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

2024

**Témakiírás**

**Hallgatói nyilatkozat**

Alulírott Bercza Ferenc Dániel hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem Villamosmérnöki és Információs Rendszerek Tanszékén készítettem a programtervező informatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Dátum: Veszprém, 2024. 11. 14.

Bercza Ferenc Dániel

**Témavezetői nyilatkozat**

Alulírott Holló Krisztina témavezető kijelentem, hogy a dolgozatot Bercza Ferenc Dániel a Pannon Egyetem Villamosmérnöki és Információs Rendszerek Tanszékén készítette programtervező informatikus végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védésre bocsátását engedélyezem.

Dátum: Veszprém, 2024. 11. 14.

Holló Krisztina

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. Jelenleg új felhasználókat az Azure felületén tudunk hozzáadni az alkalmazáshoz.

Az alkalmazás fő funkciói közé tartozik a vagyonelemek megjelenítése, új vagyonelem hozzáadása, azok szerkesztése, törlése. Emellett a vagyonelemeket hozzárendelhetjük a felhasználókhoz. 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. Currently, new users can be added to the application via the Azure interface.

The main functions of the application include displaying assets, adding new assets, editing, and deleting them. Additionally, assets can be assigned to users. 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](#_Toc182853438)

[1.1 ISO 27001:2013 2](#_Toc182853439)

[1.2 Felhasznált technológiák 2](#_Toc182853440)

[1.2.1 C# 2](#_Toc182853441)

[1.2.2 Visual Studio 2022 3](#_Toc182853442)

[1.2.3 .NET 8.0 4](#_Toc182853443)

[1.2.4 WPF (Windows Presentation Foundation) 5](#_Toc182853444)

[1.2.5 Model–view–viewmodel 6](#_Toc182853445)

[1.2.6 Entity Framework 7](#_Toc182853446)

[1.2.7 Dependency injection 7](#_Toc182853447)

[1.2.8 GitHub 8](#_Toc182853448)

[1.2.9 Microsoft Azure 8](#_Toc182853449)

[2. Versenytársak elemzése 10](#_Toc182853450)

[2.1 Asset Panda 10](#_Toc182853451)

[2.2 ManageEngine AssetExplorer 11](#_Toc182853452)

[2.3 Pulseway 12](#_Toc182853453)

[2.4 GoCodes 13](#_Toc182853454)

[2.5 Ivanti 14](#_Toc182853455)

[2.6 Elemzés összefoglalása 15](#_Toc182853456)

[3. Funkciók ismertetése 16](#_Toc182853457)

[3.1 Bejelentkezés 16](#_Toc182853458)

[3.2 Vagyonelemek listája 17](#_Toc182853459)

[3.3 Hozzárendelések listája 19](#_Toc182853460)

[3.4 Naplóbejegyzések listája 20](#_Toc182853461)

[4. Megvalósítás 21](#_Toc182853462)

[4.1 Adatbázis 21](#_Toc182853463)

[4.2 Függőségek injektálása 23](#_Toc182853464)

[4.3 Navigáció 23](#_Toc182853465)

[4.3.1 INavigationService 23](#_Toc182853466)

[4.3.2 NavigationService 24](#_Toc182853467)

[4.3.3 NavigationStore 24](#_Toc182853468)

[4.3.4 MainWindow 25](#_Toc182853469)

[4.4 Bejelentkezés, kijelentkezés 25](#_Toc182853470)

[4.4.1 AuthenticationService 26](#_Toc182853471)

[4.4.2 LoginViewModel 26](#_Toc182853472)

[4.5 Vagyonelemek listája 27](#_Toc182853473)

[5. Összegzés 28](#_Toc182853474)

[Irodalomjegyzék 29](#_Toc182853475)

# **1. Bevezetés**

Célom egy olyan alkalmazás tervezése és fejlesztése volt, mely lehetővé teszi a felhasználók számára, hogy kategorizálják a vagyonelemeket előre meghatározott kategóriákba, 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]](#forras1) 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 kockázatfelmérés folyamatában. Ha egy adott esetben ez mégsem meghatározható, akkor pedig megadhatjuk az értéket 0-nak.

Az adott elemeknek ezen tulajdonságok mellett megadhatunk elhelyezkedést és leírást is. Ezek nem kötelező mezők, viszont kitöltésükkel plusz információkat tárolhatunk el. A vagyonelemek rendelkeznek egy státusszal is, mely lehet aktív vagy nyugalmazott. Információbiztonsági szempontból fontos, hogy sose törljünk egy vagyonelemet sem, így, ha 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.

## **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, ami ebben a kontextusban az adott vagyonelemért felelős személyt jelenti. A vagyonelemeket felhasználókhoz 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]](#forras1)

## **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 zárás 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 írnunk. 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](#forras10), [11](#forras11)]

### **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]](#forras13)

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]](#forras14)

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 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]](#forras15)

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]](#forras16)

### **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]](#forras12)

### **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 rendelkeztem 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](#forras17), [18](#forras18), [19](#forras19)]

### **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.

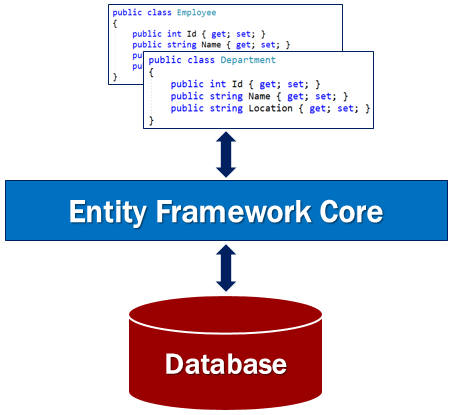
Főként azért esett a választásom erre az architektúrális mintára, mivel úgy tervezték, hogy jól működjön WPF alkalmazásokkal, és segíti a fejlesztést, mind átláthatóság, mind sebesség szempontjából. [[20]](#forras20)

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás1. *ábra: Az MVVM felépítése* [*[1]*](#forras3)

### **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](#forras22), [23](#forras23), [24](#forras24)]

2. *ábra: EF – adatbázis kommunikáció* [*[2]*](#forras4)

### **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 újrahasználhatóságát. 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]](#forras21)

### **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]](#forras25)

### **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]](#forras26)

#### **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 kezeléseit. 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 azokat az erőforrásokat és adatokat érhesse el, amelyek szükségesek a feladatukhoz. [[27]](#forras27)

#### **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]](#forras28)

# **2. Versenytársak 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 eszközkérések 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]](#forras29)

Előnyök:

•Felhőalapú, bárhonnan elérhető

•QR-kódos eszközkövetés

•Testreszabható riportálás

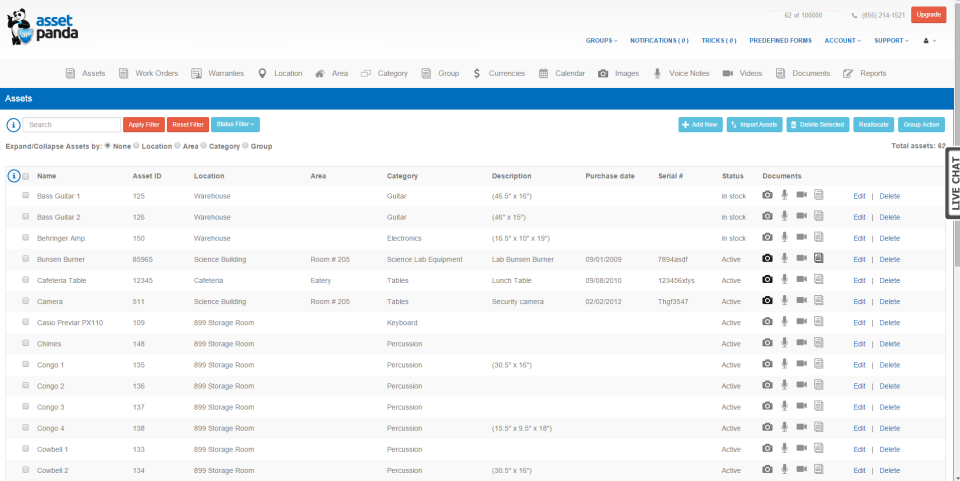
•Integrációs lehetőségek

Hátrányok:

•Korlátozott offline funkcionalitás

•Bonyolult beállítási folyamatok kezdőknek

•Magasabb árú csomagok a nagyvállalatok számára

3. *ábra: Az Asset Panda felülete* [*[3]*](#forras5)

### **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]](#forras30)

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

A képen szöveg, képernyőkép, szoftver, szám látható

Automatikusan generált leírás4. *ábra: A ManageEngine AssetExplorer felülete* [*[4]*](#forras6)

### **2.3 Pulseway**

A Pulseway egy felhőalapú eszközkezelő rendszer, amely kiemelkedik a valós idejű eszközkövetéssel és a mobil alkalmazással való integrációval. A szoftver segítségével az IT-szakemberek bárhonnan, 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]](#forras31)

Előnyök:

•Valós idejű eszközkövetés

•Távoli vezérlés és irányítás

•Mobil applikációs támogatás

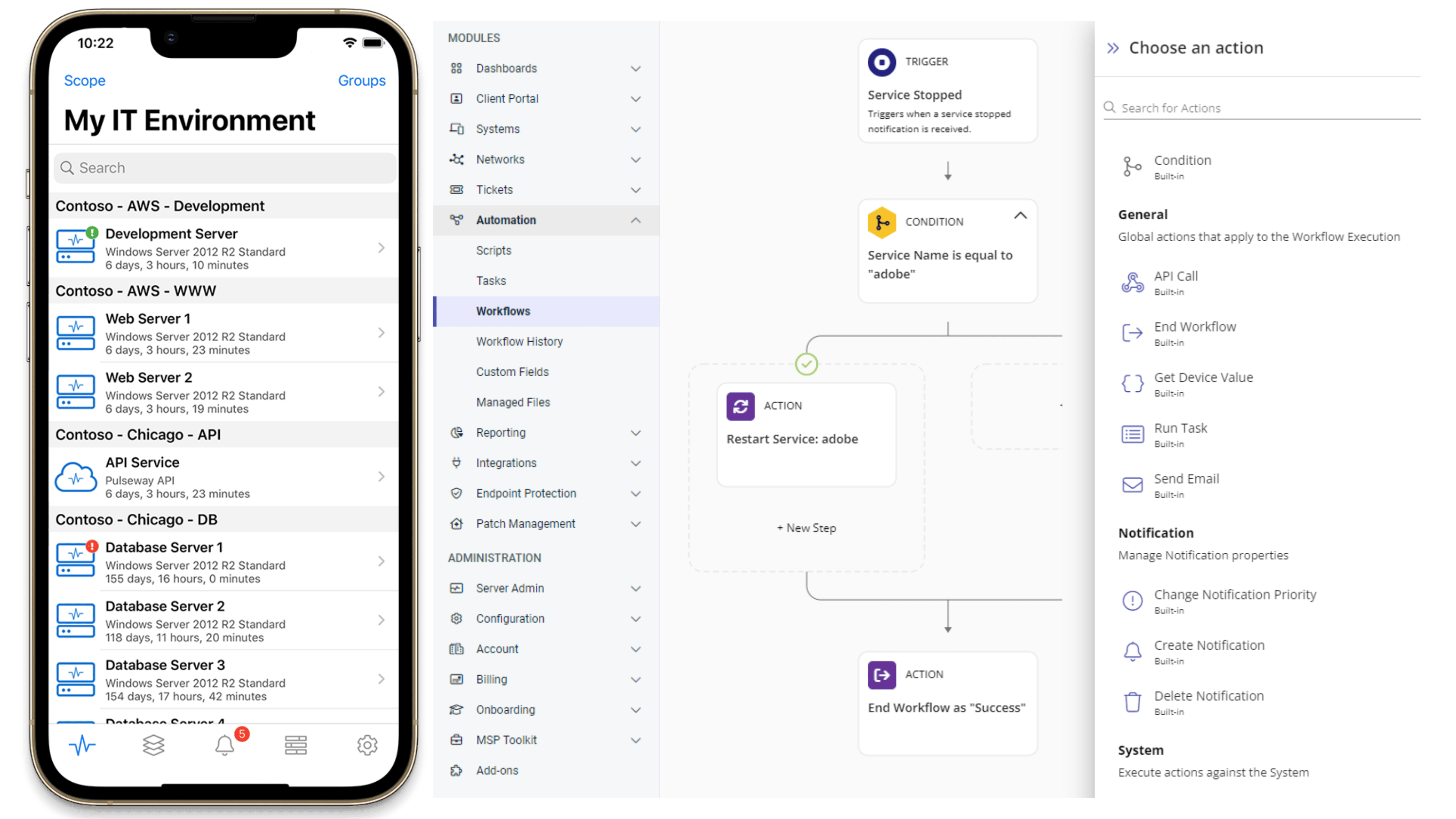
•Automatizált karbantartási feladatok

Hátrányok:

•Korlátozott funkciók az ingyenes verzióban

•Magasabb árú csomagok kisvállalatok számára

•Az interfész néha túlzsúfolt és nem intuitív

5. *ábra: A Pulseway felülete* [*[5]*](#forras7)

### **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 bérlés és karbantartás nyilvántartását is támogatja. Az alkalmazás mobilra optimalizált, így a felhasználók egyszerűen rögzíthetnek eszközkéréseket, és az eszközökkel kapcsolatos információkat bárhonnan elérhetik. A rendszer egyszerű és könnyen használható, ideális kisebb vállalkozások számára. [[32]](#forras32)

Előnyök:

•QR-kódos eszközkövetés

•Kisebb cégek számára ideális

•Könnyen kezelhető mobilalkalmazás

•Bérlés és karbantartás nyilvántartása

Hátrányok:

•Korlátozott funkciók nagyobb vállalatok számára

•Nincs lehetőség automatikus eszközkövetésre

•Az eszközkészlet nem igazán bővíthető egyes csomagokkal

A képen szöveg, képernyőkép, szoftver, Webhely látható

Automatikusan generált leírás6. *ábra: A GoCodes felülete* [*[6]*](#forras8)

### **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]](#forras33)

Előnyök:

•Komplett 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 alapcsomagok korlátozottak az igényekhez képest

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás7. *ábra: Az Ivanti felülete* [*[7]*](#forras9)

### **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](#forras2)]

# **3. Funkciók ismertetése**

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

### **3.1 Bejelentkezés**

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.

A képen képernyőkép, szöveg, számítógép, tervezés látható

Automatikusan generált leírás8. *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 3 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.

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 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.

A képernyő tetején a navigációs gombokat találhatjuk. Itt a bal oldalon válthatunk a vagyonelemek, a hozzárendelések, és a naplóbejegyzések listája között. Kijelentkezni a jobb oldali gombbal tudunk, melyre kattintva visszakerülünk a bejelentkező felületre.

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Automatikusan generált leírás9. *ábra: A vagyonelemek listája*

A képen szöveg, képernyőkép, szám, szoftver látható

Automatikusan generált leírás10. *ábra: A hozzáadás felugró ablaka*

### **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.

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Automatikusan generált leírás11. *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.

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás12. *ábra: A naplóbejegyzések listája*

# **4. Megvalósítás**

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

### **4.1 Adatbázis**

Az adatbázis megvalósításához a Microsoft.EntityFramework.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.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás13. *ábra: Az Assets.cs 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.

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

A képen szöveg, képernyőkép, Párhuzamos, tervezés látható

Automatikusan generált leírás14. *ábra: Egyed–kapcsolat diagram*

### **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.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás15. *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.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás16. *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.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás17. *ábra: A NavigationStore osztály*

### **4.3.4 MainWindow**

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírásA 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.

18. *ábra: A MainWindow.xaml egy részlete*

### **4.4 Bejelentkezés, kijelentkezés**

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.

A *LoginAsync()* metódus interaktív módon végzi el a felhasználó bejelentkezését, és megszerzi a szükséges hozzáférési tokent.

A *TryAutoLoginAsync()* metódus lehetővé teszi a felhasználó automatikus bejelentkezését, ha a hitelesítési információk korábban már el lettek mentve a helyi fájlba.

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

A bejelentkezési folyamat során a felhasználó hozzáférési tokenjét is megkapja, amely lehetővé teszi az alkalmazás számára a Microsoft Graph API elérését, valamint más erőforrások használatát.

*TokenCacheHelper* osztály felelős a tokenek helyi tárolásáért és azok kezeléséért. A tokeneket fájlban tároljuk (msal\_cache.json), és azokat a MSAL könyvtár segítségével betöltjük, illetve elmentjük.

### **4.4.2 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 listája**

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 3 gomb helyezkedik el, melyekkel a vagyonelemeken tudunk műveleteket végezni: hozzáadni, módosítani, törölni. 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*.

A hozzárendelések és a naplóbejegyzések ugyan ezen a logikán alapulnak, így azokat külön nem részletezem.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás19. *ábra: Az AssetListViewModel.cs LoadAssets metódusa*

# **5. Összegzés**

Szakdolgozatom témájaként egy vagyonelemeket leltározó alkalmazást fejlesztettem. A Tervezés I. tantárgy alatt kitűzött céljaimat sikerült megvalósítanom. Az alkalmazás alapvető funkciói elkészültek, és egy olyan alapot adnak, melyre a Tervezés II. során könnyedén fogok tudni építkezni. Sikerült megvalósítanom az adatbázist, a felhasználók kezelését, és az alkalmazásom képes az adatok megjelenítésére és alapvető műveletek elvégzésére.

Hátra van még a kockázatkezelés implementálása, illetve a felhasználók szerepek szerinti kezelése. Szeretnék továbbá kialakítani egy admin felületet is, mellyel lehetővé tenném, hogy a felhasználókat ne csak az Azure felületéről lehessen kezelni.

Összességében úgy érzem, hogy a szakdolgozatom első fázisában sikerül megvalósítanom azokat a funkciókat, melyeket kitűztem magamnak, a témavezetőm iránymutatásával.

# **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/AVvXsEhDj0kwdZXEFwiTtWH QLbJQF2yMFplbFM\_7\_UBFij4nEo5qThu6An4UhPSyxs3kBnE\_C76Kt4Q0wDn9T8Xx WbaKrNJN5Iz1LU9rsEix8qRP7MMZyTDyshFOtPBx6lbZysVRDcM50yR54Fmd/s1600 /entity+framework+core+tutorial.png](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhDj0kwdZXEFwiTtWH%09QLbJQF2yMFplbFM_7_UBFij4nEo5qThu6An4UhPSyxs3kBnE_C76Kt4Q0wDn9T8Xx%09WbaKrNJN5Iz1LU9rsEix8qRP7MMZyTDyshFOtPBx6lbZysVRDcM50yR54Fmd/s1600%09/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](https://gdm-catalog-fmapi-prod.imgix.net/ProductScreenshot/847ec224-4475-4662-bcfd-%096e356b543aad.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/goc odes-asset-management-screenshot-2.png?1669223933](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](https://static.ivanti.com/sites/marketing/media/images/solutions/asset-lifecycle/it-asset-%09lifecycle-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](https://learn.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-%09022) (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](https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-%09core-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](https://learn.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-%09overview?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.)