

**Vizuális\_programozás\_L-K-GINFBAL-VIZUPROG-1**

**L\_LA01\_Vizprog\_WPF– Hallgatói Nyilvántartó applikáció**

**Schiller Viktor – GWOQXX**

**Második Évfolyam – 2. Szemeszter**

**NJE GAMF Műszaki és Informatikai Kar**

Tartalom

[1. Feladat leírás 2](#_Toc132995439)

[1.1. A projekt követelményei 2](#_Toc132995440)

[1.2. A WPF technológia 2](#_Toc132995441)

[1.3. A kezdeti lépések – adatbázis séma tervezése 3](#_Toc132995442)

[2. Elrendezés kialakítása VS2022-ben 6](#_Toc132995443)

[2.1 Az Adatbázis osztályainak létrehozása 9](#_Toc132995444)

[3. Az adatbázis generálása 12](#_Toc132995445)

[3.1 Felület hozzárendelése az adatokhoz 16](#_Toc132995446)

[3.1.1 MainWindow.xaml.cs 16](#_Toc132995447)

[3.1.2 Rogzites.xaml.cs 18](#_Toc132995448)

[3.1.3 A MainWindow alsó három gombja és funkciója 19](#_Toc132995449)

[4. Az osztályok 22](#_Toc132995450)

[5. Az eredmény: 24](#_Toc132995451)

[FELHASZNÁLT IRODALOM & KÉPEK 27](#_Toc132995452)

# 1. Feladat leírás

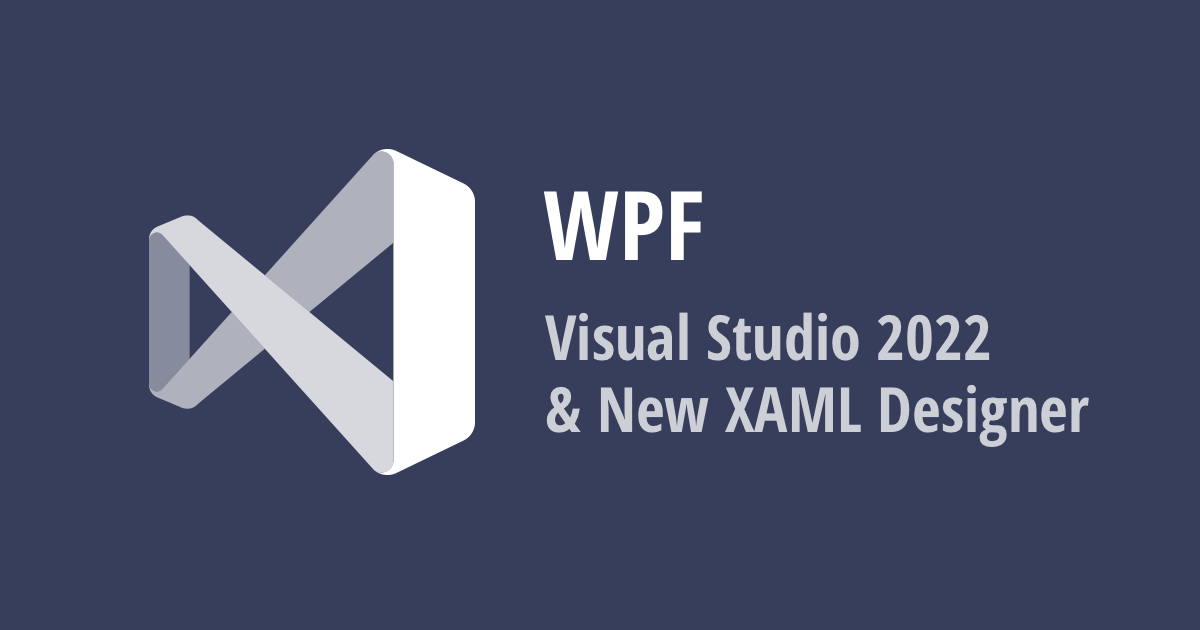
## **1.1. A projekt követelményei**

Egy szabadon választott asztali alkalmazás fejlesztése. Minimum követelmények:

* Grafikus felület, menü, eszköztár.
* A felület WPF vagy Windows Forms vagy MAUI technológiát kell használjon.
* Az alkalmazás adatokat kell tároljon egy MS SQL Express (Localdb) vagy SQLite adatbázisban vagy XML állományban vagy JSON formátumú állományban.
* Az adatbázis legalább két olyan táblával kell rendelkezzen, amelyek között több a többhöz kapcsolat áll fenn.
* Az adatok felvitelének, módosításának, visszakeresésének és törlésének lehetősége legyen biztosítva a felhasználó számára.
* Az adatok kezelése történhet Entity Framework Core vagy DataSet megoldással.

## **1.2. A WPF technológia**

A WPF mozaikszó a Windows Presentation Foundation mozaikszavak rövidítése.



*1. ábra. WPF design ábra*

A Windows Presentation Foundation (WPF) és az Entity Framework (EF) két külön technológia, de lehet őket együtt használni az alkalmazások fejlesztéséhez.

Az Entity Framework egy .NET keretrendszer, amely lehetővé teszi az adatok tárolását és elérését az adatbázisokban. Az EF segítségével a .NET fejlesztők objektum-orientált módon dolgozhatnak az adatokkal, és az EF automatikusan leképezheti az objektumokat az adatbázis tábláira és az adatbázis tábláit objektumokra.

A WPF egy felhasználói felület (UI) keretrendszer a Windows alkalmazásokhoz. A WPF lehetővé teszi a fejlesztők számára, hogy gazdag, interaktív és testre szabható felhasználói felületet hozzanak létre, amely könnyen kezelhető és támogatja a sokféle multimédiás tartalmat.

Ha együtt használják a WPF-et az Entity Framework-fel, a fejlesztők könnyen létrehozhatnak olyan alkalmazásokat, amelyek lehetővé teszik a felhasználók számára az adatok megtekintését, szerkesztését és mentését az adatbázisban. Az EF objektumokat kínál a WPF számára, amelyeket a felhasználói felületen megjeleníthetnek és szerkeszthetnek.

Összességében a WPF és az Entity Framework együtt lehetővé teszi a fejlesztők számára, hogy könnyen és hatékonyan dolgozzanak az adatokkal, és gazdag, testre szabható felhasználói felületet hozzanak létre a felhasználók számára.

A UI leírásához egy új nyelv is született, ami az XAML (eXtensible Application Markup Language) nevet kapta. Ez egy XML alapú leírónyelv, amit a HTML és a korai SVG szabvány inspirált. A UI leírása így vektorgrafikus, korlátok nélkül skálázható és valamilyen szinten reszponzív is (A HTML5-től azért messze áll a WPF).

A WPF használható .NET Framework alatt és .NET Core 3 vagy újabb .NET rendszereken is. Az alkalmazásomat .NET Core 3.1 segítségével készítettem el.

## **1.3. A kezdeti lépések – adatbázis séma tervezése**

Az adatbázis séma tervezési folyamata

A tervezés az alábbi lépésekből állt:

* Az adatbázis céljának meghatározása
* A szükséges információk megkeresése és rendszerezése.
* Összegyűjteni minden információt, amelyet rögzíteni szeretnék az adatbázisban, például a tanulók nevét és NEPTUN kódját, stb...

Az információk táblákra osztása

* Az információt felosztottam nagyobb entitásokra vagy témákra. Ezután majdnem minden témából tábla lesz.

Információs elemek oszlopokká alakítása

* El kellett dönteni, milyen információkat szeretnék tárolni az egyes táblákban. Minden elem mezővé alakul, és a táblában oszlopként fog megjelenni.

Elsődleges kulcsok megadása

* Meghatároztam az egyes táblák elsődleges kulcsát. Az elsődleges kulcsok az egyes sorokat egyedileg azonosító oszlopok. Elsődleges kulcs lehet(-ne) például a Neptun-Kód vagy adott esetben a TagozatNév.

Táblakapcsolatok beállítása

* A táblákat áttekintve el kellett döntenem, hogyan kapcsolódnak az egyes táblák adatai a többi tábla adataihoz. Ha egyértelművé szeretném tenni a táblák kapcsolatát, létrehozhatok új mezőket a táblákban, vagy létrehozhatok új táblákat.

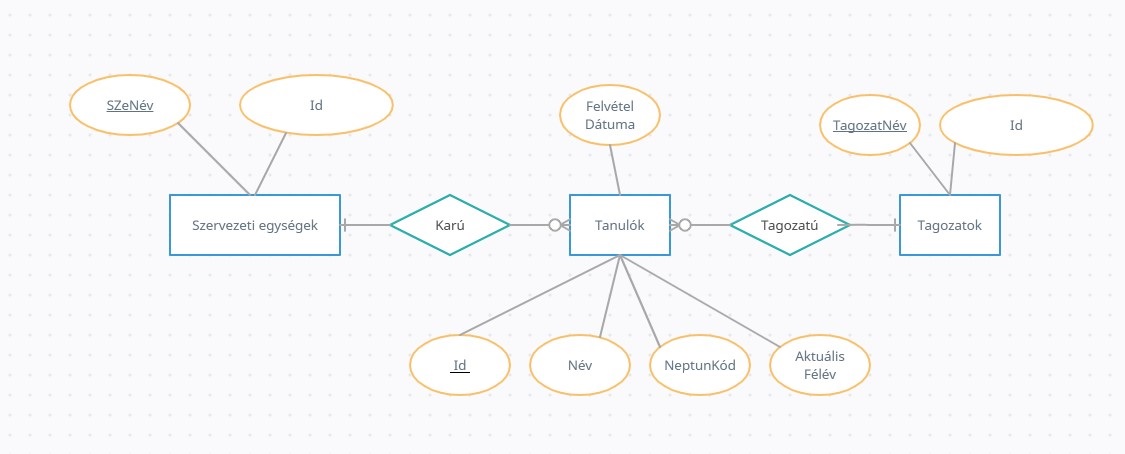
A terv finomítása

* Meggyőződtem róla, hogy hibáktól mentes a terv. Létrehoztam a táblákat, és hozzáadtam néhány rekordnyi mintaadatot. Ellenőriztem, hogy a táblák a kívánt eredményeket adják-e. Szükség esetén módosítani kellett a tervet.

A normalizációs szabályok alkalmazása

* Az adatnormalizációs szabályokat alkalmazva ellenőriztem, hogy a táblák szerkezete megfelelő-e. (1NF, 2NF, 3NF, stb.)

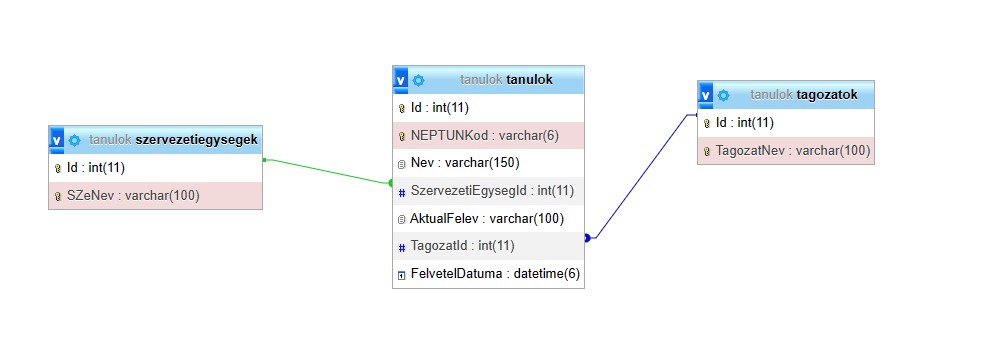
A projekt alapgondolata egy egyszerű tanuló / hallgató nyilvántartó adatbázis létrehozása volt (fiktív adatokkal feltöltve), ami a projekt követelményeinek megfelel. Azaz lehessen új tanulót regisztrálni-felvenni, módosítani, törölni, rákeresni adott kritériumokkal. Úgy gondoltam, kezdetnek három tábla elég lesz, de a későbbi bővíthetőség / rugalmasság potenciája benne maradt a feladatban. A megtervezett E-K Diagram (A Creately diagram-készítő alkalmazást használtam) így nézett ki:



*2. ábra. A Tanulók E-K Diagram*

Az információk táblákba való felosztásának egyik stratégiája az, hogy szemügyre vesszük az egyes dolgokat és meghatározzuk, hogy ezek mire vonatkoznak. Keressük meg azokat az információkat, amelyekhez az adatbázisból hozzá kívánunk férni, és azt osszuk fel olyan témakörökre, amelyeket nyomon kívánunk követni! Ezen témakörök mindegyikéhez szükséges lehet egy-egy külön tábla.

A fenti terv alapján az alábbi adatmodell került kialakításra:



*3. ábra. Relációs adatmodell*

Tanulók Adatbázis:

1. Szervezeti Egységek (Id (int), Szervezeti Egység Név (varchar))

2. Tanulók (Id (int), NEPTUN Kód (varchar), Név (varchar), Aktuális félév (varchar), Felvétel dátuma (dateTime))

3. Tagozatok (Id (int), Tagozat Név (varchar))

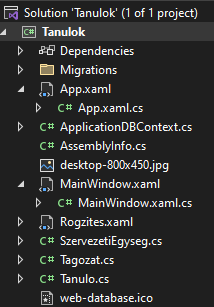
Az adatbázist lokális környezetben, sajátgépen (localhost) a PHPMyAdmin felületen, a XAMPP Control Panel applikáció segítségével üzemeltettem SQL alapokon.

# 2. Elrendezés kialakítása VS2022-ben

Az új Projektet Visual Studio-ból indítottam, ennek hatására létre jön a munka és az összes szükséges fájl ami egy WPF alkalmazás működéséhez szükséges. De pontosan milyen fájlok jönnek létre és mi a feladatuk?

[ App.xaml, App.xaml.cs, MainWindow.xaml, MainWindow.xaml.cs, AssemblyInfo.cs ]

Ami elsőre feltűnhet, hogy nincs Program.cs fájl a generált fájlok között, ami az alkalmazás belépési pontját, a Main metódust definiálná. Ennek az oka az, hogy ez az App osztály ősében, az Application osztályban van definiálva, ezért nem kell. Természetesen készíthetünk saját Main metódust, mert előfordulhat, hogy szükségünk van rá. Ebben az esetben viszont definiálnunk kell, hogy a CLR melyik osztály Main metódusát tekintse belépési pontnak, valamint nekünk manuálisan kell beállítanunk elég sok mindent.



*4. ábra. VS2022-ben létrejövő fájlok*

App.xaml.cs

Ez az alkalmazás fő osztálya, ami az Application osztályból öröklődik. Feladata a WPF alkalmazásunk futtatása és elindítása.

App.xaml

Az alkalmazás leíró fájlja. Ez egy XAML dokumentum, ami az alkalmazásunk tulajdonságait állítja be. Ezek közül a legfontosabb a StartupUri tulajdonság, ami meghatározza a futtatáskor elsőnek elinduló ablakot.

A tulajdonságok beállításán kívül másik fontos szerepe ennek a fájlnak a stílusok betöltése. A WPF rendelkezik egy stílusrendszerrel, ami a CSS-hez hasonlóan lehetővé teszi, hogy alkalmazás szinten definiáljuk egyes vezérlő elemek kinézetét. Ezeket a stílusokat vagy ebben a fájlban definiáljuk, vagy behivatkozzuk azokat a fájlokat, amelyek tartalmazzák a stílusokat.

MainWindow.xaml

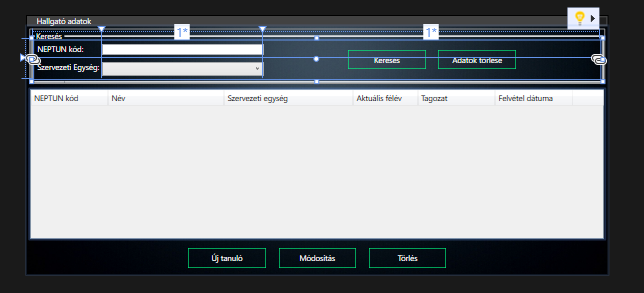
Az alkalmazás fő ablaka. Ez fog megjelenni a képernyőn a program elindítása után. Ha az alkalmazás MVVM-et használ, akkor mondhatjuk, hogy ez egy View.

MainWindow.xaml.cs

Ez a fő ablak ’code behind’ fájlja, ami az XAML dokumentumhoz tartozó C# osztály. Ebben a fájlban jön létre minden, az XAML-ben definiált eseménykezelő metódus, illetve ha logikát akarunk az ablakhoz rendelni, akkor azt itt tehetjük meg.

Munkámat a GRID kialakításával kezdtem, a MainWindow.xaml-ban. Oszlop és sorkialakítással 2X3-as hálót vettem fel. Felső részébe egy „GroupBox” került, „Keresés” header-rel.

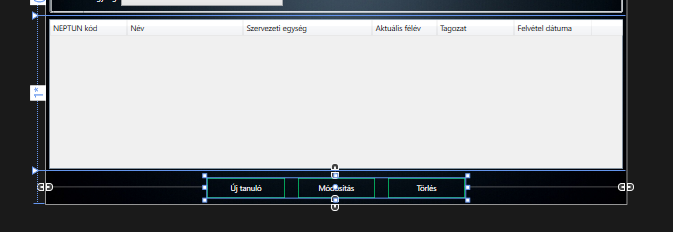
Ebben az elrendezésben a felső két sor a két darab ’Label’ és hozzájuk tartozó ’TextBox’ és ’ComboBox’-é lett, a harmadik oszlop pedig egy WrapPanel-en ’RowSpan’ paranccsal központosítva, a két gombnak adott helyet, ami a „Keresés” és az „Adatok törlése” lett.



*5. ábra. A GRID ’header’ és felsőrész kialakítása*

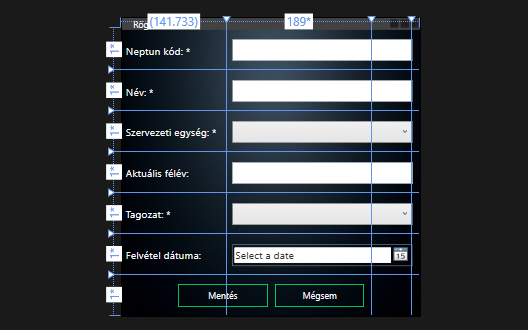
A két gomb kialakításával egy időben, azokra kattintva létrejöttek a függvények a MainWindow.xaml.cs - fájlban.

Alatta került kódolásra a „DataGrid”, ami az adatbázisból kiolvasott eredmények demonstrálására hivatott, tulajdonképpen a ’display’. Ennek ’Grid.Row=1’ fejlécében lett kialakítva hat darab oszlop (DataGridColumn), amik majd a megadott névvel hivatkoznak az adatbázis adott rekordjaira. Az adat-ablak alatt alakítottam ki egy újabb WrapPanel-t a három gombnak (Új Tanuló, Módosítás, Törlés), a ’Grid.Row=2’ -ben. Ezek margóval beállítva középre igazítással nyerték el végső helyzetüket.



*6. ábra. A DataGrid és alsó WrapPanel kialakítása*

Ezek után jobbklikk és új .xaml fájl lett létrehozva „Rögzítés” címmel, ami egy új ablakot, és vele együtt egy új osztályt is (Rogzites.xaml.cs) létrehozott. Ebben három oszlop és hét sor került kialakításra, az alsó sor oszlopösszevonással a gombok megfelelő elhelyezése miatt. A harmadik oszlop egyfajta „margóként” funkcionál csak, hogy ne kelljen minden TextBox és ComboBox után külön-külön margót beírogatni. Ezzel a design elemek kialakítása többé-kevésbé véget ért.



*7. ábra. A „Rogzites.xaml” GUI kialakítása*

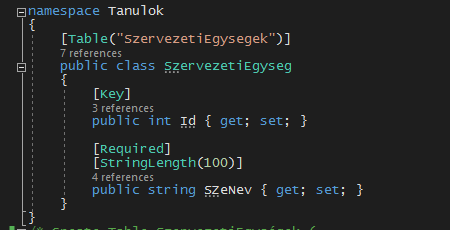
Az ablak szürke default design-ját eldobva, beállítottam mindkét ablak hátterének egy sötét background picture-t. Ezt a .xaml fájlban és a VS2022 Properties úszómenüjében is be lehet állítani.



*8. ábra. Az ablakok háttérszínének beállítása képpel*

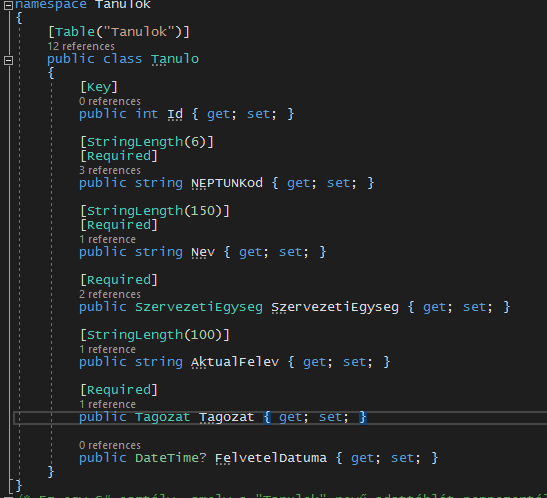
## **2.1 Az Adatbázis osztályainak létrehozása**

Létre kellett hoznom a tábláknak megfelelő osztályokat a Solution Explorer-ben. Elsőnek a ’SzervezetiEgység.c’ osztályt hoztam létre, ami a példában szereplő „Tanulónyilvántartó” app, főiskola/egyetem karait és azok megnevezését fogja tartalmazni. Az osztálynak két adattagja van: egy "Id" azonosító, és egy "SZeNev" elnevezési adattag, ami szöveges típusú és kötelező megadni és a ’Rogzites.xaml’ ablak-oldalon lévő DisplayMemberPath hivatkozik rá, egy legördülő menü formájában. Az osztályt az Entity Framework használja adatbázis táblaként, azért van megadva a [Table("SzervezetiEgysegek")] jelölés, ami a tábla nevét határozza meg. Az [Key] attribútum jelöli az elsődleges kulcsot az adatbázisban, azaz az "Id" adattagot, amely egy egész típusú (int) érték.



*9. ábra. A ’SzervezetiEgység.cs’ osztály*

Másodikként a jóval bonyolultabb és szerte ágazóbb ’Tanulo’ osztályt hoztam létre. Ez egy C# osztály, amely egy "Tanulo"-t reprezentál, azaz egy diákot. Az osztálynak hat adattagja van: egy "Id" azonosító, egy "NEPTUNKod" azonosító, egy "Nev" nevű adattag, amely szöveges típusú és kötelező megadni, egy "SzervezetiEgyseg" nevű adattag, amely egy másik osztály típusú és kötelező megadni, egy "AktualFelev" nevű adattag, amely szöveges típusú és opcionálisan megadható, egy "Tagozat" nevű adattag, amely egy másik osztály típusú és kötelező megadni, és végül egy "FelvetelDatuma" nevű adattag, amely dátum típusú és opcionálisan megadható.

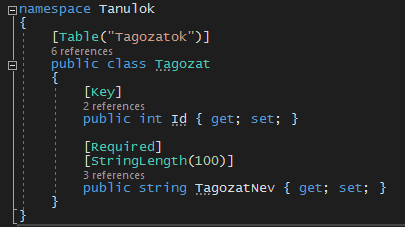


*10. ábra. A ’Tanuló.cs’ osztály*

Az osztályt az Entity Framework szintén használja adatbázis táblaként, azért van megadva a [Table("Tanulok")] jelölés, ami a tábla nevét határozza meg. Az [Key] attribútum jelöli az elsődleges kulcsot az adatbázisban, azaz az "Id" adattagot, amely egy egész típusú érték. Az [Required] attribútum jelzi, hogy az adott adattagnak kötelező a megadása.

Az "SzervezetiEgyseg" és "Tagozat" adattagok olyan típusúak, amelyek az előbb bemutatott "SzervezetiEgyseg" és "Tagozat" osztályokra utalnak, így ezt az adatmodellt az elkészített relációs adatmodellből is lehet reprezentálni, a Tanuló táblával, ami tartalmazza a különböző relációkat az egyéb táblák felé. Az elsődleges kulcsnak sok gondolkodás után az ’Id’ lett beállítva és NEM a NeptunKód, mivel ha később bővítésre kerül az adatbázis más táblákkal kiegészülve, az idegen kulcs karaktereinek száma „csupán” egy int típusú szám lesz, nem pedig 6 karakteres VARCHAR.

Harmadik táblának a ’Tagozat.cs’ osztály került megszerkesztésre. Az osztálynak két adattagja van: egy "Id" azonosító, amely egész (int) típusú és az elsődleges kulcs az adatbázisban, és egy "TagozatNev" nevű adattag, amely szöveges típusú és kötelező megadni. Az osztályt szintén az Entity Framework használhatja adatbázis táblaként, azért van megadva a [Table("Tagozatok")] jelölés, ami a tábla nevét határozza meg. Az [Key] attribútum jelöli az elsődleges kulcsot az adatbázisban, azaz az "Id" adattagot, amely egy egész típusú érték. Az [Required] attribútum jelzi, hogy az adott adattagnak kötelező a megadása. Ez az osztály szintén a relációs adatmodell egyik része, ahol a Tanuló táblában használt Tagozat oszlop egy másik táblára mutat. A Tagozatok táblában vannak a további TagozatNev-ek, amelyek egy-egy Tanulóhoz tartoznak.



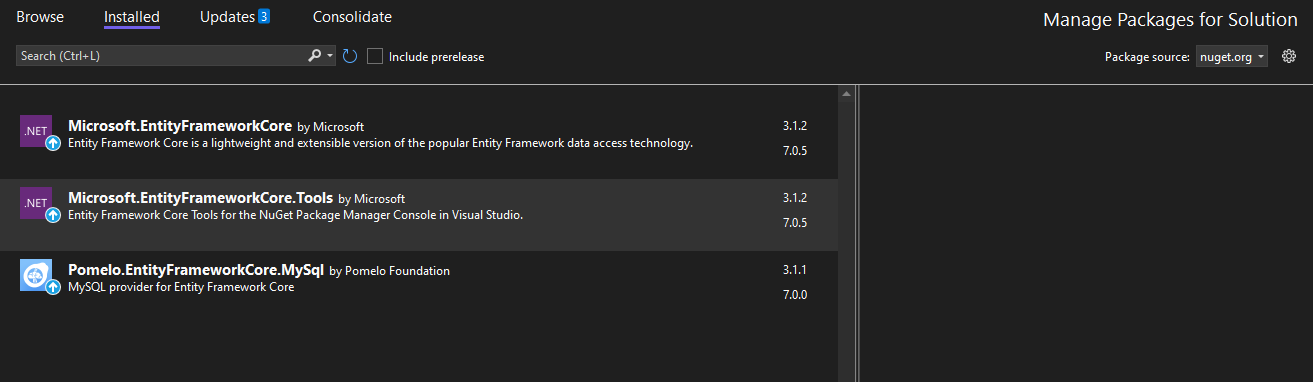
*11. ábra. A ’Tagozat.cs’ osztály*

# 3. Az adatbázis generálása

A .Net Core project igényel különböző NuGet package-eket, amiket külön kell telepíteni ahhoz, hogy az adatbázis migráció rendben megtörténjen. Ehhez legfőképp három kiegészítő csomag kellett:

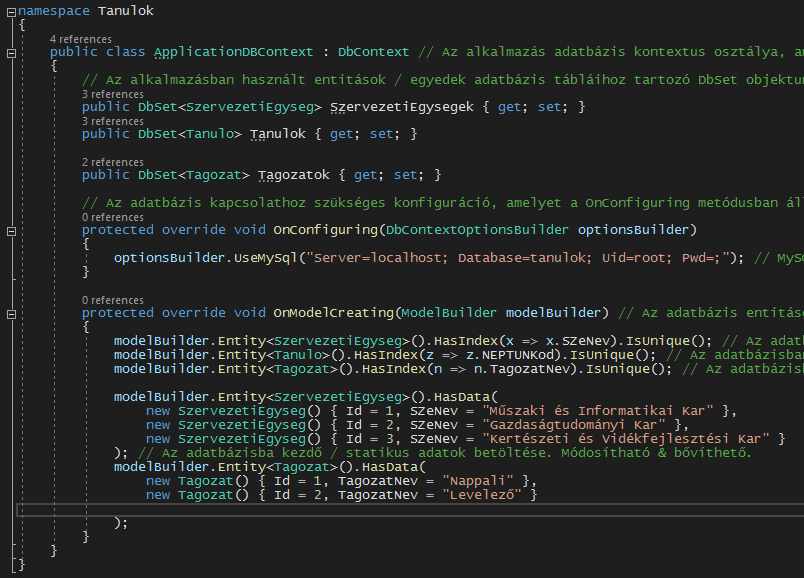
* Pomelo.EntityFrameworkCore.MySql
* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Tools

Ezeket a csomagokat a NuGet Package Manager Console-ból, parancssorból, vagy eleve a NuGet Solution-ból kattintással is telepíthetjük, ahol menedzselhetjük is telepített csomagjainkat.



*12. ábra. A ’NuGet Package Manager’ már telepített csomagjai*

Következő lépésként a DBContext létrehozása volt a cél. Az itt látható kód egy osztálydefiníciót tartalmaz, amely az Entity Framework használatával egy adatbázis kontextust reprezentál. Az Entity Framework egy olyan programozási keretrendszer, amely lehetővé teszi az alkalmazások számára, hogy adatbázisokkal dolgozzanak, anélkül, hogy közvetlenül SQL nyelvet kellene írnom. Az ApplicationDBContext osztály az Entity Framework DbContext osztályából származik, és a már megalkotott SzervezetiEgyseg, Tanulo és Tagozat típusokhoz tartozó adatbázistáblákat tartalmazza. Az adatbázis kapcsolatot az OnConfiguring metódusban állítom be, ahol a MySQL adatbázis **kapcsolati stringjét** adtam meg. Az OnModelCreating metódusban pedig az adatbázisban található entitások modellezéséhez tartozó konfigurációkat állíthatunk be. Az adatbázisban egyedinek kell lennie a SzervezetiEgyseg, Tanulo és Tagozat nevének, amelyet az Entity Framework ’Index’ funkciójával érhetünk el. Ezen kívül a metódusban kezdő / statikus adatokat is betölthetünk az adatbázisba, például a SzervezetiEgyseg és Tagozat osztály példányait.

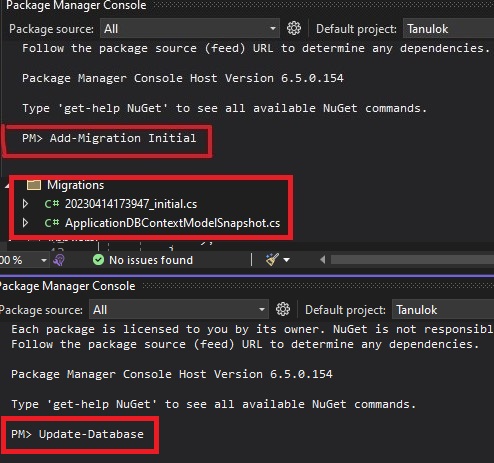


*13. ábra. DBContext osztály létrehozása*

Az ’OnConfiguring’ függvényben definiáljuk a ’Connection-string’ azaz a kapcsolati sztring-et, az ’OnModelCreating’ függvényben pedig megadom azokat az egyedeket, melyek egyediek (.IsUnique()), illetve a .Hasdata object-el pedig részletesen megadom azokat a tulajdonságokat, amik a legördülő menükben fix (egyedi) attribútumként beállíthatóak lesznek. Ezt követően a „Build” paranccsal még egyszer ellenőrzöm, hogy ne legyen futtatási hiba, és ezzel kész.

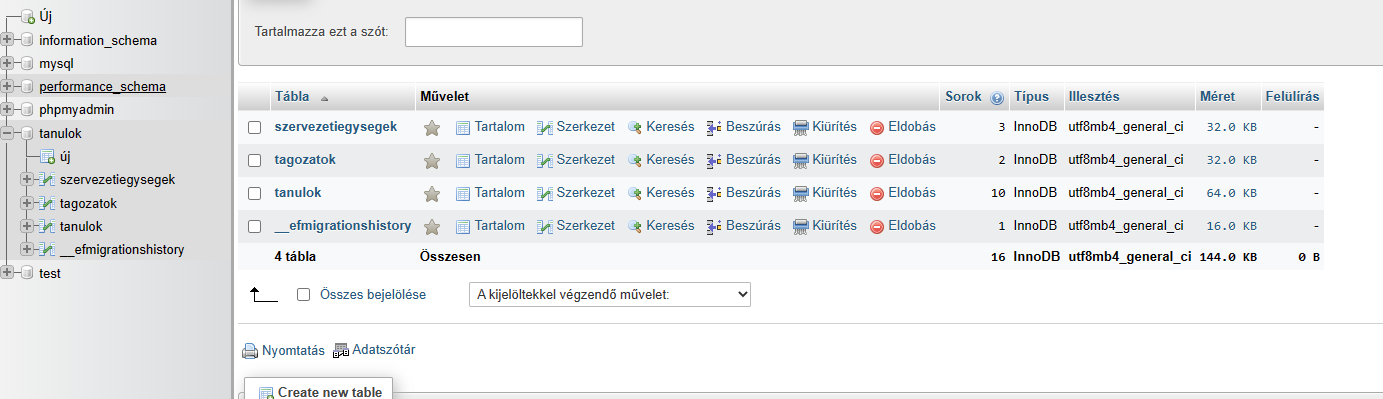
A NuGet Package Manager Console megnyitása után ezek után akkor ki is adhatom az „Add-Migration Initial” parancsot, ami legenerálja az adatbázisunkat MySql környezetben, illetve az „Update-Database” paranccsal fel is tölti azt. Ehhez a művelethez korábban már elindítottam a sajátgépen üzemelő XAMPP Control Panel-t és aktiváltam localhost-on az Apache és MySql szervereket. A NuGet konzolon lefuttatott parancsok rendeltetésszerűen le is futottak, megalkotva a Solution Explorer-ben a ’Migrations’ mappát, melyben automatikusan létrejönnek a migrációs osztályok.

Leellenőrizve továbbá a XAMPP admin felületét, tapasztalhatjuk, hogy az adatbázis sikeresen létrejött, a megadott táblákkal, kulcsokkal, indexekkel, hivatkozásokkal.



*14. ábra. Migrációs parancs futtatása és az eredmény, illetve a frissítés*

Ezzel kész az adatbázis, és az adatbázis reprezentációra szolgáló osztály adatstruktúra is.

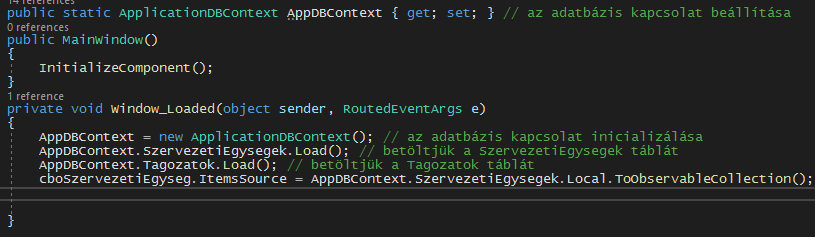


*15. ábra. Az elkészült adatbázis XAMPP MySql felületen*

## **3.1 Felület hozzárendelése az adatokhoz**

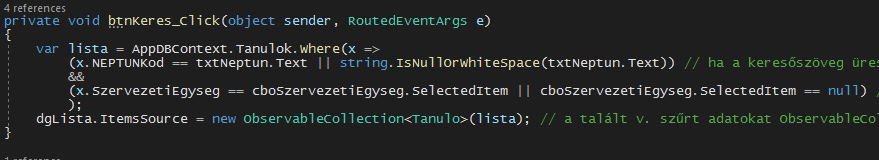
## **3.1.1 MainWindow.xaml.cs**

A klikkesemények és gombok valamint legördülő menük hozzárendelése az adatokhoz elkészítése a következő lépés. Nézzük először a MainWindow.xaml.cs fájlt. Elsőként megadtam az oldal tetején a Microsoft.EntityFrameworkCore névteret. Az ablakkezelő .xaml fájlban elhelyeztem egy „Loaded” new event – eseményt, ezzel párhuzamosan pedig a .cs fájlban, közvetlenül a konstruktor után létrehoztam egy eseményt. Ez azért fontos, hogy az ablak betöltődésével egyidejűleg a legördülő menü(-k) kapott adatai ne késsenek egy esetleges hosszabb adatbázis betöltődés / kapcsolat esetén sem. Fontos volt létrehozni egy statikus jellemzőt a DBContext-re is, amit a ’Loaded’ eseményen belül példányosítottam, így a MainWindow eléri az adatbázisban tárolt adatokat, és a ’Szervezeti Egységek’ ComboBox-ban láthatóvá és választhatóvá válnak az adott tábla adatai.



*16. ábra. Az adatbázis adatainak betöltése a fő ablakba*

Következő a „Keresés” gomb paramétereinek megadása volt. A metódus egy adatbázis-kapcsolatot tartalmazó AppDBContext változóban tárolt Tanulok táblából keres adatokat a megadott feltételek alapján. Az első feltétel azt ellenőrzi, hogy a felhasználó által megadott NEPTUN-kód (txtNeptun.Text) egyezik-e a Tanulok táblában található adatok egyik NEPTUN-kódjával vagy üres-e az érték. A második feltétel pedig azt vizsgálja, hogy a felhasználó által kiválasztott szervezeti egység (cboSzervezetiEgyseg.SelectedItem) egyezik-e a Tanulok táblában található adatok valamelyik SzervezetiEgyseg értékével, vagy nincs-e kiválasztva semmi.

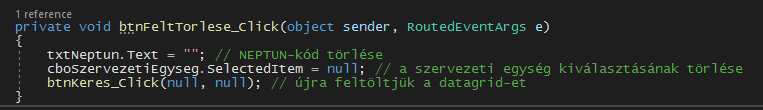


*17. ábra. Az „Keresés” gomb funkciója*

Ha mindkét feltétel igaz, akkor a metódus létrehoz egy ObservableCollection<Tanulo> típusú változót, amelybe beletölti a keresés eredményeként megtalált Tanulo típusú adatokat. Az ObservableCollection<Tanulo> egy olyan típus, amely lehetővé teszi az adatok dinamikus hozzáadását, eltávolítását vagy módosítását, így ez az ideális típus, amelyet az adatok tárolására és a DataGrid táblázat megjelenítésére használhatunk. Végül a metódus átadja az adatokat az ItemsSource tulajdonságán keresztül a DataGrid-nek (dgLista), amely megjeleníti az adatokat a felhasználó számára.

A harmadik kód a „Feltételek törlése” gombhoz kapcsolódik. Egy eseménykezelő metódus, amely akkor fut le, amikor a felhasználó rákattint a "btnFeltTorlese" gombra. A metódus három dolgot csinál:

* Az első sorban törli a "txtNeptun" nevű TextBoxban található NEPTUN-kódot.
* A második sorban a "cboSzervezetiEgyseg" ComboBox-ban kiválasztott elemet törli. Ha nincs elem kiválasztva, akkor az érték null lesz.
* Végül a metódus meghívja a "btnKeres\_Click" metódust null értékekkel, hogy újratöltse a táblázatot. Az átadott null értékek azt jelentik, hogy a "sender" és az "e" paraméterek értéke null lesz, és így az esemény argumentumai sem lesznek átadva a metódusnak. Azonban, mivel a "btnKeres\_Click" metódusnak nincs szüksége ezekre a paraméterekre, az átadott null értékek nem befolyásolják a metódus működését, ergo az egész táblázat újratölt a grid-ben teljes (updated) értékekkel.

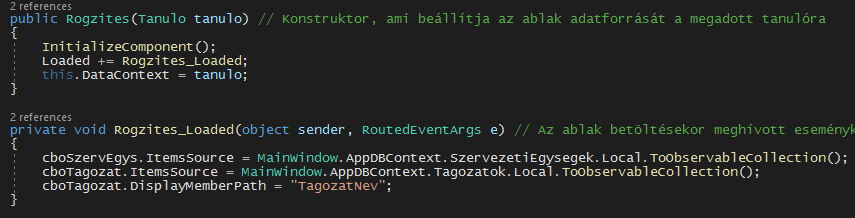


*18. ábra. Az „Feltételek törlése” gomb funkciója*

Ahhoz, hogy tovább lépjek a további gombok eseménykezelői kódolásához, a „Rögzítés” ablak eseményeit kellett kidolgoznom. (Rogzites.xaml.cs)

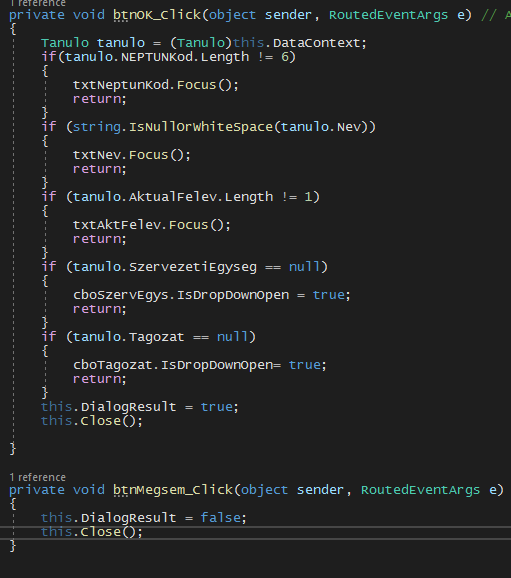
## **3.1.2 Rogzites.xaml.cs**

A konstruktor a "tanulo" nevű Tanulo objektumot várja paraméterként, és beállítja az ablak adatforrását erre az objektumra. Az adatforrás az ablakban található vezérlőkhez (például TextBox-okhoz és ComboBox-okhoz) köti a Tanulo objektum tulajdonságait, így amikor a felhasználó módosítja a vezérlők értékét, akkor a Tanulo objektum is módosul. Az osztályban található másik metódus a "Rogzites\_Loaded", amelyet az ablak betöltésekor hívódik meg. Ez a metódus betölti a "cboSzervEgys" és "cboTagozat" ComboBox-okat adatokkal az adatbázisból. A "MainWindow.AppDBContext" pedig közvetetten hivatkozva, mint a MainWindow.xaml.cs-ben, egy adatbázis-kapcsolatot reprezentáló objektum, amelyen keresztül az adatbázisból le lehet kérdezni adatokat. Az "ToObservableCollection" metódus egy ObservableCollection<T> objektumot hoz létre a lekérdezett adatokból, amelyet az ablak ComboBox-ja "ItemsSource" tulajdonságához köthetünk. A "cboTagozat.DisplayMemberPath" tulajdonsággal beállítjuk, hogy a ComboBox-ban a Tagozat objektumok "TagozatNev" tulajdonsága jelenjen meg a felhasználó számára.



*19. ábra. Az adatbázis adataira történő hivatkozás és betöltés funkciója*

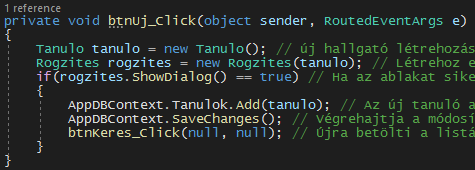
Alatta a kód, ami "Rögzítés" és "Mégsem" gombok eseménykezelőit tartalmazza, amelyek a Rögzítés párbeszédablakban jelennek meg. Az "OK" gomb eseménykezelője le ellenőrzi az ablakban lévő adatok érvényességét, és ha minden adat megfelelő, akkor a "DialogResult" tulajdonságát "true"-ra állítja, majd bezárja az ablakot. Ha valamelyik adat érvénytelen, akkor a fókusz a hibás adatra kerül, és az eseménykezelő visszatér. A "Mégsem" gomb eseménykezelője egyszerűen csak beállítja a "DialogResult" tulajdonságot "false"-ra, majd bezárja az ablakot. A "DialogResult" tulajdonsága arra szolgál, hogy az ablakot megjelenítő kód tudja, hogy a felhasználó mit választott az ablakban (pl. "OK" vagy "Mégsem"), és ennek megfelelően tudja folytatni a műveletet.



*20. ábra. A „Mentés” és „Mégsem” gombok eseménykezelői, a feltételekkel*

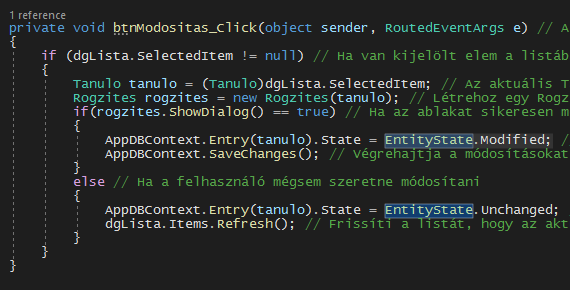
## **3.1.3 A MainWindow alsó három gombja és funkciója**

Ha a felhasználó rákattint az "Új" gombra, akkor létrehoz egy új ’Tanulo’ objektumot, amelyet a "Rogzites" ablak adatforrásaként használ. Ezzel automatikusan megnyitja a "Rogzites" ablakot, amely lehetővé teszi a felhasználó számára az új hallgató adatainak megadását, paramétere az új ’Tanulo’ objektum. Ha a felhasználó sikeresen beírta / kiválasztotta az adatokat és bezárta a "Rogzites" ablakot a "OK" gombra kattintva, akkor az új Tanulo objektumot hozzáadja az adatbázishoz, majd elmenti (SaveChanges) a módosításokat az adatbázisban. Végül az eseménykezelő frissíti a hallgatók listáját a frissített adatokkal.



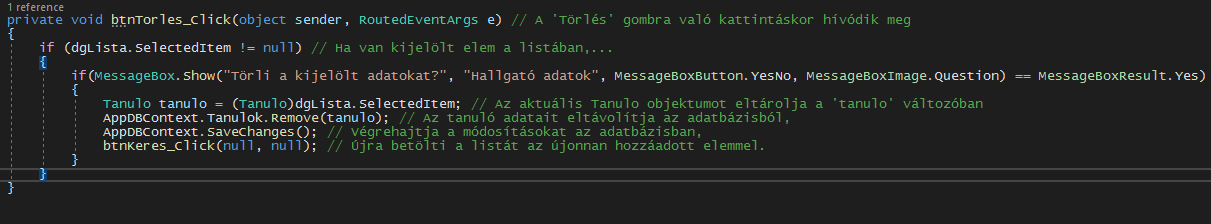
*21. ábra. Az „Új” gomb funkciója*

A „Módosítás” gomb eseménykezelő lefutásakor a függvény ellenőrzi, hogy van-e kijelölt elem a listában (dgLista). Ha van, akkor az aktuális Tanulo objektumot eltárolja a "tanulo" változóban, majd létrehoz egy "Rogzites" ablakot, amelynek paramétere az aktuális Tanulo objektum. Ha a felhasználó sikeresen megnyitja az ablakot, átír, módosít a felhasználó az adatokon, akkor az "Entry" metódus beállítja az aktuális Tanulo objektum állapotát "Módosítva" állapotra, majd az "SaveChanges" metódus végrehajtja a módosításokat az adatbázisban. Ha a felhasználó nem akar módosítani, akkor az állapotot "Változatlan" állapotra állítja, azaz nem történt változtatás, majd a MainWindow és adatbázis lista frissül az aktuális elem új vagy régi adataival.



*22. ábra. Az „Módosítás” gomb funkciója*

Utolsó gomb lent a „Törlés” gomb lett. Ez szintén egy metódust valósít meg, amelyet a btnTorles\_Click eseménykezelő hív meg, amikor a felhasználó rákattint a "Törlés" gombra. A kód ellenőrzi, hogy van-e kijelölt elem a dgLista (grid) adatrácsban. Ha van, akkor egy megerősítő ablakot jelenít meg, amelyben megkérdezi a felhasználót, hogy biztosan törölni akarja-e a kiválasztott adatokat. Ha a felhasználó megerősíti a törlést, akkor a kód eltávolítja az adott Tanulo objektumot az adatbázisból a Remove metódus segítségével. Ezután elmenti a változásokat az adatbázisban a SaveChanges metódus segítségével. Végül újra betölti a listát a btnKeres\_Click metódus meghívásával, amely egy új lekérdezést hajt végre a frissített adatokkal, így azok megjelennek a felhasználó számára a dgLista adatrácsban.



*23. ábra. Az „Törlés” gomb funkciója*

# 4. Az osztályok

*Tanulo.cs*

Ez az osztály egy adatbázis táblát reprezentál a "Tanulok" nevű entitáson belül. Az osztály tartalmazza a táblához kapcsolódó oszlopokat, amelyek az adatbázisban tárolt diák adatait tartalmazzák, mint például a diák NEPTUN kódját, nevét, szervezeti egységét, aktuális félévét, tagozatát és felvételének dátumát. Az osztály az Entity Framework Code First funkcióit használja a tábla létrehozására és az adatok kezelésére.

*Tagozat.cs*

Ez az osztály egy adatbázis táblát reprezentál a Tagozatok nevű entitáson belül. Az osztály tartalmazza a táblához kapcsolódó oszlopokat, amelyek az adott tagozat nevét tartalmazzák. Az osztály az Entity Framework Code First funkcióit használja a tábla létrehozására és az adatok kezelésére, és az adatok érvényességének ellenőrzésére.

*SzervezetiEgyseg.cs*

Ez egy C# osztály a Tanulok névtérben, amely az adatbázis SzervezetiEgysegek nevű táblájához tartozó adatokat tárolja. Az osztályban találhatóak olyan tulajdonságok, mint az elsődleges kulcsot reprezentáló Id és a SzervezetiEgyseg nevét tároló SZeNev tulajdonság.

*Rogzites.xaml.cs*

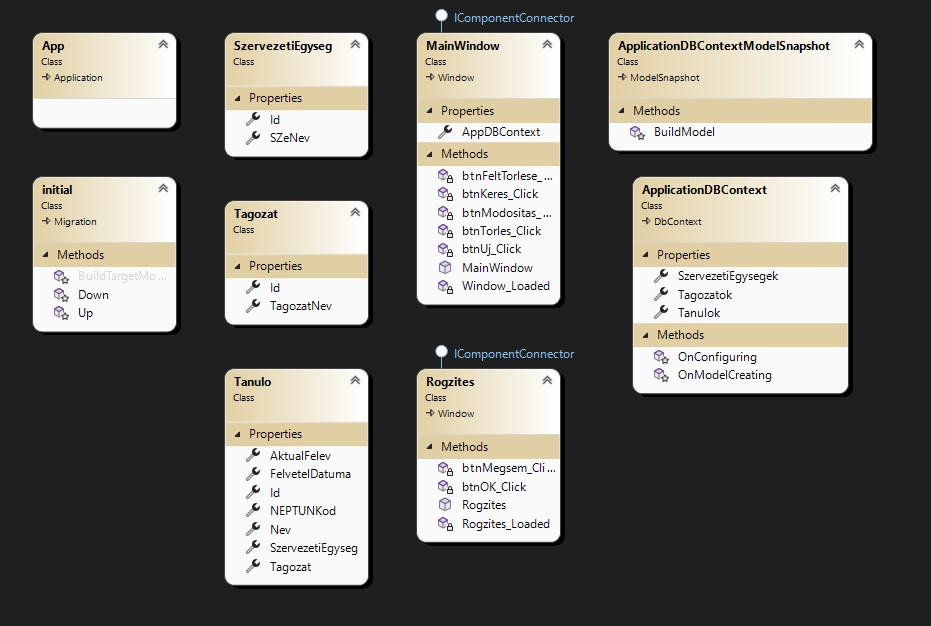
Ez egy C# osztály, amely az alkalmazás felhasználói felületének egy ablakját reprezentálja. Az ablak a Tanulok névtérben található, és lehetővé teszi egy adott tanuló adatainak szerkesztését vagy új tanuló hozzáadását az adatbázishoz. Az osztály az Entity Framework Core-t használja az adatbázis eléréséhez, és az ablak elemeként ComboBoxokat tartalmaz, amelyeket adatokkal tölt fel az adatbázisból. Az osztályban van egy konstruktor, amely beállítja az ablak adatforrását a megadott tanulóra, valamint két gomb eseménykezelője: az egyik a rögzítést végzi, ellenőrzi az adatok érvényességét és bezárja az ablakot, ha minden mező kitöltve; a másik pedig bezárja az ablakot anélkül, hogy elmentené az adatokat.

*MainWindow.xaml.cs*

Az "MainWindow" osztály az alkalmazás főablakát reprezentálja, és a felhasználói felület kialakítását, valamint az adatbázis-kezelést végzi. Az osztály két adatbázistáblát, a "SzervezetiEgysegek" és "Tagozatok" táblákat betölti, majd ezekből feltölti a megfelelő vezérlőelemeket (pl. ComboBox, DataGrid) a felhasználói felületen. A "Tanulok" táblából keresést végez a felhasználó által megadott keresőszöveg és a kiválasztott szervezeti egység alapján, majd az eredményt megjeleníti a DataGrid-ben. Az új tanulók felvételét, valamint a meglévő tanulók módosítását / törlését is lehetővé teszi a felhasználói felületen található gombok segítségével. Az osztály fontos része az adatbázis-kezelési függvények implementálása, amelyek az Entity Framework Core használatával történnek.

*ApplicationDBContext.cs*

Ez a kód egy C# osztályt tartalmaz, amely az Entity Framework Core - DbContext osztályából származik. Az osztály egy adatbázis kontextust definiál az alkalmazás számára, és tartalmazza az adatbázis entitások / egyedek DbSet objektumait, amelyek a táblákhoz tartoznak. Az OnConfiguring metódusban az adatbázis kapcsolatához szükséges konfiguráció kerül beállításra, míg az OnModelCreating metódusban az adatbázis entitások modelljéhez szükséges konfiguráció található. Az osztály tartalmaz kezdőadatokat is, amelyek az adatbázisba betöltésre kerülnek, amikor az alkalmazás először indítja el az adatbázis kontextust.

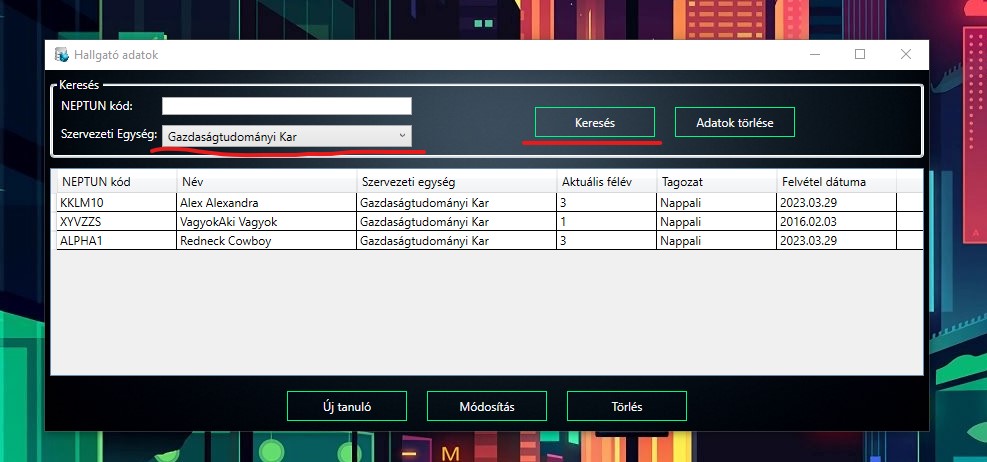


*24. ábra. A projekt osztály diagrammja*

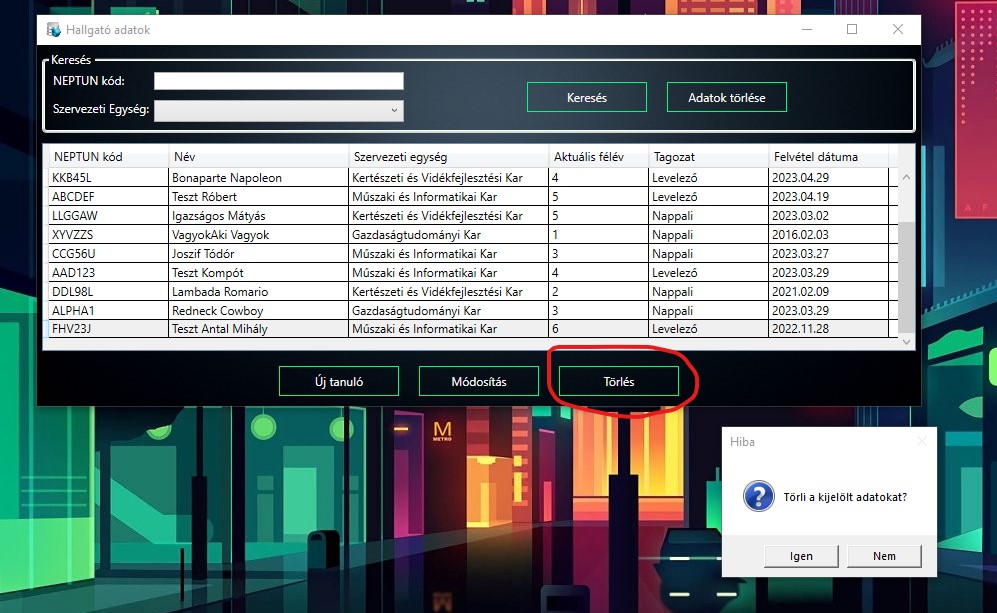
Az űrlap teteje

# 5. Az eredmény:

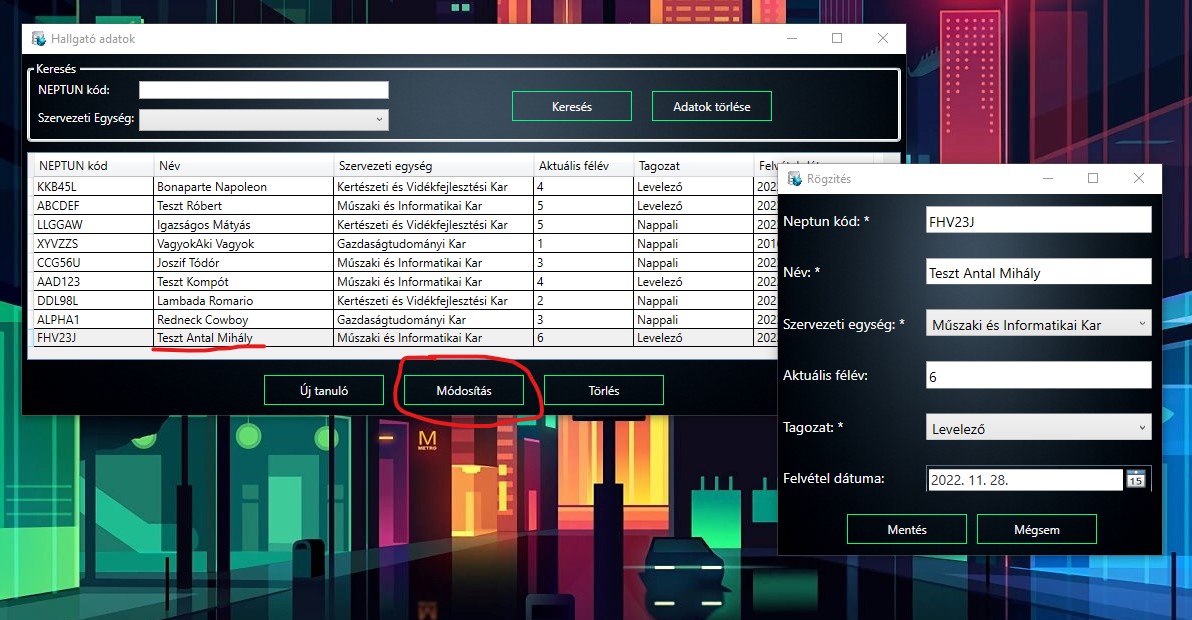
**Keresés, legördülő menüből választással:**



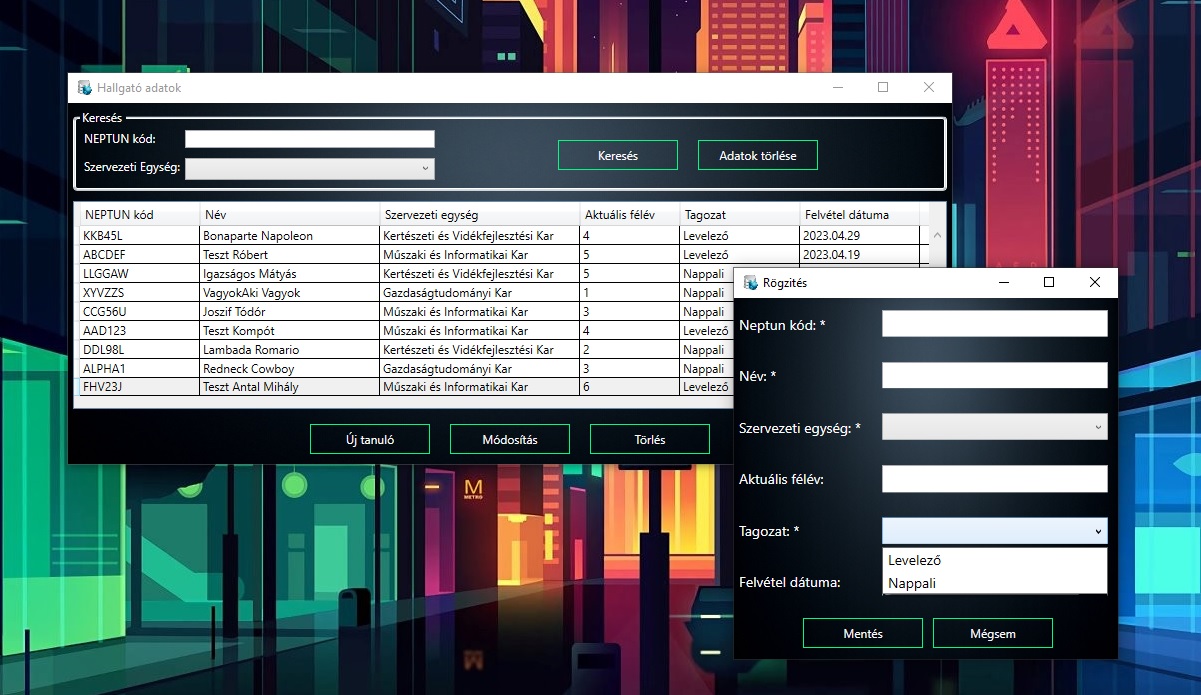
**Törlés funkció, a felugró ablakkal:**



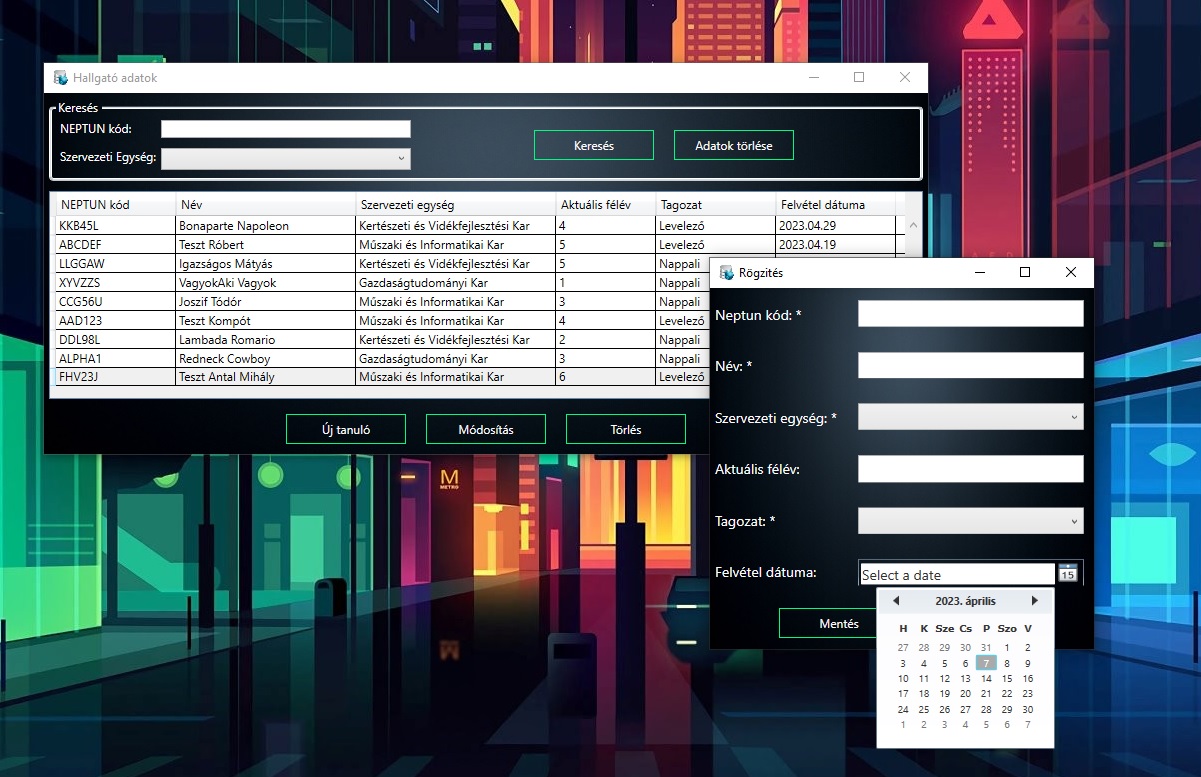
**Módosítás:**



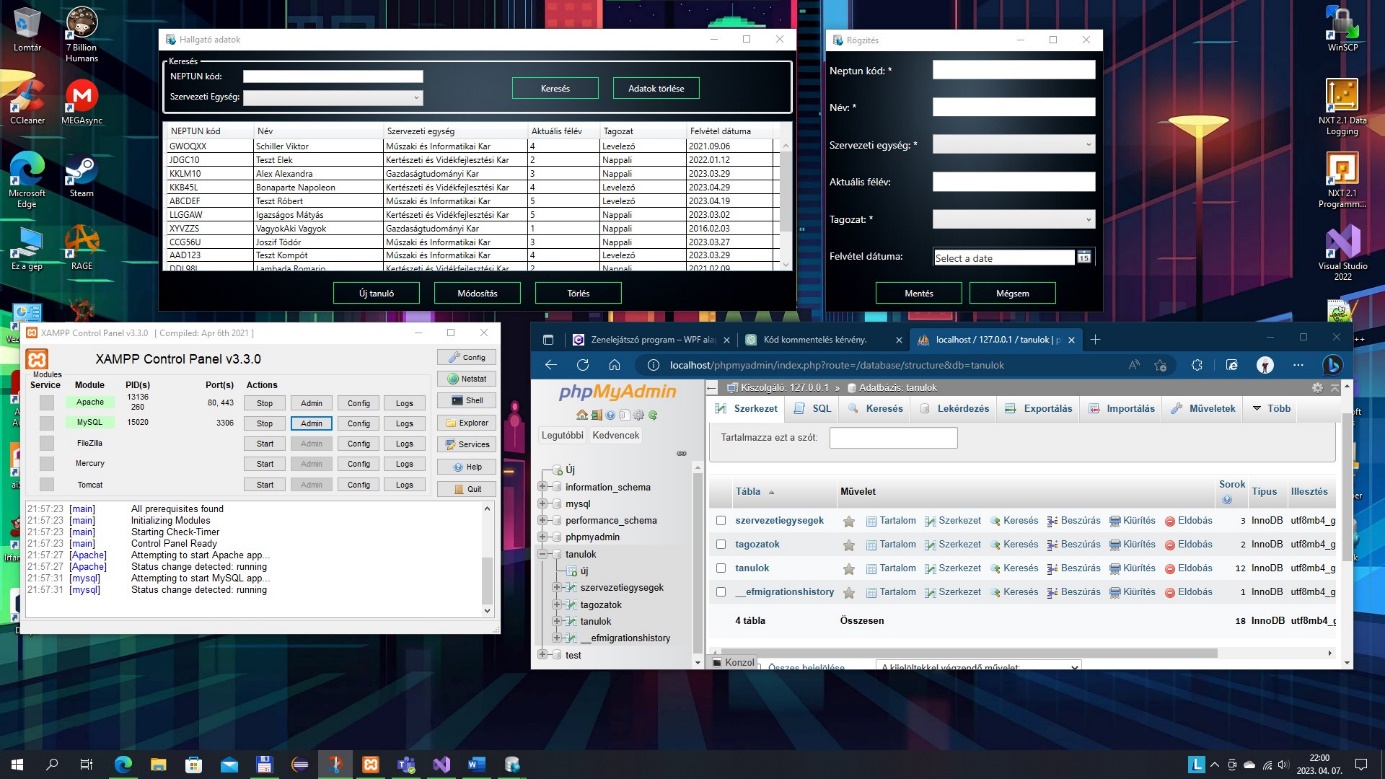
**Új tanuló rögzítése, az egyik lenyíló menüvel:**



**Dátum kiválasztás:**



**Minden működik 😊:**

****

FELHASZNÁLT IRODALOM & KÉPEK

* XAMPP letöltés [XAMPP Installers and Downloads for Apache Friends](https://www.apachefriends.org/hu/index.html)
* WPF Documentation [Windows Presentation Foundation for .NET 7 documentation | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-7.0)
* [Data Binding in WPF DataGrid Control Using MySQL Database (c-sharpcorner.com)](https://www.c-sharpcorner.com/UploadFile/47548d/data-binding-in-wpf-datagrid-control-using-mysql-database/)
* 1. kép https://images.app.goo.gl/rTh5zHKjBwNwjtmb6