**Tartalomjegyzék**

[**1.** **Előszó** 2](#_Toc71316635)

[**2.** **A C# programozási nyelv** 2](#_Toc71316636)

[**3.** **A TypeScript programozási nyelv** 2](#_Toc71316637)

[**4.** **Az Angular2+ kliens oldali keretrendszer** 3](#_Toc71316638)

[**5.** **A .NET 5 alapú szerver oldali programozás** 4](#_Toc71316639)

[**6.** **Alkalmazott fejlesztői környezetek** 5](#_Toc71316640)

[**7.** **Követelményspecifikáció** 6](#_Toc71316641)

[**7.1.** **Funkcionális követelmények** 6](#_Toc71316642)

[**7.2.** **Nem funkcionális követelmények** 7](#_Toc71316643)

[**8.** **Az alkalmazásban található jogkörök** 8](#_Toc71316644)

[**8.1.** **Adminisztrátor szerepkör** 8](#_Toc71316645)

[**8.2.** **Ügyvezetői szerepkör** 8](#_Toc71316646)

[**8.3.** **HR vezetői szerepkör** 9](#_Toc71316647)

[**8.4.** **Gazdasági vezető szerepkör** 9](#_Toc71316648)

[**8.5.** **Munkatárs szerepkör** 9](#_Toc71316649)

[**9.** **Adatbáziskezelés** 9](#_Toc71316650)

[**9.1.** **Az adatbáziskezelő rendszerek és az adatbázisok** 9](#_Toc71316651)

[**9.2.** **Az adatbázis-kezelő rendszer meghatározása** 10](#_Toc71316652)

[**9.3.** **EntityFramework Core ORM** 10](#_Toc71316653)

[**9.4.** **Felhasználók kezelése az adatbázisban** 12](#_Toc71316654)

[**9.5.** **Az adatbázisban használt táblák** 14](#_Toc71316655)

[**9.5.1.** **AspNetUsers tábla** 14](#_Toc71316656)

[**9.5.2.** **AspNetRoles tábla** 14](#_Toc71316657)

[**9.5.3.** **AspNetUserRoles tábla** 15](#_Toc71316658)

[**9.5.4.** **ApplicationLog tábla** 15](#_Toc71316659)

[**9.5.5.** **Device tábla** 15](#_Toc71316660)

[**9.5.6.** **DeviceHistory tábla** 16](#_Toc71316661)

[**9.5.7.** **Software tábla** 16](#_Toc71316662)

[**9.5.8.** **SoftwareHistory tábla** 16](#_Toc71316663)

[**10.** **Alkalmazott programtervezési minták** 17](#_Toc71316664)

[**10.1.** **Command and Query Responsibility Segregation (CQRS) minta.** 17](#_Toc71316665)

[**10.2.** **Mediator programtervezési minta** 18](#_Toc71316666)

[**11.** **Az alkalmazás architektúrális felépítése** 18](#_Toc71316667)

[**12.** **Globális hibakezelés és a szerver oldali hibák naplózása** 20](#_Toc71316668)

[**13.** **Az alkalmazásban található felületek** 22](#_Toc71316669)

[**13.1.** **Bejelentkezés** 22](#_Toc71316670)

[**13.2.** **404 Oldal** 22](#_Toc71316671)

[**13.3.** **Felhasználók listája** 22](#_Toc71316672)

[**13.4.** **Felhasználó regisztrálása** 22](#_Toc71316673)

[**13.5.** **Felhasználó szerkesztése** 22](#_Toc71316674)

[**13.6.** **Eszközök listája** 22](#_Toc71316675)

[**13.7.** **Eszköz létrehozása és szerkesztése** 22](#_Toc71316676)

[**13.8.** **Szoftverek listázása** 22](#_Toc71316677)

[**13.9.** **Szoftver létrehozása és szerkesztése** 22](#_Toc71316678)

[**13.10.** **Egy eszköz életútjának megtekintése (melyik felhasználóknál volt milyen időtartományban)** 22](#_Toc71316679)

[**13.11.** **Egy szoftver életútjának megtekintése (melyik felhasználóknál volt milyen időtartományban)** 22](#_Toc71316680)

[**13.12.** **Bejelentkezett felhasználóhoz tartozó eszközöket listázó felület** 22](#_Toc71316681)

[**13.13.** **Bejelentkezett felhasználóhoz tartozó szoftvereket listázó felület** 22](#_Toc71316682)

[**14.** **Esettanulmány** 22](#_Toc71316683)

[**14.1.** **Adminisztrátor szerepkör – Felhasználó regisztrálása** 22](#_Toc71316684)

[**14.2.** **HR Vezető szerepkör – Felhasználó módosítása** 22](#_Toc71316685)

[**14.3.** **Gazdasági vezető szerepkör – Egy eszköz áthelyezése más felhasználóra, azaz az eszközmozgás bemutatása.** 22](#_Toc71316686)

[**14.4.** **Munkatárs szerepkör – Saját szoftverek lista megtekintése** 22](#_Toc71316687)

[**14.5.** **Ügyvezető szerepkör – A rendszerben található szoftverek exportálása Excel és PDF formátumba.** 23](#_Toc71316688)

[**15.** **Utószó** 23](#_Toc71316689)

[**Bibliográfia** 24](#_Toc71316690)

# **Előszó**

Szakdolgozatom célja, hogy bemutassam, a szerveroldali C# programozási nyelv, illetve a kliens oldali Angular2+ (Typescript programozási nyelv) keretrendszer lehetőségeit, egy olyan webes alkalmazáson keresztül, amely lehetővé teszi, egy általam elképzelt kisvállalat teljes hardver és szoftverleltár kezelését.

# **A C# programozási nyelv**

A C# (kiejtése: szí-sárp), a .NET egyik fő programozási nyelve. Maga a nyelv alapjaiul a C++ és a Java programozási nyelvek szolgálnak. A ’90-es években a Microsoft úgy döntött, hogy a saját Java környezetét különböző szolgáltatásokkal ki fogja bővíteni, azonban a Java programozási nyelv licencét más cég birtokolta, így a Microsoft beperelését követően (ugyanis ehhez a bővítésekhez nem volt semmilyen engedélye), a cégnek kötelezően el kellett távolítani a Javat a Windows operációs rendszerekből. A Microsoft végül egy saját keretrendszerbe kezdett, ez lett a .NET (kiejtése: dotnet), és ehhez adta ki programnyelvként a C# első verzióját.

A C# tervezésében az egyik legfontosabb szempont az volt, hogy az egyensúlyt megteremtse a fejlesztő nyelvi szabadsága és a gyors alkalmazásfejlesztés lehetősége között, továbbá a legkönnyedebb módon tükrözi az alatta működő .NET keretrendszert.

A C# programozási nyelvet összehasonlíthatnánk más programozási nyelvekkel is, mint például a C vagy a C++ programozási nyelvek, azonban ezekkel szemben a C# az több korlátozást, és továbbfejlesztést is tartalmaz az utóbbiakhoz képest. Néhány, ezen lehetőségek közül:

* Csakis Unsafe Mode-ban használhatóak a mutatók.
* A programozási nyelv csak egyszeres öröklődést támogat.
* A C++ -hoz képest a C# az egy sokkal típusbiztosabb nyelv.

# **A TypeScript programozási nyelv**

Maga a TypeScript (kiejtése: tájp-szkript), az egy objektum-orientált script nyelv, amit maga a Microsoft készített. A legfőbb filozófiája a programozási nyelvnek az, hogy maga a TypeScript programozási nyelv, az egy bővebb halmaza legyen a JavaScript programozási nyelvnek. A JavaScript programozási nyelvvel szemben, a legnagyobb újdonsága az, hogy ez egy statikusan típusos nyelv.

A nyelv egyik legfőbb célja, hogy egy olyan eszközt biztosítson a programozók/ fejlesztők számára, amely nagy segítséget nyújt egy igazán nagy és komplex rendszer elkészítésében.

Maga a TypeScript nyelv az egy teljesen nyílt forráskódú programozási nyelv, teljesen mértékben operációs rendszer független, továbbá mivel maga a fordító a TypeScript típusú forráskódból JavaScript kódot generál így az is kimondható, hogy ez a nyelv teljesen böngészőfüggetlen is, futtatásához nem szükséges semmilyen külső program vagy pedig bővítmény telepítése sem.

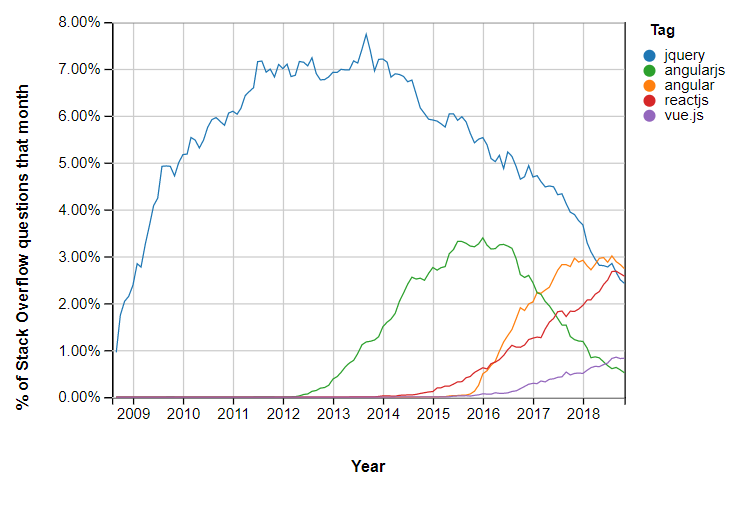
# **Az Angular2+ kliens oldali keretrendszer**

Az Angular (kiejtése: angulár), egy Google által fejlesztett, nyílt forráskódú JavaScript keretrendszer, amelyet dinamikus webes alkalmazás készítéshez használnak. Maga a keretrendszer segítéségével, nagyban egyszerűsödik a webes alkalmazások kliens oldali fejlesztése.

Az első verziója még 2010-ben jelent meg angularJS néven, azonban 2016-ban, ez a keretrendszer egy teljes újragondoláson és ezt követően egy teljes újraíráson esett át. Ekkor jelent meg a már ismert Angular2+, amely egy teljesen független keretrendszernek tekinthető.

Maga az Angular2+ keretrendszer, az TypeScript programozási nyelvet használ, de támogatja a más típusú JavaScript kódok írását is.

Napjainkban az Angular2+ már a stabil 11-es verziónál jár, amely folyamatos támogatottságot élvez, továbbá egy teljesen stabil kliens oldali keretrendszer ugyan úgy, mint például egy React, Vue.js vagy pedig egy jQuery kliens oldali keretrendszer. A keretrendszer technológiai követése, az új verziókra történő átállásra teljesen rugalmas, megszámlálhatatlan harmadik fél által fejlesztett komponens könyvtár áll rendelkezésre a fejlesztők számára, továbbá akár (cross-platform) mobilalkalmazás is fejleszthető a segítségével a webes felület mellé. A világ egyik legnépszerűbb JavaScript keretrendszerének mondható.



. ábra A JavaScript alapú kliens oldali keretrendszerek népszerűségét prezentáló ábra

Az Angular2+ kliens oldali keretrendszer előnyei egy jQuery kliens oldali technológiával szemben:

* Látványos, könnyen kezelhető felületek fejleszthetőek a segítségével.
* Funkciókban gazdagabb eszközök/ komponensek érhetők el.
* A komplex felületek sokkal jobban karbantarthatóbbak és hibatűrőbbek.
* Komponens/ template alapú fejlesztés, azaz a komponensek újrahasznosíthatóak és ennek köszönhetően a fejlesztés során egy jól átlátható kódbázis áll rendelkezésre.
* Kétirányú adatkötésének köszönhetően, a felületeknek sokkal gyorsabb az adatkiszolgálása és a betöltése.
* A szerver oldal teljes mértékben elválik a kliens oldaltól, ezzel sokkal rugalmasabb üzemeltetést biztosít.
* A Microsoft új, utóbbi évek szerver oldali webes technológia fejlesztéseivel a legkompatibilisebb kliens oldali keretrendszer.

# **A .NET 5 alapú szerver oldali programozás [.net 7.0]**

Az elmúlt években a Microsoft technológiájával fejlesztett alkalmazások szerver oldali réteg alapját képző .NET keretrendszer folyamatosan fejlődött. A jelenleg legmagasabb és egyben utolsó verziója a 2019-ben kiadott .NET Framework 4.8, amihez a továbbiakban már csak hibajavítást tartalmazó csomagok várhatóak.

A jövőben a .NET Framework helyét, a Microsoft számos verziót megélt nyílt forráskódú, mára már teljes mértékben stabil Cross-Platform keretrendszere, a .NET Core veszi át. 2020. novemberében a Microsoft kiadta a .NET Core legújabb verzióját, amiből már a „Core” márkanevet elhagyva - és a 4.0 verziót kihagyva, hogy elkerüljék a .NET Framework 4.x-el történő összetévesztését - .NET 5.0 néven jegyez.

A .NET Core keretrendszer használatának előnyei:

* A teljesítményt a Microsoft verzióról verzióra folyamatosan optimalizálja, aminek köszönhetően a .NET Core-ban írt alkalmazások sokkal optimalizáltabban futnak, elődjeihez képest.
* A harmadik fél által készített különféle csomagok készítői is már túlnyomórészt .NET Core-ra fejlesztik, optimalizálják és tesztelik a termékeiket.
* A dokumentációk/ forrásanyagok elérhetősége a .NET Core esetén könnyebben és nagyobb mennyiségben elérhetőek, mint .NET Framework esetén.
* Cross-Platform támogatottság, azaz működik Linux környezetben is, ahol sokkal biztonságosabb az üzemeltetés, rendszerint sokkal olcsóbbak a licenszek és jóval alacsonyabbak az üzemeltetési költségek is.
* Az új irányelveknek megfelelően támogatja az elosztott rendszerek fejlesztését is:
  + Konténerizáció (pl: Docker).
  + Felhő alapú üzemeltetés (Azure).
  + Jó skálázási lehetőség.
* A .NET Core kifejezetten jól támogatja az Angular2+ kliens oldali keretrendszert.
* A .NET Core-ban az Angular száméra készített Web API-t könnyedén használhatják akár más alkalmazások is (pl.: asztali alkalmazás, mobilalkalmazás vagy akár más szerverek is csatlakozhatnak az API-hoz, ha szükséges).

# **Alkalmazott fejlesztői környezetek**

A mai modern szoftverfejlesztés világában, a programozók számára elengedhetetlen eszközök az integrált fejlesztői környezetek (Integrated Development Environment, **IDE**). Ezek a fejlesztői környezetek egy olyan szoftveralkalmazások, amelyek egy átfogó eszközt biztosít a számítógépes programozók számára a szoftverfejlesztéshez.

Egyes IDE-k egy-egy konkrét adott programozási nyelvhez vannak elkészítve, azonban több olyan eszköz létezik már, amely támogatja a többnyelvűséget.

Az általam választott technológiákhoz, két különböző IDE-t választottam, amelynek oka a technológiák különbségéből adódó széleskörű fejlesztői támogatottság volt.

A szerver oldali C# programozáshoz a Microsoft integrált fejlesztő környezetét használtam, az ingyenesen használható **Microsoft Visual Studio 2019 Community Edition**-t. [2022] A terméknek ezen kívül még két verziója létezik, azonban az alap alkalmazás elkészítéséhez az ingyenes verziótól többre egyáltalán nem volt szükség továbbá ezek az eszközök már licensz kötelesek. Ez a fejlesztői környezet az egyik legelterjedtebb IDE a C# programozáshoz, ugyanis nagyon sok támogatást nyújt a programozók számára, mint például az AI-jal támogatott kódkiegészítés, Diagnostic Tools, Git repository támogatottság, továbbá nagyon sok harmadik fél által nyújtott Extension-ök érhetőek el, amelyek segítségével a fejlesztő teljes mértékben személyre tudja szabni a programozáshoz használt környezetét.

A kliens oldali Angular2+ - Typescript programozáshoz, szintén a Microsoft által kiadott integrált fejlesztő környezetet használtam, az ingyenesen használható **Visual Studio Code**-ot. A termék, az egy ingyenes, nyílt forráskódú kódszerkesztő, amely mind elérhető Windows, Linux és az OS X operációs rendszerekhez. Ez a fejlesztői környezet a legelterjedtebb a webes kliens alkalmazások fejlesztéséhez, ugyanis nagyon sok nyelvi támogatás létezik hozzá, azonban itt meg kell, hogy jegyezzem, hogy ezeket a támogatásokat alapból a szoftver nem tartalmazza, hanem külön extensionként kell feltelepíteni hozzá. Továbbá maga az IDE ugyan úgy képes az intelligens kódkezelésre, beépített Console-al és Git repository támogatással rendelkezik, továbbá ugyan úgy ehhez az környezethez is nagyon sok harmadik fél által nyújtott csomagok érhetőek el, amelyek segítségével a fejlesztő teljes mértékben személyre tudja szabni a programozáshoz használt környezetét.

# **Követelményspecifikáció**

# **Funkcionális követelmények**

Az elkészült alkalmazás funkcionális követelményei azokat határozza meg, hogy magának a rendszernek milyen szolgáltatásokat/ funkciókat kell ellátnia, továbbá hogyan kell reagálnia az adott bemenetek esetén, illetve, hogy hogyan kell adott helyzetekben viselkednie.

Az elkészült szoftver legfőbb célja az, hogy kiváltsa az esetleges papír alapon, vagy más szoftverek által kezelt (például egy táblázatkezelő alkalmazás) leltározási formulákat, így egy sokkal egyszerűbb, modernebb, átláthatóbb és gyorsabb munkavégzést biztosító webes felületet biztosít az alkalmazást használó felhasználók számára.

Az alkalmazás funkcionális követelményei pontokban meghatározva:

* A rendszer több felhasználói szintet tud meghatározni/ kezelni, így a különböző felhasználói szintek külön jogosultsággal rendelkeznek az alkalmazáson belül.
* A rendszer, az ő hozzá tartozó adatbázissal tudnia kell kapcsolatot létesíteni, továbbá ezen kapcsolaton keresztül lehetőséget kell, hogy biztosítson a felhasználóknak az új adatok beszúrására, azoknak lekérdezésére, esetleges módosítására és törlésére, a megfelelő jogosultsági szerepkörökön belül.
* Lehetőséget biztosít arra, hogy a megfelelő jogosultsági szerepkörökön belül új felhasználókat tud hozzáadni a már a rendszerben található jogosultsági szint meghatározásával, új eszközök és szoftverek felvételére, majd azoknak módosítására és törlésére.
* Lehetőséget biztosít arra, hogy az egyes eszközöket és szoftvereket, amelyek megtalálhatóak a rendszer adatbázisában, azokat felhasználóhoz kössék a megfelelő szerepkörökkel rendelkező személyek.
* Lehetőséget nyújt arra, hogy a rendszer adatbázisában található eszközöket és szoftvereket összeségében.
* A rendszer lehetőséget biztosít, az egyedi azonosítóval ellátott eszközök és szoftverek nyomon követésére.
* Az alkalmazás lehetőséget biztosít arra, hogy a felhasználók a kívánt adatokat le tudják menteni Excel formátumban.

# **Nem funkcionális követelmények**

Az elkészült alkalmazás nem funkcionális követelményei azokat határozza meg, amelyek a funkcionális követelményekkel ellentétben, a rendszer egészére vonatkozó rendszertulajdonságokkal foglalkoznak.

Az alkalmazás nem funkcionális követelményei pontokban meghatározva:

* A rendszerben található felhasználóknak egyedi azonosítóval kell rendelkezniük.
* A rendszernek biztonságos authentikációt és token alapú authorizációt kell biztosítania a felhasználók számára.
* A rendszerben található felhasználók szenzitív adatát, azaz a jelszót biztonságos módon kell tárolnia, tehát HASH-elve kell eltárolnia, nem visszaolvasható módon.
* Az alkalmazás felhasználói felületének felhasználóbarátnak, jól átlátható és könnyen kezelhetőnek kell lennie a szoftver ergonómiák szem előtt tartásával.
* A felhasználói felületeknek több különböző/ különféle eszközökön is jól átláthatónak továbbá könnyen kezelhetőnek kell lennie, azaz reszponzivitást kell, hogy biztosítson.

# **Az alkalmazásban található jogkörök**

Egy irodai szoftver egyik legfontosabb alappillére az alkalmazásban található jogkörök, azaz maga a szoftverben található jogosultsági rendszer. A jogosultsági rendszerek legfőbb célja, hogy az egyes felhasználók korlátozva legyenek az egyes funkciók elérésének céljából, ugyanis, ha minden felhasználó el tudna érni és végre tudna hajtani minden műveletet, akkor az üzleti folyamat egy vállalatra nézve, teljesen kontrollálatlanná tudna válni.

Az alkalmazáson belül öt különböző jogkört/ szerepkört határoztam meg, amelyek az alábbiak: Adminisztrátor, Ügyvezető, HR vezető, Gazdasági vezető, Munkatárs.

# **Adminisztrátor szerepkör**

Minden alkalmazásban szükség van egy adminisztrátori szerepkörre, továbbá egy olyan felhasználóra, aki ezzel a szerepkörrel rendelkezik. Ennek oka, hogy ezzel a jogkörrel rendelkező felhasználónak az alkalmazás minden pontjához hozzá kell tudni férni, ugyanis adminisztrátorként bármilyen műveletet végre kell tudnia hajtani. Ez a szerepkör nem összekeverendő az ügyvezetővel, ugyanis az ügyvezető az nem adminisztrátor! Az adminisztrátor tud felhasználót, eszközt, szoftvert létrehozni és módosítani, tud eszközt és szoftvert bármilyen felhasználóhoz hozzárendelni, hozzáfér minden listázó felülethez és tudja azokat exportálni.

# **Ügyvezetői szerepkör**

Az ügyvezetői szerepkörrel rendelkező felhasználó úgymint az adminisztrátori szerepkörrel rendelkező felhasználó, az alkalmazás minden pontjához hozzá kell tudnia férni, továbbá ugyan úgy bármilyen műveletet végre kell tudnia hajtani, ugyanis egy vállalat ügyvezetőjének az alkalmazáshoz korlátlan hozzáféréssel kell, hogy rendelkezzen. Tehát az alábbi műveleteket tudja végrehajtani: tud felhasználót, eszközt, szoftvert létrehozni és módosítani, tud eszközt és szoftvert bármilyen felhasználóhoz hozzárendelni, hozzáfér minden listázó felülethez és tudja azokat exportálni.

# **HR vezetői szerepkör**

A HR vezetői szerepkörrel az a felhasználó rendelkezik, aki a rendszerben szereplő felhasználókat menedzseli. Ezzel a jogosultsággal a felhasználó az alábbi műveleteket tudja végrehajtani: Felhasználó hozzáadása és módosítása, továbbá a bejelentkezett felhasználóhoz (azaz saját magához) tartozó eszközök és szoftverek listájának megtekintése.

# **Gazdasági vezető szerepkör**

A gazdasági vezető szerepkörrel az a felhasználó rendelkezik, aki a rendszerben szereplő eszközöket és szoftvereket tartja karban. Ezzel a szerepkörrel a felhasználó az alábbi műveleteket tudja végrehajtani: Eszköz és szoftver hozzáadása és módosítása, tud eszközt és szoftvert bármilyen felhasználóhoz hozzárendelni, továbbá hozzáfér az eszközök és szoftverekhez kapcsolatos listázó felületekhez, amelyeket ki tudja exportálni.

# **Munkatárs szerepkör**

Az alkalmazásban található legegyszerűbb szerepkör a munkatárs szerepköre. Felhasználói jogosultsága nagyon kevés, csak nagyon minimális műveletet tud végrehajtani az alkalmazásban. Ezzel a jogosultsági szinttel a felhasználó az alábbi műveleteket tudja végrehajtani: A bejelentkezett felhasználóhoz (azaz saját magához) tartozó eszközök és szoftverek listájának megtekintése.

# **Adatbáziskezelés**

# **Az adatbáziskezelő rendszerek és az adatbázisok**

Az adatbázis-kezelő rendszerek azok olyan programok, amelyek lehetővé teszik az adatbázisok kezelését, továbbá kezelik az adatbázisokon végrehajtott műveletek elvégzését, mint például az adatbázisból történő adatok lekérdezését, azok módosítását vagy akár törlését.

Maga az adatbázis egy olyan halmaz, amelynek elemei egy vagy akár több meghatározott tulajdonságuk alapján összetartozónak tekinthetőek. „*Az adatbázis-kezelőknek meg kell oldani ezen adatok rendezését, a köztük lévő kapcsolat nyilvántartását, az adatokhoz való hozzáférés szabályozását, az adatok védelmét, az integritás megőrzését, az adatok módosíthatóságát, lekérdezését, különféle szempontok szerinti kigyűjtését, válogatását és egyéb statisztikai funkciókat is*” [[1]](#footnote-1).

# **Az adatbázis-kezelő rendszer meghatározása**

Mivel az alkalmazásnak képesnek kell lennie adatokat tárolni, így meg kellett azt határoznom, hogy az adatokat milyen formában/ sémában fogom eltárolni. Mivel olyan adathalmazról van szó, amelyben az adatok azok kapcsolatban állnak egymással, továbbá a tárolandó és kezelendő adathalmaz számossága az nagyon magas lehet, így az olyan tárolási formák, mint a fájl, JSON, XML stb. egyáltalán nem kerülhettek szóba, ugyanis a kapcsolatok azok vagy nagyon nehezen vagy pedig egyáltalán nem alakíthatóak ki, továbbá ezeknek az állományoknak a kezelése az vagy nagyon lassú lenne, vagy egyszerűen nem lehetne kivitelezhető.

Mint ahogyan korábban is említésre került, az ilyen típusú adathalmazok tárolására és kezelésére az adatbázis-kezelő rendszerek és az adatbázisok a legalkalmasabbak. Az adatbázis-kezelő rendszer meghatározására, a C# nyelvhez legtöbb támogatottságot nyújtó megoldást választottam, azaz a Microsoft által fejlesztett Microsoft SQL Server-t (MSSQL) abból is az ingyenesen használható Express Edition-t.

Mint ahogyan az előző bekezdésben is említettem, az SQL Server az nagy támogatottsággal rendelkezik a C# programozási nyelvhez, ugyanis mind a kettő Microsoft által fejlesztett „termék”. Saját adatkezelő nyelvvel rendelkezik, amely kompatibilis az SQL szabvánnyal (ez a Transact-SQL azaz a T-SQL), illetve egy könnyen kezelhető kliens eszközzel rendelkezik (SQL Server Management Studio), amelyből könnyen menedzselhetőek a céleszközön található adatbázisok, azonban itt meg kell, hogy jegyezzem, hogy ezeket az adatbázisokat nem csak az SSMS-ből lehet kezelni, hanem erre maga a Visual Studio 2019 Community Edition is lehetőséget biztosít.

# **EntityFramework Core ORM**

Az adatbázisok kezelésének több lehetséges módja van a .NET keretrendszerben, mint például a natív kapcsolat, logikai relációs modell, egyszerű objektumrelációs modell és az entitás alapú objektumrelációs modell.

Az ORM (Object Relational Mapping), az egy magas szintű transzformációját adja az adatbázisnak, amely a programkódban nagyon könnyen használható, továbbá szabályozza az adatok kezelésének módját. Itt fontos kiemelni, hogy az ORM használatához létrejött osztályok azok csak szigorúan adatot tárolnak, műveleteket egyáltalán nem végeznek!

A .NET keretrendszerben az egyik legismertebb és legelterjedtebb objektumrelációs adatkezelője az Entity Framework, azonban a .NET Core alapú alkalmazások esetében már az EF Core valósítja meg az adatok objektumrelációs leképzését.

Néhány szóban magáról az EF Core-ról:

* Alapja az entitás adatmodell (Entity Data Model, EDM).
* Egy entitás egy tábla sorának objektumorientált reprezentációja.
* Az entitások között kapcsolatok állíthatóak fel, építhetőek ki.
* Támogatja a nyelvbe ágyazott lekérdezéseket (LINQ – Lambda).
* Támogatja a dinamikus adatbetöltést továbbá az aszinkron adatkezelést.
* A modellek létrehozása az három megközelítési módból lehetséges:
  + Adatbázis alapján (Database First): Maga az adatbázis-szerkezetet képezzük le az entitás modellre, így maguk a modellek az adatbázis séma alapján fognak generálódni.
  + Tervezés alapján (Model First): Magát a modelleket azokat manuálisan építjük fel és állítjuk be a modellek közötti kapcsolatokat, így maguk a modellek alapján kerül kigenerálásra az adatbázis séma.
  + Kód alapján (Code First): A modellek és a modellek közötti kapcsolatok azok kódban kerülnek létrehozásra, így az adatbázis séma az a modellek alapján kerül kiépítésre.

Az alkalmazáshoz tartozó adatbázis szerkezetének kialakításához a Code First (Kód alapú) alapú séma kialakítást választottam. Ennek okai az alábbiak voltak:

* Ebben az esetben nincs semmilyen generált kód.
* Az adatbázisban található táblák egyszerűen megfeleltethetőek C# osztályok segítségével.
* A táblák közötti kapcsolatok egyszerűen leírhatóak és meghatározhatóak különböző modell konfigurációs beállításokon keresztül.
* Az adatbázis felépítését esetleges szerkezeti módosításait mind az ORM végzi különböző migrációs fájljain keresztül.
* Mivel az adatbázis szerkezeti módosításai migrációs fájlokon keresztül történik így könnyen nyomon követhető, hogy milyen változások mentek végbe az adatbázison, tehát a változások nem fognak elveszni az esetleges kézi módosításokkal ellentétben, ugyanis ott nincs 100%-os garancia arra, hogy minden módosítást végző SQL Script naplózva lesz.

Az EF Core használatához és az EF Core és SQL Server közötti kapcsolat kialakításához az alábbi NuGet csomagok állnak rendelkezésünkre:

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Tools
* Microsoft.EntityFrameworkCore.SqlServer

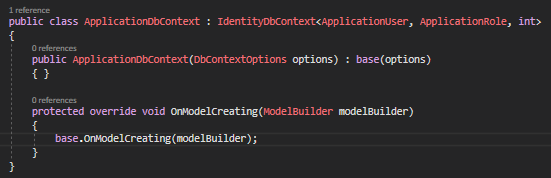
# **Felhasználók kezelése az adatbázisban**

Az adatbázisban történő felhasználók kezeléséhez nem alakítottam ki egyedi táblaszerkezetet és az azok közötti kapcsolatokat, hanem az ASP.NET által nyújtott ASP.NET Core Identity-t használtam fel erre a célra.

Maga az ASP.NET Core Identity egy olyan Microsoft által nyújtott API, amely támogatja a felhasználó bejelentkezési funkcióit. Kezeli magukat a felhasználókat, a hozzájuk tartozó jelszavakat, profil adatokat, jogosultsági szerepköröket, tokeneket és még nagyon sok mást. Az API használatához nem kell mást tennünk, minthogy telepítsük a Microsoft.AspNetCore.Identity NuGet csomagot.

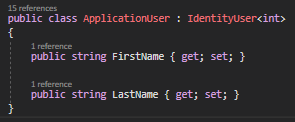
De hogyan is valósul meg az Identity API-n keresztül a felhasználók kezelése az adatbázisban? Nagyon egyszerűen! A Microsoft által nyújtott Entity Framework Core (EF Core) biztosítja, az ASP.NET Identity interfészeinek alapvető készletét az SQL Server számára. Ehhez nem szükséges mást tenni, minthogy a következő NuGet csomagot kell használni: Microsoft.AspNetCore.Identity.EntityFrameworkCore. Ha ez a csomag implementálásra került a megfelelő projekt vagy projektekbe, akkor a következő módosítást kell elvégezni az adatbázis contextet megvalósító osztályon:

* Az adatbázis contextet megvalósító osztálynak nem DbContext osztályból kell leszármaznia, hanem az IdentityDbContext osztályból.
* Az IdentityDbContext első típus paramétere maga a felhasználót (User) megvalósító osztálynak kell lennie. Megkötése, hogy a felhasználót megvalósító osztálynak az IdentityUser<TKey> osztályból kell leszármaznia.
* Az IdentityDbContext második típus paramétere maga a felhasználóhoz tartozó jogosultságokat (Role) megvalósító osztálynak kell lennie. Megkötése, hogy a jogosultságot megvalósító osztálynak az IdentityRole<TKey> osztályból kell leszármaznia.
* Az IdentityDbContext harmadik paramétere maga az elsődleges kulcs típusát fogja meghatározni, amelyet az IdentityUser<TKey> és az IdentityRole<TKey> ősosztályok TKey típusát fogja meghatározni.



2. ábra Példa az IdentityDbContext megvalósításához

Ha az adatbázist megvalósító DbContext az IdentityDbContext-ből származtatjuk és az adatbázist felépítjük például egy CodeFirst Migration-ön keresztül, akkor létrejönnek azok az adatbázis táblák, amelyek a felhasználói adatok kezeléséért szolgálnak. Itt fontos kiemelnem, hogy ezek a táblák kiegészíthetőek, több új property-vel vagy más táblák közötti kapcsolattal, ehhez nem kell mást tennünk, minthogy például az IdentityUser-t megvalósító osztályba új egyedi property-ket veszünk fel.



3. ábra Példa az ApplicationUser kiegészítésére

Az Microsoft.AspNetCore.Identity.EntityFrameworkCore felhasználó kezeléséhez biztosított táblákból az alábbiakat használtam fel: **AspNetUsers**, **AspNetRoles**, **AspNetUserRoles**.

# **Az adatbázisban használt táblák**

# **AspNetUsers tábla**

Az alábbi tábla tartalmazza a felhasználóhoz tartozó adatokat, mint a HASH-el ellátott jelszót, vezeték nevet, keresztnevet, email címet, tokeneket stb. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| FirstName | NVARCHAR(MAX) – ALLOW NULLS |
| LastName | NVARCHAR(MAX) – ALLOW NULLS |
| UserName | NVARCHAR(256) |
| NormalizedUserName | NVARCHAR(256) – ALLOW NULLS |
| Email | NVARCHAR(256) – ALLOW NULLS |
| NormalizedEmail | NVARCHAR(256) – ALLOW NULLS |
| EmailConfirmed | BIT |
| PasswordHash | NVARCHAR(MAX) – ALLOW NULLS |
| SecurityStamp | NVARCHAR(MAX) – ALLOW NULLS |
| ConcurrencyStamp | NVARCHAR(MAX) – ALLOW NULLS |
| PhoneNumber | NVARCHAR(MAX) – ALLOW NULLS |
| PhoneNumberConfirmed | BIT |
| TwoFactorEnabled | BIT |
| LockoutEnd | DATETIMEOFFSET(7) – ALLOW NULLS |
| LockoutEnabled | BIT |
| AccessFailedCount | INT |

# **AspNetRoles tábla**

Az alábbi tábla tartalmazza az alkalmazás rendszerben található előre rögzített jogosultsági szerepköröket. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| Name | NVARCHAR(256) – ALLOW NULLS |
| NormalizedName | NVARCHAR(256) – ALLOW NULLS |
| ConcurrencyStamp | NVARCHAR(MAX) – ALLOW NULLS |

# **AspNetUserRoles tábla**

Az alábbi tábla tartalmazza a felhasználók és a jogosultsági szerepkörök közötti kapcsolatokat, tehát ebben a táblában lesz eltárolva, hogy melyik felhasználó, melyik szerepkörbe tartozik. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| UserId | INT (PRIMARY KEY, FOREIGN KEY) |
| RoleId | INT (PRIMARY KEY, FOREIGN KEY) |

# **ApplicationLog tábla**

Az alábbi tábla tartalmazza a rendszer által elkészített naplóbejegyzéseket, például ide kerülnek be az INFO vagy WARNING típusú bejegyzések is, illetve az ERROR típusú, azaz az alkalmazás által kezelt és kezeletlen kivételek naplóbejegyzései is, amely nagyban segíti a fejlesztőket az esetleges hibák keresésében és feltárásában. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | BIGINT (PRIMARY KEY) |
| MachineName | NVARCHAR(50) – ALLOW NULLS |
| Logged | DATETIME2(7) |
| [Level] | NVARCHAR(50) – ALLOW NULLS |
| Message | NVARCHAR(MAX) – ALLOW NULLS |
| Logger | NVARCHAR(250) – ALLOW NULLS |
| Callsite | NVARCHAR(MAX) – ALLOW NULLS |
| Exception | NVARCHAR(MAX) – ALLOW NULLS |

# **Device tábla**

Az alábbi tábla tartalmazza az alkalmazás felhasználói által rögzített, majd a rendszerben megtalálható aktív és nem aktív (elérhető és leselejtezett) eszközöket. Ezek az eszközök lesznek hozzárendelve az egyes felhasználókhoz. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| SerialNumber | NVACRHAR(50) – UNIQUE CONSTRAINT |
| Name | NVARCHAR(250) |
| Description | NVARCHAR(250) |
| Price | DECIMAL(18, 2) |
| Link | NVARCHAR(250) – ALLOW NULLS |
| UserId | INT (FOREIGN KEY) – ALLOW NULLS |
| IsActive | BIT |

# **DeviceHistory tábla**

Az alábbi tábla tartalmazza az egyes eszközök - felhasználók közötti mozgását. A tábla tartalmából akár le lehet kérdezni, hogy melyik eszköz melyik felhasználóhoz került hozzá, mikor került vissza a készletre, vagy akár át más felhasználóhoz az az eszköz stb. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| UserId | INT (FOREIGN KEY) |
| DeviceId | INT (FOREIGN KEY) |
| StartDate | DATETIME(7) |
| EndDate | DATETIME(7) – ALLOW NULLS |

# **Software tábla**

Az alábbi tábla tartalmazza az alkalmazás felhasználói által rögzített, majd a rendszerben megtalálható aktív és nem aktív (elérhető és leselejtezett) szoftvereket. Ezek a szoftverek lesznek hozzárendelve az egyes felhasználókhoz. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| License | NVACRHAR(50) – UNIQUE CONSTRAINT |
| Name | NVARCHAR(250) |
| Description | NVARCHAR(250) |
| Price | DECIMAL(18, 2) |
| ProductLink | NVARCHAR(250) – ALLOW NULLS |
| UserId | INT (FOREIGN KEY) – ALLOW NULLS |
| IsActive | BIT |

# **SoftwareHistory tábla**

Az alábbi tábla tartalmazza az egyes szoftverek - felhasználók közötti mozgását. A tábla tartalmából akár le lehet kérdezni, hogy melyik szoftver melyik felhasználóhoz került hozzá, mikor került vissza a készletre az a szoftver stb. A tábla felépítése a következő:

|  |  |
| --- | --- |
| **Attribútum megnevezése** | **Attribútum típusa** |
| Id | INT (PRIMARY KEY) |
| UserId | INT (FOREIGN KEY) |
| SoftwareId | INT (FOREIGN KEY) |
| StartDate | DATETIME(7) |
| EndData | DATETIME(7) – ALLOW NULLS |

# **Alkalmazott programtervezési minták**

A programtervezési mintákat (Software Design Patterns) azoknak nevezzük, amik az alkalmazás fejlesztésben olyan problémákra adhatnak általános, illetve újra felhasználható megoldást vagy megvalósítási tervet, amelyek gyakran előforduló vagy ismétlődő problémák. Ezek a minták egyáltalán nem kötődnek programozási nyelvekhez, ezek attól sokkal általánosabb „megoldásvázak”, amelyeknek alkalmazhatósága könnyen eldönthető egy adott probléma esetén.

# **Command and Query Responsibility Segregation (CQRS) minta.**

A hagyományos architektúrák esetében mind az olvasási műveletekre, mind pedig az írási műveletekre ugyan az az adatmodell van használva, amely tökéletesen működik, ha egy alapszintű CRUD műveleteket végrehajtó alkalmazásról van szó, azonban, ha az alkalmazás már elkezd nőni, kezd összetettebb lenni, akkor ezzel a megközelítéssel egyre nehezebbé válik akár az üzleti logika implementálása is.

Az alábbi programtervezési minta, az teljes mértékben szétválasztja az adattáron történő olvasási, illetve frissítési (Read-Write) műveleteket, tehát két különböző adatmodellt használ. Az adatolvasás műveleteket **lekérdezéseknek** nevezzük, míg az adatfrissítő műveleteket pedig **parancsoknak**.

A CQRS minta az alábbiakat határozza meg:

* A parancsoknak (Command) feladatalapúnak kell lennie (például: „Hozzon létre egy eszközt”, „Frissítse a felhasználó adatait” stb.).
* A lekérdezések (Query) soha, semmilyen körülmény között nem módosíthatják az adatbázist! A lekérdezés eredményének egy olyan modellnek kell lennie, amely semmilyen módon nem reprezentálja az adatbázist.

A CQRS minta néhány előnye:

* Biztonságos, ugyanis meghatározhatóak, hogy mely entitások végezhetnek írási műveletet az adattáron.
* Lehetőség van függetlenül skálázni az olvasási és írási feladatokat.
* Az adatsémák egymástól függetlenül optimalizálhatóak, például amíg az olvasási oldalt azt a lekérdezésekre optimalizálhatjuk, addig az írási oldalt az adatok frissítési műveleteire optimalizálhatjuk.

# **Mediator programtervezési minta**

A közvetítő (Mediator) programtervezési minta lehetővé teszi azt, hogy a rendszer által létrehozott különböző objektumok egymással kommunikáljanak egy központilag használt objektumon keresztül. Amennyiben ezek az objektumok egymással közvetlenül kommunikálnának, azzal egy teljesen szoros viszonyt alakítanának ki egymással, amely teljesen rugalmatlanná, karbantarthatatlanná és nehezen bővíthetővé válna.

A központilag használt közvetítő megelőzi azt, hogy az egyes objektumok ne közvetlenül hivatkozzák egymást, tehát teljes mértékben arra fókuszál, hogy ezek az objektumok csakis rajta keresztül lépjenek interakcióba egymással, így biztosítva a laza kapcsolatot közöttük.

A Mediator programtervezési minta alkalmazásához akár a fejlesztő az saját implementációt is alkalmazhat, azonban .NET Core alatt lehetőségünk van a saját implementáció kidolgozása helyett a MediatR NuGet csomagot használva, egyszerűen implementálni ezt a programtervezési mintát. Az alkalmazásom elkészítése során én is ezt a csomagot használtam fel a Mediator programtervezési minta használatához.

# **Az alkalmazás architektúrális felépítése**

Az alkalmazás tervezésekor egy többrétegű architektúrát határoztam meg, melynek célja, hogy az alkalmazás különböző rétegekre való bontás segítéségével, egy könnyen karbantartható, fejleszthető rendszert hozzak létre. A többrétegű alkalmazás architektúra előnye, hogy szükség esetén elég akár egy-egy réteget javítani vagy módosítani, illetve bővíteni az helyett, hogy magát az egész alkalmazást módosítanánk.

A leggyakrabban használt többrétegű alkalmazás fejlesztésekor a három rétegű architektúrát szokták alkalmazni, amely alapján a megjelenítésért felelős réteget, az üzleti logika megvalósításáért felelős réteget és az adatkezelésért felelős réteget szokták külön választani.

Az általam elkészített rendszer két fő részre bontottam fel, kliensre és a szerverre (kliens-szerver architektúra), azonban magát mindkét réteget több-több alrétegre, alegységre bontottam fel. Ennek oka, hogy egy jól átlátható rendszert kapjunk, esetleges igények esetén könnyebb legyen bővíteni, módosítani, hozzáadni új egységeket vagy esetleg eltávolítani belőle.

Az alkalmazás szerver oldali felépítése:

* **Application.Web:** Ez a WEB API réteg biztosítja a kommunikációt a kliens oldali alkalmazással. Itt találhatóak meg a DI (Dependency Injection) konténerbe történő regisztrációk, a szerver oldali globális hibakezelés és a naplózás implementációja is. Ez a réteg tekinthető az alkalmazás magjának, ugyanis innen indul minden.
* **Application.DataAccessLayer:** Ez a réteg tartalmazza az adatbázis eléréshez szükséges adatbázis contexteket, minden függőségével együtt, mint például a migrációs fájlok, az adatbázishoz tartozó kiterjesztések stb), Továbbá itt kerülnek implementálásra a CQRS programtervezési mintának megfelelő Read és Write adatbázis contextek is. Ez a réteg szigorúan csak az adatbázis réteghez szükséges osztályokat, konfigurációkat stb. tartalmazhat!
* **Application.Core:** Ez a réteg olyan központi elemeket, osztályokat, modelleket, statikus adathalmazokat tartalmaz, amely nem függ egyik rétegtől sem, így az alkalmazás bármely pontja felhasználhat. Ez a réteg szigorúan nem függhet semelyik másik rétegtől, ennek mindig függetlennek kell lennie!
* **Application.BusinessLogicLayer:** Ez a réteg tartalmazza az üzleti logika megvalósítását. Itt találhatóak meg az üzleti logikát megvalósító lekérdezések és parancsok, amelyek a CQRS programtervezési mintának megfelelnek, továbbá a Mediator programtervezési minta megvalósításához szükséges absztrakciók. Ez a réteg csak szigorúan üzleti logikához tartozó osztályokat tartalmazhat!

Az alkalmazás kliens oldali felépítése:

* **Core module:** Ez a modul tartalmazza az alkalmazás magját. Olyan központi szolgáltatásokat tartalmaz, mint például a hitelesítési szolgáltatás, de olyan elfogókat (interceptorokat) is tartalmaz, amely a szerver oldaltól érkező hibák kezelésére szolgál stb. Itt kerülnek regisztrálásra a globális szolgáltatások a DI konténerbe, továbbá itt kerülnek definiálásra az alkalmazásban felhasznált különböző Guardok is.
* **Shared module:** Ez a modul tartalmazza az egyes modulokban, vagy globálisan használt megosztott elemek. Ide tartoznak a különböző Directive és Pipe elemek, de itt található meg maga az alkalmazásban megjelenítendő Header és Footer, Toaster stb., továbbá az alkalmazásban felhasználható általános komponensek is.
* **App module:** Ez a modul tekinthető az alkalmazás magjának. Itt találhatóak meg az alkalmazás útelválasztási (app-routing) beállításai, amely a megjelenítendő felületeket szolgálja ki, továbbá itt kerül definiálásra a Core és a Shared modulok is.
* **Konténerek és a hozzájuk tartozó komponensek:** Itt találhatóak meg azok a UI elemek (konténerek és a konténereket összekötő egyedi komponensek), amelyek az üzleti logika reprezentálásához szükségesek. Itt található meg minden olyan HTML, CSS/SCSS és TypeScript fájl, amely az aktuális oldal megjelenítéséhez szükséges.

# **Globális hibakezelés és a szerver oldali hibák naplózása**

Az alkalmazás egyik nem funkcionális követelménye, hogy minden olyan nem kezelt hibára/ kivételes esetre fel kell készülni, amely az alkalmazás futása során létrejöhet és ezekről naplóbejegyzést kell készíteni mind fájlba, mind pedig adatbázisba.

Az alkalmazások egyik legfontosabb kitétele, hogy a lehető legtöbb kivételes eset kezelve legyen ezáltal biztosítva azt, hogy akár egy kódolási hiba miatt, az alkalmazás ne omoljon össze, továbbra is használható legyen, ne kelljen ez miatt teljes újraindítást végrehajtani stb.

De hogyan is oldottam meg ezeknek a nem kezelt kivételeknek/ hibáknak a kezelését? Az ASP.NET Core biztosít számunkra egy egyszerű megoldást, azaz a Middleware-k használatát. A Middleware-ek biztosítják számunkra, hogy az egyes HTTP kérések végrehajtása előtt vagy a HTTP válaszadások előtt különböző műveleteket hajtsunk végre. Tehát én is létrehoztam egy olyan Middleware-t amely a globális hibakezeléséért felelős.

A Middleware segítségével, minden kivétel el lesz kapva és a kivétel típusának megfelelően lesz lekezelve. A kivételekből két különböző típust határoztam meg. Léteznek azok a kivételek, amelyek nem a fejlesztő által keletkeznek (például egy IOException), ezek lesznek az Internal Server Error-ok, illetve azok a kivételek, amelyeket ugyan úgy futási időben keletkeznek, azonban ez a fejlesztő által meghatározott kivételesesetek, amelyek nem fordulhatnának elő, tehát ezek lesznek a Bad Request-ek. A keletkezett hibákról az információk belekerülnek a HTTP válaszba, azonban az információknak a tartalma függ attól, hogy fejlesztői vagy más környezetben fut az alkalmazás, ugyanis a keletkezett hibát a kliens alkalmazás fel fogja dolgozni és akár maga a kivétel teljes leírása csak fejlesztői környezetben tud értékes információ lenni egy üzenetben.

Mielőtt a HTTP válasz el lenne küldve a kliens számára, előtte történik meg a kivételes eset naplózása. A naplózáshoz megvalósításához az NLog.Web.AspNetCore NuGet csomag által biztosított eszközöket használtam. Az NLog konfigurációs fájlján keresztül egyszerűen be tudtam konfigurálni, hogy a naplózás az mind fájlba, mind pedig adatbázisba is megtörténjen.

Kliens oldalon az Angular által biztosított HTTP\_INTERCEPTOR-t használtam fel. A HTTP\_INTERCEPTOR-ok segítségével lehetőségünk van műveleteket végrehajtani a HTTP kérések elküldése előtt, illetve ellenőrizni a kérésre adott választ, mielőtt bármilyen más soron következő művelet végrehajtásra kerülne.

Ezek ezáltal definiáltam egy olyan HTTP\_INTERCEPTOR-t, amely minden kérésre adott válasz előtt megvizsgálja, hogy a HTTP válasz az milyen HTTP kóddal tért vissza. Ha a státusz kód az 500 (Internal Server Error) vagy pedig 400 (Bad Request), akkor valamilyen hiba keletkezett a szerver oldalon. A hibáról a felhasználó tájékoztatást kap, egy felugró (toast) üzeneten keresztül, így értesülni fog arról, hogy az adott művelet, amelyet szeretett volna végrehajtani, az sikertelen volt.

Kiegészítésként szeretném azt megemlíteni, hogyha az alkalmazás az fejlesztői környezetben fut, akkor minden olyan kivétel, amelyet a kliens oldali elfogó elkap, az a böngészőkbe beépített Developer Tool-ban megtalálható konzolra kiírásra kerül, a teljes InnerException és a StackTrace szövege, így segítve, hogy a fejlesztőnek ne kelljen átnéznie a napló fájlokat, hanem csak elég a konzolt megtekintenie. Természetesen, mint ahogyan ezt korábban említettem, ezek az üzenetek más környezetben (legyen az egy Staging vagy Product környezet) nem fognak megjelenni, így biztosítva, hogy szenzitív adat ne kerüljön ki az alkalmazásról!

# **Az alkalmazásban található felületek**

# **Bejelentkezés**

# **404 Oldal**

# **Felhasználók listája**

# **Felhasználó regisztrálása**

# **Felhasználó szerkesztése**

# **Eszközök listája**

# **Eszköz létrehozása és szerkesztése**

# **Szoftverek listázása**

# **Szoftver létrehozása és szerkesztése**

# **Egy eszköz életútjának megtekintése (melyik felhasználóknál volt milyen időtartományban)**

# **Egy szoftver életútjának megtekintése (melyik felhasználóknál volt milyen időtartományban)**

# **Bejelentkezett felhasználóhoz tartozó eszközöket listázó felület**

# **Bejelentkezett felhasználóhoz tartozó szoftvereket listázó felület**

# **Esettanulmány**

# **Adminisztrátor szerepkör – Felhasználó regisztrálása**

# **HR Vezető szerepkör – Felhasználó módosítása**

# **Gazdasági vezető szerepkör – Egy eszköz áthelyezése más felhasználóra, azaz az eszközmozgás bemutatása.**

# **Munkatárs szerepkör – Saját szoftverek lista megtekintése**

# **Ügyvezető szerepkör – A rendszerben található szoftverek exportálása Excel és PDF formátumba.**

# **Utószó**

# **Bibliográfia**

1. https://angular.io (utolsó letöltés: 2021.03.10)
2. ELTE IK, Programozási Nyelvek és Fordítóprogramok Tanszék, *Frontend keretrendszerek*.  
   http://nyelvek.inf.elte.hu/leirasok/JavaScript/index.php?chapter=27, (utolsó letöltés: 2021.04.28.)
3. Dr. BILICKI Vilmos, *Angular + IONIC*, 2016. 10. 12.,  
   http://eta.bibl.u-szeged.hu/2380/8/9.%20Angular%202.pdf, (utolsó letöltés: 2021.04.02.)
4. ELTE IK, Programozási Nyelvek és Fordítóprogramok Tanszék, *A TypeScript programozási nyelv*.  
   http://nyelvek.inf.elte.hu/leirasok/JavaScript/index.php?chapter=27, (utolsó letöltés: 2021.04.28.)
5. FICSOR Lajos, KRIZSÁN Zoltán, MILEFF Péter, *Szoftverfejlesztés*.  
   https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0046\_szoftverfejlesztes/ch05s02.html, (utolsó letöltés: 2021.04.05.)
6. https://tudasbazis.sulinet.hu/hu/szakkepzes/informatika/adatbazis-kezeles/adatbazis-kezeles-alapfogalmai/alapfogalmak-az-adatbazis-kezelo-rendszer-adatbazis (utolsó letöltés: 2021.04.15.)
7. CSERÉP Máté, *Objektumrelációs adatkezelés (Entity Framework)*.  
   https://mcserep.web.elte.hu/data/education/2017-2018-2\_WAF/elte\_waf\_ea03.pdf, (utolsó letöltés: 2021.04.21.)
8. https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio (utolsó letöltés: 2021.04.20.)
9. https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity (utolsó letöltés: 2021.04.20.)
10. https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.entityframeworkcore.identitydbcontext-3?view=aspnetcore-5.0 (utolsó letöltés: 2021.04.20.)
11. BARTHA Tibor Máté, *Tervezési minták bemutatása*.  
    http://users.iit.uni-miskolc.hu/ficsor/Infterv2004Hallg/BarthaTibor/TervezesiMintak.pdf, (utolsó letöltés: 2021.02.20.)
12. https://docs.microsoft.com/hu-hu/azure/architecture/patterns/cqrs (utolsó letöltés: 2021.02.21.)
13. https://amp.hu.google-info.org/1240689/1/kozvetito-programtervezesi-minta.html (utolsó letöltés: 2021.02.22.)
14. https://pkief.medium.com/global-error-handling-in-angular-ea395ce174b1 (utolsó letöltés: 2021.03.22.)
15. https://jasonwatmore.com/post/2020/10/02/aspnet-core-31-global-error-handler-tutorial (utolsó letöltés: 2021.03.23.)
16. REITER István, *C# programozás lépésről lépésre*, Jedlik Oktatási Stúdió, 2012
17. FEKETE Zoltán, *Szakdolgozat*, Debrecen, 2007
18. SIPOS Marianna, *Programozás „élesben”*, Infó-Szakkönyv, 2004
19. GALI Gyula, *Szakdolgozat*, Debrecen, 2008
20. Bradley L. JONES, *C# mesteri szinten*, Kiskapu Kiadó, 2004.

1. <https://tudasbazis.sulinet.hu/hu/szakkepzes/informatika/adatbazis-kezeles/adatbazis-kezeles-alapfogalmai/alapfogalmak-az-adatbazis-kezelo-rendszer-adatbazis> lásd Az adatbázis adathalmaza, utolsó letöltés: 2021.05. [↑](#footnote-ref-1)