ZÁRÓDOLGOZAT

Gurúz Tamás

2024

Magyar Agrár- és Élettudományi Egyetem

Károly Róbert Campus

Műszaki Intézet

Programtervező informatikus felsőoktatási szakképzési szak

Elavult szoftver (újra)tervezése konkrét példán keresztül

Belső konzulens:	Sike Zoltán Mesteroktató
Belső konzulens intézete/tanszéke:	Műszaki Intézet / Alkalmazott Informatika Tanszék
Készítette:	**Gurúz Tamás** **AR3C8A**

Gyöngyös

2024

TARTALOMJEGYZÉK

1. BEVEZETÉS

Dolgozatomban egy elavult szoftver újra tervezésének első lépéseit szeretném bemutatni. Két okból esett erre a témára a választásom: egyrészt a szakmai gyakorlatom egy részét ez a feladat töltötte ki, másrészt rengeteg kihívással és megoldandó problémával jár egy kezdő fejlesztőnek egy kerek program elkészítése, sok hasznos tanulási lehetőséget hordozva magában.

Egy szoftver elkészítése több lépésből áll: igényfelmérés, tervezés, fejlesztés, tesztelés, hibajavítás, kiadás, karbantartás. Ebben az esetben ez a folyamat sokkal több időt fog igénybe venni, mint a dolgozat és tanulmányaim többi határideje megenged. Így most csak az első lépések megvalósítását mutatom be:

1. A szoftverrel szemben támasztott igények meghatározása. Ez a lépés magában foglalja a következőket: azon folyamatok megértését, melyek a program használatával kapcsolatosak, felmérni a felhasználói igényeket, átlátni, hogyan használják majd a felhasználók a programot.
2. A szoftver tulajdonságainak megtervezése, vagyis architektúra meghatározása, a fejlesztéshez használt eszközök, technológiák kiválasztása.
3. A szoftver főbb elemeinek megtervezése, úgymint a felhasználói felület, felhasználóval való kommunikációs felületek és módok megtervezése, adatbázisok megtervezése, a program működési elveinek logikájának lefektetése. (Ez utóbbi nem azonos az alkalmazáslogika kidolgozásával.)

A fenti lépések bár nem olyan látványosak és izgalmasak, mint a kódolás, mégis nagyon lényegesek a későbbi siker szempontjából (Rierson, 2013). Amennyiben befektetünk a tervezés fázisába, később sokszorosan megtérül és sok felesleges munkától kíméli meg a fejlesztőt.

A témául szolgáló szoftver a gyöngyösi Városgondozási Zrt. részére készül, és kiváltja a jelenleg használatban lévő programot. Ez a program egy adatbáziskezelő szoftver, amelyhez egy, a feladat ellátásához szükséges adatbázist építettek a korábbi dolgozók.

2. A VÁROSGONDOZÁSI ZRT. BEMUTATÁSA

Fennállása óta a Városgondozási Zrt. több formában működött. Jelenlegi gazdálkodási formáját 1991-ben vette fel, azóta részvénytársaságként működik. A tulajdonosi jogokat Gyöngyös Város Önkormányzata gyakorolja.

A Zrt. egy komplex, több területen működő cég, melynek célja Gyöngyös város szebbé-jobbá tétele, bizonyos szolgáltatásainak fenntartása, működtetése. E tevékenységet az alábbi ágazatok végzik:

- Távfűtés
- Parkolás
- Temetkezés
- Vagyongazdálkodás
- Építőipar
- Köztisztaság
- Ebrendészet

Az ágazatok önálló működéssel és költségvetéssel bírnak, de gazdálkodásuk erősen kötött és felügyelt. Az informatikai fejlesztések éppen ezért hiányosak, több területen nem megfelelően szolgálják ki a felhasználókat.

Az ágazatok közötti koordinációt, illetve a gazdasági döntések nagy részét a központi irányítás végzi, ez a szerv van közvetlen kapcsolatban a fenntartóval. A központi irányításon belül létezik informatikai részleg, azonban jellemzően a használt célszoftverek kiválasztásában, véleményezésében nem vesz részt. Az ilyen jellegű szoftvereket jellemzően a dolgozók kutatják fel.

A dolgozatban ismertetett program a Vagyongazdálkodási Osztály munkáját hivatott segíteni. Ez az Osztály kezeli a cég tulajdonában lévő ingatlanokat, nyomon követi az ingatlanokkal kapcsolatos eseményeket és nyilvántartja a költségeket. Fontos részlet, hogy ez a költségnyilvántartás más szemléletű, mint a Könyvelési Osztály nyilvántartása. Ez az oka annak, hogy a költségkezelés nem a központi számlázó programban történik.

3. A SZOFTVERFEJLESZTÉSI IGÉNY OKAI

3.1. Az információgyűjtés módjai

Egy szoftver tervezésekor elengedetlen a felhasználók által végzett tevékenységek megismerése. Az igények felismeréséhez nem elég csupán elméleti ismeretket szerzünk az adott területről, hanem törekedünk kell a felhasználók fejével gondolkodni és a számukra előnyös és kényelmes megoldásokat megtalálnunk (Tömösközi, 2013).

Ahogy a bevezetőben említett szoftver, melyet újra szeretnénk tervezni, a céges környezetben teljesen izoláltan működött: nem kapcsolódott hálózatra, a bementi és kimeneti adatok papíralapúak voltak és csak egy dolgozó használta. A megismeréshez vele kellett többször interjút készítenem, illetve készségen megmutatta, hogyan használja a programot.

Az interjúk során a munkatárs többször elmondta, hogy „nem számítógépes guru”. A használatot az elődöktől örökölt papír alapú utasításgyűjteménnyel valósították meg. Ez leginkább egy lépésről-lépésre típusú leírás volt, amely nem tartalmazott leírásokat, magyarázatokat. A programban rejlő eredeti lehetőségeket és logikát már senki sem érti a cégnél.

A teljes leképezésen túl szerettem volna minőségi fejlesztést végrehajtani. Ennek érdekében nem csak a közvetlen felhasználóval egyeztettem, hanem azokkal a munkatársakkal is, akik a program által feldolgozott adatokat használják vagy használhatják. Így derült fény olyan funkciókra, melyek a többiek munkáját is nagyban megkönnyíthetik, és ennek révén a szoftver szervesebben tud kapcsolódni a mindennapi feladatokhoz.

A személyes interjúk mellett betekintést nyerhettem a dolgozók által használt Excel táblákban. A teljes kép átlátásához, a szervezeti felépítés és a működési alapelvek megértéséhez a cég Működési és Üzletszabályzata nyújtott segítséget.

3.2. A jelenlegi informatikai környezet bemutatása

A cég a LIBRA felhőalapú vállalatirányítási rendszert használja. Bár a szoftver folyamatosan fejlesztés alatt áll, új funkciókkal gazdagodik, gazdasági, illetve menedzsmenti döntésekből kifolyólag, a rendszer egy régi változata áll rendelkezésre. Ebben nem szerepelnek olyan modulok, melyek a különböző ágazatok speciális igényeit hivatottak kielégíteni.

Maga a Libra Szoftver Zrt. egy magyar hátterű vállalkozás, melynek fő terméke a LIBRA vállalatirányítási rendszer. Kategóriájában nem kiemelkedő piaci pozíciót tudhat

magáénak, de a teljes értékű programcsomag kielégíti a hasonló rendszerekkel szemben támasztott alapkövetelményeket. A rendszer modulárisan felépíthető és széleskörűen paraméterezhető. Használatával hatékonyabb számlakezelés, folyamatmenedzsment, anyaggazdálkodás, általános ügyvitel valósítható meg.

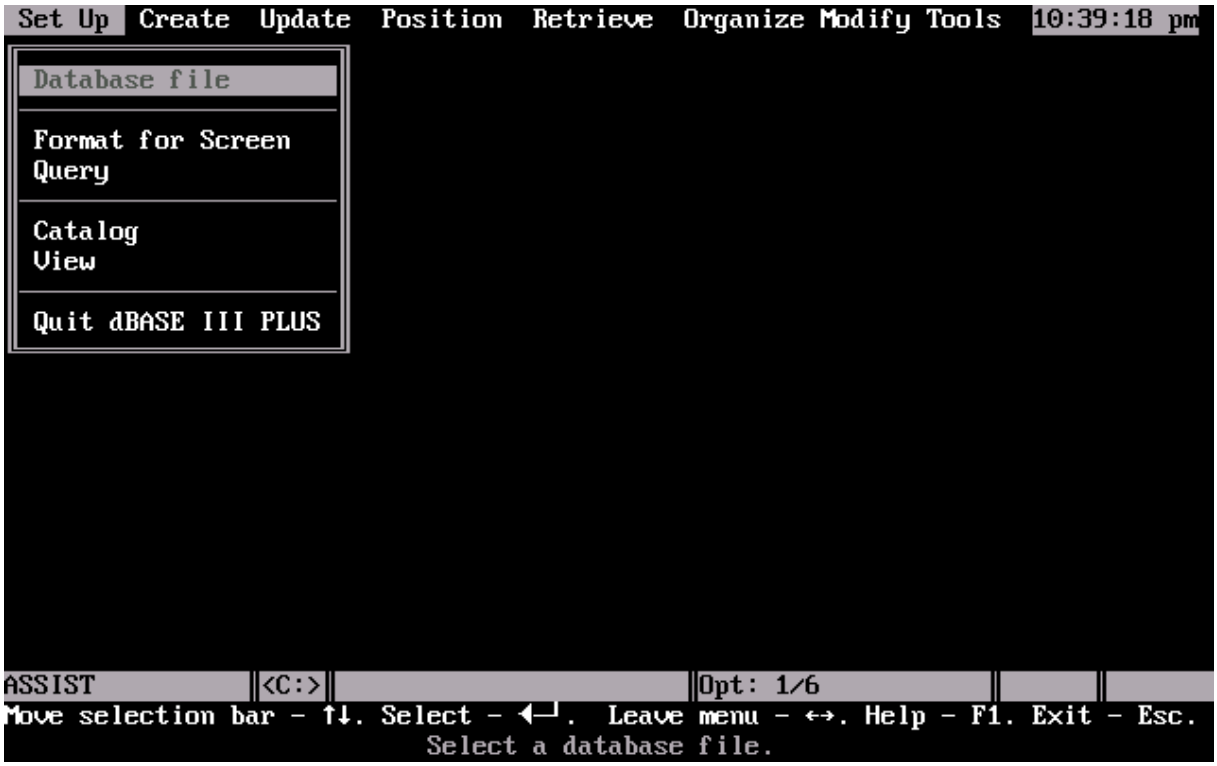
A Városgondozási Zrt-nél a ERP funkciója a számlázás, legfőképpen a Könyvelési Osztály munkáját segíti, de jellegéből adódóan minden ágazat érintett a használatában. Bár a szoftvercsomag számos modult kínál, az általam megismert területeken (távhő, vagyongazdálkodás) csak a számlázási funkciót használják a dolgozók.

A napi feladatok ellátásában nagymértékben támaszkodnak a Microsoft Office programcsomag alkalmazásaira, elsősorban a Word-re és Excel-re. Word-öt a hivatalos levelezéshez és az általános szövegszerkesztési feladatokhoz használják. Egy gyakran használt hatékonyságnövelő, munkagyorsító funkció a körlevél készítés. A munkavégzés javát Excel táblákban végzik, akár több százas nagyságrendű is lehet egy-egy ágazat tábláinak száma. Sok a korábbi dolgozóktól örökölt táblázat és megfigyelhető az egyes táblázatok összekapcsolásának, bővítésének, illetve az olyan Excel-funkciók kihasználásának hiánya, mint a képletek használata (az alapműveletek kivételével), vagy a hivatkozások.

A céges szoftverkörnyezetből kilóg a projekt előzmény-szoftvere, a dBase III. Plus, amely próbált többet nyújtani az Excel-Word kombináció lehetőségeinél. Bevezetésekor népszerű és hatékony megoldás volt adatbázisok kezelésére, mára azonban a termék elavultnak számít.

3.3. A jelenlegi alkalmazás bemutatása

A dBase III. Plus egy MS-DOS alapú adatbáziskezelő program, melyet az Ashton-Tate nevű garázscég készített (Shukla, 1995). Ez a szoftver lehetővé teszi az adatbázisok létrehozását, szerkesztését, lekérdezését és kezelését egy könnyen kezelhető felhasználói felületen keresztül. Ennek megvalósításához egy strukturált lekérdező nyelvet (SQL) használ. A programcsalád hármas verzióját 1986-ban adták ki, majd készült még egy negyedik kiadás. Ezután az újabb technológiák megjelenése és a negyedik kiadás hibái miatt a program hanyatlani kezdett, majd megszűnt. Az azonban érdemes megjegyezni, hogy a dBase az 1980-as években az adatbázis-kezelők piacának 70%-át uralta, 150 000 eladott példánnyal, ami abban az időben hatalmas sikert és népszerűséget jelentett.



1. ábra: A dBase program felülete telepítés után

A DOS operációs rendszerre írt programokra jellemzően a dBase sem rendelkezik grafikus felhasználói felülettel. A kor elvárásainak megfelelően a dBase úgynevezett TUI-t (Text-Based User Interface) használ a parancssoros vezérlés helyett (1. ábra). A TUI egy semigrafikus felhasználó felület, mely főként szöveges alapú, de lehetnek benne grafikai elemek is (Chen és tsa., 1998). A TUI általában speciális karakterek segítségével, illetve azok tulajdonságainak megváltoztatásával ér el grafikai hatásokat.

A dolgozat készítésekor egy Windows XP-ről indítva használták DOS emulátorral. Az indítás a dbase.exe állomány segítségével történt, a mai szemnek szokatlanul parancssorból. Több navigációs utasítás után a *dbase start* utasítással indul el a program. Ezután különféle scriptek meghívásával lehetett a funkciókat használni. Például a főmenü a *do fomenu* utasítással érhető el.

A főmenü (és az almenük is) szöveges listákkal operál, a menüelemeket a listaelem sorszámának megadásával, majd az Enter gomb leütésével lehet kiválasztani. Többszintű menürendszer segít elérni a beviteli és a lekérdező felületeket. Ezeken belül a TUI-ra jellemzően a nyílbillenyűkkel, illetve a Tab és Backspace billentyűkkel lehet navigálni a menüpontok vagy utasítások között.

A program a kimeneti adatokat kétféle formában közölheti. Egyrészt a menüben találhatunk lekérdező pontokat, melyek egy szöveges listát generálnak, esetleg összegsorok segítségével további összegző, átlagoló vagy megszámoló jellegű adatot közölnek. Másrészt lehetőség van a listák kinyomtatására is. Ennek jelentősége van, ugyanis a mindennapi munkavégzés folyamán rendszeresen ilyen nyomtatott riportok formájában kerülnek továbbításra az adatok más osztályokra, például a könyvelésre, ahol manuálisan újra fel kell vinni az összes adatot a rendszerbe.

Összegzésképpen elmondható, hogy a dBase III Plus az alábbi okok miatt nehézkesen integrálható a jelenlegi szoftver-, hardver- és munkakörnyezetbe:

- Hardver és operációsrendszer: A program nem kompatibilis a modern operációs rendszerekkel, így az eredetileg használt hardverkörnyezetet továbbra is fent kell tartani. Ez a gyakorlatban azt jelenti, hogy két pc működik egymás mellett, a mindennapi munka során ki-be kapcsolgatják a dolgozók a gépeket, hogy a használni kívánt programokat működtetni tudják.
- Általános használat: Már a program megnyitása is perceket igénylő „procedúra". A kívánt funkciók elérése többlépcsős összetett egyirányú folyamat, számos idegőrlő hibalehetősséggel. A funkciók váltása között a teljes menürendszeren végig kell lépkedni.
- Adatbevitel: Az adatbevitel során ismétlődő adatokat kell újra és újra megadni, mely szintén időigényes és sok hibalehetőséget hordoz magában.
- Kimeneti adatok: Alacsony hatékonysággal használhatók, mert csak papíralapon készülnek, nincs exportálható állomány.

A fenti listából kitűnik az igény egy újabb, modernebb programra, mely teljesíti a 21. századi szoftverekkel szembeni elvárásokat és hatékonyan használható a feladatra.

3.4. Célok meghatározása

A legfőbb igény a dBase III Plus leváltása, illetve helyettesítése egy olyan szoftverrel, amely a mai kornak megfelelően grafikus felhasználói felülettel rendelkezik, kompatibilis a jelenlegi céges operációs rendszerekkel és hatékonyan használható a napi feladatok elvégzésére. Képes adatbázisok kezelésére, azokon műveletek elvégzésére, kimutatások, riportok készítésére.

Általános igény a gyors, hibamentes működés és az egyszerű, logikus, könnyen tanulható felhasználói felület.

A felhasználókkal folytatott interjúk és a munkafolyamatok megismerése után megállapítottam, hogy a kezelendő feladatkör két részre bontható: 1. a fogyasztás-számlázás nyomon követése, 2. költségfelosztás.

A bemeneti adatok a cég által leolvasott óraállások és a kapott közműszámlák adatai. Ezekből az adatokból a nyomon követés során az alábbi információkat kell előállítani tetszőleges épületekre és időszakra: tényfogyasztás és számlázott mennyiségek összevetése, azaz minden tétel ki lett e számlázva, átalány esetébe mennyi az adott pillanatban az alul vagy túlfizetés mértéke; fogyasztások alakulása hosszabb időszakot figyelembe véve; szolgáltatások összköltségeinek meghatározása (például közös költség kalkuláláshoz); kimutatások készítése (például éves elszámoláshoz).

Egy másik nagy terület a költségfelosztás. A vagyongazdálkodás által kezelt épületek sokféle rendeltetésűek és tulajdonviszonyúak. Ezen épületek költségeinek kezelése, megfelelő helyre történő könyvelése, illetve tovább számlázása a könyvelés feladata. Az alapja ennek a tevékenységnek egy riport, céges szóhasználatban „feladás”. A feladás tulajdonképpen egy meghatározott paraméterek szerinti lekérdezés-sorozat, mely különféle szempontok szerint csoportosítja az adott időszak mennyiségeit és költségeit. Ez kerül továbbításra a könyvelésre, akik tovább dolgoznak a kapott adatokkal. Jelenleg ez papír alapon működik. Egy konkrét cél a fejlesztés során, hogy az összes lekérdezés exportálható legyen *xlsx* vagy *csv* formátumban esetleg finomítani az exportálást, hogy a LIBRA által értelmezhető struktúrába és formátumba kerüljenek az adatok.

A szoftver használatával kapcsolatos tevékenységek a napi munkavégzéshez tartoznak, így nagyon fontos, hogy minél gördülékenyebb, könnyedebb legyen a használta. Ezt azáltal lehet elérni, hogy egyértelmű és könnyen használható, jól olvasható felületet alakítunk, és a működés során törekszünk a szükséges kattintások és gombnyomások számát csökkenteni.

4. FELHASZNÁLT ESZKÖZÖK ÉS TECHNOLÓGIÁK

4.1 .NET

A .NET platform egy szoftverfejlesztési platform és keretrendszer, amelyet a Microsoft fejlesztett ki a kilencvenes évek végén. Ekkor indult el a Microsoft berkeiben a Next Generation Windows Sevices fedőnevű projekt, melyből a .NET született. A platform fejlesztése válasz volt a Sun MicroSystems által kiadott Java platform térhódítására. A fejlesztői piac visszanyerése mellett egy cél volt a COM platform leváltása is. A .NET Framework az évek során nagy népszerűségre tett szert, jelen van az asztali alkalmazásokban, a weben és az okostelefonokban is (Reiter, 2012). Maga a keretrendszer a CLI (Common Language Infrastructure) egy imlpementációja. A CLI egy szabályrendszer, mely három fő részből áll:

1. CTS (Common Type System): Közös típusrendszer. Egy olyan specifikáció, amely meghatározza azokat a típusokat és azok viselkedését, amelyeket a .NET keretrendszerben használni lehet.
2. CLS (Common Language Specification): Közös nyelvi specifikáció. Meghatározza a .NET-kompatibilis nyelvek közös funkcióit és szerkezetét.
3. CLR (Common Language Runtime): A futtatási környezet specifikációja.

A .NET platform számos funkciót és eszközt biztosít a fejlesztőknek az alkalmazásaik gyors és hatékony létrehozásához és kezeléséhez. Az alaposztály könyvtárai számos típust definiálnak, lehetővé teszik adatbázisok kezelését és a velük való műveleteket, fájlokkal való műveleteket és még sok minden mást.

4.2 C#

A C# egy modern, objektumorientált programozási nyelv, melyet a Microsoft fejlesztett ki. A nyelvet 2000-ben mutatták be, alapjául a Java és a C++ nyelvek szolgáltak. Bemutatása óta nagy népszerűségnek örvend, elterjedt az asztali, a webes és a mobilalkalmazások fejlesztői körében is. Néhány kulcsfontosságú jellemzője:

- Objektumorientáltság: A C# tisztán objektumorientált nyelv, ami azt jelenti, hogy mindent objektumokként kezel. Ez a megközelítés lehetővé teszi a modellalkotást, az adatrejtést, az öröklődést és a polimorfizmust.
- Típusbiztonság: A C# erősen típusos nyelv, vagyis a típusokat azonosítani és ellenőrizni kell már fordításkor. Ez segít csökkenteni a futáshibák számát és növeli a kód biztonságát.
- Garbage Collection: A C# automatikus szemétgyűjtéssel rendelkezik, ami azt jelenti, hogy a fejlesztőnek nem kell manuálisan kezelnie a memóriát, például felszabadítani a nem használt objektumokat. Ez segít elkerülni a memóriaszivárgást és a memóriakezelési hibákat.
- Modern nyelvi elemek: A C# folyamatosan fejlődik, és rendelkezik olyan modern nyelvi elemekkel, mint például lambda kifejezések, LINQ (Language Integrated Query), async-await aszinkron programozási támogatás, pattern matching és sok más.
- Fejlett fejlesztői eszközök: A C#-hoz számos kiváló fejlesztői eszköz érhető el, mint például a Visual Studio és a Visual Studio Code, amelyek segítenek a kódírásban, hibakeresésben és tesztelésben.

A C# program fordítása után bináris állomány(ok) keletkeznek, melyek platformfüggetlen köztes nyelvet és típus-metaadatokat tartalmaznak. A .NET alapú alkalmazások futtatásához a keretrendszer telepítése is szükséges.

4.3 Visual Studio

Bár a fejlesztés során több segédprogramot is használunk, például a diagrammokhoz, kapcsolati térképekhez, adatbázis vagy interakciók modellezéshez, a legfőbb szoftvere a fejlesztőnek az IDE (Integrated Development Environment), vagyis az integrált fejlesztői környezet. A C# nyelvet támogató IDE-k közül kiemelkedik a Visual Studio.

A fejlesztéshez jelenleg a Visual Studio 2022 áll rendelkezésre. Ez a verzió már teljes mértékben 64 bites architektúrára van fejlesztve, emellett számos egyéb teljesítmény- és stabilitásnövelő fejlesztés teszi eredményesebbé a használatát. Hasznos funkciója az IntelliCode. Az IntelliCode egy olyan funkció, mely szerkesztési időben ad kódjavaslatokat a fejlesztőnek, képes különféle kódrészletek sorozatainak előállítására a felismert és értelmezett

kódok függvényében (Svyatkovskiy és tsa, 2020). Jelenleg az egyik legmodernebbnek számító generatív transzformátor modellen alapul.

4.4 Entity Framework

Az Entity Framework (EF) egy nagyon népszerű objektum-relációs leképzési (ORM) eszköz a .NET keretrendszerhez. Segítségével a fejlesztők objektumokat használhatnak az adatbázisokkal való kommunikációhoz, anélkül, hogy közvetlen lekérdezéseket kellene írniuk. Az Entity Framework lehetővé teszi az adatbázis-objektumok közötti transzformációkat és műveleteket, valamint az adatbázis-séma és az objektummodell közötti átjárhatóságot (Moldován, 2019).

A használathoz először telepíteni kell a Entity Framework NuGet csomagot. Ez a Package Manager Console-ból történik az következő parancs futtatásával:

Install-Package EntityFramework

Ezek után választhatunk a Code First vagy a Database First megközelítés között. Mivel esetünkben a tervezéskor már használtuk az Access-t az adatbázis felépítésre, volt egy létező *Varosgondozas.accdb* fájlunk, vagyis adott volt az utóbbi szemlélet szerint eljárnunk. Az adatbázisfájl segítségével generálhatóak az osztályok az adatbázishoz:

Scaffold-DbContext "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=C:\Allomanykezelo\Adatbazis\Varosgondozas.accdb" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

Ez a parancs létrehoz egy DbContext osztályt és az adatbázis tábláihoz tartozó osztályokat a Models mappában. Az így létrejött osztályok segítségével végezhetjük el az adatbázisok módosításait.

5. A PROGRAMTERV RÉSZEINEK BEMUTATÁSA

5.1 Adatbázis

A várható adatmennyiségeket figyelembe véve adott volt az új állománykezelő programban is adatbázisokat használnunk. A tanulmányaink során az Access adatbázisokkal ismerkedtünk meg részletesebben, így a programban is ezt használtam.

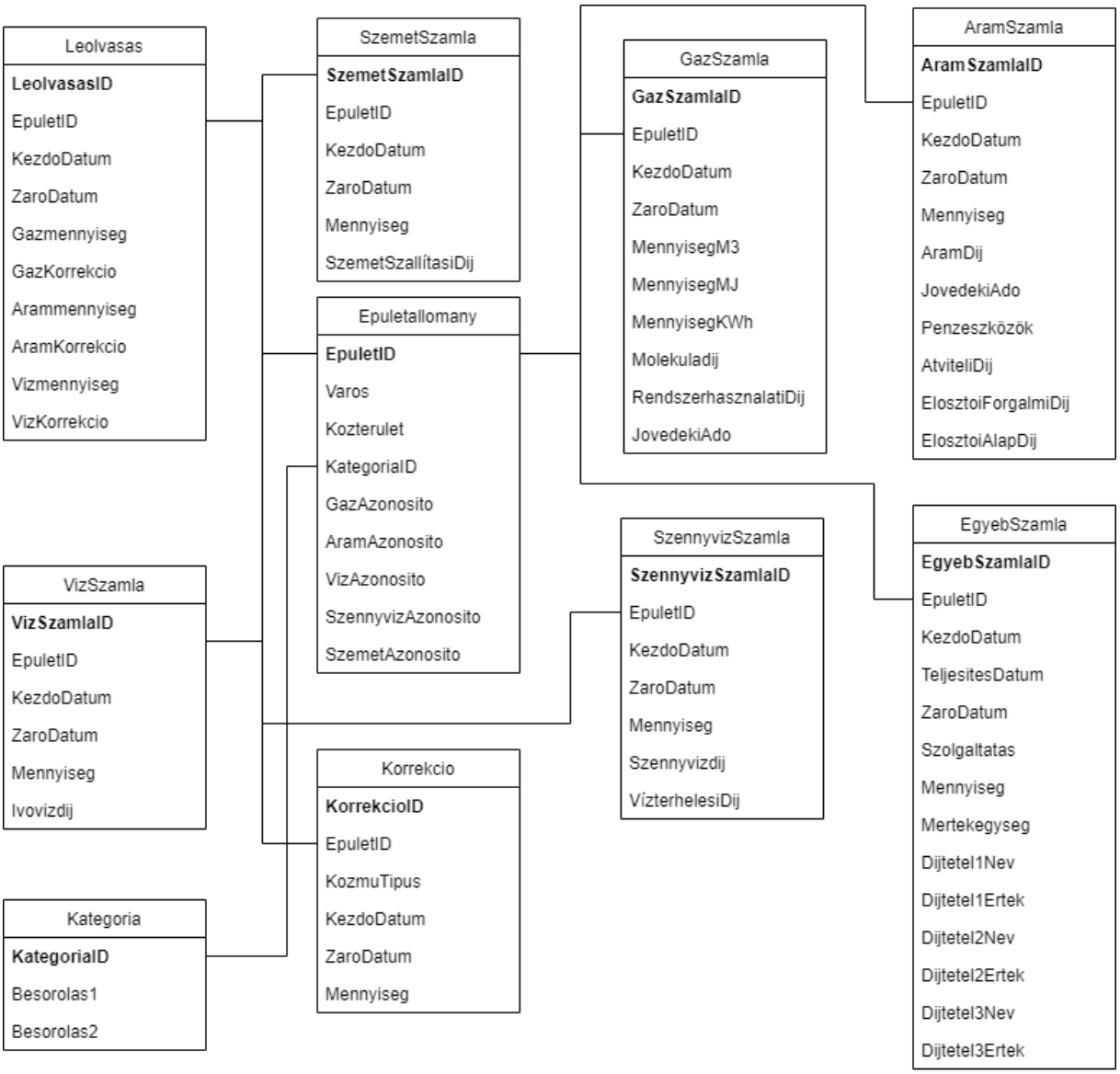
A tervezési folyamat során törekedtem olyan adatbázis struktúra kialakítására, mely használható a célkitűzésekben lefektetett feladatokra. Természetesen a professzionális programozás során egy sokkal összetettebb, de rugalmasabban fejleszthető adatbázisstruktúrát lehetett volna létrehozni, azonban a cél most csak egy, a helyi körülményekre szabott szoftver létrehozása volt. Így a számba jöhető közműtípusok (amik eddig is használatban voltak), kaptak egy-egy táblázatot, a típusuknak megfelelő mezőkkel.

A törzsállományt az *Epuletallomany* táblában találjuk, ahol az épületek tulajdonságai és a különböző közműveknél bejegyzett azonosítóik vannak felsorolva. Külön táblát kaptak a leolvasott értékek. A teljes felépítést a 2. ábra mutatja.

A praktikusság és a további fejlesztési megfontolások miatt minden táblába került egy ID ami elsődleges kulcsként funkcionál (Balázs – Németh, 2019). A legtöbb kapcsolat Az *EpuletID* tulajdonságon keresztül létesül. Ez érthető is, hiszen a bevitt adatok a legtöbb esetben az egyes épületekhez rendelhetők, másképp fogalmazva az épületek köré szerveződnek.

A *Kategoria* tábla felelős a feladás lekészítésének automatizálásáért. Már szó volt róla, hogy a feladás egy speciális lekérdezés-sorozat nyomatott formában. Ez a tábla lehetővé teszi az egyszerű kategorizálás szerinti lekérdezéseket. Az alkalmazáslogika kidolgozásakor kiemelt figyelmet kell majd szentelni ennek a táblának a szerkeszthetőségére. A jelenlegi két mező elegendő egyféle szempont szerinti felosztásra, de ha több lekérdezést szeretnénk automatizálni, akkor ezt a táblát valószínűleg módosítani, bővíteni kell majd.

A szerkezetbe nem került kapcsolótábla, mert a kapcsolatot az alkalmazáslogikai oldalon szeretnénk megteremteni a Hatékonyságnövelő megoldások című részben részletezettek szerint.



2. ábra: Az adatbázis felépítése

Az Entity Framework segítségével az adatbázisokból osztályok generálhatók a már ismertetett módon. Az alábbiakban látható egy példa a SzemetSzamla táblából képzett osztályra:

```
public class MyContext : DbContext
{
   public DbSet<SzemetSzamla> SzemetSzamlak { get; set; }
}

public class SzemetSzamla
```

```
{
    public int SzametszamlaID { get; set; }
    public DateTime KezdoDatum { get; set; }
    public DateTime ZaroDatum { get; set; }
    public int SzemetSzallitasiDij { get; set; }
}
```

Az így képzett osztályok példányai felelnek meg az adatbázis egy-egy sorának. A táblázat bővítése, azaz a tulajdonképpeni adatbevitel új példányok létrehozásával történik:

```
var ujSzamla = new SzemetSzamla {
KezdoDatum = InputKezdoDatum,
ZaroDatum = InputZaroDatum,
SzemetSzallitasiDij = InputSzemetSzallitasiDij };
context.SzemetSzamlak.Add(ujSzamla);
context.SaveChanges();
```

A fenti kódrészletben még mindig a SzemetSzamla táblában maradva láthatjuk egy új beérkező számla rögzítését. Az input adatok a felhasználói felület beviteli mezőiből kerülnek átemelésre.

Az Entity Framework leírásában egy kiemelkedő előnyének említik, hogy integrálódik a LINQ (Language Integrated Query) nyelvvel, amely lehetővé teszi az adatok hatékony lekérdezését és manipulációját C# kódban. Ez egy nagyon erős eszköz a fejlesztők számára, mivel lehetővé teszi az adatok összetett lekérdezését és feldolgozását a kódban, anélkül, hogy lekérdezéseket kellene írni. Erre egy példa a szemétszámlák közül a legutolsó bevitt adat lekérdezése:

```
var utolsoSzamla = context.SzemetSzamlak.OrderByDescending(s =>
s.SzametszamlaID).FirstOrDefault();
```

Eddig a példában a szemétdíjak hoztam fel példának, mert az egyik legkevesebb mezővel rendelkező számlatípus, tekintettel arra, hogy nincs leolvasott mennyiség, csak adott időszakban nyújtott szolgáltatás. A leolvasható szolgáltatások (gáz, áram, víz) esetében a

mennyiségek nem csak a leolvasott értékekből adódnak, hanem úgynevezett korrekcióból is. Ez akkor kerül használatra, ha a mérőegység valamilyen okból meghibásodik, nem vagy nem helyesen mér. Ilyenkor előfordulhat, hogy a leolvasott mennyiség nulla, míg a korrigált mennyiség az adott időszak becsült fogyasztása.

5.2. Felhasználói felület

A GUI (Graphical User Interface) megjelenése hatalmas minőségi ugrást jelentett a szoftverfejlesztésben. A grafikus felhasználói felület nem csak esztétikumában jelentett jobb felhasználói élményt, hanem a használatban is. A projekt szoftverét Windows Form alkalmazásként fejlesztem. Windowsra történő desktop alkalmazás fejlesztésekor adja magát, hogy a felhasználói felületet Windows Presentation Foundation (WPF) technológiával készítjük. Ez a Windows Form-hoz képest újabb, korszerűbb, fejlettebb. Többek között erőteljesebb stílusolási lehetőségeket és szabadabb elrendezést biztosít a felületek kialakítása során. Ez többek között annak köszönhető, hogy a WPF XAML (Extensible Application Markup Language) nyelvet használ a felületek létrehozásához, amely lehetővé teszi a különféle vizuális elemek és stílusok leírását. (Kovásznai – Biró, 2013).

A WPF vitathatatlan előnyei ellenére mégis a Windows Form alkalmazás mellett döntöttem. Erre három okot is fel tudok sorolni: 1. A Windows Form alkalmazás is tökéletesen alkalmas a célkitűzéseknek megfelelő alkalmazás kivitelezésére, amellett, hogy egyszerűbb és gyorsabb fejlesztést biztosít. 2. A Modell-View-Controller szemléletű fejlesztés miatt igény és kapacitás esetén lehetőség van a GUI WPF-en történő elkészítésre úgy, hogy az alkalmazáslogikán és az adatbázisokon nem kell változtatnunk. 3. Form alkalmazások fejlesztésében szereztünk a legtöbb tapasztalatot a tanulmányaink során.

A GUI megalkotásával úgynevezett eseményvezérelt alkalmazásokat kapunk. Ez azt jelenti, hogy a program futását nem az algoritmusban meghatározott folyamatok vezérlik, hanem a felhasználói interakciók, például, hogy hol történik egy kattintás vagy milyen billentyű kerül lenyomásra (Achs – Szendrői, 2015). Amíg ilyen esemény nem történik, egyéb utasítás hiányában a program várakozik.

A maximális hatékonyság elve szerint a főoldalon (mint a legkönnyebben elérhető felületen) a leggyakrabban használt funkcióknak kell lennie. Ez esetünkben a számlák rögzítése lesz. Hatféle számlát rögzíthetünk és ezek sokszor vegyesen kerülnek az adatrögzítő elé.

A főoldalon elegendő hely áll rendelkezésre a 6 számla gombnak, továbbá egy Leolvasás és egy Bezárás gombnak. Mivel a C# tisztán objektumorientált nyelv, ezért a gombok is, az összes többi alkotóelem is objektum, egészen pontosan a *System.Windows.Forms* névterek osztályainak példányai. Azonban a gombok és más hasonló objektumok különleges abból a szempontból, hogy hozzájuk eseményeket kapcsolhatunk. Az ilyen elemek a vezérlőelemek (Controls). Ebben az esetben a gombok megnyomásakor az adott számla beviteli űrlapja jelenik meg. Például a Gázszámla gomb megnyomásakor az alábbi esemény történik:

```
GazSzamlaUrlap gazszamlaurlap = new GazSzamlaUrlap();
GazSzamlaUrlap.Show();
GazSzamlaUrlap.Activate();
```

A gyors adatbevitel érdekében nagy szerepe van a körbejárási sorrendnek, azaz annak, hogy a beviteli mezők milyen sorrendben és minek a hatására kapnak fókuszt. Alapesetben a bejárási sorrend megegyezik a létrehozás sorrendjével, de a TabIndex tulajdonságnál ez felülírható. A számlák beviteli űrlapján az Enter gomb hatására a következő beviteli mezőre ugrunk, majd, az összes mező kitöltése után, a Rögzít gombra. Ezen a gombon Entert ütve lefut a számla példányának létrehozása, majd a fókusz visszakerül az első beviteli mezőre, a *GazAzonosito* TextBoxra. A többi funkciónak, például az Epuletallomany szerkesztésének, vagy a havi leolvasásokna vagy a lekérdezéseknek is külön form készül. Az adatok keresése, rendezése és összegyűjtése az oszályokként kezelt rekordok segítségével az alkalmazáslogikában történik. A gáz leolvasásokhoz például a *GazAzonosito*-k kellenek:

```
var gazAzonositok = context.Epuletallomany.Select(e => e.GazAzonosito).ToList();
```

A Windows Form alkalmazások lehetővé teszik a dinamikus objektumkezelést a felületen. Az előző lekérdezést folytatva:

```
int darabSzam = gazAzonositok.Count;
for (int i =1; i <= darabSzam; i++)
{
        Label labelGazAzonosito = new Label();
        TextBox textboxMert = new TextBox();
```

```
        labelGazAzonosito.Text = gazAzonositok [i]
        labelGazAzonosito.Location = new Point(30, labelTitle.Bottom + (i  * 30));
        labelGazAzonosito.Autosize = true;

        textboxMert.Location = new Point(LabelGazAzonosito.Width,
        labelGazAzonosito.Top -3);
        inputTextBoxes.Add(textboxMert);

        this.Controls.Add(labelGazAzonosito);
        this.Controls.Add(textboxMert);
}
```

Ekképpen a vázát meg is kaptuk a leolvasási űrlapnak, mely elkészíti a GazAzonositok listáját és generál egy beviteli mezőt is a leolvasott értékeknek, amely a létrehozás sorrendjében bejárható. A Mentés gombra kattintva a mentés előtt ellenőrizzük, hogy az összes mező ki van e töltve a *labelGazAzonosito.Text == String.Empty* feltétel vizsgálatával. Mivel előfordulhat, hogy valamilyen oknál fogva nem sikerül leolvasni, ezért az ellenőrzés során talált üres textboxokat nem hibaként adjuk vissza, hanem csak figyelmeztetjük a felhasználót az üres mezőkre. Éppen ezért lehetőséget kell teremteni a leolvasások módosítására és hiányzó adatok pótlására.

Egy további fejlesztési lehetőség a mennyiségellenőrzés funkció bevezetése. A funkció szinte minden közműszolgáltató okostelefonos appjában benne van. A lényege, hogy leolvasáskor valamilyen szempont szerint ellenőrzik a bevitt adatokat mennyiségi szempontból. Itt tehát nem adattípus vizsgálatról vagy üres-e vizsgálatról van szó. Az ilyen esetekben vagy az előző mennyiségek, vagy a korábbi adatok statisztikai vizsgálata alapján meghatározzuk a várható értéket és egy tűréshatárt vagy toleranciasávot, amit még elfogadunk. Ha bevitt adat a toleranciasávon kívül esik, a felhasználó figyelmeztetést kap a valószínűsíthetően hibás adatra. Ezzel a módszerrel elkerülhető például a véletlenül duplán leütött billentyű miatti téves adatrögzítés. Bár mennyiségi vizsgálatról beszélünk, a bevitelkor óraállásokat rögzítünk. Az előző rögzített adatok elérése nem bonyolult, viszont az elmúlt évek hasonló időszakainak fogyasztása összetettebb algoritmust és az adattáblák fejlesztését is igényelheti. Nagyobb

adatbázisok esetén, vagyis, ha sikerülne a dBase adatbázisát importálni, lehetővé válna a toleranciasáv képletes meghatározása, a korábbi évek mennyiségeinek szórását használva.

5.3. Hatékonyságnövelő megoldások

5.3.1 Kalkulálható értékek előrejelzése

A számlarögzítési folyamat elemzéséből megállapítható, hogy az idő nagy részét előre megállapítható értékek beírása teszi ki. Ilyen adat például a felhasználóazonosítóhoz tartozó épületcím. Az ilyen időveszteség elkerülése az adatbázisok egyik legfőbb előnye. Az általános elv az, hogy egy adatot csak egyszer tárolunk és a továbbiakban csak hivatkozunk rá (Halassy, 1994).

Esetünkben a számlák rögzítésekor is mindig van egy adat, amely azonosítja az épületet. Ezt különféle néven említik a különféle számlák: Felhasználó azonosító, Fogyasztási hely azonosító stb. Ezek az azonosítók egyértelműen meghatározzák az épületet. És fordítva is igaz: minden épületnek nem csak címe van, hanem minden közműszolgáltatónál létezik egy azonosítója is. Ennélfogva a számlák felvitelekor csak a fogyasztóazonosító alapján meg kell keresnünk az épületet, és az *EpuletID* tulajdonságot eltárolnunk a cím helyett.

Az azonosító beírásakor automatikusan lefut egy keresés az *Epuletallomany* táblában és kiírja a talált épület címét. Amennyiben nem talál azonosítót, figyelmezteti a felhasználót és felajánlja az épületállomány szerkesztését. Az adatbevitel rögzítésekor csak az *EpuletID* tárolódik el, de vizuálisan ellenőrizhető az adatok helyessége. Ezzel a módszerrel kiváltható a teljes címek beírása, melyek akár 20-30 karaktert is kitehetnek.

A másik kalkulálható értékcsoport a szolgáltatásokért fizetendő díjtételek összege. Általánosan jellemző, hogy a szolgáltatások egységára nagyon ritkán változik, általában évente vagy még ritkábban. Éppen ezért amikor rögzítünk egy mennyiséget, például áram esetében egy kilowattóra értéket, nagy valószínűséggel tudhatjuk az árát is, mivel az egységár hasonló lesz a korábbiakhoz. Problémát jelent viszont, hogy az egységárak épületenként változhatnak, az igénybe vett szolgáltatástól függően.

Az adatrögzítés felgyorsítására bevezetésre került egy olyan funkció, amely képes kiszámítani a mennyiségek alapján a várható részösszegeket. Az elképzelés az volt, hogy az épület és mennyiség beírása után a program számítsa ki a költségeket (a szolgáltatás adott, hiszen tudjuk, hogy milyen számlát rögzítünk). Így ahelyett, hogy manuálisan írnánk be a forintösszegeket, csak jóvá kell hagyni a program által kiszámolt értékeket. Ezzel a módszerrel

a radikálisan csökkenthető a gépelési hibák száma és a bevitelre, illetve ellenőrzésre fordított idő.

A felvázolt funkcióra több megoldás lehetséges. Az egyik, hogy létrehozunk egy külön adattáblát, amely tárolja az egységárakat és a mennyiség beírása után ebben keresünk, majd a találat(ok) alapján készítjük el a számítást. A funkciót tovább egyszerűsíthetjük, ha az *Epuletallomany* táblába további attribútumokként felvisszük az összes egységárat, és innen olvassuk be a számlafelvitelhez.

A tervezéskor viszont egy sokkal rugalmasabb és felhasználók számára barátságosabb megoldást találtam. A mennyiség beírása után itt is lefut egy keresés, de ez az adott szolgáltatás táblájában keres. Ezen belül is az adott épület legutolsó számlájának adatait keresi meg. Ezután viszont ennek a számlának az adataiból számolja ki az egységárakat, majd a kapott értékek segítségével készíti el a költségek becslését. A funkció így nagyobb részben támaszkodik az alkalmazáslogikára, mintsem az adatbázis oldalra. Ebből kifolyólag viszont több számítási kapacitást igényel és valamennyivel lassabb is. Az viszont vitathatatlan előny, hogy az egységárak frissítése így automatikusan történik, változás esetén nincs szükség módosításra, hanem a beviteli mezőben csak felülírja a felhasználó a kiszámolt értékeket, és a jövőben az új árak lesznek érvényesek. Mivel csak maximum hat költségtétel számításáról van szó, még a nagyobb erőforrásigény mellett is jelentős időmegtakarítást eredményez, hiszen az esetek túlnyomó többségében a forintértékek beírása helyett csak Enter gombbal jóvá kell hagyni a kiszámolt értékeket.

Szükség volt a módszer tesztelésére is, mert az egységárak esetében 3-5 tizedesjeggyel is számolnak a szolgáltatók. A mennyiség-érték visszaosztásból a kerekítés miatt a legritkább esetben kapjuk meg a pontos egységárakat. A becslés pontossága a mennyiség-érték adatpár nagyságával arányos, vagyis nagyobb értékek esetén pontosabb egységárat kapunk. A tesztek és a tapasztalatok alapján a mindennapi használat során megállapítható értékek elég magasak ahhoz, hogy ne kelljen néhány forintos eltéréseket javítani. Hasonló elvet használó Excel táblák használatban vannak más ágazatokban is közüzemi számlák nyilvántartására, és kielégítően működnek, pontos eredményeket adnak.

5.3.2. Exportálás

A célkitűzések között kiemelt helyen szerepel az adatok exportálhatósága. Az Excel exportáláshoz először az adatainkat egy olyan struktúrába kell átalakítanunk, amelyet az Excel

könnyen tud kezelni. Többféle adatstruktúra jöhet szóba, de legmegfelelőbb formátum erre a DataTable. A fejlesztés során viszont kiderült, hogy a korábban választott Enity Framework által visszadott objektumok nem kompatibilisek a DataTable-vel. Ennek oka, hogy a EF objektumai más típusúak és összetettebbek lehetnek, mint amit a DataTable kezelni tud.

A fejlesztés innen két úton mehet tovább: vagy megoldást találni az EF-DataTable átalakításra, vagy más módszert találni az adatbázis kezelésére. A kezelésre megoldás lehet az ADO.NET vagy az OLEDB (Object Linking and Embedding Database) általános adatbáziskezelő interfész. A webes fórumokon viszont találtam egy megoldást arra, hogy egy szerű metódussal hogyan lehet megoldani az átalakítást. A folyamat során az EF objektumokból levágásra kerülnek a nem értelmezhető részek. A *System.Reflection* névtér használatával lehetőség van a típusok közötti dinamikus interakcióra. Azt a folyamatot, mely során futásidőben lekérdezzük és manipuláljuk a típusokat vagy azok tulajdonságait, reflexiónak nevezzük. Az alábbi kód segítségével elvégezhetjük az átalakítást:

```
public static class DataTableExtensions
{
        public static DataTable ToDataTable<T>(this IEnumerable<T> items)
   {
      DataTable dataTable = new DataTable(typeof(T).Name);
         PropertyInfo[] props = typeof(T).GetProperties(BindingFlags.Public |
         BindingFlags.Instance);

      foreach (PropertyInfo prop in props)
      {
         dataTable.Columns.Add(prop.Name,
         Nullable.GetUnderlyingType(prop.PropertyType) ?? prop.PropertyType);
      }

      foreach (T item in items)
      {
         object[] values = new object[props.Length];
         for (int i = 0; i < props.Length; i++)
         {
```

```
            values[i] = props[i].GetValue(item, null);
        }
        dataTable.Rows.Add(values);
    }
    return dataTable;
  }
}
```

Ha kész a DataTable, akkor több lehetőségünk van Excel munkalappá alakítani. Ilyen az NPOI, a EPPlus vagy az Interop Excel. A programban az utóbbit használtam, úgy hogy a Visual Studioban a Hivatkozásokra kattintva hozzáadtam a "Microsoft.Office.Interop.Excel" könyvárat. Ezek után az *Excel = Microsoft.Office.Interop.Excel* névtér használatával az alábbi kóddal végezhetjük el az exportálást:

```
Excel.Application excelApp = new Excel.Application();
excelApp.Visible = true;
Excel.Workbook workbook = excelApp.Workbooks.Add();
Excel.Worksheet worksheet = workbook.Sheets[1];
for (int i = 0; i < dataTable.Rows.Count; i++)
{
  for (int j = 0; j < dataTable.Columns.Count; j++)
  {
    worksheet.Cells[i + 1, j + 1] = dataTable.Rows[i][j].ToString();
  }
}
Workbook.SaveAs(@"C:\Allomanykezelo\Exportalas\Export.xlsx");
workbook.Close();
excelApp.Quit();

string mentesekHelye = @"C:\Allomanykezelo\Exportalas";
Process.Start(mentesekHelye);
```

Az utolsó két sor szerepe csupán az, hogy megnyitja az Exportalas mappát, hogy még hamarabb elérhető legyen a létrehozott Excel tábla. További fejlesztési lehetőség a feladás még jobb automatizálása, például ütemezéssel vagy automatikus tovább küldéssel emailben.

6. ÖSSZEFOGLALÁS

Szakmai gyakorlatomat a Városgondozási Zrt-nél töltöttem, ahol tudomást szereztem egy olyan fejlesztési kihívásról, mely lehetőséget ígért a szakmai fejlődésre és a tanulmányaim gyakorlati alkalmazására.

A projekt keretében egy, az 1980-as évek vége óta használt szoftver újabb „verzióját” szeretném elkészíteni. A kutatás során kiderült, hogy az abban az időben használt legnépszerűbb adatbázis-kezelő szoftverről van szó. Hasonló kaliberű fejlesztés eredményeként minden bizonnyal milliókat érő szoftvert lehetne készíteni. Ezért a célkitűzések annyiban módosultak, hogy egy olyan programot szeretnék készíteni, amely azokat a funkciókat tudja ellátni, melyekre az eredeti, 1980-as szoftver adatbázisát készítették. Ez azt jelentette, hogy egy, az eredetihez hasonló felépítésű adatbázishoz kellett kezelőfelületet készíteni. Emellett néhány olyan újítást alkalmaztam, amelyek a grafikus felhasználói felület előnyeit használják ki és a kimeneti adatok még sokrétűbb felhasználását teszik lehetővé.

Az alkalmazott technológiák kiválasztásakor leginkább az motivált, hogy a tanulmányaim alatt szerzett ismereteket minél szélesebb körben alkalmazzam. Mindazonáltal a korszerű technológiák messzemenően alkalmasak a feladat ellátására. A speciálisabb feladatoknál, mint az Excel exportálás, érzékeltem leginkább a tapasztalat hiányát.

Windows környezetben és környezetre történt a fejlesztés C# programnyelven. Az IDE a Visual Studio 2022 volt, ezen kívül még diagramkészítő programokat használtam. Az adatbázist Microsoft Access-vel készítettem el, ebből kifolyólag a kódoláskor a Database First szemlélet szerint kellett eljárnom. A program még nem készült el teljesen. A fejlesztés legnehezebb része, az alkalmazáslogika kidolgozása jelenleg is folyamatban van. Reményeim szerint néhány hónapon belül használatra, de legalábbis külső tesztelésre kész állapotban kerül.

IRODALOMJEGYZÉK

1. Achs Á. – Szendrői E.: *Programozás 2., II. kötet Windows form alkalmazások.* Pécsi Tudományegyetem, 2015

2. Balázs P. – Németh G.: *Adatbáziskezelés*. Szegedi Tudományegyetem, 2019

3. Chen, J. W., – Jiajie, Z.: *"Comparing text-based and graphic user interfaces for novice and expert users."* AMIA annual symposium proceedings. Vol. 2007. American Medical Informatics Association, 2007.

4. Halassy B.: *Az adatbázistervezés alapjai és titkai*. IDG Magyarország Lapkiadó Kft., 1994

5. Kovásznai G. – Biró Cs.: *.NET-es programozási technológiák*. Eszterházy Károly Főiskola, 2013

6. Moldován Á.: *C# programozási nyelv adta adatbáziskezelési lehetőségek, mint oktatástechnológiai eszközök összehasonlítása.* Transactions on IT and Engineering Education 2.1 (2019): 46-61.

7. Tömösközi P.: *Szoftverfejlesztés I.* Eszterházy Károly Főiskola, 2013

8. Reiter I.: *C# programozás lépésről lépésre*. Jedlik Oktatási Stúdió Kft., 2012

9. Rierson, L.: *Developing Safety-Critical Software*. CRC Press, 2013

10. Shukla, R. K.: *Automation of Libraries and Information Centres: Using DBase III Plus.* Vol. 63. Concept Publishing Company, 1995

11. Svyatkovskiy, A. et al.: *"Intellicode compose: Code generation using transformer."*. European software engineering conference and symposium on the foundations of software engineering. 2020

ÁBRÁK JEGYZÁKE

MATE Szervezeti és Működési Szabályzat
III. Hallgatói Követelményrendszer
III.1. Tanulmányi és Vizsgaszabályzat
6.13. sz. függeléke: A MATE egységes szakdolgozat / diplomadolgozat / záródolgozat / portfólió készítési útmutatója
4.2. sz. melléklete: Nyilatkozat a záródolgozat/szakdolgozat/diplomadolgozat/portfólió nyilvános hozzáféréséről és eredetiségéről

NYILATKOZAT

a záródolgozat/szakdolgozat/diplomadolgozat/portfólió[1] nyilvános hozzáféréséről és eredetiségéről

A hallgató neve: GURÚ TAMÁS

A Hallgató Neptun kódja: AR3C8A

A dolgozat címe: ELAVULT SZOFTVER ÚJRATERVEZÉSE KONKRÉT PÉLDÁN KERESZTÜL

A megjelenés éve: 2024.

A konzulens intézetének neve: MŰSZAKI ~~TANSZÉK~~ INTÉZET

A konzulens tanszékének a neve: ALKALMAZOTT INFORMATIKA TANSZÉK

Kijelentem, hogy az általam benyújtott záródolgozat/szakdolgozat/diplomadolgozat/portfólió[2] egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, és az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe. Tudomásul veszem, hogy a megvédett és

- nem titkosított dolgozat a védést követően
- titkosításra engedélyezett dolgozat a benyújtásától számított 5 év eltelte után

nyilvánosan elérhető és kereshető lesz az Egyetem könyvári repozitori rendszerében.

Kelt: 2024. év 05. hó 21 nap

Hallgató aláírása

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.

NYILATKOZAT

Gurúz Tamás (hallgató Neptun azonosítója: **AR3C8A**) konzulenseként nyilatkozom arról, hogy a záródolgozatot áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A záródolgozatot a záróvizsgán történő védésre **javaslom / nem javaslom**[1].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[2]

Kelt: 2024 év április hó 20. nap

belső konzulens

[1] A megfelelő aláhúzandó.
[2] A megfelelő aláhúzandó.