

DEBRECENI SZC BEREKSZÁSZI PÁL TECHNIKUM

4032 Debrecen, Jerikó u. 17.
OM azonosító: 203033



☎: 52/ 503-150 Fax: 52/314-204
<http://www.dszcberegszaszi.hu>
E-mail: titkarsag@dszcberegszaszi.hu

Szakképesítés megnevezése: **Szoftverfejlesztő és -tesztelő technikus**

5-0613-12-03

Vizsgaremek

BOOKS

ASZTALI ALKALMAZÁS

Készítette:

Orszáczky Anna Katalin

Csapattagok:

Kovács István Zoltán

Debrecen, 2023

Tartalomjegyzék

Bevezetés	3
1. Fejlesztői dokumentáció	4
1.1. Fejlesztői környezet.....	4
1.2. Felhasznált programok, szolgáltatások.....	4
1.3. Tervezés	5
1.4. Vizuális felület	7
1.4.1. MainForm: Felhasználók	7
1.4.2. FormInsertUser: Új felhasználó	8
1.4.3. FormUpdateRole: Jogosultság módosítása.....	8
1.4.4. FormBooks: Könyvek	8
1.4.5. FormUpdateBook: Könyv adatainak módosítása	8
1.5. A programkód	9
1.5.1. User és Book osztályok	9
1.5.2. Database osztály.....	10
1.5.3. Program.cs	15
1.5.4. Felhasználók (MainForm)	16
1.5.5. Új felhasználó hozzáadása (FormInsertUser)	20
1.5.6. Felhasználó jogosultságának módosítása (FormUpdateRole)	21
1.5.7. Könyvek (FormBooks).....	23
1.5.8. Könyv adatainak módosítása (FormUpdateBook).....	23
1.5.9. BooksMessageBox osztály	23

1.6.	Futtatható fájl létrehozása	24
2.	Tesztelés	25
2.1.	Általam végzett tesztek	25
2.2.	Mások által végzett tesztek	27
3.	Felhasználói dokumentáció	28
3.1.	Az alkalmazás rendszerkövetelménye	28
3.2.	Az alkalmazás telepítése	28
3.3.	Az alkalmazás használata	29
3.3.1.	Felhasználók	29
3.3.2.	Új felhasználó hozzáadása	32
3.3.3.	Jogosultság módosítása	33
3.3.4.	Könyvek	34
3.3.5.	Könyv adatainak módosítása	36
4.	Fejlesztési lehetőségek	37
5.	Összegzés	38
6.	Hivatkozásjegyzék	39
7.	Ábrajegyzék	40

Bevezetés

A Debreceni SZC Beregszászi Pál Technikum szoftverfejlesztő és -tesztelő képzését záró szakmai vizsgára egy házi könyvtár elkészítése mellett döntöttük. Az alkalmazás alapvetően a csapattársam felesége számára készült, aki szeretne volna könyveit digitális környezetben rendszerezni.

Az én feladatom az erre a célra szánt webalkalmazáshoz tartozó adminisztrációs felület elkészítése volt, natív asztali alkalmazás formájában.

Fő feladata az oldal felhasználóinak karbantartása, ami a következő funkciókat foglalja magában:

- felhasználók keresése név és jogosultság alapján,
- új felhasználó felvétele, ahol beállítható admin jogosultság is,
- már felvett felhasználók hozzáférési jogosultságának módosítása
- és felhasználó törlése.

Ezenfelül az adatbázisban szereplő könyvek is hozzáférhetők:

- kereshetünk cím, szerző, típus szerint és az alapján, hogy a könyvet olvastuk-e már,
- módosíthatjuk a felvett könyvek adatait
- és törölhetünk a könyvek közül.

1. Fejlesztői dokumentáció

1.1. Fejlesztői környezet

A programot *Microsoft Windows 11* operációs rendszer alatt, a *Visual Studio 2022 Community* programfejlesztői környezetben készítettem *C#* nyelven.

A projekt típusa *Windows Forms App (.NET Framework)*.

1.2. Felhasznált programok, szolgáltatások

- Visual Studio 2022 Community
 - MySql.Data 8.0.32.1 NuGet package
 - BCrypt.Net-Next 4.0.3 NuGet package
- Figma 116.7.103
- XAMPP v3.3.0
 - Apache Web Server
 - MySql Database
- phpMyAdmin Database Manager 5.2.0
- GIMP 2.10.10
- SFX packager 2.0
- Resource Hacker 5.1.7

1.3. Tervezés

A felület tervezéséhez a *Figma* nevű webalkalmazást használtam. A legfontosabb szempont az volt, hogy a felület letisztult és áttekinthető legyen. A tervek a következő ábrákon láthatók.

1. ábra: A felhasználókat kezelő felület terve

Felhasználók

MainForm
tableLayoutPanelMain

dgvUsers

	id	Felhasználónév	Jogosultság

Felhasználó keresése

Felhasználó keresése

admin/user

Keresés

↻

Új felhasználó

Jogosultság módosítása

Felhasználó törlése

Könyvek

Bezárás

tbSearch

cmbFilter

btnSearch

btnRefresh

btnInsertUser

btnUpdateRole

btnDeleteUser

btnBooks

btnClose

Új felhasználó

FormInsertUser
panelInsertUser

Felhasználónév

tbUsernameIn

Jelszó

tbPasswordIn1

Jelszó ismét

tbPasswordIn2

Jogosultság

cmbRoleIn

Hozzáadás

Mégse

btnInsert

btnCancel

Jogosultság módosítása

FormUpdateRole
panelUpdateRole

Felhasználónév

JohnDoe123

tbUsernameUp

Jogosultság

admin/user

cmbRoleUp

Módosítás

Mégse

btnUpdate

btnCancel

2. ábra: A könyveket kezelő felület terve



A tervezés során meghatároztam a komponensek neveit, ami a későbbiekben megkönnyítette számomra a programkód megírását.

1.4. Vizuális felület

A formok felületének kialakítását nagyban leegyszerűsítették a fentebb bemutatott tervek.

A programba való belépés után először a felhasználókat megjelenítő ablak nyílik meg, aminek a *MainForm* nevet adtam.

1.4.1. MainForm: Felhasználók

A tartalom reszponzív megjelenítése érdekében az elemeket egy *TableLayoutPanel*-ben helyeztem el.

Az adatbázisban lévő felhasználók egy *DataGridView* komponensben jelennek meg. A táblázatban az azonosítójukat, nevüket és hozzáférési jogosultságukat láthatjuk, miszerint egyszerű felhasználóról, vagy adminról van szó.

A kereséshez a felhasználónevet – vagy annak egy részét – egy *TextBox*-ba kell beírni, a jogosultságot pedig egy *ComboBox* elemei közül választhatjuk ki. A kereső elemeket tartalmazó *GroupBox*-ban még két gomb található, az egyik a keresés elindításáért felel, a másikkal pedig keresés után ismét visszatölthetjük az összes felhasználót.

A fő formon ezenkívül még öt gomb található. Az „Új felhasználó” és a „Jogosultság módosítása” gombra kattintva modális ablakok jelennek meg, így amíg ezek meg vannak nyitva, más műveletet nem végezhetünk a programban. A „Felhasználó törlése” gombra kattintva egy egyszerű felugró ablakban erősíthetjük meg, hogy valóban törölni kívánjuk-e a kijelölt felhasználót.

A „Könyvek” gombra kattintva a program elrejtí a jelenlegi ablakot, és átnavigál minket a következőre, ahol értelemszerűen a könyvek jelennek meg. A „Bezárás” gomb segítségével kiléphetünk az alkalmazásból.

1.4.2. FormInsertUser: Új felhasználó

Ebben az ablakban *label* elemek teszik egyértelművé a beviteli mezők funkcióját. A felhasználó nevet és a jelszót egyszerű *TextBox* komponensekbe kell beírni, viszont a jelszavak bevitelére szánt mezők az operációs rendszer alapértelmezett helyettesítő karaktereivel fogják megjeleníteni a beírt tartalmat.

A jogosultságot itt is egy *ComboBox* elemei közül választhatjuk ki.

A felületen még két gomb látható: a „Hozzáadás” gombbal felvehetjük a felhasználót, a „Mégse” gomb segítségével pedig bármilyen művelet végrehajtása nélkül bezárhatjuk az ablakot.

1.4.3. FormUpdateRole: Jogosultság módosítása

A kiválasztott felhasználó neve egy csak-olvasható *TextBox*-ban jelenik meg, a jogosultságot pedig továbbra is az eddigiekkel megegyező módon választhatjuk ki. Az új felhasználó felvételére alkalmas formával megegyezően itt is két gombot láthatunk: az egyikkel végrehajthatjuk a módosítást, a másikkal bezárhatjuk az ablakot.

1.4.4. FormBooks: Könyvek

Ugyanazokat az elemeket használja a könyvek adatainak megjelenítésére, mint a *MainForm* a felhasználók megjelenítésére. Mindössze annyi az eltérés, hogy eggyel több *ComboBox* és *TextBox* található rajta, mivel több feltétel szerint kereshetünk. Ezenkívül eggyel kevesebb gomb, mert az asztali alkalmazásból jelenleg nincs lehetőségünk új könyvet felvenni.

1.4.5. FormUpdateBook: Könyv adatainak módosítása

A szerkezete megegyezik a felhasználó jogosultságának módosításához szükséges form szerkezetével.

1.5. A programkód

1.5.1. User és Book osztályok

Először létrehoztam a felhasználók kezeléséhez szükséges osztályt a *User.cs* fájlban. Ez tartalmazza a *User* (felhasználó) típus definícióját. A *User* típusú entitások mindössze három tulajdonsággal rendelkeznek: azonosítóval (*id*), felhasználónévvel (*username*) és hozzáférési jogosultsággal (*role*). A név és a jogosultság *string*, azaz szöveg típusú, az azonosító pedig – az adatbázisnak megfelelően – előjel nélküli egész szám (*uint*), amiben 32 biten tárolhatunk pozitív számokat 0-tól 4294967295-ig. Az azonosító csak-olvasható.

Az osztály ezenkívül tartalmazza a konstruktort, ami az objektum létrehozásakor beállítja az előbb felsorolt változók kezdőértékeit. Ezenkívül egy felüldefiniált *ToString()* metódust, aminek segítségével egy *User* objektum kiírásakor értelmezhető szöveget kapunk vissza, konkrétan az osztályban szereplő változók tartalmát. Az utóbbit a későbbiekben csak hibakereséshez használtam.

A *Book* osztály definiálása ugyanerre a sémára épül, csupán a tulajdonságok különböznek. A könyvek azonosítója ugyanúgy egy *uint* típusú változóban van eltárolva, a következők pedig *string*-ben: cím (*title*), szerző (*author*), típus (*type*) és befejezett (*finished*).

A *type* változó azt tárolja, hogy az adott könyv nyomtatott vagy digitális formában van-e meg, a *finished* tartalma pedig megmutatja, hogy a tulajdonos olvasta-e már a könyvet, vagy sem.

1.5.2. Database osztály

A program a *kapcsolat alapú adatelérési modellt* használja, hogy a felhasználó által látott adatok mindig megegyezzenek az adatbázisban tárolt legfrissebb adatokkal, és hogy elkerüljük a lehetséges ütközéseket. Ezzel az adatelérési móddal – a kapcsolat nélküli adatelérési modellel szemben – nem készül lokális másolat az adatokról, hanem kérés esetén kiépül a kapcsolat, majd a forgalom lebonyolítása után lezárul.

Adatbáziskapcsolat létrehozása

Ennek megvalósításához hozzáadtam a projekthez egy harmadik féltől származó kiegészítő programot, név szerint az *Oracle* cég által kezelt *MySql.Data NuGet* csomagot. Ez biztosítja a szükséges *Connection*, *Command* és *DataReader* objektumokat. Ezután a *Database.cs* fájlban először a *using* kulcsszó segítségével használatba vettem a felsorolt osztályok alkalmazásához szükséges *MySql.Data.MySqlClient* névteret.

Az adatbázis kezeléséhez szükséges műveleteket tartalmazó osztály első feladata a kapcsolat felépítése. Az ehhez szükséges *MySqlConnection* típusú *connection* objektumot osztályváltozóként határoztam meg, mint ahogy az adatbázis parancsok futtatásához szükséges *MySqlCommand* típusú *cmd* objektumot.

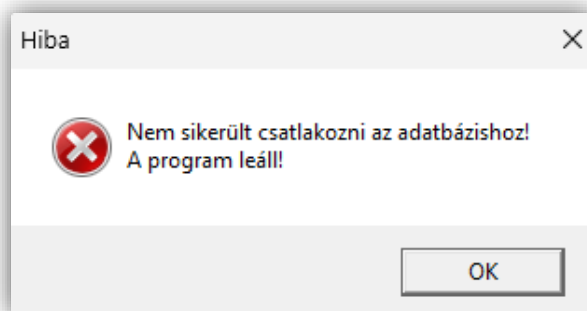
A kapcsolat tényleges felépítése a konstruktor segítségével történik, ahol az eléréshez szükségesek adatok átadhatók paraméterként, illetve amennyiben példányosításkor nem adunk meg argumentumlistát, úgy az alapértelmezett paraméter értékei lesznek érvényesek, miszerint a *books* nevű adatbázis a saját gépen található, és a *root* felhasználóval férünk hozzá, aminek a *XAMPP* használata esetén nincs jelszava.

Ezeket az értékeket a *MySqlConnectionStringBuilder* típusú *sb* objektum segítségével határozzuk meg, majd ezeket a paramétereket lekérdezzük, és az *sb.ConnectionString* felhasználásával átadjuk a *connection* objektumnak.

A kapcsolat felépítésének sikerességének tesztelése egy *try-catch* blokkpáros segítségével történik. A *try* blokkban a program megpróbálja megnyitni a kapcsolatot, majd inicializálja a korábban deklarált *cmd* objektumot, végül lezárja a kapcsolatot.

Amennyiben a végrehajtás közben valamilyen hiba lép fel, a vezérlés a *catch* ágba kerül, ahol a felhasználót egy hibaüzenet tájékoztatja a csatlakozás sikertelenségéről, majd a program leáll, mivel az adatbázis elérése nélkül minden funkciója értelmetlen.

3. ábra: Sikertelen csatlakozást jelző hibaüzenet



Felhasználók adatainak lekérdezése

A *Database* osztály következő metódusa – a *SelectedUsersList()* – egy az adatbázisban szereplő rekordokat tartalmazó listát ad vissza. A függvényt úgy írtam meg, hogy alkalmas legyen felhasználók közötti keresésre is. Erre azért volt szükség, mert amikor a keresett felhasználók közül törölünk, vagy módosítjuk valamelyik jogosultságát, akkor a művelet után a *MainForm* ablak újratöltésekor is a keresett felhasználók jelenjenek meg.

A metódus paraméterként egy *User* objektumot vár, ebből szerzi meg a keresési feltételeket: a felhasználónevet és a jogosultságot, amiket beilleszt az sql utasításba. Ha nincs keresés, tehát üres stringet kap, akkor a keresési feltétel helyén csak két százalékjel marad, így minden elem megfelel a feltételnek. A program első betöltődésekor természetesen nincs megadva semmilyen keresés, így az összes rekordot betölti a listába.

Az sql utasítást a *cmd* objektum *CommandText* tulajdonságában adjuk meg.

A kapcsolatot megnyitását ismét *try-catch* kivételkezelés ellenőrzi. A *cmd* objektum *ExecuteReader()* metódusával a program lefuttatja a *SELECT* utasítást. Az így létrehozott tábla tartalmát egy *dr* nevű *MySqlDataReader* típusú objektumban tároljuk.

Ebből az adatokat a *Read()* metódus segítségével érjük el. A listát egy *while* előtesztelő ciklusban töltjük fel. A ciklus addig fut, amíg a *Read()* metódus hamis értéket nem ad vissza, ami akkor következik be, amikor már nem tud több adatot kiolvasni a táblából.

A cellák értékeit a *dr.GetUInt32()* és *dr.GetString()* utasításokkal szerezzük meg, majd ezen értékek alapján létrehozunk egy új *User* típusú objektumot, amit hozzáadunk a listához

.

A listához adás előtt a jogosultság megnevezését átalakítottam, hogy ha az adatbázisból a „user” értéket kapja, akkor azt „felhasználóként” vegye fel. Az „admin” megnevezés átalakítását szintén megadtam, arra az esetre, hogy ha a későbbiekben az adatbázisban a jogosultságok azonosítóval lennének jelölve, és nem szöveggel, akkor a kódot könnyen hozzá lehessen alakítani.

Ha a lista feltöltése valamilyen oknál fogva sikertelen, a felhasználó hibüzenetet kap, majd a program leáll.

Felhasználónevek lekérdezése

A *UsernameList()* nagyon hasonló az előzőhöz, azzal a különbséggel, hogy az ezzel létrehozott lista csak a felhasználók neveit tartalmazza. Ezt a metódust akkor használom, amikor a program azt ellenőrzi, hogy az új felhasználó neve elérhető-e.

Azért döntöttem úgy, hogy erre a feladatra különálló függvényt hozok létre, és nem az előzőt használom fel, hogy ennél az egyszerűbb feladatnál a programnak ne kelljen az adatbázis összes mezőjének tartalmát lekérdeznie, és a ne az objektumok tulajdonságait kelljen vizsgálnia, hanem egy egyszerűbb *string* listában kelljen keresnie.

Új felhasználó hozzáadása

Az *InsertUser()* eljárás két paramétert vár: egy *User* objektumot valamint egy *string* típusú jelszót, mivel az *User* osztálynak ilyen adattagja nincs, hiszen a programnak egyetlen funkciója sincs ezenkívül, ami hozzáférést igényelne az adatbázis *password* mezőjéhez. Ezenfelül biztonságosabb is, ha a program nem kérdezi le a tárolt a jelszavakat.

Az új felhasználó felvétele az *INSERT* utasítás segítségével történik. Az *sql injection* támadás elkerülése érdekében a felvenni kívánt adatokat nem közvetlen a parancsban adom meg, hanem külön paraméterként. Így a parancs és az adatok külön, két lépésben lesznek elküldve, ami hozzájárul a biztonságosabb működéshez.

Mivel az *sql* utasítás nem lekérdezés, a *cmd.ExecuteNonQuery()* utasítással futtatom le az előbb megadott parancsot. Ez visszaadja az érintett sorok számát, így könnyen ellenőrizhetem, hogy az adatbázis művelet helyesen végbement-e. Amennyiben az eredmény 1, a művelet sikeres volt, más esetben nem. Ezt az *IsSuccessfulMessageBox()* metódusban kezelem.

Jogosultság módosítása

Az *UpdateRole()* metódus szerkezetében megegyezik az előbbivel, azzal a különbséggel, hogy itt paraméterként csak egy *User* objektumot adunk át, aminek csak két tulajdonságát fogja felhasználni: a felhasználó jogosultságát, amit a későbbiekben módosítani kívánunk, és az azonosítót, ami alapján meghatározzuk, hogy mely felhasználón végezzük el a műveletet.

A feladatnak megfelelően a *cmd.CommandText* tartalma egy *UPDATE* sql utasítás. Az adatok itt is paraméteres átadással történnek, a parancs lefutásának sikerességét az érintett sorok vizsgálatával ellenőrzöm, majd az *IsSuccessfulMessageBox()* metódus segítségével jelzem a felhasználónak, hogy sikeres volt-e a művelet.

Felhasználó vagy könyv törlése

Az elem törlését végrehajtó metódust úgy alakítottam ki, hogy argumentumok megadásával alkalmas legyen felhasználó és könyv törlésére egyaránt. Ehhez két paraméterre van szükség:

- Az *id* segítségével lehet meghatározni a törölni kívánt elemet. Szöveggént adom át, mert az sql parancsba így kell majd beilleszteni, felesleges oda-vissza alakítani szövegből számmá, majd számból szöveggé.
- A *tableName* helyére kell majd behelyettesíteni a kiválasztott tábla nevét, ami esetünkben *book* vagy *users* lehet. Ez az érték kerül be az sql parancsba, ez dönti el, hogy a könyveket, vagy a felhasználókat tartalmazó táblából törölünk rekordot.

A metódusban először felvettem két változót, amiket a program automatikusan állít be annak megfelelően, hogy paraméterként milyen táblanevet kapott.

- A *messageText* értéke „Könyv” vagy „Felhasználó” lehet. Ezt az értéket nem az sql utasítás használja, hanem arra szolgál, hogy a sikerességről tájékoztató felugró ablakban helyes szöveg jelenjen meg.
- Az *idFieldName* változó bevezetésére azért volt szükség, mert a *rent* adattábla miatt meg kellett különböztetni a felhasználó azonosítóját a könyv azonosítójától, és nem maradhatott mindkettőben *id* a mezőnév. Így ez is felvehet kétféle értéket: *userid*-t és *bookid*-t.

Ezt követően a *try* ágban lefut a *DELETE* sql parancs. Amennyiben futás közben valamilyen hiba keletkezik a program átlép a *catch* ágba, ami a *MySqlException* típusú hibákat kezeli.

Fontos, hogy itt is le legyen zárva a kapcsolat.

A különböző *MySQL* hibaszámú esetek kezelését *switch-case* szerkezettel valósítottam meg, így a programrész könnyen bővíthető további egyedi hibaüzenetekkel.

```
catch (MySqlException ex)
{
    connection.Close();

    switch (ex.Number)
    {
        case 1451:
            BorrowedBookErrorMessage(tableName);
            break;
        case 1064:
            BooksMessageBox.Error("A program nemlétező  
adattáblából próbál törölni.\nA művelet sikertelen.");
            break;
        default:
            BooksMessageBox.Error(ex.Message);
            break;
    }
}
```

4. ábra: Felhasználó vagy könyv törléséhez tartozó kivételkezelés

Az 1451-es hiba akkor jelentkezett, amikor a felhasználó olyan könyvet próbált törölni, ami ki volt kölcsönözve, vagy olyan felhasználót, akinél kölcsönvett könyv volt. Az ehhez tartozó üzenet a *BorrowedBookErrorMessage()* metódusban adtam meg, az áttekinthetőség érdekében.

Ez a megadott tábla neve alapján eldönti, hogy könyv vagy felhasználó törlésének sikertelenségéről kell-e tájékoztatni a felhasználót.

A másik specifikus hiba akkor keletkezett, amikor a program nem megfelelő táblanevet kapott, így nemlétező adattáblából próbált törölni.

Könyvek adatainak lekérdezése és módosítása

A *SelectedBooksList()* és *UpdateBook()* metódusok működésüket tekintve megegyeznek a *SelectedUsersList()* és *UpdateRole()* metódusokéval. A különbség, hogy míg a felhasználók esetén *User* típusú objektumokat kezelnek, a könyvek esetén értelemszerűen *Book* típusút, és mivel az utóbbinak több tulajdonsága van – hiszen a hozzátartozó adatbázistáblában is több mező van –, a metódusainak is több paramétert kell kezelniük.

Visszajelzés a művelet sikerességéről

A *IsSuccessfulMessageBox()* eljárást használja minden metódus, ami visszajelzést ad a művelet sikerességéről. Két paraméterre van: az egyik egész szám típusú, ami az érintett sorok számát vizsgálja, a másik pedig szöveg. Meghíváskor az utóbbinál adható meg, hogy az értesítés milyen műveletre vonatkozik: új elem felvételére, módosításra vagy törlésre. Ha 1 sor érintett, akkor sikeres végrehajtásról kapunk üzenetet, ha nem, akkor sikertelenről.

1.5.3. Program.cs

Az előbb részletezett *Database* osztályt, ezenkívül a *MainForm* és *FormBooks* osztályokat – amikről ezután lesz szó – a *Program.cs* fájlban példányosítottam, így ezek a későbbiekben bárhol felhasználhatók.

1.5.4. Felhasználók (MainForm)

A program elindítása után az első ablak, ami megjelenik az a felhasználókat kezelő form. Ennek betöltődésekor a program először feltölti a jogosultság szűréséhez szükséges *ComboBox*-ot a választható értékekkel. A *SelectedIndex* nullára állításával alapértelmezetten a „minden jogosultság” verzió jelenik meg.

A következő két metódus a *DataGridView* táblázat létrehozásáért és adatokkal való feltöltéséért felel.

Táblázat létrehozása

A *DataGridViewCreate()* metódusban először a táblázat általános tulajdonságait adtam meg, majd az egyes oszlopokét.

- Először a *SelectionMode* tulajdonság segítségével beállítottam, hogy ha a felhasználó rákattint egy cellára, akkor a hozzátartozó egész sort jelölje ki, így a program mindig számíthat arra, hogy kiválasztáskor az egész rekordot visszakapja.
- Az *AutoSizeColumnsMode* tulajdonság *Fill* értékre állításával az oszlopok kitöltik a teljes rendelkezésre álló területet, így nem keletkezik hézag a táblázatban, mikor elfogynak a mezők.
- A *dgvUsers.MultiSelect = false* parancs megakadályozza, hogy a felhasználó egynél több sort jelölhessen ki. Mivel a program minden funkciója csak egy felhasználóra vonatkozik, ez elengedhetetlen.
- A *RowTemplate.Height* beállításával növeltem a sormagasságot, hogy a táblázat szellősebb, áttekinthetőbb legyen.
- A *RowHeadersWidth* a sorok fejlécének szélességére vonatkozik, ami az automatikus értékkel aránytalanul széles volt.

A következő feladat az egyes oszlopok megadása és tulajdonságainak beállítása volt. Minden oszlopot a *Name* értéke azonosít. A *HeaderText*-ben megadott szöveg jelenik meg az oszlopok fejlécén. A *CellTemplate* megadja, hogy milyen típusú celláról van szó, ami itt minden esetben *TextBoxCell*. A *SortMode* tulajdonság automatikusra állítva azt eredményezi, hogy a fejlécekre kattintva az adott oszlop cellái növekvő sorrendbe rendeződnek, illetve második kattintásra csökkenőbe.

A *colId* azonosítókat tartalmazó oszlop még két további tulajdonságot kapott, az egyikkel a benne lévő számok jobbra igazítva jelennek meg (*colId.DefaultCellStyle.Alignment*), a másik pedig eléri, hogy a cellák szélessége a benne lévő tartalomhoz igazodjon (*colId.AutoSizeMode*). Ezt azért állítottam be, mert nélküle – az *AutoSizeColumnsMode* tulajdonság miatt – egyenlően osztotta el az oszlopok között a rendelkezésre álló helyet, így a néhány számjegyes azonosítók feleslegesen széles cellákban helyezkedtek el.

Táblázat feltöltése adatokkal

A *DataGridViewUpdate()* metódus feladata a táblázatot feltölteni adatokkal. Mivel ez többször is lefut, az első parancs kiüríti a sorokat. Enélkül a program újra és újra hozzáadná az adatbázis tartamát a táblázathoz.

Egy *foreach* ciklus bejárja a paraméterként kapott *User* típusú listát (ami a felhasználók adatait fogja tartalmazni), és a benne lévő értékekkel feltölti a *DataGridView* táblázatot. Ez úgy történik, hogy először egy sort adunk hozzá, majd ebbe a sorba beillesztjük a cellákat a korábban megadott formázási beállításokkal együtt.

Felhasználók keresése

A *DataGridViewUpdateSearch()* eljárás felel azért, hogy a táblázatban csak az aktuális keresésnek megfelelő rekordok jelenjenek meg. A keresési feltételeket a vizuális felületen megadott *tbSearch* nevű *TextBox*-ból és a *cmbFilter* nevű *ComboBox*-ból nyeri. Az utóbbit a *ComboBoxSelectedItem()* metódus állítja be a megfelelő szöveges értékre („admin” vagy „user”) a kiválasztott *ComboBox* elem indexe alapján.

A keresett értékek alapján a program létrehoz egy *User* típusú objektumot. Mivel *id* alapján nem tudunk keresni, az első paraméter értéke lényegtelen. Ezután lefut a *Database* osztályban definiált *SelectedUsersList()* metódus, ami argumentumként megkapja az előbb létrehozott *User* típusú objektumot. Az így kapott listát – amiben már csak a keresési feltételeknek megfelelő felhasználók fognak szerepelni – kapja meg a *DataGridViewUpdate()* eljárás, és ezekkel az értékekkel fogja feltölteni a táblázatot.

A *DataGridViewUpdateSearch()* azért lett publikus, hogy az új felhasználót hozzáadó és a jogosultságot módosító form is elérhesse, és az adott művelet elvégzése után frissíthesse *dgvUsers* táblázatot.

A *btnSearch* „Keresés” gomb lenyomásakor ugyanez a metódus fut le, ahogy a *btnRefresh* „Frissítés” gomb lenyomásakor is, azzal a különbséggel, hogy ez utóbbi a meghívás előtt kiüríti a keresett felhasználónév megadására szánt mezőt, és nullára állítja a jogosultságokat tartalmazó *ComboBox* kiválasztott elemének indexét. Ezzel éri el, hogy az adatbázis *users* táblájának minden eleme megjelenjen.

Keresés enter leütésével

Mivel kényelmesebbnek tartom enter leütésével keresni, megadtam, hogy mind a *tbSearch* mezőben, mint a *cmbFilter* listában leütött enter esetén végrehajtsa „Keresés” gomb kattintás eseményét. Ezt az *EnterSearch()* metódus végzi el, ami publikus, hogy a könyvek közötti kereséskor is felhasználhassam.

ComboBox elem indexének és értékének megfelelő összerendelése

A *ComboBoxSelectedItem()* szintén egy publikus metódus, ami három paramétert vár. Az első egy egész szám: az adott *ComboBox* kiválasztott elemének indexe, a másik kettő pedig szöveg, ahol meg kell adni, hogy az index alapján milyen szöveges értéket szeretnénk visszakapni (ez a felhasználók esetén „admin” vagy „user” lehet). Ha az index 1, akkor az első szöveges értéket kapjuk vissza, ha 2, akkor a másodikat. Minden más esetben pedig üres stringet. Így ha a 0. indexű elem van kiválasztva, a szűrési feltételnek minden elem megfelel majd.

Felhasználó törlése

A *btnDeleteUser* gomb lenyomásakor a program először ellenőrzi, hogy egy sor van-e kiválasztva. Ha nem, hibaüzenetet kapunk, ha igen, akkor először az alkalmazás egy felugró ablakban megerősítést kér, hogy valóban törölni szeretnénk-e a felhasználót. Ha a válasz igen, akkor a kiválasztott sorból kinyerjük az adott felhasználó azonosítóját, majd ezt átadjuk a *Database* osztályban létrehozott *DeleteUserOrBook()* metódusnak, ami a kapott *id* és táblanév alapján törli az adatbázisból a megfelelő elemet. Végül automatikus frissíti a táblázat elemeit, hogy a törölt felhasználó már ne jelenjen meg.

Megjelenő ablak tulajdonságainak beállítása

A *ShowForm()* publikus eljárás egy *Form* objektumot vár paraméterként. A kapott formot úgy állítja be, hogy felvegye annak az ablaknak a tulajdonságait, amelyről meg lett nyitva. Amennyiben az előző ablak teljes képernyős módban volt, úgy a megjelenő ablak is abban lesz. Ha nem, akkor a program beállítja, hogy ugyanazon a pozíción és ugyanabban a méretben jelenjen meg.

Ez a metódus fut le a *MainForm*-on lévő *btnBooks* gomb lenyomásakor, és *FormBooks*-on lévő *btnUsers* gomb kattintás eseményében is ezt használtam fel.

1.5.5. Új felhasználó hozzáadása (FormInsertUser)

Mivel a webalkalmazás *bcrypt*-et használ a jelszavak titkosítására, így az asztali alkalmazásnak is erre a megvalósításra volt szüksége. Hogy ez működjön, telepítettem a *BCrypt.Net-Next* csomagot. Ennek segítségével egyszerűen, két parancs segítségével elvégezhettem a titkosítást. A *BCrypt.Net.BCrypt.GenerateSalt(10)* létrehozza a *salt*-ot, majd ennek felhasználásával a *BCrypt.Net.BCrypt.HashPassword(password, salt)* végrehajtja a titkosítást.

Az új felhasználó létrehozásához szükséges adatokat a *tbUsernameIn*, *tbPasswordIn1*, *tbPasswordIn2* és *cmbRoleIn* elemekből kapjuk meg. Ahhoz, hogy a felhasználó felvétele megtörténjen, több feltételnek kell teljesülnie.

- Az első ellenőrzés arra vonatkozik, hogy minden mező ki van-e töltve. Ezt a logikai értékkel visszatérő *FieldsAreFilled()* függvény vizsgálja. Ha a mezők nincsenek kitöltve, vagy a kiválasztott *ComboBox* elem értéke nem „admin” vagy „user”, akkor hamis eredménnyel tér vissza.
- A felhasználónév elérhetőségét a *UsernameIsAvailable()* függvény ellenőrzi. Ehhez meghívja a *Database* osztályban definiált *UsernamesList()* metódust, ami egy string típusú listát ad vissza, mely tartalmazza az adatbázisban szereplő összes felhasználó nevét. Egy *foreach* ciklus bejárja ezt a listát, és a beírt felhasználónevet összehasonlítja velük. Ha egyezést talál, hamis értéket ad, és tájékoztatja a felhasználót, hogy a név már foglalt.
- A *UsernameMinimumLength()* biztosítja, hogy a felhasználónév minimum 3 karakterből álljon.
- Hogy a felhasználónév ne legyen túl bonyolult, a következő korlátozást vezettem be: csak az angol abc kis- és nagybetűit, számokat, pontot, kötőjelet és aláhúzásjelet tartalmazhat. Ezt egy reguláris kifejezés segítségével oldottam meg, amihez a *Regex* osztály felhasználására volt szükség, ami a *System.Text.RegularExpressions* névtérben található.
- Ahhoz, hogy a jelszó megfeleljen bizonyos biztonsági elvárásoknak, szintén reguláris kifejezést használtam. A felhasznált kifejezést a következő oldalon találtam: <https://www.c-sharpcorner.com/UploadFile/jitendra1987/password-validator-in-C-Sharp/> Ez biztosítja, hogy a jelszó legalább 8 karakter hosszú legyen, tartalmazzon kis- és nagybetűt, számot és speciális karaktert.

- Végül, az utolsó feltétel a jelszavak egyezését ellenőrzi egy egyszerű összehasonlítással, amit a *PasswordMatch()* függvényben adtam meg.

Ha a beírt adatok minden feltételnek megfelelnek, a jelszó titkosítás végbemegy és az *InsertUser()* metódus beilleszti az új felhasználót az adatbázisba.

1.5.6. Felhasználó jogosultságának módosítása (FormUpdateRole)

Ha a *MainForm* „Felhasználók” ablakban a „Jogosultság módosítása” gombra kattintunk, akkor először egy *if-else* szerkezet vizsgálja, hogy van-e kiválasztott felhasználó. Ha nincs, hibaüzenetet kapunk, ha van, akkor a *FormUpdateRole* form jelenik meg modális ablak formájában. Tehát amíg be nem zárjuk, vagy a módosítás után magától be nem záródik, más műveletet nem végezhetünk a programban.

A *FormUpdateRole.cs* fájlban először osztályváltozókként deklaráltam a kiválasztott felhasználó adatait, mivel ezeket több metódusban is változatlanul használtam fel.

Kiválasztott felhasználó adatainak betöltése

Az ablak betöltésekor megjelenik a *MainForm*-on kiválasztott felhasználó neve és jogosultsága. Ezeket az adatokat a kiválasztott sor celláinak értékeiből szerzi meg a program.

A név megjelenítéséhez a *tbUsernameUp* mező *Text* tulajdonságának kellett átadni a megfelelő cella szöveges tartalmát.

A *cmbRoleUp ComboBox* megjelenítendő elemének indexét a *SelectedRoleIndex()* függvény adja meg, ami egy szöveg típusú paramétert vár, mégpedig a kiválasztott felhasználó jogosultságát (ami „admin” vagy „felhasználó” lehet). Ha ez az érték „admin”, akkor a visszaadott érték 0, ha más, akkor 1. Ennek segítségével állítható be a *ComboBox* a megfelelő értékre.

Módosítás végrehajtása

A *btnUpdate* nevű „Módosítás” gomb megnyomásakor a program először kiolvassa a beállított jogosultság indexét, amit összehasonlít a kiválasztott felhasználó módosítás előtti jogosultságának indexével. Ha a két szám egyezik, értesítést kapunk róla, hogy nem történt módosítás.

Amennyiben azonban eltérnek – tehát más jogosultságot választottunk, mint ami eredetileg volt –, a program a *RenameRoleValue()* függvénnyel a módosított jogosultság indexe alapján meghatározza, hogy milyen szöveges értéket küldjön majd az adatbázisnak („admin” vagy „user”).

Ezután a kiválasztott felhasználó azonosítójával, nevével és a módosított jogosultsággal egy új *updatedUser* nevű *User* típusú objektumot hoz létre, amit megkap az adatbázis műveleteket tartalmazó osztályban létrehozott *UpdateRole()* metódus, mely elvégzi a felhasználó tényleges frissítését az adatbázisban.

1.5.7. Könyvek (FormBooks)

A *FormBooks* struktúráját tekintve megegyezik a *MainForm*-mal.

Miután feltöltötte a *ComboBox*-okat a megfelelő értékekkel, három metódus segítségével létrehozza és feltölti a *dgvBooks DataGridView* táblázatot:

- A *DataGridViewBooksCreate()* meghatározza a táblázat és az oszlopok tulajdonságait,
- a *DataGridViewBooksUpdate()* feltölti sorokkal, azon belül cellákkal,
- végül a *DataGridViewBooksUpdateSearch()* elvégzi a keresést, hogy csak a kiválasztott elemek jelenjenek meg.

A *btnUpdateBook* gombra kattintva egy *FormUpdateBook* objektum jelenik meg modális ablakként, a *btnDeleteBook* gomb kattintás eseménye pedig meghívja az adatbázis osztály *DeleteUserOrBook()* metódusát, amivel a program eltávolítja az adatbázisból a kiválasztott könyvet.

1.5.8. Könyv adatainak módosítása (FormUpdateBook)

A *FormUpdateBook* szerkezete azonos a *FormUpdateRole* szerkezetével. Először betölti az előzőleg kiválasztott könyv adatait, majd ellenőrzi, hogy történt-e módosítás, illetve a bevitt adatok helyesek-e. Amennyiben igen, az új értékek alapján példányosít egy *updatedBook* nevű *Book* objektumot, amit az *UpdateBook()* metódussal elküld az adatbázisnak.

1.5.9. BooksMessageBox osztály

Ebben az osztályban egyedi felugró ablakokat definiáltam, hogy amennyiben valamit változtatni akarok a megjelenésen (például lecserélni az üzenet mellett megjelenő ikont), ne kelljen végigböngészni az egész kódot *MessageBox*-okat keresgélve, hanem elég legyen egy helyen átírni, és a változtatás mindenhol megtörténjen.

1.6. Futtatható fájl létrehozása

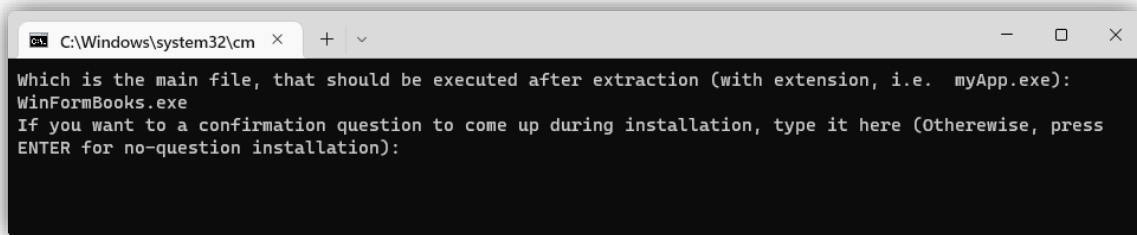
A hordozható (*portable*) exe fájlt az *SFX packager* program segítségével készítettem el.

Ennek a menete mindössze annyi, hogy az elkészített alkalmazást tartalmazó mappát (*WinFormBooks\bin\Debug*) a *packager_installer.bat* fájl fölé kell húzni.

Ezután a program konzolos felületen megkérdezi, hogy a mappában lévő fájlok közül melyik induljon el. Itt kiterjesztéssel együtt megadtam a *WinFormBooks.exe* fájlt.

Ezenkívül még azt a kérdést teszi fel, hogy milyen megerősítő kérdés jelenjen meg a „telepítés” alatt, itt csak entert ütöttem, mivel ezt nem ítéltam szükségesnek.

Ezt követően az *SFX packager* legenerálja a hordozható futtatható fájlt, így a felhasználónak nem kell a telepítéssel foglalkoznia.



5. ábra: Az *SFX packager* felülete

Továbbá az operációs rendszer által automatikusan beállított ikont egyedire cseréltem a *Resource Hacker* elnevezésű programmal.

2. Tesztelés

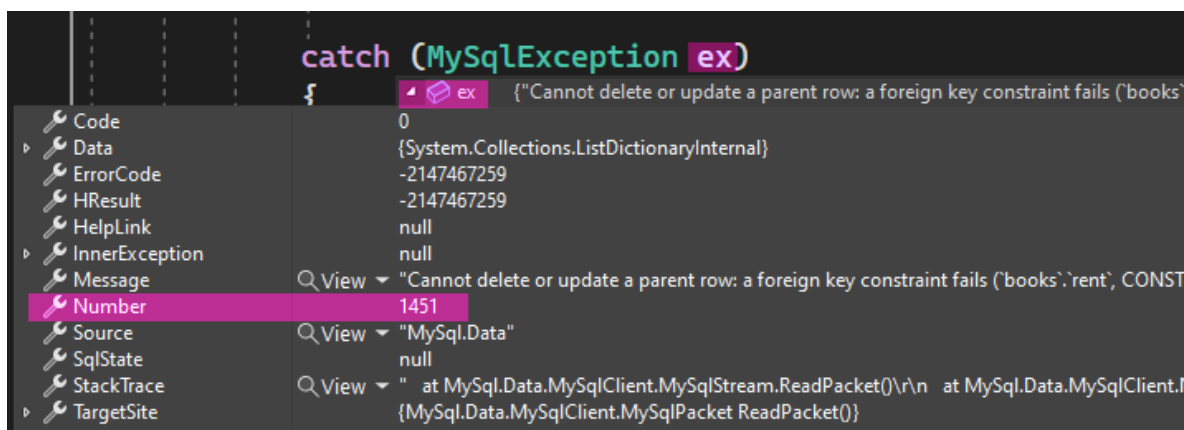
2.1. Általam végzett tesztek

Az első teszteket én végeztem *Microsoft Windows 11* operációs rendszer alatt.

Az egyik hiba, amivel találkoztam, a kölcsönzés funkció bevezetése után keletkezett. Egyes felhasználók és könyvek törlése meghiúsult, mivel az adatbázis nem engedte a *rent* táblában is szereplő elemek eltávolítását.

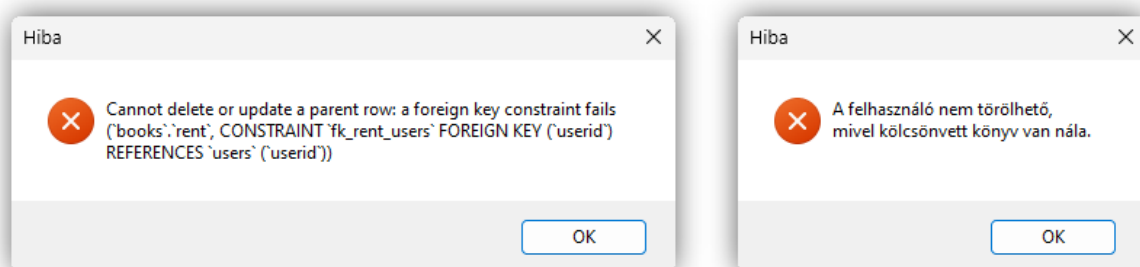
A megjelenő hibaüzenet azonban egy átlagfelhasználó számára egyáltalán nem volt informatív. Ráadásul a hiba elkapása után a program valamilyen oknál fogva összeomlott.

Ahhoz, hogy ezeket a problémákat megoldjam, a *catch* ágban elhelyeztem egy töréspontot (*breakpoint*), hogy hibakezelő módban elérhessem a kivétel részleteit. Az *ex* objektum vizsgálatából kiderült a *MySQL* hiba száma.



6. ábra: Az *ex* objektum részletei

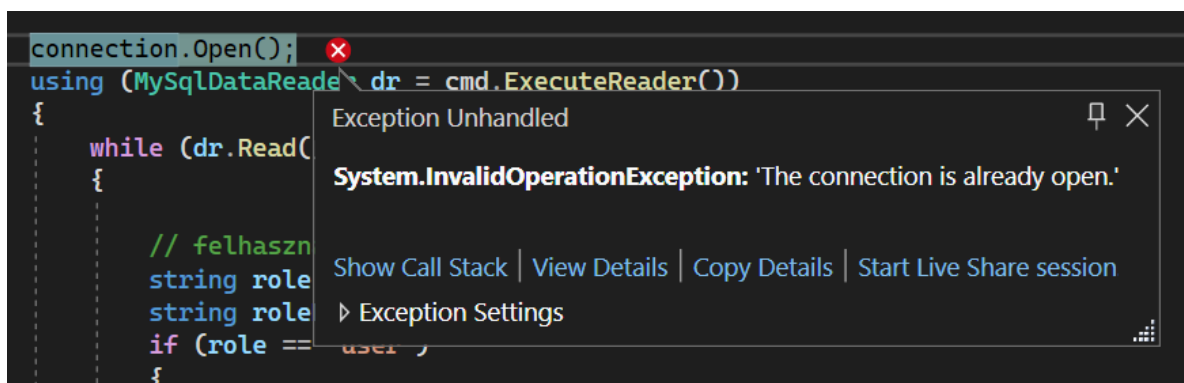
Ezt a hibaszámot felhasználtam arra, hogy az *ex* objektumhoz tartozó alapértelmezett hibaüzenet helyett egyedi, közérthető értesítéseket adjak meg.



7. ábra: Sikertelen törléskor megjelenő alapértelmezett és egyedi hibaüzenet

Az is egyértelművé vált, hogy a program miért omlott össze a hiba elkapása ellenére is.

Mivel a *catch* ágban nem zártam le a megnyitott adatbáziskapcsolatot, így amikor az alkalmazás megpróbálta lekérdezni az adatokat, nem kezelt kivétel jött létre.



8. ábra: Lezáratlan kapcsolat esetén jelentkező kivétel

Miután pótoltam ezt a hiányosságot, és minden *catch* ág elejére beillesztettem a *connection.Close()* parancsot, az alkalmazás problémamentesen működött.

2.2. Mások által végzett tesztek

Az alkalmazás tesztelését *Windows 10* operációs rendszeren a projektfeladatban résztvevő csapattársam, Kovács István Zoltán végezte. Ő segített megoldást találni arra, hogy a jelszavak titkosítása *bcrypt* által történjen.

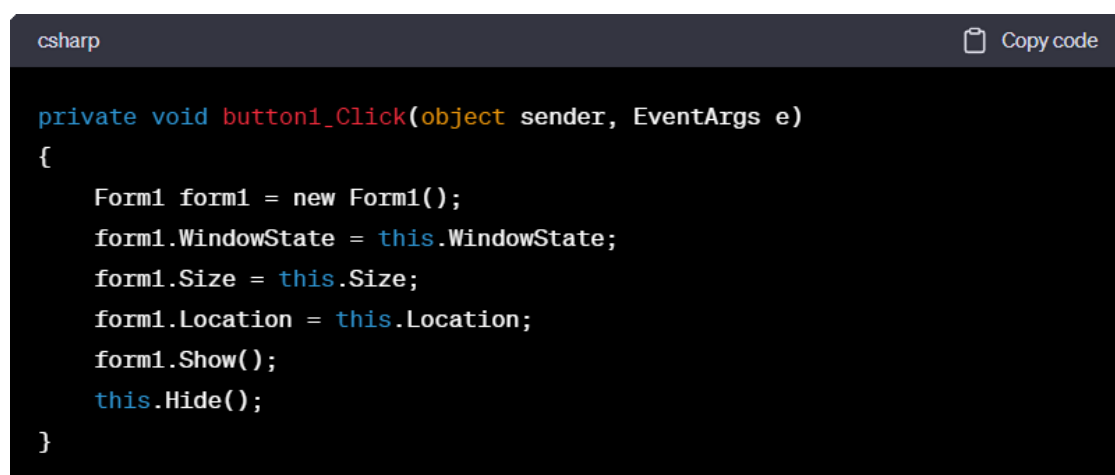
Ezenkívül még egy átlagfelhasználó, de informatikai eszközök felhasználásában otthonosan mozgó ismerősöm segítségét kértem a program működésének ellenőrzéséhez.

Használat előtt nem olvasta el a felhasználói dokumentációt, ennek ellenére jól áttekinthetőnek és könnyen használhatónak ítélte a programot. Ezt azért tartom fontosnak kiemelni, mert ez fontos célom volt a készítés folyamán. Továbbá pozitívként említette, hogy a keresések enter leütésekor is lefutnak.

Ezenfelül ő vette észre azt a problémát, hogy miután a *MainForm* ablakot teljes képernyős módba tette, majd rákattintott a *btnBooks* gombra, a *FormBooks* nem vette fel az előző ablak méretét, hanem továbbra is normál módban nyílt meg.

Mivel ebben az időszakban kezdtem el kísérletezni az *OpenAI* által fejlesztett *ChatGPT* használatával, úgy döntöttem megpróbálom ennek segítségével megoldani ezt a feladatot.

Miután sikerült egyértelműen megfogalmaznom a kérdésemet, a chatbot a 8. ábrán látható kódrészletet javasolta, amit kisebb változtatásokkal fel is tudtam használni.

A screenshot of a code editor window with a dark theme. The title bar at the top left says 'csharp' and the top right has a 'Copy code' button. The code is a C# method named 'button1_Click' that takes 'object sender' and 'EventArgs e' as parameters. Inside the method, a new 'Form1' object is created, and its 'WindowState', 'Size', and 'Location' are set to match the current window ('this'). Then, 'form1.Show()' is called, and 'this.Hide()' is called. The method is enclosed in curly braces.

```
csharp Copy code

private void button1_Click(object sender, EventArgs e)
{
    Form1 form1 = new Form1();
    form1.WindowState = this.WindowState;
    form1.Size = this.Size;
    form1.Location = this.Location;
    form1.Show();
    this.Hide();
}
```

9. ábra: A ChatGPT javaslata az ablakok tulajdonságainak beállítására

3. Felhasználói dokumentáció

A következőkben a házi könyvtár webalkalmazáshoz tartozó asztali alkalmazásról fog olvasni, melyen keresztül kezelheti az oldal felhasználóit, és a tárolt könyveket is elérheti.

3.1. Az alkalmazás rendszerkövetelménye

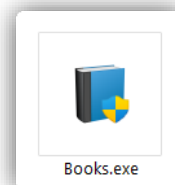
A program működéséhez a *.NET Framework 4.7.2* telepítésére van szükség, így a hardver- és szoftverigényt ez határozza meg. Ez a *microsoft* hivatalos oldalán ellenőrizhető.

Minden olyan eszköz, ami alkalmas ezen keretrendszer futtatására, az alkalmas a *Books* asztali alkalmazás futtatására is.

3.2. Az alkalmazás telepítése

A program használatához telepítésre nincs szükség. Kattintson duplán a *Books.exe* fájlra, és az alkalmazás máris elindul.

A program minden funkciója elérhető regisztráció nélkül.

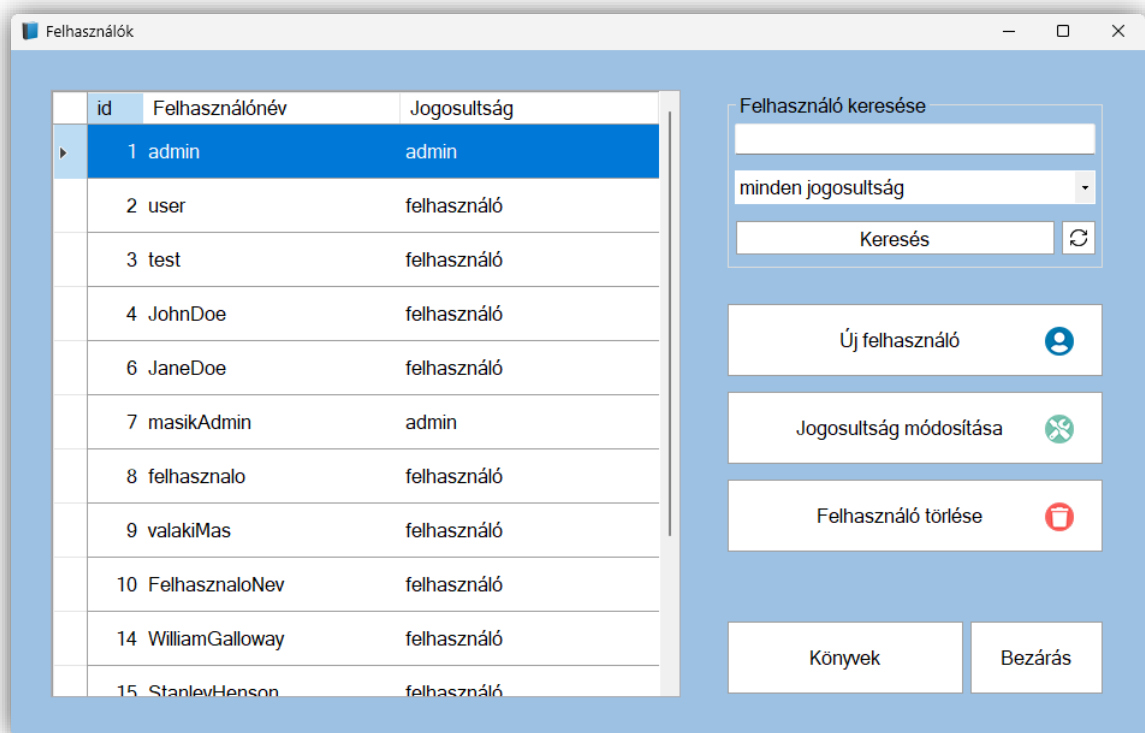


10. ábra:
Futtatható fájl

3.3. Az alkalmazás használata

3.3.1. Felhasználók

A program indítása után a nyitóablakban azonnal megjelennek az adatbázisban szereplő felhasználók, a hozzáférési jogosultságukkal együtt.



11. ábra: Nyitóablak: felhasználók kezelése

Táblázat

Ha abc sorrendbe szeretné rendezni a neveket, elég a „Felhasználónév” fejlécre kattintania. Ugyanez vonatkozik a többi oszlopra is. Az első kattintás növekvő sorrendbe rendez, a második csökkenőbe.

A fejléc szélének megragadásával tetszőlegesen átméretezheti az oszlopok szélességét, kivéve az „id” oszlopot.

Ha ki akar jelölni egy felhasználót, kattintson az azonosítójára, a nevére vagy a jogosultságára, vagy akár az adott sor fejlécére, a program minden esetben kijelöli az egész sort.

Keresés

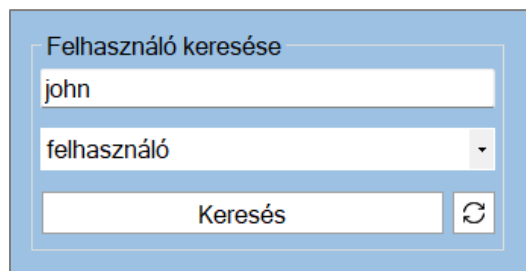
Ahhoz, hogy megtalálja a keresett felhasználót, be kell írnia a nevét – vagy a nevének egy részletét – az első beviteli mezőbe. Kattintson a „Keresés” gombra vagy üssön enter, hogy elvégezze a műveletet.

Az alatta lévő legördülő lista segítségével jogosultság alapján is szűrhet: kiválaszthatja, hogy az adminok vagy csak az egyszerű felhasználók jelenjenek meg. Itt is kereshet enterrel, vagy a gombra kattintva.

A keresés figyelembe veszi mindkét feltételt (akár enter leütésével, akár a gombra kattintva keres). Ha a felhasználónévhez nem ír be semmit, akkor csak jogosultság alapján szűr.

Ha szeretné ismét megjeleníteni az összes felhasználót, kattintson a frissítés ikonnal jelzett gombra!

12. ábra: Keresés felhasználók között

The image shows a web interface for searching users. It has a light blue border. At the top, it says "Felhasználó keresése". Below this is a text input field containing the word "john". Underneath the input field is a dropdown menu with the text "felhasználó" and a small downward arrow. At the bottom of the form, there is a button labeled "Keresés" and a circular refresh icon to its right.

Felhasználók kezelése

A felhasználók karbantartására a következő három gomb szolgál:

- Új felhasználó
- Jogosultság módosítása
- Felhasználó törlése

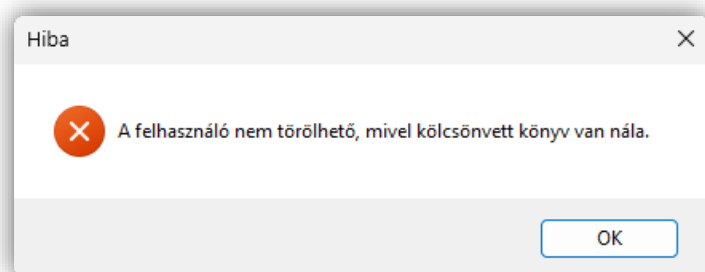
Az első kettő új ablakot nyit meg, amelyekről a későbbiekben lesz szó.

Ahhoz, hogy töröljön egy felhasználót, csak ki kell jelölnie, majd a „Felhasználó törlése” gombra kattintania. Az alkalmazás megerősítést kér, hogy biztosan hajtsa-e a végre a műveletet.

Ha az „Igen” gombra kattint, a felhasználó véglegesen törlődik az adatbázisból. Visszavonásra nincs lehetőség.

Nem lehet olyan felhasználót törölni, akinél kölcsönvett könyv van. Ha véletlenül mégis ilyen felhasználót próbálna törölni, a program egy hibaüzenetben tájékoztatja a művelet sikertelenségéről.

13. ábra: Hibaüzenet: kölcsönzés miatt nem törölhető



Ezeket túl a „Könyvek” gomb átnavigálja az adatbázisban található könyvek kezelésére alkalmas ablakba, a „Bezárás” gomb pedig értelemszerűen leállítja az alkalmazást.

3.3.2. Új felhasználó hozzáadása

Az „Új felhasználó” gombra kattintva megjelenik egy új ablak, ahol ki kell töltenie minden mezőt.

Ahhoz, hogy sikeresen hozzáadja a felhasználót, a névnek és a jelszónak meg kell felelnie bizonyos feltételeknek.

Olyan felhasználónevet kell megadnia, ami még nem szerepel az adatbázisban, legalább 3 karakterből áll, és csak a következőket tartalmazza:

- az angol abc kis- és nagybetűi,
- számok,
- pont,
- kötőjel
- és aláhúzásjel.

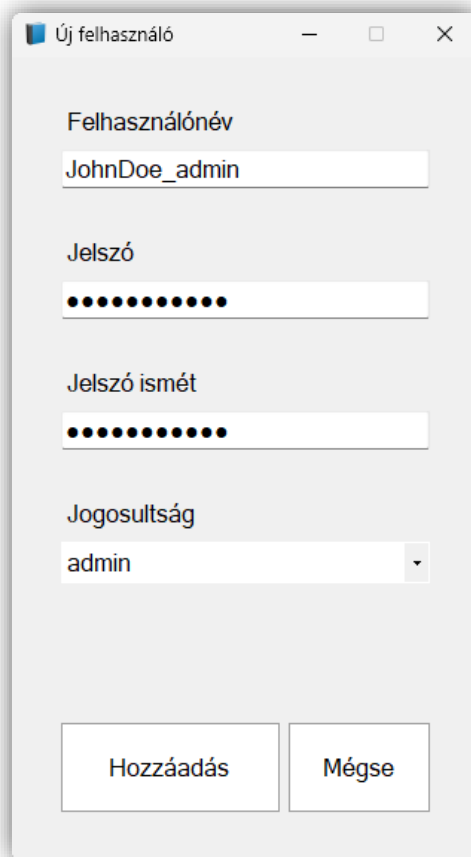
A jelszóra vonatkozó korlátozások:

- legalább 8 karakter hosszú legyen,
- tartalmazzon legalább egy kis- és nagybetűt,
- legalább egy számot
- és legalább egy speciális karaktert.
- Ezenkívül a beírt jelszavaknak meg kell egyezniük.

Ha a megadott értékek valamelyik feltételnek nem felelnek meg, arról a program hibaüzenetet küld.

A webalkalmazáson keresztül történő regisztrációtól eltérően itt kiválaszthatja, hogy egyszerű felhasználót vagy admint kíván felvenni.

A „Hozzáadás” gomb felveszi a megadott felhasználót az adatbázisba, a „Mégse” gomb lenyomásával pedig elvetheti a beírt adatokat, és kiléphet az ablakból.



14. ábra: Új felhasználó adatainak kitöltése

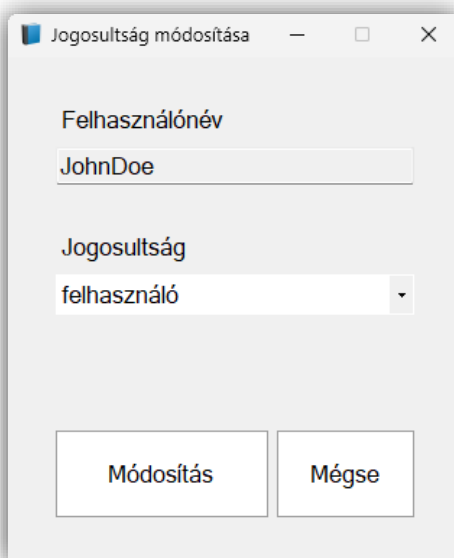
3.3.3. Jogosultság módosítása

Ebben az ablakban egy már létező felhasználó hozzáférési jogosultságát változtathatja meg.

Alapértelmezetten az az érték jelenik meg, amivel a felhasználó a kiválasztáskor rendelkezett. Ha ezt módosítani szeretné, választhat a legördülő listából, illetve a billentyűzeten lévő Le-Fel vagy Jobbra-Balra nyilak segítségével is kijelölheti a kívánt jogosultságot.

Ha változtatás nélkül kattint a „Módosítás” gombra, figyelmeztetést kap, miszerint „Nem történt módosítás”.

Ha változtatás után kattint, akkor az új jogosultság rögzítésre kerül.



15. ábra: Hozzáférési jogosultság módosítása

3.3.4. Könyvek

Ha a nyitóablakban a „Könyvek” gombra kattint, egy hozzá nagyon hasonló ablak jelenik meg, ahol az adatbázisban található könyveket kezelheti.

16. ábra: Könyvek kezelésére szolgáló ablak

id	Cím	Szerző	Típus	Befejezett
1	Árva korunkb...	Kazuo Ishiguro	nyomtatott	nem
2	Az eltemetett...	Kazuo Ishiguro	nyomtatott	nem
3	Az eltemetett...	Kazuo Ishiguro	nyomtatott	igen
4	1q84 - első k...	Murakami Ha...	nyomtatott	nem
5	1q84 - másod...	Murakami Ha...	nyomtatott	nem
6	1q84 - harma...	Murakami Ha...	nyomtatott	nem
7	Édes a bossz...	Jonas Jonas...	nyomtatott	igen
8	Az analfabét...	Jonas Jonas...	nyomtatott	igen
9	A százéves e...	Jonas Jonas...	nyomtatott	igen
10	A Szilmarilok	J. R. R. Tolkien	nyomtatott	nem
11	Tűz és vér	Goerge R. R....	nyomtatott	nem
12	Az utazó mac...	Hiro Arikawa	ebook	nem
13	Az eltemetett...	Kazuo Ishiguro	ebook	nem
14	Kékszakáll	Kurt Vonnegut	nyomtatott	igen
19	Cujo	Stephen King	nyomtatott	igen

Könyv keresése

Cím

Szerző

Típus
mind

Befejezett
mind

Keresés

Adatok módosítása

Könyv törlése

Felhasználók Bezárás

Táblázat

A táblázat ugyanúgy működik, mint az, amelyik a felhasználókat tartalmazza, mindössze annyi a különbség, hogy itt több oszlop található, mivel a könyvekről több adat van eltárolva:

- az azonosító,
- a cím,
- a szerző,
- a típus, amiből az derül ki, hogy a könyv nyomtatott vagy e-book formátumú,
- és a befejezettség, ami azt jelzi, hogy olvasta-e már az adott könyvet.

Ezekon kívül minden könyvhöz tartozik egy kép, de az jelenleg nem elérhető az asztali alkalmazásból.

Könyvek kezelése

A keresés ugyanúgy működik, mint a felhasználóknál, csak több feltétel adható meg.

Új könyvet az asztali alkalmazásból jelenleg nem lehet hozzáadni.

Az „Adatok módosítása” gombra kattintva új ablak nyílik meg, ahol megváltoztathatja a kiválasztott könyv adatait.

A „Könyv törlése” gombbal eltávolíthatja a kijelölt könyvet, akárcsak a felhasználók esetében.

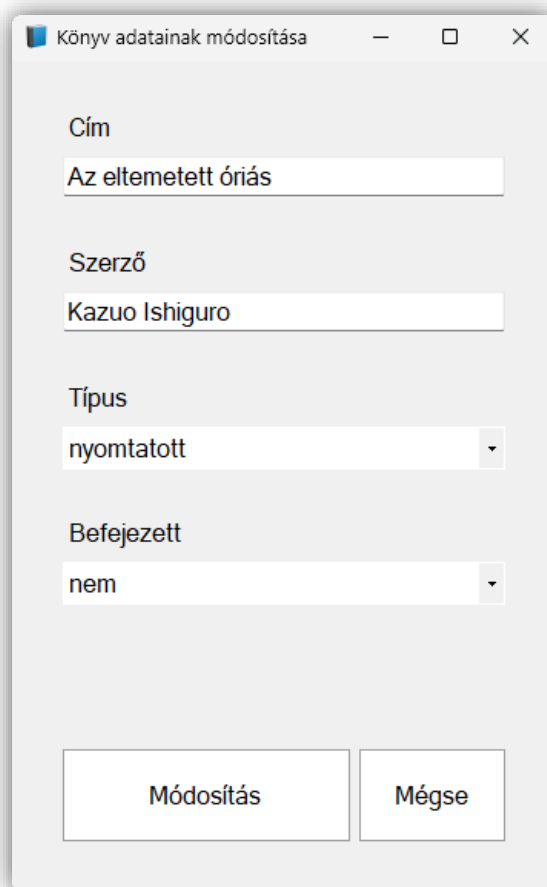
A „Felhasználók” gomb visszaviszi a felhasználókat kezelő nyitóoldalra, a „Bezárás” gomb pedig leállítja az alkalmazást.

3.3.5. Könyv adatainak módosítása

Ebben az ablakban módosíthatja annak a könyvnek az adatait, amit kiválasztott. Ahhoz, hogy elmenthesse az új értékeket, minden mezőt ki kell töltenie. Más megkötés nincs.

Amennyiben anélkül kattint a „Módosítás” gombra, hogy bármit változtatott volna, az alkalmazás jelzi, hogy nem történt módosítás.

A „Mégse” gombra kattintva itt is elvetheti a beállításokat.



Könyv adatainak módosítása

Cím
Az eltemetett óriás

Szerző
Kazuo Ishiguro

Típus
nyomtatott

Befejezett
nem

Módosítás Mégse

17. ábra: A módosítani kívánt könyv adatai

4. Fejlesztési lehetőségek

A program az első tervek alapján mindössze egy személy könyveinek rendszerezését tette volna lehetővé, ehhez képest már a készítés közben is további funkciókkal bővült. Ebből is látható, hogy az alapötlet még nagyszámú további lehetőséget rejt magában.

Az egyik legfontosabb, amivel ki szeretném egészíteni az asztali alkalmazást, hogy a kölcsönzések kezelhetőek legyenek innen is, ne csak a webes felületről.

Ezzel együtt a könyveket és a felhasználókat megjelenítő táblázatok kiegészülnének egy oszloppal, ami jelezné, hogy a könyv kölcsönben van-e, illetve a felhasználónál van-e kölcsönvett könyv.

A könyveket kezelő funkciók közé fel tervezem venni, hogy új könyvet is hozzá lehessen adni, és hogy a könyvhöz tartozó képet is lehessen módosítani.

Továbbá, ha a webalkalmazás bővül, úgy az ott megjelenő újításokat is követnie kell az asztali alkalmazásnak. Ilyen lehetséges újítások a következők:

- kedvenc könyvek jelölése,
- könyvek értékelése, véleményezése,
- fórum kialakítása,
- további szűrési feltételek felvétele (pl. kölcsönvett könyvek),
- adminisztrátori oldalon többféle visszajelzés a felhasználók tevékenységeiről,
- regisztráció e-mail címmel,
- és elfelejtett jelszó kezelése.

Amennyiben ezek – illetve egyéb, a későbbiekben felmerülő funkciók – megvalósulnak, a hozzájuk tartozó adminisztrátori feladatoknak hozzáférhetőnek kell lenniük az asztali alkalmazáson keresztül.

5. Összegzés

Az asztali alkalmazás az első tervek szerint csak a felhasználók kezelésére biztosított volna lehetőséget. Az ekkor meghatározott elvárásokat a kész program teljesíti.

A későbbiekben merült fel, hogy a könyvek is hozzáférhetők legyenek innen is, így lehetőségem nyílt megtapasztalni, hogy milyen egy majdnem-kész program kibővítése, amiből sokat tanultam.

A program tervezéséhez a *Figma* nevű webalkalmazást használtam, amit korábban nem ismertem. Biztos vagyok benne, hogy ennek a szoftvernek ezután is nagy hasznát veszem majd, bármilyen felületet kell megterveznem, legyen az akár webre, mobil eszközre vagy asztali felhasználásra szánt alkalmazás.

A konkrét *Windows Forms* applikáció elkészítése közben nagyobb jártasságra tettem szert a *C#* programozási nyelvben, azon belül is elsősorban az adatbáziskezelést érintő feladatok megvalósításában. Megtapasztaltam az objektumorientált programozás előnyeit, többek közt, hogy a programkód kisebb részekre bontása, osztályokba és metódusokba való szervezése mennyivel áttekinthetőbbé és egyszerűbben karbantarthatóvá teszi azt.

Mivel a felhasználók és a könyvek adatait is egy-egy *DataGridView* táblázatban jelenítettem meg, ennek sok tulajdonságával megismerkedtem. Természetesen a számtalan beállítás közül közel sem minddel, de így is sok érdekes dolgot kipróbáltam, mire elértem a kívánt működést és megjelenést. A megismert tulajdonságok közül több nem került be a végleges programba, mert nem járultak hozzá annak áttekinthetőségéhez.

Tanulságos volt továbbá az új felhasználó felvételekor szükségessé vált jelszótitkosítás megvalósítása. Az első ötletem szerint az asztali alkalmazásban megadott felhasználói adatokat a webalkalmazásban meghatározott regisztrációs végpontnak küldtem volna el. Ennek kivitelezése azonban feleslegesen komplikáltnak tűnt, így más megoldás után néztem. A végleges verzióban a titkosítást két egyszerű parancs végzi el a *BCrypt.Net-Next NuGet package* segítségével, aminek eredményeképp sokkal áttekinthetőbb, tisztább a kód.

Összességében úgy vélem, ezen vizsgaremek elkészítése nagyban hozzájárult a szakmai fejlődésemhez.

6. Hivatkozásjegyzék

- <https://csharptutorial.hu/>
- <https://www.c-sharpcorner.com/>
- <https://learn.microsoft.com/hu-hu/>
- <https://dev.mysql.com/>
- <https://code-maze.com/>
- <https://stackoverflow.com/>
- <https://openai.com/blog/chatgpt/>
- <https://puvox.software/blog/creating-portable-exe-or-self/>
- <http://www.angusj.com/resourcehacker/>
- Jánosi-Rancz Katalin Tünde: Adatbázisok I.9

Az alkalmazásban felhasznált ikonok a következő oldalakról származnak:

- <https://iconduck.com/>
- <https://www.iconarchive.com/>
- <https://twitter.com/doublejdesign/>

7. Ábrajegyzék

1. ábra: A felhasználókat kezelő felület terve	5
2. ábra: A könyveket kezelő felület terve	6
3. ábra: Sikertelen csatlakozást jelző hibaüzenet	11
4. ábra: Felhasználó vagy könyv törléséhez tartozó kivételkezelés	14
5. ábra: Az <i>SFX packager</i> felülete.....	24
6. ábra: Az <i>ex</i> objektum részletei	25
7. ábra: Sikertelen törléskor megjelenő alapértelmezett és egyedi hibaüzenet.....	26
8. ábra: Lezáratlan kapcsolat esetén jelentkező kivétel	26
9. ábra: A <i>ChatGPT</i> javaslata az ablakok tulajdonságainak beállítására	27
10. ábra: Futtatható fájl.....	28
11. ábra: Nyitóablak: felhasználók kezelése	29
12. ábra: Keresés felhasználók között	30
13. ábra: Hibaüzenet: kölcsönzés miatt nem törölhető	31
14. ábra: Új felhasználó adatainak kitöltése	32
15. ábra: Hozzáférési jogosultság módosítása	33
16. ábra: Könyvek kezelésére szolgáló ablak	34
17. ábra: A módosítani kívánt könyv adatai	36