

Pannon Egyetem
Műszaki Informatikai Kar
Villamosmérnöki és Információs Rendszerek Tanszék
Programtervező informatikus BSc

SZAKDOLGOZAT

Kockázatkezelést támogató vagyonelem leltár alkalmazás
fejlesztése

Bercza Ferenc Dániel

Témavezető: Holló Krisztina

Külső/belső konzulens:

2025

Témakiírás

Hallgatói nyilatkozat

Alulírott Bercza Ferenc Dániel hallgató (Neptun kód: H0ZMX9) kijelentem, és a dolgozat feltöltésével egyidejűleg nyilatkozom, hogy a Kockázatkezelést támogató vagyonelem leltár alkalmazás fejlesztése című szakdolgozatot (a továbbiakban: dolgozat) a Pannon Egyetem Műszaki Informatikai Kar Villamosmérnöki és Információs Rendszerek Tanszékén készítettem a programtervező informatikus oklevél megszerzése érdekében.

Kijelentem, hogy a dolgozatban csak a megadott és hivatkozott forrásokat használtam fel, és ezekre a vonatkozó idézési szabályok szerint hivatkoztam.

Nyilatkozom, hogy a dolgozat érdemi része saját szellemi alkotásom eredménye, és azt más intézményben, szakon, vagy felsőfokú képzítés megszerzésére nem nyújtottam be. Tudomásul veszem, hogy a plágium vagy szerzői jogsértés esetén a dolgozatom elutasításra kerülhet, és ellenem fegyelmi eljárás indulhat. Tudomásul veszem továbbá, hogy szerzői jogsértés esetén az Egyetem jogosult a dolgozat elérhetőségét korlátozni, valamint eltávolítani a dokumentumot a dolgozatok tárolására szolgáló, a témát vezető szervezeti egység által meghatározott elektronikus zárt rendszerből.

Tudomásul veszem továbbá, hogy a Pannon Egyetem a dolgozat eredményeit saját céljaira eltérő írásbeli megállapodás hiányában a Pannon Egyetem Szellemi Tulajdon Kezelési Szabályzatában foglaltaknak megfelelően szabadon felhasználhatja.

Nyilatkozom, hogy a dolgozat elkészítése során mesterséges intelligencia eszközöket használtam/nem használtam.

Nyilatkozom, hogy a dolgozat elkészítése során az alábbi táblázatban feltüntetett mesterséges intelligencia eszközöket kizárólag a kutatási, illetve fejlesztési feladat támogatására használtam fel, az érdemi munka, elemzés és következtetések teljes mértékben saját szellemi alkotásomat képezik.

Alkalmazott technológia	Alkalmazás módja	Előállított tartalom	MI használat aránya
GPT-4o (OpenAI)	technológiák rövid ismertetése	1.2 fejezet	25%
GPT-4o (OpenAI)	versenytársak elemzése, összehasonlítása	2. fejezet	25%

Dátum: Veszprém, 2025. április 29.


Bercza Ferenc Dániel

Témavezetői nyilatkozat

Köszönetnyilvánítás

Külön köszönetet szeretnék mondani témavezetőmnek, Holló Krisztinának, hogy szakmai tudásával, hasznos tanácsaival és támogató hozzáállásával végig segítette a dolgozat elkészítését. Külön hálával tartozom azért is, hogy idejét és figyelmét rám fordította, valamint mindig számíthattam a visszajelzéseire és útmutatására a munka során.

Tartalmi összefoglaló

Szakdolgozatom témájaként egy vagyonelemeket leltározó alkalmazás fejlesztését választottam, mely támogatja a kockázatkezelést is. Témaválasztásom fő oka az volt, hogy a kockázatkezelés, és a különböző vállalati elvárások, melyeket egy ilyen szoftverrel szemben támaszthatunk, új témák voltak számomra, és ezáltal rengeteg hasznos ismerettel tudtam bővíteni a tudásomat a szakdolgozatom elkészítése során.

Az alkalmazás fejlesztéséhez a C# programozási nyelvet választottam, mivel megfelelőnek tartottam a .NET keretrendszer nyújtotta eszközöket egy ilyen alkalmazás elkészítéséhez, és kellően jártas is vagyok az adott programozási nyelvben.

Maga az alkalmazás egy asztali, WPF app, az autentikáció, és az adatbázis pedig a Microsoft Azure szolgáltatásaira épül. Mivel mind a programozási nyelv, mind az Azure a Microsoft termékei, így gördülékenyen működnek egymással, ezért ez egy kézenfekvő választás volt.

Amint az alkalmazást elindítjuk, egy bejelentkező felület fogad minket, majd a felugró böngészőablakban megadhatjuk az adatainkat. A felhasználókat mind az Azure felületén, mind az alkalmazáson belül kezelhetjük.

Az alkalmazás fő funkciói közé tartozik a vagyonelemek megjelenítése, új vagyonelem hozzáadása, azok szerkesztése, törlése. Emellett a vagyonelemeket hozzárendelhetjük a felhasználókhoz, és azokat kockázatkezelés szempontjából értékelni is tudjuk. Sikeres bejelentkezést követően a vagyonelemek listája fogad minket. Ez a fő nézetünk, itt tudunk hozzáadni, szerkeszteni, törölni vagyonelemeket, szűrni azok között, illetve váltani a különböző nézetek között.

Kulcsszavak: C#, vagyonelem, leltár, kockázatkezelés, döntéstámogatás

Abstract

For the topic of my thesis, I chose to develop an application for inventorying assets that also supports risk management. The main reason for choosing this topic was that risk management and the various corporate requirements that such a software must meet were new areas for me, which allowed me to expand my knowledge significantly during the process of writing my thesis.

I selected C# as the programming language for developing the application, as I considered the tools provided by the .NET framework to be well-suited for creating this type of application, and I am sufficiently skilled in this programming language.

The application itself is a desktop WPF app, with authentication and the database built on Microsoft Azure services. Since both the programming language and Azure are Microsoft products, they work seamlessly together, making this a natural choice.

Upon launching the application, the user is greeted with a login screen, and in the popup browser window, they can enter their credentials. User accounts can be managed via the Azure interface and in the app as well.

The main functions of the application include displaying assets, adding new assets, editing, and deleting them. Additionally, assets can be assigned to users, and can be evaluated from a risk management perspective. After successful login, the user is presented with the asset list. This is the primary view, where assets can be added, edited, deleted, filtered, and where the user can switch between different views.

Keywords: C#, asset, inventory, risk management, decision support

Tartalomjegyzék

1. Bevezetés	1
1.1 ISO 27001:2013	2
1.2 Felhasznált technológiák	3
1.2.1 C#	3
1.2.2 Visual Studio 2022	4
1.2.3 .NET 8.0	5
1.2.4 WPF (Windows Presentation Foundation)	6
1.2.5 Model–view–viewmodel	7
1.2.6 Entity Framework	8
1.2.7 Dependency injection	8
1.2.8 GitHub	9
1.2.9 Microsoft Azure	9
1.2.10 OxyPlot	11
2. Versenytársak elemzése	12
2.1 Asset Panda	12
2.2 ManageEngine AssetExplorer	13
2.3 Pulseway	14
2.4 GoCodes	15
2.5 Ivanti	16
2.6 Elemzés összefoglalása	17
3. Funkciók ismertetése	18
3.1 Bejelentkezés	18
3.2 Vagyonelemek listája	18
3.3 Hozzárendelések listája	21
3.4 Naplóbejegyzések listája	22

3.5 Felhasználók kezelése	23
3.6 Alkategóriák listája	24
3.7 Kockázatkezelési nézet	25
4. Megvalósítás	28
4.1 Adatbázis.....	28
4.2 Függőségek injektálása	30
4.3 Navigáció	30
4.3.1 INavigationService	31
4.3.2 NavigationService	31
4.3.3 NavigationStore	31
4.3.4 MainWindow	32
4.4 Autentikáció, felhasználók kezelése	33
4.4.1 AuthenticationService.....	33
4.4.2 ManageUsersViewModel	37
4.4.3 LoginViewModel.....	38
4.5 Vagyonelemek, hozzárendelések, naplóbejegyzések, alkatégóriák listája, kockázatkezelés.....	39
4.6 Adatok hozzáadása, szerkesztése	40
5. Összegzés.....	43
Irodalomjegyzék.....	44

1. Bevezetés

Célom egy olyan alkalmazás tervezése és fejlesztése volt, amely lehetővé teszi a felhasználók számára, hogy kategorizálják a vagyonelemeket előre meghatározott kategóriák szerint, azokat értékelni tudják kockázatkezelés szempontjából, és könnyen, átlátható módon kezelni tudják azokat, ezáltal elősegítve az információbiztonsági elvek betartását. A mai világban egy ilyen alkalmazás meglehetősen fontos egy cég számára, mivel az internet széleskörű elterjedtsége hatalmas támadásai felületet jelent. Ezért nagyon fontos, hogy egy adott vállalat betartsa az információbiztonsági alapelveket, ezzel csökkentve az esetleges támadások kockázatát, valamint egy esetleges elszenvedett támadás esetén az okozott károk mértékét is. Ehhez a vagyonelemek leltározása nagyban hozzájárul, mivel a bizonyos eszközök, erőforrások nyilvántartása biztosítja, hogy ne veszítsük szem elől azokat.

Vállalati környezetben egy vagyonelem jelenthet egy adott számítógépet, laptopot, de akár egy szoftvert, valamilyen információt vagy emberi erőforrást is. A témához szorosan kapcsolódik az ISO 27001-es szabvány, mely az információbiztonság kezelését határozza meg. A szabvány elsősorban információs elemekkel foglalkozik, de előírja, hogy készítsünk nyilvántartást a különböző vagyonelemekről is, melyek hozzájárulnak az információbiztonsághoz, hogy ezeket kockázatkezelés szempontjából értékelni tudjuk. [8] Ezáltal a következő kategóriákat határoztam meg: hardver, szoftver, információ, infrastruktúra, emberi erőforrás. Az alkalmazásban ilyen típusú vagyonelemeket tudunk nyilvántartani.

A nyilvántartás megvalósításához meg kellett határoznom a különböző elemek közös tulajdonságait. Minden vagyonelemnek rendelkeznie kell névvel, típussal, tulajdonossal, beszerzési dátummal, valamint egy pénzbeli értékkel is. Ezek közül a pénzbeli érték meghatározása nem mindig egyértelmű, hiszen például egy információnak vagy egy emberi erőforrásnak nincsen egyértelmű pénzbeli értéke. Ettől függetlenül ilyen esetekben is fontos, hogy meghatározzuk az adott vagyonelem értékét, mivel ez is szerepet játszik a kockázatelemzés folyamatában. Ha egy adott esetben ez mégsem meghatározható, akkor pedig megadhatjuk az értéket 0-nak.

Az adott elemeknek ezen tulajdonságok mellett megadhatunk alkategóriát, elhelyezkedést és leírást is. Ezek nem kötelező mezők, viszont kitöltésükkel plusz információkat tárolhatunk el. Továbbá a saját alkategóriák létrehozása növeli a szoftver testreszabhatóságát, így szélesebb körben válik alkalmazhatóvá.

A vagyonelemek rendelkeznek egy státusszal is, mely lehet aktív vagy nyugalmazott. Információbiztonsági szempontból fontos, hogy sose töröljünk egy vagyonelemet sem, így, ha a törlés opcióját kiválasztjuk, az csupán az állapotot állítja át nyugalmazottra. Ezáltal bármikor visszakereshetjük a már használaton kívüli vagyonelemek adatait, ha ezekre bármilyen okból szükségünk lenne.

A vagyonelemek kockázatkezelés szempontjából történő értékelése több lépésben történik. Először szükséges meghatározni, hogy az adott vagyonelemmel szemben milyen sérülékenységeket és fenyegetettségeket tudunk megállapítani. Ezután meg kell határoznunk az adott esetre nézve, hogy a bekövetkezés mennyire valószínű, és milyen súlyos hatással lenne a szervezetre. Ez a két tulajdonság egy 1 és 5 közötti skálán értékelendő, és a két érték szorzata alapján tudjuk értékelni az adott sérülékenység vagy fenyegetettség súlyosságát az adott vagyonelemre nézve. [36]

5x5 Risk Matrix Example

Impact
How severe would the outcomes be if the risk occurred?

→

	Insignificant 1	Minor 2	Significant 3	Major 4	Severe 5
5 Almost Certain	Medium 5	High 10	Very high 15	Extreme 20	Extreme 25
4 Likely	Medium 4	Medium 8	High 12	Very high 16	Extreme 20
3 Moderate	Low 3	Medium 6	Medium 9	High 12	Very high 15
2 Unlikely	Very low 2	Low 4	Medium 6	Medium 8	High 10
1 Rare	Very low 1	Very low 2	Low 3	Medium 4	Medium 5

← **Probability**
What is the probability the risk will happen?

SafetyCulture

1. ábra: 5*5-ös kockázati mátrix [35]

1.1 ISO 27001:2013

Az ISO 27001 szabvány 2013-as kiadásának A melléklete leírja, hogy az információkhoz kapcsolódó vagyonelemekről leltárt kell tartanunk. Ezeknek a vagyonelemeknek meg kell határoznunk a tulajdonosát (vagyonelemgazda), ami ebben a kontextusban az adott vagyonelemért felelős személyt jelenti. A vagyonelemeket felhasználókhöz rendelhetjük, és ezesetben fontos, hogy ne keverjük össze a tulajdonost a hozzárendelt felhasználóval.

Például egy céges laptop esetében a tulajdonos, azaz felelős személy lehet az adott részleg vezetője. Ha ezt a laptopot megkapja egy alkalmazott, azt a leltárban hozzárendeljük. Ekkor az alkalmazott tulajdonában van a laptop, de a felelős személy ettől függetlenül a részlegvezető marad.

Fontos továbbá, hogy a vagyonelemeket az alkalmazott visszaszolgáltassa a szerződésében, vagy közös megegyezés alapján meghatározott dátumig. Ezért, ha egy vagyonelemet hozzárendelünk egy felhasználóhoz, az alkalmazásomban szükséges megadni a dátumot, ameddig az adott vagyonelemet vissza kell szolgáltatni.

Kockázatkezelés szempontjából fontos, hogy a vagyonelemek megfelelően legyenek kategorizálva, könnyen megállapítható legyen, hogy ki férhet hozzájuk, és a nyilvántartásuk naprakész legyen. Ez elősegíti a kockázatok pontos felmérését, és lehetővé teszi, hogy megfelelő kockázatkezelési terveket készítsünk a vagyonelemek védelmének érdekében. [8]

1.2 Felhasznált technológiák

Az alkalmazás fejlesztéséhez a C# programozási nyelvet használtam, és a Visual Studio 2022 fejlesztői környezetben dolgoztam. Az alkalmazás egy WPF app, és az MVVM architektúrális mintát alkalmaztam. Az autentikációhoz Microsoft Entra ID-t, az adatbázishoz pedig egy Azure SQL adatbázist választottam.

1.2.1 C#

A programozási nyelv kiválasztásánál számos szempontot figyelembe kellett vennem. Elsősorban az volt a legfontosabb, hogy mely programozási nyelvekben van elegendő tapasztalatom egy ilyen, nagyobb hangvételű projekt megvalósításához. Ezek közül pedig ki kellett választanom azt, amely leginkább illeszkedik a problémára, mind elérhető eszközök, funkciók, fejlesztői környezetek, keretrendszerek stb. tekintetében. Ezáltal esett a választás a C# programozási nyelvre. Rendelkezem megfelelő tapasztalattal ebben a nyelvben, és eszköztára lehetővé teszi, hogy hatékonyan megvalósítsam benne a feladatot.

A C# programnyelv a .NET keretrendszer részeként került fejlesztésre. Nagy mértékben eltér a C és C++ nyelvektől, azoknak egyfajta továbbfejlesztése, de korlátozásokat is tartalmaz. A mutatók csak az „unsafe” mód engedélyezésével használhatóak, míg az objektumok felszabadítása nem történik manuálisan, hanem ezt automatikusan a szemétgyűjtő (garbage collector) végzi. Sokkal típusbiztosabb nyelv, mint a C++.

A nyelv az objektumorientált programozási paradigmákat követi, amely azt jelenti, hogy osztályokat hozhatunk létre, melyek egységbe zárják az adatokat, és a rajtuk műveleteket végrehajtó metódusokat. Ezekből az osztályokból hozhatunk létre példányokat. Az objektumorientált programozás erősen ösztönzi az újra felhasználható kódok írását (code reuse), az öröklődés és a polimorfizmus által. Ez csökkenti a megírandó kód mennyiségét. Ezentúl nagyban megnöveli a kód fenntarthatóságát, mivel a kód nagy mértékben moduláris lesz. Az egységbe záras által minden osztálynak megvan a saját feladata, így, ha módosítani kell a kódot, azt csak egy helyen kell véghez vinni, így elkerülve a rendezetlen kódból következő hibákat. Emellett kényelmi funkciókkal is rendelkezik, mint például a tulajdonságok (property), melyek lehetővé teszik, hogy az osztályok mezőit az adattagok szintaxisát használva kezeljük.

Hátrányai, közé tartozik, hogy sokszor hiába segíti elő az újra felhasználható kódrészletek írását, emellett sok ismétlődő kódot (boilerplate) is kell írunk. A .NET keretrendszerből adódóan pedig kevésbé platformfüggetlen, mint más nyelvek, viszont manapság erre is több megoldás elérhető. Emellett a teljesítménye sem a legjobb, mivel a Common Language Runtime (CLR) virtuális gép miatt több memóriát, erőforrásokat használ, mintha natív kódot futtatnánk. [10, 11]

1.2.2 Visual Studio 2022

A felhasznált fejlesztőkörnyezet a Visual Studio 2022. Ez egy eléggé kézenfekvő választás volt, mivel támogatja a C# nyelvű, WPF appok fejlesztését a .NET keretrendszerben.

Ez az integrált fejlesztőkörnyezet (IDE) számos hasznos funkcióval rendelkezik, mint például fejlett debugging eszközök, melyekkel hatékonyan diagnosztizálhatjuk az esetleges hibákat. A kód futtatását szakaszokra bontva végezhetjük el, ha töréspontokat helyezünk el benne, ezáltal kisebb részleteket analizálva, így pontosan meghatározhatjuk a hibák forrását. Különböző változók értékeit is figyelhetjük futás közben, mely szintén megkönnyíti a hibák forrásának meghatározását. [13]

Emellett rendelkezik IntelliSense intelligens kódkiegészítő funkcióval is, mely valós idejű kódelemzést és kódkiegészítést tesz lehetővé, ezzel nagy mértékben megkönnyítve a fejlesztést és gyorsítva a hibamentes kód írását. A kód refactoring funkciók pedig lehetővé teszik a kód könnyű átszervezését, javítják a karbantarthatóságot. A Visual Studio GitHub integrációval is rendelkezik, mely megkönnyíti a verziókövetést és a csapatmunkát. [14]

Egy másik fontos funkciója NuGet Package Manager, mely egy eszköz a .NET fejlesztők számára, amely lehetővé teszi külső könyvtárak és csomagok egyszerű integrálását alkalmazásokba. Segítségével gyorsan hozzáadhatunk, frissíthetünk vagy eltávolíthatunk csomagokat, miközben a függőségek kezelését is automatikusan végzi. A NuGet támogatja a verziókezelést, így biztosítja, hogy a projektek mindig a megfelelő könyvtárverziókkal működjenek. Az alkalmazásomban például használom a Microsoft.EntityFrameworkCore bővítményt az adatbáziskezeléshez, a Microsoft.DependencyInjection bővítményt a függőségi injektálás kezeléséhez, illetve a Microsoft.Identity.Client bővítményt az Entra ID-val történő autentikációhoz. [15]

Továbbá egy különösen hasznos funkciója a Visual Studio Live Share, mellyel valós időben tud több fejlesztő együttműködni a folyamat során. Lehetővé teszi, hogy megosszuk a fejlesztőkörnyezetünket, mely által egyszerre többen szerkeszthetjük akár ugyan azt a fájlt is, illetve segítsük egymást a hibák keresésében. Ez jelentősen növeli a produktivitást, hiszen azonnal meg tudjuk osztani a kódunkat anélkül, hogy GitHub repository-kat kellene klónoznunk, vagy fájlokat megosztanunk egymással. Esetemben természetesen ez a funkció nem játszott szerepet, ettől függetlenül viszont egy meglehetősen hasznos funkciója a fejlesztőkörnyezetnek. [16]

1.2.3 .NET 8.0

A .NET keretrendszer egy Microsoft által kifejlesztett, többplatformos szoftverfejlesztési környezet, amely lehetővé teszi gyors alkalmazásfejlesztést és platformfüggetlen futtatást. Eredetileg a COM platform helyettesítésére készült, azonban mára az eszköztára a szoftverfejlesztés széles spektrumát lefedi, beleértve a kliens- és szerveroldali alkalmazásokat, adatbáziskezelést és játékfejlesztést is.

A .NET alapja a Common Language Infrastructure (CLI), egy szabályrendszer, amely lehetővé teszi a különböző nyelvek közötti együttműködést és biztosítja a futtatókörnyezetek egységességét. A CLI működését a Common Language Runtime (CLR) valósítja meg, amely a kód végrehajtását irányítja.

A CLI 4 részre bontható:

- **Common Language Specification:**
A CLS azokat a szabályokat tartalmazza, amelyeket a .NET környezetben használt nyelveknek be kell tartaniuk ahhoz, hogy egymással kompatibilisek legyenek. Például egyes típusok, mint az `ulong`, nem felelnek meg a CLS előírásainak, így nem használhatóak minden .NET nyelvben.
- **Common Type System:**
A CTS meghatározza azokat az adattípusokat, amelyek a .NET alkalmazásokban használhatóak. Továbbá előírja, hogyan kell ezeket a típusokat a memóriában tárolni, és hogyan kommunikálhatnak egymással a különböző típusú objektumok.
- **Common Language Runtime:**
A CLR a .NET alkalmazások futtatási környezetét biztosítja. Feladatai közé tartozik a programok betöltése és végrehajtása, a memória kezelése, a kivételkezelés, valamint a kód biztonságának ellenőrzése. A CLR virtuális gép formájában működik, és biztosítja a platformfüggetlen futtatást.
- **Common Intermediate Language:**
A CIL a .NET programok köztes kódja, amelyet minden .NET nyelv lefordít a saját forráskódjából. A futás során a CIL-t a just-in-time (JIT) fordító natív kódra alakítja, amelyet a processzor közvetlenül végrehajthat. Ez a megoldás hasonlít a Java bytekódjára, amelyet a Java virtuális gép (JVM) futtat. [12]

1.2.4 WPF (Windows Presentation Foundation)

A WPF egy grafikus felhasználói felületek készítésére alkalmas osztálykönyvtár, melyet szintén a Microsoft fejlesztett. Léteznek más alternatívák, mint például az Avalon, vagy a MAUI, viszont ezekben nem rendelkezem semennyi tapasztalattal sem, a WPF appok fejlesztésében pedig igen, így ezt választottam.

A WPF egyik legfontosabb újítása, hogy lehetővé teszi a felhasználói felület és az üzleti logika elkülönítését. A WPF alkalmazások felhasználói felületét az XAML segítségével definiáljuk, ami egy XML alapú jelölőnyelv. Ez az elválasztás lehetőséget ad arra, hogy a fejlesztők és a UI tervezők zökkenőmentesen dolgozhassanak együtt. [17, 18, 19]

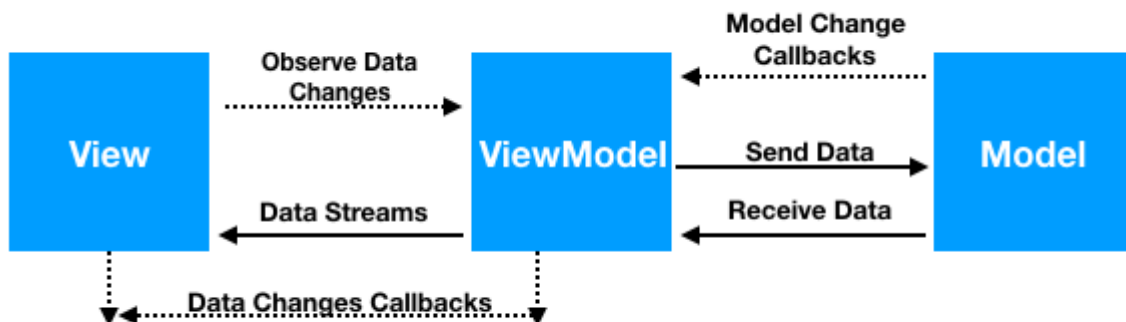
1.2.5 Model–view–viewmodel

Az MVVM (Model-View-ViewModel) egy szoftverfejlesztési minta, amely segít elválasztani a felhasználói felületet az üzleti logikától, így biztosítva a jobb karbantarthatóságot és tesztelhetőséget.

A modell az alkalmazás adatainak és állapotának kezeléséért felel, míg a nézet (view) a felhasználó számára látható grafikus felületet jelenti. A nézetmodell (viewmodel) a nézet absztrakciója, publikus tulajdonságokat és metódusokat tartalmaz. Ellentétben az MVC mintával, ahol a vezérlő irányítja az adatkezelést, az MVVM-ben a nézetmodell végzi el az összekötő szerepet, biztosítva az adat és a felület közötti kapcsolatot. Emellett az XAML nyelv is kulcsszerepet játszik a nézet és a modell összekapcsolásában.

A tesztelhetőség javulása is az MVVM egyik előnye, mivel az üzleti logika elkülönül a felhasználói felülettől, így egyszerűbbé válik az egységtesztelés.

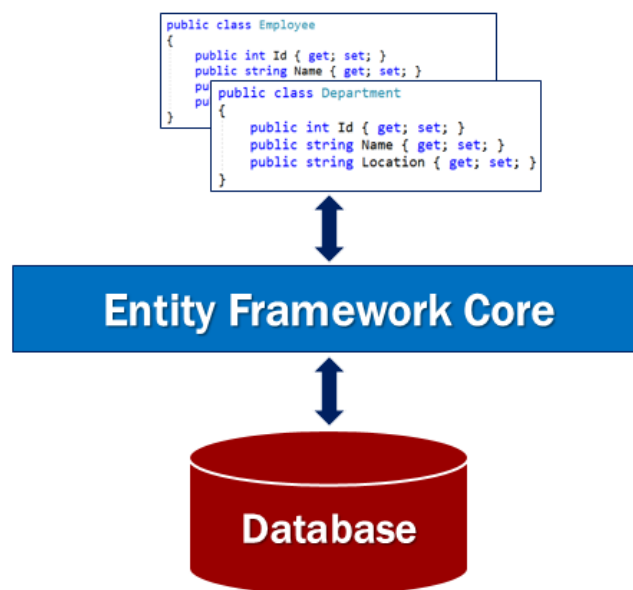
Főként azért esett a választásom erre az architektúrális mintára, mivel úgy tervezték, hogy jól működjön WPF alkalmazásokkal, és segíti a fejlesztést, mind átláthatóság, mind sebesség szempontjából. [20]



2. ábra: Az MVVM felépítése [1]

1.2.6 Entity Framework

Az Entity Framework (EF) egy objektum-relációs leképező (ORM) eszköz a .NET alkalmazásokhoz, amely lehetővé teszi az adatbázisokkal való egyszerű interakciót objektum-orientált módon. Az EF automatikusan generálja az SQL lekérdezéseket a C# objektumokhoz, így a fejlesztőknek nem kell manuálisan megírniuk az adatbázis műveleteket. Támogatja a különböző adatbázis-menedzsment rendszerekkel való integrációt, és biztosítja a tranzakciók kezelését, az adatok validálását és a migrációk egyszerű kezelését, ami megkönnyíti az alkalmazások fejlesztését és karbantartását. [22, 23, 24]



3. ábra: EF – adatbázis kommunikáció [2]

1.2.7 Dependency injection

A Dependency Injection (DI) egy olyan tervezési minta, amely lehetővé teszi, hogy az osztályok ne maguk kezeljék a szükséges függőségeket, hanem külső forrás biztosítja azokat számukra. Ahelyett, hogy az osztályok maguk hozzák létre a szükséges objektumokat, a DI külső forrásból injektálja őket. Ez a megközelítés segít abban, hogy az osztályok jobban összpontosítsanak a saját feladataikra, nem pedig a függőségek kezelésére.

A DI egyik legfontosabb előnye, hogy javítja a kód karbantarthatóságát és újrahasználatosságát. Mivel az osztályok nem tartalmaznak közvetlen függőségeket, könnyebben módosíthatók és tesztelhetők, anélkül, hogy más részekre hatással lennének. A DI csökkenti az osztályok közötti szoros kapcsolatokat, így a komponensek függetlenebbé válnak egymástól. Ennek következményeként a rendszer könnyebben bővíthető és alkalmazkodóképesebb lesz. Ez különösen előnyös nagy, összetett rendszerekben, ahol a változtatások gyorsan integrálhatók anélkül, hogy a teljes struktúrát újra kellene tervezni. [21]

1.2.8 GitHub

A Git egy nyílt forráskódú elosztott verziókezelő szoftver, amely lehetővé teszi, hogy többen dolgozzanak ugyan azon a projekten, gördülékenyen.

A Git lehetőséget biztosít arra, hogy minden módosítást naprakészen rögzítsünk, és a különböző fejlesztők munkáját egyesíthessük anélkül, hogy a kód sérülne. A GitHub ezt a szolgáltatást kínálja a felhőben.

Fő funkciói közé tartozik a verziók nyomon követése, így, ha valami félresikerül, vagy bármilyen oknál fogva szeretnénk visszatérni a szoftver egy korábbi verziójára, ezt könnyedén megtehetjük. A Git lehetővé teszi a különböző fejlesztési irányok elkülönítését ágak segítségével, így különböző funkciók, hibajavítások párhuzamos fejlesztésére van lehetőség, amelyeket a végén könnyen egyesíthetünk.

Ezen funkciók által minimalizálni tudjuk a hibák keletkezésének vagy az adatvesztésnek az esélyét. Emellett hatékonyan dolgozhatunk bármekkora méretű projekten, legyen az egyszemélyes vagy nagy fejlesztőcsapatok munkája, minden esetben nagyban növeli a produktivitást. [25]

1.2.9 Microsoft Azure

A Microsoft Azure egy felhőalapú szolgáltatásplatform, amely számos eszközt és szolgáltatást kínál a vállalatok számára, hogy alkalmazásaikat, adatbázisaikat és infrastruktúrájukat biztonságosan és rugalmasan futtassák a felhőben. Az Azure támogatja a skálázható webalkalmazások, virtuális gépek és konténerek futtatását, emellett különböző adatkezelési, analitikai és mesterséges intelligencia szolgáltatásokat is biztosít.

A platform lehetővé teszi, hogy a fejlesztők gyorsan építhessenek, teszteljenek és üzemeltessenek alkalmazásokat, miközben rugalmasan alkalmazkodnak az üzleti igényekhez.

Az Azure integrációja segít a cégeknek a költségek optimalizálásában, miközben biztosítja a magas rendelkezésre állást és a biztonságot. [26]

1.2.9.1 Microsoft Entra ID

Az Entra ID (korábban Azure Active Directory) egy felhőalapú identitás- és hozzáférés-kezelési szolgáltatás, amely lehetővé teszi a vállalatok számára a felhasználók és eszközök biztonságos hitelesítését és kezelését.

Az Entra ID támogatja az egységes bejelentkezést (SSO), többfaktoros hitelesítést (MFA) és az alkalmazásokhoz való hozzáférés finomhangolt szabályozását, így biztosítva a vállalati környezetek védelmét. Ezen kívül integrálódik különböző Microsoft és harmadik féltől származó alkalmazásokkal, lehetővé téve a könnyű felhasználói és eszközkezelést. Az Entra ID segítségével a cégek gyorsan reagálhatnak a biztonsági fenyegetésekre, miközben egyszerűsítik a felhasználói élményt és csökkentik az adminisztrációs terheket.

Az Entra ID ezen tulajdonságai lehetővé tették, hogy könnyen implementáljam az autentikációt az alkalmazásomban, és kihasználjam az RBAC (Role-Based Access Control) funkciót a felhasználói jogosultságok kezelésére. Az RBAC segítségével finoman szabályozhatom, hogy ki férhet hozzá az alkalmazás különböző részeihez, és milyen műveleteket végezhet el. A felhasználókat szerepkörök (pl. adminisztrátor, fejlesztő, olvasó) alapján csoportosíthatom, így biztosítva, hogy mindenki csak az erőforrásokat és adatokat érhesse el, amelyek szükségesek a feladatukhoz. [27]

1.2.9.2 Azure SQL adatbázis

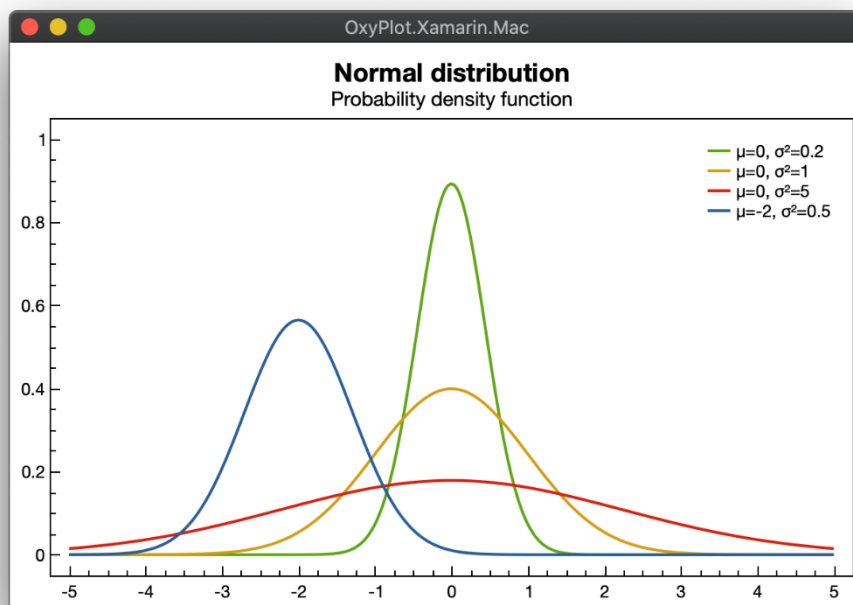
Az Azure SQL Database egy felhőalapú, teljes mértékben menedzselt adatbázis-szolgáltatás, amely lehetővé teszi az alkalmazások számára, hogy skálázható és biztonságos adatbázisokat használjanak anélkül, hogy saját infrastruktúrát kellene fenntartaniuk. Az Azure SQL DB automatikusan kezeli az adatbázisok karbantartását, biztonsági mentéseit és frissítéseit, így a fejlesztők a legfontosabb feladatokra, például a teljesítmény optimalizálására és a kód fejlesztésére összpontosíthatnak. A szolgáltatás támogatja a dinamikus skálázást, a magas rendelkezésre állást és a több régióban történő replikációt.

Az Azure SQL DB rugalmas árazási lehetőségeket kínál, így könnyen illeszkedik különböző üzleti igényekhez, mind a kis, mind a nagyvállalatok számára.

Esetemben teljes mértékben megfelelt az ingyenesen elérhető verziója a szolgáltatásnak, és az Entra ID, illetve az Azure SQL DB együttes használata nagyban megkönnyítette a fejlesztést, mivel a teljes mértékben a Microsoft ökoszisztémáján belül elérhető szolgáltatások és nyelv használata biztosította a zökkenőmentes integrációt. [28]

1.2.10 OxyPlot

Az OxyPlot egy népszerű nyílt forráskódú grafikonkönyvtár C# és .NET alkalmazásokhoz, amely lehetővé teszi a diagramok és grafikonok létrehozását minimális erőfeszítéssel. Számos diagramtípust támogat, beleértve a vonalas, pont-, oszlop-, kör- és hőtérképeket, így alkalmas tudományos, mérnöki és üzleti alkalmazásokhoz. Az OxyPlot könnyen integrálható WPF, Windows Forms, Xamarin, Avalonia és más .NET-alapú keretrendszerekbe. Lehetővé teszi, hogy néhány sor kóddal testreszabott grafikonokat hozzunk létre. Rugalmassága és bővíthetősége révén hatékony eszközt jelent az összetett adathalmazok vizualizálására C# alkalmazásokban. A szakdolgozatomban különböző kimutatások generálásához alkalmazom ezt a könyvtárat. [34]



4. ábra: OxyPlot példa diagram [37]

2. Versenyársak elemzése

A piacon jelenleg számos hasonló alkalmazás érhető el. Ezen alkalmazások előnyeit, illetve hátrányait kielemezve rengeteg hasznos következtetést vonhatok le, melyeket alkalmazhatok a saját alkalmazásom elkészítése során. A továbbiakban a jelenleg elérhető legnépszerűbb alkalmazásokról fogok írni bővebben.

2.1 Asset Panda

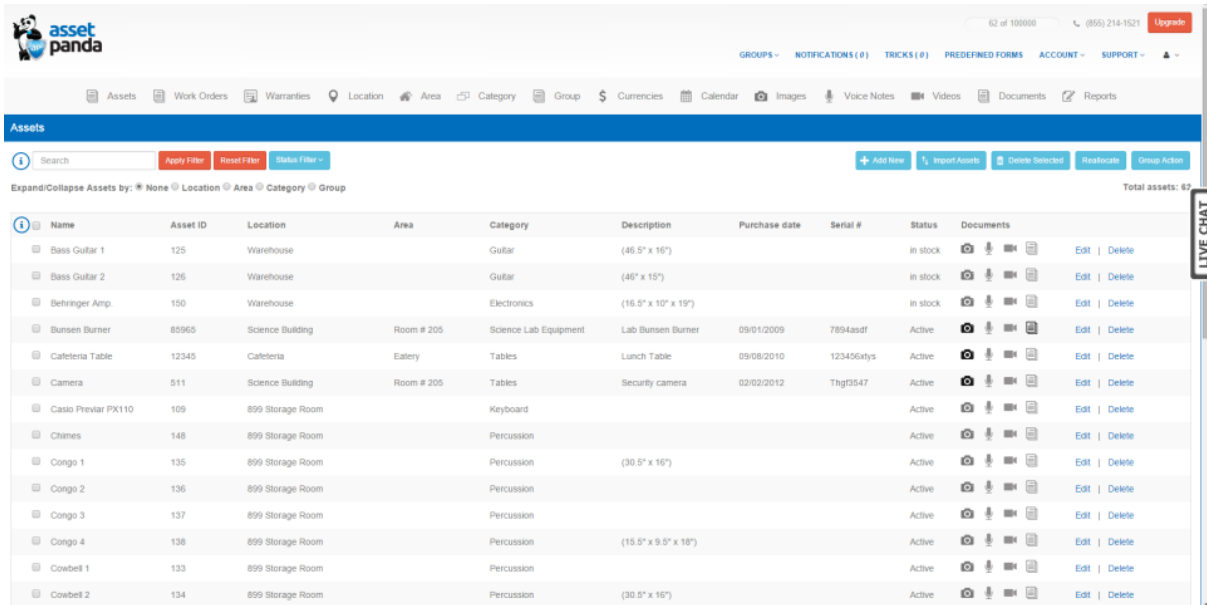
Az Asset Panda egy felhőalapú eszközkezelő szoftver, amely lehetővé teszi a vállalatok számára, hogy egyszerűen nyomon kövessék, kezeljék és karbantartsák az eszközeiket. A rendszer rugalmas, és számos integrációval rendelkezik, amelyek megkönnyítik az adatkezelést és a működési folyamatokat. Az Asset Panda alkalmazásában eszközök nyomon követése QR-kódokkal történik, és lehetőség van a munkavállalók számára a közvetlen eszközkeresek indítására is. Az eszközkészlet és a riportálás funkciók is testreszabhatók, így a vállalat igényei szerint alakítható. Az alkalmazás szintén támogatja a különböző eszközkategóriák, például hardverek, szoftverek, irodai eszközök és járművek nyilvántartását. [29]

Előnyök:

- Felhőalapú, bárholnan elérhető
- QR-kódos eszközkövetés
- Testreszabható riportálás
- Integrációs lehetőségek

Hátrányok:

- Korlátozott offline funkcionalitás
- Bonyolult beállítási folyamatok kezdőknek
- Magasabb árú csomagok a nagyvállalatok számára



The screenshot shows the Asset Panda web application interface. At the top, there's a navigation bar with the Asset Panda logo, user information (62 of 10000, (855) 214-1521, Upgrade), and tabs for GROUPS, NOTIFICATIONS (0), TRICKS (0), PREDEFINED FORMS, ACCOUNT, and SUPPORT. Below this is a secondary navigation bar with icons for Assets, Work Orders, Warranties, Location, Area, Category, Group, Currencies, Calendar, Images, Voice Notes, Videos, Documents, and Reports. The main section is titled 'Assets' and includes a search bar, filter buttons (Apply Filter, Reset Filter, Status Filter), and action buttons (+ Add New, % Import Assets, Delete Selected, Realize, Group Action). A table lists assets with columns: Name, Asset ID, Location, Area, Category, Description, Purchase date, Serial #, Status, and Documents. The table contains 15 rows of asset data. On the right side, there's a vertical 'LIVE CHAT' button.

Name	Asset ID	Location	Area	Category	Description	Purchase date	Serial #	Status	Documents
Bass Guitar 1	125	Warehouse		Guitar	(46.5" x 16")			In stock	
Bass Guitar 2	126	Warehouse		Guitar	(46" x 15")			In stock	
Behringer Amp.	150	Warehouse		Electronics	(16.5" x 10" x 19")			In stock	
Bunsen Burner	85965	Science Building	Room # 205	Science Lab Equipment	Lab Bunsen Burner	09/01/2009	7894asdf	Active	
Cafeteria Table	12345	Cafeteria	Eatery	Tables	Lunch Table	09/08/2010	123456xlys	Active	
Camera	511	Science Building	Room # 205	Tables	Security camera	02/02/2012	Thgt0547	Active	
Casio Previar PK110	109	899 Storage Room		Keyboard				Active	
Chimes	148	899 Storage Room		Percussion				Active	
Congo 1	135	899 Storage Room		Percussion	(30.5" x 16")			Active	
Congo 2	136	899 Storage Room		Percussion				Active	
Congo 3	137	899 Storage Room		Percussion				Active	
Congo 4	138	899 Storage Room		Percussion	(15.5" x 9.5" x 18")			Active	
Cowbell 1	133	899 Storage Room		Percussion				Active	
Cowbell 2	134	899 Storage Room		Percussion	(30.5" x 16")			Active	

5. ábra: Az Asset Panda felülete [3]

2.2 ManageEngine AssetExplorer

A ManageEngine AssetExplorer egy átfogó eszközkezelő szoftver, amely segít a vállalatoknak az eszközök teljes életciklusának kezelésében. Az alkalmazás lehetővé teszi az IT-eszközök automatikus nyomon követését, beleértve a hardvereket, szoftvereket és hálózati eszközöket is. Az AssetExplorer kiemelkedő tulajdonsága az eszközök automatikus felfedezése, amely folyamatosan naprakészen tartja az eszközkészletet. Emellett a rendszer támogatja az eszközökhöz kapcsolódó szerződések, licencinformációk és garanciák nyilvántartását is. A szoftver integrálható más ManageEngine termékekkel, így egy átfogó IT-kezelési megoldást kínál. [30]

Előnyök:

- Automatikus eszközkövetés
- Eszközök teljes életciklusának kezelése
- Integrálható más ManageEngine termékekkel
- Részletes licenckezelési funkciók

Hátrányok:

- Viszonylag magas tanulási görbe
- Az interfész nem túl felhasználóbarát
- Az alapszintű csomagok korlátozott funkcionalitást kínálnak

Software	Purchased	Installed	Allocated	Available	Allocated To Downgrades	Compliance Type	Type
Google Chrome	0	11	0	0	0	NA	Unidentified
Kaspersky Endpoint Security 10 for Windows	0	7	0	0	0	Under Licensed	Managed
Kaspersky Security Center 10 Network Agent	0	7	0	0	0	NA	Unidentified
Google Update Helper	0	7	0	0	0	Under Licensed	Managed
Mozilla Maintenance Service	0	7	0	0	0	Under Licensed	Managed
ManageEngine Desktop Central - Agent	0	6	0	0	0	NA	Unidentified
Microsoft Visual C++ 2010 x86 Redistributable	0	6	0	0	0	NA	Unidentified
Microsoft Visual C++ 2008 Redistributable - x	0	6	0	0	0	NA	Unidentified
Windows Internet Explorer 9	0	5	0	0	0	NA	Unidentified
iTunes	0	5	0	0	0	NA	Unidentified
Microsoft Visual C++ 2008 Redistributable - x	0	4	0	0	0	NA	Unidentified
Microsoft Visual C++ 2010 x64 Redistributable	0	4	0	0	0	NA	Unidentified
DbVisualizer	0	4	0	0	0	NA	Unidentified
Windows Internet Explorer 11	0	4	0	0	0	NA	Unidentified
Microsoft .NET Framework 4 Multi-Targeting	0	4	0	0	0	NA	Unidentified

6. ábra: A ManageEngine AssetExplorer felülete [4]

2.3 Pulseway

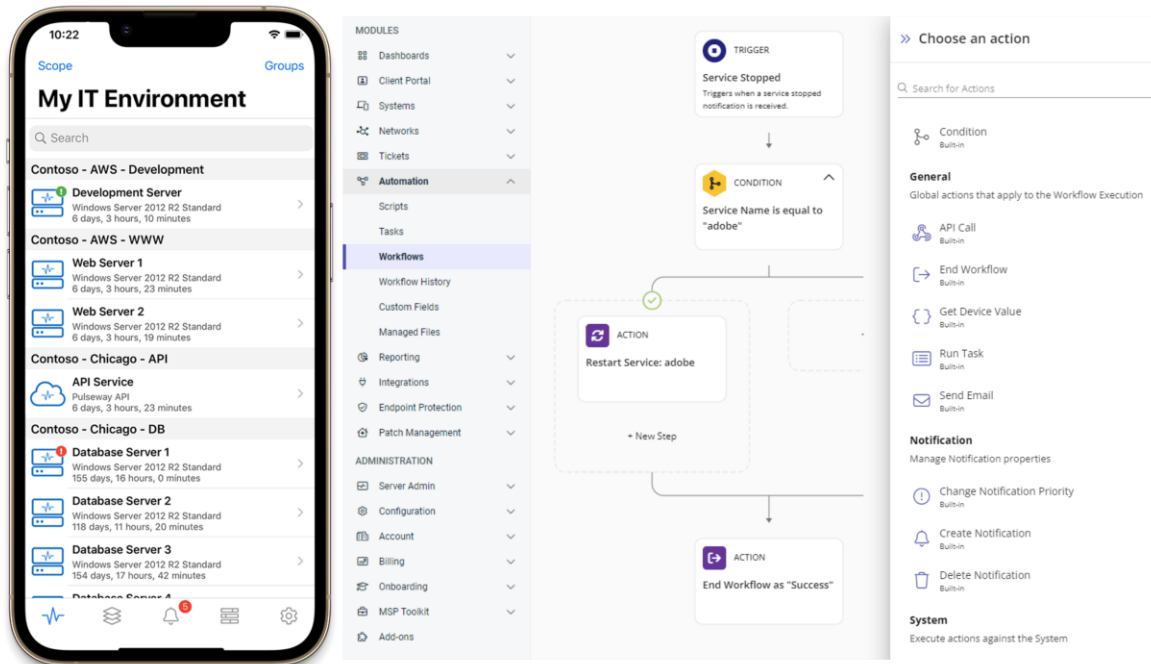
A Pulseway egy felhőalapú eszközközelő rendszer, amely kiemelkedik a valós idejű eszközkövetéssel és a mobil alkalmazással való integrációval. A szoftver segítségével az IT-szakemberek bárholonnan, bármikor figyelemmel kísérhetik az eszközöket, és azonnali értesítést kapnak a rendszer problémáiról. A Pulseway különleges funkciója a távoli vezérlés lehetősége, amely lehetővé teszi a felhasználók számára, hogy a rendszeren keresztül közvetlenül kezeljék az eszközöket. A rendszer támogatja az automatizált frissítéseket és karbantartási feladatokat is, ezáltal csökkentve a manuális munkát. [31]

Előnyök:

- Valós idejű eszközkövetés
- Távoli vezérlés és irányítás
- Mobil applikációs támogatás
- Automatizált karbantartási feladatok

Hátrányok:

- Korlátozott funkciók az ingyenes verzióban
- Magasabb árú csomagok kisvállalatok számára
- Az interfész néha túlszűfolt és nem intuitív



7. ábra: A Pulseway felülete [5]

2.4 GoCodes

A GoCodes egy QR-kód-alapú eszközkövető rendszer, amely egyszerűsíti az eszközök kezelését és karbantartását. A szoftver lehetővé teszi az eszközök gyors azonosítását és nyomon követését egyedi QR-kódok segítségével. A GoCodes nemcsak eszközkövetést, hanem a bérlet és karbantartás nyilvántartását is támogatja. Az alkalmazás mobilra optimalizált, így a felhasználók egyszerűen rögzíthetnek eszközkereséseket, és az eszközökkel kapcsolatos információkat bárholnan elérhetik. A rendszer egyszerű és könnyen használható, ideális kisebb vállalkozások számára. [32]

Előnyök:

- QR-kódos eszközkövetés
- Kisebb cégek számára ideális
- Könnyen kezelhető mobilalkalmazás
- Bérlet és karbantartás nyilvántartása

Hátrányok:

- Korlátozott funkciók nagyobb vállalatok számára
- Nincs lehetőség automatikus eszközkövetésre
- Az eszközkészlet nem igazán bővíthető egyes csomagokkal

The screenshot displays the GoCodes Asset Management web application. The top navigation bar includes links for Asset List, Map, Pictures, and GoCodes Support. The main content area is titled 'Asset Tag DGM3-M58F' and shows the asset's status as 'Checked Out' and 'Status: In Service'. A 'Private Info' section lists various details such as Sub Type (Tools), Date Purchased (Jun 30, 2013), Salvage Value (\$100.00), Current Value (\$500.00), Last Serviced (Nov 30, 2014), Next Service (Feb 14, 2019), Assigned To (Steve), and Documentation. A 'Check In/Out Log' table shows a history of check-in and check-out events with dates and durations. A 'Messages' section at the bottom shows a message from Jim at 3/3/19, 9:39 AM with the text 'Won't start, please send a replacement.' A map of the location is also visible on the left side of the interface.

8. ábra: A GoCodes felülete [6]

2.5 Ivanti

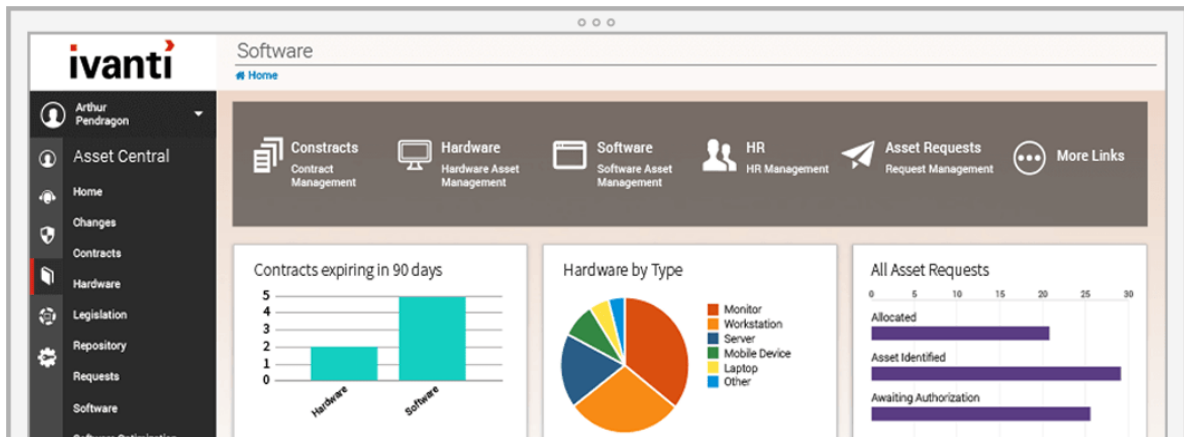
Az Ivanti egy komplex eszköz- és IT-kezelő szoftver, amely kifejezetten a nagyvállalatok számára lett kifejlesztve. Az Ivanti eszközkezelő rendszere támogatja az eszközök teljes életciklusának kezelését, beleértve a beszerzést, nyilvántartást, karbantartást és leltározást. A szoftver egyesíti az eszközkövetést a biztonságkezeléssel, így a vállalatok biztosíthatják az eszközeik védelmét és biztonságát. Az Ivanti rendszer az automatizálásra helyezi a hangsúlyt, így minimalizálva az emberi hibák lehetőségét és biztosítva a zökkenőmentes működést. Az Ivanti több integrációval rendelkezik más IT- és biztonsági rendszerekkel, így egy átfogó megoldást kínál a vállalatok számára. [33]

Előnyök:

- Komplet eszköz- és biztonságkezelési rendszer
- Erőteljes automatizálás és integrációk
- Nagyvállalati környezetekre optimalizált
- Kiváló riportálási lehetőségek

Hátrányok:

- Magas árképzés nagyvállalatok számára
- Nagy tanulási görbe a felhasználóknak
- Az alapsomagok korlátozottak az igényekhez képest



9. ábra: Az Ivanti felülete [7]

2.6 Elemzés összefoglalása

A vizsgált szoftvermegoldások fő különbsége az én alkalmazásommal szemben, hogy fizetősek. Habár rendelkeznek ingyenes próbaverzióval, ami lehetővé tette, hogy elemezzem őket, hosszútávú használatuk pénzbe kerülne. Ezzel szemben viszont rengeteg plusz funkcióval is rendelkeznek, melyeket az én témám nem foglal magába, így ezeket nem is tervezem implementálni, viszont továbbfejlesztési lehetőségként érdemes lehet ezeket is figyelembe venni. [2]

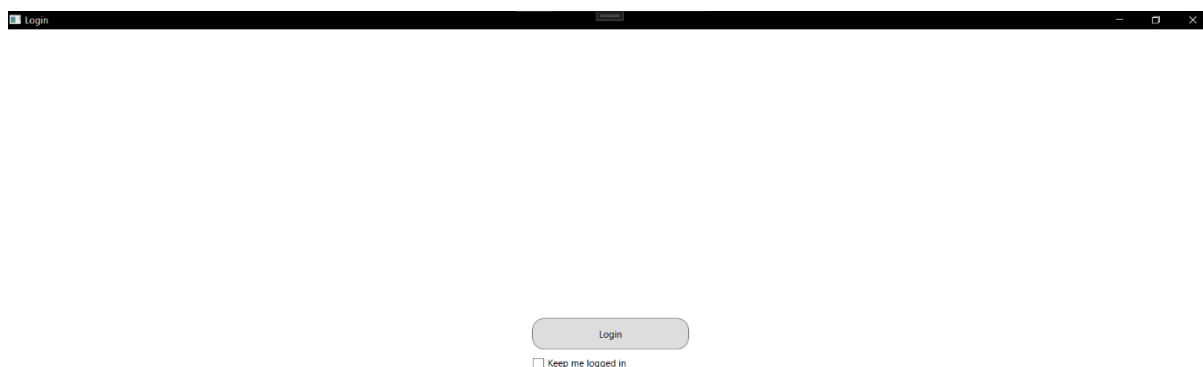
3. Funkciók ismertetése

A következő fejezet során részletesen ismertetem az alkalmazás funkcióit, felhasználói megközelítésből.

3.1 Bejelentkezés

Az alkalmazás felületének nyelve angol, mivel ezáltal szélesebb körben válik alkalmazhatóvá. Indítást követően a felhasználót egy egyszerű bejelentkezési felület fogadja. Itt egyetlen gomb, és egy jelölőnégyzet található. A gombra kattintva az alapértelmezett böngészőben megnyílik a Microsoft bejelentkező felülete, ahol email-cím – jelszó páros megadását követően jelentkezhetünk be. A bejelentkezés kötelező, anélkül nem használható az alkalmazás.

A „Keep me logged in” (maradjak bejelentkezve) jelölőnégyzetet bejelölve az alkalmazás eltárolja a bejelentkezésünket, így a következő indításkor nem kell újból megadnunk az adatainkat, hanem egyből a vagyonelemek listájára ugrik az alkalmazás.



10. ábra: A bejelentkező felület

3.2 Vagyonelemek listája

Sikeres bejelentkezést követően a felhasználót a vagyonelemek listája fogadja. Itt a képernyő 2 részre van osztva, ahol bal oldalt vagyonelemek tulajdonságai táblázatos formában jelennek

meg. A cellák fejlécére kattintva a felhasználó rendezheti a listát az adott mező értékei alapján csökkenő vagy növekvő sorrendbe.

A képernyő jobb oldalán egy szűrő ablak fogad minket, ahol az elemek tulajdonságai alapján szűrhetünk a rekordok között. Alapértelmezetten a státusz tulajdonság szűrője az aktív állapotra van beállítva, így csak az aktív elemek jelennek meg. A szűrőfeltételek beállítása után egy gombbal tudjuk alkalmazni azokat. A feltételeket alapértelmezett állapotba állítani szintén egy gomb segítségével tudjuk.

A képernyő alján 4 gomb helyezkedik el. Az első gomb segítségével tudunk vagyonelemet hozzáadni a listához. A gombra kattintva egy felugró ablak jelenik meg, ahol megadhatjuk a vagyonelem tulajdonságait. A név, elhelyezkedés, érték, és a leírás szöveges mezőben, a típus, tulajdonos, és a státusz lenyíló listával, a beszerzés dátuma pedig dátumválasztó segítségével adható meg. A „Submit” gomb csak akkor aktív, ha minden kötelezően kitöltendő mező értékét megadtuk, és ezek az értékek helyesek is. Helytelen érték esetén a mező körvonala pirosra vált. Az ablakot bezárni, és a műveletet megszakítani a „Cancel” gomb segítségével tudjuk.

The image shows a web application dialog box titled "Add Asset". It contains the following fields and controls:

- Name:** A text input field containing "Laptop".
- Asset Type:** A dropdown menu with "Hardware" selected.
- Subtype:** An empty dropdown menu.
- Owner:** A dropdown menu with "Ferenc Dániel Bercza" selected.
- Location:** A text input field containing "Fejlesztői iroda".
- Purchase Date:** A date picker showing "2025. 03. 17." with a calendar icon.
- Value:** A text input field containing "125000".
- Status:** A dropdown menu with "Active" selected.
- Description:** A large text area containing "Dell XPS 15".
- Buttons:** "Cancel" and "Submit" buttons at the bottom.

11. ábra: A hozzáadás felugró ablaka

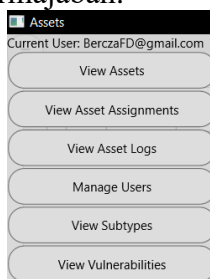
A hozzáadás gomb mellett helyezkedik el a frissítés gomb, mely akkor aktív, ha kijelöltünk egy elemet a listában. A gombra kattintva ugyan az az ablak fogad minket, mint hozzáadáskor, a mezőkben pedig a kiválasztott elem megfelelő értékei jelennek meg. Ha a

Funkciók ismertetése

frissíteni kívánt elem adatait nem megfelelő módon módosítjuk, tehát például nem adunk meg értéket egy kötelezően kitöltendő mezőnek, akkor a „Submit” gomb itt is inaktívvá válik. A „Cancel” gomb itt is ugyan úgy megtalálható. Ha a kiválasztott elem rendelkezik hozzárendeléssel, akkor a státusza nem módosítható, a lenyíló lista inaktív.

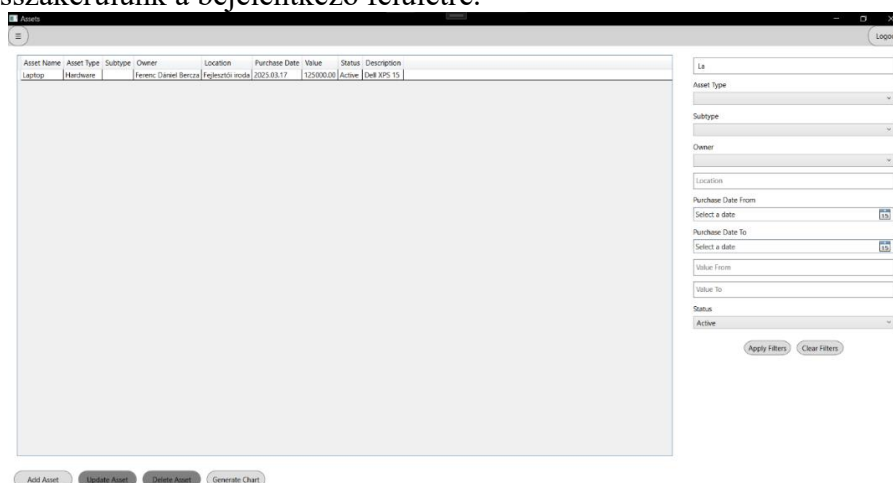
Emellett a törlés gombja található meg. Ez a gomb csak akkor aktív, ha kijelöltünk egy elemet, és ez az elem nem rendelkezik hozzárendeléssel. A gombra kattintva egy felugró ablakban meg kell erősítenünk, hogy biztosan törölni kívánjuk az elemet, melyet követően az elem státusza nyugalmazott lesz.

Végül a diagramok generálásának funkciója helyezkedik el itt. A gombra kattintva egy felugró ablak fogad minket, 2 opcióval. Készíthetünk kimutatást a vagyonelemek darabszámáról kategóriánként, oszlopdiagram formájában, vagy a százalékos eloszlásukról szintén kategóriánként, kördiagram formájában.



12. ábra: A navigációs hamburger-menü

A képernyő tetején a navigációs sávot találjuk. A bal oldalon egy hamburger-menüt találunk, ahol a különböző nézetek között válthatunk, és ahol megjelenik a jelenleg bejelentkezett felhasználó e-mail címe. Kijelentkezni a jobb oldali gombbal tudunk, melyre kattintva visszakerülünk a bejelentkező felületre.



13. ábra: A vagyonelemek listája

3.3 Hozzárendelések listája

A hozzárendelések listája ugyan azt a felépítést követi, mint a vagyonelemek listája. Ugyan úgy táblázatosan tekinthetjük meg a hozzárendeléseket. A jobb oldali szűrő panel szintén megtalálható itt is, és a navigációs sáv sem változott. A szűrő alapértelmezetten az adott naptól szűri le a hozzárendeléseket.

A képernyő alján megtalálhatjuk a hozzáadáshoz és frissítéshez szükséges gombokat is. Hozzárendelés törlésére nincs lehetőség. Ezek a gombok szintén felugró ablakokat eredményeznek, működésük megegyezik a vagyonelemek listáján található gombokkal. Egy vagyonelem egyszerre csak egy felhasználóhoz lehet hozzárendelve, viszont egy felhasználóhoz több elemet is rendelhetünk.

Asset Name	Assigned User	Assignment Date	Return Date
Laptop	Ferenc Dániel Bercza	2025.03.17	2025.03.17

Asset Name

Assigned User

Assignment Date From

2025. 03. 17.

Assignment Date To

Select a date

Return Date From

Select a date

Return Date To

Select a date

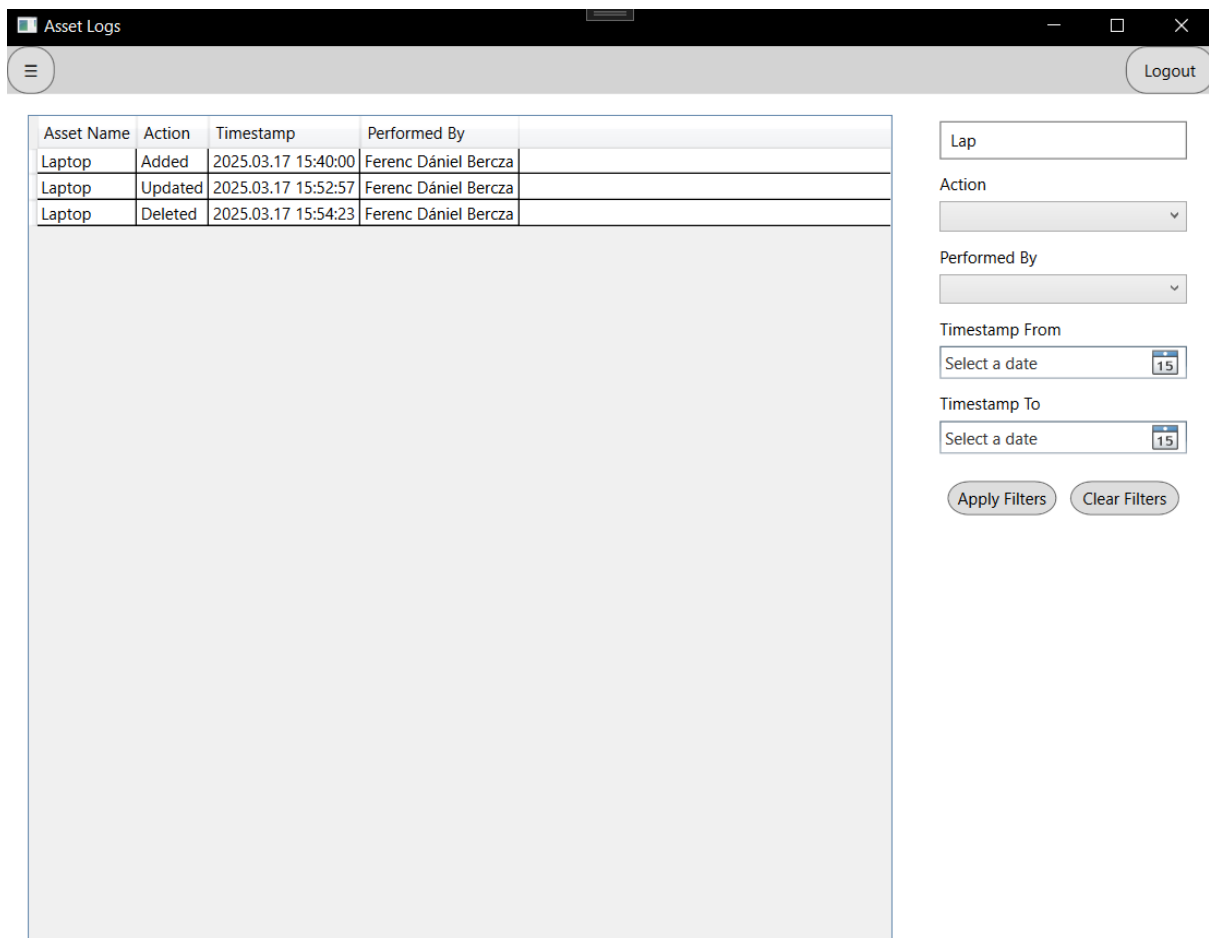
Apply Filters Clear Filters

Add Assignment Update Assignment

14. ábra: A hozzárendelések listája

3.4 Naplóbejegyzések listája

A naplóbejegyzéseket is ugyan olyan módon jelenítem meg, mint a vagyonelemeket, táblázatosan. Az ablak elrendezése itt is azonos, viszont itt nem tudunk manuálisan hozzáadni, szerkeszteni vagy törölni elemeket. Ezek a naplóbejegyzések automatikusan kerülnek hozzáadásra, amikor egy új vagyonelemet rögzítünk, vagy egy meglévőt módosítunk. A naplóbejegyzés részét képezi a vagyonelemen végzett művelet, bejegyzést előidéző felhasználó neve, és egy időbélyegző, ami a bejegyzés létrejöttének idejét jelzi.



The screenshot shows a web application window titled "Asset Logs". The main content area displays a table with the following data:

Asset Name	Action	Timestamp	Performed By
Laptop	Added	2025.03.17 15:40:00	Ferenc Dániel Bercza
Laptop	Updated	2025.03.17 15:52:57	Ferenc Dániel Bercza
Laptop	Deleted	2025.03.17 15:54:23	Ferenc Dániel Bercza

Below the table is a large empty rectangular area. To the right of the table is a sidebar with filters:

- Lap**: A text input field containing "Lap".
- Action**: A dropdown menu.
- Performed By**: A dropdown menu.
- Timestamp From**: A date picker showing "15".
- Timestamp To**: A date picker showing "15".
- Buttons**: "Apply Filters" and "Clear Filters".

15. ábra: A naplóbejegyzések listája

3.5 Felhasználók kezelése

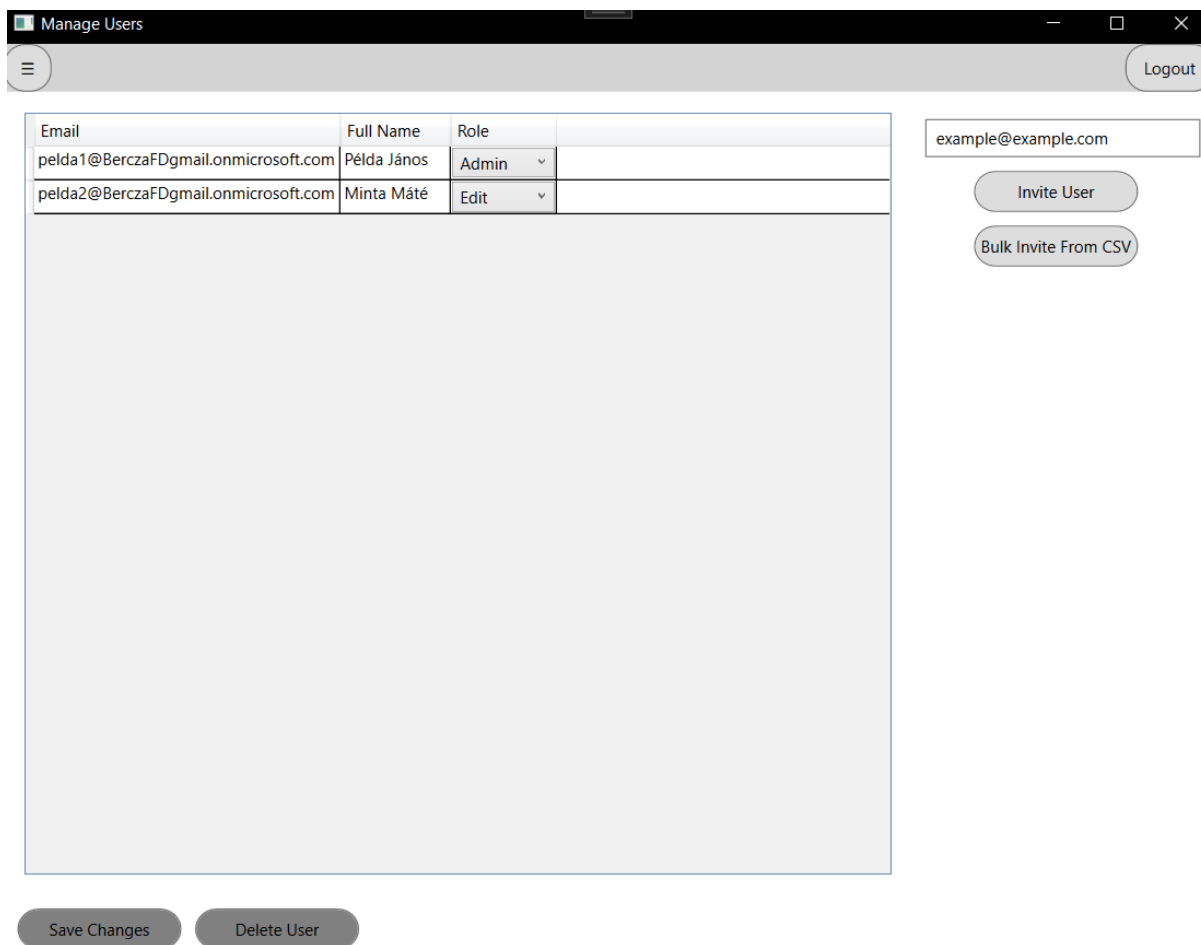
A felhasználók kezelésére az Azure felülete mellett az alkalmazáson belül is van lehetőség. A felhasználók listája a többi nézethez hasonlóan táblázatos formában jelenik meg. A lista tartalmazza a felhasználók teljes nevét, e-mail címét, és az alkalmazáson belüli jogosultsági szintjét, melyet egy legördülő lista jelenít meg. A listában a jelenleg bejelentkezett felhasználó nem kerül megjelenítésre.

Egy adott felhasználó jogosultságait a legördülő lista segítségével módosíthatjuk, mely 3 opciót tartalmaz: admin, edit, view. A „view” szerepkörrel rendelkező alkalmazások megtekinthetik az adatokat, de azokat szerkeszteni nem képesek. Az „edit” szerepkör felhasználói szerkeszthetik is azokat, míg az adminisztrátorok képesek a felhasználók kezelésére és a kockázatkezelésre is. A szerepkörök hierarchiájában feljebb elhelyezkedő szerepkörök képesek mindenre, amire az alattuk elhelyezkedőek is. Egy felhasználó szerepkörének megváltoztatása után a „Save Changes” feliratú gomb aktívvá válik, és erre kattintva elmenthetjük a módosításainkat, akár egyszerre több felhasználó jogosultságait is egyszerre módosítva.

Emellett helyezkedik el a felhasználók törlésére szolgáló gomb, mely akkor aktív, ha a listában kiválasztunk egy felhasználót. A gombra kattintva egy felugró ablakban a törlés megerősítését követően a kiválasztott felhasználó fiókja inaktívvá válik, ezáltal a továbbiakban nem tud többé bejelentkezni, és minden aktív bejelentkezése invalidálódik.

Az alkalmazáshoz új felhasználókat a jobb oldalon elhelyezkedő szövegmezőbe e-mail címüket beírva adhatunk hozzá. Ha a mező kitöltésre kerül, és a bevitt e-mail cím formátuma helyes, akkor az „Invite User” feliratú gomb aktívvá válik, és erre kattintva a felhasználó hozzáadásra kerül. A művelet sikerességéről a felhasználót felugró ablakban tájékoztatom. Amennyiben a művelet sikeres, a felhasználó alapértelmezetten „view” jogosultságokkal kerül hozzáadásra, és a felhasználók listája frissül, megjelenítve az új felhasználó adatait.

Lehetőségünk van továbbá CSV fájlból egyszerre több felhasználót is felvenni. A „Bulk Invite From CSV” gombra kattintva a felugró párbeszédablakban kiválaszthatjuk a kívánt forrásfájlt, és a művelet során felugró ablakban folyamatosan tájékoztatom a felhasználót a sikerességről.



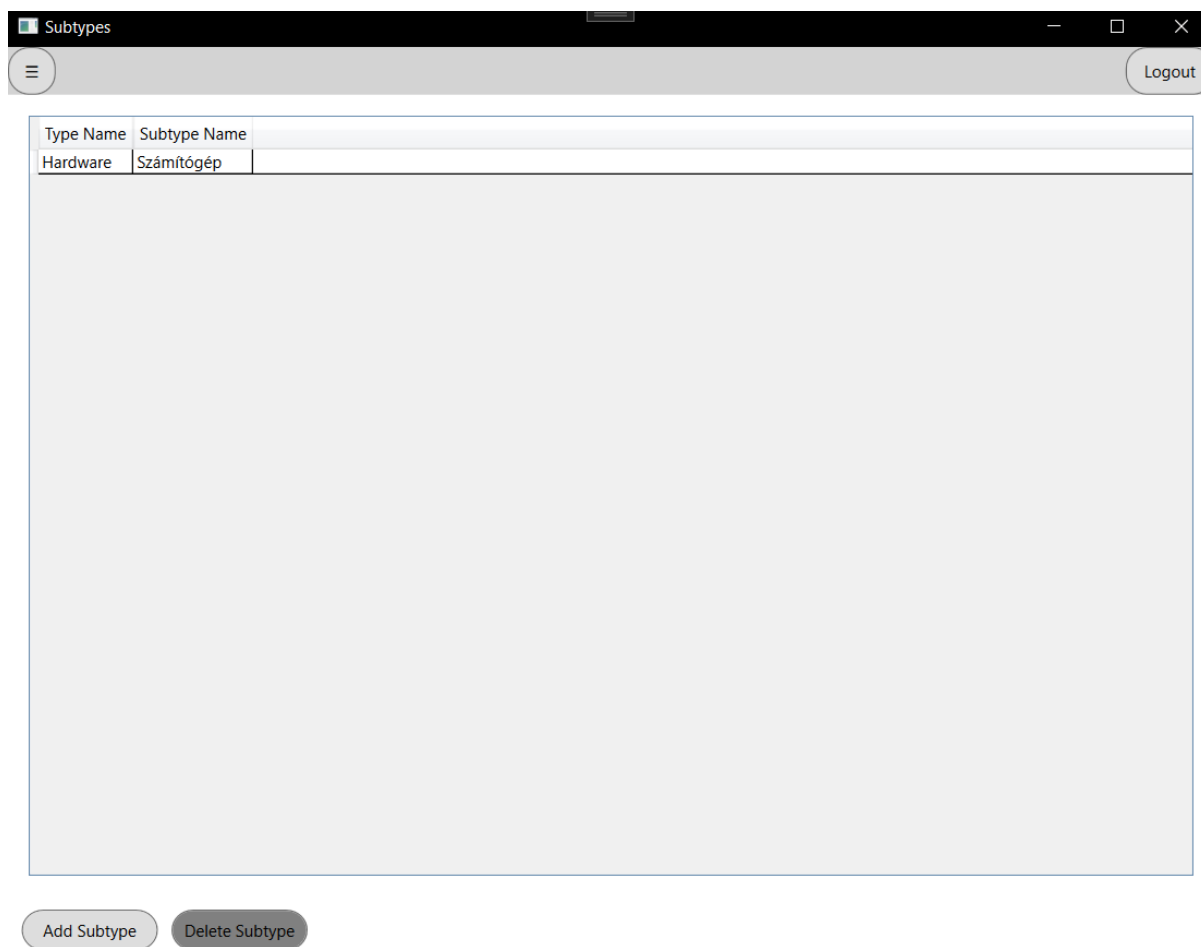
16. ábra: A felhasználók kezelésének nézete

3.6 Alkategóriák listája

Az alkalmazásban definiálhatunk alkategóriákat, melyek segítségével még alaposabban, személyre szabott módon tudjuk kategorizálni a vagyonelemeket. Minden alkategória rendelkezik névvel, és a fő kategóriával, amely alá tartozik. Ezeknek a megjelenítése is táblázatos módon történik.

Új alkategóriát az „Add Subtype” gombra kattintva hozhatunk létre. A gombra kattintva megjelenik egy felugró ablak, melyen egy szövegbeviteli mezőben adhatjuk meg az alkategória nevét, és egy legördülő listából választhatjuk ki a fő kategóriát. A „Submit” gombra kattintva az alkategória hozzáadásra kerül a listához.

Alkategóriát törölni a „Delete Subtype” gombbal tudunk. Ha a kijelölt alkategória nincsen egyetlen vagyonelemhez sem hozzárendelve, a gomb aktív lesz, és a felugró ablakban történő sikeres megerősítést követően törlésre kerül.



17. ábra: Az alkategóriák listája

3.7 Kockázatkezelési nézet

A vagyonelemek kockázatkezelés szempontjából történő értékelését egy szintén hasonló felépítésű nézetben tudjuk megtenni. A középpontban itt is egy táblázat áll, mely megjeleníti az eddig felvitt rekordokat. Adott vagyonelem értékeléséhez meg kell határoznunk, hogy milyen sérülékenységeknek és fenyegetettségeknek van kitéve, illetve értékelnünk kell egy 1 és 5 közötti skálán a bekövetkezés valószínűségét, és azt, hogy bekövetkezés esetén milyen súlyos hatással van a szervezetre. Ezen értékek szorzata alapján meghatározható, hogy az adott eset mennyire súlyos. Emellett opcionálisan megadható egy intézkedés, melyet a bekövetkezés megelőzésének érdekében tehetünk, és amelyet a kockázat mértékétől függően ajánlott elvégezni.

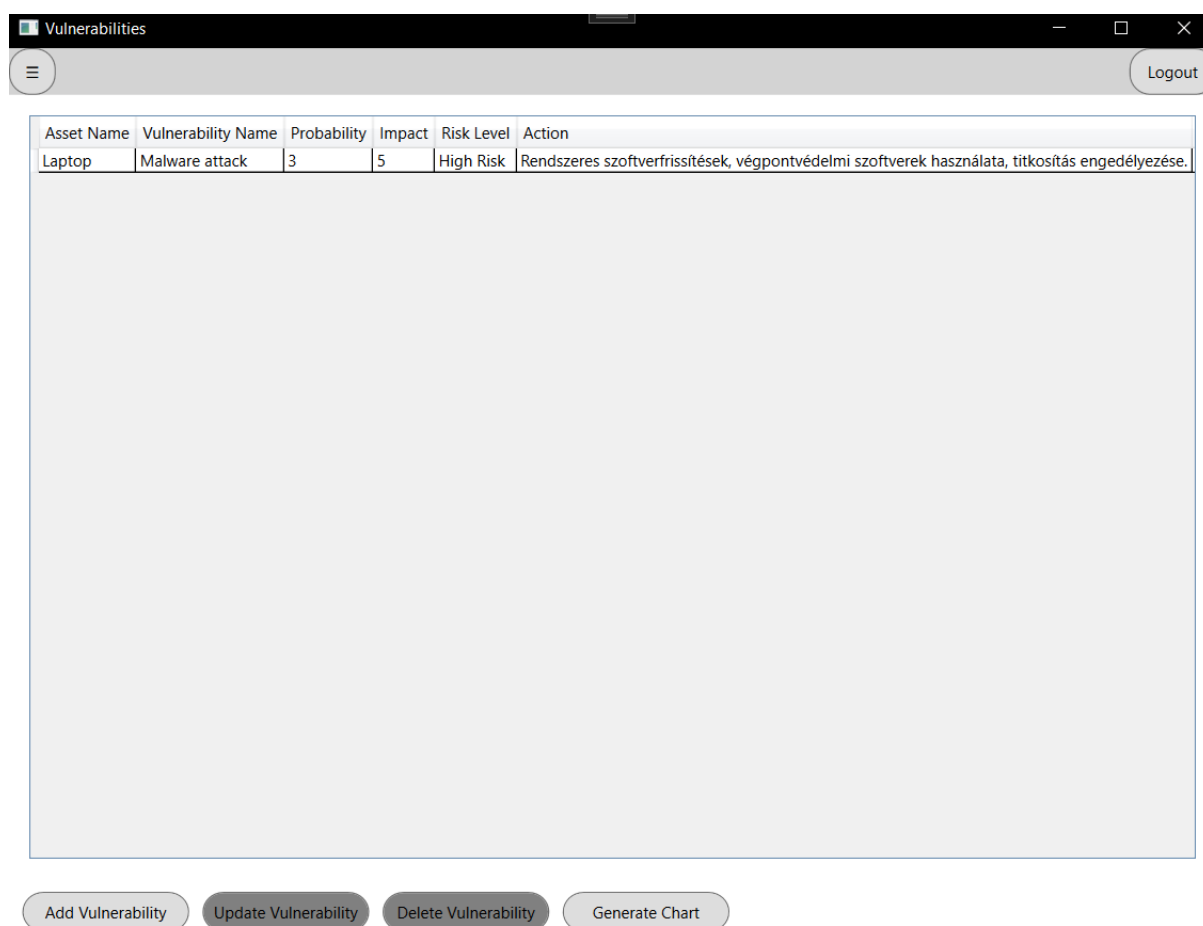
Új rekordot az „Add Vulnerability” gomb segítségével adhatunk hozzá. A gombra kattintva megjelenik egy felugró ablak, melyben kiválaszthatjuk a vagyonelem – sérülékenység párost, és megadhatjuk az 1 és 5 közötti valószínűség és hatás értékeket. Emellett opcionálisan

megadható az intézkedés. Egy vagyonelem – sérülékenység páros csak egyszer szerepelhet, így, ha egy már meglévő párost adunk meg, a „Submit” gomb inaktív lesz, illetve akkor is, ha a valószínűség és hatás értékek nem 1 és 5 közé esnek. Az adatok helyes megadását követően a „Submit” gombra kattintva a rekord hozzáadásra kerül.

Rekordot frissíthetünk az „Update Vulnerability” gombra kattintva, mely szintén felugró ablakban történik, ahol a valószínűség, hatás és intézkedés mezőket módosíthatjuk.

Rekord törlését a „Delete Vulnerability” gomb segítségével tudunk, mely akkor aktív, ha van kiválasztott rekord. Felugró ablakban történő sikeres megerősítést követően a kiválasztott rekord törlésre kerül.

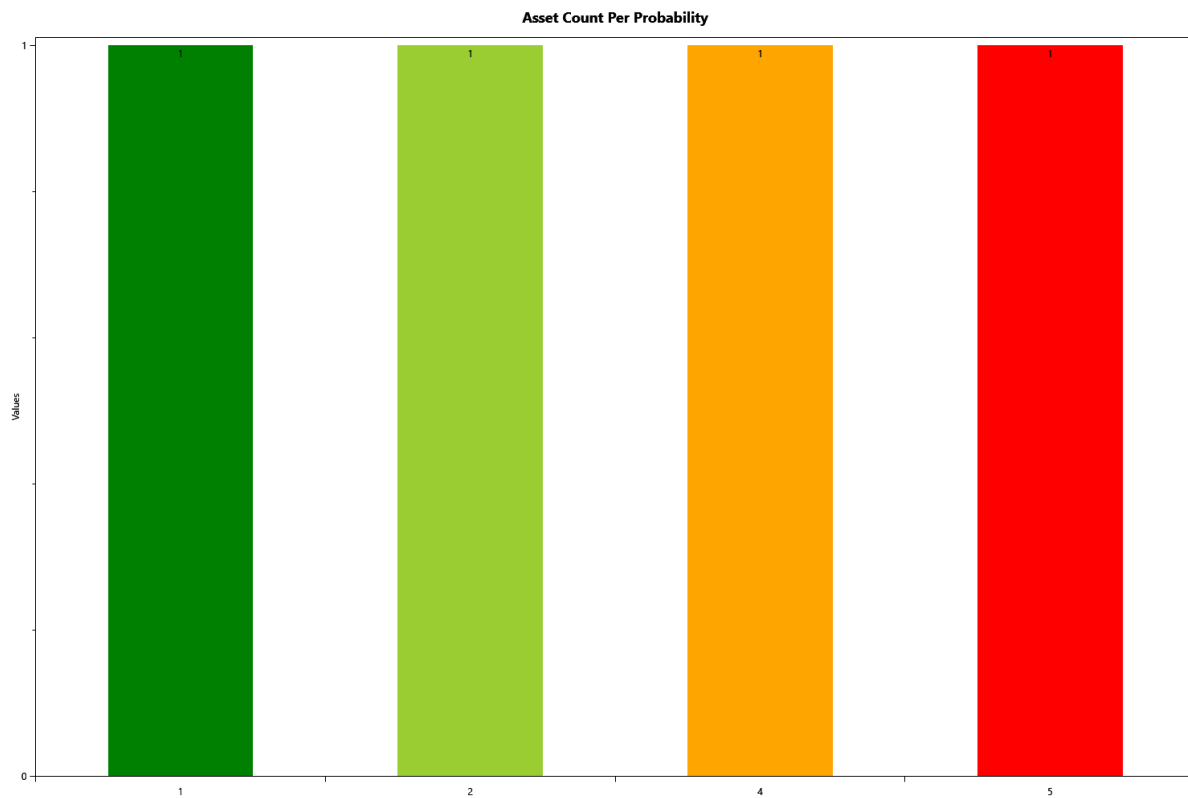
Készíthetünk továbbá kimutatásokat oszlopdiagram és kördiagram formájában is. Oszlopdiagram formájában a vagyonelemek darabszámát jeleníthetjük meg sérülékenység, valószínűség, hatás és kockázat szerint. Ugyan ezen tulajdonságok szerint készíthetünk kördiagramot a vagyonelemek százalékos eloszlásáról is.



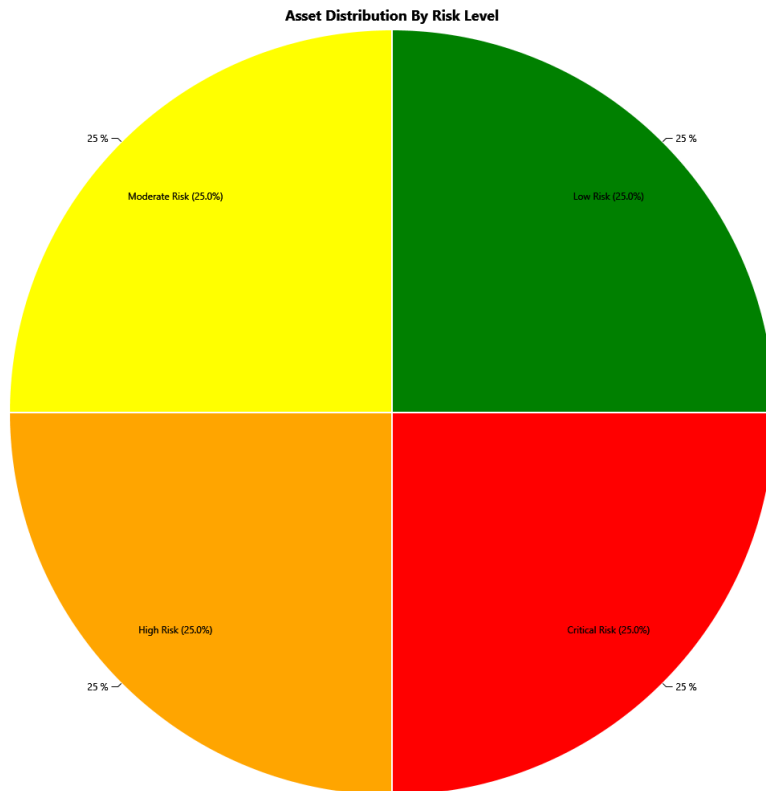
Asset Name	Vulnerability Name	Probability	Impact	Risk Level	Action
Laptop	Malware attack	3	5	High Risk	Rendszeres szoftverfrissítések, végpontvédelmi szoftverek használata, titkosítás engedélyezése.

18. ábra: A kockázatkezelés nézete

Funkciók ismertetése



19. ábra: Oszlopdiagram példa



20. ábra: Kördiagram példa

4. Megvalósítás

A következő fejezetben az alkalmazás funkcióinak a technikai hátterét, megvalósítását fogom részletesen leírni.

4.1 Adatbázis

Az adatbázis megvalósításához a Microsoft.EntityFrameworkCore.Core NuGet bővítményt használtam. Ez a bővítmény lehetővé tette, hogy a tábláimat modell osztályokként definiáljam, majd ezeket a NuGet Package Manager konzol segítségével létrehozzam az adatbázisban.

Ezekben a modell osztályok tartalmazzák a változókat, melyeket szeretnénk az adott táblában tárolni, és a megadott típusuk alapján lesznek a mezők típusai kialakítva. Emellett elláthatjuk az osztályunkat különböző annotációkkal, melyekkel olyan tulajdonságokat tudunk jelölni, mint például az elsődleges kulcs, vagy a mező maximális hossza.

```
public class Asset
{
    [Key]
    21 references
    public int AssetID { get; set; }

    [Required]
    [MaxLength(100)]
    15 references
    public string AssetName { get; set; }

    [Required]
    10 references
    public int AssetTypeID { get; set; }

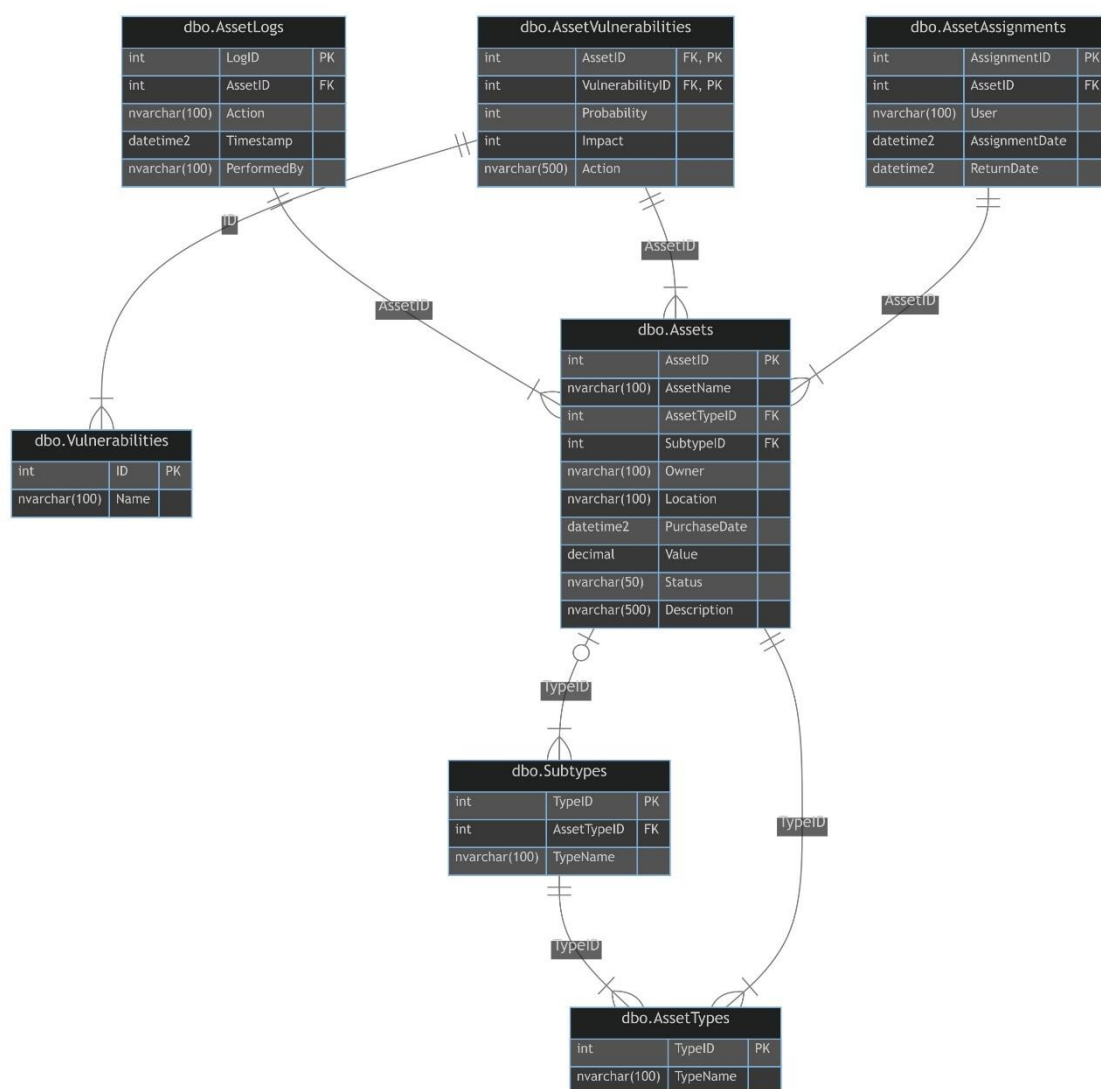
    [ForeignKey("AssetTypeID")]
    20 references
    public AssetType AssetType { get; set; }
}
```

21. ábra: Az Assets modell osztály egy részlete

Az Entity Framework segítségével a modell osztályok alapján migrációs fájlokat hozhatunk létre, melyek definiálják a modellek táblaként történő létrehozásához szükséges SQL utasításokat. Ezeket a migrációs fájlokat alkalmazhatjuk, és szükség esetén vissza is vonhatjuk, ha a modell osztályok által definiált táblák egy korábbi verziójára szeretnénk visszatérni. Migrációs fájl létrehozásához az *Add-Migration* parancs, a migráció alkalmazásához pedig az *Update-Database* parancs kiadása szükséges.

Megvalósítás

Ahhoz, hogy ezeket a parancsokat a konzolból kiadva tudjam alkalmazni a migrációs fájlokat, létre kellett hoznom egy *DesignTimeDbContextFactory* osztályt, mely implementálja az *IDesignTimeDbContextFactory* interfészt. Ennek az osztálynak a *CreateDbContext* metódusában meg kellett adnom az adatbázisom connection string-jét, hogy dizájn időben is tudjak kapcsolódni az adatbázishoz és migrációkat alkalmazni.



22. ábra: Egyed-kapcsolat diagram

Az adatbázis-séma középpontjában az *Assets* tábla áll, mely a vagyonelemekkel kapcsolatos tulajdonságokat tárolja. A tábla tartalmaz 2 idegen kulcsot, melyekkel a *Subtypes* és *AssetTypes* táblákhoz kapcsolódik, melyek a fő és alkategóriákat tárolják. Az alkategóriák táblája szintén kapcsolódik a fő kategóriák táblájához, és mivel ez nem kötelezően kitöltendő, ezért mindkét tábla kapcsolódik az *Assets* táblához, mivel ez alapján nem egyértelműen meghatározható a fő kategória. Emellett a tábla az *AssetID* elsődleges kulccsal kapcsolódik a

különböző táblákhoz. Az *AssetLogs* tábla a naplóbejegyzéseket tárolja, az *AssetAssignments* pedig a vagyonelem-hozzárendeléseket. Az *AssetVulnerabilities* tábla a sérülékenységi és fenyegetettség, és a vagyonelem párosításokat tárolja. Itt az összetett elsődleges kulcsot az *AssetID* és *VulnerabilityID* idegen kulcsok adják, ahol a *VulnerabilityID* a *Vulnerabilities* táblából származik, mely a sérülékenységek és fenyegetettségek listáját tárolja.

4.2 Függőségek injektálása

Az *App.xaml.cs* tartalmaz egy *IServiceProvider* típusú *ServiceProvider* változót, amelyben tudunk különböző osztályokat regisztrálni, és ezeket később más osztályokban lekérdezni. Ebbe a változóba az alkalmazás indulásakor regisztráljuk az adatbázis kontextusunkat, illetve nézeteinket és nézetmodelljeinket singleton-ként, mely biztosítja, hogy pontosan csak egyszer jöjjenek létre. Ez biztosítja, hogy ne példányosítsuk ezeket az osztályokat fölöslegesen, és a nézetek közötti navigáció során pedig a nézetek állapota nem veszik el.

```
services.AddSingleton<SubtypeListViewModel>();
services.AddSingleton<AssetVulnerabilityViewModel>();
services.AddSingleton<ChartExportService>();

services.AddSingleton<MainWindow>(x => new MainWindow()
{
    DataContext = x.GetRequiredService<MainWindowViewModel>(),
    WindowState = WindowState.Maximized
});
services.AddSingleton<LoginView>();
services.AddSingleton<AssetListView>();
```

23. ábra: Az osztályok regisztrálásának kódrészlete

4.3 Navigáció

A navigáció a különböző nézetek között az alkalmazásban az MVVM architektúrára épít, amely lehetővé teszi a nézetek dinamikus váltását anélkül, hogy közvetlenül manipulálnánk a felhasználói felület elemeit. A navigációt az *INavigationService* interfész és annak implementációja, a *NavigationService<TViewModel>* osztály biztosítja, míg az aktuális nézetet a *NavigationStore* tárolja és frissíti.

Magát a nézetek közötti váltást a képernyő tetején elhelyezett gombok idézik elő. Ezek a gombok a *Toolbar* részei, melyet minden nézet tartalmaz, és a mögöttes logikáját a *ToolbarViewModel* definiálja.

4.3.1 INavigationService

Az *INavigationService* egy interfész, amely meghatározza a navigációt biztosító szolgáltatás alapvető működését. Az interfész egyetlen metódust tartalmaz: *Navigate()*.

4.3.2 NavigationService

A *NavigationService<TViewModel>* osztály a *INavigationService* interfészt valósítja meg, és gondoskodik a nézetek közötti navigációról. A *Navigate()* metódus beállítja az aktuális nézetmodellt a *NavigationStore*-ban, amelynek hatására a felhasználói felület frissíti a megjelenített nézetet. A generikus típus (*TViewModel*) lehetővé teszi, hogy különböző típusú nézetmodellekkel dolgozhassunk, miközben a kód újrafelhasználhatósága megmarad.

```
public class NavigationService<TViewModel> : INavigationService where TViewModel : BaseViewModel
{
    private readonly NavigationStore _navigationStore;
    private readonly Func<TViewModel> _createViewModel;

    7 references
    public NavigationService(Func<TViewModel> createViewModel)
    {
        _navigationStore = App.ServiceProvider.GetRequiredService<NavigationStore>();
        _createViewModel = createViewModel;
    }

    7 references
    public void Navigate()
    {
        _navigationStore.CurrentViewModel = _createViewModel();
    }
}
```

24. ábra: A *NavigationService* osztály

4.3.3 NavigationStore

A *NavigationStore* felelős az aktuálisan használt nézetmodell tárolásáért és a változásairól történő értesítésért. Az *CurrentViewModel* tulajdonság tárolja az aktuális nézetmodellt, míg az *OnCurrentViewModelChanged* metódus értesíti a felhasználói felületet, hogy új nézetet kell megjeleníteni. Minden egyes nézetmodell váltásakor az előző nézetmodell *Dispose()* metódusát is meghívja, ezáltal biztosítva a memóriakezelést.


```
public class NavigationStore
{
    public event Action CurrentViewModelChanged;

    private BaseViewModel _currentViewModel;
    2 references
    public BaseViewModel CurrentViewModel
    {
        get => _currentViewModel;
        set
        {
            _currentViewModel?.Dispose();
            _currentViewModel = value;
            OnCurrentViewModelChanged();
        }
    }

    1 reference
    private void OnCurrentViewModelChanged()
    {
        CurrentViewModelChanged?.Invoke();
    }
}
```

25. ábra: A *NavigationStore* osztály

4.3.4 MainWindow

A navigáció a nézetek között a *MainWindow* XAML-jében meghatározott *DataTemplate*-ek segítségével történik. Minden nézetmodellhez (*LoginViewModel*, *AssetListViewModel* stb.) egy-egy *DataTemplate* van rendelve, amely biztosítja a megfelelő nézetek dinamikus megjelenítését. A *ContentControl* a *CurrentViewModel*-hez rendelt nézetet jeleníti meg, így a nézetek automatikusan frissülnek, amikor a navigációs logika új nézetet állít be.

```
<Window x:Class="szakdolgozat.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="clr-namespace:szakdolgozat.ViewModels"
    xmlns:v="clr-namespace:szakdolgozat.Views"
    xmlns:local="clr-namespace:szakdolgozat"
    Title="{Binding WindowTitle}" Height="710" Width="900" MinHeight="710" MinWidth="900">
    <Window.Resources>
        <DataTemplate DataType="{x:Type vm>LoginViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>AssetListViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>AssetAssignmentListViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>AssetLogListViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>ManageUsersViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>SubtypeListViewModel}" ...>
        <DataTemplate DataType="{x:Type vm>AssetVulnerabilityViewModel}" ...>
    </Window.Resources>
    <Grid>
        <ContentControl Content="{Binding CurrentViewModel}" />
    </Grid>
</Window>
```

26. ábra: *MainWindow.xaml*

4.4 Autentikáció, felhasználók kezelése

Az alkalmazás hitelesítési rendszere a Microsoft Entra ID platformot használja a felhasználók bejelentkezéséhez, amely az OAuth 2.0 és OpenID Connect protokollokra épít. A bejelentkezési folyamat célja, hogy biztosítsa a felhasználói adatok védelmét és lehetővé tegye a jogosultságok kezelését az alkalmazásban. A hitelesítés végrehajtása az *AuthenticationService* osztály felelőssége, míg a felhasználói bejelentkezéshez szükséges interakciókat a *LoginViewModel* végzi.

4.4.1 AuthenticationService

Az *AuthenticationService* osztály biztosítja a felhasználó hitelesítését a Microsoft Entra ID segítségével. A hitelesítés az MSAL (Microsoft Authentication Library) használatával történik. Az osztály felelős a bejelentkezéshez szükséges tokenek beszerzéséért, azok tárolásáért, és a felhasználó jogosultságainak kezeléséért. Emellett a felhasználók kezelésével kapcsolatos metódusok is itt találhatók.

Maga az osztály a singleton mintát valósítja meg, azaz a program életciklusa során egyetlen példánya jön létre, ami bárhol elérhető. Konstruktora privát, és rendelkezik egy publikus statikus *Instance* változóval, mely az osztály példányát tárolja, és amelyen keresztül a program bármely pontján elérhető. Az osztály emellett tartalmaz adattagokat a jelenleg bejelentkezett felhasználó, az Entra ID eléréséhez, és az adatbázis eléréséhez szükséges tokenek tárolására is.

A konstruktorban először a *.env* fájlból betöltésre kerülnek a szükséges kliens és bérlő azonosítók, melyek a bejelentkezés megvalósításához szükségesek. Ezután létrehozásra kerül az MSAL kliens a megfelelő kliens azonosítóval, és megadásra kerül az Azure Entra ID végpont URL-je, amire a bejelentkezési kérélmeket fogjuk küldeni. Megadjuk továbbá a visszahívási URI-t, amelyen az alkalmazás várja a választ.

```
private AuthenticationService()
{
    Env.TraversePath().Load(".env");
    ...
    _authority = $"https://login.microsoftonline.com/{_tenantId}";

    _publicClientApp = PublicClientApplicationBuilder.Create(_clientId)
        .WithAuthority(_authority)
        .WithRedirectUri("http://localhost:5000")
        .Build();

    InitializeTokenCache();

    _httpClient = new HttpClient();
}
```

27. ábra: Az *AuthenticationService* konstruktora

A *LoginAsync()* metódus interaktív módon végzi el a felhasználó bejelentkezését, és megszerzi a szükséges hozzáférési tokent. Sikeres bejelentkezés esetén *true*, sikertelen bejelentkezés vagy kivétel esetén *false* értéket ad vissza.

A *TryAutoLoginAsync()* metódus lehetővé teszi a felhasználó automatikus bejelentkezését, ha a hitelesítési információk korábban már el lettek mentve a helyi fájlba. Sikeres bejelentkezés esetén *true*, sikertelen bejelentkezés vagy kivétel esetén *false* értéket ad vissza.

A *Logout()* metódus törli a felhasználói adatokat és a hozzáférési tokent, valamint eltávolítja a hitelesítési cache-t.

A *TokenCacheHelper* segédosztály felelős a tokenek helyi tárolásáért és azok kezeléséért. A tokeneket fájlban tároljuk (*msal_cache.json*), és azokat a MSAL könyvtár segítségével betöltjük, illetve elmentjük. A tokenek titkosítva kerülnek tárolásra az adatok védelme érdekében. A titkosításhoz a Windows Data Protection API-t használom, amely az AES algoritmust használja.

Mivel a felhasználók kezelésére az Entra ID-t használom, és az adatbázisban csak az Entra ID-beli azonosítójukat tárolom, így a felhasználók lekérdezéséhez, mely pl. a szűrők lenyíló listájához szükséges, itt található a *GetAllUsersAsync* metódus. A metódus aszinkron módon kérést küld a megfelelő Microsoft Graph API végpontnak, és az *accountEnabled+eq+true* szűrőfeltétel segítségével csak az aktív felhasználói fiókok listáját kérdezi le, amelyet visszatérési értéként visszaad. A végponttól érkező választ a *UserListResponse* segédosztály segítségével a *JsonConvert.DeserializeObject* metódus deszerializálja.

```
public async Task<List<UserProfile>> GetAllUsersAsync()
{
    var users = new List<UserProfile>();
    var userApiUrl = "https://graph.microsoft.com/v1.0/users?$filter=accountEnabled+eq+true";

    var requestMessage = new HttpRequestMessage(HttpMethod.Get, userApiUrl);
    requestMessage.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", AccessToken);

    var response = await _httpClient.SendAsync(requestMessage).ConfigureAwait(false);

    if (response.IsSuccessStatusCode)
    {
        var responseBody = await response.Content.ReadAsStringAsync().ConfigureAwait(false);
        var userData = JsonConvert.DeserializeObject<UserListResponse>(responseBody);

        if (userData?.Value != null)
        {
            users.AddRange(userData.Value);
        }
    }

    return users;
}
```

28. ábra: Az *AuthenticationService* *GetAllUsersAsync* metódusa

Szintén a felhasználók tárolásának módja miatt található itt a *GetFullNameByGuidAsync* metódus, mely egy adott felhasználó teljes nevét adja vissza, a Graph API *users* végpontjára küldve a kérést, azonosító alapján.

Egy adott felhasználóhoz tartozó szerepköröket a *GetUserRolesAsync* metódus segítségével kérdezhetjük le. Paraméterként a felhasználó azonosítóját várja, amennyiben a paraméter nem kerül megadásra, akkor a jelenleg bejelentkezett felhasználó szerepköreit kérdezi le. A deszerializálást a *UserRolesResponse* segédosztály segíti. Mivel ez a végpont a szerepkör nevét nem adja vissza, csupán az azonosítóját, így le kell kérdeznünk az összes szerepkört, és azonosító alapján megkeresni a nevét. Ezt a *GetRoleNameByAppRoleIdAsync* metódus végzi.

```
public async Task<List<string?>> GetUserRolesAsync(string id = "")
{
    var userRolesApiUrl = "";
    if (string.IsNullOrEmpty(id))
    {
        userRolesApiUrl = "https://graph.microsoft.com/v1.0/me/appRoleAssignments";
    }
    else
    {
        userRolesApiUrl = $"https://graph.microsoft.com/v1.0/users/{id}/appRoleAssignments";
    }

    var requestMessage = new HttpRequestMessage(HttpMethod.Get, userRolesApiUrl);
    requestMessage.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Bearer", AccessToken);

    try
    {
        var response = await _httpClient.SendAsync(requestMessage).ConfigureAwait(false);

        if (response.IsSuccessStatusCode)
        {
            var responseBody = await response.Content.ReadAsStringAsync().ConfigureAwait(false);
            var userRolesData = JsonConvert.DeserializeObject<UserRolesResponse>(responseBody);
        }
    }
}
```

29. ábra: Az *AuthenticationService* *GetUserRolesAsync* metódusának részlete

Adott felhasználóhoz adott szerepkört az *AssignAppRoleToUserAsync* metódus segítségével tudunk hozzárendelni. Ez a metódus szintén egy aszinkron kérést küld a megfelelő végpontnak a felhasználó és a szerepkör azonosítójával, és a visszatérési értéke a válasz sikerességétől függ.

Ahhoz, hogy új felhasználót rendeljünk az alkalmazáshoz, nem elég az Entra ID-ban létrehozni, hanem külön hozzáférést kell biztosítani az adatbázishoz is. Ezt az *AddAzureAdUserAsync* metódus végzi. A metódus e-mail cím alapján létrehozza a felhasználót az adatbázisban, és írási és olvasási jogokat rendel hozzá. A metódus a következőképpen kezeli a kivételeket: ha a felhasználó nem található, másodpercenként újra próbálkozik. Ez azért szükséges, mivel a metódus új felhasználó felvételekor hívódik meg, és amíg az Entra ID nem frissül, addig az adatbázis nem ismeri fel az új felhasználót. Kivételt eredményez az is, ha a felhasználó már létezik az adatbázisban. Ekkor egyszerűen *true* értékkel tér vissza a metódus.

Továbbá, ha sikeres a hozzáadás, szintén *true* értékkel tér vissza, bármilyen más kivétel esetén pedig *false* értékkel.

```
string userEmailWithBrackets = $"[{azureAdUserEmail}]";

string createUserSql = $"CREATE USER {userEmailWithBrackets} FROM EXTERNAL PROVIDER;";
string addDbReaderRoleSql = $"ALTER ROLE db_datareader ADD MEMBER {userEmailWithBrackets};";
string addDbWriterRoleSql = $"ALTER ROLE db_datawriter ADD MEMBER {userEmailWithBrackets};";

while (true)
{
    try
    {
        await context.Database.ExecuteSqlRawAsync(createUserSql);
        await context.Database.ExecuteSqlRawAsync(addDbReaderRoleSql);
        await context.Database.ExecuteSqlRawAsync(addDbWriterRoleSql);

        return true;
    }
    catch (Exception ex)
    {
        if (ex.Message.Contains("Principal") && ex.Message.Contains("could not be found or this principal type is not supported"))
        {
            await Task.Delay(1000);
        }
        else if (ex.Message.Contains("User, group, or role") && ex.Message.Contains("already exists in the current database"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

30. ábra: Az *AuthenticationService AddAzureAdUserAsync* metódusának részlete

Új felhasználót a *SendInvitationAsync* metódussal adhatunk hozzá az alkalmazáshoz. Paraméterként a felhasználó e-mail címét várja, és ehhez az e-mail címhez tartozó Microsoft fiókot rendeli hozzá az alkalmazáshoz. Amennyiben a felhasználó már hozzá van rendelve az alkalmazáshoz, a metódus egyből *false* értékkel tér vissza. A metódus a megfelelő API végpontra kérést küld, és amennyiben a válasz sikeres, a felhasználóhoz rendeli a megtekintéshez szükséges szerepkört az *AssignAppRoleToUserAsync* metódus segítségével, majd felveszi az adatbázisba az *AddAzureAdUserAsync* metódussal. Ha a művelet során bármelyik lépés sikertelen, a felhasználó törlésre kerül a *DeleteUserAsync* metódus segítségével, és a metódus *false* értékkel tér vissza. Sikeres esetén a metódus *true* értéket ad vissza.

```
if (response.IsSuccessStatusCode)
{
    if (AssignAppRoleToUserAsync(invitedUserId, "View").Result)
    {
        if (AddAzureAdUserAsync(invitedUserEmailAddress).Result)
        {
            return true;
        }
        else
        {
            await DeleteUserAsync(invitedUserId).ConfigureAwait(false);
            return false;
        }
    }
}
```

31. ábra: Az *AuthenticationService SendInvitationAsync* metódusának részlete

Adott felhasználói fiók hozzáférését a *DisableUserAccountAsync* metódus segítségével tudjuk letiltani. A metódus paraméterként a felhasználó azonosítóját várja, és a megfelelő végpontra küld egy aszinkron kérést, melynek a sikerességét adja vissza visszatérési értéként.

Adott felhasználó szerepkörének változtatásakor a bejelentkezési munkameneteinek visszavonásáért a *RevokeSignInSessionsAsync* metódus felel. A metódus paraméterként szintén a felhasználó azonosítóját várja, és visszatérési értéként szintén a sikerességet adja vissza.

4.4.2 ManageUsersViewModel

Az *AuthenticationService*-ben definiált, felhasználók kezelésére szolgáló metódusokat a *ManageUsersViewModel* hívja meg. A hozzátartozó nézetben táblázatos formában kerül megjelenítésre az összes felhasználó, és legördülő listával választhatjuk ki a módosítani kívánt szerepkört. A jobb oldalon elhelyezkedő szövegbeviteli mezőben adhatunk hozzá új felhasználót, vagy egyszerre többet, CSV fájlból.

A nézetmodell tartalmaz adatokat a felhasználók listájának, illetve a kiválasztott felhasználónak a tárolásához. A konstruktorban betöltésre kerülnek a felhasználók, és feliratkozunk az eseményekre, melyek a többi nézetmodellt értesítik a felhasználók listájának változásairól, hogy a szűrők lenyíló listái mindig naprakészek legyenek.

A felhasználók szerepköreinek változtatását a „Save Changes” gombra kattintva tehetjük meg, melyre kattintva a *SaveChanges* metódus végig iterál a felhasználók listáján, és ahol változás történt, meghívásra kerül az *AssignAppRoleToUserAsync* és *RevokeSignInSessionsAsync* metódus.

Felhasználói fiók hozzáféréseinek letiltásához a „Delete User” gombra kattintva a kiválasztott felhasználóval meghívásra kerül a *DeleteUserAsync* metódus. Amennyiben a felhasználó bármely vagyonelem gazdája, vagy rendelkezik olyan hozzárendeléssel, mely még nem fejeződött be, a gomb nem aktív. Ezért a *CanDeleteUser* metódus felel.

Új felhasználó felvételére az *InviteUser* metódus szolgál, mely meghívja a *SendInvitationAsync* metódust a beírt e-mail címmel. Amennyiben az e-mail cím formátuma nem megfelelő, az „Invite User” gomb nem aktív. A validációért az *IsValidEmail* metódus felel, reguláris kifejezés segítségével.

Több felhasználó egyidejű felvételét CSV fájlból a *BulkInvite* metódus végzi. A metódus először meghívja a *OpenFileDialogToSelectCsvFile* metódust, mellyel párbeszédablakban megadható a fájl elérési útja. Ezután a *ReadEmailsFromCsv* metódus olvassa be az e-mail címeket és adja vissza a címek listáját. Ezután a listán végig iterál a

metódus, és minden címre meghívja a *SendInvitationAsync* metódust. Ez a metódus szintén az *IsValidEmail* metódust alkalmazza a formátum validálásához. A metódus minden lépés során felugró ablakban tájékoztatja a felhasználót az esetlegesen nem megfelelő formátumú címekről, és a művelet sikerességéről.

```
private async Task UpdateUsersAsync()
{
    var userProfiles = await AuthenticationService.Instance.GetAllUsersAsync();
    Users.Clear();
    foreach (var userProfile in userProfiles)
    {
        if (userProfile.Email == AuthenticationService.Instance.CurrentUser.Username)
        {
            continue;
        }

        var user = new User
        {
            Id = userProfile.Id,
            Name = userProfile.DisplayName,
            Email = userProfile.Email,
            Role = userProfile.GetRole(),
            CurrentRole = userProfile.GetRole()
        };
        Users.Add(user);

        user.RoleChanged += (sender, e) => OnPropertyChanged(nameof(CanSaveChanges));
        user.CurrentRoleChanged += (sender, e) => OnPropertyChanged(nameof(CanSaveChanges));
    }
    OnPropertyChanged(nameof(Users));

    UsersChanged?.Invoke(this, EventArgs.Empty);
}
```

32. ábra: A *ManageUsersViewModel* *UpdateUsersAsync* metódusa

4.4.3 LoginViewModel

A *LoginViewModel* osztály az MVVM (Model-View-ViewModel) architektúra szerint az alkalmazás bejelentkező nézetéhez kapcsolódó logikát tartalmazza. A bejelentkezési folyamatot a *LoginCommand* parancs indítja el, amely egy aszinkron műveletet hajt végre az *ExecuteLoginAsync()* metódusban.

A bejelentkezés sikeres végrehajtása után, ha a felhasználó választotta a „Keep Me Logged In” (maradjak bejelentkezve) opciót, a tokeneket megőrizzük, és az alkalmazás a fő nézethez, az *AssetListView*-hoz navigál. Ha a bejelentkezés nem sikerül, a felhasználó nem kerül át a következő nézetre.

4.5 Vagyonelemek, hozzárendelések, naplóbejegyzések, alkategóriák listája, kockázatkezelés

A vagyonelemek listájának nézete 3 fő részre bontható. Az adatok táblázatos megjelenítéséért egy *DataGrid* felel, melynek a forrása az MVVM modellnek megfelelően adatkötéssel van megadva. A vagyonelemek az alkalmazás indulásakor lekérésre kerülnek az adatbázisból. A *DataGrid* alatt 4 gomb helyezkedik el, melyekkel a vagyonelemeken tudunk műveleteket végezni: hozzáadni, módosítani, törölni, és kimutatásokat készíteni. Szintén az MVVM-nek megfelelően, az adatok változásáról a *PropertyChanged* esemény előidézése által értesül a nézet.

A nézet tartalmaz emellett 2 komponenst, melyek a navigációval és az elemek szűrésével kapcsolatosak. Az *AssetFilter* definiálja a szűrőablak megjelenését, míg az *AssetFilterViewModel* tartalmazza a logikát. A szűrőfeltételeket gombnyomásra tudjuk alkalmazni, illetve alaphelyzetbe állítani. Az adatok változásakor a szűrőfeltételek újbóli alkalmazását szintén egy eseménnyel valósítottam meg. Az *AssetFilterViewModel* konstruktorában feliratkozok az *AssetsChanged* eseményre, melyen keresztül az *AssetListViewModel* képes értesíteni az *AssetFilterViewModel*-t, ezáltal biztosítva, hogy az adatok frissülése esetén is újra alkalmazásra kerüljenek a szűrőfeltételek. Emellett elhelyezkedik a képernyő tetején a navigációt megvalósító *Toolbar*.

```
private async Task LoadAssets()
{
    using (var scope = App.ServiceProvider.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService<AssetDbContext>();
        var assetsList = await context.Assets
            .Include(a => a.AssetType)
            .Include(a => a.Subtype)
            .Where(a => a.Status == "Active")
            .Select(a => new Asset
            {
                AssetID = a.AssetId,
                AssetName = a.AssetName,
                AssetTypeID = a.AssetTypeID,
                AssetType = a.AssetType,
                SubtypeID = a.SubtypeID,
                Subtype = a.Subtype,
                Owner = a.Owner,
                Location = a.Location,
                PurchaseDate = a.PurchaseDate,
                Value = a.Value,
                Status = a.Status,
                Description = a.Description
            })
            .ToListAsync();
        Assets = new ObservableCollection<Asset>(assetsList);
        OnPropertyChanged(nameof(Assets));

        AssetsChanged?.Invoke(this, EventArgs.Empty);
    }
}
```

32. ábra: Az *AssetListViewModel* *LoadAssets* metódusa

A hozzárendelések és naplóbejegyzések listája szintén ezt a felépítést követi, itt is megtalálható a navigációs sáv, a szűrő panel, és a hozzárendelések listáján az adatok manipulálására szolgáló gombok. Működésükben is megegyeznek, betöltéskor lekérdezésre kerülnek az adatok, és események segítségével értesül a szűrő a változásokról, és fordítva.

Az alkategóriák nézete hasonló felépítést követ, de itt nem található szűrő panel. A kockázatkezelés nézete ezzel megegyező módon szintén nem tartalmaz szűrő panelt, viszont itt az adatok manipulálására szolgáló gombok mellett szintén megtalálható a diagramok készítésének funkciója.

4.6 Adatok hozzáadása, szerkesztése

Azon nézetek esetében, ahol új adatokat vehetünk fel, vagy meglévőket módosíthatunk, mint pl. a vagyonelemek nézete, felugró ablak felelős ezért. Vagyonelemek esetében az *AssetDialogView.xaml* fájl tartalmazza a felugró ablak elrendezését, amely tartalmazza a megfelelő beviteli mezőket az adatok megadásához. Az ehhez tartozó *AssetDialogViewModel* osztály tartalmazza az üzleti logikát.

A konstruktorban opcionálisan megadható egy *Asset* típusú paraméter, mely alapján meghatározható, hogy új vagyonelemet szeretnénk felvenni, vagy meglévőt szerkeszteni. Ha a paraméter megadásra kerül, akkor a megfelelő beviteli mezők értékét a megadott *Asset* tulajdonságai alapján állítjuk be, máskülönben pedig vagy üresen hagyjuk azokat, vagy az alapértelmezett értékükre állítjuk őket. A párbeszédablak „Submit” gombbal történő bezárását követően az *Asset* visszaadásra kerül az ablakot megnyitó nézetmodell számára, ahol azon elvégezzük a megfelelő műveletet (hozzáadás, frissítés). Hozzárendelések és alkategóriák esetében is megegyezik a felépítés és működés.

```
private async void UpdateAsset()
{
    if (SelectedAsset != null)
    {
        var viewModel = new AssetDialogViewModel(SelectedAsset);
        var addAssetWindow = new AssetDialogView(viewModel);
        addAssetWindow.Title = "Update Asset";
        if (addAssetWindow.ShowDialog() == true)
        {
            var updatedAsset = viewModel.Asset;

            using (var scope = App.ServiceProvider.CreateScope())
            {
                var context = scope.ServiceProvider.GetRequiredService<AssetDbContext>();
                context.Assets.Update(SelectedAsset);
            }
        }
    }
}
```

33. ábra: Az *AssetListViewModel* *UpdateAsset* metódusának egy részlete

```
public AssetDialogViewModel(Asset? asset = null)
{
    _asset = asset ?? new Asset();
    Users = new ObservableCollection<UserProfile>(AuthenticationService.Instance.GetAllUsersAsync().Result);
    AssetTypes = new ObservableCollection<AssetType>(GetAllAssetTypes());
    Subtypes = new ObservableCollection<Subtype>();

    StatusOptions = new ObservableCollection<string> { "Active", "Retired" };

    if (_asset.Owner != null)
    {
        SelectedUser = Users.Where(u => u.Id == _asset.Owner).First();
    }
    else if (Users.Any())
    {
        SelectedUser = Users.First();
    }
}
```

34. ábra: Az *AssetDialogViewModel* konstruktorának részlete

4.7 Diagramok generálása

A diagramok generálásáért és mentéséért a *ChartExportService* osztály felelős. Két publikus metódust tartalmaz: az *ExportColumnChart* metódus az oszlopdigramok exportálását végzni, az *ExportPiechart* metódus pedig a kördiagramokét.

Mindkét metódus egy *PlotModel* típusú változót vár paraméterként, mely a diagram forrását reprezentálja. A metódusok először egy párbeszédablakot nyitnak, ahol a felhasználó kiválaszthatja a mentés helyét és a fájlnévet. Az alapértelmezett fájlnév tartalmazza a diagram típusát és a jelenlegi dátumot és időt. Ezután a diagram mentésre kerül, majd ezt követően automatikusan megnyílik.

A diagram mentését az *ExportChartToPng* metódus végzi. A diagram hátterét fehérre, a méretét 1500*1000-re állítja, és a paraméterben megkapott, felhasználó által választott elérési útra menti a diagramot, PNG kiterjesztéssel. A mentést követő megnyitást az *OpenImage* metódus végzi.

```
private void ExportChartToPng(PlotModel plotModel, string filePath)
{
    plotModel.Background = OxyColors.White;

    var exporter = new PngExporter
    {
        Width = 1500,
        Height = 1000
    };

    using (var fileStream = new FileStream(filePath, FileMode.Create))
    {
        exporter.Export(plotModel, fileStream);
    }
}
```

35. ábra: A *ChartExportService* *ExportChartToPng* metódusa

A diagramok forrásaként szolgáló adatokat az adott nézetmodell számítja, és adja át a *ChartExportService* megfelelő módszerének. A megfelelő gombra kattintva egy felugró ablak fogad minket, melyet programkódban állítok össze, és ahol kiválaszthatjuk, hogy milyen diagramot szeretnénk generálni. Ezután a nézetmodell meghívja a megfelelő módszert, amely feldolgozza és összeállítja az adatokat. A kiválasztott diagram típusától függően beállításra kerül a diagram címe, a tengelyeken elhelyezkedő feliratok, az oszlopok címkéi, a körcikkelyek címkéi stb.

```
private void GeneratePiechart()
{
    var pieModel = new PlotModel { Title = "Asset Distribution By Type" };

    ConcurrentDictionary<string, int> data = new ConcurrentDictionary<string, int>();
    foreach (var asset in Assets)
    {
        data.AddOrUpdate(asset.AssetType.TypeName, 1, (key, oldValue) => oldValue + 1);
    }

    int totalAssets = data.Values.Sum();

    var pieSeries = new PieSeries
    {
        StrokeThickness = 2.0,
        InsideLabelPosition = 0.8,
        AngleSpan = 360,
        StartAngle = 0,
        InsideLabelFormat = "{1} ({0:F1}%"
    };

    foreach (var kvp in data)
    {
        double percentage = (double)kvp.Value / totalAssets * 100;
        pieSeries.Slices.Add(new PieSlice(kvp.Key, percentage));
    }

    pieModel.Series.Add(pieSeries);

    _chartExportService.ExportPiechart(pieModel);
}
```

36. ábra: Az *AssetListViewModel* *GeneratePiechart* módszere

5. Összegzés

Szakedolgozatom témájaként egy vagyonelemeket leltározó, azokat kockázatkezelés szempontjából értékelő alkalmazást fejlesztettem. Az alkalmazás általános szemléletből közelíti meg a problémát, viszont rendelkezik a testreszabást elősegítő funkciókkal is.

Az alkalmazásban előre meghatározott kategóriákba sorolhatjuk a vagyonelemeinket, de saját alkategóriákat is definiálhatunk, ezzel elősegítve a testreszabhatóságot. Az alkalmazásban nyomonkövethetjük a hozzárendeléseket is, ezáltal biztosítva, hogy sose veszítsük szem elől a vagyonelemeket. Az alkalmazás rögzíti naplóbejegyzések formájában a vagyonelemeken végrehajtott műveleteket, ezáltal biztosítva, hogy mindig vissza tudjuk követni ki, mikor és milyen műveletet hajtott végre rajtuk.

Az alkalmazás képes a felhasználók kezelésére is, így nem kell az Azure felületére bejelentkeznünk, hogy ezt megtegyük. A felhasználók jogosultságai szerepkörökhöz vannak kötve, ezáltal biztosítva, hogy minden felhasználó csak azokhoz az adatokhoz férjen hozzá, amelyekhez rendelkezik a megfelelő jogosultsággal. Ezeket a szerepköröket módosíthatjuk egyszerre akár több felhasználónál is, és CSV fájlból hozzáadhatunk egyidejűleg több felhasználót is, ezáltal növelve a folyamat gyorsaságát és a kényelmet használat közben.

Az egyik legfontosabb funkció, a kockázatkezelés is megvalósul az alkalmazásomban. A felhasználók egy előre meghatározott listából kiválaszthatják vagyonelemenként, hogy milyen sérülékenységek és fenyegetettségek veszélyeztetik azt, és értékelhetik a bekövetkezés valószínűségét és súlyosságát, ezáltal elősegítve az eszközök védelmét és az információbiztonságot. Jelenleg az alkalmazásban manuálisan adható meg az ajánlott intézkedés minden kockázat esetében. Ezen a döntéstámogatás megvalósításával lehetne javítani, ami a legfőbb továbbfejlesztési irány lehetne.

Emellett a felhasználók különböző kimutatásokat is készíthetnek diagramok formájában, amely egy könnyen átlátható, szemléletes opciót ad az adatok vizualizációjára.

Összességében minden tervezett funkciót sikerült megvalósítanom, és rengeteg új ismeretet szereztem szakdolgozatom elkészítése során.

Irodalomjegyzék

- [1] <https://journaldev.nyc3.cdn.digitaloceanspaces.com/2018/04/android-mvvm-pattern.png>
(Megtekintés dátuma: 2024. 11. 18.)
- [2] https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhDj0kwdZXEFwiTtWHQLbJQF2yMFplbFM_7_UBFij4nEo5qThu6An4UhPSyxs3kBnE_C76Kt4Q0wDn9T8XxWbaKrNjN5Iz1LU9rsEix8qRP7MMZyTDyshFOtPBx6lbZysVRDcM50yR54Fmd/s1600/entity+framework+core+tutorial.png (Megtekintés dátuma: 2024. 11. 18.)
- [3] <https://www.appvizer.com/media/application/3358/screenshot/8815/16723.png>
(Megtekintés dátuma: 2024. 11. 18.)
- [4] <https://gdm-catalog-fmapi-prod.imgix.net/ProductScreenshot/847ec224-4475-4662-bcfd-6e356b543aad.webp?auto=format&q=50> (Megtekintés dátuma: 2024. 11. 18.)
- [5] <https://pulseway.s3.amazonaws.com/website/home-new-images/hero.png> (Megtekintés dátuma: 2024. 11. 18.)
- [6] https://static.crozdesk.com/web_app_library/screenshots/images/000/018/821/original/gocodes-asset-management-screenshot-2.png?1669223933 (Megtekintés dátuma: 2024. 11. 18.)
- [7] <https://static.ivanti.com/sites/marketing/media/images/solutions/asset-lifecycle/it-asset-lifecycle-half-min.png> (Megtekintés dátuma: 2024. 11. 18.)
- [8] <https://online.fliphtml5.com/liuhx/pioa> (Megtekintés dátuma: 2024. 11. 18.)
- [9] <https://www.forbes.com/advisor/business/software/best-asset-tracking-software/>
(Megtekintés dátuma: 2024. 11. 18.)
- [10] <https://www.javatpoint.com/csharp-features> (Megtekintés dátuma: 2024. 11. 18.)
- [11] <https://www.javatpoint.com/c-sharp-object-and-class> (Megtekintés dátuma: 2024. 11. 18.)
- [12] <https://www.javatpoint.com/net-framework> (Megtekintés dátuma: 2024. 11. 18.)
- [13] <https://learn.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-022> (Megtekintés dátuma: 2024. 11. 18.)

- [14] <https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2022>
(Megtekintés dátuma: 2024. 11. 18.)
- [15] <https://learn.microsoft.com/en-us/nuget/what-is-nuget> (Megtekintés dátuma: 2024. 11. 18.)
- [16] <https://visualstudio.microsoft.com/services/live-share/> (Megtekintés dátuma: 2024. 11. 18.)
- [17] <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0>
(Megtekintés dátuma: 2024. 11. 18.)
- [18] <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-7.0>
(Megtekintés dátuma: 2024. 11. 18.)
- [19] <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0>
(Megtekintés dátuma: 2024. 11. 18.)
- [20] <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (Megtekintés dátuma: 2024. 11. 18.)
- [21] <https://learn.microsoft.com/en-us/dotnet/architecture/maui/dependency-injection>
(Megtekintés dátuma: 2024. 11. 18.)
- [22] <https://learn.microsoft.com/en-us/ef/core/> (Megtekintés dátuma: 2024. 11. 18.)
- [23] <https://learn.microsoft.com/en-us/ef/core/modeling/> (Megtekintés dátuma: 2024. 11. 18.)
- [24] <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli> (Megtekintés dátuma: 2024. 11. 18.)
- [25] <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>
(Megtekintés dátuma: 2024. 11. 18.)
- [26] <https://learn.microsoft.com/en-us/azure/developer/intro/azure-developer-overview>
(Megtekintés dátuma: 2024. 11. 18.)
- [27] <https://learn.microsoft.com/en-us/entra/fundamentals/whatis> (Megtekintés dátuma: 2024. 11. 18.)
- [28] <https://learn.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview?view=azuresql> (Megtekintés dátuma: 2024. 11. 18.)
- [29] <https://www.assetpanda.com/> (Megtekintés dátuma: 2024. 11. 18.)

- [30] <https://www.manageengine.com/> (Megtekintés dátuma: 2024. 11. 18.)
- [31] <https://www.pulseway.com/> (Megtekintés dátuma: 2024. 11. 18.)
- [32] <https://gocodes.com/> (Megtekintés dátuma: 2024. 11. 18.)
- [33] <https://www.ivanti.com/> (Megtekintés dátuma: 2024. 11. 18.)
- [34] <https://oxyplot.github.io/> (Megtekintés dátuma: 2025. 03. 17.)
- [35] https://safetyculture.com/_next/image/?url=https%3A%2F%2Fwp-content%2Fuploads%2Fsites%2F3%2F2023%2F12%2F5x5-Risk-Matrix.png&w=1920&q=75 (Megtekintés dátuma: 2025. 03. 21.)
- [36] <https://www.isaca.org/resources/isaca-journal/issues/2021/volume-2/risk-assessment-and-analysis-methods> (Megtekintés dátuma: 2025. 03. 21.)
- [37] <https://oxyplot.github.io/public/images/example1-xamarin-mac.png> (Megtekintés dátuma: 2025. 03. 21.)