ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**O.Nagy Ármin Ivó**
2022	Hallgató törzskönyvi száma:	**T/006334/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **O.Nagy Ármin Ivó**
Törzskönyvi száma:
Neptun kódja: [redacted]

A dolgozat címe:

Pluginnal bővíthető platformfüggetlen fájlkezelő rendszer
Plugin extensible cross-platform file manager software

Intézményi konzulens: Kiss Dániel
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

Minden operációs rendszer egyik leggyakrabban használt programja a fájlkezelő. Grafikai felületének köszönhetően megkönnyíti számítógépen tárolt képeink, videóink, mappáink és egyéb dokumentumaink különféle műveleteinek elvégzését.
A dolgozat célja egy olyan fájlkezelő rendszer kialakítása, amely nem csak az alapvető menedzselési műveleteket látja el, hanem lehetőséget biztosít külön szoftverspecifikus bővítményekkel kiegészíteni a program funkcióit. Alapvető műveletek közé tartozik az adatok áthelyezése, törlése, másolása, átnevezése és megnyitása. A felhasználónak legyen lehetősége egy átlátható, két paneles grafikai felületen megvalósítani a szükséges műveleteket, ahol a panelek az adott mappa tartalmát mutatják. Legyen lehetőség választani különböző fájlrendszer megjelenítési nézetek közül is, tetszés szerint. Elvárás továbbá, hogy a szoftver platformfüggetlen keretrendszerben készüljön.

A dolgozatnak tartalmaznia kell:

- az elkészítendő alkalmazással szembeni pontos elvárásokat,
- a fontosabb, grafikus felület készítésére alkalmas keretrendszerek bemutatását,
- a jelenleg aktívan használt platformfüggetlen keretrendszerek ismertetését és összehasonlítását,
- az elkészítendő alkalmazás részletes rendszertervét,
- az implementáció folyamatának részletes dokumentációját,
- az elkészített alkalmazás tesztelési tervét, valamint a teszteredmények ismertetését.

ÓBUDAI EGYETEM
Neumann János Informatikai Kar
Budapest

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

....
külső konzulens intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.12.

O. Nagy Tímea
.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: **O. Nagy Ármin Ivó**

Neptun Kód: [redacted]

Tagozat: **nappali**

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

Pluginnal bővíthető platformfüggetlen fájlkezelő rendszer

Szakdolgozat címe angolul:

Plugin extensible cross-platform file manager software

Intézményi konzulens:

Kiss Dániel

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 23.	Tématerv egyeztetése	Kiss Dániel
2.	2021. 11. 04.	Platformfüggetlen technológiák	Kiss Dániel
3.	2021. 11. 25.	Specifikáció és a rendszer felépítése	Kiss Dániel
4.	2021. 12. 09.	Dolgozat és prezentáció áttekintése	Kiss Dániel

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

Kiss Dániel

Intézményi konzulens

Budapest, 2021. december 10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: **O. Nagy Ármin Ivó**

Neptun Kód: [redacted]

Tagozat: **nappali**

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

Pluginnal bővíthető platformfüggetlen fájlkezelő rendszer

Szakdolgozat címe angolul:

Plugin extensible cross-platform file manager software

Intézményi konzulens: **Kiss Dániel**

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2022. 02. 23.	Implementáció ütemezése	Kiss Dániel
2.	2022. 03. 16.	Bővítmények illesztése a rendszerhez	Kiss Dániel
3.	2022. 03. 29.	Viselkedés-alapú ajánlás modellezése	Kiss Dániel
4.	2022. 05. 10.	Dolgozat szövegének véglegesítése	Kiss Dániel

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

Kiss Dániel

Intézményi konzulens

Budapest, 2022. május 11.

TARTALOM

1. BEVEZETÉS, CÉLMEGHATÁROZÁS

1.1 MOTIVÁCIÓ

A dolgozatom célja egy olyan jártasságon alapuló módszertan kialakítása, ami visszajelzést és tapasztalati megosztás biztosít minden olyan egyén számára, aki a 2020-as évek elején platformfüggetlen keretrendszerben tervez projektet kialakítani.

Az ilyen nemű fejlesztési folyamat demonstrálására egy fájlkezelésre alkalmas rendszer megtervezését és implementálását tűztem ki célul. A választott téma miértje egy olyan szoftver kialakítása platformfüggetlen térben, amely egy jelentősen operációs rendszer függő, ezáltal előre láthatóan prominens nehézségekre számítok a fejlesztési folyamat során.

A dolgozat során első fejezetek között szó lesz az alapvető felhasználói felület készítésére alkalmas rendszerekről. Ezt követően ismeretésre kerülnek a már aktívan használt, vagy éppen újonnan beérkező platformfüggetlen keretrendszerek és azok tulajdonságai. Az működtetési eszköz kiválasztását követően, egy tervezési folyamat során demonstálásra kerülnek az implementáláshoz szükséges részegységek és azok kapcsolatai. A megvalósítás rész tartalmaz személyes jegyzeteket, nehézségeket, amelyekkel a fejlesztés során találkoztam. Utolsó pontként ismertetek minden olyan gyakorlati tényezőt, amely releváns lehet minden platformfüggetlen fejlesztéssel kapcsolatos érdeklődés mutató egyén számára.

A fájlkezelő rendszer definiálása és platformfüggetlenség jelentésének tisztázása a 1.2 és a 1.3 alfejezetekben történik.

1.2 FÁJLKEZELŐ RENDSZER ÉS ALAPÉRTELMEZETT FUNKCIÓI

A különböző operációs rendszereken megtalálható beépített fájlkezelő rendszerek szinte minden kezdő és egyaránt minden haladó informatikai tudással rendelkező személy számára ismertek. Az ilyen szoftverek a számítógép háttértárán található dokumentumok, képek, videók, zenék, mappák és egyéb fájlok vizuális megjelenítésére és kezelésére specializálódtak.

Minden ilyen rendszer képes az egyes megnyitásra alkalmas fájlok **futtatására**. Továbbiakban az egyes dokumentumokat képes egyik virtuálisan definiált pozícióból, esetleg egyik meghajtóról a másikra **átmásolni** vagy **áthelyezni**. Támogatja nem mellesleg az egyes állományok szabadon **átnevezését**, a nem kívánt adathalmazok **törlését**, valamit új **mappa létrehozását**.

Az új típusú fájlkezelő rendszerek jellemzően versengenek egyrész a megjelenésükben, funkciólistájukban, illetve a felhasználói kezelhetőség barátságos megközelítésében.

1.3 MIT JELENT A PLATFORMFÜGGETLENSÉG?

A Statista nevezetű cég már több mint 12 éve piac vezető a megbízható üzleti adatok szolgáltatásában. Egy 2012 és 2021 között tartott felmérésük alapján, a mai napig a leggyakrabban használt számítógépes operációs rendszerek a Windows, a MacOS és a Linux. A platformokra írt szoftverek szükségessének eloszlása közel sem koncentrálódik le kizárólag csak az egyik rendszerre [39].

A szoftverfejlesztési világban ismert egyének tudják, hogy az ezekre alkotott programok írásának megvannak a maga korlátai, szükségletei. A programozók természetesen próbálják a felhasználók széles spektrumát megcélozni, de szükség van arra, hogy egy alkalmazás mind a 3 fent említett rendszeren egyaránt működőképes legyen. Ehhez viszont szükségszerű lenne minden egyes platformra külön-külön megírni az applikációt, amely sok esetben eltérő nyelvezetet és fejlesztési környezetet igényel.

Ezen kihívás megoldására jelentek meg a platformok közötti fejlesztési megközelítések. Ezek lehetővé teszik a fejlesztők számára, hogy egy lépésben valósítsák meg alkalmazásaikat több platformra, elkerülve az ismétléseket és növelve a termelékenységet [13].

1.4 A MEGVALÓSÍTANDÓ RENDSZER

A beépített fájlkezelő mellé társulva számos internetről letölthető és telepíthető különböző cégek/magánszemélyek által készített fájlkezelő szoftverek is elérhetőek. Ezek az esetek túlnyomó részében első sorban a funkciólistájukban másodrészt a grafikai megjelenítésükben, illetve felhasználóbarát kezelésükben versengenek.

A célként kitűzött megvalósítandó rendszer egy olyan fájlkezelő, amely bár, kizárólag az egyszerűbb fájlkezelő rendszerek alapvető vonásait örököli, de olyan felhasználói keretrendszerben íródik, amelynek különlegessége a platformfüggetlenség. Továbbá a megtalálható eszköztár kibővítésére lehetőséget fog kapni a program használója, szabadon injektálható bővítmények segítségével.

Elsődleges törekvésünk az, hogy olyan rendszert válasszunk, amely világ szinten a legelterjedtebb és legtöbb személy által használt operációs rendszerek számára kompatibilis legyen. Ezek név szerint a Windows, a Linux, illetve a MacOS. Különleges előnyt jelentene, ha a keretrendszer iOS és Android telefonos operációs rendszereken egyaránt használható lenne.

Másodlagos cél a natív futtathatóság és ehhez minél közelebb álló megjelenítés. Értjük ezalatt, hogy a szoftver egyes megjelenítési eszközei, köztük leginkább a vezérlők, az egyes operációs rendszerekbe beépített elemekből épüljenek fel.

Magas prioritás még a fejlesztői környezet ismertsége, dokumentációjának terjedelme és a program stabilitása. Ezeket legfőképpen ismert nagyobb cégek által kiadott szoftvereknél, vagy régóta kiadott és aktívan fejlesztett népszerű termékeknél tapasztalhatjuk meg.
Szoros versenyhelyzet esetén a megjelenítés mikéntje is lehet döntő szempont. Két hasonlóan felépített és célnak megfelelő keretrendszer közül az kerül választásra, amely inkább közel áll egy 2020-as évekhez illő modern letisztul grafikai formához.

1.5 KERETRENDSZER KIVÁLASZTÁSÁNAK MÓDJA

Kezdetben említésre kerülnek a leggyakrabban használt desktop alkalmazásokat készítő .NET rendszerek. Névlegesen a Windows Forms és a Windows Presentation Foundation. Ezek története és lényegi működése leírásra kerül a később létrehozott keretrendszerekkel való összehasonlítás gyanánt.

Az ezt követő pár fejezetben bemutatásra kerül számos korszerű és aktívan használatban lévő, vagy frissen megjelent rendszer. Ezek történeti és funkcionális áttekintése, működtetési stratégiája, illetve egy kismértékű összehasonlítás rovat a régi/új grafikus felhasználói felületekkel.

2. KLASSZIKUS GRAFIKUS FELHASZNÁLÓI FELÜLET KÉSZÍTÉSÉRE ALKALMAS KERETRENDSZEREK

2.1 WINDOWS FORMS

2002-ben a .NET Foundation megjelentette a Windows Forms nevű ingyenes és szabad forráskódú grafikai felhasználói felületét, amellyel egyszerűen készíthetünk Windows asztali alkalmazásokat [7].

Átláthatóságát segíti a Visual Studio-ban található vizuális megjelenítő rész, ahol drag-and-drop funkció segítségével könnyedén megtervezhetjük a programunk felhasználó felé mutatott megjelenését. Futását események szabják meg, tehát a szoftver vár egy felhasználói interakcióra és az alapján határozza meg a működését. Ez lehet akár egy szövegdoboz használata, vagy egy gomb lenyomása is, ezek teljes mértékben személyre szabhatóak és funkcióik könnyedén implementálhatók a fejlesztők számára.

Ezek mellett tartalmaz egy úgynevezett „Common Language Infrastructure”-t amely lehetővé teszi, hogy a magasabb programozási nyelveken megírt kód futtatható legyen különböző számítógépes felületeken anélkül, hogy a kódot esetlegesen újra kelljen írni az adott architektúrára. Tehát az eltérő nyelvezetű programkódokat átalakítja egy olyan kommunikációs kódra, amit minden rendszer egyöntetűen megért [22].

2.1.1 **Előkészítés**

Jelenleg kétféle implementálási lehetőségünk van. Használhatjuk a GitHubról letölthető nyílt forráskódú verziót, ami .NET 5 és .NET 3.1-en fut. Feltétele még, hogy legalább 2019-es Visual Studio-val rendelkezzünk.

Másik opció a .NET Framework 4-es implementáció, ami a 2019-es Visual Studio mellett a 2017-es is támogatja [22].

2.1.2 **Használata**

Új projekt létrehozása során egy úgynevezett „form”-al találkozunk, ez adja a visual designer magját. A form egy vizuális felület, ahol információkat tudunk eljuttatni a felhasználó felé és ennek a fordítottja. Windows Forms applikáció készítése során első lépésként általában vezérlőket helyezünk el a számukra biztosított téren. A keretrendszerünk számos vezérlő típussal rendelkezik, érdemes megemlíteni a címkét, gombot és szöveges dobozt, amelyek a legalapvetőbb és legtöbbet használt alappillérei a felhasználó felületeknek. Minden vezérlő egy adott osztály példánya, ezeket kedvünk szerint formázhatjuk és alakíthatjuk. A drag-and-drop funkció nagyban megkönnyíti a dolgunkat. Az eszköztárban kiválasztjuk a használandó egységet és behúzzuk a felületen számunkra kedvező pozícióba.

Ezek használatát és funkcióit metódusokkal és eseményekkel tudjuk változtatni. Az eljárás nagyon egyszerű. Ha a felhasználó csinál valamit, interakcióba kerül valamelyik, a form-on megtalálható vezérlővel, azzal egy eseményt vált ki. Ezekre az eseményekre az applikáció reagál és azok függvényében folytatja működését. Ilyen eset lehet egy gombra kattintás vagy akár egy szövegdobozba írás is [7,12].

2.2 WINDOWS PRESENTATION FOUNDATION (WPF)

A .NET Foundation 2006-ban publikálta a WPF-et, azaz a Windows Presentation Foundation-t. Ez a .NET 3.0-ás keretrendszer egyik grafikus alrendszere, amellyel Windows asztali alkalmazásokat tudunk készíteni.

Érdekesség a Windows Forms után, hogy ez egy úgynevezett „XAML" alkalmazás-leíró nyelvet használ. A XAML egy XML bázisú nyelv, amivel különböző forrásokat és grafikai elemeket tudunk definiálni és létrehozni. Minden XAML definíció egy assembly objektum példányt reprezentál. Nagyon fontos, hogy elkülöníti a vizuális megjelenítést az üzleti logikától. Fontossága, hogy ezáltal olyan munkafolyamat alakítható ki, ahol egymástól különálló csoport dolgozhat a UI-on és az applikáció funkcionális működésén.

Minden XAML-ben létrehozott elem egy objektum, ami valamilyen típusnak a példánya. Elemekhez és tulajdonságokhoz a már ismert XML formátumot használja. A megjelenítési eszközöket lehetőségünk van a tervezőben létrehozni, illetve formázni a már Windows Forms-ból megismert módon. Ez után a rendszer automatikusan legenerálja nekünk a beállításaink alapján a szükségszerű XAML kódot. Ebből következik, hogy kézzel írott kóddal is képezhetünk elemeket, ezek utána ugyancsak megjelennek számunkra a tervezőben vizuális formátumban.

Valamilyen úton-módon szükségessé fog válni számunkra, hogy a dinamikus módon változó adatok és tulajdonságok a biznisz logikában, szinte módosulás pillanatában a felhasználói felületen is frissüljön. Ezt WinForms-os ismeretek alapján események és eseménykezelők kapcsolatával tudnánk elérni. Ezzel együtt ez a módszer hihetetlenül nagy időveszteséggel járna minden egyes tulajdonsághoz külön eseménykezelőt létrehozni és lekezelni. Ez a probléma orvoslásra került a WPF-ben.

Lehetőségünk van összekapcsolni egy forrás objektumot egy számunkra szimpatikus vezérlővel, ezt hívjuk adatkötésnek. Adatkötésnél több típust különböztetünk meg, amelyek közül a leggyakrabban használtak:

- Az egyirányú kapcsolatnál csak a forrást kötjük a vezérlőhöz. Tehát az objektumunk értéke és viselkedése módosítja a vizuális megjelenését a vezérlőnknek.

- Kétirányú kapcsolatnál míg a forrás módosítja a célvezérlőt, ez fordított esetben is megtörténik. Fontos olyankor amikor a felhasználói viselkedés alapján szeretnénk módosítani a rendszer belső funkcionalitását.

Összesítve a WPF olyan könyvtárak tárháza, amelyek funkcionalitásával képesek vagyunk Windows asztali alkalmazásokat létrehozni, futtatni és szükség szerint karbantartani. Ezeket mind jelentősen modernebb megjelenítési környezetben, sokkal effektív módon [1,16,27].

2.2.1 **Előkészítés**

Kétféle implementálási lehetőségünk van. Használhatjuk a GitHubról letölthető nyílt forráskódú verziót, ami .NET 3.0-n fut. A vizuális tervező előfeltétele, hogy legalább egy 2019-es Visual Studio 16.3-as verziójával rendelkezzünk.

Másik opció a .NET Framework implementáció, ami a 2019-es Visual Studio mellett a 2017-es is támogatja [32].

2.2.2 **Használata**

Projekt létrehozása során egy MainWindow.xaml állománnyal találkozhatunk. Osztott képernyőn jelenik meg számunkra a vizuális tervező, illetve a hozzá tartozó és a formázására alkalmas XAML kódot.

Hasonlóan a Windows Forms-hoz itt is rendelkezésünkre áll egy eszköztár, ahol könnyedén választhatunk az egyes vezérlők közül. Fontos, hogy itt minden egyes irányítási egység létrehozása és módosítás egy XAML kódot von maga után, amit kézzel is szerkeszthetünk. Mögöttes kóduk kinézete XML szintaxist használ, ezeket különböző tulajdonságokkal szerkeszthetjük is alakíthatjuk. Példával élve, elhelyezhetünk egy gombot. A „Content” tulajdonságával a szöveg tartalmát szerkeszthetjük, míg egy „Margin” paranccsal a szegélyét szabhatjuk személyre.

Vezérlők létrehozása és beállításai során, nekikezdhetünk az úgynevezett adatkötésnek. Szeretnénk elérni, hogy az egyes objektumok tulajdonságainak értékei visszaköszönjenek a felhasználói felületen, még pedig olyan gyorsan amennyire csak lehet.
WPF-en belül a System.Windows.Data névtérben található egy Binding nevű osztály. Minden egyes adatkötés során ilyen Binding nevezetű objektum kerül példányosításra. Ez az osztály, mint egy felügyelő, megállás nélkül szinkronizálja a grafikai felületen lévő vezérlő és a hozzá tartozó tulajdonság értékét. Következőek a legfontosabb beállítható tulajdonságai:

- A *Source* maga a forrás objektum, amelynek valamilyen adattagját fogjuk kötni egy vezérlőhöz.

- A *Path a Source* tulajdonsága, amivel szinkronizáljuk a vezérlő értékét. Megtalálható még egy Mode, amely fent említett adatkötési irány kiválasztása. Leggyakrabban használt a OneWay és a TwoWay.
- A *Converter* egy opcionális megoldás, egyes típusok közötti konvertálás között. Például integer bemeneti adat szöveg típusúvá való alakítás.
- Egyéb speciális beállítások is megtalálhatók, ilyen például az *ElementName*. Elképzelhető olyan eset, amikor a felhasználói felületen elhelyezett vezérlők között szeretnénk adatkötést kialakítani. Például egy slider beállítása függjön egy textbox tulajdonságától és ugyanez visszafele.
- Egy másik ilyen speciális eset a *StringFormat*, ami szöveg megjelenítésének a formálása alkalmas. Hasonlóan, mint egy ToString metódus paraméteres formázás beállítása.
- Utolsó pontként, a leggyakrabban használt tulajdonságok közül van még egy említésre méltó speciális fajta, a *ValidationRules*. Ennek funkciója különböző validációs szabályok ellenőrzése. Például a megkapott tulajdonság értékeként egy valós számot várunk, megnézzük, hogy valóban azt kaptuk-e.

Vezérlők tulajdonságában kapcsos zárójelek között hozzuk létre a Binding objektumot [1,40].

3. PLATFORMFÜGGETLEN KERETRENDSZEREK

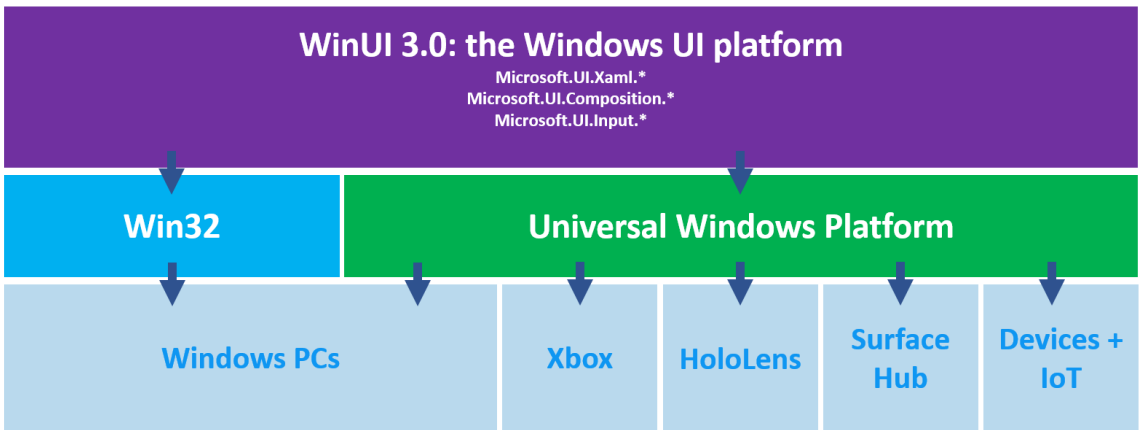
3.1 WINDOWS UI

A Microsoft legújabb UI keretrendszere, amivel UWP és Windows asztali alkalmazásokat is tudunk konstruálni. Mint natív, azaz kizárólag Windows alapú rendszerekre készült framework, letisztult és modern tervezést és kialakítási módokat biztosít számunkra.

Az UWP rövidítés a „Universal Windows Platform” kifejezésre. Microsoft leírása alapján ez egy közös felületet alkot a megírt applikációknak minden olyan készülék számára, ami Windows 10-es operációs rendszerrel kompatibilis. Lényegében az előnyt az jelenti számunkra, hogy elég a programot egyszer megírni, továbbiakban pedig ez futtatható lesz minden Windows 10-et támogató eszköz számára. Ez az készülék lehet akár számítógép, tablet, HoloLens és így tovább.

Jelenleg két aktív verzióval találkozhatunk, ha a WinUI körül érdeklődünk, a 2.0 és a 3.0. Az előbbi UWP applikáció készítésére alkalmazható. Modern és flexibilis vezérlőkkel ellátott keretrendszer, például rendelkezik egy úgynevezett „Navigation View” funkcióval, ami segítségével különböző navigációs füleket tudunk elhelyezni a programunkon belül mind a képernyő tetején, mind a megjelenítő ablak bal oldalán. Továbbá megjelenik a „XAML Islands” fogalma, ami segítségével a fejlesztő új és modernizált WinUI grafikai komponensekkel tudja kiegészíteni a meglévő asztali alkalmazását. A 3.0-val robusztus kiegészítést kapott a Windows UI. Újonnan tartalmazza már a teljes Windows 10 natív UI felületet elkülönítve a sima UWP-s eszközöktől. Gyakorlatban olyan könnyedén és oly módon tudunk Windows asztali alkalmazásokat készíteni, mint a Windows Presentation Foundation-ben vagy a Windows Forms-ban [3,28].

Összesítve a WinUI 3 minden Windows applikációval működik, míg ellenben a 2.0 kizárólag a UWP-vel. Az előbbi közvetlenül használható egy felhasználói felületnek mind az asztali



3.1. ábra WinUI által támogatott platformok [35].

alkalmazásoknál, mind az UWP-s applikációknál. Ha esetlegesen modernizálni szeretnénk a programunk kinézetét, szintén jó választás lehet számunkra a 3.0-ban elérhető letisztult és flexibilis vezérlők tárháza. A XAML Islands segítségével a grafikai felületet vegyíthetjük a WPF, Windows Forms és egyéb hasonló elven működő keretrendszerekkel. A Windows UI elkezdte megalapozni a több felületen működő alkalmazások lehetőségét.

3.1.1 **Előkészítés**

1. Szükségünk lesz először is egy Windows 10 operációs rendszerre, ami legalább 1803 vagy magasabb verziószámú.
2. 2019-es 16.9-es Visual Studio telepítését két „workload" hozzáadása fogja követni. A „.NET Desktop Development", amivel asztali alkalmazásokat tudunk készíteni, ez automatikusan telepíti a .NET 5-ös verzióját. A másik pedig a „Universal Windows Platform development", ami a fejlesztésünk UWP-s részét fogja képezni.
3. Fontos, hogy a NuGet csomagok használata engedélyezve legyen a nuget.org oldalról. Ezek használata fontos és kihagyhatatlan szinten minden projekt készítése során.
4. Tovább szükségünk lesz még Windows UI projekt sablonokra is, ezeket „WinUI 3 Project Templates" néven megtalálhatjuk a marketplace.visualstudio.com oldalon [33].

3.1.2 **Összehasonlítás**

Projekt indítás után első benyomás szerint, mintha egy WPF alkalmazást fejlesztenénk. Első ismerősként a XAML kóddal találkozhatunk, amely a megjelenítés alap építőegysége. Rendelkezésünkre áll egy MainWindow.xaml fájl és a hozzátartozó C# nyelvezetű „code behind" objektum, amelyben könnyedén kezelhetjük a UI elemek funkcionalitását. Szembetűnő lehet, hogy a megszokott egy projekt helyett itt generáláskor rögtön kettőt hoz létre a Visual Studio. Az egyik az általunk elnevezett és létrehozott WinUI applikáció, a másik pedig egy MSIX-be csomagló projekt. Az MSIX egy modern Windows applikáció csomagoló formátum. MSIX-ben összetett alkalmazások megtartják az alap programunk funkcionalitásait, így azok továbbítva ugyanúgy használhatóak más Windows alapú rendszereken [31].

Windows UI **kiemelkedőbb** tényezői**,** a WPF-vel és a Windows Forms-al szemben:

- Az applikáció kinézete letisztultabb, modernebb, az animációk interaktívabbak és jobban kidolgozottabbak. A felhasználót magával ragadja csupán a megjelenítés varázsa.
- A fent említett grafikus alrendszerekkel szemben a WinUI minden Windows 10 operációs rendszert használó eszközzel kompatibilis. Ad egy előnyt, hogy a megírt kódunk több felületen is használható anélkül, hogy új kinézetet kelljen gyártanunk minden asztali géphez, laptophoz, tablethez és így tovább.
- Jobban optimalizált memóriahasználat és teljesítmény. WPF-ben való fejlesztés alapvetően memóriaigényesebb mint egy natív rendszernél. UWP kód .NET Native-ba

fordul le így majdnem olyan magas teljesítményt érünk el mintha egy sima natív applikációt készítettünk volna [33].

Windows UI **elenyészőbb** tulajdonságai a WPF-vel szemben:

- Először is a Windows UI 3.x egy tesztelési és fejlesztési folyamat alatt álló próba rendszer, nem létezik még hivatalosan kiadott teljes verziója. Ismert hiányosságai és hibái vannak, amiken a fejlesztők aktívan dolgoznak. Ezzel szemben a 2006-ban kiadott Windows Presentation Foundation egy sokkal stabilabb rendszernek nevezhető.
- A WinUI-al ellenkezőleg a WPF Windows 10 alatti verziókon is működik és futtatható.
- Bár a WinUI lehetőséget ad, hogy több felületen is használhassuk ugyanazon megírt programunkat, ez önmagában nem elég érv arra, hogy a jelenlegi félkész verzióját priorizáljuk 2020-ban egy új applikáció készítésénél. Sok helyen használnak Windows 10 alapú operációs rendszert, de ez leginkább asztali számítógépek és laptopok esetében a legtekintélyesebb, ahol **ugyanúgy használható a WinForms vagy a WPF i**s.

3.2 AVALONIA UI

Az Avalonia UI egy platformfüggetlen XAML alapú keretrendszer, a .NET Framework, .NET Core és Mono szoftverfejlesztési platformokra. Megjelenésében látszólag a WPF rendszer arculatát örökölte, de érezhető a túlszárnyalási vágy a fejlesztői csapat részéről. Ellenben az Avaloniában írt applikációk átalakíthatóak és futtathatóak Windows operációs rendszeren, MacOS-en, Linuxon. Hozzávetőlegesen tesztelési fázis alatt áll az Androidos és iOS-es fordíthatósága is egyaránt.

Az Avalonia projektek grafikus felületének kialakításáért ugyanúgy egy XAML-szerű, az XML alapú alkalmazás-leíró nyelv felel. Ilyen kódolásban hozzuk létre az egyes vezérlőket, implementáljuk a felület kinézetét és azok viselkedését is. Említésre méltó, hogy a Windows Presentation Foundation-nel ellentétben itt nem áll rendelkezésünkre eszköztár, amivel kényelmesen drag-and-drop funkcióval be tudnánk húzni az egyes elemeket. Ellenben megjelenítő rész itt is megtalálható, ahol a legépelt kódunk vizuálisan is életre kel.
Fontos, hogy a Visual Studio-s Avalonia bővítményének használatánál nem pontosan .xaml hanem .axaml végződésük fájlokkal fogunk találkozni. Dokumentációk alapján arra hivatkoznak, hogy az XML fájlok és funkcióik kötöttek a Microsoft-os szabványhoz, ezáltal nem módosíthatóak és nem felülírhatóak. Megoldásképpen új fájl kiterjesztést kellett konstruálniuk a keretrendszer készítőinek.

Új projekt létrehozásánál választhatunk, hogy egy üres vagy esetlegesen egy MVVM tervezési minta alapján, számunkra előre definiált osztályokkal és mappákkal felvértezett projekttel szeretnénk kezdeni. Az MVVM a UI rendszereknél egy gyakran használt struktúra, amely

elkülöníti és leválasztja a grafikus megjelenési felületet a biznisz logikától. Rendelkezésünkre áll egy előre definiált MVVM felosztású projekt készítési szisztéma. Ezek alapján arra következtethetünk arra, hogy a keretrendszer készítői is azt javasolják, hogy ilyen módon járjunk el alkalmazás készítésünk során [4].

Avalonia specifikus megjelenítési felületekből kettőt különböztetünk meg. A sima „Window" egy teljesen új ablak megnyítására és felépítésére alkalmas. Ezzel szemben a „User Control" egy ablakon belüli ablak konstruálását teszi lehetővé. Megemlítendő, hogy minden platformfüggetlen keretrendszernél szükségszerű kizárólag az adott framework által támogatott megjelenítési felületeket és vezérlőket használni, mivel azok vannak bebiztosítva arra, hogy fordíthatók legyenek más rendszerekre is.

Fontos megjegyezni, hogy hasonló módon, mint a Windows Presentation Foundation esetében, mind a Windows és mind a UserControl rendelkezik saját működést kialakító mögöttes logikai fájlal, ezt hívjuk „code-behind"-nak. Avalonia UI esetében is megtalálható egy „InitializeComponent", ami esetünkben a XAML kód betöltéséért felelős, futási időben.

Elkészült projektünket és függőségeit konzolos parancsok segítségével fordíthatjuk le Windows, Linux és MacOS operációs rendszerekre. Ezek .NET specifikus parancsok, nem az Avalonia sajátossága. A szintaktika, a legfontosabb paraméterekkel a következő:

```
dotnet publish -c [Release/Debug] -f [Framework] -r [Operációs rendszer]
```

A Microsoft a hivatalos oldalán figyelmeztet, hogy minden ilyen jellegű publikálás előtt ellenőrizzük, hogy a konfigurációs állomány tartalmazza-e a megfelelő Runtime Identifier[1]-t. Egy példa egy 64 bites Linux-ra való Release folyamatra:

```
dotnet publish -c Release -f netcoreapp3.1 -r linux-x64
```

Az elkészült állomány a \bin\Release\ mappában lesz megtalálható a projekt könyvtárunkon belül, amennyiben az útvonalat nem módosítjuk a parancsunkban [23].

3.2.1 Előkészítés

Az Avalonia Visual Studio-s bővítménye tartalmazza az alapvető sablonokat, amivel el lehet kezdeni a projekt építést. Másik megközelítésben, .NET Core CLI parancsokkal ugyanúgy elérhető ez az eredmény [4].

3.2.2 Összehasonlítás

Avalonia UI **kiemelkedőbb** tényezői**,** a WPF-el szemben:

[1] Specifikálja a kívánt architektúrát. Milyen OS-re szeretnénk majd publikálni és azt is, hogy hány bites a cél rendszer.

- Alapvetően egy platformfüggetlen frameworkról beszélünk, ami a jelenlegi projektünk fő prioritását élvezi, tehát mindenképpen pozitív tulajdonságként értékeljük ezt. Windows mellett nem csak Linux-ra, hanem MAC OS-re is kiterjeszthető az elkészült alkalmazásunk. Továbbá Android és iOS rendszerekre tesztelési fázisban jár a fejlesztés.
- A fejlesztő csapat nagy hangsúlyt fektet a megjelenítésre, pár részegységen talán a WPF-et is túlszárnyalva. Hatalmas előnyt tud kovácsolni, ha azt szem előtt tartjuk, hogy már csak a WPF vizuális megjelenítése mennyit javult a Windows Forms óta. Ide köthető még az is, hogy bár Triggerek-eket nem tartalmaz az Avalonia UI, ellenben ők az igazi erősséget a CSS stílusozó rendszeren érzik [5].
- A mai napig egy aktívan fejlesztés alatt álló projektről beszélünk. A fejlesztő csapat nyitott mindenféle javaslatra és hibabejelentésre. Ezeket a véleményezéseket internetes blogokon és egyéb oldalakon szerzett információk alapján szerzik be. Fejlesztési prioritásaik erőteljesen ezek irányába orientálódik.

Az Avalonia UI **elenyészőbb** tulajdonságai a WPF-vel szemben:

- Az Avalonia UI Framework használatának legnagyobb hátulütője a dokumentáció és annak hiányossága. Ezáltal nehézséget okozhat kész példák és információ felkutatása az egyes működési elemekkel kapcsolatban. Sok helyen megoldható lehet kombinálni és helyettesíteni ezt a problémát WPF-es dokumentációval, de valójában a két framework alkalmazás-leíró nyelve nem teljesen egyezik meg minden pontban.
- Tapasztalatok alapján érdemes tesztelni minden egyes platformon a kész programot, mielőtt késznek tekintenénk az adott funkciót. Ez egy logikus gondolatmenet, ha egy stabil és működő képes platformfüggetlen applikációt szeretnénk csinálni, ellenben egyértelműen visszaveti a fejlesztési folyamatot időtartam szempontjából.
- XAML-szerű leíró nyelv egy közös pont a vizsgált framework és a WPF között. De, mint említettem sokszor problémát okozhat a két nyelv különbözősége, a WPF sokkal átfogóbb tartalommal rendelkezik. Ez magával hozzá azt a tényt, hogy jelentős WPF-es tapasztalatra lesz szükség bárkinek is, aki Avalonia UI-os applikáció fejlesztésbe szeretne kezdeni, WinForms-os alapokkal bonyolult lehet az AXAML átláthatósága.

3.3 .NET MULTI-PLATFORM APP UI (MAUI)

3.3.1 **Előzmények**

A .NET MAUI története egészen a Xamarin.Forms-ig nyúl vissza. A Xamarin egy Microsoft tulajdonában álló San Francisco központú szoftverfejlesztési vállalat. 2014-ben adták ki a nyílt forráskódú platformfüggetlen keretrendszerüket Xamarin.Forms néven. A framework, amely platformfüggetlensége telefonos operációs rendszerek között érvényesült, ami egészen pontosan az Android, az iOS, illetve aztán a Windows Phone 8.1 volt. Nyilvánvaló

szoftverfejlesztői probléma volt, hogy egy alkalmazás fejlesztése nem használható mindegyik platformon, hisz lényegesen különböznek felépítés és működés szempontjából is. .NET fejlesztőknek itt jött képbe a Xamarin először, ami egy közös nyelvezetet alakított ki a három operációs rendszer között, így a biznisz logikát mindenki ugyanolyan módon tudta kezelni. Ellenben további probléma volt, hogy bár a funkcionalitás értelmezése közös, a grafikai felhasználói felületeket megtervezése ugyanúgy különálló volt, amely kivitelezése nem feltétlenül egy rövid munkálat. Erre szolgált megoldásként 2014-ben a Xamarin.Forms, ami maga a felhasználói felületen alkalmaz közös vezérlőket, közös navigációs opciókat, tehát egy közös réteget melyet mind a három operációs rendszer kezelni tud [6,30].

3.3.2 **MAUI megjelenése**

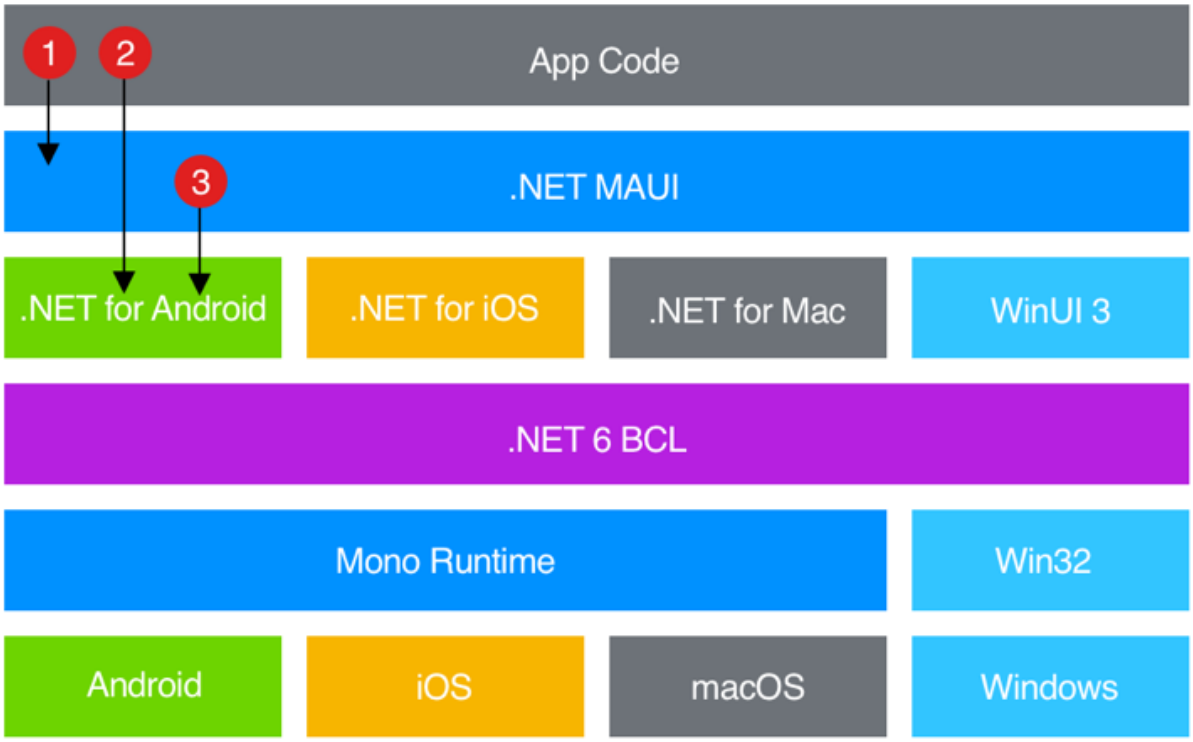
2020-ban a .NET fejlesztői tovább gondolták ezt a felépítést és bejelentették az új Xamarin.Forms-ra épülő keretrendszerüket, a .NET Multi-Platform App UI-t, amely 2021 novemberében érkezett egyszere a .NET 6-tal. Ez lett az a Microsoft által kiadott framework, amely Android és iOS operációs rendszerek mellett már asztali Windows-t és MacOS-t is támogat. Xamarin.Forms-on alapuló keretrendszer úgyszintén XAML leíró nyelvet alkalmaz a grafikai megjelenítéshez, viszont jelentősen sok mindenben kiemelkedőbb és kényelmesebb a használata. Az egyes operációs rendszerek legújabb verzióin elérhető, míg a Xamarin-nál a renderelés erősen egy úgynevezett „BindableObject” objektumhoz osztályhoz kapcsolódik, a MAUI-nál nincs semmi ilyen erős központi megkötés.

Áttértek egy úgynevezett single project rendszerre, ami azt jelentette, hogy míg a Xamarin-es applikáció fejlesztése során a már megszokott Xamarin.Android, Xamarin.IOS projekt elnevezések eltüntek. MAUI-nál már egy projekt létrehozása is elég, minden kompatibilis operációs rendszerrel történő fordítási folyamatot automatikusan végzi a keretrendszer [29].

A .NET MAUI támogatja a hot reload funkció, amely lehetővé teszi, hogy futási időben módosításokat végezzünk a C# és a XAML kódokban egyaránt. Az újonnan elmentett kódbázis újrafordítás nélkül alkalmazásra kerül. A felhasználói felületen végzett módosítások megtartják az adott navigációs állapotot, nem tér vissza a szoftver a kiindulási stádiumba. Tehát ha a program tesztelési fázisa időigényes olyan téren, hogy a használt funkció egymásra épülnek és módosítják például az adattáblákat, XAML kód változtatás során pontosan onnan folytatjuk a munkálatunkat, ahol éppen félbeszakítottuk [30].

3.3.3 **Működése**

.NET 6 számos platformspecifikus keretrendszert szolgáltat alkalmazások készítéséhez, ezek a korábban említett Android, iOS, macOS és egy külön fejezetben tárgyalt Windows UI 3. Mind hozzáfér egy úgynevezett „Base Class Library” könyvtárhoz. Ennek dolga, hogy elszigetelje a mögöttes platform specifikus részleteket a kódbázistól. Ez biztosítja a MAUI erősségét



3.2. ábra .NET MAUI architektúrális felépítése [30].

miszerint a kompatibilis platformok egy és ugyanazon biznisz logikát tudnak alkalmazni és értelmezni, miközben a felhasználói felületet sajátos módon valósíthatják meg.

A 3.3.3. ábra a .NET MAUI architektúrális felépítését igyekszik bemutatni. Az adott fordítási folyamat 3 lépéssel lett leírva, jelen példában a demonstráció Androidon történik.

1) Az általunk megírt kód elsődlegesen a .NET MAUI API-val kommunikál.
2) Szükség esetén a kód közvetlenül kapcsolatba léphet magával a platform API-val.
3) Az első lépésben leírt kapcsolat kialakítását követően a .NET MAUI közvetlen a natív platform API-t használja, jelen esetünkben az Android-ét [30].

3.3.4 **Előkészítés**

Használatához első sorban szükségünk lesz a Visual Studio 2022 (17.1) preview verziójára. A már ismert Workload telepítési ablakon kiválasztjuk a „Mobile Development with .NET” csomagot, ami alapvetően tartalmazza a .NET MAUI 10-es preview verzióját. Ezt követően a keretrendszer használhatóvá válik számunkra [9].

3.3.5 **Összehasonlítás**

MAUI **kiemelkedőbb** tényezői, a WPF-vel, Windows Forms-al és az Avalonia-val szemben:

- Elsődleges pont ismételten a platformfüggetlenség megléte, amelyet se a Windows Forms se a WPF nem támogat. Ellenben az Avalonia UI-al, itt mobil eszközökre is kész megoldást kínál a Xamarin utód.

- Microsoft által kiadott termék. Multinacionális vállalat révén valószínűleg szélesebb terjedelmű és minőségű dokumentáció és gyorsabb problémamegoldás várható, mint egy kisebb cég által finanszírozott Avalonia UI-nál.
- A Xamarin.Forms egy már régebb óta meglévő rendszer, amely hibáit és előnyeit 6 év alatt lényegesen feltérképezték. Ezáltal a MAUI egy olyan stabil alapot kapott, ami a korábbi felhasználó és fejlesztők számára egy biztos pontot tud jelenteni.
- Már a Xamarin.Forms-al is képesek voltunk úgynevezett „Tizen" operációs rendszerre applikációt készíteni, amelyet Samsung eszközök (televíziók, okosórák, IoT termékek) használnak. Ilyen típusú modernizációt és terjeszkedést még nem tudott elérni az Avalonia példának élve [34].

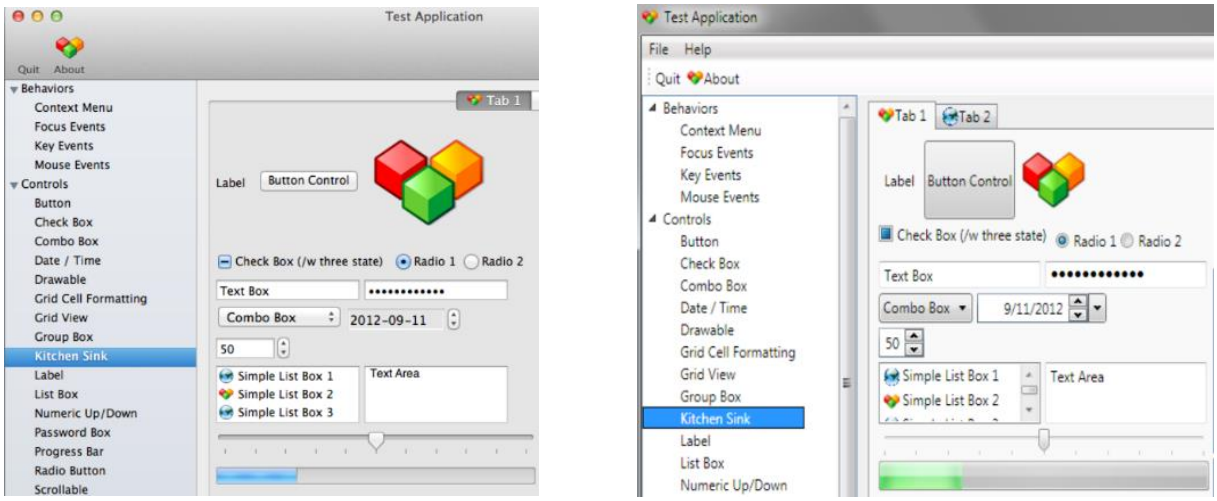
MAUI **negatív** tulajdonságai:

- Jelenleg a MAUI előzetesen kiadott teszt verzióban létezik csak. A Microsoft a hivatalos oldalán hívja fel a figyelmet arra, hogy a termék jelentősen módosulhat még a közeljövőben ezért nem vállal semmi felelősséget a fejlesztési ügyletekben.
- Más platformfüggetlen keretrendszerekkel szemben itt nincs opció Linux-ra való deploy folyamatra.

3.4 ETO.FORMS

2012-ben egy Picoe Solutions Consulting nevű szoftverfejlesztői cég egy akkori lényeges problémát és nehézkes megoldást jelentő kérdésen gondolkozott, a platformfüggetlen projekt készítésen. Már korábban megjelent számos technika arra, hogyan is osszuk meg a már egyszer megírt biznisz logikát, eltérő platformokra szánt keretrendszerekhez. De a legnagyobb probléma ott ütötte fel a fejét, amikor a felhasználói felületet akarták megosztani és használni több helyen.

A cég munkatársai kezdetben a .NET-es GTK# és Windows Form keretrendszereket vizsgálták. Az utóbbi bár Linux és OS X kompatibilitást is ígért, hihetetlenül gyenge minőségű megjelenítés és funkcionalitást mutatott a gyakorlatban. Ehhez képest a GTK# jóval kiemelkedőbb munkát végzett, de a natív működés nem feltétlenül számított az erősségei közé, leginkább OS X terén.



3.3. ábra ETO.Forms-al készített applikációk különböző platformokon [36].

Nyilvánvalóan a legkézenfekvőbb megoldás még mindig az maradt, hogy közös biznisz logikát megvalósító projektet minden egyes platformon új felhasználói felülettel ruházzuk fel. De ez továbbra is rengeteg időt igénylő munkálatnak bizonyult.

Végül 2012-ben a cég kiadásra bocsájtotta a nyílt forráskódú platformfüggetlen keretrendszerét ETO.Forms néven. A keretrendszer kizárólag csak egy absztrakt réteget képez a logikai backend felé, illetve a beépített stílus rendszernek köszönhetően külön kezelhető az alkalmazás azon része, ami platformfüggetlen és ami függő, teljes mértékben egy natív kiszolgálást kaphatunk a keretrendszer jóvoltából. Illetve az utóbbi pontnak köszönhetően minden egyes platformra íródott kód értékei teljes mértékben kihasználhatók.

ETO.Forms által támogatott platformok:

- **Windows:** Windows Forms, illetve Windows Presentation Foundation-t használva.
- **OS X:** működéséhez MonoMac féle .app csomag-ot használ.
- **Linux:** A már korábban említett GTK# keretrendszert használja [36, 37].

A 3.2. ábrán OS X és Windows operációs rendszerekre készített ETO.Forms-os projektek figyelhetők meg. Szemmel látható az egyes rendszerek grafikai megjelenítésének egyediségei, leginkább a gombok, folyamat jelző sáv és a menü pontok kinézetében.
Projekt létrehozás során különböző választási opciók közül tudunk választani, abban a kérdésben, hogy milyen kódolásban szeretnénk a grafikai felületünket elkészíteni. A következő választási lehetőségeink vannak:

- Code: Ebben az esetben nem áll rendelkezésünkre vizuális tervező ablak, kizárólag egy MainForm.cs osztályban kapunk egy kezdetleges C# nyelvezetű kódot, amelyben megtervezhetjük a felhasználói felületünket. Vizuálisan a projekt indítása után tudjuk

megtekinteni elkészült munkákat, miután a framework kiválasztásnál „-windows” parancsot a végére teszünk.

- Xaml: Az ETO.Forms az Avaloniához hasonlóan készített egy hordozható saját xaml alapú állomány, „xeto” néven. Ezen választási opciónál találkozhatunk az eddigi keretrendszereknél ismert vizuális tervezővel. WPF-hez hasonlóan osztott képernyőn látjuk a grafikát és az XML alapú deklaratív leíró nyelv kódját. Észrevehető, hogy az ETO-nál nincs lehetőségünk eszköztár használatára, kizárólag kézzel írott kóddal tudunk vezérlőket deklarálni. A xeto kód nem egyezik meg teljes mértékben a WPF-nél megismerttel.
- Json: Projekt létrehozás után egy MainForm.jeto és MainForm.jeto.cs code behind osztállyal találkozhatunk. A „jeto” kiterjesztésű fájlt megnyitva, WPF-hez hasonló vizuális tervezővel találkozunk, ugyancsak eszköztár hiányában, viszont az osztott képen található grafikus felületet létrehozó kód xaml helyett json nyelven íródott.
- Code Preview: A code után történő kisebb módosítás. Lényegében megkapta a C#-os kóddal készített UI is a vizuális tervezőt. Észrevehető, hogy a MainForm.cs osztály egy a már WinForms-tól megszokott, InitializeComponent metódussal kezdődik, ahogy a grafikai felületet kialakító kód szerepel. Az applikáció megjelenése alapján inkább egy Windows Forms rokon juthat eszünkbe. Bár XAML kódot használhatunk, mint a WPF vagy az Avalonia esetében, de közel sem tud olyan letisztult és modern érzést kínálni, mint a most említett keretrendszerek.

3.4.1 **Előkészítés**

- ETO.Forms használatához legkönnyebben a Visual Studio vagy Xamarin Studio bővítményeivel tudunk hozzákezdeni.
- Visual Studio-ban a bővítmények kezelése fül alatt egy gyors keresés után rátalálhatunk „ETO.Forms Visual Studio Addin” néven.
- Letöltés gomb után az újraindított program VSIX bővítmény telepítő automatikusan előkészíti nekünk a megfelelő beállításokat és ezt követően már elérhetőek lesznek projekt készítésnél az ETO.Forms sablonok [37].

3.4.2 **Összehasonlítás**

Az ETO.Forms **kiemelkedőbb** tényezői a korábban említett keretrendszerekkel szemben:

- A legnagyobb erősségét a natív funkciók elválaszthatósága jelenti. Minden operációs rendszerre készült projekt verzió külön személyre szabható, mint a platformfüggetlen részen.
- Más keretrendszerekkel szemben, az ETO grafikai felületet kódolás terén széles tárházat kínál számunkra. A megszokottól eltérően nem csak XAML, hanem C# és json nyelveken is alkothatunk felhasználói felületet, amely meglétre igencsak kibővíti a fejlesztők lehetőségeit.

- A Windows Forms is képes lehet Linux és OS X környezetben futtatni, de az Picoe Solutions Consulting nagy hangsúlyt fektetett annak elemzésére, hogy ezek a platformfüggetlen próbálkozások gyenge minőségű felhasználói felületet és funkcionalitást eredményeznek. Ezt kiküszöbölve és mivel a WPF kizárólag Windows rendszeren futtatható kijelenthetjük, hogy az ETO.Forms kielégítő platformfüggetlenséget kínál használóinak.
- Mai napig aktívan fejlesztés alatt álló projektről beszélünk. Illetve nyílt forráskódú, így funkciólistáját saját magunk is tudjuk bővíteni.
- 2012-be hozták forgalomba a keretrendszert, ami a platformfüggetlen társai közül nagyon korainak mondható. Kiadástól számított idő szoftverfejlesztésben nagyon fontos, hisz egyre több hiba kerül felszínre használat során, amelyek javításával egyre biztosabb, hogy egy stabil lábakon álló rendszert kapunk kézhez.

Az ETO.Forms **elenyészőbb** tulajdonságai a WPF-vel, Windows Forms-al és Avalonia-val szemben:

- Nem található eszköztár a WPF és a WinForms-al ellentétben. Itt a program felhasználó felé mutatott megjelenítését osztályon belül kézzel létrehozott vezérlők és utasítások sorozatával tudjuk formázni xaml-szerű, C# vagy json nyelven. Más keretrendszerek vizuális tervező drag-and-drop funkciója jelentősen meggyorsította a fejlesztési folyamat grafikai részének tervezését és implementációját. Az ETO.Forms-nál lényegesebben több számolás és odafigyelést fog igényelni egy komplexebb felület megalkotása.
- Grafikája szempontjából erőteljesen Windows Forms jellemvonásokat kapott. Ha olyan projekten dolgozunk, aminek prioritása nem korlátozódik le a funkcionalitás meglétére, hanem az is kizárólagos fontosságú, hogy a felhasználó számára mennyire kecsegtető és kifinomult felhasználói felületet akarunk varázsolni, akkor valószínűsíthetőleg nem ez a megfelelő keretrendszer számunkra. A XAML alapú Windows Presentation Foundation, vagy annak a platformfüggetlen mása, az Avalonia sokkalta inkább tud szolgáltatni számunkra egy letisztultabb, modern hatást keltő kinéztet.

3.5 BLAZOR

A Blazor a WPF és Windows Forms-hoz hasonlóan .NET Foundation kezdeményezés. Célja, hogy web alkalmazásokat tudjuk készíteni a megszokott javascript helyett, C# nyelvezetű felhasználói felülettel. A fejlesztői környezet elérhető Windows, Linux és Mac operációs rendszereken is.
A 2018-ban megjelent Blazor egy úgynevezett „Single Page Application” fejlesztésére alkalmas keretrendszer. Ez azt jelenti, hogy egy olyan weboldalt tudunk készíteni vele, ami dinamikus frissíti és felülírja a megnyitott oldalunkat felhasználói interakciók során. Ez eltérő a megszokott internetes rendszerek tevékenységével szemben, ahol a böngészőnek minden

egyes változtatás után teljesen új oldalt kell megnyitnia. Ennek célja, hogy gyorsítsa és gördülékenyebbé tegye a weblapunkat. Ezzel együtt a weboldal használata során olyan érzést kelthet bennünk mintha egy natív applikációt használnánk [8,20].

4. PLATFORMFÜGGETLEN RENDSZEREK ÉRTÉKELÉSE

Keretrendszer	*Készítők*	*Megjelenés*	*OS kompatibilitás*	*Megjelenítés stílusa*	*Teljes verzió elérhető?*
WinUI 3.0	Microsoft	2020	Windows 10	XAML alapú WPF stílusú letisztult, modern UI	Nem
Avalonia UI	Avalonia	2018	Windows, Linux, MacOS	XAML alapú WPF stílusú letisztult, modern UI	Igen
.NET MAUI	Microsoft	2021 november	Windows, MacOS, iOS, Android	XAML alapú WPF stílusú letisztult, modern UI	Nem
ETO.Forms	Picoe Solutions Consulting	2012	Windows, Linux, MacOS	XAML-szerű, JSON vagy C# alapú, WinForms szerű UI	Igen
Blazor	.NET Foundation	2018	Windows, Linux, MacOS	Gyors, natív érzetet adó weboldal	Igen

4.1. táblázat Platformfüggetlen keretrendszerek tulajdonságai.

A választott projekt keretében egy platformfüggetlen asztali alkalmazás készítése a kitűzött cél. A **Blazor** single page web applikációk készítésére tökéletes, ellenben natív asztali alkalmazás fejlesztésére **nem alkalmas**, így a választás nem is eshet rá.

A **WinUI** 3.0 egy tesztelési és fejlesztési folyamat alatt álló próba rendszer, amivel jelen helyzetben így nem feltétlenül a legkedvezőbb új projekt készítés. Továbbá a kizárólagos Windows 10-es kompatibilitás nem elégséges feltétele egy olyan projektnek, ahol a platformfüggetlenség kihasználása az elsődleges prioritások között szerepel. Ezen problémák által azt mondhatjuk, hogy most a WinUI jelen állapotában **nem alkalmas** a szükséges feladatok elvégzésére.

A Microsoft által fejlesztés alatt álló **Multi-Platform App UI** talán az egyik legígéretesebb kezdeményezés, amely maximálisan kielégíti a platformfüggetlenség fogalmát. Már az elődje is a Xamarin.Forms, egy nagyon népszerű és számos szoftverfejlesztő által preferált platformfüggetlen keretrendszer volt. Mivel a MAUI jelen helyzetben még csak preview

változatban van és erre a Microsoft is nagy hangsúlyt fektet, **nem célszerű egy teszt fázis alatt álló rendszerre** jelentős prioritást élvező **projektet alapozni**.

Választási opciók két legerősebb keretrendszere az **Avalonia UI** és az **ETO.Forms**. Az utóbbi egy 2012-ben publikált fejlesztés, amely a fent említett rendszerek között a legkorábban publikált, ezáltal több mint valószínű, hogy az egyik leginkább bejáratott és tesztelt termék. Az Avalonia projekt bemutatását a felhasználók számára egy barátságos weboldali környezetben kínálják a fejlesztők. Aktív munkálatok és frissítések tömkelege a szoftverfejlesztők bejelentései és kérései alapján történnek, látszólag az Avalonia csapat tagjai a legtöbb fejlesztői oldalon megtalálhatók és aktívan figyelik, hogyan is tehetnék jobbá terméküket. A honlapjukon található oktatási és bemutatási felületet folyamatos rendszerességgel frissítik. Bár ettől függetlenül sok panasz érkezett a dokumentációjuk hiányosságáról és számos esetben ezen problémák megoldása a fejlesztők általi tapasztalatok szájról szájra való közlésén tud csak megoldódni. Másik oldalról a hiányosságot ETO.Forms grafikai felülete jelenti. Míg az Avalonia erősen hajaz a WPF szintű megjelenítésre vagy annak túlszárnyalására, addig vetélytársa inkább WinForms szintű technológiát választott, eszköztár és drag-and-drop funkciók hiányában. A https://dotnet.libhunt.com/ összehasonlító oldala szerint és számos .NET-el kapcsolatos blog felhasználók általi véleményezés erőteljesen azt mutatják, hogy az Avalonia UI megjelenése óta az ETO.Forms népszerűsége jelentősen visszaesett. Platformfüggetlen keretrendszert kereső fejlesztő egyre inkább bízzák rá magukat az Avalonia projektjére, a folyamatos frissítések és javítások hatására. Ezen információk által elmondható, hogy 2020-as évek elején mind a két platformfüggetlen keretrendszer elegendő lehet egy új projekt megkezdésére, de mindent összevetve az **Avalonia UI a legalkalmasabb** a feladat elvégzésére.

5. RENDSZERTERV

5.1 SPECIFIKÁCIÓ

A projekt megtervezése során először definiáltam a kitűzött célt. Ezt követően említésre kerültek a legalapvetőbb és széles körben használt grafikai felhasználói felület készítésére alkalmas rendszerek. A soron következő vizsgálandó terület, a különböző platformfüggetlenséget biztosító keretrendszerek felderítése, tesztelése, analizálása és összehasonlítása. A fejezet lezárásaképp döntéshozatal született a kérdésben, hogy melyik framework rendelkezik elég kvalitással a cél projekt elkészítéséhez.

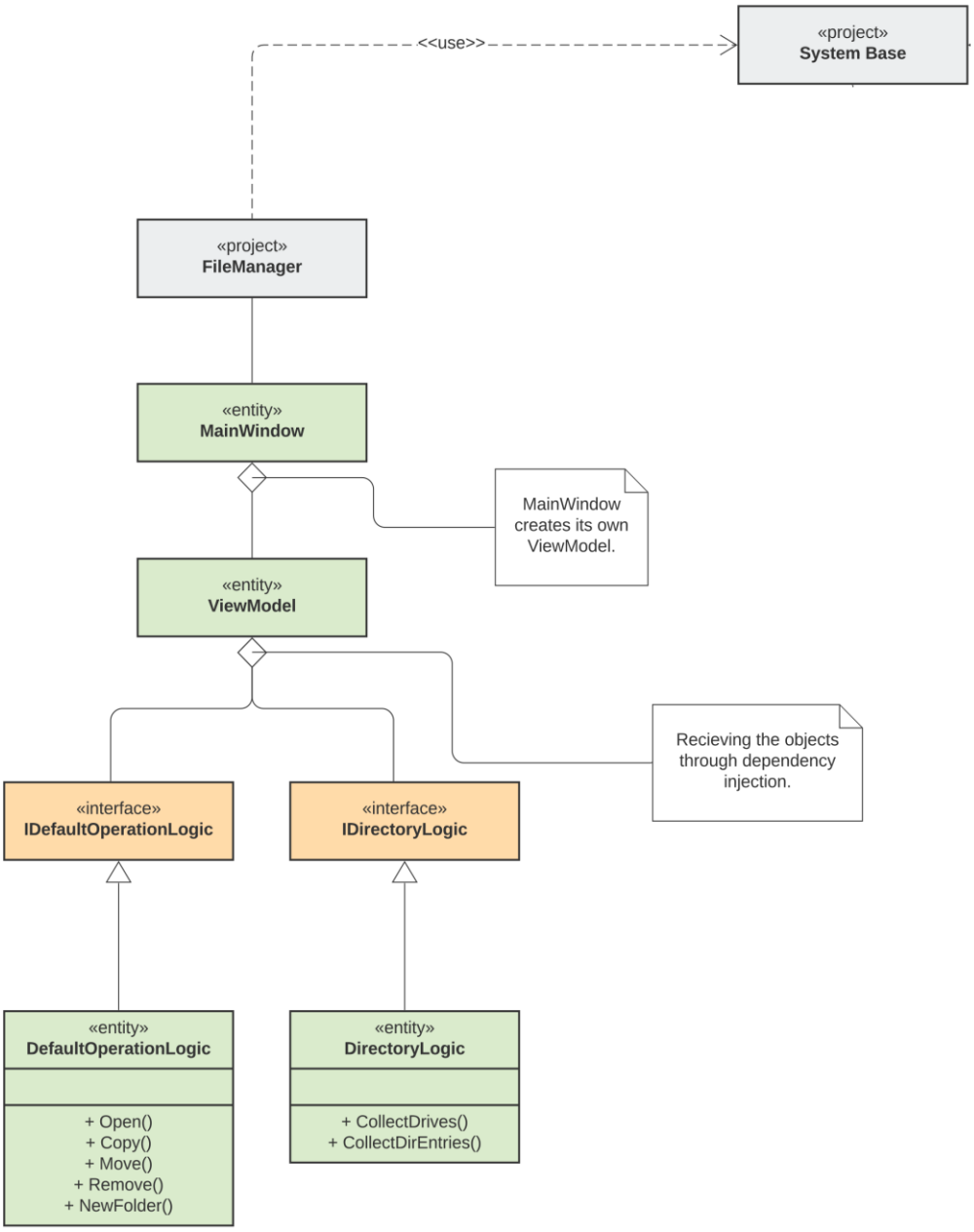
A fájlkezelő rendszer operációs rendszer függetlenségéért az Avalonia UI fog felelni, a korábban említett tulajdonságai végett. Ezáltal a rendszer grafikai felülete axaml kódolásban, míg a háttér folyamatok végrehajtását és a funkcionalitást C# nyelvben fogom biztosítani. A választott integrált fejlesztői környezet a Visual Studio 2019. Mivel az Avalonia publikálási technikája nem mondható egyelőre stabilnak a legújabb .NET 5 keretrendszernél, ezért a .NET Core 3.1 fogja adni a szoftver alapját.

A termék szoftvertani felépítése szempontjából, ahol lehet követve lesz a Gang of Four által készített tervezési mintákat tartalmazó könyv irányelvei. Ezt alapul véve a következő alfejezetekben UML diagrammok reprezentálják majd az elkészítendő tervezet vázát [11].

Az egyes objektumok és entitások kapcsolatrendszerét Model-View-ViewModell minta fogja meghatározni. Választásom azért erre a tervezési mintára esett, mert az Avalonia felépítés tekintetében erőteljesen Windows Presentation Foundation hasonmás. Köztudott, hogy a WPF alapú rendszerek felépítésére erőteljesen az MVVM a legkedvezőbb irányelv a szoftverfejlesztők körében. Az MVVM lehetővé teszi a szoftver számára, hogy a megjelenítésért felelős részegységeket maximálisan elkülönítse és függetlenné tegye a funkcionalitásért felelős végrehajtóktól. Ennek kivitelezése egy középső réteg bevezetésével történik, amit ViewModel néven reprezentálunk. Ez az objektum az, ami fenntartja a kapcsolatot a View és a Model között, elvégzi a szükséges módosításokat, kéréseket anélkül, hogy a két réteg szoros ismeretségbe kerülne egymással. Az MVVM implementálása részletes tervezést és odafigyelést igényel. Ellenben egy elv helyes kialakítás lényegesen leegyszerűsíti a szoftver karbantartását és módosítását. Mivel nem kiemelt, de említésre méltó célunk az is, hogy a projekt alkalmas legyen bővítésre a közeljövőben, akár külső személy/cég által is.

5.2 ÁLTALÁNOS JELLEMZŐK, FELÉPÍTÉS

Megfelelő átláthatóság végett az osztály diagram két részben lesz prezentálva és jellemezve. Első lépésként bemutatásra kerül a fő projekt és felépítése, második felvonásban ismertetve lesz a bővítmény rendszer megvalósítási tervezete.



5.1. ábra A fő projekt felépítése és kapcsolata az összekötő objektummal.

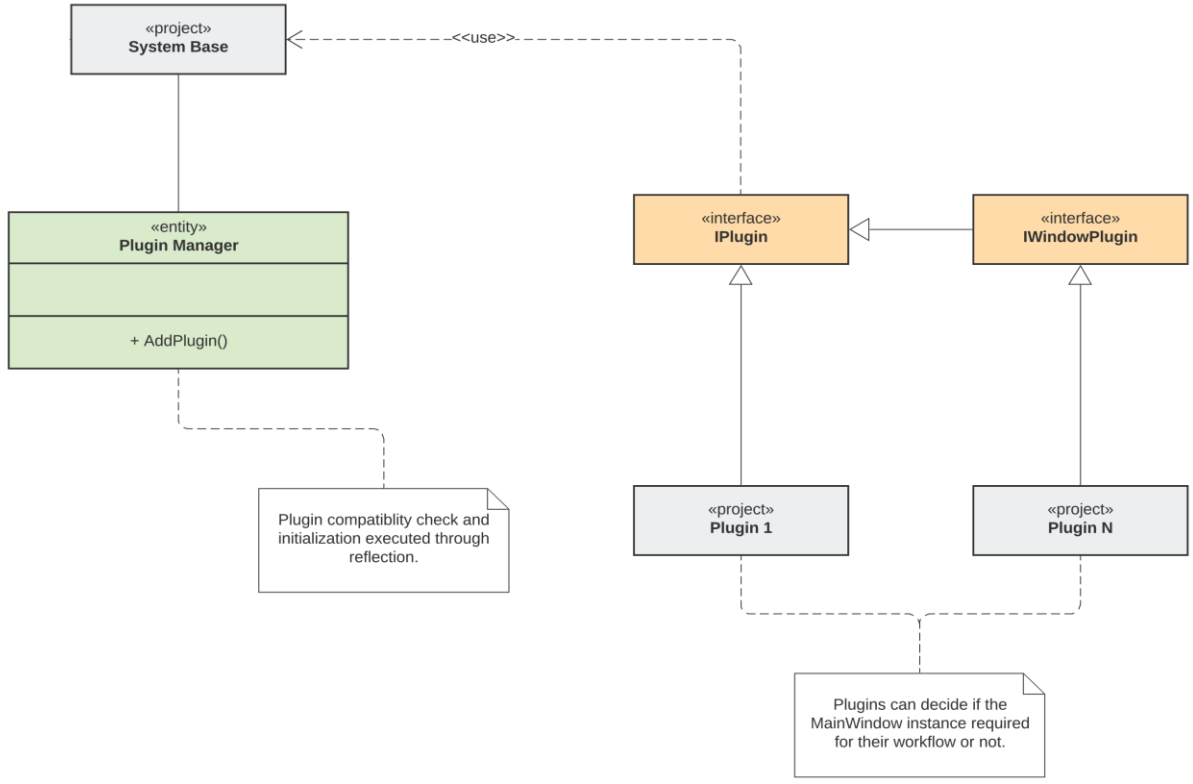
Az elkészítendő szoftver középpontjában egy rendszer összekötő projekt fog állni. Feladata a fő project és a különböző pluginok között egy híd kiépítése, olyan objektumok megvalósítása, ami közös pontként szolgálhat a két építőelem között.

A 5.2.1. ábrán megfigyelhetjük az MVVM értékeinek a tiszteletben tartását. A fő megjelenítési réteg elválasztásra kerül a végrehajtóktól egy ViewModel által. A két ábrázolt funkcionális egység megvalósít külön-külön interface-ket ezzel részlegesen biztosítva a Gang of Four egyik legismertebb követelményét, a Dependency Inversion-t. A végrehajtók egyik felelősségköre az 5 fájlkezelő rendszerek ismérvét képviselő operációk implementálása és végrehajtása. A másik objektum feladata minden más a UI-on végrehajtandó felhasználói interakció véghezvitele.

A másik általunk specifikált követelmény a bővítménykezelés. Az 5.2. ábra bal felső sarkában kiemelten szerepel a rendszerösszekötő projekt, aminek elsőszámú feladata már korábban részletezésre került. Viszont tartalmaz egy további kritikus fontosságú elemet, a Plugin Manager-t.

A Plugin Manager egy olyan objektum, aminek felelősségkörébe tartozik a beérkező pluginok, tehát DLL fájlok kezelése. Reflexió segítségével kerül kinyerésre az adott DLL által

tartalmazott objektumok és azok által megvalósított interface-ek. Kompatibilitás vizsgálat történik, hogy a beérkező objektum tartalmaz-e a szoftverrel összeillő részeket. Továbbá a



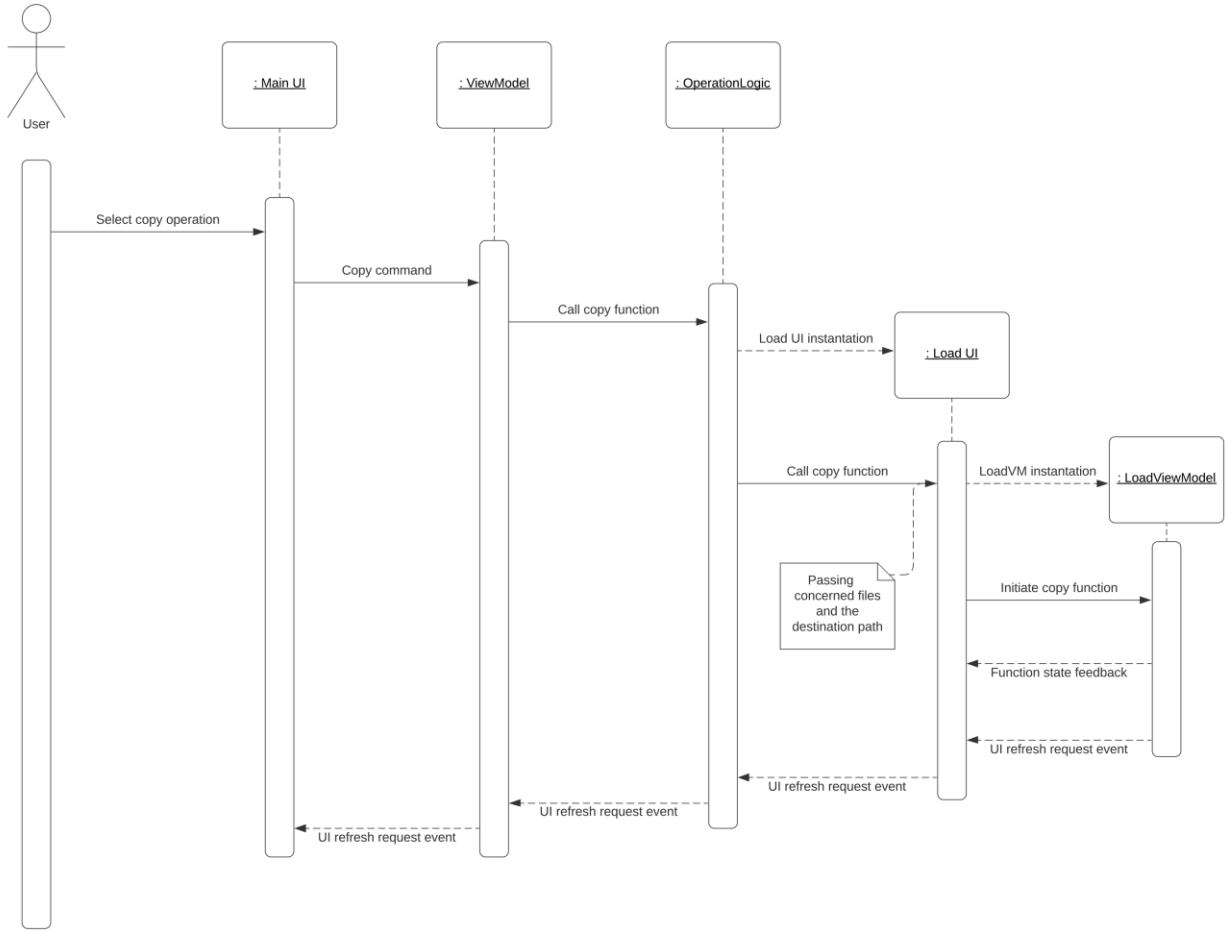
5.2. ábra Plugin kezelő rendszer felépítése.

rendszernek tudnia kell, hogy a beérkező bővítmény kizárólag belső működést módosít csak, vagy szüksége van hozzáférésre a megjelenítési felülethez is.

Amennyiben a kijelölt DLL megfelelt a kompatibilitási vizsgálatnak, szintén reflexiós lépésekben lekérjük a bővítmény nevét, visszajelzünk a felhasználónak a művelet sikerességéről, ezt követően pedig végrehajtuk a plugin inicializáló metódusosát, ezzel bővítve a szoftver funkcionális tárházát.

5.3 ALAPVETŐ FUNKCIÓK RENDSZERTERVE

Egy fájlkezelő esetében a legfrekvrentáltabban használt operációk a fájlok törlése, másolása, áthelyezése, megnyitása, illetve a mappa létrehozás. Ezen opciók megléte biztosítva lesz ebben az Avalonia alapú rendszerben is. Felépítésük erőteljesen hasonló, demonstrálásuk és bemutatásuk a másolási folyamaton keresztül fog történni.



5.3. ábra Másolási funkciót bemutató folyamatábra.

A következő szekvenciális diagram egy felhasználó által kiválasztott fájlok másolásának folyamatát írja le. Kérését a grafikus felhasználói felületen hajtja végre. Ezt követően az MVVM tervezési mintát tiszteletben tartva a kérés egy command-ként ütemeződik tovább a

ViewModel irányába. A VM tisztában van a lehetőségeivel és az alatta lévő végrehajtók által implementált funkciók tárházával, meghívja a szükséges, jelen esetünkben a másolási metódust.

A jelen esetünkben OperationLogic-ként elkeresztelt objektumnak két feladata van. Egyrészt, hogy elvégezze a másolási műveletet, másrészt a procedúra állapotáról folyamatos visszajelzés biztosítása a felhasználó számára. Az utóbbi feladat megvalósítására szükség lesz még egy megjelenítési rétegre, ami kizárólag állapotjelzési célokra definiált. A ViewModel-Model kapcsolat fenntartására adatkötés lesz konfigurálva. A visszajelzésre szolgáló ablak ViewModel-jének a feladata a másolás végrehajtás és állapot visszajelzése párhuzamosítva úgy, hogy a felhasználó ne észleljen befagyást/lassulást a grafikus felületén, az esetlegesen nagyobb időigényű művelet végrehajtása során sem.

A rendszertervben kijelölt megoldási módok és architektúrák több tényező szerint lettek kiválasztva, priorizálva. A választásokat és indoklásukat az 5.4. táblázat foglalja össze.

	Választott megoldás	**Választás indoklása**
Keretrendszer	Avalonia UI	Platformfüggetlen, modern UI kinézet, xaml-szerű alap, naprakész rendszer.
IDE	Visual Studio 2019	C#-hoz és grafikai felhasználói felület tervezéséhez a piacon az egyik legjobb fejlesztői környezet.
.NET verzió	.NET Core 3.1	Avalonia UI deploy ezen a .NET verzión biztosan stabilan és megfelelően kivitelezhető.
Tervezési minta	Model-View-ViewModel (MVVM)	GUI-val rendelkező alkalmazásoknál legelterjedtebb tervezési minta. Flexibilitás és könnyebb hibajavítás biztosítása.
Módszertan	GoF módszerek, tervezési minták	A tervezési minták működésére és fontosságára először a Gang of Four írói hívták fel a figyelmet. Mai napig aktívan használt technikák.

5.4. táblázat Választott rendszerek / megoldások összegzése.

6. IMPLEMENTÁCIÓ

Következő alfejezetek sorában tárgyalásra kerül a platformfüggetlen fájlrendszer megvalósításának egységei. A fejezet egyrészt egy olyan madártávlatból közeletíti meg az implementálási folyamatot, aminek tanulmányozását követően egy külső olvasó számára érthető és elkészíthetővé válik számára egy hasonló rendszer, ami tartalmazza a legfontosabb irányvonalakat. A leírásban tárgyalásra kerülnek nehézségek, egyedi megoldások és egyéb tapasztalatok is.

6.1 FEJLESZTŐI KÖRNYEZET MEGVALÓSÍTÁSA

Integrált fejlesztői környezetként Visual Studio 2019 került kiválasztásra. Ahogy az Avalonia tárgyalásnál a 3.2.1. fejezetben említésre került, egy VS[2] specifikus bővítmény telepítésével használhatóvá válik számunkra a keretrendszer.

A később deployolásra szánt projektek .csproj állományában definiálnunk kell az RID-kat, azaz a *Runtime Identifier* célpontokat. A platformfüggetlenség végeztében kell ezt megadnunk, ezek lesznek a platformok, amelyeket megcélzunk a programunkkal. Ezeket legfőképpen a .NET által használt NuGet csomagok platformspecifikus megjelenítésének segítésére szolgálnak. 64 bites rendszereket prioritizáltam, ezáltal az én RID gyüjteményem a következő képpen néz ki [19]:

```
<RuntimeIdentifiers>linux-x64;win-x64;osx-x64</RuntimeIdentifiers>
```

A 6.2. fejezetben a különböző projektek felépítése és azok részegységei kerülnek feltérképzésre.

6.2 FŐKÉPERNYŐ FELÉPÍTÉSE

A program felépítését tekintve egy single page applikációként centralizálódik. Ez azt jelenti, hogy a szoftver futtatása során egy adott fő ablakon jelenik meg majdnem minden információ és kezelési elem a használat közben. További célkitűzés egy responsive kialakítás, tehát ablak méretének megváltozására a megjelenítési eszközök alkalmazkodjanak megfelelően.

Középpontban egy két paneles megoldás szükséges, ahol a fájlrendszer megjelenítése történik. A panelek segítségével a felhasználó párhuzamosan két lokáción ejtheti meg a keresését és hajthatja végre a műveleteket. Lényege, hogy jelentősebb átláthatóságot biztosít az egyén számára és ezzel együtt az egyszerre több fájlrendszeri pozíciót érintő műveletek (másolás, áthelyezés) könnyebben reprezentálhatók a vizuális térben. Fontos támpont még a mappák és a fájlok elkülönítése szintén valamilyen vizuális elem segítségével. Ez lehet különböző szín vagy esteleg ikon használat az adott típustól függően.

[2] Visual Studio

Megjelenítési mód	Tulajdonság
Brief view	Kizárólag a fájlnév kerül megjelenítésre.
Full view	A név, kiterjesztés, fájlméret és a dátum jelenítődik meg.
Tree view	Mappa tartalmának megnyitása lépcsőszerű megjelenítéssel történik.
Custom view	Személyre szabhatók a megjelenítendő fájl tulajdonságok.

6.1. táblázat Megjelenítési módok ismeretetése.

Mind a két panelhoz egyénileg hozzárendel legördülő menü megléte, a meghajtók kiválasztására úgyszintén alapvető fontosságú. A fájlkezelőnek tisztában kell lennie, hogy milyen meghajtókat használ a számítógép operációs rendszere.

A fájlkezelő támogat különböző megjelenítési módokat, név szerint *brief* (rövid/részleges), *full* (teljes), *tree* (fa), illetve *custom* (egyéni) változatok közül határozhat a felhasználó. Ezek részletezése a 6.1. táblázatban ismertetésre kerül.

Továbbiakban szükség lesz a 6 alapvető funkció használatára specializálódott gombok létrehozására. Ezek a már korábban is említett *megnyitás, másolás, áthelyezés, új mappa lérehozás* és *törlés* operációk összessége. Gombok mellé barátságosabb megjelenítést keltve érdemes ikonokat rendelni. Mivel a szoftver támogatja a gyorsgombok (hotkey) használatát ezért érdemes ezeket a kombinációkat is feljegyezni a funkciók neve mellé.

Utolsó eleme a főképernyőnek egy felület, ahol a kívánt plugin fájlok betallózásra kerülhessenek.

6.3 A FŐ PROGRAM FUNKCIONÁLIS FELÉPÍTÉSE

A következőkben a tárgyalt vezérlő egységek mögöttes tartalmába és működési elveibe kaphatunk rálátást.

6.3.1 **Az első találkozás AXAML kóddal**

Az első lépést a panelok megalkotása jelentette. Egy új Avalonia használó számára már itt megtapasztalhatja a *xaml* és a keretrendszer által szolgáltatott *axaml* kódok közötti különbségeket. Aki nem rendelkezik erős, akár több éves rutinnal WPF alkalmazások készítésében, számára eleinte nehéz lehet a kiindulás az UI tervezése során. Mint ennek a csoportnak egy képviselője hasonló, problémákkal találkoztam. Számos a már WPF-ből ismert kényelmi funkció hiányossága komoly fejtőrést okozott. Részemről megoldásként szolgált, hogy kezdetben felhasználói felület elemeit WPF-ben először megterveztem és leimplementáltam, ezt követően pedig átvittem az Avaloniás fejlesztői környezetembe.

A keretrendszerről azt érdemes tudni, hogy az IntelliSense[3] sokszor kihagy, nem segít, vagy esetenként épp, hogy olyan opciókat jelenít meg amik valójában nem relevánsak. Hasonló probléma volt, amikor a megjelenítési felület nem várt hiba miatt eltűnt. Amennyiben az előző problémával egyszerre lép ez a hiba, bátran mondhatjuk, hogy a fejlesztő teljesen vakon gépel, a kódjának hitelességéről kizárólag a fordítás után tud csak megbizonyosodni. Érdemes megjegyezni, hogy egy Avalonia projekt, még natív rendszerre való fordítása is igencsak hardver igényes. Ezáltal elmondhatjuk, hogy egy gyenge-, vagy középkategóriás számítógép nem feltétlenül küzd meg vele olyan sebességgel, mint mondjuk egy Windows Forms vagy mint egy WPF projekttel.

További hiányosságnak éreztem azt, hogy mivel az Avalonia is támogatja azt a funkciót, hogy fordítási kérelem nélkül is használható néhány UI opció tervezési időben, így a WPF-nél megszokottan egy UI elemre rányomva nem a létrehozási kódra pozicionál minket, ezáltal valamennyivel lassítja egy nagyobb méretű axaml fájlnál a megfelelő részegységek megtalálását. Bár az Avalonia módszere úgyszintén hasznosnak tűnik, valójában számomra feleslegesnek érződött, egyszer sem használtam.

6.3.2 **Fájlrendszer megjelenítésére szolgáló panelek**

A fájlok rendszerezett megjelenítésére a már WPF-ből ismert és itt is implementált *ListBox* került kiválasztásra, mint használandó egység. Működése szinte teljesen ugyanaz, mint az előbb említett keretrendszernél, fontos megjegyezni, hogy az Avalonia szinte mindenhol, ahol tudott szinoníma kifejezéseket használt az egyes beépített funkciók és események elnevezésénél.

Maga a fájlok megjelenítéséhez erősen ajánlott DataTemplate-t használni, ahol elkülönítésre kerül a fájnév és az esetleg mappa ikon, amennyiben szükséges a megléte. Kelleni fog továbbá egy fejléc is, ami arra szolgál, hogy közölje a felhasználó számára, hogy az adott oszlopban milyen információ jelenik meg az aktuális fájllal kapcsolatban. Ennek implementálására ControlTemplate a javasolt. Elkészítése közben érdekes hibával találkoztam. Mikor kettőnél több SharedSizeGroup[4]-ot alkalmaztam, ItemsPresenter[5] nélkül, a program futása során egy véletlenszerű nem várt pillanatban lefagyott. Ez egy olyan probléma volt, amelyet WPF-ben való kipróbálás során nem volt jelen. Fontos még megjegyezni, hogy a template-ek használatánál amennyiben a fejlesztő úgy dönt, hogy konkrét adattípust is szeretne definiálni hozzá, fantom hibával találkozhat. Bármilyen típust is ír az adott sablonhoz, a fordító hibát fog jelezni, olyan hibát, ami valójában nem áll fent. Hasonló problémák voltak WPF-nél is, viszont ott egy újrafordítás megoldásként szolgált, itt nem ez a helyzet.

[3] A fejlesztői környezet által szolgáltatott általános kifejezés segítő eszköz, ami segít többek között a kódkiegészítésben, gyors információk megjelenítésében és még sok egyéb adat megjelenítésében [25].
[4] Olyan tulajdonság, amely segít a sorok és oszlopok részegységeit úgy tudjuk megosztani, hogy maga a méretezés konzisztens maradjon [24].
[5] Egyfajta jelzőérték, ami azt mondja meg, hogy az elem vezérlőben definiált adatok hova és miként kerüljenek be a kódon belül [26].

6.3.3 **Megjelenítési módok**

Következő kihívásként szolgált az egyes megjelenítési módok implementálása. Fontos megjegyezni, hogy amíg a részleges és a teljes megjelenítésnél, használhatunk statikus, előre megírt sablonokat, amik igencsak megkönnyítik a munkákat. Ellenben a fa megjelenítés és az egyéni választásnál, igencsak megdolgoztat minket ez a keretrendszer.

A **fa szintű megjelenítéshez** látszólag a *TreeView* nevezetű vezérlő használata tűnik elsőnek a legcélravezetőbbnek. Aki már dolgozott WPF-en vele tudja, hogy a dinamikus megvalósítása ugyanolyan egyszerű, mint a statikus, sőt véleményem szerint könnyebb is. Ellenben Avaloniában ez közel sem úgy működik, ahogy az előbb említett keretrendszerben. Mivel itt nem rendelkezik, vagy nem publikus a vezérlőnél csomópontok tárolására alkalmas lista, ezáltal a dinamikus megoldás is jelentősen problémássá vált. Két teljes nap keresés és próbálgatás után, a ListBoxos megoldásnál maradtam, sokkal egyszerűbb volt megkerülni a problémát és saját megoldással szolgálni.

Az egyes fájlok/mappák példányának adatai között felvételre került egy *szint* nevezetű tulajdonság. Ennek segítségével azonosítható be, hogy az adott objektum milyen fájl hálózaton keresztül érhető el, ha úgy tetszik, hány mappán keresztül ágazik az útvonal hozzá. Továbbiakban minden egyes fájl a kiíratásnál egy Grid-en helyezkedik el, ami fel van osztva annyi részre ahány mappán keresztül vezet az út az adott objektumhoz. Ezt követően az első helyre a mappa ikon kerül (amennyiben szükséges), az utolsó pozícióba pedig maga az elnevezés. Ezáltal minden egyes dupla kattintás vagy a megnyitás gomb lenyomása egy fa szintű megjelenítést von maga után.

Szintén dinamikus megoldás kigondolását követelte meg az egyénileg választható tulajdonságok alapján történő listázás, a **custom view** kialakítása is. Itt kerültünk bele abba az állapotba, amikor mind a DataTemplate-et és mind a ControlTemplate-et kódból kellett létrehozni. Nem meglepő, hogy hasonlóan a tree view-nál, itt sem úgy épült fel a sablon létrehozása kódból, mint a WPF-nél. Ellenben nem hozott nagy segítséget a tény, hogy az Avalonia hivatalos oldalán lévő példa, egy dinamikus kódból létrehozható Template felépítésére, nem működött. Hosszas keresés és felhasználók alapján publikált kódok, kérdések és válaszok segítségével sikerült egy látszólag működő megoldáshoz jutnom. Ezen a ponton olyan problémába ütköztem, hogy az ItemTemplate kódból való létrehozása egyáltalán nem működött, csak abban az esetben amikor axaml-ból történt a meghívás. Mivel ehhez nem találtam sehol segítséget, kénytelen voltam az Avalonia csapatához fordulni. Pozitív csalódásként ért, hogy pár nap alatt, mondhatni gyorsan kaptam választ tőlük. Ellenkező esetben negatív tényként értékelném azt, hogy a megoldást, amit kaptam, az interneten szinte sehol máshol nem fellelhető.

Fontos kijelenti, hogy mivel a szoftver középpontjában a megjelenítési panelek állnak, ez magával vonza azt a tényt is, hogy maga a megjelenítési módok is hasonló prioritást élveznek. Értve ezalatt azt, hogy minden egyes funkció megírásánál figyelni kell arra, hogy hogyan implementálható mind a 4 megjelenítési mód szeszélyei szerint.

6.3.4 **Alap funkciók implementálása**

Az alap funkciók elkészítésénél az MVVM használata elsődleges prioritású volt. Maga a főképernyő minden egyes beérkező felhasználói kérést a saját ViewModel-je számára továbbítja, a munkavégzésért pedig specializált biznisz logikai entitások feleltek meg. Fontos kijelenteni, hogy a két panel bár ugyanazon operációk végrehajtására alkalmasak, külön ViewModel-t implementálnak a párhuzamos munkavégzés elősegítése végett.

A másolás, törlés és áthelyezés műveletek konstans visszajelzést követelnek a felhasználó számára. Ennek megoldásaként a lánc a főképernyőtől meghosszabodott, mivel egy mellékes, visszajelzésre alkalmas ablak számára kerülnek továbbításra az elvégzendő funkció operandusai. Továbbiakban a főképernyő és részegységei hátradőlhetnek és az ellátásra szoruló feladataik leszűkülnek arra, hogy teljesen egyszerűen megvárják a másodlagos ablak által elvégzett művelet visszatérési értékét.

Míg a másolás művelet kapott egy drag and drop[6] kiegészítő kényelmi lehetőséget, minden egyes funkció elérhető egy gyorsgomb lenyomásának segítségével egyaránt.

Fontos megjegyezni, hogy jelentős problémákat okozott egyrészt a korábban említett keretrendszerekkel szemben a beépített események jelentős hiányossága. További nehezségnek számít az, hogy mint a 6.3.3. fejezet végén említésre került, minden funkciónak működnie kell minden megjelenítési módon. Ezeknek az implementálása viszont jelentősen eltér, olyan szinten, hogy valamelyik teljes mértékben dinamikus és kódból épül fel, míg a másik statikusan axaml támogatással került elkészítésre. Ezáltal több alkalommal nem lehetett egyértelmű és általános megoldást adni például arra, hogy mikor kérvényezzük egy egyik panel megjelenítésének frissítését.

6.3.5 **Plugin hozzáadás**

Utolsó pontként pedig szüksége lesz a plugin hozzáadásért felelős gomb beüzemelésére. Tallózási opcióként dll fájlokra szűrűnk. A főprogram felelőssége lekorlátozódik a fájl lokáció megállapítására, minden más munkálatot és összeköttetést a Plugin Manager végez, amelynek leírása a következő fejezetben tárgyalódik.

[6] Az egér középső gombját lenyomva tartva egy fájl felett és áthúzva a másik panel megfelelő pozíciójára, automatikusan elindul a másolási művelet.

6.4 ÖSSZEKÖTTETÉSÉRT FELELŐS PROJEKT FELÉPÍTÉSE

A fejlesztés során szükségessé vált egy új projekt létrehozása, amely egyfajta hídként szolgál a főprogram és a beérkező bővítmények között.

6.4.1 Közös építőegységek

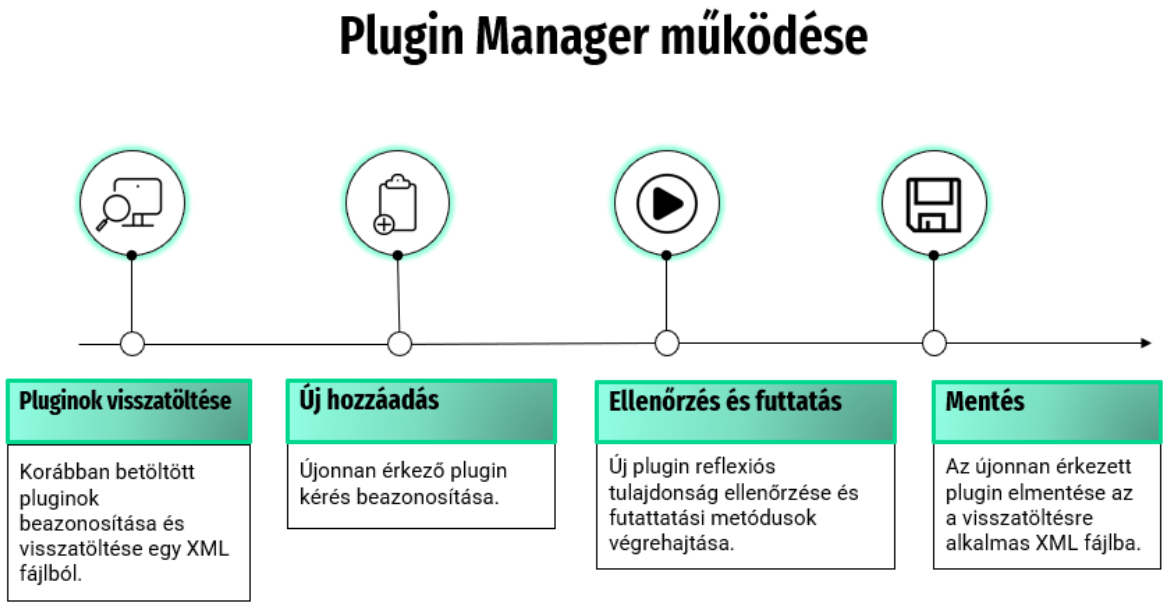
A köztes projekt létrehozásának egyik legfontosabb oka, a közös részegységek használata. Példával élve, az objektum-orientált programozást követve, minden egyes éppen aktuálisan használandó fájl külön objektumként van leképezve. Minden egyes fájl objektumnak saját adattagjai vannak. Ezek reprezentálnak minden olyan tulajdonságot, amely szükségszerű mind a főprogram, mind várhatóan az adott plugin funkcionális szükségleteinek ellátására.

További fontos objektumok az egyes interfészek és absztrakt osztályok megléte és előre definiálása. Ezekre azért van szükség, hogy a főprogram részei olyan entitásakból származzanak le, amelyek mindenképpen megvalósítanak olyan tulajdonságokat, amelyekkel később a bővítmények operálni tudnak majd.

Úgyszintén ezen projekt keretei között kerül létrehozása a pluginok által megvalósított interfészek véges számú objektum listája. Ezek biztosítják azt, hogy az adott bővítmény alkalmaható és ellenőrizhető legyen a Plugin Manager számára. Ez a következő, a 6.4.2. fejezetben kerül tárgyalásra.

6.4.2 Plugin kezelés

Az összeköttetésért felelős Plugin Manager működését a következő ábra foglalja össze:



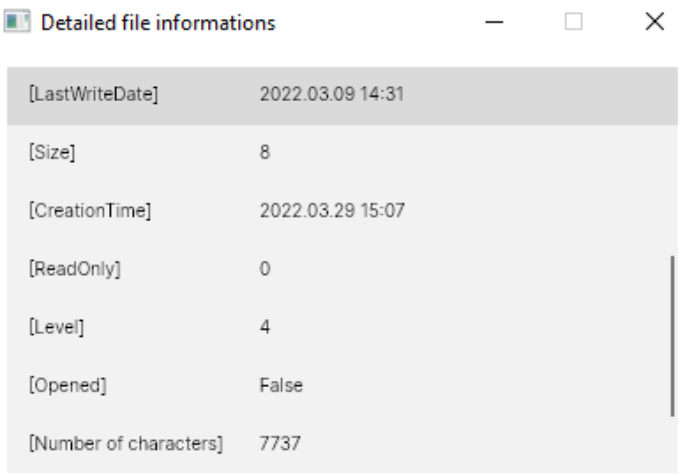
6.2. ábra Plugin Manager felépítést összefoglaló ábra.

Mint látható a plugin rendszer egy XML tároló társulatával működik együtt. Ennek célja a már korábban használatra kitűzött bővítmények, a következő futtatásra való visszatöltése. A beérkező új plugin reflexió segítségével ellenőrzésre kerül, miszerint megfelelően megvalósítja-e a szükséges interfészeket ahhoz, hogy egy kompatibilis bővítményként titulálhassuk. Minden egyes plugin tartalmaz egy futtatási parancsot, amely automatikusan elvégzi a beépítést a főprogramba, ezt a plugin manager indítja el automatikusan. Végső folyamat az új bővítmény lokációjának elmentése a már korábban tárgyalt XML dokumentumba.

6.5 FÁJL INFORMÁCIÓ MEGJELENÍTÉS, PLUGIN FORMÁJÁBAN

Az egyes pluginok tervezése és elkészítése példaként és bizonyítékként szolgál az elkészített projekt bővíthetőségének meglétéről. Az első ilyen jellegű bővítmény olyan formában egészíti ki a szoftver funkcionális tárházát, hogy egy gyorsgomb lenyomásával részletesebb információt kaphatunk az általunk kiválasztott fájl tulajdonságairól.

Fontos kitétel, hogy mint minden bővítmény, tehát ez sem egy már meglévő kódot „ébreszt fel” a főprogramból, hanem egyénileg implementálja az általa generált kódbázist, csatolva a fő projekthez. Az Avalonia a Windows Forms és a WPF-el ellentétben a dialógus megjelenítéshez kérvényezi az alatta lévő, eddig aktívnak titulált ablak referenciáját. Ennek megoldásaként a plugin egy olyan interfészt valósít meg, ami bár az alapvető pluginoknak definiált „ős interfészből” származik le, de a futattási metódusa egy főképernyő referenciát vár paramétereként. Ezzel együtt hozzáférhet a szoftver által használt megjelenítési modellhez, így beazonosítva az éppen kiválasztott fájlt.



6.3. kép A fájl információ megjelenítésére szolgáló bővítmény megjelenése.

6.6 RUTIN MEGFIGYELŐ RENDSZER, PLUGIN FORMÁJÁBAN

6.6.1 Célmeghatározás

A bővítmény a fájlkezelő rendszeres használatán alapul. A háttérben megfigyeli a felhasználó által végzett fájl választási rutint és belső kalkulációk segítségével, kellő információ megszerezése során becslést és ajánlást ad a felhasználónak, hogy hogyan haladjon tovább.

Egyszerű példa egy számlázással foglalkozó ügyintéző mindennapi munkájának elősegítéséhez. Tételezzük fel azt, hogy a munkávallalónk pénzügyi adatokat visz be, ehhez Excelt használ. Egy huzamosabb ideig mindig az adott hónapra vonatkozó Excel fájlt nyitja meg, de ez előtt szinte mindig egy levelezőprogramot indít el minden egyes munkanap reggelén. Az eseményt rutinként kezelhetjük, ezt pedig a plugin észleli. Egy bizonyos idő után tudni fogja, hogy jelentős valószínűségű az tény, hogy a levelező szoftver után az adott excel fájl fog következik. Ennek elősegítése képpen egy gyors elérést biztosít a felhasználó számára, akinek immáron nem kell kikeresnie a fájl lista közül a pénzügyi adatokat tartalmazó fájlt.

6.6.2 Markov-láncok

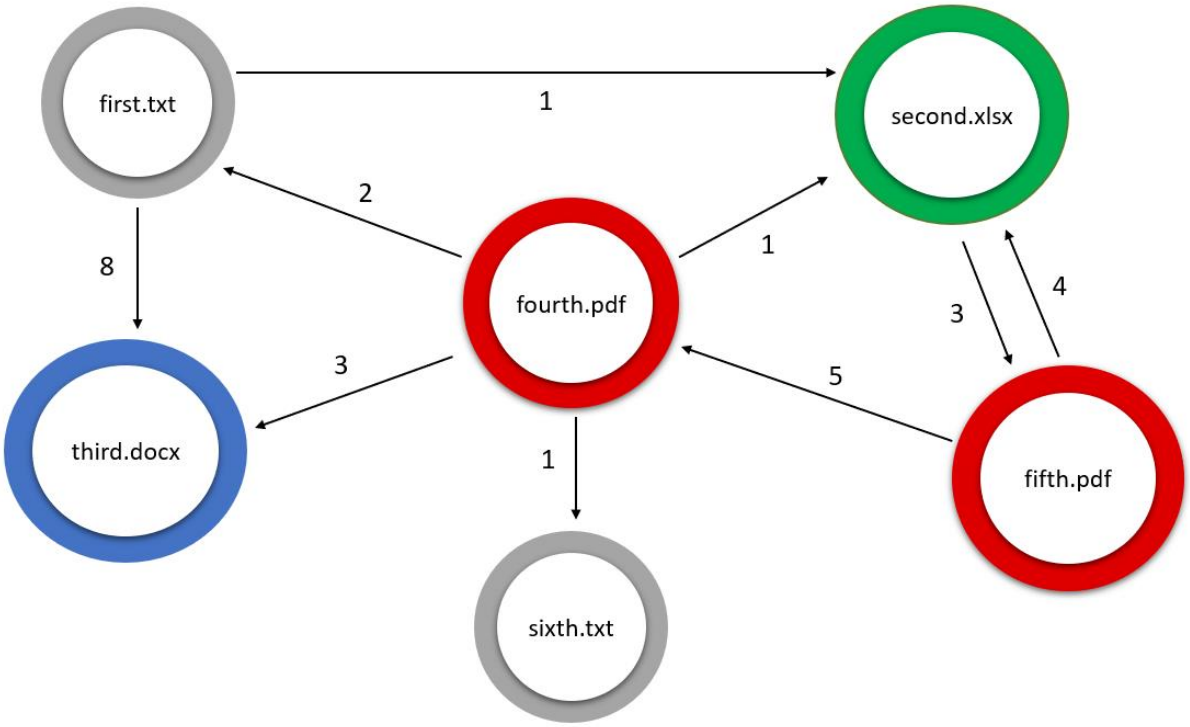
A fent vázolt példát matematikailag jól modellezhetjük Markov-láncok segítségével. Diszkrét idejű, diszkrét állapotterű Markov-lánc alatt egy olyan sztochasztikus (véletlen) folyamatot értünk, amely egész értékű időpillanatokban véges sok lehetséges állapot egyikét veszi fel, továbbá az állapotok közötti váltás teljesíti az ún. Markov-tulajdonságot. Ennek lényege, hogy az adott állapotból egy másik állapotba történő átmenet valószínűsége kizárólag az adott állapottól függ, a korábban felvett állapotoktól (a rendszer előéletétől) nem. Ez a tulajdonság az elsőrendű Markov-láncokra igaz, de egyszerűen kiterjeszthető magasabb rendű láncokra is. A Markov-tulajdonság formálisan a

$$P(X_{n+1} = x \,|X_n = x_n, X_{n-1} = x_{n-1}, \dots, X_1 = x_1) = P(X_{n+1} = x \,|X_n = x_n)$$

összefüggéssel írható fel, amelyet gyakran szokás „emlékezetnélküli” tulajdonságként is értelmezni. Egy Markov-láncot egyértelműen definiál az állapottere (az összes felvehető állapotok halmaza), valamint az állapotátmenet-valószínűségek [15]. Épp ezért egy Markov-lánc reprezentálható akár egy irányított, élsúlyozott gráf segítségével is, ahol az egyes állapotoknak a gráf csúcsait, a közöttük lévő átmenetvalószínűségeknek a csúcsok közötti élek súlyát feleltetjük meg, ahogyan az az 6.4. ábra mutatja.

6.6.3 Megvalósítás

A megvalósított Markov-lánc egyes állapotai megnyitott fájlokat reprezentálnak. Egy állapot akkor kerül felvételre, ha teljesül a feltétel, hogy legalább egy állapotátmenet vezet bele, vagy magából a tárgyalt állapotból vezet átmenet legalább egy másikba.

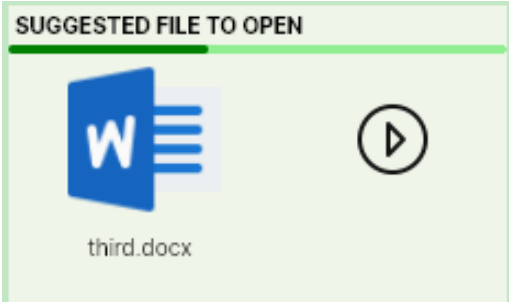


6.4. ábra Gyakorlati példa a Markov-lánc bemutatásához.

Az egyes éleken a megtörtént állapotátmentek száma kerül feljegyzésre.

A 6.4. ábrán egy konkrét példát láthatunk, amely segítséget nyújthat a megértésben. A nyíl mindig annak a fájlnak az irányába mutat, ami a korábban futtatott dokumentumunk után került megnyitásra. Vegyük kiinduló pontnak a *first.txt* állományt. Megfigyelhető, hogy a *second.xlsx* kizárólag 1 alkalommal került megnyitásra a korábban említett fájl után. Ellenben a *third.docx* egy jóval népszerűbb választásnak bizonyul a felhasználó életében. Amennyiben tisztában vagyunk vele, hogy összesen hány fájl megnyitást tudtunk feljegyezni egy adott csúcsból, kiszámolható az egyes továbbhaladási pontokra való átlépés valószínűsége. Tudjuk, hogy *first.txt*-t összesen 8+1, tehát 9 fájl futtatási esemény követte. Ebből kiszámítható az, hogy a *third.docx-re* 8/9, tehát 88% eséllyel esik a felhasználó választása.

A szoftveres megvalósításnál a bővítmény egy saját XML dokumentumban tárolja a már eddig megtapasztalt fájl megnyitási rutinról alkotott ismereteit. Amennyiben két egymást követő műveletről elengedő mennyiségű információt szedett össze, a főképernyő sarkában egy dialógust jelenít meg. Az ablakon látható az ajánlott fájl neve, az ikonja amennyiben előre definiálva lett, illetve egy gomb az azonnali indítására. Megjelenik egy visszafele haladó előrehaladási sáv, amely leteltével az felajánlási ablak eltűnik. Az említett ajánlást tartalmazó dialógus a 6.5. képen tekinthető meg.



6.5. kép A rutin megfigyelő rendszer ajánlása egy fájl megnyitására.

7. TESZTELÉS

A kövekezőkben a szoftver tesztelésének részegységei kerülnek megemlítésre. Első részben a program belső metódusainak funkcionális működését egységtesztek segítségével ellenőriztem. A második fázisba a platformfüggetlenség és a műveletvégzők összehangolt működésének tesztelése került. Ezeket úgynevezett *deploy* tesztekkel ellenőrzöm, ahol minden egyes platformon a megfelelő funkciók működése kerül áttekintésre.

7.1 TESZTELÉSI MÓDSZEREK ÉS ESZKÖZÖK

A most következő fejezetekben a két tesztelési fogalom és az ezek elvégézéhez szükséges eszközök kerülnek ismertetésre.

7.1.1 **Egységteszt**

Az objektum-orientált programozásnál jelentősen elterjedt és népszerű tesztelési direktíva. Lényege az, hogy a kódbázis egy kisebb egységét izolálva, kivülről tekintve megfigyeljük és összehasonlítjuk a működését egy általunk elvárt viselkedéssel [17].

Az egységtesztek lebonyolításához NUnit keretrendszer alkalmaztam. Az NUnit egy szabad felhasználású, nyílt forráskódú tersztelési framework, amely pontosan megfelel azoknak az elveknek és szükségleteknek, amelyek ehhez a teszteléshez szükségesek. A Visual Studio által használt csomagkezelőben könnyedén telepíthető és használható C# környezetben, illetve várhatóan minden .NET implementációs környezetben [2].

7.1.2 **Deploy teszt**

Deploy teszt folyamán lefordításra kerül a megírt kódbázis Windows, Linux és Mac operációs rendszerekre. Definiálásra került általam egy lista, amely minden egyes pontja egy szoftver elemének a tesztelését reprezentálja. Minden egyes pont, minden egyes operációs rendszeren tesztelésre kerül, amelynek eredményét a 7.3. fejezetben részletezem.

A deployolási folyamatot a .NET Core által biztosított *dotnet publish* funkció segítségével hajtom végre. A terminál ablakból futtatható parancs segítségével a CLI[7] egy publish nevezetű mappába felépíti a programunkat, a már fent említett operációs rendszereknek futtatható állapotba. Természetesen meg kell adnunk, hogy melyik OS-t szeretnénk megcélozni, illetve maga a telepítési útvonal is személyre szabható [23].

7.2 EGYSÉGTESZTELÉS

A következő táblázat az egységtesztek célcsoporját, célját és kimenetelét demonstrálja.

[7] A Visual Studio saját beépített parancssoros interfésze. Lehetőséget ad fájlok megnyitására, bővítmények telepítésére, megjelenítés nyelv megváltotatására és különböző diagnosztikák elvégézésére [21].

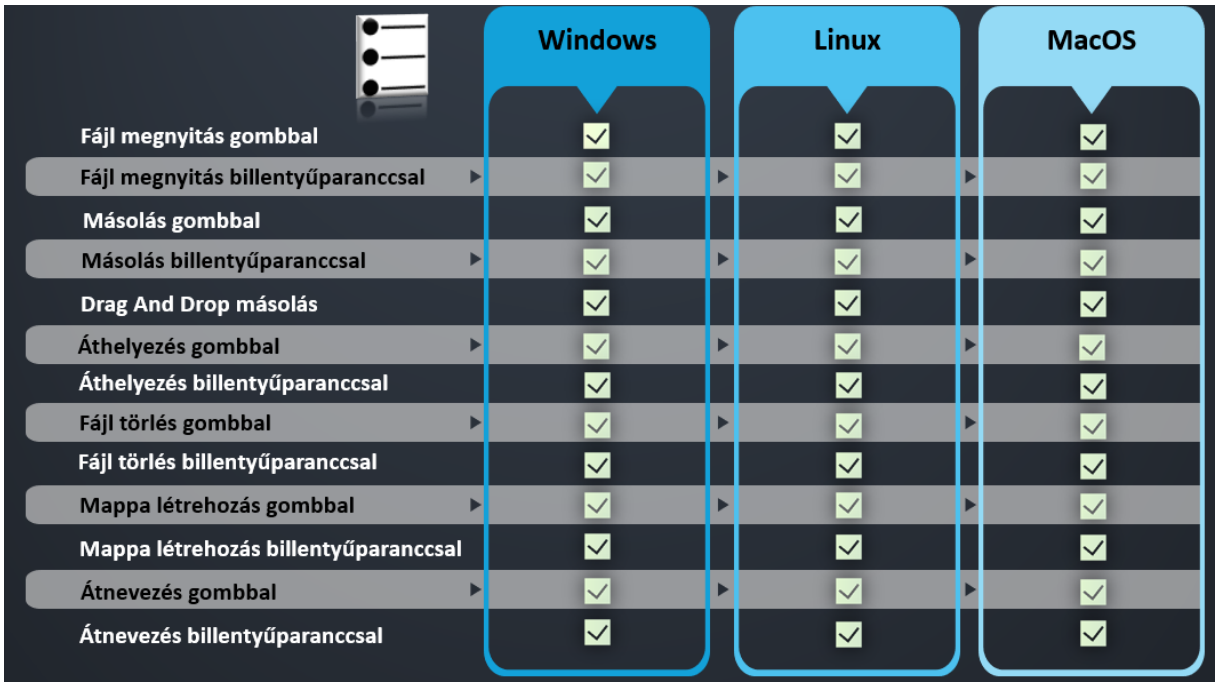
FUNKCIÓ	TESZTELENDŐ FUNKCIÓ LEÍRÁSA	KIMENETEL
Átnevezés	Megfelelő, nem üres bemeneti szöveg paraméter esetén sikeresen átnevezésre kerül a kiválasztott fájl/mappa.	**Sikeres**
Átnevezés	Nem megfelelő útvonal megadásánál IO kivétel dobódik.	**Sikeres**
Mappa létrehozás	Létező útvonal megadása esetén sikeresen létrehozásra kerül egy mappa, a megfelelő pozícióba.	**Sikeres**
Mappa létrehozás	Útvonalnak megadott *null* paraméter hiányos útvonal kivételt dob.	**Sikeres**
Mappa létrehozás	Útvonalnak megadott üres paraméter hiányos útvonal kivételt dob.	**Sikeres**
Meghajtók keresése	A célnak megfelelő számú meghajtót talál a szoftver.	**Sikeres**
Fájlok keresése útvonal alapján	Pontosan akkora fájl lokációkat tartalmazó listát ad vissza a metódus, amennyi mappa/fájl található az adott pozícióból kiindulva.	**Sikeres**
Fájlok keresése útvonal alapján	Pontosan annyi fájlt talál az adott pozícióból és abból leágazva, amennyi ténylegesen megtalálható.	**Sikeres**
Fájlok keresése útvonal alapján	Üres listával tér vissza amennyiben a megadott útvonal nem létezik.	**Sikeres**
Az adott fájlt tartalmazó mappa beazaonosítása	Visszaadja a megfelelő szülő mappát, amennyiben a megadott útvonal létezik.	**Sikeres**
Fájlrendszer lekérdezése, mappa megnyitáskor, fa megjelenítés módban	Megfelelő útvonal hívással visszaadja a látható fájlok számát, miután a felhasználó fa megjelenítés módban bezárt egy mappát.	**Sikeres**
Fájlrendszer lekérdezése, mappa bezárásakor, fa megjelenítés módban	Megfelelő útvonal hívással visszaadja a látható fájlok számát, miután a felhasználó fa megjelenítés módban megnyitott egy mappát.	**Sikeres**
Egyéni megjelenítési opciókból állított példányból másolat készítés	Megfelelően létrehoz egy objektumot és átmásolja a tulajdonságokat a paraméterből kapott forrásból.	**Sikeres**

7.1. táblázat Egységtesztek futtatása során kapott eredmények összessége.

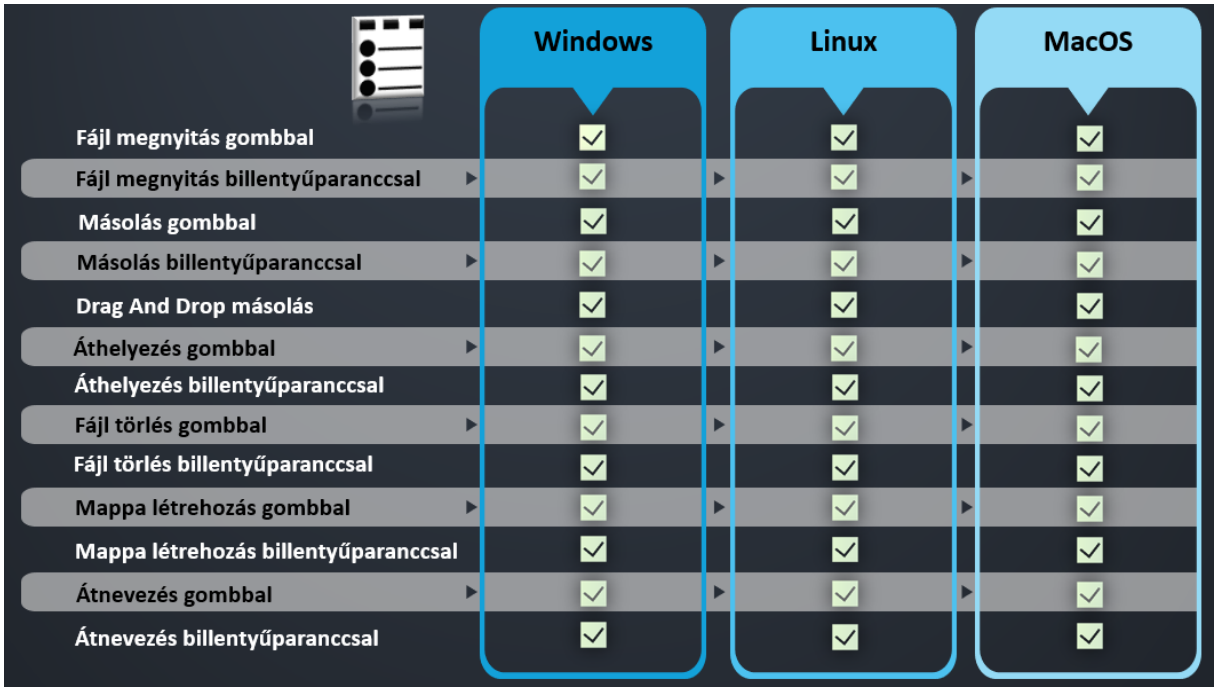
7.3 DEPLOY TESZTELÉS

A következő táblázat reprezentálja a deploy tesztelés vizsgálandó funkciót. Minden egy művelet tesztelés alá került mind a három fent említett platformon.

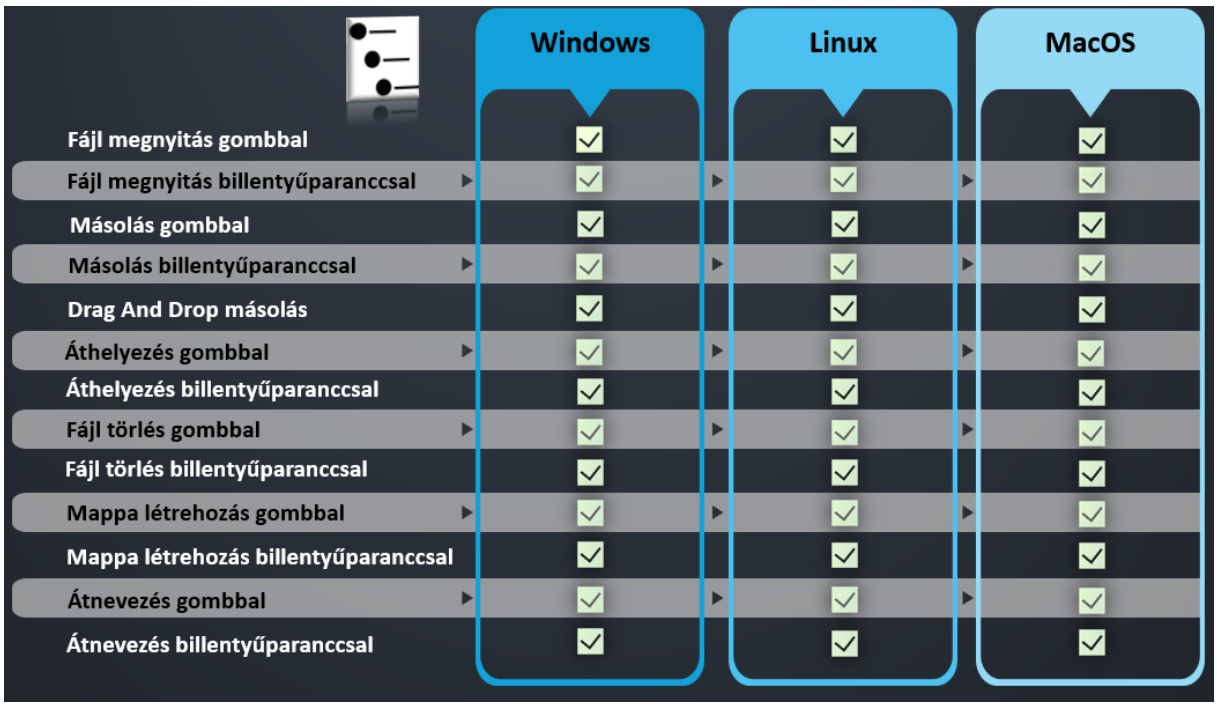
7.3.1 Alapértelmezett funkciók tesztelése különböző megjelenítési módok szerint



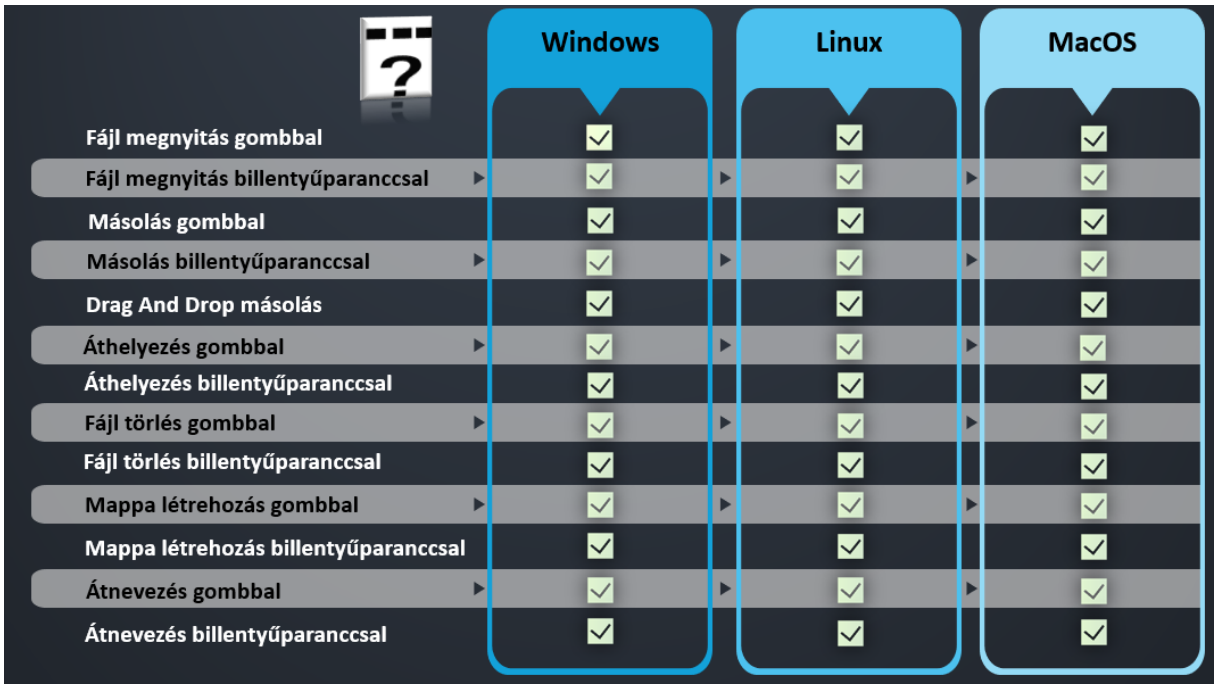
7.2. ábra Alapvető funkciók tesztelésének eredményei részleges megjelenítési módban.



7.3. ábra Alapvető funkciók tesztelésének eredményei teljes megjelenítési módban.



7.4. ábra Alapvető funkciók tesztelésének eredményei fa szerkezetű megjelenítési módban.



7.5. ábra Alapvető funkciók tesztelésének eredményei fa szerkezetű megjelenítési módban.

7.3.2 Bővíthetőségi tesztek a különböző platformokon

Ebben az alfejezetben mind a 3 korábban említett operációs rendszerre, a már előre kitelepített alkalmazás bővíthetőség tesztelésének az eredménye kerül közlésre.

Az alkalmazás mind a három platformon képes:

- Pluginokat fogadni, ellenőrizni és telepíteni.
- A korábban kiválasztott bővítmények tárolására, amelyek ezáltal a szoftver újraindítása során ugyanúgy alkalmazásra kerülnek.
- A 6.5. fejezetben említett fájl információ megjelenítésére alkalmas plugin injektálására és működtetésére. Képes egy billentyűparancs segítségével részletes fájl információt szolgáltatni a felhasználó számára, amennyiben egy fájl kiválasztásra került.
- A 6.6. fejezetben említett rutin megfigyelő bővítmény injektálására és működtetésére. Képes a felhasználónak ajánlani megnyitandó programot, a felhasználó általános tevékenységei által megbecsülve.

7.3.3 **További tesztek eredményei**

A szoftver 7.1.2. alfejezetben említett platformokon elvégzett deploy tesztek alapján:

- Teljes mértékű responsive megjelenítés és felhasználói élményt biztosít. A vezérlő elemek alkalmazkodnak a használó által preferált ablak méretezéshez.
- Mind a két panel engedélyezi, hogy egy legördülő ablakból kiválaszva a felhasználó meghajtó váltást hajtsón végre.
- Az ikonok, a képek és az egyes vezérlő elemek mindegyik operációs rendszeren egyedi módon, megfelelő minőségben jelennek meg.

8. ÉRTÉKELÉS

A végső **értékelés** legfőbb **célja** nem maga a fájlkezelő rendszer minősítése, hanem sokkal inkább a **platformfüggetlen fejlesztéssel kapcsolatos tapasztalatok ismertetése**.

A platformfüggetlen fejlesztés egy kapuként, egy megoldásként szeretne nekünk szolgálni egy régi nehézségre, problémára. Alapvetően az egyes operációs rendszerekre megírt szoftverek, a platformok különbözősége végett különböző fejlesztési környezeteket és programozási nyelveket követelnek meg. Ennek kiküszöbölésére sok próbálkozás született, melynek megtapasztalására és tesztelésére készítettem el ezt a dolgozatot. Az a személy, aki a 2020-as évek környékén kezd neki egy platformfüggetlen keretrendszer által asszisztál projektbe, számos nehezséggel találkozhat.

Az első és talán legfontosabb probléma az, hogy ezek a framework-ök közel sincsenek eléggé bejáratva. Egy szoftverfejlesztő élete jelentős részét internetes keresésekkel tölti, valamint források áttekintésével. Jelen esetben ez a lehetőség vagy nem áll fent, vagy nagyon részlegesen. Az új rendszerek kecsegtető lehetőségekkel és modern grafikai felületekkel próbálják a programozók figyelmét felkelteni, ezekben viszont általánosságban a legnehezebb a fejlesztés a hiányos dokumentációk végett. Tapasztalataim alapján ezek a keretrendszerek legfőképpen a 2.1. és 2.2. fejezetben szereplő rendszerekre próbálnak hasonlítani, de fontos kijelenti, hogy közel sem ugyanazok. Ez a probléma egyik forrása. Hiszen, ha a platformfüggetlen keretrendszerünk a megszokottól egyedibb megvalósítást használ egy eset megoldásához és emellett viszont hiányos a dokumentálása, jelentősen korlátozza a felhasználó fejlesztési idejét és kapacitását. Ezzel együtt az interneten megtalálható felhasználói kérdések száma is jelentősen csekély egy bejáratott keretrendszerhez képest, nem biztos, hogy elég pontos és megbízható információt tudunk szerezni, főleg, ha időkorlátos projekten dolgozunk.

További gyenge láncszámként titulálhatjuk a tényt, hogy a legtöbb szimpatikusnak tűnő keretrendszer fejlesztés alatt van. Pozitívum az, hogyha valami újítást szívesen látnál a rendszerben, általánosságban a fejlesztők aktívak és gyorsan reagálnak a felhasználói visszajelzésekre. Ellenben viszont, ha fontos projekten dolgozol, nem lehet az ember 100%-ig biztos benne, hogy a framework elég stabil lábakon áll ahhoz, hogy komoly erőfeszítéseket és időt öljünk bele.

Egy aktívan fejlesztés alatt álló projektől mit várhatunk el? Számíthatunk olyan nem várt hibákra és hiányosságokra, amelyet nem mi generáltunk, nem vagyunk felelősek érte, ezáltal saját magunk nem tudjuk kijavítani egyik pillanatról a másikra. Ez szintén egy fejlesztési folyamatot lassító tényezőnek. Az implementációs fejezetben említésre került számos hiányosság is előfordul a keretrendszeri hibák mellett. Sok esetben tapasztaltam olyat, hogy több napba telt egy olyan funkció megkerülése csak azért, mert nem volt olyan megoldási

lehetőség, mint amit példával élve egy WPF-nél megszokhattunk. Rengetek időt vett el az interneten való keresés, a dokumentáció hiányossága végett, felhasználói beszélgetésekben és úgyszintén a keretrendszerrel próbálkozó egyének által publikált kódokban. Ezek mellett egy alkalommal kénytelen voltam a keretrendszer fejlesztőitől közvetlenül kérdezni egy olyan problémával kapcsolatban, amivel vagy nem találkozott még senki, vagy csak egyszerűen nem publikálta az általam felderített oldalak egyikére sem.

A tesztelési folyamat időtartalma jelentősen hatványozódik a célplatformok számával. Akkor is kizárólag csak akkor jelenthetjük ki, hogy egy megírt funkció készen áll és a hivatalos programozási paradigmák szerint megfelel a tesztelési követelménynek, miután minden egyes célként meghatározott operációs rendszeren elvégeztük a szükséges ellenőrzéseket. Nagyon gyakran kerülnek elő platform specifikus hibák, olyanok, amelyekre nem feltétlenül számíthatunk a megszokott egy platformos fejlesztői életünkben. Gyakorlati példa volt számomra, hogy az útvonalak összefűzése szöveg formátumban kézzel nem elvégezhető, ellenben a beépített útvonal kombinálásra alkalmas függvény tudja, hogy az adott operációs rendszer „\” vagy „/” jelet használ elkülönítésre [10].

Pozitív tényezőként emelném ki, hogy a különböző platformokra lefordított kódok működőképesnek bizonyultak és igencsak natívhoz hasonló megjelenítési módot demonstráltak. További támogatást jelent a platformfüggetlen keretrendszerek felé, hogy sok esetben könnyebb és gyorsabb egy ilyen rendszert kiismerni, mint különböző programozási nyelveket megtanulni és a hozzájuk tartozó integrált fejlesztői környezetet feltérképezni.

9. ÖSSZEFOGLALÁS

A dolgozatom témájaként egy olyan tapasztalaton alapuló módszertan kialakítást tűztem ki célul, amely a 2020-as évek elején, platformfüggetlen keretrendszerrel való projekt készítés tulajdonságait vizsgálja. Ennek megvalósítására egy jelentősen platformfüggő célprojektbe, egy fájlkezelésére alkalmas rendszer elkészítésébe kezdtem bele.

Kezdetben ismertetésre kerültek a dolgozat során felmerülő fogalmak. Definiálva lett a fájlkezelő rendszer meghatározása és alapvető funkciói, valamint a platformfüggetlenség. Ezt követte a klasszikus, felhasználói felület készítésére alkalmas, kizárólag Windows rendszerre fordítható keretrendszerek bemutatása. Ezen rendszerek ismertetését követően áttekintettem a 2020-as évek elején aktívan használt vagy éppen frissen megjelenő platformfüggetlenséget biztosító architektúrákat. Ezen a ponton, az utoljára említett rendszerek összehasonlításra kerültek a klasszikus rendszerekkel, valamint egymással.

Egy, az irodalomkutatás lezáró végső összevetés keretében kiválasztásra került az adott projekthez viszonyítottan a legideálisabb platformfüggetlen keretrendszer, amely végső soron támogatta a szoftver elkészítését.

A tervezési folyamat során láthattuk az egyes építőelemek és objektumok tervezett kapcsolatát és meglétét. Ennek megvalósítása az implementációs részben részletezésre került, ahol személyes vélemények és felmerülő problémák kerültek további tárgyalásra. Az elkészült projekt minőségellenőrzése egységteszteken és deploy teszteken keresztül lett abszolválva. Az eredmények többnyire táblázatok segítségével és írásos összefoglalókkal lettek demonstrálva.

Az utolsó fejezetben az egész folyamaton áthidaló problémákat, nehézségeket és tapasztalatokat mutattam be.

10. SUMMARY

The topic of my thesis is the development of an experience-based methodology to investigate the properties of project creation with a platform-independent framework in the early 2020s. To achieve this, I embarked on a significantly platform-dependent target project, the creation of a file management system.

At the beginning, the concepts used in the thesis are explained. Therefore the definition and basic functions of a file management system and platform independence were defined. This was followed by an introduction of the classical, exclusively Windows targeted UI frameworks. Once these frameworks had been introduced, we were able to talk about the platform-independent architectures which are actively in use or newly released in the early 2020s. Moreover during these presentations the previously mentioned systems were compared with the classical systems as well as with each other.

In a final comparison a platform-independent framework was selected as the most optimal for the actual project, which ultimately supported the development of the software.

Throughout the design process, we were able to get a view of the planned relationship of each building element. The final building structure of the software was detailed in the implementation section, where personal opinions and problems encountered were further discussed. Quality control of the completed project was ensured through unit tests and deploy tests. The results have been demonstrated mostly through tables and written summaries.

In the evaluation chapter, the problems, difficulties and lessons learnt throughout the development process are highlighted.

KÖSZÖNETNYILVÁNÍTÁS

Ezúton is szeretnék köszönetet nyilvánítani a korábbi témavezetőmnek Szabó-Resch Miklós Zsoltnak és a jelenlegi konzulensemnek Kiss Dánielnek, a tőlük kapott elengedhetetlen szakmai tanácsokért és témavezetési javaslatokért.

IRODALOMJEGYZÉK

[1] Adam Nathan.: Windows Presentation Foundation: Unleashed. Sams Publishing, 2007

[2] Andy Hunt, Dave Thomas, Matt Hargett.: Pragmatic unit testing: in C# with NUnit. Pragmatic Bookshelf; Second edition. 2007

[3] Alexandre Chohfi.: XAML Islands – A deep dive – Part 1, https://blogs.windows.com/windowsdeveloper/2018/11/02/xaml-islands-a-deep-dive-part-1, utoljára megtekintve: 2020.12.17.

[4] Avalonia UI.: About AvaloniaUI, https://github.com/AvaloniaUI/Avalonia, utoljára megtekintve: 2021.01.18.

[5] Avalonia UI.: Introduction, https://avaloniaui.net/docs, utoljára megtekintve: 2020.12.17.

[6] Charles Petzold.: Creating Mobile Apps with Xamarin.Forms. Microsoft Press, 2016

[7] Chris Sells, Michael Weinhardt.: Windows Forms 2.0 Programming. Addison-Wesley Professional, 2006

[8] Daniel Roth, Jeff Fritz, Taylor Southwick.: Blazor for ASP .NET Web Forms Developer. Microsoft Developer Division, .NET and Visual Studio product teams, 2020

[9] David Ortinau.: Announcing .NET MAUI Preview 10, https://devblogs.microsoft.com/dotnet/announcing-net-maui-preview-10/, utoljára megtekintve: 2021.12.07.

[10] Didacus Odhiambo.: System Design in Software Development, https://medium.com/the-andela-way/system-design-in-software-development-f360ce6fcbb9, utoljára megtekintve: 2021.12.03

[11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994

[12] Erik E. Brown.: Windows Forms Programming With C#. Manning Publications, 2002

[13] Henning Heitk¨otter, Sebastian Hanschke, and Tim A. Majchrzak.: Evaluating Cross-Platform Development Approaches for Mobile Applications. Springer-Verlag Berlin Heidelberg. 2013

[14] Kovásznai Gergely, Biró Csaba.: NET-es programozási technológiák. EKF TTK, 2013

[15] Leonard Kleinrock: Sorbanállás- kiszolgálás (Bevezetés a tömegkiszolgálási rendszerek elméletébe) Műszaki Könyvkiadó, 1979

[16] Matthew MacDonald.: Pro WPF 4.5 in C#. Aspress, 2012

[17] Michael Olan.: Unit testing: test early, test often, 2003

[18] Michele Aponte.: Building Single Page Applications in .NET Core 3. Apress. 2022

[19] Microsoft Corporation.: .NET RID Catalog, https://docs.microsoft.com/en-us/dotnet/core/rid-catalog, utoljára megtekintve: 2022.04.04.

[20] Microsoft Corporation.: Blazor Build client web apps with C#, https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor, utoljára megtekintve: 2021.01.20.
[21] Microsoft Corporation.: Command Line Interface (CLI), https://code.visualstudio.com/docs/editor/command-line, utoljára megtekintve: 2022.04.18.
[22] Microsoft Corporation.: Desktop Guide (Windows Forms .NET), https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-6.0, utoljára megtekintve: 2020.12.15.
[23] Microsoft Corporation.: dotnet publish, https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-publish, utoljára megtekintve: 2021.12.03.
[24] Microsoft Corporation.: How to: Share Sizing Properties Between Grids, https://docs.microsoft.com/en-us/dotnet/desktop/wpf/controls/how-to-share-sizing-properties-between-grids?view=netframeworkdesktop-4.8, utoljára megtekintve: 2022.04.15.
[25] Microsoft Corporation.: IntelliSense, https://code.visualstudio.com/docs/editor/intellisense, utoljára megtekintve: 2022.04.15.
[26] Microsoft Corporation.: ItemsPresenter Class, https://docs.microsoft.com/en-us/dotnet/api/system.windows.controls.itemspresenter?view=windowsdesktop-6.0, utoljára megtekintve: 2022.04.15
[27] Microsoft Corporation.: Markup Extensions and WPF XAML, https://docs.microsoft.com/en-us/dotnet/desktop/wpf/advanced/markup-extensions-and-wpf-xaml?view=netframeworkdesktop-4.8&viewFallbackFrom=netdesktop-5.0, utoljára megtekintve: 2020.12.16.
[28] Microsoft Corporation.: Navigation view, https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/navigationview , utoljára megtekintve: 2020.12.17.
[29] Microsoft Corporation.: Single project, https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/single-project, utoljára megtekintve: 2021.12.07.
[30] Microsoft Corporation.: What is .NET MAUI?, https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui, utoljára megtekintve: 2021.12.07.
[31] Microsoft Corporation.: What is MSX?, https://docs.microsoft.com/en-us/windows/msix/overview, utoljára megtekintve: 2020.12.17.
[32] Microsoft Corporation.: What is Windows Presentation Foundation (WPF .NET), https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0, utoljára megtekintve: 2020.12.16.
[33] Microsoft Corporation.: Windows UI Library (WinUI), https://docs.microsoft.com/en-us/windows/apps/winui/, utoljára megtekintve: 2021.02.08.

[34] Microsoft Corporation.: Xamarin.Forms An open-source framework for building iOS, Android, and Windows apps, https://dotnet.microsoft.com/apps/xamarin/xamarin-forms, utoljára megtekintve: 2021.01.19.
[35] Miguel Ramos.: A deep-dive into WinUI 3 in desktop apps, https://blogs.windows.com/windowsdeveloper/2020/07/07/a-deep-dive-into-winui-3-in-desktop-apps/, utoljára megtekintve: 2020.12.17.
[36] Picoe Software.: Introducing Eto.Forms – A cross platform UI for .NET, http://picoe.ca/2012/09/11/introducing-eto-forms-a-cross-platform-ui-for-net/, utoljára megtekintve: 2021.01.20.
[37] Scott Hanselman.: Cross-platform GUIs with open source .NET using Eto.Forms, https://www.hanselman.com/blog/crossplatform-guis-with-open-source-net-using-etoforms, utoljára megtekintve: 2021.12.07.
[38] Scott Hanselman.: Introducing .NET Multi-platform App UI, https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/, utoljára megtekintve: 2021.01.19.
[39] Statista.: Market share held by the leading computer (desktop/tablet/console) operating systems worldwide from January 2012 to December 2021, https://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems-since-2009/, utoljára megtekintve: 2022.04.22.
[40] Szabó-Resch Miklós Zsolt.: Dialogs, Bindings, Converters, https://users.nik.uni-obuda.hu/prog4/GyakorlatPPT_HU/P4LAB_02_DialogsBindingsConverters.pptx, utoljára megtekintve: 2021.12.10.