**Szegedi Tudományegyetem**

**Informatikai Intézet**

**SZAKDOLGOZAT**

**Lengyel Norbert Péter**

**2025**

**Szegedi Tudományegyetem**

**Informatikai Intézet**

**Kisvállalkozói Költségvetés Nyilvántartó Rendszer**

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Lengyel Norbert Péter |  | Dr. Bodnár Péter |  |
|  | informatika szakos hallgató |  | egyetemi docens |  |

Szeged

2025

# ***Feladatkiírás***

A kisvállalkozások nagy része nem használ strukturált rendszert pénzügyeinek követésére, pedig a kiadások és bevételek pontos nyilvántartása alapvető a hosszú távú működéshez. A feladat egy olyan költségvetés-kezelő asztali alkalmazás megvalósítása, amely segíti a kisvállalkozásokat napi pénzügyi adataik egyszerű és átlátható vezetésében.

A rendszer lehetőséget nyújt a bevételek és kiadások rögzítésére, kategorizálására, keresésére és szerkesztésére. Cél, hogy a felhasználó gyorsan átláthassa vállalkozása pénzügyi helyzetét, melyben grafikonok segítik az értelmezést és a statisztikák megjelenítését. Fontos, hogy az alkalmazás felhasználóbarát legyen, könnyen kezelhető felülettel, és megfelelően működjön különböző Windows eszközökön.

A megoldás bővíthetősége nagy potenciállal rendelkezik, mely funkciók közül a későbbiekben párat fel is sorolok.

## ***Tartalmi összefoglaló***

* A téma megnevezése:

Kisvállalkozói Költségvetés Nyilvántartó Rendszer

* A megadott feladat megfogalmazása:

A cél egy, olyan asztali alkalmazás megvalósítása, mely megkönnyíti egy kisvállalkozó bevételeinek és kiadásainak a nyomonkövetesét és lehetővé teszi statisztikai kimutatások megvalósítását dinamikusan a megadott keretek között.

* A megoldási mód:

Windows keretrendszerre fejlesztett asztali alkalmazás elkészítése – minimális külsős library-k/modulok alkalmazásával, amely modern köntösbe foglalja a mai világban szükséges költségvetés munkamenetét kisvállalkozók részére.

* Alkalmazott eszközök, módszerek:

A fejlesztés Visual Studio 2022-ben történt. A projekt .NET keretrendszerben készült, amely egy széles körben használt, robusztus platform asztali és webes alkalmazások fejlesztéséhez. A felhasználói felület kialakításához a WPF-et (Windows Presentation Foundation) használtam, amely lehetőséget ad modern, reszponzív asztali alkalmazások készítésére XAML-alapú felületdefiniálással. A fejlesztési logika MVVM mintára épül, mely segít az üzleti és a megjelenítési logikát tisztán elválasztani a felhasználói felülettől (UI).

* Elért eredmények:

Egy Windows-alapú asztali alkalmazás készült, amely lehetőséget biztosít költségvetési adatok sokoldalú és látványos statisztikai elemzésére különböző típusú diagramok segítségével.

* Kulcsszavak:

asztali alkalmazás, WPF, .NET, MVVM, költségvetés, diagramm, statisztikai kimutatások

**Tartalomjegyzék**

[***Feladatkiírás*** 4](#_Toc198531333)

[***Tartalmi összefoglaló*** 5](#_Toc198531334)

[***Bevezetés*** 8](#_Toc198531335)

[**1.1.** **.NET** 9](#_Toc198531336)

[**1.2.** **WPF** 9](#_Toc198531337)

[**1.3.** **MVVM** 9](#_Toc198531338)

[**2.** **Definíciók** 10](#_Toc198531339)

[**3.** **Adatbázis tervezése** 10](#_Toc198531340)

[**3.1.** **MySQL** 10](#_Toc198531341)

[**3.2.** **Adatbázis kapcsolatok** 11](#_Toc198531342)

[**3.3.** **Táblák** 11](#_Toc198531343)

[**3.3.1.** **Dolgozók** 11](#_Toc198531344)

[**3.3.2.** **Felhasználók** 11](#_Toc198531345)

[**3.3.3.** **Diagrammok** 12](#_Toc198531346)

[**3.3.4.** **Magán személyek** 12](#_Toc198531347)

[3.3.5. **Gazdálkodó** **szervezetek** 13](#_Toc198531348)

[**3.3.6.** **Kötelezettségek és Követelések** 13](#_Toc198531349)

[**3.3.7.** **Költségvetés** 13](#_Toc198531350)

[**4.** **Alkalmazás működése** 14](#_Toc198531351)

[**4.1.** **Bejelentkezés** 14](#_Toc198531352)

[**4.1.1. UI** 14](#_Toc198531353)

[**4.1.2. Authentication/Felhasználó Hitelesítése** 15](#_Toc198531354)

[**4.1.3. Főoldal szerepkörének beállítása** 15](#_Toc198531355)

[**4.2. Navigációs menü** 16](#_Toc198531356)

[**4.3. Főoldal** 18](#_Toc198531357)

[**4.3.1. Adatok megjelenítése/listázása - DataGrid** 19](#_Toc198531358)

[**4.3.2. Törlés** 21](#_Toc198531359)

[**4.3.3. Módosítás** 24](#_Toc198531360)

[**4.3.4. Új rekord felvétele** 25](#_Toc198531361)

[**4.3.5. DataGrid adat exportálás Excel formátumba** 26](#_Toc198531362)

[**4.3.6. DataGrid Keresés/Szűrés** 28](#_Toc198531363)

[**4.4. Diagram létrehozása** 29](#_Toc198531364)

[**4.4.1. Diagram típus kiválasztása** 29](#_Toc198531365)

[**4.4.2. Szűrési lehetőségek** 30](#_Toc198531366)

[**4.4.3. Csoportosítási lehetőségek** 31](#_Toc198531367)

[**4.4.4. Adatok kiválasztása megjelenítéshez** 33](#_Toc198531368)

[**4.4.5. Diagram megjelenítése** 35](#_Toc198531369)

[**4.4.6. Diagram exportálása** 39](#_Toc198531370)

[**4.4.7. Diagram mentése** 41](#_Toc198531371)

[**4.5. Létező diagrammok kártyás megjelenítése** 45](#_Toc198531372)

[**4.5.1. Kártyás megjelenítés** 45](#_Toc198531373)

[**4.5.2. Diagram visszatöltése/szerkesztése** 46](#_Toc198531374)

[**5.** **Speciális osztályok** 48](#_Toc198531375)

[**5.1. DateTimeParser** 48](#_Toc198531376)

[**5.2. PasswordHasher** 49](#_Toc198531377)

[**5.3. Mediator** 50](#_Toc198531378)

[**5.4. DataStatistics** 50](#_Toc198531379)

[**5.5. WindowHelper** 51](#_Toc198531380)

[**6. Bővíthetőségek** 52](#_Toc198531381)

[***Irodalomjegyzék*** 53](#_Toc198531382)

[***Nyilatkozat*** 54](#_Toc198531383)

# ***Bevezetés***

A szakdolgozat célja egy olyan asztali alkalmazás fejlesztése volt, amely segítséget nyújt kisvállalkozók számára a költségvetés nyilvántartásában, valamint statisztikai kimutatások készítésében. Az alkalmazás lehetőséget biztosít különféle diagramok létrehozására, amelyek .png formátumban exportálhatók, így könnyen beilleszthetők prezentációkba vagy dokumentumokba. A fejlesztés során különös figyelmet fordítottam arra, hogy a felhasználók számára minél több testreszabási és megjelenítési lehetőséget biztosítsak, ezáltal lehetővé téve komplex, vizuálisan jól értelmezhető statisztikák létrehozását.

A fejlesztés első lépéseként utánajártam, milyen WPF-kompatibilis könyvtárak vagy komponensek állnak rendelkezésre diagramok megjelenítésére. A főbb szempontok között szerepelt, hogy a választott megoldás ingyenesen felhasználható legyen, széles körű testreszabhatóságot biztosítson, valamint minél többféle diagramtípust támogasson. A különböző lehetőségek mérlegelése után a választásom a LiveCharts könyvtárra esett, mivel megfelelt ezeknek a kritériumoknak, és jól integrálható a WPF környezetbe.

Az elkészült alkalmazás egyik központi eleme a navigációs menü, amely lehetővé teszi az adatbázisban tárolt különféle adatok (dolgozók, gazdálkodási egységek, magánszemélyek, költségvetési tételek – bevételek és kiadások –, kötelezettségek és követelések) elérését és megjelenítését a főablakban. A rendszerben a legtöbb adattípushoz teljes CRUD funkcionalitás tartozik, azaz a felhasználók képesek az adatok létrehozására, módosítására, törlésére és megtekintésére.

A táblázatos formában listázott adatok szűrhetőek, így a felhasználó gyorsan megtalálhatja a számára releváns információkat. A szűrés hatékonyságát növeli, hogy az aktuálisan lekért adatlista a memóriában kerül eltárolásra, így a keresések nem terhelik feleslegesen az adatbázist, és a működés gördülékenyebb.

A program leglátványosabb és egyben legfontosabb funkciója a statisztikai adatok vizuális megjelenítése különféle diagramok segítségével. A megvalósítás alapját a LiveCharts könyvtár biztosítja, amely számos diagramtípust támogat, többek között a LineSeries, PieSeries, RowSeries, ColumnSeries és StackedColumnSeries osztályokon keresztül. A különböző nézetek és csoportosítások megjelenítéséhez a háttérben dinamikus adatfeldolgozás történik, amely lehetővé teszi, hogy az adatok többféle szempont szerint rendszerezhetők és vizualizálhatók legyenek, így biztosítva a diagramok sokoldalúságát és testreszabhatóságát.

Ezek mellett fontosnak találtam, hogy a felhasználónak lehetősége legyen az adatok exportálására is, ezért az alkalmazás támogatja a táblázatok Excel formátumban történő mentését.

1. **Fejlesztői környezet és keretrendszer választása**

A fejlesztéshez olyan környezetet és technológiákat kívántam választani, amelyekben már rendelkezem tapasztalattal, és amelyek jól illeszkednek az asztali alkalmazás fejlesztésének követelményeihez. A cél egy stabil, jól strukturált, karbantartható és felhasználóbarát rendszer létrehozása volt. Ennek érdekében az alábbi technológiák mellett döntöttem:

### **.NET**

A .NET keretrendszer kiválasztását elsősorban a stabilitása, érettsége és széles körű támogatottsága indokolta. A .NET rendkívül jól integrálható különféle technológiákkal, és kiválóan alkalmas Windows-alapú asztali alkalmazások fejlesztésére. Emellett előnye a kiváló dokumentáció, valamint a folyamatos Microsoft támogatás, amely hosszú távon is biztosítja a rendszer megbízhatóságát. Fontos szempont volt az is, hogy a .NET nyelvi szinten támogatja a C#-ot, amelyben már előzetes tapasztalattal rendelkeztem.

### **WPF**

Az alkalmazás felhasználói felületét WPF-ben (Windows Presentation Foundation) valósítottam meg, mivel ez a technológia kifejezetten asztali alkalmazásokhoz készült, és lehetővé teszi modern, reszponzív és vizuálisan vonzó UI kialakítását. A WPF előnye, hogy XAML-alapú felületleírást használ, ami nagyfokú elválasztást tesz lehetővé a megjelenítés és az üzleti logika között. Továbbá könnyen integrálható adatforrásokkal és támogatja az adatközpontú megközelítést, ami különösen előnyös volt az általam fejlesztett statisztikai és táblázatos funkciók szempontjából.

### **MVVM**

A fejlesztés során az MVVM (Model-View-ViewModel) tervezési mintát alkalmaztam, amely kifejezetten jól illeszkedik a WPF struktúrájához. Az MVVM alkalmazása lehetővé tette az üzleti logika, a megjelenítés és az adatkezelés szétválasztását, így a rendszer könnyebben karbantartható és tesztelhető lett. Emellett az MVVM elősegítette a kód újrafelhasználhatóságát és a moduláris felépítést, ami különösen fontos volt a rendszer bővíthetősége szempontjából. A ViewModel réteg segítségével egyszerűen valósíthattam meg a felhasználói interakciók és az adatok közötti kapcsolatot, így a felhasználói élmény gördülékeny és jól kontrollált lett.

# **Definíciók**

Bevétel: A kisvállalkozó felé befolyt összeg, ami már ki lett fizetve. A bevétel származhat magánszemélytől vagy akár gazdálkodó szervezettől.

Kiadás: A kisvállalkozó által kifizetett összeg, ami már ki lett fizetve. A kiadás megtörténhet magánszemély, illetve gazdálkodó szervezet felé is.

Kötelezettség: Olyan kiadása a kisvállalkozónak, ami még megtervezett kiadás vagy csoportos kiadás, aminek még nincs a teljes összege kifizetve.

Követelés: Olyan bevétel, amire a kisvállalkozó számít, de még nem folyt be az összeg.

Adatsor: Azoknál a diagrammoknál, ahol ez a lehetőség fennáll a diagram felépítése a háttérben a következő: például a RowSeries property egy SeriesCollection, amiben RowSeries osztály példányosításával hozunk létre adatsorokat. Például, ha Pénznem és BeKiKód alapján csoportosítok adatokat, ahol a Pénznem alapján készülnek a címkék és a BeKiKód alapján az adatsorok, akkor 4 RowSeries példányosítás történik a BeKiKód alapján, ahol a Value értéke egy ChartValues<double> lista 4 értékkel reflektálva a Pénznemre.

Címke: Diagramok megjelenítéséhez szükséges, például oszlop diagrammok esetén az oszlopok alatti szöveg. Ha például Pénznem alapján készítjük el a címkézést, akkor 4 címke fölött fognak az oszlopok megjelenni (Forint, Dollár, Font, Euró). Ez a diagramok értelmezésében kulcs szerepet játszik.

Az Adatsor és a Címke közötti kapcsolatot el lehet képzelni akár egy mátrix formájában, ahol az egyik az oszlopokat míg a másik a sorokat írja le.

# **Adatbázis tervezése**

## **MySQL**

Az adatbázis kialakítása kulcsszerepet játszik az alkalmazás működésében, ezért a megfelelő adatbáziskezelő rendszer kiválasztására különös figyelmet fordítottam. Első lépésként áttekintettem azokat az adatbázis-technológiákat, amelyekkel korábban már dolgoztam. A döntés során két opció – a MySQL és a Firebase – merült fel komoly lehetőségként.

Végül a választásom a MySQL-re esett, két fő indok alapján. Egyrészt, ez az a rendszer, amellyel a legtöbb fejlesztési tapasztalatom van, így a használata során gyorsabban tudtam haladni, és hatékonyabban tudtam elvégezni a szükséges szerkezeti módosításokat. Másrészt, a fejlesztés során gyakran előfordulnak olyan változtatási igények, amelyek a relációs adatbázis szerkezetének módosítását teszik szükségessé. A MySQL esetében ez egyszerűen megvalósítható, különösen XAMPP segítségével, amely lehetővé teszi a MySQL adatbázisok lokális futtatását és tesztelését – korlátozások és internetkapcsolat nélkül. Ez a rugalmasság rendkívül előnyösnek bizonyult a fejlesztési és tesztelési fázis során.

## **Adatbázis kapcsolatok**

A képen szöveg, képernyőkép, szoftver, Weblap látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**3.2.1. ábra, adatbázis tábla kapcsolatok**

## **Táblák**

### **Dolgozók**

A „dolgozok” tábla azon személyek adatait tartalmazza, akik jogosultak az alkalmazás használatára. Kisvállalkozói környezetben ezek a rekordok jellemzően a vállalkozás alkalmazottait reprezentálják. Az azonosító mező (ID) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak manuális kezelése a felhasználói oldalról nem szükséges. Ez egyszerűsíti az adatfelvitelt és biztosítja az egyedi azonosítók konzisztenciáját a rendszerben.

### **Felhasználók**

A „felhasznalok” tábla azon felhasználói fiókok adatait tartalmazza, akik jogosultak az alkalmazás használatára. A tábla tárolja a bejelentkezéshez szükséges felhasználónevet (*felhasznalonev*) és jelszót (*jelszo*), valamint a „*dolgozo\_id*” külső kulcs attribútumot. Ez a külső kulcs biztosítja a dolgozói adatok és a belépéshez szükséges adatok közötti kapcsolatot. A felhasználót azonosító mező (*id*) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak kezelése nem igényel egyéb erőfeszítést. Az felhasználói szerepeket pedig a szerep attribútumban tárolom.

### **Diagrammok**

A „diagrammok” tábla azokat a diagrammokat tartalmazza, amiket a felhasználók hoztak létre. A tábla elsődleges kulcsát képező azonosító mező (*id*) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak kézi kezelése nem szükséges. A diagrammok a következő alap adatokat tárolják: név (*nev*), leírás (*leiras*), létrehozás dátuma (*letrehozasi\_datum*), módosítás dátuma (*modositasi\_datum*).

Ezeken kívül a komplexebb adatok a következőek:

* *diagram\_tipus* – az alapján kerül beállításra, hogy milyen típusú diagrammot hozott létre a felhasználó
* *adathalmaz* – két táblázat adatai alapján lehet létrehozni diagrammokat (Koltsegvetes – bevetelek\_kiadasok, KotelezettsegKoveteles – kotelezettsegek\_kovetelesek)
* *diagram\_ertekek\_adat* – azokat az adatsorokat tárolja, amik felépítik az adott diagrammot
* *szuresi\_beallitasok* – azok a szűrési feltételek, amiket használt a felhasználó, amikor létrehozta az adott diagrammot
* *csoportositasi\_beallitasok* – azok a csoportosítási beállítások, amiket a felhasználó kiválasztott és amik megadják, hogy a diagram milyen csoportosítások alapján jelenítse meg a kiválasztott adatokat
* *kijelolt\_diagram\_csoportositasok* – a felhasználó által megadott csoportosítási beállításokból azokat tárolja, amiket a felhasználó megjelenít a diagrammon
* *kijelolt\_elemek\_ids* – azoknak az adatoknak az id-ja, amiket a felhasználó kiválasztott
* *adat\_statisztika* – az a felhasználó által kiválasztott statisztikai megnevezés, amit szeretne megjeleníteni a diagrammon

Mindezek mellett a *letrehozo\_id* egy külsős kulcs, ami összeköti a diagrammot a felhasználó *id*-jával.

### **Magán személyek**

A „magan\_szemelyek” tábla azon magánszemélyek adatait tartalmazza, akik a kisvállalkozó által valamilyen szolgáltatásokat igénybe veszik, és ezekért fizetnek. A tábla elsődleges kulcsát képező azonosító mező (ID) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak kézi kezelése nem szükséges. A tárolt adatok közé tartozik a magánszemély neve, valamint elérhetőségi adatai. A fizikai elérhetőséget a lakcím biztosítja, míg az elektronikus kapcsolattartás céljából telefonszám és e-mail cím is rögzítésre kerül.

### **Gazdálkodó** **szervezetek**

A „gazdalkodo\_szervezetek” tábla azon gazdálkodási szervezetek adatait tartalmazza, akik a kisvállalkozó által valamilyen szolgáltatásokat igénybe vesznek, és ezekért fizetnek. A tábla elsődleges kulcsát képező azonosító mező (ID) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak kézi kezelése nem szükséges. A tárolt adatok közé tartozik a gazdálkodási intézmény neve, valamint a kapcsolattartó nevét és elérhetőségi adatait. Ebben az esetben csak az elektronikus elérhetőségek tárolására van szükség.

### **Kötelezettségek és Követelések**

A „kotelezettsegek\_kovetelesek” tábla azon költségvetési adatokat tartalmazza, melyek még nincsenek teljesítve a kisvállalkozó vagy a magánszemély, illetve a gazdálkodási szervezet részéről. Az azonosító mező (ID) automatikusan generálódik az adatbázis beállításainak megfelelően, így annak manuális kezelése a felhasználói oldalról nem szükséges. Ez egyszerűsíti az adatfelvitelt és biztosítja az egyedi azonosítók konzisztenciáját a rendszerben. Az azonosító mellett tárolom továbbá a követelés/kötelezettség típusát, összegét, pénznemét, a kifizetés határidejét, illetve, hogy ki van-e fizetve.

### **Költségvetés**

A „bevetelek\_kiadasok” tábla a kisvállalkozás pénzügyi nyilvántartásában szereplő, már teljesített bevételeket és kiadásokat tartalmazza. Ezek a tranzakciók származhatnak magánszemélytől, gazdálkodó szervezettől, vagy a vállalkozástól magától. A tábla elsődleges kulcsát képező azonosító mező (ID) automatikusan generálódik, amely egyszerűsíti az adatfelvitelt, és biztosítja az egyedi azonosítók konzisztens kezelését a rendszerben.

A tárolt adatok közé tartozik a tranzakció típusa (bevétel vagy kiadás), összege, pénzneme, valamint a teljesítés dátuma. Ezen túlmenően több külső kulcs is található a táblában, amelyek lehetővé teszik az adatok összekapcsolását más entitásokkal:

* kotel\_kovet\_id – kapcsolat a kötelezettségek és követelések táblával,
* gazdalkodo\_szerv\_id – kapcsolat a gazdálkodó szervezetek táblával,
* magan\_szemely\_id – kapcsolat a magánszemélyek táblával.

# **Alkalmazás működése**

A UI 6 főbb részből áll:

* bejelentkezéshez szükséges LoginView
* a főoldal, amin a navigáció és a fő tartalom megjelenítés történik MainView
* navigációs menü
* módosításhoz és törléshez szükséges ablakok
* kártyás megjelenítő a létező diagrammokhoz
* egy diagram létrehozásához szükséges ablak CreateChartsView

## **Bejelentkezés**

### **4.1.1. UI**

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.1.1. ábra, belépés ablak**

A bejelentkező felületen két beviteli mezőt (felhasználónév és jelszó)valamit egy „Bejelentkezés” gombot tartalmaz. Mindkét mező elfogad bármilyen karaktert, ugyanakkor a jelszó mező csillag karaktereket jelenít meg minden bevitt karakter helyett, így biztosítva a jelszó bizalmasságát. Abban az esetben, ha a bejelentkezés nem volt sikeres a megadott felhasználónév-jelszó párossal, akkor egy hibaüzenet jelenik meg.

### **4.1.2. Authentication/Felhasználó Hitelesítése**

A bejelentkezés során megadott adatok (felhasználónév és jelszó) átadásra kerülnek a UserRepository.AuthenticateUser metódusnak, amely az adatbázisban tárolt adatokkal veti össze azokat. Az eredmény egy bool érték (*isValidUser*), amely megmondja, hogy a hitelesítés sikeres volt-e, illetve visszatér a bejelentkezett felhasználó azonosítójával.

private void ExecuteLoginCommand(object obj)

{

Thread.CurrentPrincipal = new GenericPrincipal(

new GenericIdentity(Username), null);

IsViewVisible = true;

bool isValidUser = false;

int userId = 0;

(isValidUser, UserID, UserRole) = \_userRepository.AuthenticateUser(new NetworkCredential(Username, Password));

if (isValidUser)

{

Thread.CurrentPrincipal = new GenericPrincipal(

new GenericIdentity(Username), null);

IsViewVisible = false;

}

else

{

ErrorMessage = "\* Érvénytelen felhasználónév vagy jelszó";

}

}

Amint ahogy az a fenti kódrészletből látható, az *IsViewVisible* változó értéke a hitelesítés sikerességétől függően módosul. Ennek a változónak a szerepe a következő pontban kerül részletesebben kifejtésre.

### **4.1.3. Főoldal szerepkörének beállítása**

Amit tudni kell az IsViewVisible property-ről, hogy ennek értéke tükrözi a LoginView láthatóságát. A program indulásakor a főablak szerepkört a LoginView kapja, amely szerepkört szükséges átadni a MainView-nak sikeres bejelentkezés esetén annak érdekében, hogy a MainView ablak bezárása valóban az egész alkalmazás futását letudja állítani.

protected void ApplicationStart(object sender, StartupEventArgs e)

{

var loginView = new LoginView();

loginView.Show();

loginView.IsVisibleChanged += (s, ev) =>

{

if (loginView.IsVisible == false && loginView.IsLoaded)

{

try

{

int userID = 0;

string role = "";

if (loginView.DataContext is LoginViewModel viewModel)

{

userID = viewModel.UserID;

role = viewModel.UserRole;

}

var mainView = new MainView(userID, role);

mainView.Show();

loginView.Close();

}

catch { }

}

};

}

A *LoginView.IsVisibleChanged* eseményre történő feliratkozás lehetővé teszi, hogy a rendszer érzékelje a bejelentkezési felület eltűnését, és automatikusan megnyissa a *MainView* ablakot. Ez biztosítja, hogy a LoginView ne maradjon aktív vagy nyitva a háttérben a bejelentkezés után, így az alkalmazás működése átláthatóbb és stabilabb lesz.

## **4.2. Navigációs menü**

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.2.1. ábra, navigációs menü**

A navigációs menü működését *RadioButton* vezérlőelemek biztosítják, amelyek mindegyike egy-egy külön commandot futtat le. A vezérlők egy csoportba tartoznak (*GroupName="Content"*), így egyszerre mindig csak egy lehet kijelölve.

A navigációs menü struktúráját az *Expander* vezérlőelem biztosítja, amely tárolja a menüpontokat megjelenítő *RadioButton*-öket. Bár jelenleg a menü egyszintű, az Expander használata lehetővé teszi a menüpontok logikai csoportosítását, valamint biztosítja a jövőbeni bővíthetőséget, például többszintű vagy kategorizált menük létrehozásával. Emellett az Expander funkciója révén a menücsoportok ki- és becsukhatók, ami átláthatóbb és rendezettebb felhasználói felületet eredményez.

Amennyiben egy *RadioButton* aktívvá válik, a hozzá tartozó szöveg színe magenta színűre változik, valamint a gomb bal oldalán megjelenik egy magenta színű keret (*border*). Ez vizuálisan is jelzi a kiválasztott állapotot a felhasználónak.

A kiválasztás során a megfelelő *UserControl* betöltését egy command biztosítja, amely a *CurrentChildView* értékét állítja be a kiválasztott nézet ViewModel példányára.

Példa egy *Expander* XAML kódjára:

<Expander x:Name="ExpanderMenuDiagrammok" Header="Diagrammok" IsExpanded="False" Width="250" Background="{x:Null}" Foreground="White" FontSize="13.5" HorizontalAlignment="Center">

<StackPanel>

<RadioButton GroupName="Content" Style="{StaticResource menuButton}"

Tag="#E285DE"

IsChecked="True"

Command="{Binding ShowUjDiagrammokViewCommand}">

<StackPanel Orientation="Horizontal">

<TextBlock Text="Új diagramm létrehozása" Style="{StaticResource menuButtonText}" Margin="30,0,0,0"/>

</StackPanel>

</RadioButton>

<RadioButton GroupName="Content" Style="{StaticResource menuButton}"

Tag="#E285DE"

IsChecked="True"

Command="{Binding ShowLetezoDiagrammokViewCommand}">

<StackPanel Orientation="Horizontal">

<TextBlock Text="Létező diagrammok" Style="{StaticResource menuButtonText}" Margin="30,0,0,0"/>

</StackPanel>

</RadioButton>

</StackPanel>

</Expander>

Példa a lefuttatott Command-ra:

private void ExecuteShowDolgozoViewCommand(object obj)

{

CurrentChildView = new DolgozoViewModel();

}

## **4.3. Főoldal**

A főoldal két fő részből épül fel: a navigációs menüből – melynek működését az előző pontban részleteztem –, valamint a fő tartalomterületből, ahol a UserControl-ok dinamikus betöltésével történik az aktuális nézet megjelenítése. A főoldal feladata az adatbázis táblákhoz tartozó adatok megjelenítése, amelyekhez kapcsolódóan minden egyéb funkció is elérhetővé válik a felhasználó számára.

Három különböző fő tartalom megjelenítés történik:

* dolgozok, magan\_szemelyek, gazdalkodasi\_szervezetek, bevetelek\_kiadasok, kotelezettsegek\_kovetelesek táblák DataGrid-be való megjelenítése
* diagrammok táblázat kártyás megjelenítéssel történik
* új diagrammok létrehozása is a fő oldalon történik, ahol a diagramm típusok vannak felsorolva

### **4.3.1. Adatok megjelenítése/listázása - DataGrid**

A különböző adatbázis-táblák tartalmának megjelenítését DataGrid vezérlőelemmel oldottam meg. Az alábbi példa egy ilyen táblázat XAML-kódját mutatja:

<DataGrid Grid.Column="0"

x:Name="dataGrid" SelectionMode="Extended" SelectionChanged="DataGrid\_SelectionChanged"

AutoGeneratingColumn="dataGrid\_AutoGeneratingColumn"

CanUserResizeColumns="True"

ColumnWidth="\*"

AutoGenerateColumns="True"

ItemsSource="{Binding FilteredBevetelekKiadasok, Mode=TwoWay, NotifyOnTargetUpdated=True, UpdateSourceTrigger=PropertyChanged, NotifyOnSourceUpdated=True}"

IsReadOnly="True"

SelectionUnit="FullRow"

SelectedItem="{Binding SelectedRow, Mode=TwoWay}"

Grid.ColumnSpan="2"

ContextMenuOpening="DataGrid\_ContextMenuOpening">

<DataGrid.ContextMenu>

…

</DataGrid.ContextMenu>

</DataGrid>

A fontosabb beállítások röviden:

* SelectionMode="Extended": lehetővé teszi több sor kijelölését, például a SHIFT billentyű lenyomásával.
* SelectionUnit="FullRow": teljes sorokat lehet kijelölni, nem csak egyes cellákat.
* IsReadOnly="True": megakadályozza az adatok közvetlen szerkesztését a táblázatban.
* AutoGenerateColumns="True": automatikusan létrehozza az oszlopokat a forrásobjektum alapján.
* ColumnWidth="\*": az oszlopok kitöltik a rendelkezésre álló helyet.
* CanUserResizeColumns="True": lehetővé teszi az oszlopok szélességének kézi állítását.
* AutoGeneratingColumn="dataGrid\_AutoGeneratingColumn": egy eseménykezelő, amellyel a generált oszlopokat utólag módosíthatom, például elnevezések vagy formázás szempontjából.
* ItemsSource="{Binding FilteredBevetelekKiadasok…: a *FilteredBevetelekKiadasok* kollekcióhoz köti a táblázat tartalmát. Ez a gyűjtemény tartalmazza az adatbázisból lekért költségvetési adatokat, amelyeket a felhasználó szűrhet. Ha a felhasználó nem alkalmazott szűrést, akkor a teljes adathalmaz jelenik meg.

Az automatikusan generálódó táblázat oszlopainak felülírásához használt esemény egy példa kódja:

private void dataGrid\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

if (e.PropertyName == "IsSelected")

{

e.Cancel = true; // Prevents this column from being generated

}

if (e.PropertyType == typeof(DateTime))

{

// Get the column and cast to DataGridTextColumn

var textColumn = e.Column as DataGridTextColumn;

if (textColumn != null)

{

// Set the StringFormat for date formatting

textColumn.Binding = new Binding(e.PropertyName)

{

StringFormat = "yyyy.MM.dd" // Format the date as desired

};

}

}

}

A *dataGrid\_AutoGeneratingColumn* esemény minden automatikusan generált oszlopra lefut. Amennyiben az oszlop neve *IsSelected*, annak megjelenítését letiltom, így ez a belső logikai mező nem jelenik meg a táblázatban. Emellett, ha az oszlophoz tartozó adattípus *DateTime*, akkor a megjelenített érték formátumát beállítom (például: *yyyy.MM.dd*), ezzel biztosítva az egységes és jól olvasható dátummegjelenítést a táblázatban.

Az alábbi kódrészlet egy *DataGrid* jobb kattintásos (*ContextMenu*) menüjének példáját mutatja be:

<DataGrid.ContextMenu>

<ContextMenu>

<MenuItem Header="Törlés" Command="{Binding DeleteBevetelKiadasCommand}"/>

<MenuItem Header="Módosítás"

Command="{Binding OpenBevetelKiadasModifyOrAddWindowCommand}"

CommandParameter="Modify"

Visibility="{Binding IsSingleRowSelected, Converter={StaticResource BoolToVis}}"/>

</ContextMenu>

</DataGrid.ContextMenu>

A *ContextMenu* segítségével a felhasználó jobb kattintással elérhető műveleteket hajthat végre az adott táblázat egyes soraival kapcsolatban – például egy bejegyzés törlését vagy módosítását.

A "Módosítás" menüpont kizárólag abban az esetben érhető el, ha pontosan egy sor van kijelölve a táblázatban. Ezt a viselkedést a *Visibility* tulajdonság szabályozza, amely az *IsSingleRowSelected* nevű, bool típusú tulajdonsághoz van kötve. Az érték megjelenítéséhez egy konverter (*BoolToVis*) kerül alkalmazásra, amely a logikai értékeket *Visibility.Visible* vagy *Visibility.Collapsed* értékké alakítja, a felhasználói élmény és a logika támogatása érdekében.támogatása érdekében.

A „Törlés” menüpont több rekord törlésére is használható egy időben.

### **4.3.2. Törlés**

Egy törlésre példa:

public void DeleteDolgozo(int id)

{

try

{

// Check for related records and get information about them

var hasRelatedRecords = \_dolgozoRepository.CheckForRelatedRecords(id, out string relatedInfo);

// If there are related records, ask for confirmation

bool shouldDelete = true;

if (hasRelatedRecords)

{

shouldDelete = ConfirmCascadeDelete(

"Biztosan törölni szeretné ezt a dolgozót?",

relatedInfo);

}

if (shouldDelete)

{

\_dolgozoRepository.DeleteDolgozo(id, true);

// Remove from filtered collections

for (int i = 0; i < FilteredDolgozok.Count; i++)

{

if (FilteredDolgozok.ElementAt(i).ID == id)

FilteredDolgozok.RemoveAt(i);

}

}

}

catch (Exception e)

{

System.Windows.MessageBox.Show(

$"Hiba történt a törlés során: {e.Message}",

"Hiba",

MessageBoxButton.OK,

MessageBoxImage.Error);

}

}

A törlési folyamat első lépéseként lefut a \_dolgozoRepository.CheckForRelatedRecords metódus. Ez ellenőrzi, hogy a törölni kívánt rekordhoz tartozik-e más adat egyéb táblákból.

* Ha talál kapcsolódó rekordokat, a metódus true értékkel tér vissza, és a relatedInfo változóban megadja, hogy pontosan hány kapcsolódó adat található. Ilyen esetben egy felugró ablak jelenik meg, amely megerősítést kér a felhasználótól: valóban törölni szeretné-e ezeket a kapcsolódó rekordokat is.
  + Ha a felhasználó az „Igen” lehetőséget választja, a kapcsolódó rekordok is törlésre kerülnek.
  + Ha a „Nem” lehetőséget választja, a törlési művelet megszakad.
* Ha nincs kapcsolódó adat, a kiválasztott rekord azonnal törlésre kerül.

A következő kódsorok futnak le, ha a törlés végbemegy:

public bool DeleteDolgozo(int id, bool confirmCascade = true)

{

List<string> relatedRecords = new List<string>();

int diagrammCount = 0;

using (MySqlConnection connection = GetConnection())

{

connection.Open();

int felhasznaloId = 0;

string getFelhasznaloIdQuery = "SELECT id FROM felhasznalok WHERE dolgozo\_id = @id";

using (MySqlCommand command = new MySqlCommand(getFelhasznaloIdQuery, connection))

{

command.Parameters.AddWithValue("@id", id);

felhasznaloId = Convert.ToInt32(command.ExecuteScalar());

}

string checkDiagrammQuery = "SELECT COUNT(\*) FROM diagrammok WHERE letrehozo\_id = @id";

using (MySqlCommand command = new MySqlCommand(checkDiagrammQuery, connection))

{

command.Parameters.AddWithValue("@id", felhasznaloId);

diagrammCount = Convert.ToInt32(command.ExecuteScalar());

if (diagrammCount > 0)

{

relatedRecords.Add($"Diagramm bejegyzések: {diagrammCount} db");

}

}

if (felhasznaloId > 0 && confirmCascade == false)

{

return false;

}

using (MySqlTransaction transaction = connection.BeginTransaction())

{

try

{

if (diagrammCount > 0)

{

string deleteDiagrammQuery = "DELETE FROM diagrammok WHERE letrehozo\_id = @id";

using (MySqlCommand command = new MySqlCommand(deleteDiagrammQuery, connection, transaction))

{

command.Parameters.AddWithValue("@id", felhasznaloId);

command.ExecuteNonQuery();

}

}

if (felhasznaloId > 0)

{

string deleteFelhasznalokQuery = "DELETE FROM felhasznalok WHERE dolgozo\_id = @id";

using (MySqlCommand command = new MySqlCommand(deleteFelhasznalokQuery, connection, transaction))

{

command.Parameters.AddWithValue("@id", id);

command.ExecuteNonQuery();

}

}

string deleteQuery = "DELETE FROM dolgozok WHERE id = @id";

using (MySqlCommand command = new MySqlCommand(deleteQuery, connection, transaction))

{

command.Parameters.AddWithValue("@id", id);

int result = command.ExecuteNonQuery();

transaction.Commit();

return result > 0;

}

}

catch (Exception)

{

transaction.Rollback();

throw;

}

}

}

}

A törlés első lépése a *getFelhasznaloIdQuery* lekérdezés futtatása, amely eredményét a *felhasznaloId* változóba mentem. Ennek segítségével ellenőrzöm, hogy a dolgozóhoz kapcsolódó felhasználóhoz tartozik-e rekord a diagrammok táblában. Ha tartozik, akkor a *relatedRecords* változóhoz hozzáadódik a kapcsolódó rekordok száma.

Ezek után, ha a *felhasznaloId* nem nulla, tehát valóban van kapcsolódó felhasználó, és a *confirmCascade* paraméter értéke *false*, akkor a metódus *false* értékkel tér vissza, és a törlés nem történik meg. Egyéb esetben elindul egy adatbázis tranzakció, amelyben a következő lépések hajtódnak végre:

1. Diagramok törlése: Ha a *diagrammCount* értéke nagyobb, mint nulla, akkor törlésre kerülnek azok a diagrammok, amelyek *letrehozo\_id* mezője megegyezik a lekért *felhasznaloId*-val.
2. Felhasználó törlése: Ha a *felhasznaloId* létezik (nagyobb, mint nulla), akkor a kapcsolódó felhasználó rekord is törlésre kerül.
3. Dolgozó törlése: Végül megtörténik a dolgozó rekord törlése is.

Ha a fenti lépések hiba nélkül lefutnak, a tranzakció *commit*() metódusa véglegesíti a változtatásokat.  
Hiba esetén a *rollback*() metódus lefutása visszavonja az összes végrehajtott műveletet.

### **4.3.3. Módosítás**

Az alkalmazás lehetőséget biztosít az adatok módosítására egy külön ablakban, amelyet a felhasználó a táblázatból egy sor kijelölését követően, a jobb kattintásos *ContextMenu* „Módosítás” opciójára kattintva érhet el. Fontos megjegyezni, hogy egy időben kizárólag egy rekord módosítása engedélyezett táblázatonként, ezzel biztosítva a pontos és célzott adatkezelést. A kijelölt rekord betöltésre kerül a módosítást végző ablak beviteli mezőiben.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.3.3.1. ábra, módosítás ablak példa**

Az adatok kétirányú kötése (*TwoWay* binding) biztosítja, hogy a felhasználó által módosított értékek automatikusan frissüljenek a nézetmodellben (*ViewModel*). A „Mentés” művelethez rendelt parancs (*Command*) végzi a validálást és az adatbázis műveletek indítását.

### **4.3.4. Új rekord felvétele**

Az alkalmazás lehetőséget biztosít új rekordok hozzáadására egy külön ablakban. Fontos megjegyezni, hogy ebben az esetben is egy időben kizárólag egy új rekord vihető fel az adatbázisba táblázatonként, ezzel biztosítva a pontos és célzott adatkezelést.

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.3.4.1. ábra, létrehozás ablak példa**

Azon adatok megadása minden esetben kötelező, ahol a hibaüzenetek megjelennek és a szöveg kifejezetten írja, hogy nem lehet üresen hagyni a mezőt. Amennyiben az új rekord hozzáadása sikeres, egy felugró ablak értesíti erről a felhasználót. Ha sikertelen, akkor szintén egy felugró ablak értesíti a felhasználót.

### **4.3.5. DataGrid adat exportálás Excel formátumba**

Az aktuálisan megjelenített táblázat adatai három különböző módon exportálható excel formátumban.

A következőek a lehetséges módok:

* Összes adat exportálása – az adatforrásban elérhető összes rekord kimentésre kerül. ExportToExcel(Dolgozok.ToList(), "Osszes\_Dolgozok");
* Csak a szűrt adatok exportálása – kizárólag azok az adatok kerülnek exportálásra, amelyek megfelelnek a felhasználó által beállított szűrési feltételeknek. ExportToExcel(FilteredDolgozok.ToList(), "Szurt\_Dolgozok");
* Csak a kijelölt adatok exportálása – csak a felhasználó által a táblázatban kijelölt sorok kerülnek bele az Excel fájlba. ExportToExcel(SelectedItems.ToList(), "Kijelolt\_Dolgozok");

Mindhárom esetben a fájl helyét és nevét a felhasználónak van lehetősége megadni így biztosított a személyre szabhatóság és a könnyű visszakereshetőség.

A felhasználó számára biztosított továbbá a több sor kijelölésének lehetősége, amelyet egy gomb kezel, amit a következő command metódus kezel:

private void ExecuteToggleMultiSelectionModeCommand(object obj)

{

IsMultiSelectionMode = !IsMultiSelectionMode;

IsExportModeActive = IsMultiSelectionMode; // This will control context menu

// Clear selections when exiting multi-selection mode

if (!IsMultiSelectionMode)

{

SelectedItems.Clear();

OnPropertyChanged(nameof(SelectedItems));

}

}

Az *IsMultiSelectionMode* állapot váltása teszi lehetővé a felhasználónak, hogy több rekordot is kijelölhessen, Excel exportálás céljából. Az *IsExportModeActive* flag irányítja, hogy a *ContextMenu* látható-e - kijelölt rekordok exportálása során a *ContextMenu* nem használható.

Az Excel formátumba való exportálási folyamat egy részének a kódja:

SaveFileDialog saveFileDialog = new SaveFileDialog

{

Filter = "Excel Files (\*.xlsx)|\*.xlsx",

FileName = $"{defaultFileName}\_{DateTime.Now:yyyyMMdd}.xlsx",

Title = "Save Excel File"

};

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

using (var workbook = new XLWorkbook())

{

// Create a new worksheet

var worksheet = workbook.Worksheets.Add("Dolgozok");

// Add headers

worksheet.Cell(1, 1).Value = "ID";

worksheet.Cell(1, 2).Value = "Vezeteknev";

worksheet.Cell(1, 3).Value = "Keresztnev";

worksheet.Cell(1, 4).Value = "Email";

worksheet.Cell(1, 5).Value = "Telefonszam";

// Format header row

var headerRange = worksheet.Range(1, 1, 1, 5);

headerRange.Style.Font.Bold = true;

headerRange.Style.Fill.BackgroundColor = XLColor.LightBlue;

headerRange.Style.Font.FontColor = XLColor.White;

// Add data rows

for (int i = 0; i < dataToExport.Count; i++)

{

worksheet.Cell(i + 2, 1).Value = dataToExport[i].ID;

worksheet.Cell(i + 2, 2).Value = dataToExport[i].Vezeteknev;

worksheet.Cell(i + 2, 3).Value = dataToExport[i].Keresztnev;

worksheet.Cell(i + 2, 4).Value = dataToExport[i].Email;

worksheet.Cell(i + 2, 5).Value = dataToExport[i].Telefonszam;

}

// Auto-fit columns

worksheet.Columns().AdjustToContents();

// Create a table

var range = worksheet.Range(1, 1, dataToExport.Count + 1, 5);

var table = range.CreateTable("DolgozokTable");

table.Theme = XLTableTheme.TableStyleMedium2;

// Save the file

workbook.SaveAs(saveFileDialog.FileName);

System.Windows.MessageBox.Show(

$"Data successfully exported to {saveFileDialog.FileName}",

"Export Success",

System.Windows.MessageBoxButton.OK,

System.Windows.MessageBoxImage.Information);

}

}

Először is megnyílik a SaveFileDialog (itt adható meg a menteni kívánt fájl neve és helye). Amennyiben a felhasználó rákattintott az OK gombra, akkor a paraméterben kapott adatok alapján elkészítek egy workbook-ot. Amint minden oszlop és cella kitöltésre került (az oszlopok statikusan vannak elkészítve és a cellák dinamikus vannak feltöltve), a workbook a megfelelő névvel a megadott helyen elmentésre kerül.

### **4.3.6. DataGrid Keresés/Szűrés**

A szűrés funkció két vezérlőelemmel valósul meg: egy beviteli mezővel (*input*) és egy *ComboBox*-szal.

* A beviteli mező mindenféle karaktert elfogad, és az adatokat részszöveg (*substring*) alapján szűri: visszaadja azokat a rekordokat, amelyek bármelyik mezője tartalmazza a megadott szöveget.
* A ComboBox lehetőséget biztosít arra, hogy a felhasználó kiválassza, melyik oszlop(ok) alapján történjen a szűrés. Ezáltal a keresés pontosabban szabályozható.

Ez a megoldás rugalmas és felhasználóbarát, mivel lehetővé teszi az általános és célzott keresést is az adatok között. A *ComboBox* minden táblázatban a hozzátartozó oszlop nevek alapján kerül beállításra.

Egy példa kód a táblázatban való szűrésről (a kódrészlet egy foreach ciklusban található szűrési feltétele):

if (checkboxStatuses["idCB"] == true)

{

if (d.ID.ToString().Contains(searchQuery))

{

FilteredDolgozok.Add(d);

continue;

}

}

Ha a *ComboBox*-ban az „*ID*” *ComboBoxItem* be volt pipálva és a felhasználó adott meg szűrési szöveget, akkor megnézi, hogy a rekord (ez esetben *Dolgozo* – d) „*ID*” property-je tartalmazza-e a keresett szöveget. Amennyiben tartalmazza, akkor hozzáadja a szűrt dolgozó listához a rekordot (ez esetben *FilteredDolgozok*) és folytatja a foreach ciklust a következő objektummal.

## **4.4. Diagram létrehozása**

### **4.4.1. Diagram típus kiválasztása**

A navigációs menüből az „Új diagramm létrehozása” gombra kattintva megjelenik az a felhasználói felület (UjDiagrammokView UserControl), ahol példa adatokkal megjelenítve láthatóak a választható diagram típusok.

Jelenleg öt diagram típus hozható létre:

* Oszlop diagram (ColumnSeries)
* Sor diagram (RowSeries)
* Kör diagram (PieSeries – DoughnutSeries)
* Vonal diagram (LineSeries)
* Halmozott oszlop diagram (Stacked Column Series)

Ahhoz, hogy az új diagram létrehozása elkezdődhessen a felhasználónak ki kell választania az általa kívánt diagram típust a megfelelő diagram alatti „Új létrehozása” gombra kattintva.

A képen szöveg, képernyőkép, kör, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.4.1.1. ábra, digram kiválasztás példa**

Minden gomb a hozzá tartozó diagramnak megfelelő *Command* paramétert adja át (pl.: CommandParameter="DoughnutSeries").

A gomb lenyomását követően a következő *Command* fut le:

private void ExecuteCreateChartCommand(object obj)

{

CreateChartsView createCharts = new CreateChartsView(obj.ToString());

createCharts.Show();

}

A *CreateChartsView* ablak paraméterében a diagram típusa átadásra kerül és a *Show*() metódus segítségével megjelenítésre kerül a diagram létrehozásához szükséges ablak.

A CreateChartsView ablak a következő részekből épül fel:

### **4.4.2. Szűrési lehetőségek**

Azonos szűrési metódus, mint ami a [4.3.6. DataGrid Keresés/Szűrés](#_4.3.6._DataGrid_Keresés/Szűrés) pont alatt került kifejtésre, azzal a kiegészítéssel, hogy itt két újabb dedikált beviteli mező gondoskodik a dátum szerinti szűrésről. Mindkét szűrés használható dátum szerinti szűrésre, viszont a dedikált dátum kereső segítségével lehetséges két dátum közötti intervallum szűrés, illetve adott dátum-tól, illetve dátum-ig való szűrés (zárt és nyílt intervallum szűrési lehetőség).

A dedikált dátumos szűrés működése:

1. A felhasználó megadja a kívánt dátum(ot)(okat).
2. Kezdő (*StartDate*) és vég dátum (*EndDate*) esetén is az *IsValidDate*() metódus gondoskodik a megfelelő dátum beállításról.
3. Amennyiben a felhasználó által megadott dátum érvényes dátum, akkor a megfelelő flag property-k beállítása segítségével a szűrés végbemegy.

Az *IsValidDate*() metódus működése:

private bool IsValidDate(string date, bool IsStartDate)

{

DateTime? parsedDate = DateTimeParser.ParseDateTime(date);

if (parsedDate.HasValue)

{

if(IsStartDate)

{

StartingDate = parsedDate.Value.Date.ToString("yyyy.MM.dd");

}

else

{

EndDate = parsedDate.Value.Date.ToString("yyyy.MM.dd");

}

return true;

}

else

{

return false;

}

}

A felhasználó által string formátumban bekért dátum átadásra kerül a metódusnak, és a bool *IsStartDate* helyes beállításával megadjuk a meghívás során, hogy kezdő vagy vég dátum esetén hívódott-e meg a metódus. Az *IsStartDate* *true* beállítása kezdő dátumra utal, míg a *false* érték a vég dátumra. A *DateTimeParser* osztály *ParseDate*() metódusának átadjuk az ellenőrizendő string-et, ami sikeres dátum megadás esetén visszatér a *DateTime* típusra konvertált értékkel. Ha sikeres volt a dátummá való konvertálás, akkor a paraméterben kapott IsStartDate segítségével a megfelelő dátum property beállításra kerül és a metódus *true* értékkel tér vissza. Abban az esetben, ha nem volt sikeres a dátum megadása (így a dátum konvertálása sem), akkor a metódus *false* értékkel tér vissza. Az *IsValidDate*() metódus visszatérési értéke a következő két flag-et állíthatja be: *IsValidEndDateExists*, *IsValidStartDateExists*. Ez a két flag segíti a szűrés során a megfelelő dedikált dátum szűrést.

A *DateTimeParser* osztály kifejtése később az [5.1 DateTimeParser](#_5.1._DateTimeParser) pont alatt kerül kifejtésre.

### **4.4.3. Csoportosítási lehetőségek**

A diagram típusok tartalmi használatának eltérő különbözete miatt több, az adott diagram típusokhoz szabott csoportosítási opciókra volt szükség, így ezek felépítésének különbségei külön-külön bemutatásra kerülnek.

Kördiagram (PieSeries – DoughnutSeries):

A képen szöveg, Betűtípus, szám, sor látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.4.3.1. ábra, csoportosítás példa**

A kijelölt adatok Adatsorokra bontásáról a következő 4 opció gondoskodik:

* Penznem
* BeKiKód – Kifizetett (Kötelezettség és Követelés táblából kiválasztott adatok során)
* Év
* Hónap – tartozik hozzá egy ComboBox is, aminek tartalma attól függ, hogy milyen adatok lettek kiválasztva (A kiválasztott adatokban szereplő évek vannak listában szedve, ezek közül kell kiválasztania a felhasználónak, hogy melyik év hónapai alapján kívánja az adatsorokat létrehozni)

Az Év és a Hónap csoportosítás nem lehetséges egyszerre. Néhány példa a csoportosítás beállítására: Penznem – BeKiKód – Év, Penznem – Hónap, nem támogatott csoportosítás például a Penznem – Év – Hónap.

Oszlop diagram, Sor diagram, Halmozott oszlop diagram (ColumnSeries, RowSeries, StackedColumnSeries):

A képen szöveg, sor, diagram, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.4.3.2. ábra, csoportosítás példa**

A kijelölt adatok Adatsorokra bontásáról a következő 4 opció gondoskodik:

* Penznem
* BeKiKód – Kifizetett (Kötelezettség és Követelés táblából kiválasztott adatok során)
* Év
* Hónap

Ebben az esetben a csoportosítási komplexitás miatt megkötésekre volt szükség a helyes megjelenítés céljából. A csoportosítási opciók ebben az esetben két funkciót láthatnak el: Címke vagy Adatsor.

A megkötések:

* Egy csoportosítás egyszerre csak egy szerepet láthat el.
* Egy Címkének minden esetben kell lennie a helyes megjelenítéshez és diagram konfiguráláshoz.
* A következő csoportosítási párok vannak jelenleg megvalósítva: Penznem – BeKiKód és Év - Hónap

Az Év és Hónap csoportosítás esetén a következőek a lehetséges megvalósítások: Címke: Év és Adatsor: Hónap; Címke: Hónap és Adatsor: Év.

A Penznem és BeKiKód csoportosítás esetén pedig a következőek a lehetséges megadási opciók: Címke: Penznem, BeKiKód; Címke: Penznem és Adatsor: BeKiKód; Címke: BeKiKód és Adatsor: Penznem.

Vonal diagram (LineSeries):

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4.4.3.3. ábra, csoportosítás példa**

A kijelölt adatok Adatsorokra bontásáról a következő 4 opció gondoskodik:

* Penznem
* BeKiKód – Kifizetett (Kötelezettség és Követelés táblából kiválasztott adatok során)
* Dátum

### **4.4.4. Adatok kiválasztása megjelenítéshez**

Az adatok kiválasztása két *DataGrid* segítségével történik: a Költségvetés és a Kötelezettség és Követelés táblázat adatai vannak megjelenítve. Ugyan, olyan működési elven történik, mint ahogy az a [4.3.1. Adatok megjelenítése/listázása – DataGrid](#_4.3.1._Adatok_megjelenítése/listázá) pontban van taglalva, azzal a különbséggel, hogy a *ContextMenu*-ben a „Kijelölés” és „Kijelölés(ek) törlése” *Command* elérhető és nem a Törlés és Módosítás.

A két táblázat adatai közül egyidőben csak az egyik tábla adatiból jelölhetőek ki adatok. Ez azt jelenti, hogy ha a Költségvetés tábla legalább egy eleme már ki lett jelölve, akkor a Kötelezettség és Követelés tábla adatai már nem megtekinthetőek, ezzel biztosítva, hogy csak egy tábla adatai legyenek egy időben kijelölve. Az összes adat kijelölésére és jelölés törlésére található egy külön gomb a megfelelő felhasználói élmény biztosítása érdekében.

Példa kódrészlet abban az esetben, ha a Költségvetés táblából jelölt ki a felhasználó adatot:

if (dataGrid.Name.Equals("bevetelek\_kiadasok"))

{

int[] changeableItems = new int[dataGrid.SelectedItems.Count];

int a = 0;

foreach (var item in dataGrid.SelectedItems)

{

BevetelKiadas i = (BevetelKiadas)item;

changeableItems[a] = i.ID;

if (senderName == "cellSelectionTrue")

{

if(viewModel.SelectedBevetelekKiadasok.FirstOrDefault(x => x.ID == i.ID) == null)

{

viewModel.SelectedBevetelekKiadasok.Add(i);

viewModel.IsEnabledChangerOnTabItems();

}

}

else

{

if (viewModel.SelectedBevetelekKiadasok.FirstOrDefault(x => x.ID == i.ID) != null)

{

foreach(var b in viewModel.SelectedBevetelekKiadasok)

{

if(b.ID == i.ID)

{

viewModel.SelectedBevetelekKiadasok.Remove(b);

viewModel.IsEnabledChangerOnTabItems();

break;

}

}

}

}

a++;

}

if (senderName == "cellSelectionTrue")

{

viewModel.ChangeIsSelectedAndRefreshBevetelKiadas(changeableItems, true);

}

else

{

viewModel.ChangeIsSelectedAndRefreshBevetelKiadas(changeableItems, false);

}

if (viewModel.GroupByMonthCheckBoxIsChecked)

{

viewModel.Years = new ObservableCollection<int>(

viewModel.\_selectedBevetelekKiadasok

.Select(x => x.TeljesitesiDatum.Year)

.Distinct()

.OrderBy(year => year)

);

}

}

A *changeableItems* int tömb tárolja azokat az id-ket, amiket módosítani kell az adott táblázatban. Jelen esetben a módosítás csak az objektum *IsSelected* property-jének az értékét változtatja meg. A *cellSelectionTrue* az a „Kijelölés” *MenuItem* neve, ami abban segít, hogy tudjuk, hogy a felhasználótól jött a „Kijelölés” *Command*, illetve, hogy kijelölést és nem jelölés törlését végezzük el. A foreach végig megy az összes kijelölt soron, minden sort egy megfelelő class-ra konvertál (Költségvetés esetén *BevetelKiadas*). Ha a *sender* neve cellSelectionTrue, akkor hozzáadjuk az adott *BevetelKiadas* objektumot a *SelectedBevetelekKiadasok* listához, ellenkező esetben pedig eltávolítjuk azt. Mindkét esetben a két táblázat elérhetőségét az *IsEnabledChangerOnTabItems*() metódus biztosítja. Amennyiben a *GroupByMonthCheckBoxIsChecked* csoportosítás (*Hónap*) ki van választva, akkor egyben a *Hónap* csoportosításhoz tartozó *ComboBox*, ami az éveket tárolja feltöltésre kerül a kijelölt adatokban előforduló évekkel.

### **4.4.5. Diagram megjelenítése**

A diagram megjelenítésére az „Előnézet” gomb szolgál, amely a felhasználói interakcióra reagálva meghívja a *SetSeries*() metódust. A *SetSeries*() metódus egy egyszerű *switch(SeriesType)* szerkezetet használ, amely minden esetben egy adott diagramtípust létrehozó segédfüggvényt hív meg. Ezek a metódusok felelősek a megfelelő diagram osztály példányosításáért, az adatok hozzárendeléséért, valamint a megjelenítési paraméterek beállításáért.

Egy diagram beállítás folyamatát mutatom be kódrészletekkel együtt:

1) A csoportosítási flag-ek közül az Év és a Hónap lett kiválasztva.

if (GroupByYearCheckBoxIsChecked && GroupByMonthCheckBoxIsChecked)

{

2) Az Év csoportosítási flag az a SelectedCimke listában van, míg a Hónap csoportosítási flag a SelectedAdatsorok listában található.

if (SelectedCimkek.Contains("Év") && SelectedAdatsorok.Contains("Hónap"))

{

3) Készítettem egy honapok listát a minden hónap első három karakteréből

List<string> honapok = new List<string> { "Jan", "Feb", "Mar", "Apr", "Maj", "Jun", "Jul", "Aug", "Sep", "Okt", "Nov", "Dec" };

4) A *groupedByYearAndMonth* szótárba mentem a *\_selectedBevetelekKiadasok* (a felhasználó által kiválasztott adatok listája) lista minden elemét a TeljesitesiDatum.Year (Év) és TeljesitesiDatum.Month (Hónap) alapján csoportosítva.

var groupedByYearAndMonth = \_selectedBevetelekKiadasok

.GroupBy(p => new { p.TeljesitesiDatum.Year, p.TeljesitesiDatum.Month })

.ToDictionary(

g => g.Key,

g => g.ToList()

);

5) A *yearsDict* szótárba mentem a *\_selectedBevetelekKiadasok* (a felhasználó által kiválasztott adatok listája) lista minden elemét a *TeljesitesiDatum.Year* (Év) alapján csoportosítva. A szótárban lévő kulcsok az évek, míg a kulcshoz tartozó *Value* érték => egy másik szótár (*monthlyValues*), ami 1-től 12-ig tartó szám típusú kulcsokkal rendelkezik. Ebben a belső szótárban a *Value* értéket a következő képpen adom meg:

A *groupedByYearAndMonth* változóból kiszűröm azokat az adatokat, ahol az év megegyezik az aktuális feldolgozás alatt álló évvel (*kvp.Key.Year* == *g.Key*) és az éppen vizsgált hónappal (*kvp.Key.Month* == *month*). A .*SelectMany* segítségével ezekből a csoportokból lekéri az összes adatot és a .*List* segítségével pedig listát készít a lekért adatokból, melyek mentésre kerülnek a *matchingGroup* változóba. A *matchingGroup* listából lekérem az összes objektum *Összeg* property-jét és mentem azokat a *values* listába. Ezután ellenőrzöm, hogy volt-e értéke a *matchingGroup*-nak, ha volt neki, akkor meghívom a *GetDataStatisticValueBack*() metódust a *values* listát átadva, egyéb esetben pedig 0 értékkel térek vissza. Ezzel a belső szótár elkészül, amivel visszatér a belső függvény (return *monthlyValues*).

var yearsDict = \_selectedBevetelekKiadasok

.GroupBy(x => x.TeljesitesiDatum.Year)

.ToDictionary(

g => g.Key,

g =>

{

var monthlyValues = Enumerable.Range(1, 12)

.ToDictionary(

month => month,

month =>

{

var matchingGroup = groupedByYearAndMonth

.Where(kvp => kvp.Key.Year == g.Key && kvp.Key.Month == month)

.SelectMany(kvp => kvp.Value)

.ToList();

var values = matchingGroup.Select(x => Convert.ToDouble(x.Osszeg)).ToList();

return matchingGroup.Any() ? GetDataStatisticValueBack(values) : 0;

}

);

return monthlyValues;

}

);

6) Létrehozok egy *label* listát (List<string>), ami a címkéket fogja tárolni. Végig megyek a *yearsDict* szótáron és minden kulcsát hozzáadok a *label* listához.

List<string> labels = new List<string>();

foreach (var year in yearsDict)

{

labels.Add(year.Key.ToString());

}

7) Létrehozok egy listát, ami az adatsorok adatait fogja tárolni. A listát feltöltöm 12 üres *ChartValues<double>* listával, ami később feltöltésre kerül.

var monthlyData = new List<ChartValues<double>>();

for (int i = 0; i < 12; i++)

{

monthlyData.Add(new ChartValues<double>());

}

8) Sorba rendezem a *yearsDict* szótárat a kulcsai alapján és mentem egy *sortedYears* változóba.

var sortedYears = yearsDict.OrderBy(y => y.Key);

9) A külső for ciklus végig iterál a 12 hónapon. Minden hónap esetén a belső foreach végigmegy az összes év adatain (*sortedYears*). Lekéri az adott év adott hónaphoz tartozó értékét (*year.Value[month])* és menti a *monthValue* változóba. Hozzáadja ezt az értéket a *monthlyData* listához (*monthlyData[month – 1]*) – azért -1, mert a lista indexelése nullától indul.

for (int month = 1; month <= 12; month++)

{

foreach (var year in sortedYears)

{

double monthValue = year.Value[month];

monthlyData[month - 1].Add(monthValue);

}

}

10) Két incrementálandó változót hoztam létre: *„a”* – a *baseColor* lista a. indexű színét adja vissza; *„c”* – a *hónapok* lista c. indexű hónapját (string) adja vissza. A foreach végig megy az *monthlyData* listán, ahol minden iteráció egy *ChartValues<double>* lista. Minden iteráció során továbbá lefut az *AddRowSeries* és *AddGroupByDataToCollection* metódus is.

int a = 0;

int c = 0;

foreach (var b in monthlyData)

{

AddRowSeries(b, $"Bevételek és Kiadások - {honapok[c]}", honapok[c].ToString(), baseColors[a], labels);

AddGroupByDataToCollection(honapok[c].ToString(), a);

if (a + 1 > 3)

a = 0;

else

a++;

c++;

}

11) Az *AddRowSeries* metódusmagyarázata:

Öt paramétert kap a metódus:

* ChartValues<double> values – az adatok, amiket meg szeretnénk jeleníttetni az adatsorral
* string title – az adatsor címe
* string name – az adatsor neve, fontos a megjelenítés miatt, hogy minden esetben egyedi legyen
* Brush color – az adatsor automatikus beállított színe, a felhasználó által később állítható
* List<string> labels – azok a címkék, amik segítségével megjelenítjük a diagrammon az adatokat

Miután ezekkel az adatokkal létrehoztunk egy új *RowSeries* objektumot, hozzáadjuk a *RowSeries* kollekcióhoz. A *RowSeriesLabels* property-t beállítom a labels paraméter értékkel. Ezután a *RowSeriesFormatter* kerül beállításra, ami *RowSeries* esetén mindig ugyan az. Utolsó lépésnek pedig Az *OnPropertyChanged* metódus segítségével küldök egy értesítőt a UI felé, hogy változás történt a *RowSeries*, *RowSeriesLabels* és *RowSeriesFormatter* property-k esetén.

private void AddRowSeries(ChartValues<double> values, string title, string name, Brush color, List<string> labels)

{

RowSeries.Add(new RowSeries

{

Title = title,

Name = name,

Values = values,

DataLabels = true,

Fill = color,

});

RowSeriesLabels = labels;

RowSeriesFormatter = value => value == 0 ? "" : value.ToString("N");

OnPropertyChanged(nameof(RowSeries));

OnPropertyChanged(nameof(RowSeriesLabels));

OnPropertyChanged(nameof(RowSeriesFormatter));

}

12) Az *AddGroupByDataToCollection* magyarázata:

Az *isExistingAlready* definiálom *false* értékkel. Ezután egy foreach ciklus segítségével végig iterálok a *GroupBySelections* listán, ahol megnézem, hogy az épp aktuális *GroupBySelection* *Name* property-je egyezik-e a paraméterben kapott *groupByName* string-el. Ha egyezik, akkor az *isExistingAlready true* értéket kap és egy *break*-el kilépek a foreach-ből. Amennyiben a *isExistingAlready false* maradt, akkor a paraméterben kapott értékekkel létrehozok egy új *GroupBySelection* példányt és hozzáadom azt a *GroupBySelections* listához.

private void AddGroupByDataToCollection(string groupByName, int a)

{

bool isExistingAlready = false;

foreach (var b in GroupBySelections)

{

if (b.Name.Equals(groupByName))

{

isExistingAlready = true;

break;

}

}

if (!isExistingAlready)

{

GroupBySelections.Add(new GroupBySelection(groupByName, true, baseColors[a]));

}

}

### **4.4.6. Diagram exportálása**

A diagramok .png formátumú exportálására is van lehetőség. A következő kód az exportálás konfigurációját és menetét mutatja be:

public void ExportSpecificChart(string chartName)

{

UIElement chart = Mediator.NotifyGetSpecificChart(chartName);

if (chart != null)

{

var saveFileDialog = new Microsoft.Win32.SaveFileDialog

{

Filter = "PNG Image|\*.png",

Title = "Save Chart Image",

FileName = $"Chart\_{DateTime.Now:yyyyMMdd\_HHmmss}.png",

DefaultExt = ".png"

};

if (saveFileDialog.ShowDialog() == true)

{

string filePath = saveFileDialog.FileName;

string directoryPath = Path.GetDirectoryName(filePath);

if (!Directory.Exists(directoryPath))

{

Directory.CreateDirectory(directoryPath);

}

ExportChartAsImage(chart, directoryPath, Path.GetFileName(filePath));

}

}

}

A Mediator.NotifyGetSpecificChart(chartName) elkéri és visszaadja a megfelelő helyen a *CreateChartsView* ablaktól a megfelelő chart-ot (diagram) a paraméterben kapott név alapján. Ezután megjelenik egy felugró ablak, ahol nevet lehet adni a fájlnak, illetve a mentési helyét is ki lehet választani. Abban az esetben, ha ezek megtörténtek, akkor a „Mentés” gombra kattintva a *filePath* és a *directoryPath* beállításra kerül. Olyan esetben, ha a mappa útvonala nem létezik még, akkor a *Directory.CreateDirectory(directoryPath)*; létrehozza azt. Ezek után meghívásra kerül a *ExportChartAsImage*() metódus.

public void ExportChartAsImage(UIElement chart, string folderPath, string fileName = null)

{

if (!Directory.Exists(folderPath))

{

Directory.CreateDirectory(folderPath);

}

if (string.IsNullOrEmpty(fileName))

{

fileName = $"Chart\_{DateTime.Now:yyyyMMdd\_HHmmss}.png";

}

string filePath = Path.Combine(folderPath, fileName);

var size = new Size(chart.RenderSize.Width, chart.RenderSize.Height);

chart.Measure(size);

chart.Arrange(new Rect(size));

var renderBitmap = new RenderTargetBitmap(

(int)size.Width,

(int)size.Height,

96, // DPI X

96, // DPI Y

PixelFormats.Pbgra32);

DrawingVisual drawingVisual = new DrawingVisual();

using (DrawingContext drawingContext = drawingVisual.RenderOpen())

{

drawingContext.DrawRectangle(Brushes.White, null, new Rect(new Point(0, 0), size));

VisualBrush visualBrush = new VisualBrush(chart);

drawingContext.DrawRectangle(visualBrush, null, new Rect(new Point(0, 0), size));

}

renderBitmap.Render(drawingVisual);

using (var fileStream = new FileStream(filePath, FileMode.Create))

{

var encoder = new PngBitmapEncoder();

encoder.Frames.Add(BitmapFrame.Create(renderBitmap));

encoder.Save(fileStream);

}

}

A következő ellenőrzések és beállítások futnak le a kód elején:

* Először is ellenőrzésre kerül, hogy létezik-e a mentési hely (ebben az esetben már mindig létezni-e kellene, ha nem történt valami hiba a létrehozás során).
* Abban az esetben, ha a fileName üres vagy null, akkor beállításra kerül egy generált fájlnév.
* Ezután beállításra kerül a fájl útvonala is a filePath értékben a folderPath (mappa elérési útja) és a fileName (fájl neve) kombinálásából a Path.Combine metódus segítségével.

Mindez után a következő lépések szükségesek, hogy készülhessen egy kép a diagramról:

* Egy *Size* objektumot hozok létre a diagram renderelési szélessége és magassága segítségével.
* A *Measure* és *Arrange* beépített metódusok segítenek a diagram megfelelő elhelyezkedéséről és méretéről. Ezeket azért használjuk, mert nem a kijelzőn szeretnénk a diagrammot megjeleníteni.
* A *RenderTargetBitmap* egy új példányát hozzuk létre a következő paraméterekkel: a *width* és *height* a diagramból lekért és elmentett szélesség és magasság; 96-os X és Y tengelyes DPI (dots per inch) – ez adja a felbontását a képnek; *PixelFormats.Pbgra32*, ami egy 32 bites átlátszóságot is támogató színformátum.

Ezt követően létrehozunk egy *DrawingVisual* objektumot és ennek az objektumnak a *RenderOpen()* metódusának segítségével megnyitok egy *DrawingContext*-et, amire lehet rajzolni. Ezt a „felületet” először is kitöltjük fehér színnel, ami biztosítja a hátteret. Aztán a *VisualBrush* segítségével ráfestek egy másolatot a diagramról.

Amint mindez elkészült, a *DrawingVisual* objektumot átadom a *renderBitmap* (ez a *RenderTargetBitmap* előzőleg létrehozott példánya) beépített *Render()* metódusának. Ez azt jelenti, hogy itt már ténylegesen ki lesz renderelve bitképként a diagram másolata egy fehér háttérrel.

Miután elkészült a menteni kívánt kép, nyitok egy *FileSteam*-et a *filePath* változóban megadott elérési útvonalon. Ez a fájl lesz a kép mentésének célhelye Ezután létrehozok egy *PngBitmapEncoder* objektumot, amely lehetővé teszi a bitkép PNG formátumba való kódolását/konvertálását. A kirenderelt képet (*renderBitmap*) hozzáadom az *encoder* *Frames* listájához, majd az *encoder.Save*() metódus segítségével elmentem a képet a megnyitott *FileStream*-be.

### **4.4.7. Diagram mentése**

A *CreateChartsView*-on a „Mentés” gombra kattintva két egymást követő felugró ablak jelenik meg. Az első felugró ablakban kell megadnia a felhasználónak, hogy mi lesz a diagram címe/neve. Miután a cím/név megadásra került a következő felugró ablak is megjelenik. Itt kell megadni a diagramhoz tartozó leírást. Amennyiben a leírás is sikeresen meglett adva, abban az esetben a *SaveChartToDatabase*() metódus fut le.

A *SaveChartToDatabase*() metódus kódrészei és magyarázatai:

1) A *selectedItemIDs* listába gyűjtöm a kiválasztott Költségevetés vagy Kötelezettség és Követelés tábla ID-jeit.

List<int> selectedItemIDs = new List<int>();

if (IsBevetelekKiadasokTabIsSelected)

{

selectedItemIDs = \_selectedBevetelekKiadasok.Select(x => x.ID).ToList();

}

else

{

selectedItemIDs = \_selectedKotelezettsegekKovetelesek.Select(x => x.ID).ToList();

}

2) A szűrési és csoportosítási feltételeket egy-egy szótárba mentem, majd a kapott szótárat a Newtonsoft.Json.JsonConvert.SerializeObject segítségével json string formátumba konvertálom.

var filterSettings = new Dictionary<string, object>

{

{ "SearchQuery", SearchQuery },

{ "SelectedYear", SelectedYear },

{ "StartingDate", StartingDate },

{ "EndDate", EndDate },

{ "IsValidStartDateExists", IsValidStartDateExists },

{ "IsValidEndDateExists", IsValidEndDateExists },

{ "CheckboxStatuses", checkboxStatuses }

};

string filterSettingsJson = Newtonsoft.Json.JsonConvert.SerializeObject(

filterSettings,

Newtonsoft.Json.Formatting.Indented

);

var groupBySettings = new Dictionary<string, object>

{

{ "GroupByPenznem", GroupByPenznemCheckBoxIsChecked },

{ "GroupByBeKiKod", GroupByBeKiKodCheckBoxIsChecked },

{ "GroupByKifizetett", GroupByKifizetettCheckBoxIsChecked },

{ "GroupByMonth", GroupByMonthCheckBoxIsChecked },

{ "GroupByYear", GroupByYearCheckBoxIsChecked },

{ "GroupByDate", GroupByDateCheckBoxIsChecked },

{ "SelectedCimkek", cimkekArray },

{ "SelectedAdatsorok", adatsorokArray },

{ "SelectedDataStatistics", SelectedDataStatistics },

{ "InnerRadius", InnerRadius }

};

string groupBySettingsJson = Newtonsoft.Json.JsonConvert.SerializeObject(

groupBySettings,

Newtonsoft.Json.Formatting.Indented

);

3) A csoportosított adatsorok mindegyikét a *GroupBySelection* kollekcióban tárolom, aminek a *Name* (az adatsor neve), *IsSelected* (ki volt-e választva megjelenítésre) és *Color* (a színe *ARGB*-re bontott formában) property-eit mentem vele a *serializableGroupBySelections* változóba.

var serializableGroupBySelections = GroupBySelections.Select(g => new

{

Name = g.Name,

Color = new

{

A = g.Color.Color.A,

R = g.Color.Color.R,

G = g.Color.Color.G,

B = g.Color.Color.B

},

IsSelected = g.IsSelected

}).ToList();

string seriesGroupBySelectionsJson = Newtonsoft.Json.JsonConvert.SerializeObject(

serializableGroupBySelections,

Newtonsoft.Json.Formatting.Indented

);

4) A *serializableDataChartValues* listába azok az adatsorok kerülnek elmentésre, amelyek szükségesek a diagram helyes és aktuális megjelenítéséhez. Csak azokat az adatsorokat mentem el, amelyek ténylegesen láthatóak (azaz *Visibility.Visible* értékkel rendelkeznek).

A mentendő adatok az alábbi tulajdonságokat tartalmazzák:

* *Title*: az adatsor címe,
* *Name*: az adatsor neve (azonosítója),
* *DataLabels*: a címkék, amelyek az értékeket megjelenítik a diagramon,
* *Values*: az adatsorhoz tartozó értékek, *ObservableValue* típusból konvertálva egyszerű numerikus listává,
* *Fill*: az adatsor megjelenítésére használt szín.

A *switch* (*SeriesType*) szerkezet biztosítja, hogy a megfelelő típusú diagram (pl. *DoughnutSeries*, *ColumnSeries*, stb.) esetén a megfelelő adatgyűjtési logika fusson le.

Például a *DoughnutSeries* esetén a következő LINQ-lekérdezés segítségével kerülnek kiválogatásra és átalakításra az adatsorok:

serializableDataChartValues = PieSeries.Where(g => g is PieSeries pieSeries && pieSeries.Visibility == Visibility.Visible)

.Select(g => new

{

Title = ((PieSeries)g).Title,

Name = ((PieSeries)g).Name,

DataLabels = ((PieSeries)g).DataLabels,

Values = ((PieSeries)g).Values.Cast<ObservableValue>().Select(v => v.Value).ToList(),

Fill = ((PieSeries)g).Fill,

}).ToList();

Miután a látható adatsorokat átalakítottam, a *serializableDataChartValues* lista JSON formátumba kerül átalakításra a JsonConvert.SerializeObject() metódus segítségével:

string seriesDataChartValuesJson = Newtonsoft.Json.JsonConvert.SerializeObject(

serializableDataChartValues,

Newtonsoft.Json.Formatting.Indented

);

5) Létrehozok a meglévő adatok segítségével egy Diagram objektumot. A property-k közül, amik még nem kerültek említésre azok a következők:

* DataStatistic – SelectedDataStatistics – az a beállított statisztika, amit megjelenítünk a diagrammon (alapértelmezetten Sum – Összeg).
* CreatedDate - DateTime.Now – az az időpont, amikor a diagram mentésre kerül
* CreatedByUserID - Mediator.NotifyGetUserID() – annak a felhasználónak az ID-je, aki létrehozza/menti a diagrammot

var diagrammModel = new Diagram

{

ID = DiagrammID, // Assuming 0 for new chart

Name = chartName,

Description = chartDescription,

ChartType = SeriesType,

DataSource = IsBevetelekKiadasokTabIsSelected ? "Koltsegvetes" : "KotelezettsegKoveteles",

DataChartValues = seriesDataChartValuesJson,

FilterSettings = filterSettingsJson,

GroupBySettings = groupBySettingsJson,

SeriesGroupBySelection = seriesGroupBySelectionsJson,

SelectedItemsIDs = string.Join(",", selectedItemIDs),

DataStatistic = SelectedDataStatistics,

CreatedDate = DateTime.Now,

CreatedByUserID = Mediator.NotifyGetUserID()

};

6) Az utolsó dolog a mentés során az adatok adatbázisba való mentése:

diagrammRepository.SaveDiagramm(diagrammModel);

## **4.5. Létező diagrammok kártyás megjelenítése**

## **4.5.1. Kártyás megjelenítés**

3D-s kártyák formájában jelenítődnek meg a diagrammok. A diagramkártyák egy *ItemsControl* vezérlőn keresztül kerülnek kilistázásra, a *FilteredDiagrams* nevű lista alapján. Minden egyes kártya a felhasználó által létrehozott egyedi diagramot jeleníti meg, két oldallal: előlap (grafikus előnézet) és hátlap (részletes leírás és műveletek).

Főbb elemek és működés

1. *ScrollViewer* és *ItemsControl*

* A *ScrollViewer* lehetővé teszi a görgetést, ha több kártya jelenik meg, mint amennyi a rendelkezésre álló helyen elfér.
* Az *ItemsControl* az *ItemsPanelTemplate* segítségével egy *WrapPanel*-t használ a kártyák vízszintes elrendezésére.

2. 3D kártya felépítése (*Viewport3D*)

* Minden kártya egy *Viewport3D* komponenssel jelenik meg, amely lehetővé teszi a 3D animációkat és forgatást.
* A kártya a *Viewport2DVisual3D* elemen keresztül jelenik meg, ami lehetővé teszi 2D elemek (pl. *ToggleButton*, *Grid*) 3D térben való elhelyezését.
* A kamera (*PerspectiveCamera*) pozíciója előre van állítva a 3D térben, így a kártya mindig szemből látszik.

3. Kártya tartalma – előlap és hátlap

* A kártya előlapja (*FrontSide*):
  + Tartalmazza a diagram nevét (*TextBlock*) és a diagram típusától függő grafikon előnézetet (*CartesianChart* vagy *PieChart*).
* A kártya hátlapja (*BackSide*):
  + Megjeleníti a diagram részletes leírását, létrehozási dátumát, készítő nevét és két gombot:
    - Szerkesztés: diagram módosításának lehetősége
    - Törlés: diagram eltávolítása a listából

4. Interaktív viselkedés – lapozás *ToggleButton* segítségével

* A kártyák forgathatóak a *ToggleButton* segítségével.
* A gomb állapota (*Checked* vagy *Unchecked*) vezérli az elő- és hátlap láthatóságát, valamint a 3D forgatás animációját.
* A forgás az *AxisAngleRotation3D*-on keresztül történik, amely az Y-tengely körül forgatja a kártyát.
* A *Storyboard* animációk biztosítják a sima áttűnést és forgást:
  + *Checked* állapot: előlapról hátlapra vált (0° → 90°, majd -90° → 0°)
  + *Unchecked* állapot: hátlapról előlapra vált (0° → -90°, majd 90° → 0°)

5. Animációs részletek

* Az *ObjectAnimationUsingKeyFrames* animációk a *Visibility* értékeket állítják át késleltetve.
* A *DoubleAnimation* animációk a *Rotation.Angle* és az *Opacity* (átlátszóság) értékét animálják.

6. Egyéb technikai részletek

* A *ChartTypeToVisibilityConverter* értelmezi, hogy melyik típusú grafikon (*RowSeries*, *DoughnutSeries*, stb.) látszódjon.
* A *ToggleButton.Content* belső *Grid* szerkezet felel a vizuális elemek struktúrájáért.

## **4.5.2. Diagram visszatöltése/szerkesztése**

A kártyák formájában listázott létező diagrammok „Szerkesztés” gombjára kattintva a *CreateChartsView* ablak megnyílik és lefutásra kerül az ablakhoz tartozó *DataContext* *LoadDiagram* metódusa.

A *LoadDiagram* első 30 sora beállítja a következő property-ket: *DiagrammID, DiagrammName, DiagrammDescription, IsChartModifying, SeriesType, SelectedDataStatistics, IsBevetelekKiadasokTabIsSelected, SelectedBevetelekKiadasok* vagy *SelectedKotelezettsegekKovetelesek*.

1) var filterSettings = Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string, object>>(diagramm.FilterSettings);

var groupBySettings = Newtonsoft.Json.JsonConvert.DeserializeObject<Dictionary<string, object>>(diagramm.GroupBySettings);

A *filterSettings* és *groupBySettings* beállításra kerül a visszafejtett json string-ből. A két visszafejtett szótár segítségével a következő property-k kerülnek beállításra: *SearchQuery, SelectedYear, StartingDate, EndDate, IsValidEndDateExists, IsValidStartDateExists, GroupByPenznemCheckBoxIsChecked, GroupByBeKiKodCheckBoxIsChecked, GroupByKifizetettCheckBoxIsChecked, GroupByMonthCheckBoxIsChecked, GroupByDateCheckBoxIsChecked, GroupByYearCheckBoxIsChecked, InnerRadius.*

*2)* A *checkboxStatuses* szótár a következő képpen kerül beállításra:

var savedCheckboxStatuses = ((Newtonsoft.Json.Linq.JObject)filterSettings["CheckboxStatuses"]).ToObject<Dictionary<string, bool>>();

A mentett *CheckboxStatuses* értékből készítek egy szótárat, ami megegyezik a property típusával. A szótár kulcsain végig iterálok egy foreach segítségével és beállítom a *checkboxStatuses* értékeit az iterált kulcs érték segítségével.

checkboxStatuses[key] = savedCheckboxStatuses[key];

3) A *SelectedAdatsorok* és *SelectedCimkek* property-k a *groupBySettings*-ben mentett adatok *ObservableCollection<string>*-é konvertálom a json string-ből visszafejtett objektum struktúrát.

SelectedAdatsorok = ((Newtonsoft.Json.Linq.JArray)groupBySettings["SelectedAdatsorok"]).ToObject<ObservableCollection<string>>();

SelectedCimkek = ((Newtonsoft.Json.Linq.JArray)groupBySettings["SelectedCimkek"]).ToObject<ObservableCollection<string>>();

4) Lefuttatom a *SetSeries()* metódust.

5) A *GroupBySelections* beállításának menete:

var serializedSelections = Newtonsoft.Json.JsonConvert.DeserializeObject<List<dynamic>>(diagramm.SeriesGroupBySelection);

A *diagramm.SeriesGroupBySelection* json string-ből való visszafejtését a *serializedSelections* változóba mentem. Végig iterálok a *serializedSelections* listán és az aktuális iterált érték *Name*, *IsSelected*, *Color.A*, *Color.R*, *Color.G*, *Color.B* értékeit kimentem külön változókba (a változók balról jobbra: *name*, *isSelected*, *a*, *r*, *g*, *b*). A színekből készítek egy *Color* típusú értéket a *Color.FromArgb* beépített metódus segítségével. Ezt a *color* változót átadva a *SolidColorBrush* osztálynak készítettem egy színt, amit a *brush* változóban mentettem el. Az így kapott *name*, *isSelected*, *brush* változók segítségével létrehozok egy *GroupBySelection* egyedet és hozzáadom azt a *GroupBySelections* kollekciónak.

foreach (var item in serializedSelections)

{

string name = item.Name;

bool isSelected = item.IsSelected;

byte a = (byte)item.Color.A;

byte r = (byte)item.Color.R;

byte g = (byte)item.Color.G;

byte b = (byte)item.Color.B;

System.Windows.Media.Color color = System.Windows.Media.Color.FromArgb(a, r, g, b);

SolidColorBrush brush = new SolidColorBrush(color);

GroupBySelections.Add(new GroupBySelection(name, isSelected, brush));

}

# **Speciális osztályok**

Az összes speciális osztály, amit készítettem az statikus osztály, annak érdekében, hogy példányosítás nélkül lehessen használni az osztályban definiált metódusokat.

## **5.1. DateTimeParser**

public static DateTime? ParseDateTime(string dateString)

{

string[] formats = new[]

{

"yyyyMMdd", // 20240616

"yyyy.MM.dd", // 2024.06.16

"yyyy-MM-dd", // 2024-06-16

"yyyy/MM/dd", // 2024/06/16

"yyyy.MM-dd", // 2024.06-16

"yyyy-MM/dd", // 2024-06/16

"yyyy/MM-dd", // 2024/06-16

"yyyy.MMdd", // 2024.0616

"yyyy/MMdd", // 2024/0616

"yyyy-MMdd" // 2024-0616

};

if (DateTime.TryParseExact(dateString, formats, CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))

{

return result.Date;

}

return null;

}

A paraméterben kapott szöveget megpróbálja az előre megadott formátumok alapján konvertálni a *DateTime.TryParseExact*. Amennyiben ez sikerült, akkor a metódus visszatér a *DateTime* típusra konvertált string-et. Egyéb esetben null értékkel tér vissza.

## **5.2. PasswordHasher**

Az osztályban van egy beégetett *\_fixedSalt* változó, ami a hashelés biztonságosabbá tételéért felel.

public static string HashPassword(string password)

{

if (string.IsNullOrEmpty(password))

throw new ArgumentNullException(nameof(password));

string saltedPassword = password + \_fixedSalt;

using (SHA256 sha256 = SHA256.Create())

{

byte[] passwordBytes = Encoding.UTF8.GetBytes(saltedPassword);

byte[] hashBytes = sha256.ComputeHash(passwordBytes);

StringBuilder builder = new StringBuilder();

for (int i = 0; i < hashBytes.Length; i++)

{

builder.Append(hashBytes[i].ToString("x2"));

}

return builder.ToString();

}

}

A paraméterben kapott jelszót és a sót összefűzöm és a *saltedPassword* változóban eltárolom az eredményt.

Készítek egy SHA256 objektumot és a using segítségével gondoskodok arról, hogy a hash objektum felszabadításra kerüljön. Ezután a using-on belül a következő folyamatok mennek végbe:

* *passwordBytes* változóba elmentem a *saltedPassword* UTF-8-as kódolású bájt tömbként.
* *hashBytes* változóba elmentem az SHA-256 algoritmus által kiszámított hash értékét a *passwordBytes* tömbnek – a kimenete a számítása egy 32 bájtos byte tömb.
* Létrehozok egy *StringBuilder*-t, ami jó módja a string összeállításának a ciklusban.
* A for ciklus 0-tól a *hashBytes.Length* értékik fut.
* A for ciklusban *builder* értékéhez hozzáfüzöm a *hashBytes* i. elemének két karakter hosszúságú hexadecimális értékét – az x2 jelenti a két karakter hosszúságú hexadecimális formátumot.

A metódus visszatér a *builder* string-el.

public static bool VerifyPassword(string password, string hash)

{

if (string.IsNullOrEmpty(password) || string.IsNullOrEmpty(hash))

return false;

string passwordHash = HashPassword(password);

return passwordHash.Equals(hash);

}

A paraméterben kapott jelszó és hashelt jelszót egyezését vizsgálja meg a *VerifyPassword* metódus. Akkor tér vissza *true* értékkel, ha a paraméterben kapott jelszó hashelése és a paraméterben kapott hash megegyezik. Egyébként *false* értékkel tér vissza.

## **5.3. Mediator**

Az összes metódusa különböző ablakok közötti kommunikációt teszi lehetővé delegate és event segítségével.

public delegate void NewDolgozoAddedEventHandler(Dolgozo dolgozo);

public static event NewDolgozoAddedEventHandler NewDolgozoAdded;

public static void NotifyNewDolgozoAdded(Dolgozo dolgozo)

{

NewDolgozoAdded?.Invoke(dolgozo);

}

A *NewDolgozoAdded* event-re lehet feliratkozni, míg a *NotifyNewDolgozoAdded* értesíti a feliratkozás helyét, hogy új dolgozó lett hozzáadva az adatbázishoz.

## **5.4. DataStatistics**

Statisztikai metódusok megvalósításait tartalmazza.

public static double GetStandardDeviation(List<double> values)

{

double avarage = GetAvarage(values);

List<double> difference = new List<double>();

foreach (var v in values)

{

difference.Add(v - avarage);

}

List<double> squares = new List<double>();

foreach (var v in difference)

{

squares.Add(v \* v);

}

double differenceSquaresAVG = GetAvarage(squares);

return Math.Round(Math.Sqrt(differenceSquaresAVG), 2);

}

Az átlagtól való szórást adja vissza egy double listából. A számítási folyamat a következő:

* átlag kiszámítása és mentése az *avarage* változóba
* különbség lista létrehozása – *difference*
* egy foreach ciklus, ami végigmegy az összes értéken a values listában és a különbség listába az iteráció aktuális értékéből kivont átlagot hozzáadja a *differenc* listához
* készítek egy négyzeteket tároló listát – *squares*
* egy foreach ciklus végig iterál a *difference* listán és minden iteráció értékének a négyzetét veszi és hozzá adja azt a *squares* listához
* a négyzetekből készült lista értékeinek veszem az átlagát, amit a *differenceSquaresAVG* változóba mentek
* a metódus visszatér a *differenceSquaresAVG* gyökének két tizedes jegyre kerekített értékével

## **5.5. WindowHelper**

Ennek az osztálynak a metódusai ellenőrzik, hogy egy adott ablakból létezik-e már példány. Ez megakadályozza, hogy egy fajta ablakból nem lehet több megnyitva egy időben.

public static bool IsDolgozoAddWindowOpen<T>(out T window) where T : Window

{

window = Application.Current.Windows.OfType<T>().FirstOrDefault();

foreach (var a in Application.Current.Windows.OfType<T>())

{

if(a.DataContext is DolgozokModifyOrAddViewModel viewModel)

{

if(viewModel.EditMode == EditMode.Add)

{

window = a;

return true;

}

}

}

return false;

}

Ez a metódus azt vizsgálja meg, hogy a WPF alkalmazásban van-e nyitva olyan ablak (*Window*), amely:

* T típusú ablak (általános típusparaméter – például lehet *DolgozokAddWindow* vagy bármi más *Window*-t leszármaztató ablak),
* és a hozzá tartozó *ViewModel* típusa *DolgozokModifyOrAddViewModel*,
* és annak *EditMode* tulajdonsága Add értékű (tehát hozzáadás módban van az ablak).

Ha ilyen ablak létezik, akkor azt visszaadja az out paraméterben, és *true*-val tér vissza. Ha nem talál ilyet, akkor *false* a visszatérési érték.

# **6. Bővíthetőségek**

1. Diagram létrehozása során mindkét tábla adatainak használata diagrammok készítésére.
2. Témák beállítása.
3. Súgó készítése – Felhasználói dokumentáció készítése.
4. Statisztikák bővítése.
5. Szűrés/Keresés bővítése.
6. Az alkalmazás gazdasági szempontból való helyesbítése és bővítése.
7. Pályázatok kezelésének hozzáadása.
8. Diagrammok készítése során elérhető csoportosítási lehetőségek bővítése.
9. Több diagram típus hozzáadása és az azokhoz tartozó megvalósítás.
10. Pénznemek közötti automatikus váltás megvalósítása. Ehhez a pénznemek közötti váltószám lekérése és tárolása interneten keresztül.
11. Automatikus kijelentkeztetés, ha nincs felhasználói interakció.
12. Automatikus logolás a felhasználói tevékenységekről és azok megjelenítése egy külön UserControl segítségével.
13. Automatikus mentés készítése bizonyos időközönként.
14. Részletesebb validációk beviteli mezők esetén (gazdasági szempontból is).
15. Költségvetés (Bevételek, Kiadások), Kötelezettség és Követelés táblákhoz Típusok megfogalmazása. Jelenleg valami hasonló van megírva a BeKiKód segítségével, de részletesebb definíciókra volna szükség.
16. Éles környezetbe való deploy készítése.

# ***Irodalomjegyzék***

[1] LineSeries, Alberto Rodríguez, <https://v0.lvcharts.com/App/examples/v1/Wpf/Line>, utolsó megtekintés: 2025.03.14.

[2] ColumnSeries, Alberto Rodríguez, <https://v0.lvcharts.com/App/examples/v1/Wpf/Column>, utolsó megtekintés: 2025.04.05.

[3] RowSeries, Alberto Rodríguez, <https://v0.lvcharts.com/App/examples/v1/Wpf/Row>, utolsó megtekintés: 2025.01.07.

[4] PieSeries, Alberto Rodríguez, <https://v0.lvcharts.com/App/examples/v1/Wpf/Pie%20or%20Doughnut>, utolsó megtekintés: 2025.02.25.

[5] StackedColumnSeries, Alberto Rodríguez, <https://v0.lvcharts.com/App/examples/v1/Wpf/Stacked%20Column>, utolsó megtekintés: 2025.03.14.

[6] FontAwesome, @punker76, @BrainCrumbz, @MendelMonteiro, @robertmuehsig, @lufka, <https://github.com/charri/Font-Awesome-WPF/blob/master/README-WPF.md>, utolsó megtekintés: 2024.12.20.

[7] LiveCharts, Alberto Rodríguez, <https://github.com/Live-Charts/Live-Charts>, utolsó megtekintés: 2025.05.10.

[8] MVVM pattern, [RJ Code Advance EN](https://www.youtube.com/@RJCodeAdvanceEN), <https://www.youtube.com/watch?v=FGqj4q09NtA&list=PL5MBNRloYBsUG0Pe4oGRIK7QF1KOZou1U>, utolsó megtekintés: 2025.12.28.

[9] DropDown Navigation, [Abel Dutra UI](Abel%20Dutra%20UI), <https://www.youtube.com/watch?v=-JMS5z_HKIg&list=PL5MBNRloYBsUG0Pe4oGRIK7QF1KOZou1U>, utolsó megtekintés: 2025.01.05.

[10] Flip Card Animation, [C# Design Pro](https://www.youtube.com/@CSharpDesignPro), <https://www.youtube.com/watch?v=VoXk4vbTdwU&list=PL5MBNRloYBsUG0Pe4oGRIK7QF1KOZou1U>, utolsó megtekintés: 2025.05.15.

# ***Nyilatkozat***

Alulírott Lengyel Norbert Péter szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, Programtervező Informatikus Bsc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

2025. május. 19.

Lengyel Norbert Péter