, és mindig is szerettem volna megtanulni a használatát, mivel érdekelte, miként lehet modern webalkalmazásokat fejleszteni a .NET ökoszisztemában, C# nyelven, az objektumorientált programozás elveit követve. Így a választásom erre a keretrendszerre esett.

Szakdolgozatom témája egy olyan webalkalmazás tervezése és fejlesztése, amely a munkaerő-felvétel folyamatainak menedzselését végzi az azt felhasználó vállalat számára. Az alkalmazás fejlesztését a clean architecture irányelveinek követésével valósítom meg, ASP.NET és React keretrendserek felhasználásával.

## 2. fejezet

# Felhasználói dokumentáció

### 2.1. Az alkalmazás rövid ismertetése

Az Employer egy olyan webalkalmazás, amely az azt felhasználó informatikai vállalat számára felületet biztosít nyitott pozíciók meghirdetésére, illetve ezen pozíciókra történő jelentkezések kezelésére, elbírálására.

Nagyobb vállalatok esetében nem ritka, hogy az egyes pozíciók különböző részlegek, szervezeti egységek alá vannak szervezve. Az alkalmazásban ezt a szerepet a projektek töltik be, melyek élén egy menedzszer felhasználó áll. A pozíciók meghirdetését a menedzserek kezdeményezhetik pozíciókiírási kérelem megfogalmazásával. A pozíciókat a vállalat recruiter felhasználói írják ki a kérelmek alapján, a menedzsér jóváhagyásával. Később lehetőség van a pozíciók lezárására, melynek következtében az nem fogad több jelentkezést.

A nyitott pozíciókra a regisztrált candidate felhasználóknak van lehetőségük jelentkezni. A jelentkezéseket egyenként a recruiterek kezelik, akik a jelentkező profiljának megtekintése után opcionálisan meghívást küldhetnek videointerjúra, majd a jelentkezést elutasíthatják, vagy továbbíthatják a felelős menedzszer felé. A menedzsér végül a recruiter felhasználó beszámolója alapján meghozza a végső döntést.

## 2.2. Rendszerkövetelmények

Az alkalmazás futtatásához illetve telepítéséhez szükséges csomagokat és hardverigényt az alábbi felsorolás tartalmazza.

- .NET SDK 8.0.204 verzió, vagy újabb<sup>1</sup>
- .NET Framework 4.7.2 Developer pack<sup>2</sup>
- Node.js 18.14.2 verzió, vagy újabb<sup>3</sup>
- Entity Framework Core tool
- legalább 1 GB RAM
- legalább 2 GB tárhely
- audió- és videó bemeneti eszközök

Az alkalmazás fejlesztését és tesztelését a saját, személyi számítógépemen végezem. Ennek specifikációi:

- Windows 11 Pro 23H2, 64 bites operációs rendszer
- Intel Core i7-1065G7 processzor
- 8 GB memória

Az alkalmazást Google Chrome böngészőt használva teszteltem.

## 2.3. Telepítés

A dokumentációban a személyi számítógépre történő telepítést fogjuk tárgyalni. Először a telepítés előkészítéséről (szükséges programok telepítése, beállítások elvégzése), majd az ezt követő kihelyezésről lesz szó.

### 2.3.1. Előzetes lépések

Telepítsük a rendszerkövetelményekben felsorolt .NET SDK, .NET Framework és Node.js csomagokat a megadott linkeken található telepítővel. Először a .NET SDK-t telepítsük, majd indítsuk újra a számítógépet! Ezután telepítsük a .NET Framework Developer pack-et, a Node.js-t, végül pedig az Entity Framework Core Tool-t a `dotnet tool install --global dotnet-e`f parancs kiadásával. Az összes telepítés elvégzése után szükség lehet a számítógép újraindítására.

---

<sup>1</sup><https://dotnet.microsoft.com/en-us/download/dotnet/8.0>

<sup>2</sup><https://dotnet.microsoft.com/en-us/download/dotnet-framework/net472>

<sup>3</sup><https://nodejs.org/en>

Hozzunk létre egy mappát a számítógép egy tetszőleges könyvtárában! Az alkalmazást ebbe a mappába fogjuk telepíteni. Például: C:/Employer. Ezután, végezzük el az `Employer.Web/appsettings.json` fájl kitöltését.

- A `ConnectionStrings` alatti `SqliteConnection` kulcsú beállítás értéke az SQLite adatbázis `.db` fájl helyét írja le `Data source=` prefixsel. Az előzőleg példaként vett mappa alapján ez az érték lehet például: `Data source=C:/Employer/storage.db`. A `.db` fájl a telepítés során fog létrejönni az itt megadott útvonalon. Megjegyezzük, hogy amennyiben más (nem a telepítés céljából létrehozott) mappát választunk, a megadott útvonalban szereplő könyvtárak létezését ez esetben is biztosítanunk kell, mivel azok nem, csak a `.db` fájl fog létrejönni.
- Az alkalmazás JSON Web Tokenek (JWT) segítségével valósítja meg az autentikációt. A `TokenSettings` alatt a JWT-ra vonatkozó beállításokat módosíthatjuk. Személyi számítógépre történő kihelyezés esetén az ez alá tartozó beállítások módosítása opcionális. A `Key` értéke egy, a JWT titkosítása során felhasznált, minimum 32 karakter hosszú sztring. Az `Issuer` és az `Audience` a token hitelesítésének egy részét szolgálják, előbbi a token kiállítóját, utóbbi a token célcsoportját beazonosító sztring. Az `Expires` pozitív egész szám, mely a token érvényességét határozza meg, napokban.
- A `RootUserInitSettings` alatt a root felhasználó adatait adhatjuk meg.
  - Az `Email` értékéül valid email formátumú sztringet kell megadni.
  - A `FirstName` és `LastName` legfeljebb 15 karakter lehet.
  - A `Password` legalább 8 karakter kell, hogy legyen.

Fontos kiemelni, hogy a root felhasználó beállításai az inicializálásra vonatkoznak. A fentebb megadott adatokkal az alkalmazás első indításakor fog létrejönni a felhasználó. A létrehozás csak akkor fut le, ha az adatbázis nem tartalmaz egyetlen felhasználót sem (ez a kezdeti állapotban így van). Az inicializált adatok közül a jelszó megváltoztatására lesz lehetőség, az alkalmazás felületén.

### 2.3.2. Az alkalmazás kihelyezése

Az előzetes lépések elvégzése után megkezdhetjük az alkalmazás kihelyezését a korábban erre a célra létrehozott mappába. Ehhez az alább felsorolt parancsokat kell a megfelelő helyen, sorrendhelyesen kiadni.

1. Az forráskód gyökérmapjából (ahol az `Employer.sln` is található) adjuk ki a `dotnet restore` parancsot.
2. A gyökérmapban maradva adjuk ki a `dotnet ef database update -p Employer.Infrastructure -s Employer.Web` parancsot. Ezzel a `.db` fájl a megadott helyen létrejött.
3. Az `Employer.Web` mapról adjuk ki a `dotnet publish -c Release -o [a létrehozott mappa útvonala]` parancsot. Szintén a korábbi példát tekintve, amennyiben például a `C:/Employer` mappába telepítjük az alkalmazást, úgy a kiadott parancs `dotnet publish -c Release -o C:\Employer`

A fentebbi lépések elvégzése után a megadott mappában létrejött az `Employer.Web` futtatható állomány. Ennek futtatásával a program elindul. A megnyílt konzolból (2.1 ábra) leolvasható az alkalmazás böngészőből történő elérésének útvonala.

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT EXISTS (
          SELECT 1
          FROM "Users" AS "u")
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Employer
```

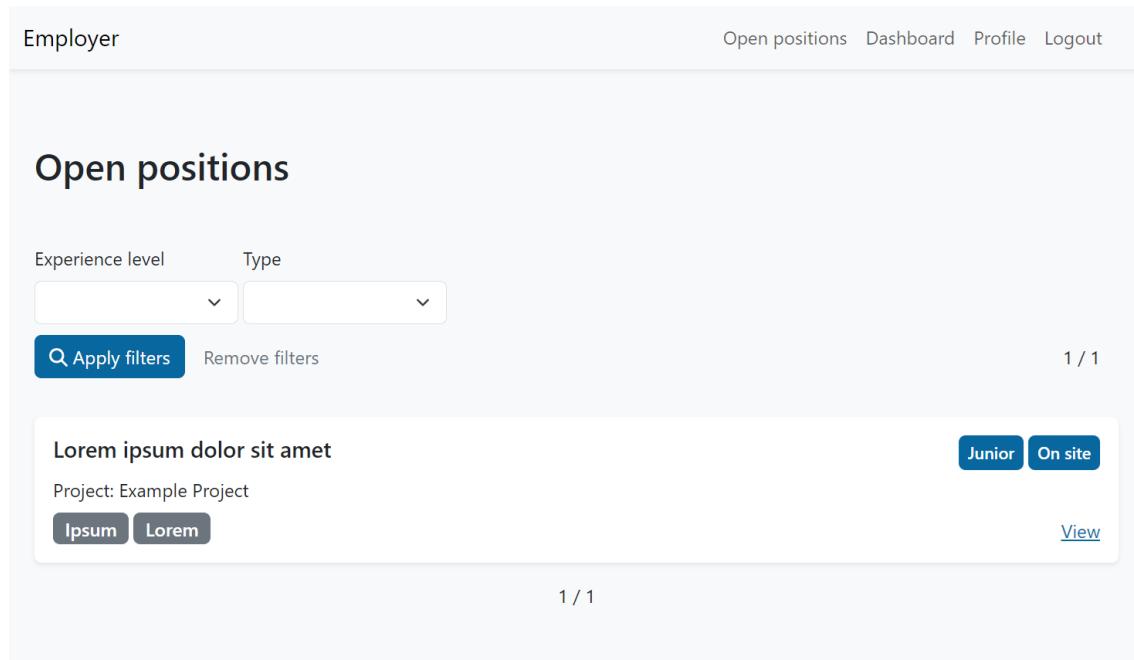
2.1. ábra. Az alkalmazás konzolja annak indítását követően

## 2.4. Az alkalmazás általános használata

### 2.4.1. A felület

Az alkalmazás felhasználói felülete a webalkalmazások tradícióinak megfelelően lett kialakítva. A böngészőben történő megnyitáskor az alkalmazás főoldala (2.2

ábra) fogad minket, ahol a nyitott pozíciók kerülnek kilistázásra. A felső sávban található a navigációs menü, ahol a különböző oldalakat, be- és kijelentkezési lehetőségeket tudjuk elérni.

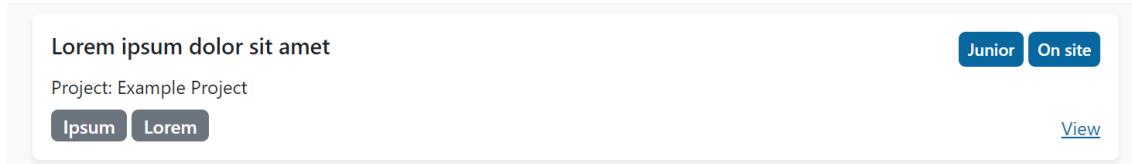


2.2. ábra. Az alkalmazás főoldala

Az aktuális oldal, valamint a navigációs menü tartalma jellemzően a bejelentkezett felhasználó rangjától függ. A felület ezen részeit konkrét funkcionálisok bemutatása mellett, a későbbiekben tárgyaljuk. Ebben a fejezetben a felület használatának általános bemutatására kerül sor.

#### 2.4.2. Felsoroló kártyák

Az alkalmazás használata során sok olyan oldallal találkozunk, amely entitások felsorolását, listázását tartalmazza. Egy entitást egy "kártya" reprezentál, ezek vertikálisan, egymás alatt kerülnek felsorolásra. Jellemzően a bal felső sarokban található az entitás megnevezése, jobb alsó sarokban pedig annak megtekintésére szolgáló, kattintható *View* szöveg. A kártyán az entitás típusától függően egyéb adatok is megjelennek. A 2.3 ábrán egy nyitott pozíciót reprezentáló kártyára látunk példát.



2.3. ábra. Nyitott pozíciót reprezentáló kártya

Ahol indokolt a megjelenített adatmennyiségek korlátozása, ott több oldalon keresztül kerülnek megjelenítésre az adatok. Az egyes oldalak közötti navigációt a felsorolás felett és alatt elhelyezkedő gombcsoporttal lehet vezérelni (2.4 ábra).

A screenshot of a grid view showing three floating cards. At the top left is a search bar with dropdown menus for "Experience level" and "Type". Below it are two buttons: "Apply filters" and "Remove filters". To the right is a navigation bar with arrows and the text "1 / 2".

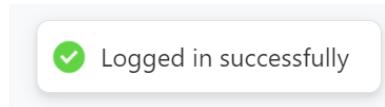
- Card 1:** "Lorem ipsum dolor" (Project: Example Project). Buttons: "Junior", "On site". "View" button.
- Card 2:** "Lorem ipsum dolor sit amet" (Project: Example Project). Buttons: "Medior", "Hybrid". "View" button.
- Card 3:** "Lorem ipsum" (Project: Example Project). Buttons: "Senior", "Home office". "View" button.

The cards have a light grey background with rounded corners. The overall interface is clean and modern.

2.4. ábra. Több oldalon keresztül felsorolt entitások

### 2.4.3. Visszajelzések

Az alkalmazás az egyes műveletek végrehajtásának sikerességéről visszajelzést ad értesítés formájában. Az értesítések a jobb alsó sarokban ugranak fel, erre példát a 2.5 ábrán láthatunk.



2.5. ábra. Példa felugró értesítésre

Hiba esetén is azonos módon kapunk értesítést, például, ha egy olyan műveletet szeretnénk végrehajtani, amelyre az adott pillanatban nincsen jogunk. Természetesen - a megfelelő felhasználói élmény biztosítása érdekében - a felületen csak olyan műveletek végrehajtását kezdeményező vezérlők jelennek meg, amelyek ténylegesen végre is hajthatók az oldal betöltésének pillanatában. Azonban előfordulhat olyan eset, hogy - például - egy másik felhasználó egy olyan műveletet végez a rendszerben, amely következtében az általunk látott - már korábban betöltött - felület nem a naprakész jogosultságainknak megfelelően jelenik meg. Így amennyiben hasonló esetet tapasztalunk, gondoljunk arra, hogy a megnyitott oldal frissítésére lehet szükség.

Felmerülhet a kérdés, hogy a frissítés ilyen esetekben miért nem automatikusan történik. A felhasználói élmény szempontjából zavaró lenne az, hogy hibaüzenetet látunk (jelezve, hogy az adott művelet nem hajtható végre), de közben az oldal állapota megváltozik (mintha mégis végrehajtottunk volna egy műveletet). Továbbá, ha a hibaüzenet megjelenítése után elrejtenénk az oldal korábbi állapotát, a felhasználónak nem lenne lehetősége felnérni, mi váltotta ki a hibaüzenetet.

Kaphatunk hibaüzenetet validációs problémák miatt is, például ha az általunk megadott bemenet formátuma nem megfelelő. Ilyen esetekben a felugró értesítés mellett a konkrét hibáról is információt kapunk az érintett bemeneti mező(k)nél, ahogy ez a 2.6 ábrán is látható.

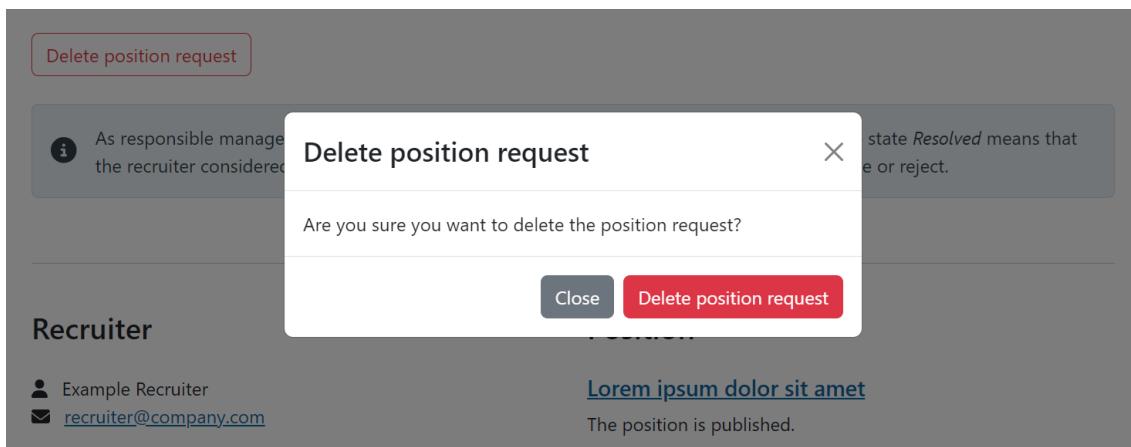
The screenshot shows a registration form titled "Register". The fields and their validation errors are:

- Email:** "example" - Error: "'Email' is not a valid email address."
- First name:** "Example" - No error.
- Last name:** (empty field) - Error: "'Last Name' must not be empty."
- Password:** "\*\*\*\*" - Error: "The length of 'Password' must be at least 8 characters. You entered 4 characters."
- Confirm password:** "\*\*\*\*" - No error.

At the bottom, there is a "Register" button and a link: "Already have an account? Login here!"

2.6. ábra. Példa mezővalidációs hibára

Egyes műveletek végrehajtása előtt a rendszer megerősítést kér. Ezek nagyobb hatású, nem (vagy nem azonnal) visszavonható műveletek esetén jellemzőek. Ilyenkor egy felugró párbeszédablakban (2.7 ábra) kell szándékunkat megerősítünk.



2.7. ábra. Példa megerősítésére szolgáló párbeszédablakra

Most, hogy megismerkedtünk az alkalmazás felületének általános jellemzőivel, a következő fejezetekben rátérünk konkrét funkcionálitások ismertetésére, felhasználói

történetek által meghatározott egy lehetséges sorrendben.

## 2.5. Vállalati entitások regisztrálása

Az alkalmazásba a root jogosultságú felhasználóval belépve regisztrálhatjuk a vállalatot képző alapentitásokat, ilyenek a projektek, a menedzserek és a recruiter felhasználók. Ezen regisztrációkat a vezérlőpulton (*Dashboard*) tudjuk elvégezni.

Recruiter felhasználók regisztrálására a *Recruiters* fül alatt van lehetőség, a jobb felső sarokban található *Add new* gomb megnyomásával. A felugró párbeszédablak (2.8 ábra) kitöltésével létrejön a felhasználó.

2.8. ábra. Recruiter felhasználó regisztrálása

Új projekt felvétele hasonló módon, a *Projects* fül alatt lehetséges. A párbeszédablakban a projekt nevét kell megadnunk. A létrejött projekt kilistázásra kerül (2.9 ábra), egyelőre regisztrált menedzser felhasználó nélkül. Ez utóbbi tényre a *No manager registered yet.* felirat hívja fel a figyelmet. Az emellett található *Register* szövegre kattintva regisztrálhatjuk a projekt felelős menedzserét egy olyan párbeszédablakban, melyet az előző képen is láttunk (2.8 ábra).

The screenshot shows a user interface for managing projects. On the left, there's a sidebar with links: 'Position requests', 'Projects' (which is currently selected and highlighted in blue), 'Recruiters', 'Keywords', and 'Applyments'. The main content area is titled 'Projects' and lists several projects. Each project is represented by a card. The first card is for 'Example Project' and notes that no manager is registered yet, with a 'Register' link and a 'View' link. The second card is for 'Example Project 1', managed by 'Example Manager1', with a 'View' link. The third card is for 'Example Project 2', managed by 'Example Manager2', with a 'View' link. At the top right of the main area, there are 'Help' and refresh icons.

2.9. ábra. A vezérlőpulton kilistázott projektek

Bár a projektet létrehoztuk, az még nem publikus. Projektet publikálni akkor lehet, ha már rendelkezik leírással, amely bemutatja azt. Ennek a szerkesztésére - illetve a publikálásra - a felelős menedzsernek van joga. Amennyiben a vezérlőpulton menedzserként tekintjük meg a kilistázott projekteket, úgy a hozzáink tartozó projekt kiemelésre kerül a lista tetején *My project* cím alatt. A kilistázott projekt jobb alsó sarkában található *View* szövegre kattintva megnyílik a projekt vezérlőpultbeli megtekintő oldala, ahol felelős menedzserként jogunk van az egyes mezők szerkesztésére. A leírást **Markdown** formátumban tudjuk megadni. Ennek előnézetére *Description preview* gomb ad lehetőséget. A változtatásokat a jobb felső sarokban található *Save changes* gombbal menthetjük. A mentést követően - mivel a projekt immár rendelkezik leírással - lehetőség van a projekt publikálására. Ezt a mezők alatt megjelenő doboz *Publish* gombjára kattintva tudjuk megtenni (2.10 ábra).

## Project

[Save changes](#)

Name

Description

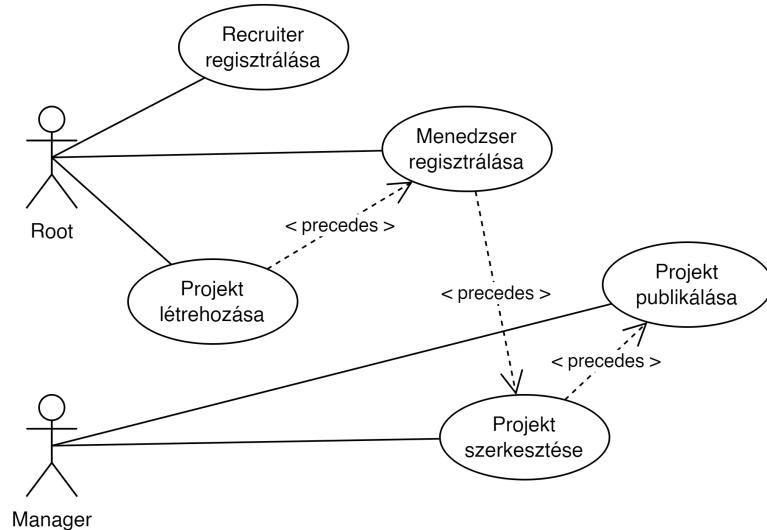
## Lorem ipsum  
  
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sed molestie enim. Suspendisse eget diam id orci tristique convallis non id purus. Duis iaculis et mauris vel dignissim. Curabitur varius lorem sed tellus consectetur maximus. Maecenas nec metus id lacus accumsan hendrerit quis in neque. Vestibulum consectetur mollis magna, ac bibendum massa malesuada ut. Vestibulum sed volutpat mi. Vivamus ut nibh efficitur, interdum diam nec, pulvinar lorem. Cras eu ullamcorper felis.

Description preview

Project is not yet published.
[Publish](#)

2.10. ábra. Publikálásra készen álló projekt szerkesztőfelülete

Az ezen alfejezetben tárgyalt funkcionálisokat az alábbi használati eset diagram foglalja össze (2.11 ábra).



2.11. ábra. Vállalati entitások regisztrálására vonatkozó használati eset diagram

## 2.6. Pozíciók meghirdetésének folyamata

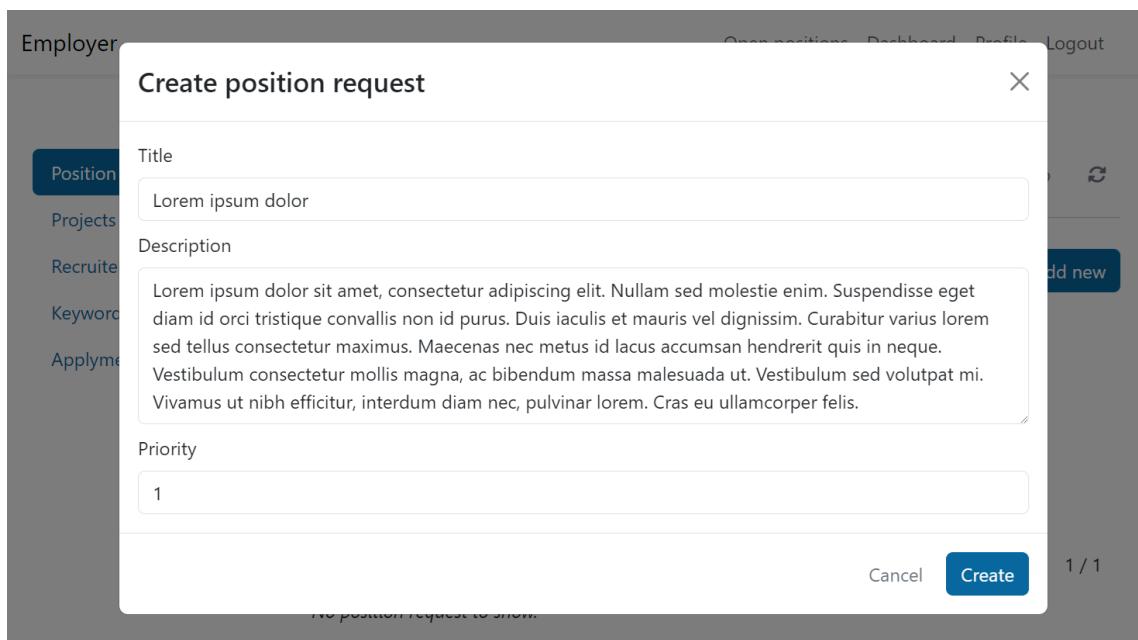
Publikus projektek esetén lehetőség van pozíciók meghirdetésére. A projekt felelős menedzsere megfogalmaz egy pozíciókiírási kérelmet, mely kérelem teljesítésére

egy recruiter felhasználó megkezdi a pozíció kiírását, melyet végül a menedzser hagy jóvá.

Menedzserként kérelmek kiírására a vezérlőpult *Position requests* menüpontja alatt van lehetőség, a jobb felső sarokban található *Add new* gomb megnyomásával felugró párbeszédablakban (2.12 ábra). A létrehozáshoz meg kell adnunk egy

- címet, amely röviden tükrözi a kérés tartalmának lényegét,
- leírást, amely a recruiter felhasználónak ad iránymutatást feladatának elvégzéséhez,
- prioritást, amely a kérés végrehajtásának sürgősséget jelzi.

Ezeket az adatokat a későbbiekben módosítani tudjuk addig, amíg a kérés lezárásra nem kerül.



2.12. ábra. Pozíciókiírási kérelem létrehozása

A létrejött kérés kilistázásra kerül a vezérlőpulton, ahonnan lehetőségünk van elérni annak szerkesztőoldalát. Az entitás adatainak (cím, leírás, prioritás) szerkesztésére csak a felelős menedzser jogosult. A recruiterek a szerkesztőoldalon tudják hozzárendelni magukat az egyes kérésekhez az *Assign* gomb használatával. Egy adott kérésen az ahhoz hozzárendelt, legfeljebb 1 recruiter felhasználó dolgozik. A hozzárendelést követően az adott recruiter megjelenik a szerkesztőoldalon, továbbá lehetőség nyílik pocíció létrehozására, kommentek írására (2.13 ábra).

## 2. Felhasználói dokumentáció

Title  
Lorem ipsum dolor

Description  
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam sed molestie enim. Suspendisse eget diam id orci tristique convallis non id purus. Duis iaculis et mauris vel dignissim. Curabitur varius lorem sed tellus consectetur maximus. Maecenas nec metus id lacus accumsan hendrerit quis in neque. Vestibulum consectetur mollis magna, ac bibendum massa malesuada ut. Vestibulum sed volutpat mi. Vivamus ut nibh efficitur, interdum diam nec, pulvinar lorem. Cras eu ullamcorper felis.

Priority  
1

As assigned recruiter, you can create a position to fulfil the position request. If a position is already created, you can edit it if the request is in state *Committed*. When you consider the position to be ready to publish, you have to move the request to state *Resolved* so the manager can approve or reject the publication.

---

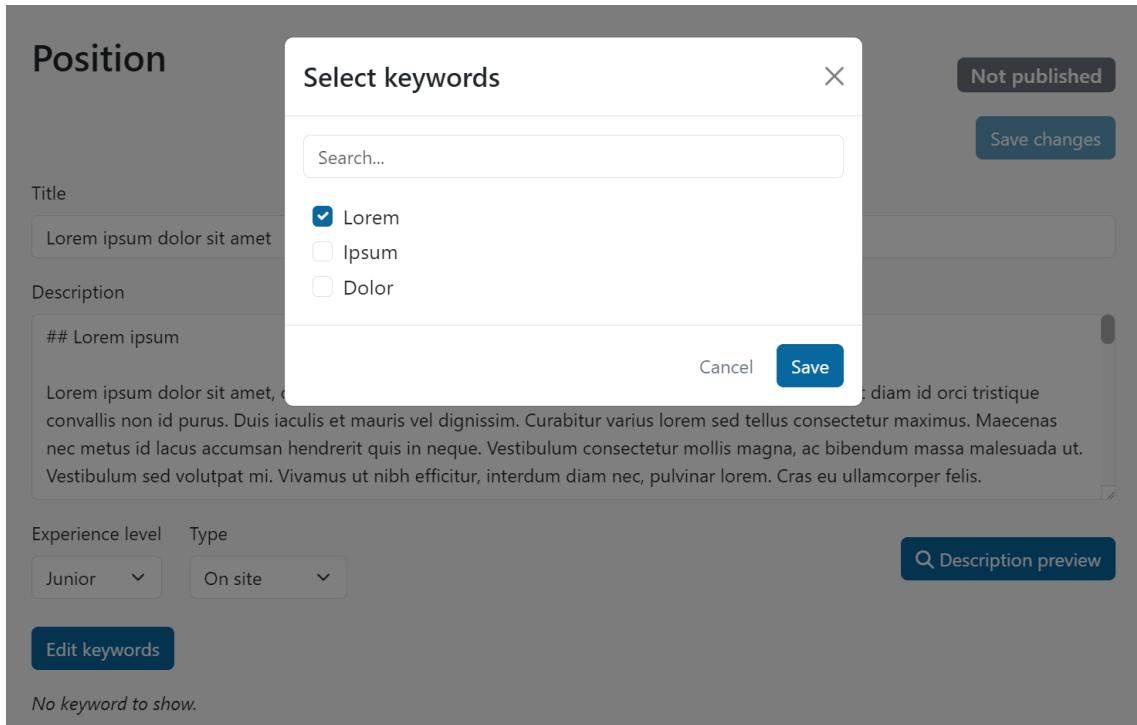
Recruiter	Position
<p>Example Recruiter <a href="mailto:recruiter@company.com">recruiter@company.com</a></p>	<p>No position has been created yet.</p> <p>Create</p>

---

Project	Comments
<p>Example Project Manager: Example Manager</p>	<p>No comment to show.</p> <p>→</p>

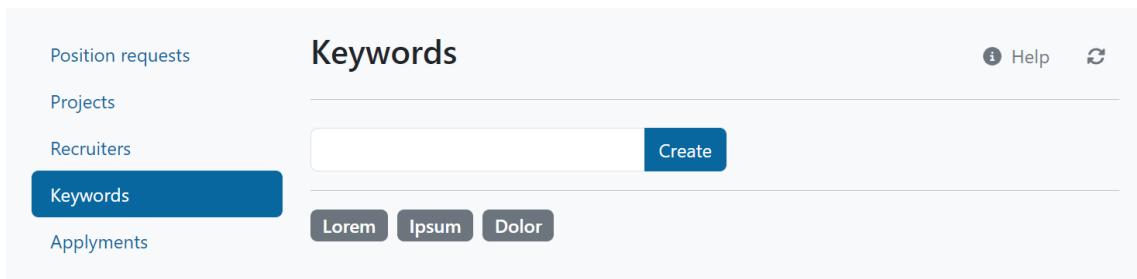
2.13. ábra. Pozíciókiírási kérelem szerkesztőoldala recruiter hozzárendelése után

A hozzárendelt recruiter a pozíciót a *Position* szekció jobb felső sarkában található *Create* gombbal előhívott párbeszédablakban tudja létrehozni. Ehhez meg kell adni a pozíció nevét, az elvárt tapasztalati szintet, valamint a munkavégzés helyének típusát. Ezek az adatok a publikálásig módosíthatók. A létrehozás után megjelenik a pozíció neve, melyre rákattintva a pozíciót szerkesztő oldalra jutunk, mely működésében hasonló a projekt szerkesztőoldalához, itt is Markdown formátumban lehet megadni a leírást. Újdonság a kulcsszóválasztó párbeszédablak (2.14 ábra), mellyel a pozícióhoz kulcsszavakat tudunk rendelni. Ezt az *Edit keywords* gombbal tudjuk előhívni.



2.14. ábra. Kulcsszavak hozzárendelése pozícióhoz

Ezeket a kulcsszavakat előzetesen regisztrálni kell, ezt a recruiterek tudják megtenni a *Keywords* fül alatt (2.15 árba). A kulcsszavak hozzáadás után szerkeszthetők és törölhetők a megfelelő párbeszédablakban, melyet az adott kulcsszóra kattintva tudunk előhívni.



2.15. ábra. Kulcsszavak listázása a vezérlőpulton

A recruiter az általa elkészültnek gondolt pozíciót késznek jelölheti a *Resolve* gomb használatával, ezzel megadva a menedzsernek a lehetőséget, hogy jóváhagyja vagy elutasítsa a kiírást. Ekkor a pozíció a recruiter által már nem szerkeszthető, csak akkor, ha a menedzser elutasítja a kiírást. Amennyiben elfogadja, a pozíció publikálásra, a kérés lezárásra kerül.

A menedzsernek lehetősége van nyitott pozíciók végleges lezárására. Egy pozíció a lezárása után nem fogad több jelentkezést. A műveletet a szerkesztőoldal alján

található *Close position* gomb megnyomásával lehet elvégezni.

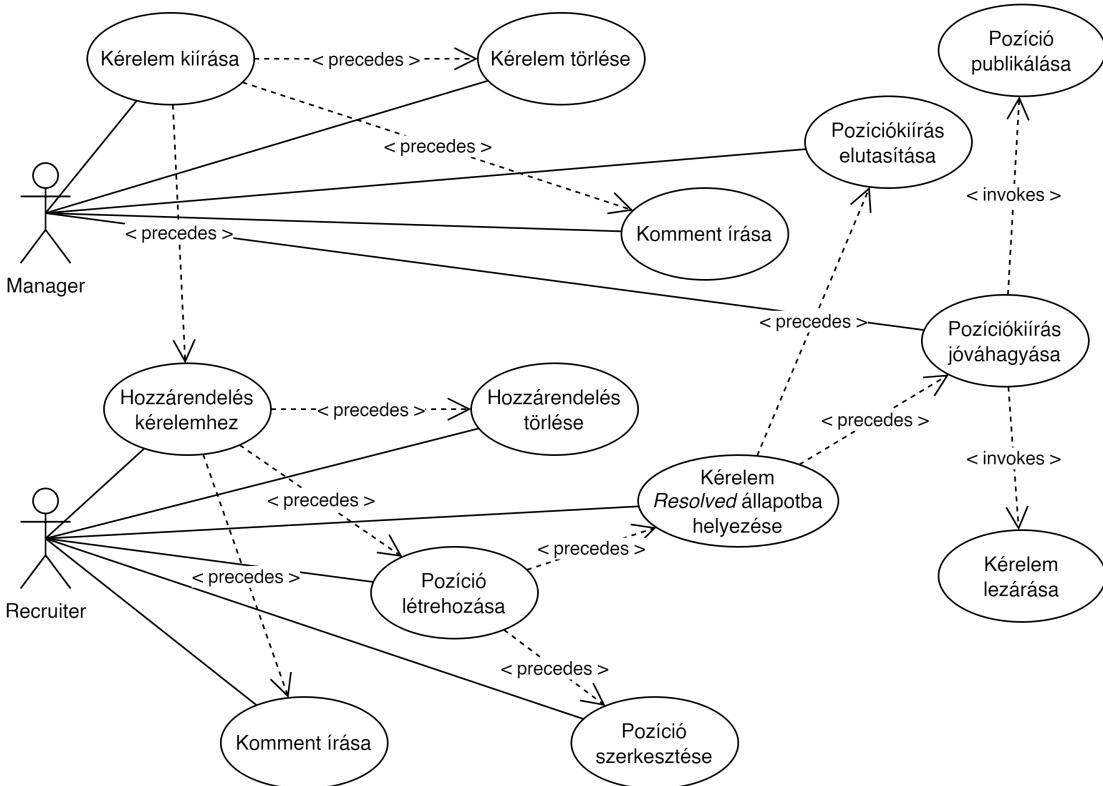
A szerkesztőoldal jobb felső sarkában egy állapotjelző jelvényt találunk, amelynek 4 értéke lehet. Ezek jelentéseit az alábbi táblázat tartalmazza.

Állapot	Jelentés
New	A kérés recruiter felhasználó hozzárendelésére vár.
Committed	A kéréshez egy recruiter felhasználó hozzárendelése megtörtént, aki jelenleg dolgozik a kért pozíció kiírásán.
Resolved	A hozzárendelt recruiter felhasználó jelezte a menedzser számára, hogy saját értékelése szerint elkészült a pozíció kiírásával.
Done	A menedzser jóváhagyta az elkészült pozíciót, a pozíció publikálásra, a kérés lezárásra került.

2.1. táblázat. Pozíciókiírási kérelem lehetséges állapotai és azok jelentései

A recruterek csak *Committed* állapotban lévő kérés esetén tudják megszünteti saját hozzárendelésüket, a menedzserek pedig csak *Resolved* állapotban lévő kérés pozíciókiírását tudják jóváhagyni vagy elutasítani.

Az ezen alfejezetben tárgyalt funkcionálisokat az alábbi használati eset diagram foglalja össze (2.16 ábra).



2.16. ábra. Pozíciók meghirdetésének folyamatát leíró használati eset diagram

## 2.7. Candidate felhasználó regisztrálása

Candidate felhasználó regisztrálása bármely nem bejelentkezett felhasználó jogosult. A regisztrációs oldalt a *Login* oldalon keresztül, a *Don't have an account yet? Register here!* szövegre kattintva tudjuk elérni. A regisztrációs ablak megjelenése és működése azonos a korábban bemutatott recruiter felhasználó regisztrálására használt párbeszédablakéval (2.8 ábra). A regisztrációt és a bejelentkezést követően a *Profile* oldalon lehetőségünk van felhasználói profilunk szerkesztésére (2.17 ábra). Kulcsszavakat a *Your interests* szekció alatt tudunk profilunkhoz asszociálni a pozíciók esetében már bemutatott módon (2.14 ábra). Önéletrajz felöltésére (legfeljebb 3MB, .pdf formátum) a *Curriculum Vitae* alatt található fájlfeltöltő bemenet használatával van lehetőség.

## Example Candidate

The screenshot shows a candidate profile page. At the top, there is an email link ([candidate@email.com](mailto:candidate@email.com)) and a user icon labeled "Candidate". Below this is a "Change password" button. A callout box contains two items: "Always have your CV uploaded! This is the best way you can introduce yourself to us!" and "Let us know you even better! Associate our predefined keywords with your profile! This could also help us to suggest you open positions." The main section is divided into two parts: "Curriculum Vitae" and "Your interests". Under "Curriculum Vitae", there is a file input field with "Fájl kiválasztása" and "Nincs fájl kiválasztva", an "Upload" button, and a "No CV available" message. Under "Your interests", there is an "Edit keywords" button and a message "No keyword to show."

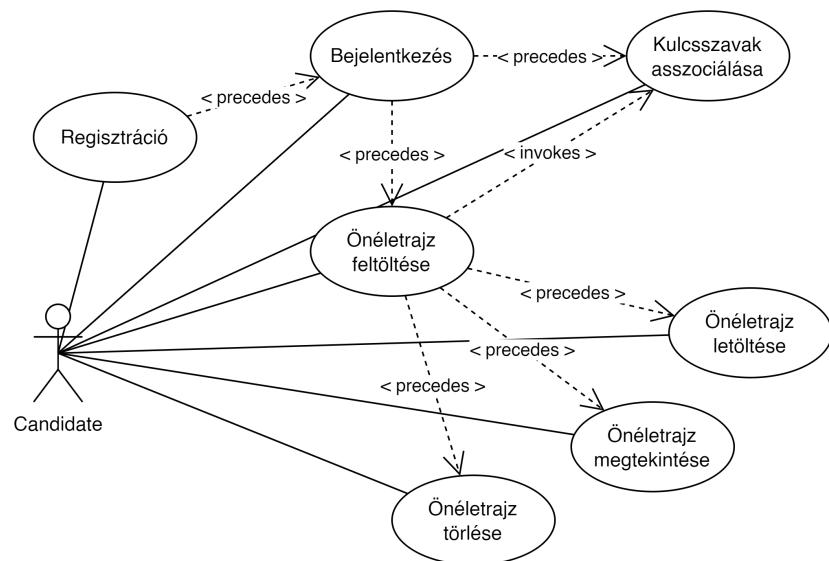
2.17. ábra. Candidate profiloldala

Az önéletrajz feltöltése során, amennyiben a rendszer a fájlban felismer kulcsszavakként regisztrált szavakat, azokat a profilunkhoz rendeli. Erről a rendszer egy felugró információs dobozban tájékoztat minket. Feltöltést követően önéletrajzunk letölthető, illetve megtekinthető az előnézete is (2.18 ábra).

The screenshot shows the same candidate profile page as above, but after a CV has been uploaded. The "Curriculum Vitae" section now shows "Fájl kiválasztása" with "Lorem.pdf" selected, an "Upload" button, and "Download" and "Preview" links. The "Your interests" section shows three suggested keywords: "Dolor", "Ipsum", and "Lorem". A callout box at the bottom states: "Based on your CV, these keywords might be interesting for you as well: Lorem, Ipsum, Dolor. We added them to your profile." with a close button.

2.18. ábra. Candidate profiloldala önéletrajz feltöltése után

Az ezen alfejezetben tárgyalt funkcionálisokat az alábbi használati eset diagram foglalja össze (2.19 ábra).



2.19. ábra. Candidate felhasználó regisztrálásának folyamatát leíró használati eset diagram

## 2.8. Jelentkezés és elbírálásának folyamata

### 2.8.1. Jelentkezés nyitott pozícióra

A nyitott pozíciók az alkalmazás főoldalán vannak listázva, ahogy az korábban bemutatásra került (2.2 ábra). A *View* szövegre kattintva az egyes pozíciók publikus oldalaira navigál minket a rendszer, ahol candidate felhasználóként lehetőségünk van jelentkezni az adott pozícióra a jobb oldali sávban található *Apply* gomb használatával. Jelentkezést követően a gomb helyére egy új gomb kerül *View applyment* felirattal (2.20 ábra), mely a jelentkezésünk oldalára navigál minket. Jelentkezéseinket a profiloldalunkon kilistázva is megtaláljuk.

## 2.2. Jelentkezés oldala

**Applied**

**Experience level**  
Junior

**Home office / In office**  
On site

**Keywords**  
Ipsum Lorem

**Project**  
[Example Project](#)

**View application**

**Text:** *Ipsum ipsum dolor sit amet. Sed pellentesque orci tellus, sed posuere mi iaculis non. Fusce iaculis placerat sem, vel laoreet eros fringilla eget. Donec ut nulla egestas, lacinia risus nec, semper lacus. Aenean vel neque auctor, ullamcorper velit in, pulvinar ligula. Ut molestie tempus nulla et molestie. Donec sed lacinia velit, at sagittis mi. Ut interdum commodo nulla, sit amet ornare urna feugiat in. Suspendisse quis fringilla libero. Ut in dolor quam. Nam vel leo tincidunt, rutrum purus eu, sodales metus. Sed at placerat lacus. Maecenas rutrum cursus est, sed consectetur lacus interdum quis. Etiam feugiat, nisl id vehicula auctor, massa ex fermentum purus, sed fringilla magna urna id ante. Aenean egestas nisl dolor, a lobortis nibh laoreet ac. Cras turpis quam, porta in egestas in, posuere in dui. Suspendisse velit eros, blandit a efficitur vitae, lacinia sed mauris.*

2.20. ábra. Pozíció oldala jelentkezés után

A jelentkezés megtekintőoldalán (2.21 ábra) tudjuk nyomon követni jelentkezésünk állapotát a jobb felső sarokban található állapotjelző jelvény segítségével. Ennek lehetséges értékeit és azok jelentését az alábbi táblázat foglalja össze.

Állapot	Jelentés
Applied	A jelentkezés megtörtént, viszont a vállalat még nem bírált el azt.
Hired	A vállalat elbírált a jelentkezést, a pályázó felvételt nyert a pozícióra.
Declined	A vállalat elbírált a jelentkezést, a pályázó nem nyert felvételt a pozícióra.

2.2. táblázat. Jelentkezés lehetséges állapotai és azok jelentései

A jelvény mellett az oldalon elhelyezett információs dobozok is segítséget nyújtanak az állapot monitorozásában.

## Appliment

Applied

**i** You have applied for the position  **Lorem ipsum dolor sit amet**. Make sure you check the state of the appliment frequently by visiting the current page!

Position: [Lorem ipsum dolor sit amet](#)  
Applied at: 2024. 05. 03. 21:10:41  
[Unapply](#)

2.21. ábra. Jelentkezés megtekintőoldala

### 2.8.2. Jelentkezés kezelése a vezérlőpulton

A jelentkezés a vezérlőpulton az *Appliments* fül alatt kerül listázásra. Recruiter felhasználóként, a pozíciókiírási kérelmekkel azonos módon hozzá tudjuk rendelni magunkat az egyes jelentkezésekhez. Kiemelendő, hogy minden jelentkezés külön van kezelve: egy candidate felhasználó egy pozícióra történő pályázása minősül egy jelentkezésnek. A hozzárendelést a jelentkezés megtekintőoldalán tudjuk elvégezni, melyet követően a candidate profilinformáció mellett megjelennek különböző műveletek elvégzésére lehetőséget adó gombok (2.22 ábra).

The screenshot shows a user interface for managing a candidate application. At the top left, it says "Candidate". Below that, under "Personal data", there is a placeholder "Example Candidate" with an email "candidate@email.com". To the right, there is a "Curriculum Vitae" section with "Download" and "Preview" buttons. Under "Keywords", there are three tags: "Dolor", "Ipsum", and "Lorem". Below this, there is a section for "Recruiter" with "Example Recruiter" and "recruiter@company.com". To the right of the recruiter's name is a link "Unassign". A note below the recruiter's information states: "As assigned recruiter, you can forward the applyment for the manager to make the decission about the applyment, or you can decline the applyment yourself. To help make your decission easier, you can also invite the candidate for a video chat interview." At the bottom, there are three buttons: "Invite for interview" (blue), "Forward" (blue), and "Decline applyment" (red).

2.22. ábra. Jelentkezés megtekintőoldala a vezérlőpulton, hozzárendelést követően

A *Decline applyment* gombbal egy megerősítést követően lehetőségünk van a jelentkezést elutasítani. A *Forward* gomb a felelős menedzser felé továbbítja a jelentkezést a döntés meghozatalára, aki szintén elutasíthatja a jelentkezést, elfogadni viszont csak neki van lehetősége. A döntés megkönnyítése érdekében, a továbbítás során a recruierteknek egy rövid beszámolót kell írni meglátásairól, észrevételeiről. Erre a *Forward* megnyomásakor megnyíló párbeszédablakban van lehetőség.

### 2.8.3. Meghívás videóinterjúra

A recruiert a jelentkezés továbbítása vagy elutasítása előtt dönthet úgy, hogy videóinterjúra hívja a pályázót. Ezt az *Invite for interview* gombbal tudja megtenni. Szükséges megadni az interjú kezdő- és befejező időpontját, mely időpontot között lehetőség van a hívásba becsatlakozni, vagy újracsatlakozni. Emellett meg kell adni az elfogadási határidőt: a pályázónak eddig van lehetősége reagálni az interjúmeghívára. Fontos, hogy az interjúban történő részvétel feltétele, hogy mind audió, mind videó eszközök rendelkezésre álljanak. Erre a felület több ponton is felhívja a fi-

gyelmet. Az interjú a kiírást követően megjelenik a megtekintőoldalon az *Interview* szekcióban (2.23 ábra).

The screenshot shows a user interface for managing job applications. At the top, it displays the title "Recruiter" and the status "Unassign". Below this, there is contact information for an example recruiter: "Example Recruiter" and "recruiter@company.com". The main section is titled "Interview" and shows the following details:

- Starts at:** 2024. 05. 10. 14:00:00
- Ends at:** 2024. 05. 10. 15:00:00
- Acceptance deadline:** 2024. 05. 09. 14:00:00

A message indicates: "The candidate has not yet accepted the interview invitation." Below this, there are three buttons: "Go to interview", "Forward", and "Decline application". A note states: "As assigned recruiter, you can forward the applyment for the manager to make the decission about the applyment, or you can decline the applyment yourself. To help make your decission easier, you can also invite the candidate for a video chat interview." At the bottom, a message says: "Forward and decline actions are disabled, because the applyment has a pending, upcoming or ongoing interview."

2.23. ábra. Jelentkezés megtekintőoldala a vezérlőpulton, interjúra történő meghívást követően

A hívásba becsatlakozni a *Go to interview* gomb megnyomásával lehetséges, a kezdő- és befejező időpont között. A szekció jobb felső sarkában található állapotjelző jelvény 6 értéket vehet fel. Ezek jelentését az alábbi táblázat tartalmazza.

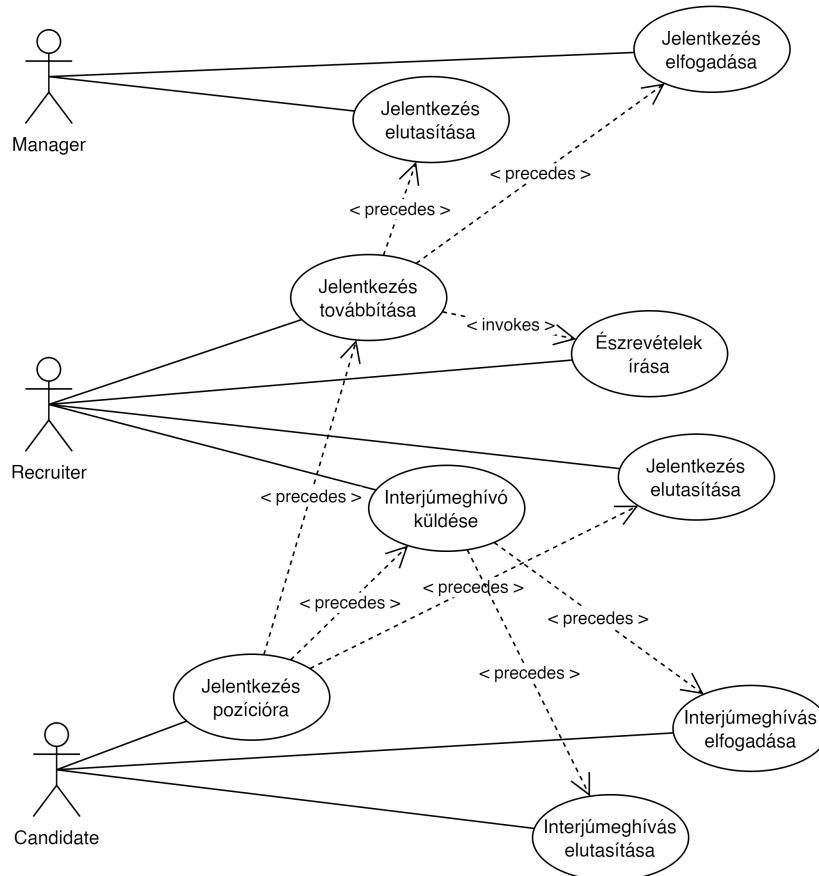
Állapot	Jelentés
Pending	A jelentkező még nem reagált a meghívást, de még van rá lehetősége.
No response	A jelentkező nem reagált a meghívásra, és már nincs is rá lehetősége.
Declined	A jelentkező reagált a meghívásra és elutasította azt.
Upcoming	A jelentkező elfogadta a meghívást, az interjú még nem kezdődött el.
Ongoing	A jelentkező elfogadta a meghívást, az interjú éppen esedékes.
Ended	A jelentkező elfogadta a meghívást, az interjúnak már vége.

2.3. táblázat. Interjú lehetséges állapotai és azok jelentései

Ameddig *Pending*, *Upcoming* vagy *Ongoing* állapotú az interjú, nincs lehetőség a jelentkezés elutasítására vagy továbbítására. Erről a felület is tájékoztatást ad. Új interjú kiírására a korábbi törlését követően van lehetőség.

A meghívást követően a candidate által látott jelentkezést megtekintő oldalon is megjelenik az *Interview* szekció. A korábbinál annyival bővebb, hogy *Pending* állapotban megjelennek a meghívást elfogadására és elutasítására szolgáló gombok.

Az ezen alfejezetben tárgyalt funkcionálisokat az alábbi használati eset diagram foglalja össze (2.24 ábra).

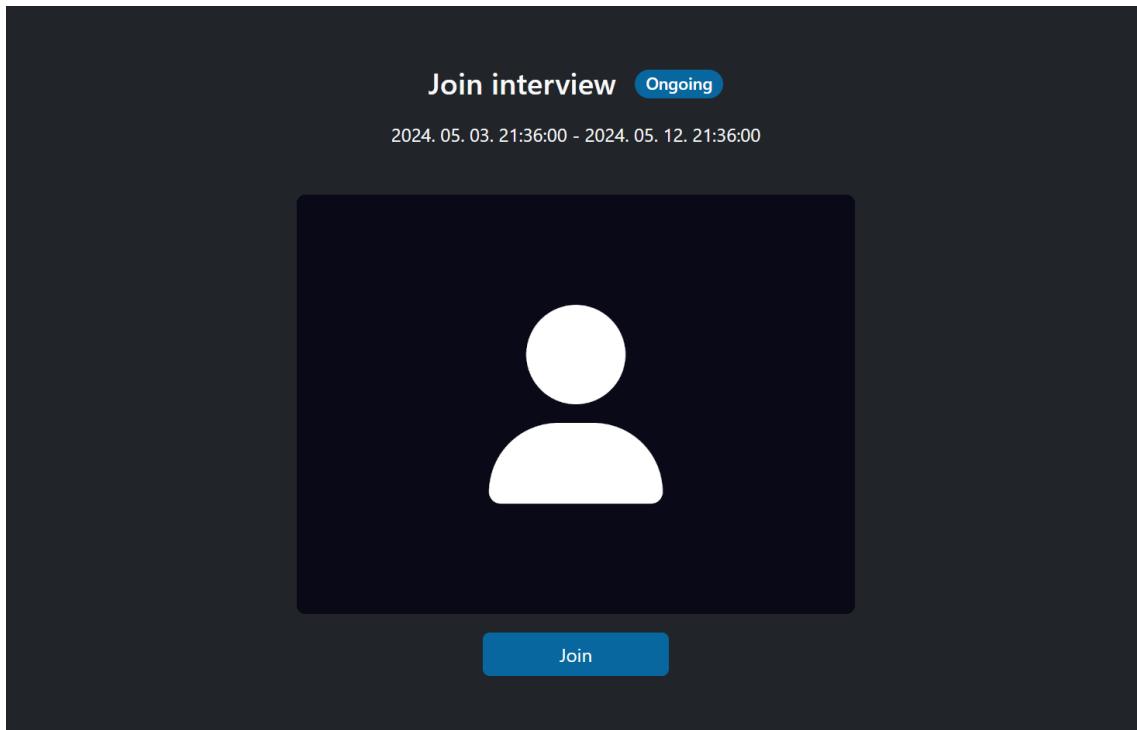


2.24. ábra. Jelentkezés és elbírálásának folyamatát leíró használati eset diagram

## 2.9. Videóinterjú

A jelentkezés megtekintőoldalán (2.21, 2.23 ábrák) található *Go to interview* gomb megnyomásával a rendszer egy új böngészőlapban megnyitja az videóhívás felületét. A hívásba történő tényleges becsatlakozást megelőzően engedélyeznünk kell

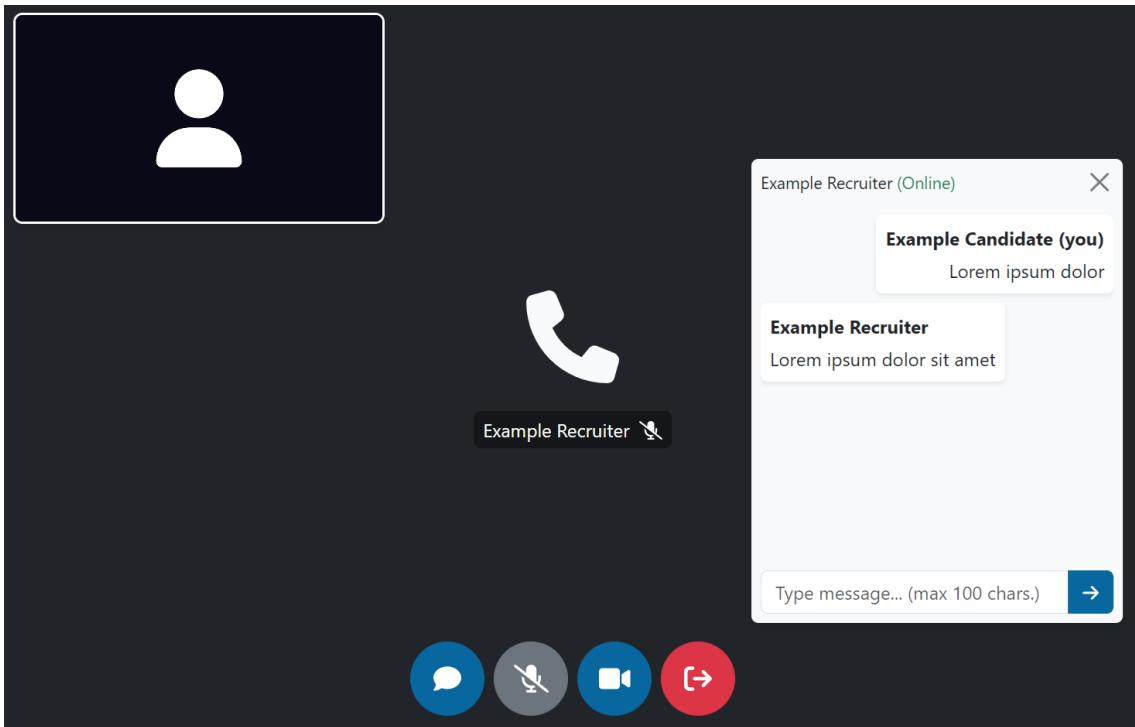
audió- és videóeszközeink elérését. Miután ezt megtettük, kameraképünk előnézete megjelenik, és a *Join* gomb megnyomására lehetőség adódik (2.25 ábra).



2.25. ábra. Videóinterjú belépési oldala a médiaeszközök engedélyezését követően

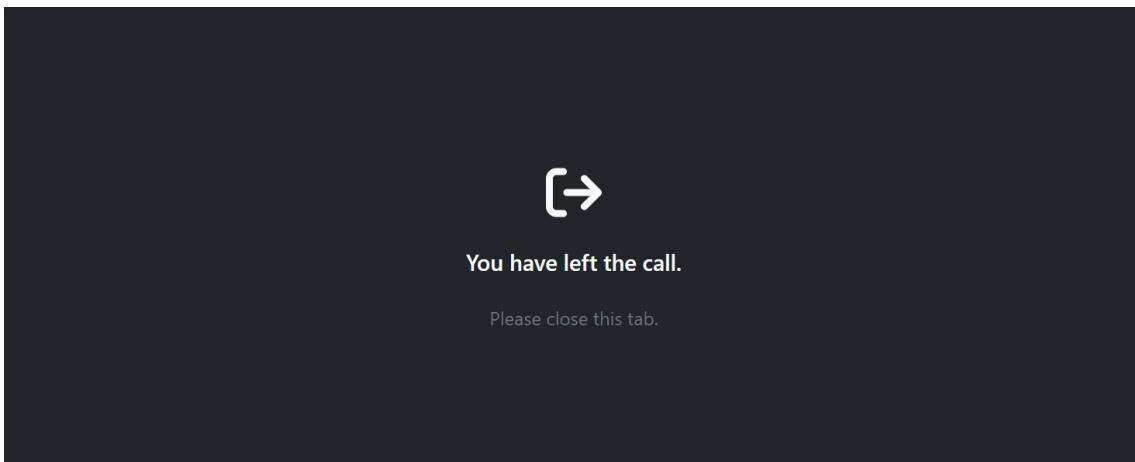
A belépést követően a hívás felülete fogad minket, amely kialakítása követi a felhasználói tradíciókat. Alul találhatóak a vezérlők, mellyel a médiaeszközök ki- és bekapcsolását, a chat megnyitását, a hívásból történő kilépést lehet kezdeményezni. A képernyő bal felső sarkában megjelenik saját videóképünk, melyet partnerünk lát, középen pedig partnerünk videóképe jelenik meg, amennyiben az be van kapcsolva. Ellenkező esetben egy telefon ikont látunk (2.26 ábra).

A hívás során lehetőség van valós időben szöveges üzenetek küldésére. A csevegés ablakot a vezérlő bal oldalán található szövegbuborék ikonnal rendelkező gomb segítségével lehet megjeleníteni, elrejteni. Amennyiben elrejtett állapotban új üzenet érkezik, arra a gomb jobb felső sarkában egy piros pötty hívja fel a figyelmet.



2.26. ábra. Videóhívás felülete, a beszélgetőpartner kikapcsolt kamerájával

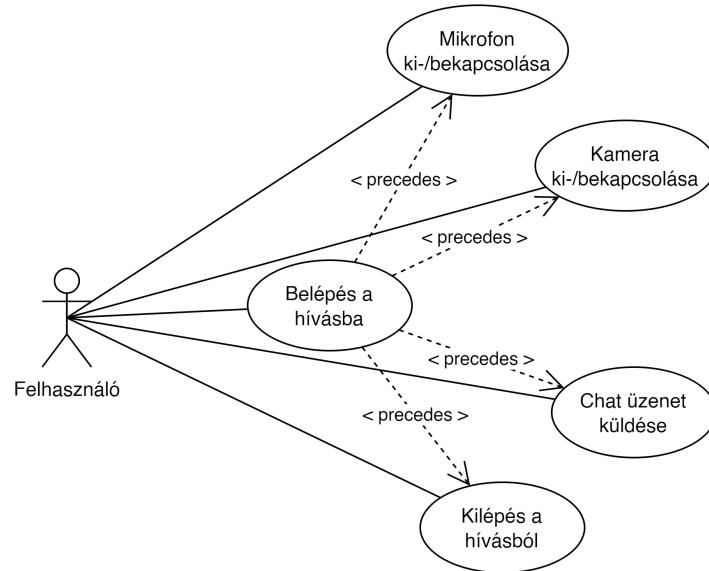
Fontos, hogy egy felhasználóval egyszerre 1 hívásban tudunk részt venni. Amennyiben megpróbálunk egy másik böngésző lapról egy másik (vagy akár ugyanabba a hívásba) becsatlakozni, a régi lapon lecsatlakoztatásra kerülünk az ottani hívásból. Ez esetben, illetve szándékos kilépés esetén az alábbi ablak fogad minket (2.27 ábra).



2.27. ábra. Videóhívás kilépő felülete

Amennyiben a hívásból más ok miatt kerülünk lecsatlakoztatásra, például hálózati hiba esetén, hasonló ablak fogad minket, az esetet leíró szöveggel és ikonnal.

Az ezen alfejezetben tárgyalt funkcionálitásokat az alábbi használati eset diagram foglalja össze (2.28 ábra).



2.28. ábra. Videóinterjú folyamatát leíró használati eset diagram

## 3. fejezet

# Fejlesztői dokumentáció

### 3.1. A megoldandó feladat

A célunk az, hogy egy olyan webalkalmazást fejlesszünk, amely a felhasználói dokumentációban leírt funkcionálisokkal hiánytalanul rendelkezik. Fullstack webalkalmazások esetében rendelkeznünk kell mind a kliens-, mind a szerveroldal megvalósításáról. A szervert C# nyelven, ASP.NET keretrendszer [1] használatával, a klienst pedig TypeScript nyelven, a React keretrendszer [2] felhasználva implementáljuk. A szerver elsődlegesen API-ként működik, vagyis kódszinten a két réteg között nincs kapcsolat, helyette a kliens API hívásokkal kommunikál a szerverrel. A valósidejű kommunikációt igénylő funkcionálisok miatt arra is szükség van, hogy a szerver WebSocket kapcsolatot is biztosítson.

#### 3.1.1. A felhasználói felületre vonatkozó elvárások

- Legyen intuitív, könnyen használható. A felület készítésekor vegyük figyelembe a felhasználói tradíciókat, hogy az könnyen és egyértelműen használható legyen.
- Ahol indokolt, álljon rendelkezésre súgó az alkalmazásban, ezzel is segítve a felhasználót.
- Legyen esztétikus. A felületen alkotó komponensek készüljenek egységes séma alapján. A térközök, margók mérete szintén legyen konzisztens.

- Adjon vizuálisan információt arról, amennyiben a szerver válaszára várunk. Nem minden válasz érkezik meg egyszerre, és mivel egyoldalas alkalmazást (SPA) készítünk, a böngésző nem végzi el helyettünk ezt a feladatot.
- Közvetítse felénk a szerverről érkező esetleges hibaüzeneteket. Validációs hiba esetén jelezük azt is, hogy mely bemeneti mező(k) érintett(ek).

### 3.1.2. A szerveroldali komponensre vonatkozó elvárások

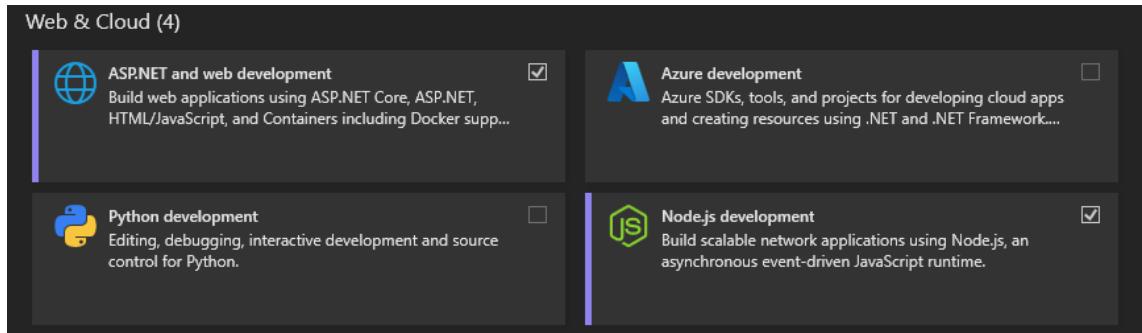
- A lekérdezések legyenek optimalizálva mind a szerver, mind a kliens számára. Használjuk ki, hogy ismerjük a klienst, ezáltal annak igényeit is. Ahol van értelme, ott az egyes entitásokkal együtt adjuk vissza a szükséges (és csak a szükséges) asszociált entitásokat is. A szerver számára is teher az, ha a kliensnek 1 darab optimalizált lekérdezés helyett akár 3-4 kérést kell küldenie. Továbbá megjegyezzük, hogy amennyiben az asszociációkat nem végezzük el a szerveroldalon, úgy egyes esetekben egy akár  $O(n \cdot m)$  asszociációs probléma megoldására kényszerítenénk a klienst, vagyis a felhasználó böngészőjét. Ezt nem engedjük meg.
- A végpontok kialakítása legyen konvencionális és egyértelmű. A hagyományos CRUD műveletek elvégzésére használatos végpontok mellett definiálunk végpontokat konkrét felhasználói esetek végrehajtására is.
- A klienstől érkező kérések kerüljenek validálásra. Amennyiben a kliens által megadott paraméterek nem megfelelők, adjunk visszajelzést a probléma okáról. Az egyes sztring paraméterek elejéről és végéről a szóközöket távolítsuk el feldolgozás előtt, valamint az alkalmazás legyen védett SQL befecskendezés ellen. minden sztring bemenet rendelkezzen maximális megengedett hosszal.

## 3.2. Fejlesztői környezet

A fejlesztéshez Windows 11 operációs rendszert futtató személyi számítógépet használtam. A szerveroldal fejlesztése a Microsoft Visual Studio Enterprise 2022, a kliensoldal fejlesztése pedig a Microsoft Visual Studio Code programmal történt. A forráskód verziókezelése GitHub-on keresztül valósult meg. Az itt felsorolt adatok a

felhasználói dokumentáció rendszerkövetelmények alfejezetében (2.2 fejezet) kerültek részletezésre.

Microsoft Visual Studio-ban történő fejlesztés során rendelkeznünk kell az *ASP.NET and web development*, valamint *Node.js development* komponensekkel. Amennyiben szükséges, módosítsuk korábbi Visual Studio telepítésünket (3.1 ábra).



3.1. ábra. Szükséges fejlesztői komponensek

Az alkalmazást Debug konfigurációban futtatni a Visual Studio-ban, az *Employer.sln* megnyitásával van lehetőség. A kliens alkalmazás külön is futtatható fejlesztői módban. Ehhez az *Employer.Client* mappán belül ki kell adnunk az `npm run dev` parancsot.

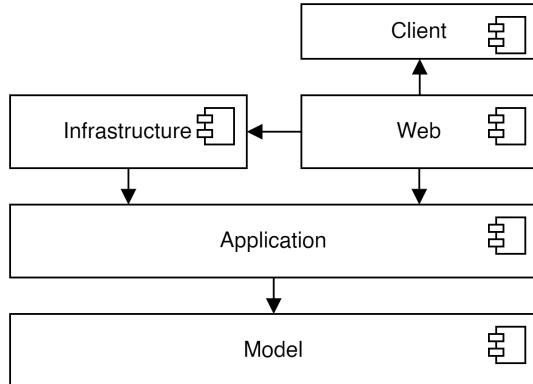
### 3.3. Az alkalmazás rétegei és felelősségi köreik

Az alkalmazás tervezésénél törekedtem az objektumorientált paradigmák követésére, tervezési minták használatára, illetve a SOLID elvek betartására. Ez utóbbi esetében különösen figyeltem arra, hogy ahol csak lehetőség van, konkrét implementációk helyett absztrakciók között állítsak fel függőségeket, és kihasználjam a függőség befecskendezés nyújtotta előnyökét.

**1. Definíció.** Függőség befecskendezés esetén a kliens biztosít egy átadási pontot a szolgáltatás interfésze számára, ahol a végrehajtás során a szolgáltatás megvalósítását adhatjuk át. Az átadási pont lehet: konstruktor, metódus, interfész; a befecskenést végző komponens lehet egy felsőbb réteg, a rétegződéstől független környezeti elem. [3]

Összetettebb alkalmazások esetén érdemes több rétegű architektúrát alkalmazni, hiszen ezzel számos előnyhöz juthatunk. A felelősségi körök szétválasztásával kódunk

sokkal átláthatóbbá válik, amely elősegíti annak bővíthetőségét és újrafelhasználhatóságát, de a tesztelhetőséget is jelentős mértékben növeli. Az Employer alkalmazás 5 rétegből épül fel, ebből 4 alkotja az alkalmazás szerveroldali-, 1 pedig a kliensoldali komponensét. A 3.2 ábra az alkalmazás rétegeit és azok egymásra épülését szemlélteti.



3.2. ábra. Az alkalmazás rétegei és azok egymásra épülésének szemléltetése

A szerveroldali rétegek, és az azok közötti kapcsolatok tervezésének esetében a clean architecture irányelveit pragmatikusan felhasználva jártam el. Ezeket az irányelveket először Robert C. Martin (Uncle Bob) írta le 2012-ben [4]. A clean architecture tulajdonképpen arra ad javaslatot, hogy a kódot miként, és milyen komponensekre bontsuk fel. A szakdolgozat készítése közben azt tapasztaltam, hogy ez az architektúra a mai napig népszerű és megállja a helyét. A clean architecture szemléletmódnak elsajátítása, és annak implementációba történő átültetésének tanulása során alapul vettem Jason Taylor [5], és Amichai Mantinband [6] GitHub könyvtárait.

### 3.3.1. A modell réteg

A modell réteg köré épül az egész szerveroldali komponens: minden más réteg függ a modell rétegtől, viszont a modell réteg nem rendelkezik függőséggel más rétegek felé. A modell réteg felelőssége, hogy leírja az alkalmazás entitásait, azok tulajdonságait, illetve meghatározza az egyes entitások között fennálló kapcsolatokat. Ezen kívül további feladata nincs. Az entitások elsősorban adatátviteli objektumként (DTO) szolgálnak, meghatározva azt, hogy az adatbázisban milyen mezők szükségesek eltárolásukra. Az entitásoknak megengedjük ezek mellett kiszámolt tu-

lajdonságok definiálását is, amelyek az adatbázisban nem kerülnek tárolásra - hiszen redundáns adatok lennének -, viszont meghatározzák, bővítik a domain leírását.

### 3.3.2. Az applikáció réteg

Az applikáció réteg rendelkezik a legfontosabb felelősségi körrel: az üzleti logika megfogalmazásával. Feladata, hogy vezényelje egyes lekérdezések és parancsok végrehajtását, természetesen miután megállapította ezek végrehajthatóságát. Ez utóbbi függ az alkalmazás mindenkorai - üzleti logikai szempontból vett - állapotától, illetve függhet a végrehajtást kezdeményező felhasználó identitásától. Amennyiben egy utasítás nem hajtató végre, elvárjuk, hogy információt kapunk annak miértjéről. Az applikáció réteg csak a modell réteg felé rendelkezik függőséggel.

### 3.3.3. Az infrastruktúra réteg és annak kapcsolata az applikáció réteggel

Az infrastruktúra réteg függ a modell- és az applikáció rétegektől. Míg az applikáció rétegen a "mit?" és "miért?" kérdésekre adtunk választ, az infrastruktúra rétegen a "hogyan?" kérdéssel foglalkozunk. Ezen réteg felelőssége, hogy konkrét implementációkkal, illetve külső csomagok bevonásával megvalósítsa azokat az absztrakciókat, amelyeket az applikáció réteg megfogalmaz. Ide tartoznak az autentikációval, fájl-kezeléssel, valósidejű kommunikációval és adatbázis-kezeléssel kapcsolatos konkrét implementációk. Ezek az absztrakciók az `Employer.Application.Common.Abstractions` névtérben vannak definiálva.

Az applikáció rétegnek nem kell tudnia, hogy hogyan kezeljük az adatbázist, milyen módon autentikálunk és melyik algoritmust használjuk a jelszavak hashelésére, ő csupán a saját maga által definiált absztrakciókon keresztül használja fel ezeket, vagyis alkalmazzuk a függőség megfordításának elvét (dependency inversion principle - DIP). Ez teszi lehetővé azt, hogy ne az applikáció réteg függjön az infrastruktúra rétegtől, hanem fordítva.

**2. Definíció.** A Dependency inversion principle a függőségek kezelésével szemben fogalmaz meg elvárást, amelynek értelmében a magasabb szintű programegységek nem függhetnek alacsonyabb szintű programegységtől, minden kettőnek az absztrakciótól kell függenie; az absztrakció nem függhet a részletektől, a részletek függenek az absztrakciótól. [7]

Összefoglalva, az infrastruktúra réteggel - és annak az applikáció réteghez való viszonyulásával - biztosítjuk, hogy az üzleti logikába ne szivárogjanak be olyan kérdések, amelyek nem odavalók. Ezzel fenntartjuk annak is a lehetőségét, hogy az absztrakciók mögött meghúzódó implementációkat a későbbiekben esetleg ki-cseréljük, mindezt anélkül, hogy a már meglévő üzleti logikát eltörnénk. A clean architecture-nek ez az egyik fő szempontja.

### **3.3.4. A prezentáció: a web- és kliens rétegek**

A web réteg felelőssége a fentebb bemutatott három réteg összefogása, a függőség befecskendezések biztosítása, valamint a kapcsolat lehetőségének megteremtése a klienssel. A függőség befecskendezést a `Employer.Application` és `Employer.Infrastructure` névterekben található `DependencyInjection` statikus osztályokban definiált kiegészítési metódosainak segítségével végzi.

A web réteg biztosítja továbbá az alkalmazás főbb konfigurációit, definiálja a WebSocket és API végpontokat, valamint ez utóbbi felelősség keretein belül kontrollereket implementál. A kontrollerekben azonban üzleti logikát nem hajtunk végre, hiszen az az applikáció réteg felelőssége. Ehelyett a közvetítő (mediator) minta használatával kommunikál az applikáció rétegen implementált parancsokkal és lekérdezésekkel.

**3. Definíció.** A közvetítő minta abban segít, hogy a szoros kapcsolatok helyett lazábban összekapcsolt objektumokat hozzunk létre, és ezzel növeljük alkalmazásunk karbantarthatóságát. Ezt úgy éri el, hogy az egyes objektumok nem közvetlenül egymással kommunikálnak, hanem egy központi objektumon, a közvetítőn keresztül. [8]

A kliens réteg a web réteg által definiált végpontokon keresztül kommunikál a szerverrel. Feladata az, hogy felhasználói felületet biztosítson a parancsok és lekérdezések végrehajtásának kezdeményezésére, azok eredményeinek megjelenítésére.

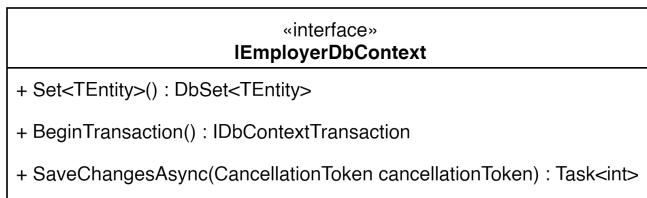
## **3.4. Perzisztencia**

A perzisztencia biztosítása az infrastruktúra réteg feladata. Mivel a program felépítése objektumorientált, így az adatkezelést ily módon szeretnénk végezni. Ehhez az Entity Framework Core (továbbiakban: EF Core) nyílt forráskódú csomagot használjuk, amely megvalósítja az adatok objektumrelációs leképzését.

**4. Definíció.** Az objektumrelációs leképzés (object-relational mapping - ORM) magas szintű transzformációját adja az adatbázisnak, amely a programban könnyen használható, ugyanakkor szabályozza az adatok kezelésének módját. A létrejött entitás osztályok csak adatokat tárolnak, műveleteket nem végeznek. [9]

Ahogy azt már tárgyaltuk, ezek az entitás osztályok a modell rétegen kerülnek definiálásra. Az eltárolt- és számolt tulajdonságok mellett az EF Core lehetőséget ad arra, hogy ún. navigációs tulajdonságokkal is ellássuk az egyes entitás modelleket, ezzel megkönnyítve az azok közötti kapcsolatok definiálását, illetve a kapcsolódó (asszociált) entitások lekérdezését. Ez ORM-ek esetén gyakori funkcionalitás.

Az applikáció réteg az adatbázist az `IEmployerDbContext` absztrakción keresztül használja (3.3 ábra). Az absztraktió mögött egy `DbContext` osztály húzódik, ezt az EF Core biztosítja. Az interfészre kihelyezzük a `Set< TEntity >()` generikus metódust, mely visszatérési értéke `DbSet< TEntity >` típusú, amely szintén EF Core objektum. Ez tulajdonképpen a `TEntity` entitás táblája, amelyen CRUD műveleteket végezhetünk. Emellett az interfészen láthatóvá teszünk a mentéshez és a tranzakciókezeléshez szükséges metódusokat. Az absztraktió megvalósítása az `Employer.Infrastructure.Persistence` névterben található. Az EF Core-ról részletesebben annak dokumentációjában olvashatunk. [10]

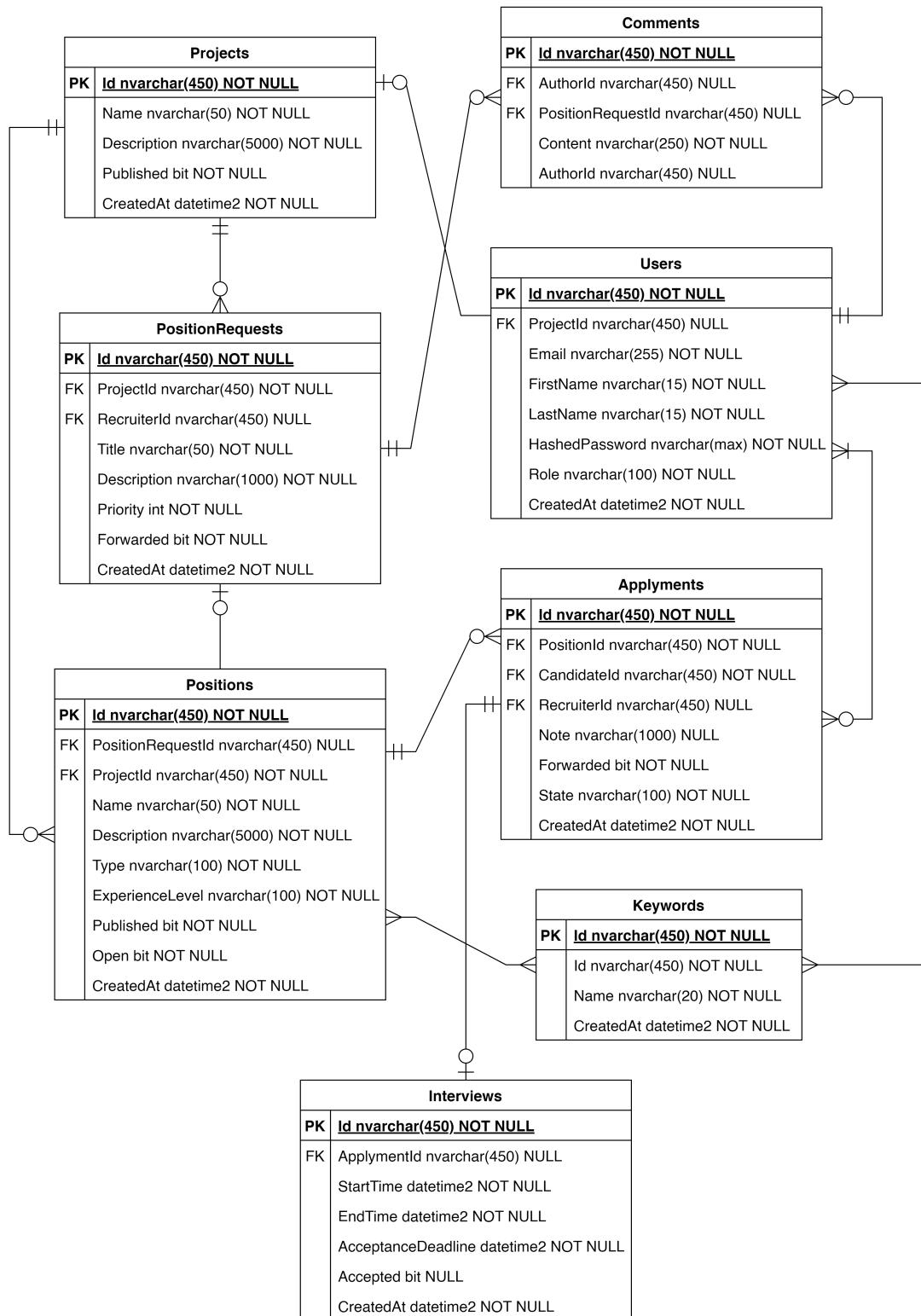


3.3. ábra. Az `IEmployerDbContext` interfész UML diagramja

A `DbSet< TEntity >` objektumot az EF Core felépítése miatt sajnos már nem tudjuk absztraktió mögé helyezni, így azzal szembesülünk, hogy az applikáció rétegen EF Core-tól való függőség keletkezik. A clean architecture ezt nem javasolja - az EF Core így már nem cserélhető ki a későbbiekben, vagyis ehhez az ORM-hez vagyunk kötve. Megoldás lehet a *repository pattern* használata, viszont ezen alkalmazás esetében ez már aránytalan többlet (overhead) lenne. Persze egy generikus repository implementálása nem nehéz feladat, viszont ha csak ennyit implementálnánk, lemondanánk az EF Core nyújtotta optimalizációs lehetőségekről, amit - mivel előnyben részesítjük a teljesítményt és a kód flexibilitását - nem tehetünk meg. Ezért is fontos

az architektúra lényegének átlátása, hogy az iránymutatások okait megértve mérlegelni tudjunk, és azokat pragmatikus módon használjuk fel ennek függvényében. Így ezt a függőséget a jelenlegi alkalmazásban megengedjük: a .NET alkalmazások esetében ha ORM-ról van szó, az EF Core szinte egy egyértelmű választás, így ennek kézeli cseréjére nagy valószínűséggel nem lesz szükség (illetve ORM révén az EF Core már maga egy absztrakció). Ezen kompromisszum megkötése a clean architecture-t alkalmazó projektek esetében nem ritka, Jason Taylor [5] is így járt el: repository pattern használata helyett hasonló módon az EF Core adatbázis-kontextus osztályát tette absztrakció mögé, kiengedve a `DbSet< TEntity >` objektumot.

Az adatbázis-kontextus regisztrálása a `DependencyInjection` osztályban, az entitások konfigurálásai pedig az `Employer.Infrastructure.Persistence.Configuration` névterben találhatók. Az alkalmazás egyed kapcsolat diagramja a 3.4 ábrán látható.



3.4. ábra. Az alkalmazás egyedkapcsolat diagramja SQL adattípusokkal

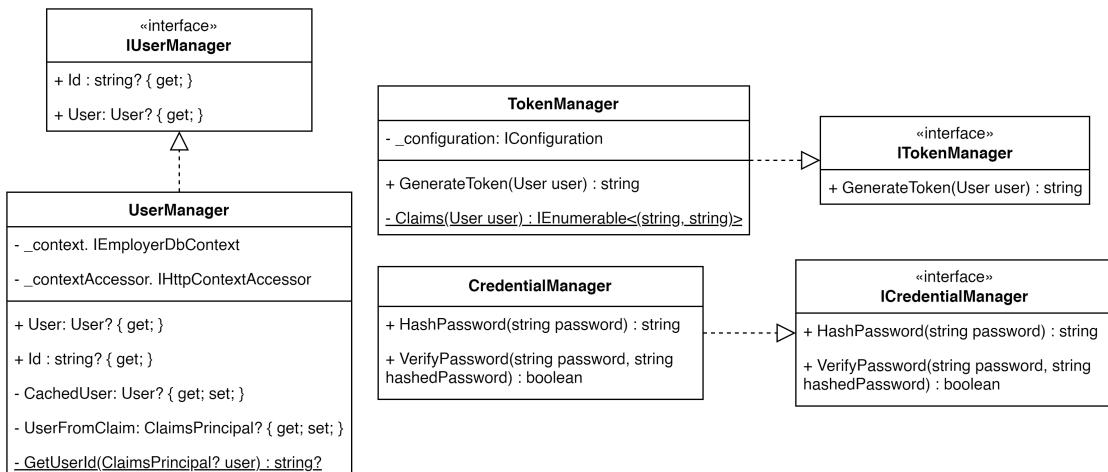
### 3.5. Autentikáció

Az alkalmazásban az autentikációt JSON Web Token (JWT) segítségével valósítjuk meg. A JWT-t a felhasználó sikeres bejelentkezését követően állítjuk elő szerveroldalon, majd a bejelentkezett felhasználó adataival együtt a válaszban visszaküldjük azt. A tokent a kliens eltárolja, és a későbbiekben minden HTTP kérés fejlécéhez csatolja azt. A JWT tartalmazza a felhasználó egyedi azonosítóját, így a szerveroldalon a token HTTP kontextusból történő kiolvasása után beazonosítható, hogy a kérés kitől érkezett.

Mind a regisztrációhoz, mind a bejelentkezéshez szükségünk van arra, hogy rendelkezzünk egy biztonságos jelszóhasítási algoritmus megvalósítással. Erre a széles körben használt Bcrypt-et választjuk. A regisztráció során a jelszó hasítva kerül eltárolásra, mellyel bejelentkezéskor a megadott jelszót hasítva hasonlítjuk össze.

Ahogy arról korábban értekeztünk, az applikáció rétegek tudnia kell arról, hogy ki és milyen jogosultságokkal kezdeményezte egy utasítás végrehajtását, hogy ez alapján tudjon eljárni a végrehajtás során, valamint az ő felelőssége levezényelni a regisztrációval és bejelentkezássel kapcsolatos üzleti logikai folyamatokat is. Azonban clean architecture projektek esetén, mivel a konkrét megvalósítások biztosítása az infrastruktúra réteg feladata, az applikáció rétegen absztrakciókat definiálunk az imént tárgyalt alacsonyabb szintű feladatokra (3.5 ábra), mint

- jelszóhasítás és -verifikálás,
- JWT generálás,
- felhasználó azonosítójának HTTP kontextusból történő kiolvasása.



3.5. ábra. Autentikációt megvalósító menedzserek UML diagramja

Megjegyezzük, hogy egy kérés teljesítése során az applikáció réteg akár többször is lekérheti a `User` tulajdonságot. Mivel ez egy számolt tulajdonság - a JWT-ból kinyert ID alapján lekérjük a felhasználót az adatbázisból - így feleslegesen, egy végrehajtás alatt többször is lekérdezzük a felhasználót az adatbázisból. A problémát cacheléssel oldjuk meg, viszont nem az applikáció rétegen. Egyszerűbb, egy végrehajtás során különböző osztályokban és metódusokban felhasználásra kerülhet a `User` tulajdonság, másrészről ez nem az Ő felelőssége. Kihasználjuk, hogy minden HTTP kérés egy *scope* alá tartozik. Mivel az `IUserManager` függőséget *scoped* élet-tartalommal regisztráltuk, így megtehetjük, hogy a felhasználót a `UserManager` implementációjában cacheljük. A fentebbi UML diagramban (3.5 ábra) erről árulkodó privát tulajdonságokat láthatunk.

## 3.6. Parancsok és lekérdezések

### 3.6.1. CQRS-minta

A clean architecture iránymutatásai mentén készült ASP.NET alkalmazások implementálása során igen gyakori a CQRS-minta [11] használata. A CQRS-minta (Command Query Responsibility Segregation) lényege, hogy elkülönítsük a parancsok és lekérdezések felelősségi körét. Parancsnak azokat az utasításokat nevezzük, amelyek az adatbázis módosításával járnak. A két utasítástípus bevezetésével és szétválasztásával olvashatóbb, fenntarthatóbb kódot teremtünk, és lehetőséget biztosítunk a két felelősségi kör alá tartozó kód független fejlesztésre, optimalizálására. A CQRS-minta megvalósítására egy nagyon népszerű választás a MediatR nuget csomag [12]. A CQRS-minta ASP.NET projektekben történő megvalósításához, illetve a CQRS-minta lényegének ezen fejezetekben történő tárgyalásához a Microsoft és Milan Jovanović által írt cikkeket [13, 14], valamint a korábban hivatkozott 2 GitHub könyvtárat [5, 6] vettetem alapul.

Az implementációban tulajdonképpen minden egyes parancshoz/lekérdezéshez egy külön osztály (rekord) tartozik, amelynek az a felelőssége, hogy az adott utasítást kezelje, végrehajtsa. Ez utóbbira a MediatR biztosítja az `IRequestHandler` generikus interfészét, amely előírja a végrehajtásra szolgáló `Handle` metódust. Lényegében ez a metódus lesz meghívva az adott utasítás végrehajtásakor. Fontos, hogy a parancsok esetében az utasítássorozatot tranzakcióba foglaljuk

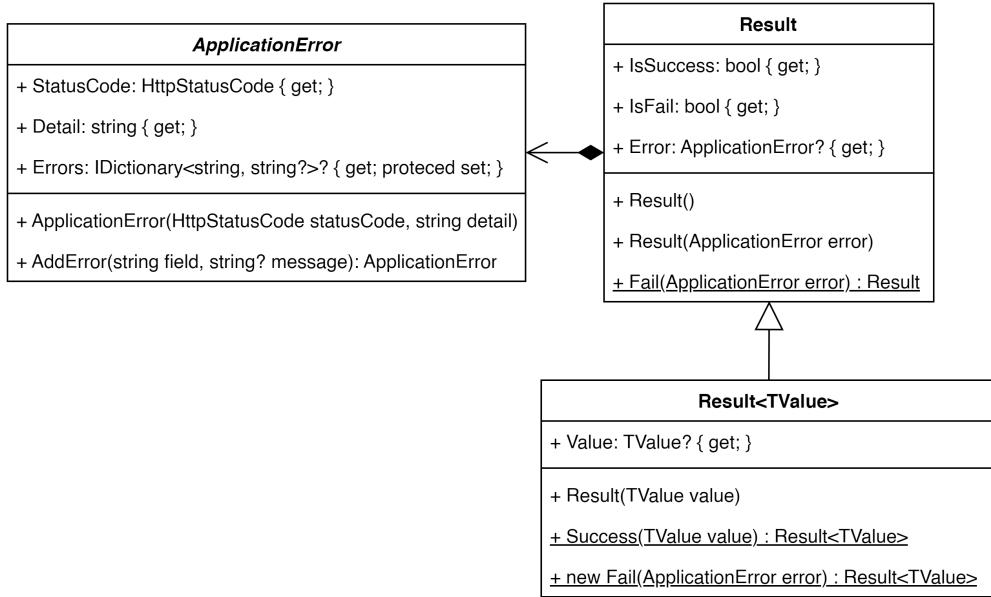
az alkalmazás aszinkronitása miatt. Az egyes parancsokat és lekérdezéseket az `Employer.Application.Features.[Entity].{Commands,Queries}` névtérben, az alkalmazás entitásai szerint csoportosítva implementáljuk.

### 3.6.2. Hibakezelés, utasítások visszatérési értéke

A hibakezelés egyik legismertebb és legkézenfekvőbb eszköze a kivétel-kezelés. A legtöbb programozási nyelv lehetőséget ad kivételek dobására és azok elkapására, megszakítva ezzel az aktuálisan futó folyamatot. Objektum-orientált nyelvekre jellemző, hogy az `Exception` osztályból történő örökléssel saját kivétel osztályokat hozhatunk létre, ezzel biztosítva azt, hogy részletesebb, a konkrét esetre szabott információkat tudunk megosztani a felmerült problémával kapcsolatban. Így ez a megoldás alkalmas lehet arra, hogy az üzleti logika végrehajtása során felmerült hibákat kivétel-kezeléssel jelezzük. Például, amennyiben a felhasználó utasítást ad egy entitás ID alapú lekérésre, és a megadott ID-vel nem létezik entitás, úgy egy saját, a hasonló esetekre újrafelhasználható `NotFoundException` dobásával jelezhetjük ezt.

Azonban ez az eset ténylegesen hibának számít? Az elvárás szerint, amennyiben egy hivatkozott entitás nem található, úgy a programnak ezt jeleznie kell. Érdemes lenne elválasztani implementáció szinten az eseteket, amikor az alkalmazásban olyan hiba történik, amire nem számítunk, illetve azokat az eseteket, amikor felhasználói esetként megfogalmazott "hiba" történik. Ugyanis, ez utóbbi tulajdonképpen nem is hiba, hanem az utasítás eredménye. Ezt a megközelítést result pattern-nek nevezik, és gyakran használják clean architecture projektekben a fentebb megfogalmazott probléma megoldására.

Az üzleti logikai hibák kezelésére bevezetjük az `ApplicationError`, valamint `Result` osztályokat. Az alkalmazásban minden utasítás visszatérési értéke egy `Result` eredmény objektum, amelyből lekérdezhető, hogy az utasítás végrehajtása sikeres volt-e, vagy sem. Továbbá, hiba esetén kiolvasható az `ApplicationError` objektum, ellenkező esetben pedig az utasítás visszatérési értéke (ez utóbbival csak akkor rendelkezünk, ha az eredmény objektum `Result<T>` típusú, ahol T a visszatérési érték típusa). Az eredmény objektum implementálását, valamint a result pattern e fejezetekben történő tanulmányozását Ben Witt és Milan Jovanović bejegyzései [15, 16] alapján végeztem. Az implementált osztályok UML diagramját a 3.6 ábra mutatja be.



3.6. ábra. Az `ApplicationError` és `Result` osztályok UML diagramja

### 3.6.3. Entitások sémái

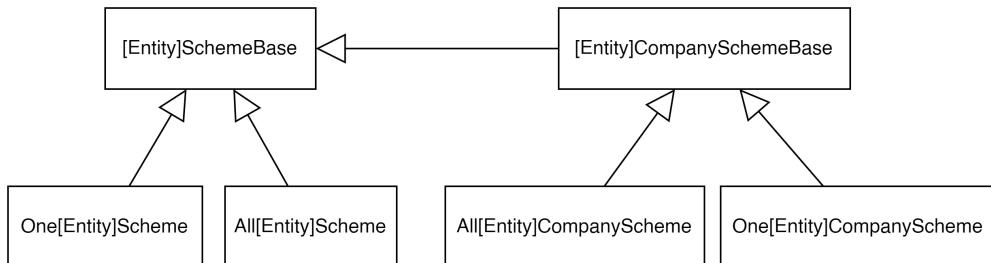
A lekérdezések (illetve néhány parancs) visszatérési értéke egy vagy több entitás. Mivel az applikáció réteg kezeli az üzleti logikai szempontból vett autentikációt, az ō felelőssége az is, hogy az egyes entitásokból "mennyit mutat". Néhány entitásnak ugyanis vannak olyan mezői, melyeket csak a vállalat felhasználói számára szeretnénk elérhetővé tenni (egy `Applayment` esetén például ilyen a `Note`, amely a recruiter észrevételeit tartalmazza a jelentkezőről). Emellett a szerver felé egy olyan elvárást is megfogalmaztunk, hogy az egyes entitásokkal együtt adjuk vissza a szükséges (és csak a szükséges) asszociált entitásokat is. Annak érdekében, hogy optimalizáljuk a lekérdezett és visszaadott adatok mennyiségét, különbséget teszünk azon esetek között, amikor 1 entitást adunk vissza, illetve amikor többet, ugyanis előbbi esetben jellemzően több asszociált entitásra lehet szükség, melyek több entitás esetében fellelhetők lehetnek. Összefoglalva tehát, két szempontot veszünk figyelembe mielőtt egy entitással visszatérnénk:

- az entitást a vállalat felhasználói számára kérdezzük le?
- 1 vagy több entitást kérdezzük le?

Mindkét esetben 2 válaszlehetőség van, így entitásonként (legfeljebb) 4 séma definíálására van szükség. Az entitást (vagy az entitásokat) ezen sémák egyikére képezzük le az adott utasítás végrehajtása során, és az így kapott DTO-t adjuk vissza ered-

ményként. Ezeket a kérdéseket a leírt sorrendben válaszoljuk meg. Mind a 4 séma esetén először vesszük az entitás azon mezőit, amelyek a felhasználó számára engedélyezettek, majd (attól függően hogy 1 vagy több entitás az eredmény) az egyes asszociált entitásokat.

Minden entitáshoz definiálunk egy `[Entity]SchemeBase` osztályt, amely a publikusan lekérdezhető mezőket tartalmazza. Továbbá - amennyiben szükséges - definiálunk egy `[Entity]CompanySchemeBase` osztályt is, amely az előző osztályból örököl - ezzel biztosítva a módosíthatóságot -, és kiegészíti a csak a vállalat számára elérhető mezőkkel (a mezők esetén ugyanis elmondható, hogy a külsős felhasználók számára látható mezők minden entitás esetében részhalmaza a vállalat felhasználói számára látható mezőknek). Ezután ezekből az osztályokból örökünk `One` és `All` prefixű osztályokat, amelyek az alapsémákat kiegészítik a szükséges asszociált entitásokkal. Ezzel elkészítettük a szükséges (legfeljebb 4) sémáinkat, melyekre az entitásokat leképezzük. A `Base` posztfixű sémák az `Employer.Application.Common.Schemes.Base` névterben, a válaszadás során ténylegesen felhasznált sémák pedig a `Employer.Application.Features.[Entity].Schemes` névterben találhatók. A sémák felépítését általánosan a 3.7 ábra mutatja be.



3.7. ábra. Entitások sémájának felépítése

Az entitások adott sémára való leképzését futási időben, az AutoMapper [17] nuget csomag segítségével valósítjuk meg. Az AutoMapper számára regisztrálni kell, hogy mely DTO-kból mely DTO-kra engedélyezzük a leképzést. Ezt az AutoMapper által biztosított `Profile` ősosztályból származó osztályokban kell megtenni. Ezek megtalálhatók a `Employer.Application.Common.Schemes` és `Employer.Application.Features.[Entity]` névterekben.

Az AutoMapper kiegészítési metódusai képesek együtt dolgozni az EF Core-ral. Amikor egy lekérdezést előkészítünk, felhasználjuk a `ProjectTo` kiegészítési metó-

dust, amely az adatbázisból történő lekérdezés közben képez le a megadott sémára. Így az adatbázisból csak azok az adatok kerülnek lekérdezésre, amelyek szükségek, illetve az asszociált entitásokat is egy azon optimalizált lekérdezésben kapjuk meg. Az AutoMapper az `IMapper` interfészen keresztül kerül befecskendezésre az azt felhasználó utasításokban.

### 3.6.4. Absztrakt végrehajtó ősosztályok

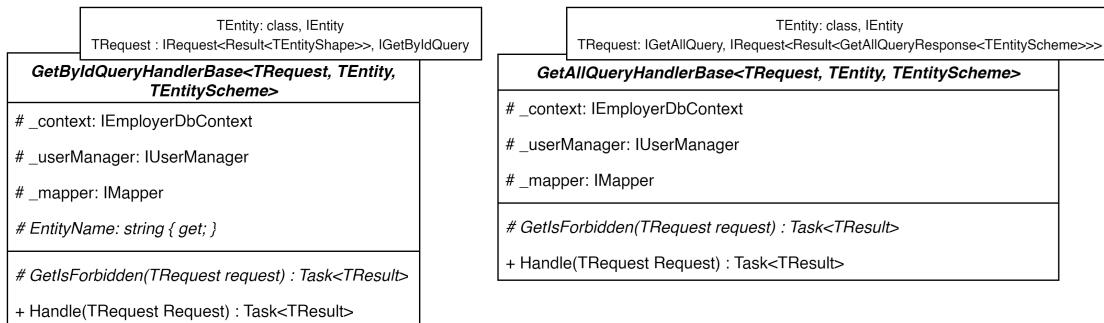
Vegyük észre, hogy egy webalkalmazás esetében sok olyan utasítás lehet, amelyek több pontban lényegében azonosan kell, hogy viselkedjenek. Tekintsük az alábbi 5 példát, illetve azok viselkedésének leírását.

- Egy meghatározott típusú entitás lekérdezése ID alapján: A típus által meghatározott táblából elsődleges kulcs alapján lekérünk 1 elemet, azt visszaadjuk amennyiben létezik, különben pedig hibaüzenetet küldünk.
- Az összes entitás lekérdezése egy meghatározott entitás típusból: A típus által meghatározott táblából lekérjük az összes entitást, alkalmazva esetleg különböző szűrőket az entitások tulajdonságaira és/vagy a visszaadott adat mennyiségére vonatkozóan.
- Egy új entitás létrehozása: Az üzleti logikai validációt követően létrehozzuk az adatbázisban az új entitást a megadott tulajdonságokkal, majd elmentjük a változtatásokat.
- Egy entitás módosítása: A megadott ID alapján lekérjük az adatbázisból az entitást, amennyiben létezik, különben pedig hibaüzenetet küldünk. Az üzleti logikai validációt követően felülírjuk az entitás meglévő tulajdonságait, majd elmentjük a változtatásokat.
- Általános, egy konkrét entitásra vonatkozó parancsok végrehajtása: A megadott ID alapján lekérjük az adatbázisból az entitást, amennyiben létezik, különben pedig hibaüzenetet küldünk. Az üzleti logikai validációt követően végrehajtjuk az érintett entitás(ok)ra vonatkozó módosításokat, elvégzünk további, az üzleti logika szempontjából szükséges lépések - amennyiben vannak -, majd elmentjük a változtatásokat.

Ezek az általánosan leírt esetek a gyakorlatban még több, az implementációból adódó ismétlődéssel rendelkezhetnek. Ezt kihasználva absztrakt ősosztályokat implementálunk, amelyekből az utasításokat kezelő osztályokat származtatjuk. Az ősosztályokban a `Handle` metódust úgy implementáljuk, hogy a megfelelő helyen absztrakt és/vagy virtuális metódusok kerüljenek meghívásra, melyeket az alosztályok felüldefiniálnak, ezáltal elérve az általuk elvárt viselkedést. Röviden: a sablonfüggvény tervezési mintát alkalmazzuk.

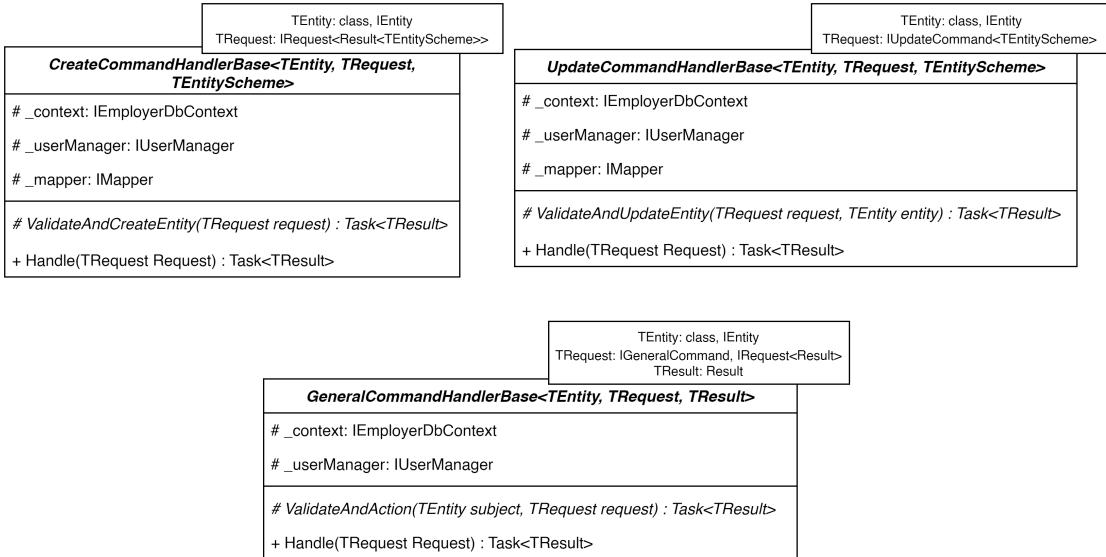
**5. Definíció.** Egy tevékenységet (sablonfüggvényt) egy ősosztály metódusaként úgy definiálunk, hogy annak speciális, altípusától függő részeit csak virtuális metódushívások jelzik (sablon-paraméterek), majd ezen metódusokat az alosztályokban az ott elvárt módon definiáljuk felül. [18]

A fentebb felsorolt első 2 esetben lekérdezést hajtunk végre, az ősosztályokat az `Employer.Application.Common.Queries` névterben implementáljuk. Ezek rendre a  `GetByIdQueryHandlerBase` és a  `GetAllQueryHandlerBase` osztályok, melyek UML diagramja a 3.8 ábrán látható.



3.8. ábra. Lekérdezést végrehajtó absztrakt ősosztályok UML diagramja

A másik 3 eset parancsvéghajtásokat ír le, az ezeket megvalósító osztályok rendre a `CreateCommandHandlerBase`, az `UpdateCommandHandlerBase`, valamint a `GeneralCommandHandlerBase`, melyek a `Employer.Application.Common.Commands` névterben találhatók, UML diagramjuk pedig a 3.9 ábrán.



3.9. ábra. Parancsot végrehajtó absztrakt ősosztályok UML diagramja

### 3.6.5. Pipelineok

A MediatR lehetővé teszi csővezetékek implementálását az `IPipelineBehavior` generikus interfész biztosításával. Így a middleware-ek használatát kiválthatjuk olyan funkcionálisok esetében, amelyek nem a web-, hanem az applikáció rétegbe tartoznak, ezzel is követve a clean architecture irányelvét. Az utasítások a végrehajtásuk előtt keresztül mennek - az általunk előzetesen meghatározott sorrendben - a pipelineokon, melyek akár meg is tudják akadályozni az utasítások végrehajtását, így kiválóan alkalmassak arra, hogy általános validációs feladatokat lassanak el. Az applikáció rétegen két pipeline osztály került implementálásra: az `AuthorizationBehavior`, valamint a `ValidationBehavior`. Ezen 2 Behavior osztály megvalósítása igen gyakori ilyen típusú projektek esetén, a korábban hivatkozott GitHub projekteket [5, 6] vettet alapul.

Az `AuthorizationBehavior` feladata az, hogy előzetes autentikációs, illetve autorizációs ellenőrzéseket hajtson végre. Az egyes utasítás rekordokat elláthatjuk ún. attribútumokkal, amelyek azt hivatottak meghatározni, hogy az utasítás végrehajtásának megkezdéséhez milyen jogosultságokkal kell rendelkeznünk. Az `Employer.Application.Common.Attributes` névtérben 3 ilyen attribútum került létrehozásra. Az alábbi táblázat azt szemlélteti, hogy a felsorolt attribútumok esetén mely felhasználók számára engedélyezett az utasítás végrehajtása.

Attribútum	Felhasználóra vonatkozó feltételek
Authorized	bejelentkezett
ForRoles([roles...])	a megadott rangok egyikével rendelkezik
ForCompanySite	vállalati ranggal (root, manager, recruiter) rendelkezik

### 3.1. táblázat. Autorizációs attribútumok által szabott feltételek

A `ValidationBehavior` felelőssége, hogy a felhasználó által megadott bemenetek validálását elvégezze. Amennyiben egy vagy több bemenet invalid, elvárjuk, hogy a hiba, amellyel a végrehajtás visszatér, tartalmazza az invalid bemeneteket és a hozzájuk tartozó indoklást. Mivel minden utasítás egyedi, így nem lehet általánosan megfogalmazni, hogy mely bemenetek számítanak validnak, és melyek nem. Ezért azon utasítások esetében, ahol validálás szükséges, implementálunk validációs osztályokat is. Ezek az osztályok az `AbstractValidator` generikus ősosztálytól örökölik, mely osztályt a FluentValidation [19] nuget csomag biztosítja. Ez a csomag segít abban, hogy könnyen, olvashatóan határozzunk meg validációs szabályokat, valamint rendelkezik beépített hibaüzenetekkel, így azokat csak speciális esetekben kell manuálisan megírni.

## 3.7. A szerveroldal belépési pontja

Most, hogy rendelkezünk azokkal az ismeretekkel, amelyek az utasítások végrehajtásával - és az ezzel járó háttérfolyamatok megvalósításával - kapcsolatosak, rátérünk a szerveroldal belépési pontjára, ahol ezen utasításokat vezérelni és kiértékelni tudjuk: a HTTP végpontokra.

### 3.7.1. Vezérlők

A végpontokat a web rétegben található vezérlőkben definiáljuk, melyeket a végpont felépítése szerint külön osztályokba szervezünk. Ezekben metódusokként definiáljuk a végpontokat, megadva azok elérési útvonalát és a vonatkozó HTTP metódust (`GET`, `POST`, `PUT`, stb.). A megfelelő utasítás mediátoron keresztül történő végrehajtását követően a kapott `Result` eredményt a `Handle` kiterjesztési metódussal HTTP válasszá alakítjuk: ez a végpont visszatérési értéke.

---

```

1 [Route("api/[controller]")]
2 public class AuthController(IMediator mediator) :
3     CustomControllerBase(mediator)
4 {
5     //...
6
7     [HttpPost("login")]
8     public async Task<IActionResult> Login([FromBody] LoginRequest
9         request)
10    {
11        var query = new LoginCommand(request.Email, request.
12            Password);
13        var result = await _mediator.Send(query);
14
15        return result.Handle(HttpStatusCode.OK, "Logged in
16        successfully");
17    }
18
19    //...
20 }
```

---

### 3.1. forráskód. Az api/auth/login végpont definiálása

A Handle kiterjesztési metódus hiba esetén az `ApplicationResult` alapján, ellenkező esetben pedig a `Result` objektum értéke, valamint a megadott HTTP-állapotkód és üzenet alapján állítja elő a HTTP választ. A Handle metódus az `Employer.Web.Controllers.Common` névtérben található `ResponseHandler` statikus osztályban került implementálásra. Result pattern használata mellett az ehhez hasonló megoldások gyakoriak.

#### 3.7.2. HTTP kérések és válaszok felépítése

Egyes HTTP kérések megkövetelhetnek olyan paramétereket, melyeket a kliensnek a kérés törzsében kell elhelyeznie. Bejelentkezés esetén ilyen a felhasználónév és a jelszó, de általánosabb példa lehet egy új entitás létrehozása, ahol az entitás tulajdonságait kell megadnunk. Segíti a kód olvashatóságát és végpontok dokumentálhatóságát az, ha ezeket az elvárt törzsparamétereket nem egyesével soroljuk fel a kontrollerek metódusaiban, hanem DTO-kat definiálunk er-

re a célra. Ezek az adatátviteli objektumok kontrollerenként csoportosítva, az `Employer.Web.Requests.[controller]` névterben találhatók. Az előző kódpéldában (3.1 forráskód) a `LoginRequest` DTO használatára láthattunk egy esetet. A `[FromBody]` annotáció jelzi, hogy az objektum által meghatározott paramétereket a kérés törzsében várjuk.

API-k esetén fontos ismertetni a válaszok felépítését, hogy a kliens megfelelő módon ki tudja olvasni abból a szükséges adatokat. Az alábbi táblázat a válaszok 5 lehetséges struktúráját tartalmazza.

Eset	Válasz struktúrája
Sikeres állapot egy entitás adatával	<ul style="list-style-type: none"> <li>• number: HTTP-állapotkód</li> <li>• message: Rendszerüzenet</li> <li>• data: Az entitás <code>One</code> prefixű sémára leképezve</li> </ul>
Sikeres állapot egy entitás adatával	<ul style="list-style-type: none"> <li>• number: HTTP-állapotkód</li> <li>• message: Rendszerüzenet</li> <li>• data: Az entitás <code>One</code> prefixű sémára leképezve</li> </ul>
Sikeres állapot több entitás adatával	<ul style="list-style-type: none"> <li>• number: HTTP-állapotkód</li> <li>• message: Rendszerüzenet</li> <li>• data: <ul style="list-style-type: none"> <li>– all: Az entitások <code>All</code> prefixű sémára leképezve</li> <li>– total: Az összes entitás száma</li> <li>– current: A lekért entitások száma</li> </ul> </li> </ul>
Hibaállapot	<ul style="list-style-type: none"> <li>• number: HTTP-állapotkód</li> <li>• details: Hibaüzenet</li> </ul>
Hibaállapot mezőnevekkel	<ul style="list-style-type: none"> <li>• number: HTTP-állapotkód</li> <li>• details: Hibaüzenet</li> <li>• errors: Kulcs-érték párok listája, ahol a kulcs a mező neve, az érték pedig a mezőre vonatkozó hibaüzenet, vagy <code>null</code></li> </ul>

3.2. táblázat. A HTTP válaszok lehetséges struktúrája

## 3.8. Videóhívás

### 3.8.1. Megvalósítás kliensoldalon

A médiaátvitel peer-to-peer módon, a hívásban résztvevő kliensek böngészői között közvetlenül történik, WebRTC technológiával, az `RTCPeerConnection` [20] interfész segítségével. Ahhoz, hogy a kapcsolat létrejöjjön, minden kliensnek el kell küldenie egy ún. *session description protocol* (SDP) objektumot a másiknak, valamint a kommunikációhoz szükséges protokollokat és az útvonalválasztást leíró ICE (Interactive Connectivity Establishment) candidate objektumok cseréjére is szükség van a résztvevők között. Miután a felek eltárolták a kapott SDP-t, létrejön a peer-to-peer kapcsolat. Az implementáció tárgyalásának megkezdéséhez az imént leírt információk a lényegesek, részletesebben a WebRTC-ről annak dokumentációjában [21] olvashatunk.

A peer-to-peer kapcsolat kezelésére a `services/interview/VideoCallHelper` osztály került implementálásra. A felületről az ezen osztály által biztosított publikus metódusok (pl. `joinAsync`, `toggleVideo`, stb.) használatával vezéreljük a hívást. A felület a hívásban történő változtatásokra az osztály által definiált események segítségével reagál. A `VideoCallHelper` függ a vele azonos névterben lévő `SocketHelper` osztálytól, mely feladata a szerverrel történő valósidejű kommunikáció kezelése. A kliensoldali implementációhoz Dennis Ivy GitHub könyvtárából [22] indultam ki.

Ahhoz, hogy kapcsolat ne csak lokálisan jöhessen létre, a túzfalak, valamint a hálózati címfordítás miatt szükséges STUN és TURN szerverek megadása. Ezt az `RTCPeerConnection` konstruktőrben tehetjük meg az `iceServers` definíálásával, a dokumentációban [21] leírt módon. Hogy a kódot e célból ne kelljen módosítani, a definíálás az `Employer.Client/env.ts` fájlba kiszervezésre került.

### 3.8.2. Megvalósítás szerveroldalon

A szerver feladata a kliensek közötti valósidejű kommunikáció biztosítása. Így adunk lehetőséget a kliensek számára az SDP objektumok cseréjére (vagyis betölthetjük a dokumentációban *signalling server*-ként megnevezett szerepet). Azonban fontos szerepe van a valósidejű kommunikációban abban is, hogy a kliensek tájékoztatást kapjanak a videóhívás eseményeiről. Ilyen események például

- a partner csatlakozása/lecsatlakozása,

- a partner egy média eszközének be-/kikapcsolása,
- szöveges chat üzenet érkezése

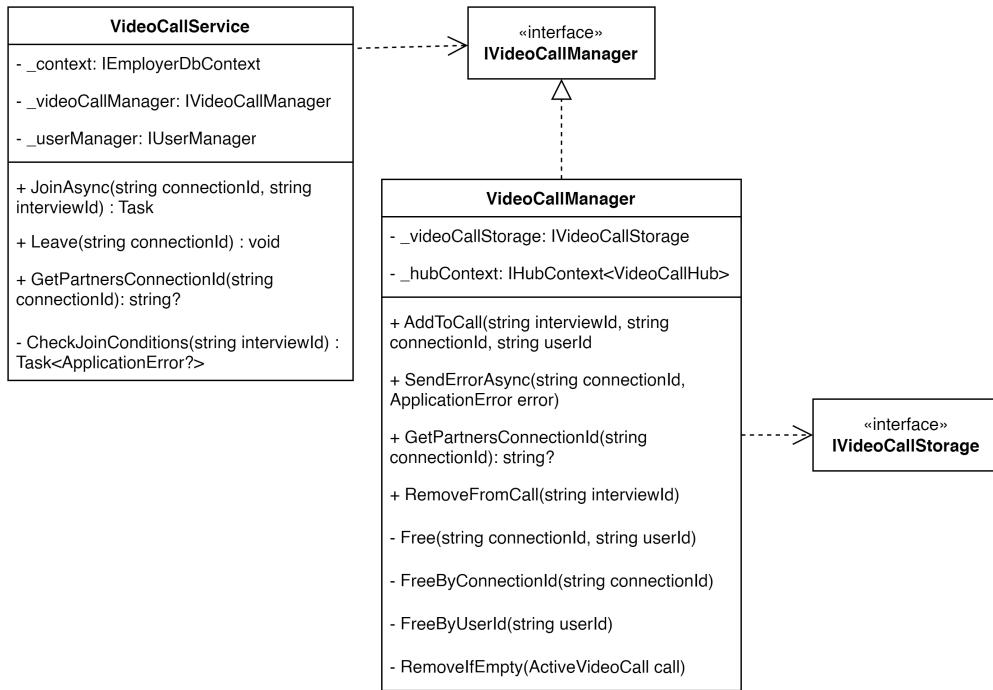
A valósidejű kommunikációt az ASP.NET keretrendszerre épülő, a nyílt forrás-kódú SignalR [23] csomag segítségével valósítjuk meg. A SignalR lehetővé teszi a kliensek ún. Hub-okhoz történő csatlakozását. minden kliens kap egy egyedi azonosítót (ez független a bejelentkezett felhasználótól és annak azonosítójától), melynek segítségével a szerver közvetlenül tud üzeneteket küldeni a kliensnek a Hub-on keresztül. Ez az azonosító a `connectionId`.

A szervernek fontos feladata az is, hogy megállapítsa, fennállnak-e az üzleti logikai feltételek ahhoz, hogy a két felhasználó között a videóhívás létrejöjjön. Biztosítani kell, hogy a videóhívásba csak a megfelelő időintervallumban, és csak az arra jogosult személyek tudjanak becsatlakozni. Gondolnunk kell arra az esetre is, amennyiben egy felhasználó egyszerre több hívásban (akár ugyanabban a hívásban kétszer) szeretne részt venni: ezt ki szeretnénk zárni.

A fentiek teljesítéséhez elengedhetetlen, hogy a `connectionId`-k felhasználókhöz történő asszociációja rendelkezésre álljon. Egy socket kérés során a kontextusból itt is ki tudjuk olvasni JWT alapján, mely felhasználó tevékenykedik, viszont fordított irányban nincs ilyen lehetőség. Szükségünk van arra is, hogy tudjuk, hogy az egyes videóhívásokban az adott pillanatban kik vesznek részt. A megoldás az, hogy a memóriában eltárolunk minden aktuálisan futó videóhívást, annak résztvevőivel együtt - ez utóbbi esetében a `connectionId` és a felhasználói azonosító megadásával. A tárolást a `VideoCallStorage` osztályban, a socket kapcsolatok menedzselését pedig a `VideoCallManager` osztályban, az `Employer.Infrastructure.Features.VideoCall` névtérben valósítjuk meg.

Az üzleti logikát az applikáció réteg `VideoCallService` osztályába szervezzük, melyben a korábban többször tárgyalt módon, absztrakciókon keresztül használjuk fel az infrastruktúra réteg implementációit.

A videóhívást szerveroldalon megvalósító osztályok UML diagramját a 3.10 ábra mutatja be.

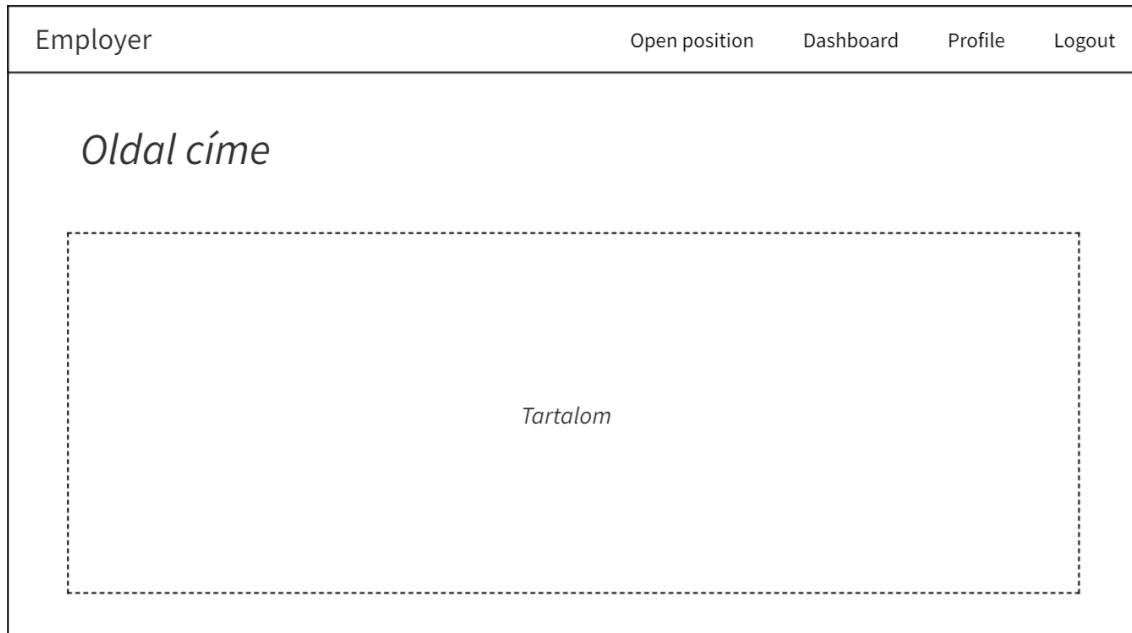


3.10. ábra. Videóhívást megvalósító osztályok UML diagramja

## 3.9. Kliens

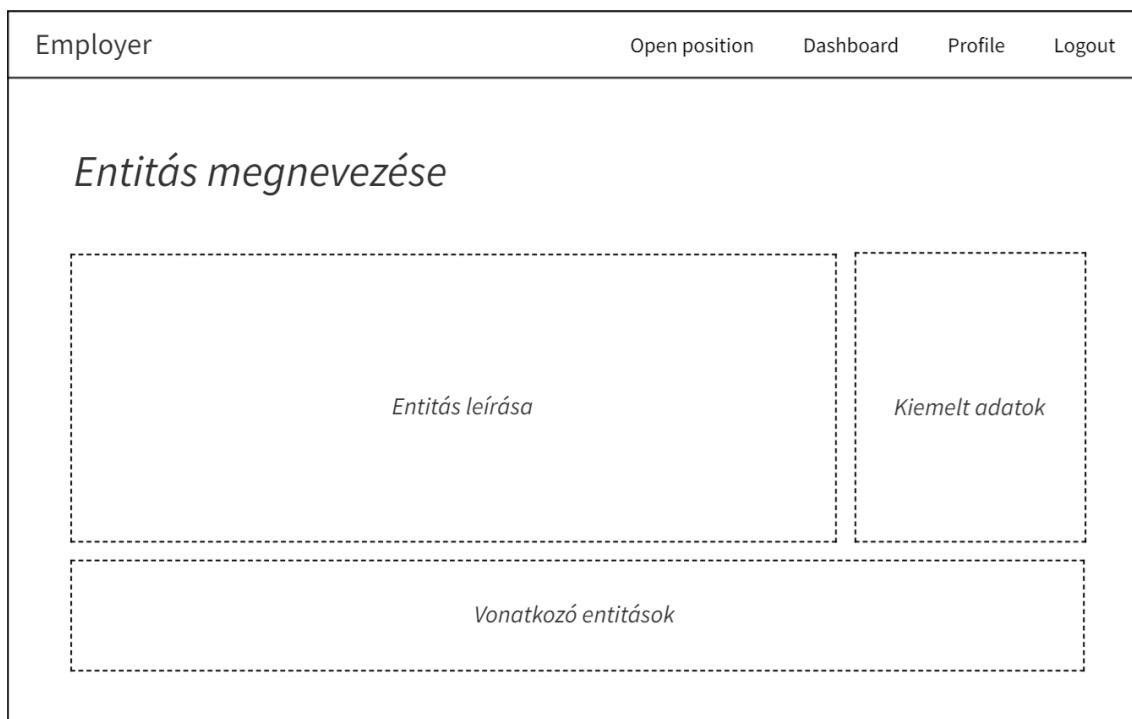
### 3.9.1. A felület tervezése

Az alkalmazás természetesen rendelkezik egy alapelrendezéssel, ahol például a navigációs sáv is helyet kap. A felület tervezése során ennek megtervezése az első lépés. A navigációs sáv az oldal tetején helyezkedik el, jobbra igazítva. Balra igazítva az oldal neve található, amely egyben kezdőlapgombként is funkcionál. Az alapelrendezés drótvázterve a 3.11 ábrán látható.



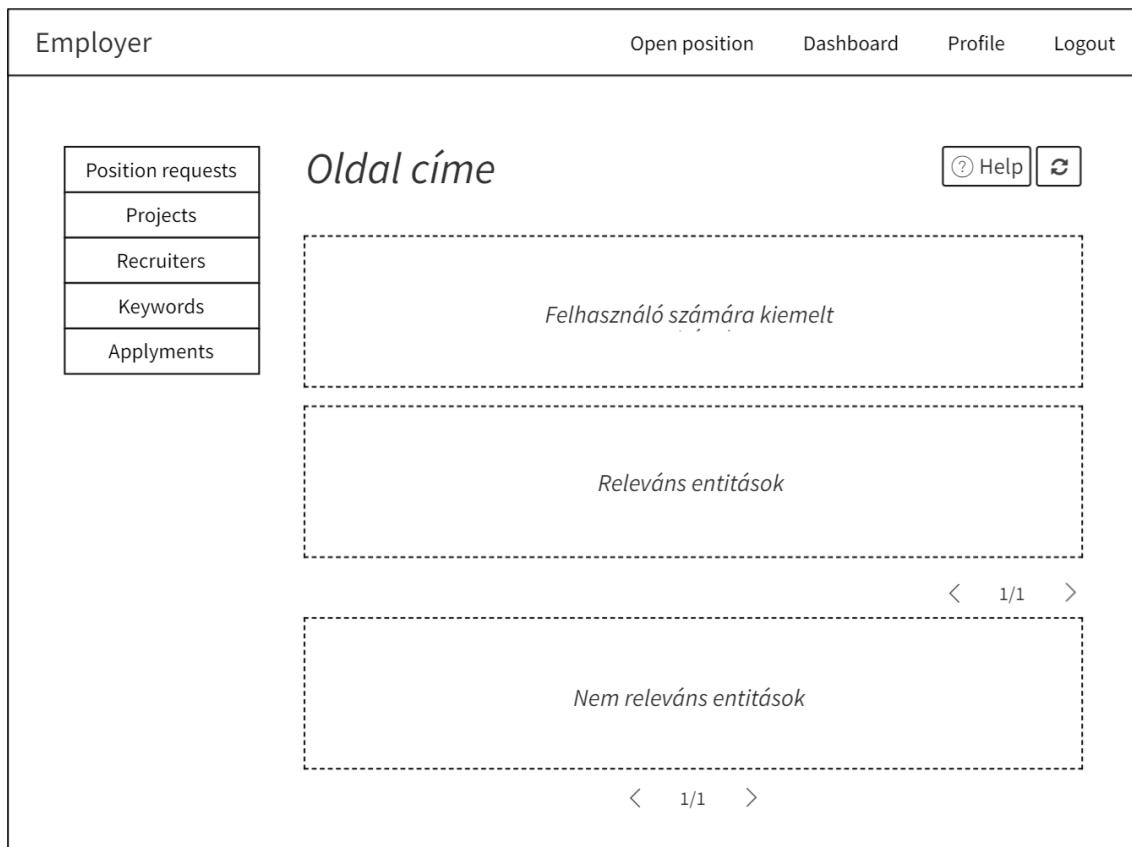
3.11. ábra. Az alapelrendezés drótvázterve

A tervezés során kiemelten fontos rendelkezni azon oldalakról, melyek publikusan is elérhetőek. Ilyen például egyes entitások (például pozíciók és projektek) megjelenítésére szolgáló oldalak. Ezek rendelkeznek leírással, rendelkezhetnek kiemelt adatokkal, valamint vonatkozó entitások felsorolásával. Az ilyen típusú oldalak drótváztervét a 3.12 ábra mutatja be.



3.12. ábra. Publikus entitást megjelenítő oldal drótvázterve

A legtöbb funkcionálitás a vezérlőpulton érhető el. A vezérlőpulton belüli navigációt tabokkal valósítjuk meg: az aktuálisan kiválasztott tab szerint jelenítjük meg az elérni kívánt oldalt. Ezeken az oldalakon jellemzően entitások kerülnek felsorolásra. Amennyiben egy entitás már nem releváns (például lezárt jelentkezés), külön felsorolásban azok is megjelenítésre kerülnek, viszont a performancia növelése érdekében csak korlátosan számban, lapozási lehetőség biztosításával. Emellett kiemelésre kerülhetnek az aktuális felhasználó számára releváns entitások, amennyiben ennek az aktuális entitás és felhasználó esetében értelme van. A vezérlőpult drótvázterve a 3.13 ábrán látható.



3.13. ábra. Vezérlőpult drótvázterve

A vezérlőpulton lehetősége van az arra jogosult felhasználóknak az egyes entitások szerkesztésére. Erre azok megtekintő oldalán van lehetőség. Amennyiben a felhasználó jogosult, úgy a beviteli mezők módosítása engedélyezett. A változtatásokat menthetjük vagy visszavonhatjuk a jobb felső sarokban elhelyezett gombbal, ahogy az a drótvázterven (3.14 ábra) is megjelenítésre kerül.

The wireframe shows a top navigation bar with links: Employer, Open position, Dashboard, Profile, and Logout. Below this is a section titled 'Entitás' with a large text input field and a 'Save changes' button with a circular arrow icon. A dashed-line box labeled 'További műveletek' (More actions) is positioned below the main input field.

3.14. ábra. Vezérlőpult szerkesztőoldal drótvázterve

A profil megtekintésére szolgáló oldalon, a felhasználó adatai mellett megjelenítünk számára releváns entitásokat is. *Candidate* felhasználó esetén például jelentkezéseit, számára ajánlott pozíciókat, kulcsszavakat. Emellett itt kap helyet a jelszó megváltoztatását kezdeményező gomb is. A profiloldal drótváztervét a 3.15 ábra mutatja be.

The wireframe shows a top navigation bar with links: Employer, Open position, Dashboard, Profile, and Logout. Below this is a section titled 'Felhasználó neve' showing email ('example@email.com') and role ('Role'). It includes a 'Change password' button and a dashed-line box labeled 'Felhasználó számára releváns' (Relevant for user).

3.15. ábra. Profiloldal drótvázterve

### 3.9.2. Állapotkezelés

Az alkalmazás állapotkezelését részben a *Redux* [24] használatával valósítottam meg, mely az egyik legnépszerűbb eszköz erre a célra. Lehetőséget ad arra, hogy olyan állapotokat kezeljünk, amelyek az alkalmazáson belül globálisan is elérhetők, így nincs szükség arra, hogy egyes állapotváltozókat (és az azokat módosító metódusokat) "lecsorgassuk" a komponensek között. Ezt kihasználjuk az autentikáció kliensoldali megvalósítására. Sok oldal és funkció reprezentációja függ attól, hogy a felhasználó be van-e jelentkezve, illetve milyen ranggal rendelkezik: ezt kiszervezzük a globális állapotba. A vonatkozó állapotot az alkalmazás betöltésekor, az `App` komponensben állítjuk be: küldünk egy kérést az `api/auth` végpontra, amely a felhasználó adataival tér vissza érvényes JWT token esetén, különben pedig 401-es hibakóddal. Az eredmény alapján beállítjuk az állapotot, melyet a ki- és bejelentkezés esetén módosítunk a továbbiakban. Az állapotkezelést a `services/auth/authSlice.ts` fájlban valósítjuk meg.

Érdemes felhasználni a Redux lehetőségeit olyan oldalak, funkciók esetében, ahol a felhasználói felületet a szokásosnál nagyobb mértékű dinamika jellemzi. Ilyen a videohívás oldala is, mely rengeteg komponensből, és annál is több állapotváltozóból áll a felhasználói élmény érdekében. Az állapotot a `views/interview/helper/interviewSlice.ts` fájlban kezeljük.

A kisebb hatókörrel rendelkező állapotok esetén a React saját, `useState` horogját használjuk. Bár ez esetben a komponensek közötti átjárást manuálisan kell megoldanunk, az optimális működést a keretrendszer biztosítja, ami fontos szempont olyan oldalak esetében, melyet a kliens böngészője renderel.

### 3.9.3. Oldalak

Minden oldal rendelkezik egy saját nézet komponenssel, melyek a `views` mapában találhatók. Amennyiben az útvonalhoz tartozik aloldal (például egy konkrét pozíció megtekintésekor a `positions/:id`), annak komponensei az almappákon belül találhatók. Az egyes nézetek a fejlesztés megkönnyítése, és a felelősségi körök elszeparálásának érdekében további komponensekre lehetnek bontva, amennyiben ez indokolt. Ezek szintén a vonatkozó mappában találhatók.

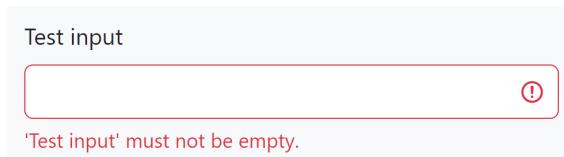
A már említett alapstruktúráját az oldalaknak (3.11 ábra) a `views/layout` mapában definiáljuk, és az egyes oldalak nézet komponenseit ezen komponensen belül

fogunk renderelni az URL alapján. Ezzel elkerüljük a kódismétlést, és biztosítjuk a módosíthatóságot,

Az URL alapján történő oldalválasztás megvalósítására a szintén széles körben használt React Router [25] könyvtárat használjuk. Az alkalmazás útvonalait az App komponensben definiáljuk. Egyes útvonalak csak a vállalat felhasználói számára érhetők el, ezért definiáljuk a `CompanySiteRoute` komponenst, mely - a korábban említett globális autentikációs állapotot felhasználva - csak abban az esetben rendezi le az oldalt, ha a felhasználó vállalati ranggal rendelkezik, különben pedig a `views/layout/errors` mappában definiált, tiltott hozzáférést jelző oldalt adja vissza.

### 3.9.4. Közös komponensek

A `components` mappában helyet kapnak azok a komponensek, amelyeknek önmagukban is van értelmük és több helyen is felhasználhatók. Ha például egy oldal nézetét bontjuk több komponensre, azoknak nem feltétlenül van önmagukban, az oldal nélkül értelme, így azok az oldal vonatkozó mappájában kapnak helyet. A közös komponensek közé tartoznak különböző egyedi gombok, bemenetek komponensei (3.16 ábra), de például különböző párbeszédablakok (pozíció létrehozás, kulcsszó választás, stb.) is.



3.16. ábra. CustomFormControl komponens - bemenet címkével és dinamikus hibaállapot megjelenítéssel

Helyet kapnak itt továbbá olyan bázis komponensek is, melyek a többi komponens ismétlődéseit szervezik ki. Ilyen többek között a `FormModal`, mely az egyes HTTP kéréseket végrehajtó felugró ablakok közös viselkedését valósítja meg.

Az egyedi komponensek alapjait, valamint az alkalmazás által felhasznált további komponenseket a szintén népszerű React Bootstrap [26] komponenskönyvtár biztosítja.

### 3.9.5. HTTP kérések kezelése

A HTTP kérések kezelésére az RTK Query-t használjuk, amely a Redux része. A könyvtár lehetővé teszi azt, hogy React horogként definiálunk különböző HTTP kéréseket, melyek lekérdezéseket (query) vagy módosításokat (mutation) hajtanak végre. Ezek a `services/api` mappában, a szerver vezérlői szerint mappákba csoporatosítva kerültek megvalósításra. A vezérlőket tükröző mappákon belül jellemzően 2 fájlt találunk. Az `ApiSlice.ts` posztfixű fájlokban találhatóak az említett HTTP kérések definíciói, míg a `Contracts.ts` posztfixű fájlokban a kérések és válaszok sémáit írjuk le. Ezek a végpontsémák azonosak azokkal, melyeket a korábbi vonatkozó fejezetben (3.6.3 fejezet) tárgyaltunk, csak most TypeScript nyelven írjuk le őket. Mivel a kliens- és szerveroldal között kódbeli összeköttetés nincs, ezért ezt a definiálást csak a TypeScript `any` kulcsszavával tudnánk megúszni, így viszont értelmét vesztené a JavaScripttel szemben történő nyelvválasztás, és futásidéjű hibák megjelenését kockáztatnánk.

A kérések definiálásának megismerése után térjünk rá azok felhasználására. A létrejött horgok rendelkeznek olyan tulajdonságokkal, melyek alapján azok állapota (folyamatban, sikeres, sikertelen, stb.) lekérdezhető. Az egyes HTTP kérések esetén fontos, hogy a hibakezelést minden esetben megvalósítsuk. Amennyiben egy olyan lekérdezés kerül sikertelen állapotba, mely az oldal megjelenítéséhez szükséges adatokat hivatott szolgáltatni, úgy ebben az esetben a kért oldal helyett hibajelző oldalt jelenítünk meg. Módosítást végrehajtó kérések esetén elég értesítést küldeni a hibaüzenetről. Ez utóbbi esetén a mutáció megváltozására a React `useEffect` horogjával figyelünk.

```

1 // hibakezeles lekerdezese eseten
2 if (query.isError)
3     return <ErrorHandlingView error={query.error} />;
4
5 // hibakezeles modositast vegrehajto keres eseten
6 useEffect(() => {
7     if (mutation.isSuccess) { /*...*/ }
8     if (mutation.isError) {
9         toast.error(getErrorDetailsOrDefault(mutation.error));
10    }
11 }, [mutation]);

```

3.2. forráskód. HTTP kérések hibakezelése kliensoldalon

## 3.10. Tesztelés

### 3.10.1. Automatikus egységesztek

Az alkalmazás szerveroldali komponensei automatikus, fehér doboz egységesztek segítségével lettek letesztelve. A clean architecture egyik fő célja az, hogy biztosítsa a könnyű tesztelhetőséget a felelősségi körök megfelelő szétválasztásával. A tesztelhetőséget segíti, hogy az implementáció során ahol csak lehetett függőség befecskendezést alkalmaztunk. Így a tesztelés során elegendő példányosítani az aktuálisan tesztelt osztályt, a függőségeket (illetve azok viselkedését) pedig a Moq [27] könyvtár segítségével mockoljuk.

Az egységesztek implementációja az NUnit [28] keretrendszer segítségével történt, amely .NET alkalmazások esetében egy igen népszerű választás. minden egységeszt egy metódus, ezek a tesztelés alá vont osztály alapján vannak tesztosztályokba csoportosítva. A metódusok elnevezése a *[tesztelt metódus neve]\_[tesztelt eset]\_[elvárt viselkedés]* konvenciót követve történt, felépítésük pedig az *arrange/act/assert* alapvető követik.

Mivel egyes osztályok esetében akár 3-4 befecskendezett függőségre is szükség van a tesztelés alá vont osztály példányosításához, sok mock létrehozására lenne szükség, és ezeket osztályszintű változókba kéne tárolnunk, hogy az egyes tesztletekben elérjük őket. Mivel rengeteg tesztelendő osztályunk van, így ez nagyon sok többlet kóddal járna. Ezért segítségül hívjuk az Autofac.Extras.Moq [29] könyvtárat, melynek segítségével dinamikusan létre tudjuk hozni a tesztelt osztályt, úgy, hogy a mockok automatikusan létrehozásra és befecskendezésre kerülnek, ezeket pedig el is tudjuk érni osztályszintű változók definiálása nélkül.

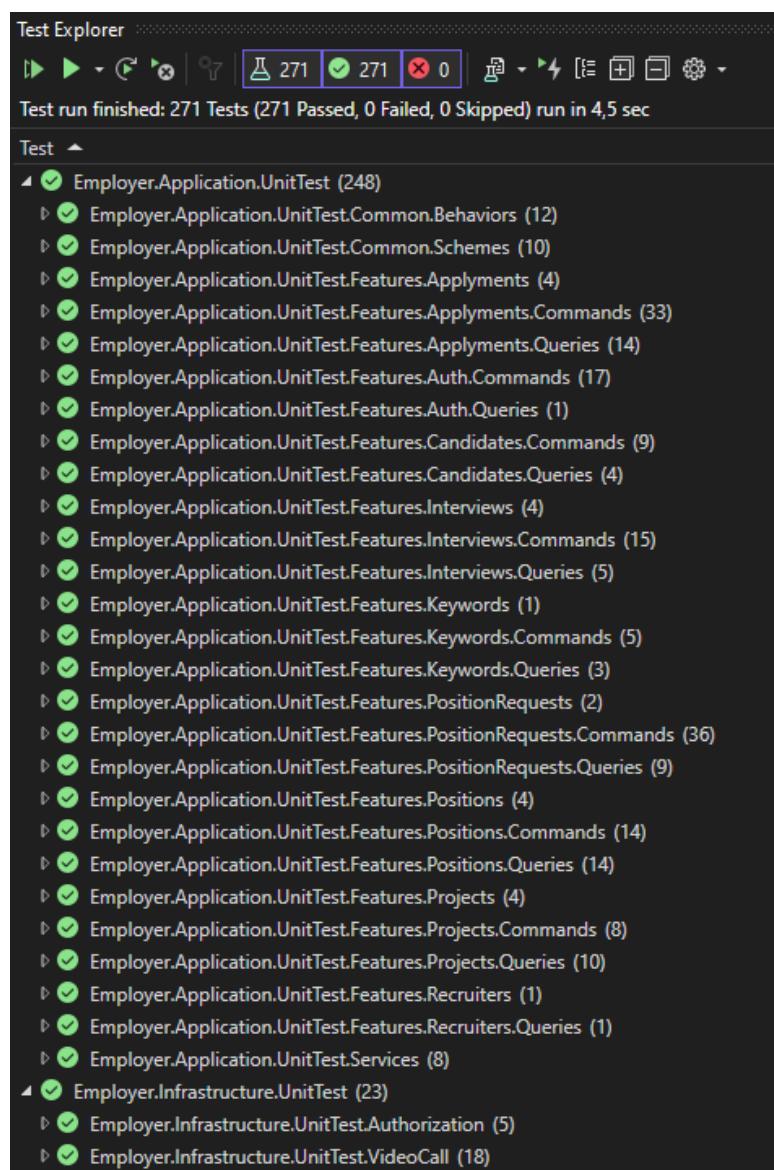
Az adatbázis mockolására a `DbTestBase` osztályt hozzuk létre, melyben olyan metódusokat definiálunk, amely megkönnyíti ezt a feladatot a tesztosztályok számára. Ebből az osztályból vagy közvetlenül tesztosztályokat származtatunk, vagy olyan osztályokat, amelyekből tesztosztályok származnak. Ezeket azért hozzuk létre, hogy az azonos beállítást megkövetelő tesztek esetén ezeket a beállításokat egy helyre szervezzük.

Automatikus egységesztek az alább felsorolt osztályok funkcionálisainak tesztelésére készültek:

- Utasítást végrehajtó osztályok
- Service osztályok

- PipelineBehavior osztályok
- Profile osztályok (entitások futásidőben történő leképzése)
- Manager osztályok

Megfigyelhető, hogy a tesztelés alá vont osztályok nagy része az applikáció réteg osztályai: az implementált egységesztek elsődleges célja az üzleti logika helyes működésének biztosítása. Az egyéb függőségeket jellemzően külső könyvtárak biztosítják, ezeket nem feladatunk tesztelni. Kivételt képez néhány általunk implementált manager osztály, ezek tesztelése szükséges. Az `Employer.Application.UnitTesting` és `Employer.Infrastructure.UnitTesting` projektekben összesen 271 teszteset került implementálásra. Ezek futási eredményét és darabszámát névtér szerint csoportosítva a 3.17 ábra mutatja.



3.17. ábra. Tesztek futási eredménye és darabszáma névtér szerint csoportosítva

Mivel utasítást végrehajtó osztályból jelentősen sok van, továbbá azonos logika mentén felépítettek, így mind a parancsok, mind az utasítások tesztelésére tesztelési terveket készítünk, melyek alapján az összes ilyen osztály letesztelhető. Az alábbi két alfejezetben ezeket tárgyaljuk.

## Parancsok tesztelése

A parancsok tesztelése az alábbi tesztelési terv szerint történt. A parancs tesztosztályának **SetUp** metódusában (amely minden egységeszt futása előtt lefut) a függőségek mockolásával szimuláljuk azt az esetet, amikor az utasítás hiba nélkül végbemegy (*happy case*), és a sikeres futás eredményét adja. Az első teszteset mindenig ezt a viselkedést ellenőrzi: ha a megfelelő körülmények rendelkezésre állnak, a parancs az elvárt módon viselkedik-e. A tesztosztály további tesztesetei az elágazások lefedésére fókuszálnak. minden parancs a végrehajtás előtt ellenőrzi, hogy a végrehajtáshoz szükséges feltételek rendelkezésre állnak-e. Ezek több, egymástól független feltételek. Mivel a **SetUp** minden beállítja a *happy case* körülményeit, nekünk elég azt az 1 feltételt "elrontani", amit éppen tesztelünk. Ezt az *arrange* fázisban tesszük meg. A **SetUp** szerepe azért is fontos, hogy biztosak legyünk abban, hogy a parancs az általunk elrontott feltétel miatt adott hibaeredményt, és nem véletlenül más miatt.

## Lekérdezések tesztelése

Az entitások ID alapján történő lekérdezését végző utasításokat aszerint teszteljük, hogy

- amennyiben az entitás megtalálható az adatbázisban, azt helyesen visszaadja,
- amennyiben nem található meg, **NotFoundError**-t kapunk,
- amennyiben nincs jogunk a lekérdezéshez, **ForbiddenError**-t kapunk.

Ez utóbbi eset opcionális, csak olyan lekérdezések tesztelésénél végezzük el, ahol a **GetIsForbidden** metódus felüldefiniálásra került.

A több entitás lekérdezését végző parancsok esetében a *happy case* teszt mellett minden egyes szűrőfeltételre készítünk tesztesetet. Ezekben azt vizsgáljuk, hogy amennyiben az adatbázisban több, eltérő tulajdonsággal rendelkező entitás van, a lekérdezés csak a szűrők által meghatározott entitásokat adja-e vissza. A szűrőket egymástól függetlenül teszteljük.

### 3.10.2. Alkalmazás manuális tesztelése

Az alkalmazás fekete doboz tesztelése manuálisan történt, mely során az elkészült alkalmazás funkciót vizsgáltam.

#### Oldalak tesztelése

A tesztesetek az alkalmazás oldalai szerint csoportosítva az alábbi táblázatban kerültek felsorolásra.

<b>Kezdőlap - /</b>
A nyitott pozíciók helyesen jelennek meg.
Több, mint 10 nyitott pozíció esetén a lapnavigáció helyesen működik.
A szűrők helyesen működnek.
<b>Profil - /profile</b>
A bejelentkezett felhasználó adatai helyesen jelennek meg.
A jelszó-megváltoztatás helyes kitöltés esetén működik.
A jelszó-megváltoztatás helytelen régi jelszó esetén jelzi a hibát.
A jelszó-megváltoztatás nem egyező új jelszavak esetén jelzi a hibát.
<b>Candidate profil - /profile</b>
A CV feltöltés, illetve - törlés helyesen működik.
A CV feltöltés 3MB-nál nagyobb, vagy nem .pdf fájl esetén hibát jelez.
A CV felöltést követően megtekinthető és letölthető.
A CV-ben szereplő, még nem hozzáadott kulcsszavak hozzáadásra kerülnek, erről értesítést kap a felhasználó.
A kulcsszavak szerkesztése helyesen működik.
A jelentkezések helyesen kerülnek felsorolásra.
<b>Recruiter profil - /profile</b>
A felhasználóhoz rendelt, még nem lezárt jelentkezések helyesen kerülnek felsorolásra.
A felhasználóhoz rendelt, még nem lezárt pozíciókiírási kérelmek helyesen kerülnek felsorolásra.
<b>Menedzser profil - /profile</b>
A menedzser projektje helyesen megjelenítésre kerül.

A menedzser számára továbbított jelentkezések helyesen felsorolásra kerülnek.
A menedzser <i>Resolved</i> állapotban lévő pozíciókiírási kérelmei helyesen kerülnek felsorolásra.
<b>Pozíció megtekintőoldala - /positions/{id}</b>
A pozíció adatai helyesen jelennek meg.
Candidate felhasználóként, ha még nem jelentkeztünk, megjelenik az <i>Apply</i> gomb, és helyesen működik.
Candidate felhasználóként, ha már jelentkeztünk, megjelenik az <i>View applyment</i> gomb, és helyesen működik.
Lezárt pozíció esetén megjelenik erről egy információs doboz, és nem lehetséges jelentkezni.
A vállalat felhasználóinak megjelenik az <i>Open in dashboard</i> gomb, és a megfelelő helyre navigál.
<b>Projekt megtekintőoldala - /projects/{id}</b>
A projekt adatai helyesen jelennek meg.
A projekt nyitott pozíciói helyesen kerülnek felsorolásra.
A vállalat felhasználóinak megjelenik az <i>Open in dashboard</i> gomb, és a megfelelő helyre navigál.
<b>Vezérlőpult - /dashboard/*</b>
Nem vállalati felhasználóként a vezérlőpult nem elérhető.
<b>Pozíciókérések fül a vezérlőpulton - /dashboard/positionrequests</b>
Az aktív és lezárt pozíciókérések helyesen kerülnek felsorolásra.
Menedzser felhasználó számára az aktív kérések közül helyesen kiemelésre a projektjéhez tartozó kérések.
Menedzser felhasználó számára megjelenik az <i>Add new</i> gomb, csak publikált projekt esetén.
Új kérések létrehozása helyesen működik.
Recruiter felhasználó esetén helyesen kerülnek kiemelésre a hozzárendelt aktív kérések.
Több, mint 10 lezárt kérés esetén megjelenik, és helyesen működik a lapozás.
<b>Projektek fül a vezérlőpulton - /dashboard/projects</b>
A projektek helyesen kerülnek felsorolásra.

Root felhasználó esetén megjelenik az <i>Add new</i> gomb.
A hozzáadás helyesen működik, azonos névvel projekt nem hozható létre.
Menedzser felhasználó esetén helyesen kiemelésre kerül a hozzá tartozó projekt.
<b>Recruiterek fül a vezérlőpulton - /dashboard/recruiters</b>
A recruiter felhasználók helyesen kerülnek felsorolásra.
Root felhasználó esetén megjelenik az <i>Add new</i> gomb.
A regisztráció helyesen működik.
<b>Kulcsszavak fül a vezérlőpulton - /dashboard/keywords</b>
A kulcsszavak helyesen kerülnek felsorolásra.
Recruiter felhasználó esetén megjelenik az kulcsszó-létrehozási lehetőség.
A létrehozás helyesen működik, azonos névvel kulcsszó nem hozható létre.
A kulcsszóra kattintva megjelenik annak szerkesztő párbeszédablaka.
A szerkesztés és a törlés helyesen működik.
<b>Jelentkezések fül a vezérlőpulton - /dashboard/applyments</b>
Az aktív és lezárt jelentkezések helyesen kerülnek felsorolásra.
Recruiter felhasználó számára az aktív kérések közül helyesen kerülnek kiemelésre a hozzárendelt jelentkezések.
Menedzser felhasználó számára az aktív kérések közül helyesen kerülnek kiemelésre a számára továbbított jelentkezések.
<b>Pozíciókérés megtékintőoldala a vezérlőpulton - /dashboard/positionrequests/{id}</b>
A kérés adatai (továbbá a hozzárendelt recruiter, a projekt, a létrehozott pozíció, a kommentek) helyesen kerülnek megjelenítésre.
A felelős menedzser a még nem lezárt kérést tudja szerkeszteni, a szerkesztést mentés előtt visszavonni.
A hozzárendelt recruiter felhasználónak megjelenik a pozíció hozzáadás lehetősége, akkor és csak akkor, ha még nincs létrehozott pozíció.
A hozzárendelt recruiter felhasználó valamint a felelős menedzser tud kommentet írni.
A comment szerzője a saját commentjét tudja törölni.
A hozzárendelt recruiter felhasználó a <i>Committed</i> állapotban lévő kérést <i>Resolved</i> állapotba tudja tenni.

A felelős menedzser <i>Resolved</i> állapot esetén el tudja utasítani vagy -fogadni a pozíciókiírást.
Elfogadás esetén a pozíció publikálásra, a projekt lezárásra kerül.
<b>Pozíció megtekintőoldala a vezérlőpulton - /dashboard/positions/{id}</b>
A pozíció adatai (továbbá a projekt) helyesen megjelennek.
A felelős menedzser a még nem publikált pozíciót tudja szerkeszteni, a szerkesztést mentés előtt visszavonni.
A pozícióhoz tartozó pozíciókiírási kérelemhez hozzárendelt recruiter felhasználó a még nem publikált pozíciót tudja szerkeszteni, a szerkesztést mentés előtt visszavonni.
A felelős menedzser a már publikált, még nem lezárt pozíciót tudja le tudja zárni.
<b>Projekt megtekintőoldala a vezérlőpulton - /dashboard/projects/{id}</b>
A projekt adatai helyesen kerülnek megjelenítésre.
A felelős menedzser a projektet tudja szerkeszteni, a szerkesztést mentés előtt visszavonni.
A felelős menedzser a még nem publikus projektet tudja publikálni, ha az rendelkezik már leírással.
A projekt publikus pozíciói helyesen kerülnek felsorolásra aszerint, hogy nyitottak vagy zártak-e.
Több, mint 10 lezárt pozíció esetén megjelenik, és helyesen működik a lapozás.
<b>Jelentkezés megtekintőoldala a vezérlőpulton - /dashboard/applyments/{id}</b>
A jelentkezés adatai (továbbá a candidate, interjú, hozzárendelt recruiter) adatai helyesen jelennek meg.
A jelentkezés adatai (továbbá a candidate, interjú, hozzárendelt recruiter) adatai helyesen jelennek meg.
A hozzárendelt recruiter írhat ki interjút, ha még nincs interjú kiírva.
A hozzárendelt recruiter törölhet interjút, ha van.
A hozzárendelt recruiter továbbíthatja, illetve elutasíthatja a jelentkezést, ha nincs <i>Pending</i> , <i>Upcoming</i> vagy <i>Ongoing</i> állapotban lévő interjúja a jelentkezésnek.
A felelős menedzser elutasíthatja vagy elfogadhatja a jelentkezést, ha továbbítva van az számára.

A felelős menedzsernek megjelennek a recruiter észrevételei, ha a jelentkezés továbbítva van számára.

### 3.3. táblázat. Oldalak tesztelését összefoglaló táblázat

#### Videóhívás tesztelése

A videóhívás tesztelését külön, az alábbi táblázatban foglaltam össze, az egyes oldalak, valamint a műveletek típusai alapján csoportosítva.

Belépés oldala
Az interjú adatai helyesen jelennek meg.
A <i>Join</i> gomb nem nyomható meg addig, amíg a médiaeszközök elérésének kérése történik.
A <i>Join</i> gomb nem nyomható meg, ha a szükséges médiaeszközélérés-engedély nem lett megadva, illetve tájékoztató felirat jelenik meg az előnézet helyén.
A <i>Join</i> gombra kattintva belépünk a hívásba.
Belépés
Elsőként belépve a hívásba a <i>Waiting video chat partner to join the call...</i> felirat jelenik meg.
A partner belépésekor létrejön a kapcsolat, megjelenik a partner videóképe.
Másodikként belépve a hívásba létrejön a kapcsolat, megjelenik a partner videóképe/neve középen.
Ha úgy lépünk be egy hívásba, hogy közben egy másik böngészőablakban is hívásban vagyunk ugyanazzal a felhasználóval, a régebbi hívásból lecsatlakoztatásra kerülünk.
Vezérlők
Saját videókép ki-/bekapcsolása helyesen működik, a bal felső sarokból eltűnik/megjelenik a videónk.
Ha a partnerünk kikapcsolja a videót, a videóképe helyére a neve kerül.
Saját hang ki-/bekapcsolása helyesen működik.

Ha a partnerünk ki-/bekapcsolja a hangját, a neve mellett szereplő mikrofon ikon helyes állapotot mutat.
A videókép méretét változtató gomb helyesen működik.
<b>Csevegés</b>
Az üzenetek helyesen, valós időben kerülnek megjelenítésre.
Legalább 1, maximum 100 karakternyi üzenetet tudunk küldeni.
Ha üzenetet kapunk becsukott csevegési ablak mellett, az csevegés ikonjának jobb felső sarkában megjelenik egy piros pötty.
<b>Kilépés</b>
A hívásból történő kilépéskor a médiaeszközök "felszabadulnak", a kilépő oldal megjelenik
Ha a partnerünk kilép a hívásból, megjelenik a <i>Waiting video chat partner to join the call...</i> felirat.

3.4. táblázat. Videohívás tesztelését összefoglaló táblázat

Manuális tesztelés esetén, amennyiben egyszerre több felhasználóval szeretnénk használni az alkalmazást (például a videohívás tesztelésekor), fontos, hogy az egyik felhasználóval a böngésző inkognitó módjában lépjünk be. Erre azért van szükség, hogy ne legyen "közös" a két felhasználó esetén a local storage, ahol a JWT is tárolásra kerül.

### Tesztelés nagy mennyiséggű adat esetén

A manuális tesztelés során vizsgáltam azt is, hogy az alkalmazás teljesítményét miként befolyásolja az, ha az adatbázist nagy mennyiséggű adattal töltjük fel. Ez elsősorban a vezérlőpulton fejtheti ki hatását, hiszen ott minden releváns adat lekérdezésre és megjelenítésre kerül. 100 egyidejűleg aktív pozíciókiírási kérelem és jelentkezés esetén (melyek relevanciájukból adódóan listázásra kerülnek a vezérlőpulton) nem volt észrevehető különbség a teljesítményben. A már nem releváns adatok minden esetben tízesével kerülnek lekérdezésre és megjelenítésre, így azok esetében nagy adatmennyiségből adódó probléma nem állhat fenn.

A teljesítményt nagy adatmennyiség esetén az segíti, hogy az asszociációkat szer-veroldalon, az adatbázisból történő lekérdezés közben végezzük el, ezzel gyorsítva a műveletet, valamint tehermentesítve a klienst. A lekérdezéseket az is optimalizálja, hogy külön sémákat definiáltunk az egy, illetve több entitás lekérdezésének esetére.

### 3.10.3. Tanulságok

Az alkalmazás egységesítését során sok helyen vettettem észre inkonziszenciát a kódban, jellemzően a régebben fejlesztett funkcionálitások tértek el az újabbak-tól pár helyen. Az olvashatóság, és a könnyebb, egységes tesztelhetőség kedvéért ezeket javítottam. Észre vettettem azt is, hogy voltak olyan metódusok manager osz-tályokban, amelyek kissé már az üzleti logika felé hajlottak. Ez megnehezítette a mockolást, és a tesztelés ott, ahol ezek használva voltak, így ezeket is javítottam. Az `IEmployerDbContext` absztrakció is korábban több metódust tartalmazott, mint a végleges verzióban. Ezek között volt olyan, amit csak nagyon nehezen lehetett volna mockolni, továbbá egyébként is felesleges volt, mert az egységes - egyébként könnyen mockolható - `DbSet` hiánytalanul át tudta venni a helyét. Üzleti logikában is felfe-deztem 2-3 hibát a tesztelésnek köszönhetően, melyek szintén javítva lettek.

A véglegesnek szánt alkalmazás manuális tesztelése során észrevettem a meg-jelenéssel kapcsolatos 1-2 inkonziszenciát (margók, placeholder szövegek eltérése), ezeket egységesítettem. Ezen kívül itt is felleltem 1-2 apróbb hibát (például, a ve-zérlőpulton volt, hogy a bal oldalon nem jól jelezte ki, épp melyik fülön vagyunk). Funkcionálással kapcsolatos hibát nem tapasztaltam.

## 3.11. Továbbfejlesztési lehetőségek

Egy alkalmazás élettartamának jelentős részét annak evolúciója teszi ki: az alkalmazás funkcionálitásának bővítése. Az Employer alkalmazás számos továbbfejlesztési lehetőséggel rendelkezik.

- Értesítési rendszer: értesítési rendszer implementálása kifejezetten hasznos le-het mind a külsős, mind a vállalat felhasználói számára. A felhasználók értesítést kaphatnának olyan esetekben, mint például jelentkezés állapotának megváltozása, pozíciókiírási kérelemmel kapcsolatos változások, stb. Egy le-hetséges megközelítés például az értesítések email-ben történő kiküldése, de

akár gondolkodhatunk olyan rendszerben is, ahol az alkalmazáson belül kap-juk meg az értesítést, esetleg minden két helyen. *Ötlet a megvalósításhoz: mivel az értesítések parancsok hatására lennének kiküldve, a parancsokat végrehajtó absztrakt ősosztályokat ki lehetne egészíteni egy olyan virtuális sablon függ-vénnyel, amely a változtatások mentését követően az értesítés kiküldéséért felel (természetesen absztrakcion keresztül, hiszen a tényleges implementációnak az infrastruktúra rétegben kell elhelyezkednie).*

- Root felhasználó tevékenységeinek kibővítése: az alkalmazás hosszútávú üze-meltetése során hasznos lehet, hogy a jelenleginél több entitás esetében tegyük lehetővé azok törlését (például nem releváns jelentkezések, lezárt pozíciók).
- Logolási rendszer: a vállalat felhasználói számára hasznos lehet egy olyan rend-szer kialakítása, amely számon tartja, hogy az egyes entitásokon mely felhasz-nálók és mit változtattak, és ezt az alkalmazás felületén a megfelelő helyen megjeleníti. Ezzel a pozíciókiírási kérelmek, jelentkezések, és egyéb vezérlőpul-ton módosított entitás életciklusa jobban visszakövethető lenne.

## 4. fejezet

### Összegzés

A szakdolgozatban egy, a clean architecture irányelveit követő webalkalmazás tervezése, fejlesztése és tesztelése került bemutatásra. Az alkalmazás felhasználói szemmel történő áttekintését a felhasználói esetek mentén végeztük, képernyőképekkel eligazítva az olvasót. A fejlesztői dokumentációban az alkalmazás rétegeinek megismérését követően az egyes részfeladatok megvalósítását tárgyaltuk, kitérve arra, hogy néhány esetben milyen döntéseket kellett meghozni, esetleg milyen kompromisszumokat kellett kötni. A tesztelésről szóló alfejezetben arról írtunk, hogy miként történt az alkalmazás validációja.

Személyes tapasztalatom az alkalmazás funkcióinak implementálása során az volt, hogy a választott architektúra valóban biztosítja a bővíthetőséget és a módosíthatóságot, a tesztelést pedig a felelősségi körök megfelelő szétválasztása, a dependency inversion principle betartása könnyítette meg jelentősen.

# Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Varga Dánielnek, hogy segítségevel hozzájárult szakdolgozatom elkészítéséhez. Köszönnettél tartozom a tanulmányaim során megszerzett tudásért a kar oktatónak, akik előadását vagy gyakorlatát hallgathattam.

# Irodalomjegyzék

- [1] *ASP.NET Core.* <https://dotnet.microsoft.com/en-us/apps/aspnet>, elérés: 2024-05-05.
- [2] *React.* <https://react.dev/>, elérés: 2024-05-05.
- [3] Giachetta Roberto. *Szoftver architektúrák alapvetései.* [https://people.inf.elte.hu/groberto/elite\\_kasz/eloadas\\_anyagok/elite\\_kasz\\_ea08\\_dia.pdf](https://people.inf.elte.hu/groberto/elite_kasz/eloadas_anyagok/elite_kasz_ea08_dia.pdf), oldal: 43, elérés: 2024-04-27.
- [4] Robert C. Martin. *The Clean Architecture.* <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>, elérés: 2024-04-28.
- [5] Jason Taylor. *CleanArchitecture.* <https://github.com/jasontaylordev/CleanArchitecture>, elérés: 2024-04-28.
- [6] Amichai Mantinband. *clean-architecture.* <https://github.com/amantinband/clean-architecture>, elérés: 2024-04-28.
- [7] Giachetta Roberto. *Szoftver architektúrák alapvetései.* [https://people.inf.elte.hu/groberto/elite\\_kasz/eloadas\\_anyagok/elite\\_kasz\\_ea08\\_dia.pdf](https://people.inf.elte.hu/groberto/elite_kasz/eloadas_anyagok/elite_kasz_ea08_dia.pdf), oldal: 39, elérés: 2024-04-27.
- [8] ELTE IK. *A közvetítő minta.* [http://webprogramozas.inf.elte.hu/tananyag/weaf1/lecke8\\_lap1.html](http://webprogramozas.inf.elte.hu/tananyag/weaf1/lecke8_lap1.html), elérés: 2024-04-28.
- [9] Cserép Máté. *Objektumrelációs adatkezelés (Entity Framework Core).* [https://mcserep.web.elte.hu/data/education/2022-2023-2\\_WAF/elite\\_waf\\_ea01.pdf](https://mcserep.web.elte.hu/data/education/2022-2023-2_WAF/elite_waf_ea01.pdf), oldal: 17, elérés: 2024-04-28.
- [10] Microsoft. *Entity Framework Core.* <https://learn.microsoft.com/en-us/ef/core/>, elérés: 2024-04-28.
- [11] Martin Fowler. *CQRS.* <https://martinfowler.com/bliki/CQRS.html>, elérés: 2024-05-05.

- [12] Jimmy Bogard. *MediatR*. <https://github.com/jbogard/MediatR>, elérés: 2024-05-05.
- [13] *Implement the microservice application layer using the Web API*. <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/microservice-application-layer-implementation-web-api>, elérés: 2024-05-05.
- [14] Milan Jovanović. *CQRS Pattern With MediatR*. <https://www.milanjovanovic.tech/blog/cqrs-pattern-with-mediatr>, elérés: 2024-05-10.
- [15] Ben Witt. *Result Pattern*. <https://medium.com/@wgyxxbf/result-pattern-a01729f42f8c>, elérés: 2024-04-28.
- [16] Milan Jovanović. *Functional Error Handling in .NET With the Result Pattern*. <https://www.milanjovanovic.tech/blog/functional-error-handling-in-dotnet-with-the-result-pattern>, elérés: 2024-05-10.
- [17] AutoMapper. *AutoMapper*. <https://github.com/AutoMapper/AutoMapper>, elérés: 2024-05-05.
- [18] Gregorics Tibor. *Testek és lények*. <https://people.inf.elte.hu/gt/oep/08.Testek%C2%A9s%C2%80L%C3%A9nyek.pdf>, oldal: 10, elérés: 2024-04-28.
- [19] FluentValidation. *FluentValidation*. <https://github.com/FluentValidation/FluentValidation>, elérés: 2024-05-05.
- [20] Mozilla. *RTCPeerConnection*. <https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection>, elérés: 2024-04-28.
- [21] *Getting started with peer connections*. <https://webrtc.org/getting-started/peer-connections>, elérés: 2024-05-05.
- [22] Dennis Ivy. *WebRTC-Simple-SDP-Handshake-Demo*. <https://github.com/divanov11/WebRTC-Simple-SDP-Handshake-Demo/>, elérés: 2024-05-05.
- [23] *Real-time ASP.NET with SignalR*. <https://dotnet.microsoft.com/en-us/apps/aspnet/signalr>, elérés: 2024-05-05.
- [24] *ReactRedux*. <https://react-redux.js.org/>, elérés: 2024-05-05.
- [25] *React Router*. <https://reactrouter.com/en/main>, elérés: 2024-05-05.

- [26] *React Bootstrap.* <https://react-bootstrap.netlify.app/>, elérés: 2024-05-05.
- [27] Daniel Cazzulino. *moq.* <https://github.com/devlooped/moq>, elérés: 2024-05-05.
- [28] *NUnit.org.* <https://nunit.org/>, elérés: 2024-05-05.
- [29] Autofac. *Autofac.Extras.Moq.* <https://github.com/autofac/Autofac.Extras.Moq/>, elérés: 2024-05-05.

# Ábrák jegyzéke

2.1.	Az alkalmazás konzolja annak indítását követően	8
2.2.	Az alkalmazás főoldala	9
2.3.	Nyitott pozíciót reprezentáló kártya	10
2.4.	Több oldalon keresztül felsorolt entitások	10
2.5.	Példa felugró értesítésre	10
2.6.	Példa mezővalidációs hibára	12
2.7.	Példa megerősítésére szolgáló párbeszédablakra	12
2.8.	Recruiter felhasználó regisztrálása	13
2.9.	A vezérlőpulton kilistázott projektek	14
2.10.	Publikálásra készen álló projekt szerkesztőfelülete	15
2.11.	Vállalati entitások regisztrálására vonatkozó használati eset diagram	15
2.12.	Pozíciókiírási kérelem létrehozása	16
2.13.	Pozíciókiírási kérelem szerkesztőoldala recruiter hozzárendelése után	17
2.14.	Kulcsszavak hozzárendelése pozícióhoz	18
2.15.	Kulcsszavak listázása a vezérlőpulton	18
2.16.	Pozíciók meghirdetésének folyamatát leíró használati eset diagram	20
2.17.	Candidate profiloldala	21
2.18.	Candidate profiloldala önéletrajz feltöltése után	21
2.19.	Candidate felhasználó regisztrálásának folyamatát leíró használati eset diagram	22
2.20.	Pozíció oldala jelentkezés után	23
2.21.	Jelentkezés megtekintőoldala	24
2.22.	Jelentkezés megtekintőoldala a vezérlőpulton, hozzárendelést követően	25
2.23.	Jelentkezés megtekintőoldala a vezérlőpulton, interjúra történő meghívást követően	26
2.24.	Jelentkezés és elbírálásának folyamatát leíró használati eset diagram	27
2.25.	Videóinterjú belépési oldala a médiaeszközök engedélyezését követően	28

2.26. Videóhívás felülete, a beszélgetőpartner kikapcsolt kamerájával . . . . .	29
2.27. Videóhívás kilépő felülete . . . . .	29
2.28. Videóinterjú folyamatát leíró használati eset diagram . . . . .	30
 3.1. Szükséges fejlesztői komponensek . . . . .	33
3.2. Az alkalmazás rétegei és azok egymásra épülésének szemléltetése . . . . .	34
3.3. Az <code>IEmployerDbContext</code> interfész UML diagramja . . . . .	37
3.4. Az alkalmazás egyedkapcsolat diagramja SQL adattípusokkal . . . . .	39
3.5. Autentikációt megvalósító menedzserek UML diagramja . . . . .	40
3.6. Az <code>ApplicationError</code> és <code>Result</code> osztályok UML diagramja . . . . .	43
3.7. Entitások sémájának felépítése . . . . .	44
3.8. Lekérdezést végrehajtó absztrakt ősosztályok UML diagramja . . . . .	46
3.9. Parancsot végrehajtó absztrakt ősosztályok UML diagramja . . . . .	47
3.10. Videóhívást megvalósító osztályok UML diagramja . . . . .	53
3.11. Az alapelrendezés drótvázterve . . . . .	54
3.12. Publikus entitást megjelenítő oldal drótvázterve . . . . .	54
3.13. Vezérlőpult drótvázterve . . . . .	55
3.14. Vezérlőpult szerkesztőoldal drótvázterve . . . . .	56
3.15. Profiloldal drótvázterve . . . . .	56
3.16. CustomFormControl komponens - bemenet címkével és dinamikus hi- baállapot megjelenítéssel . . . . .	58
3.17. Tesztek futási eredménye és darabszáma névtér szerint csoportosítva .	61