**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Programozáselmélet és Szoftvertechnológia Tanszék**

Zenelejátszó asztali alkalmazás Avalonia UI-ban, MVVM architektúra használatával

Szerző: Témavezető:

Simon Péter Dr. Gludovátz Attila

Programtervező informatikus BSc. Egyetemi adjunktus

**Budapest, 2024**

A document with text and letters

Description automatically generated with medium confidence

Tartalomjegyzék

[1. Bevezetés 1](#_Toc175236251)

[2. Felhasználói dokumentáció 2](#_Toc175236252)

[3. Fejlesztői dokumentáció 3](#_Toc175236253)

[3.1. Fejlesztői környezet 3](#_Toc175236254)

[3.1.1. Model 3](#_Toc175236255)

[3.1.2. View 3](#_Toc175236256)

[3.1.3. ViewModel 3](#_Toc175236257)

[4. Összefoglalás és további fejlesztési lehetőségek 4](#_Toc175236258)

[5. Irodalomjegyzék 5](#_Toc175236259)

[6. Melléklet 6](#_Toc175236260)

# Bevezetés

Napjainkban a streaming szolgáltatások magas szerepet kapnak a felhasználók körében. Ezek a szolgáltatások havi, több havi, esetleg évi díj ellenében adnak az előfizetőknek hozzáférést nagy mennyiségű médiához. Filmeket tekinthetünk meg, könyveket olvashatunk, zeneszámokat hallgathatunk. Nem kell külön megvásárolni az általunk fogyasztott tartalmakat, gyorsan és könnyen elérhetjük őket. Azonban a pozitívumok mellett mindig fellelhetők a negatívumok is, melyeket sok esetben céges érdekből, máskor maguk az előnyök miatt jönnek létre.

Az előfizetői modellben az ügyfelek tulajdonképpen, mint bérlők vannak jelen, nem pedig mint tulajdonosok. A szolgáltató nyújtotta tartalmakat csak addig érhetik el, amíg regisztrált és fizető felhasználók. Nincs mód például a filmek vagy albumok megvásárlására. Továbbá a kínálat folyamatosan változik. Egyáltalán nem biztos, hogy amit meg akarunk tekinteni éppen elérhető a régiónkban, vagy a szolgáltató kínálatában. Érdemes megemlíteni még, hogy legtöbb esetben folyamatos internet hozzáférés szükséges a weboldal vagy alkalmazás használatához. A világ sok részén a megbízható és stabil internet infrastruktúra még nincs megfelelően kialakítva, így az ezen területeken élő emberek elesnek az ilyen alapú megoldások használatától.

Szakdolgozatomban erre próbálok egy megoldást keresni egy olyan alkalmazás formájában, amely hálózati elérés nélkül, helyi zenefájlokat tud lejátszani. Célom az elérhetőség és felhasználó barát kialakítás biztosítása. Az alkalmazás képes különböző fájlformátumú zeneszámokat lejátszani, azokat különböző szempontok – előadó, album, műfaj – alapján rendszerezni. A megjelenés személyre szabható különböző témák segítségével, támogatott a widget alapú és tálcáról való irányítás.

Opcionálisan az alkalmazás lehetőséget ad Spotify fiókkal való összekapcsolásra, mellyel zene ajánlásokat érhet el a felhasználó. Bár a program elsődleges funkciója az offline zenehallgatás, a szolgáltatók adta lehetőségeket is képes ebben a formában kihasználni.

# Felhasználói dokumentáció

# Fejlesztői dokumentáció

## Fejlesztői környezet

Az alkalmazás fejlesztéséhez az Avalonia UI keretrendszert használom, mely a Microsoft .NET és WPF (Windows Presentation Foundation) keretrendszerére épít. Bár a .NET képes több operációs rendszeren működni, ez önmagában a WPF-ről nem mondható el. Ezt a problémát hárítja el az Avalonia, melynek segítségével Windows-on és Linux-on működőképes GUI-t lehet az alkalmazáshoz kötni. Az applikáció MVVM (Model – View – ViewModel) architektúrát követ, 3 logikailag jól elkülöníthető részre osztva annak felépítését.

### Model

A Model feladata az alkalmazás futása alatt szükséges objektumok és mezők biztosítása. Itt hoztam létre a különböző C# osztályokat, melyeket a ViewModelek-ben példányosítok. Adatokat, ha tartalmaznak is, akkor azok statikus adatok, legtöbb esetben azonban nem ezt a célt szolgálják.

### View

A View a megjelenítésért felelős egység. A GUI alkotó elemeinek létrehozása a WPF-ben használt XAML (Extensible Application Markup Language) Avalonia által bővített AXAML kiterjesztésű fájlokban történik. Az XAML szintaxisa és az Avaloniából érkező új osztályok segítségével nagy részletességgel lehet létrehozni a felhasználói felületet. Az AXAML fájlok kiegészülnek még egy C# háttér kóddal, melynek egyetlen feladata a komponens inicializálása. Érdemes megjegyezni, hogy ez a code behind használható események kezelésére is (event handling). Azonban a választott architektúra miatt a nézetet nem használom felhasználói bevitel kezelésére, szigorúan elválasztom a kódtól ilyen szempontból. Az interakciókat a ViewModellekben kezelem, „model binding” segítségével.

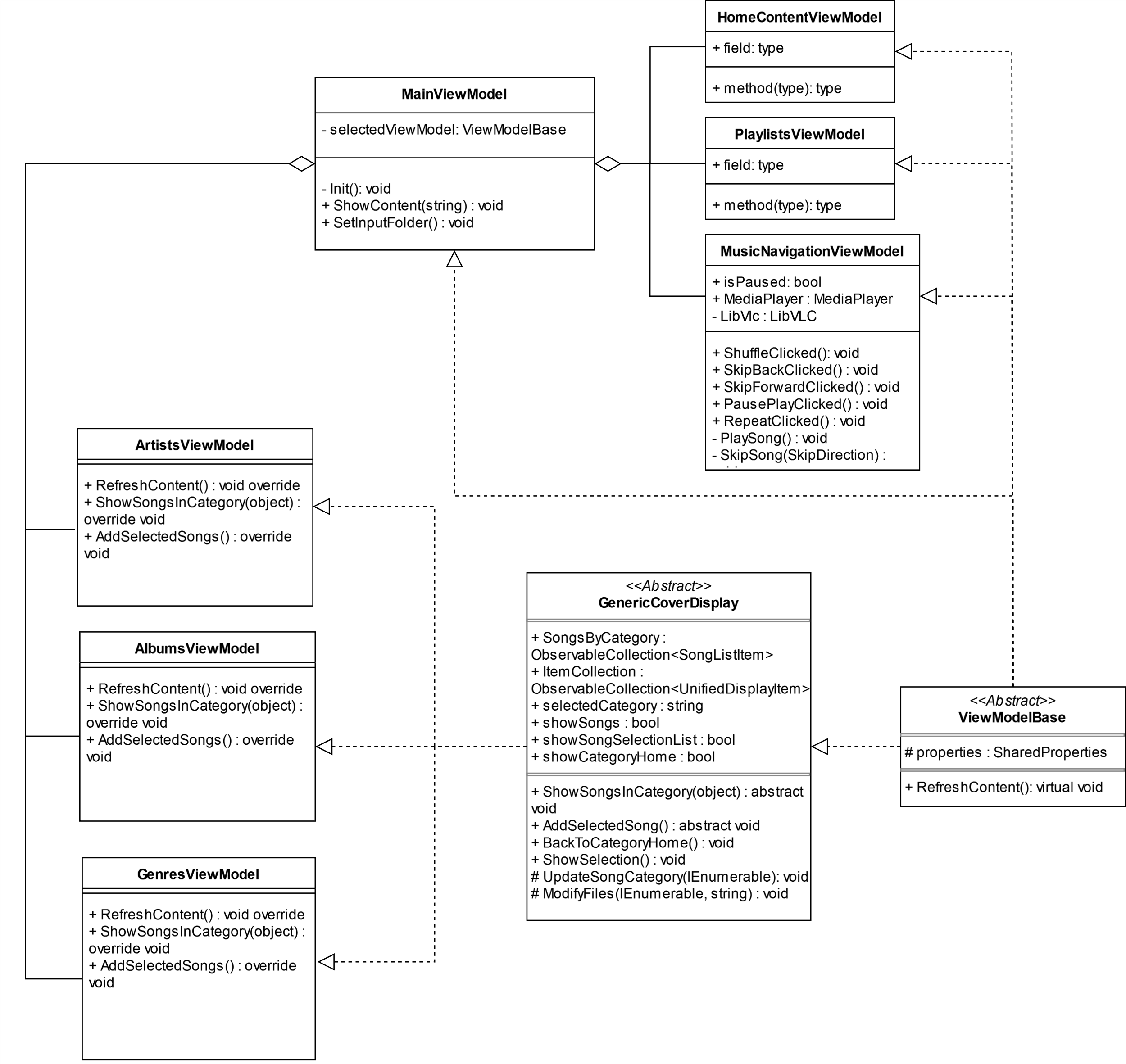
### ViewModel

Az alkalmazás üzleti logikája, a felhasználói bevitel a ViewModellekben található. A nézetben korábban említett model binding az itt található objektumokhoz és mezőkhöz bind-ol. Ebben az egységben minden osztály egy ViewModelBase osztályból van származtatva, amely a CommunityToolkit.MVVM csomag ObservableObject osztályának gyereke.

A CommunityToolkit egy olyan, Microsoft által fejlesztett csomag, mely nagyban megkönnyíti az MVVM alkalmazások fejlesztését. A csomagot elsősorban a forráskód generáló funkciója miatt választottam, melynek segítségével az ismétlődő kódrészletek úgynevezett Property Tag-ek használatával elkerülhetőek. Az ilyen tagekkel ellátott mezők helyettesítve lesznek a megfelelő kódrészletekkel a fordító program által.

## Business Logic felépítése – TODO: assess and continue this whole parapgraph after getting sufficent sleep

A ViewModel-ek UML diagram segítségével az alábbi módon írhatóak le:



1. ábra - PLACEHOLDER UML DIAGRAM

Az alkalmazás alapjául a MainViewModel szolgál, amely a többi ViewModel-t aggregálja. Mindegyik egy ViewModelBase abstract osztálynak a leszármazottja, ami pedig a CommunityToolkit által szolgáltatott ObservableObject osztályból öröklődik.

# Összefoglalás és további fejlesztési lehetőségek

# Irodalomjegyzék

# Melléklet