SZAKDOLGOZAT

Tóth Patrik Csaba

2018

Pannon Egyetem

Rendszer és Számítástudományi Tanszék

Programtervező informatikus BSc

SZAKDOLGOZAT

Importáló megvalósítása személyes pénzügyi nyilvántartáshoz

Tóth Patrik Csaba

Témavezető: Dr. Heckl István

2018

Tartalomjegyzék

[1. Bevezetés 4](#_Toc512721474)

[2. Feladat leírása 6](#_Toc512721475)

[2.1. Blokklánc technológia 7](#_Toc512721476)

[2.2. FinTech Magyarországon 8](#_Toc512721477)

[3. Irodalmi áttekintés 8](#_Toc512721478)

[3.1. Mint.com 9](#_Toc512721479)

[3.2. YNAB (You Need A Budget) 11](#_Toc512721480)

[3.3. Doxo 12](#_Toc512721481)

[3.4. Quicken Premier 13](#_Toc512721482)

[3.5. Összehasonlító táblázat 14](#_Toc512721483)

[4. Személyes pénzügyi importáló tervezése 15](#_Toc512721484)

[4.1. Használt technológiák 15](#_Toc512721485)

[4.1.1. .NET keretrendszer 16](#_Toc512721486)

[4.1.2. C# programozási nyelv 17](#_Toc512721487)

[4.1.3. WPF (Windows Presentation Foundation) vagy WinForm (Windows From Designer) 18](#_Toc512721488)

[4.1.4. Feladatkezelés 20](#_Toc512721489)

[4.1.5. Verziókövetés 21](#_Toc512721490)

[4.1.6. Képernyőtervek 21](#_Toc512721491)

[4.1.7. Grafikonok 22](#_Toc512721492)

[4.1.8. Részvény árfolyamok lekérése 22](#_Toc512721493)

[4.2. Követelmények 23](#_Toc512721494)

[4.2.1. Alapvető követelmények 23](#_Toc512721495)

[4.2.2. Grafikus felület 24](#_Toc512721496)

[4.2.3. Tranzakciók importálása 25](#_Toc512721497)

[4.2.4. Tárolt adatok megtekintése 26](#_Toc512721498)

[4.2.5. Részvényadatok lekérése Internetről 27](#_Toc512721499)

[4.2.6. Adatok tárolása 27](#_Toc512721500)

[5. Személyes pénzügyi importáló szoftver megvalósítása 27](#_Toc512721501)

[5.1. Grafikus felület 27](#_Toc512721502)

[5.2. Bejelentkezés és regisztráció 30](#_Toc512721503)

[5.2.1. Regisztráció 30](#_Toc512721504)

[5.2.2. Bejelentkezés 30](#_Toc512721505)

[5.3. Banki tranzakcióik importálása 31](#_Toc512721506)

[5.3.1. Felhasználó által deklarált importálás 33](#_Toc512721507)

[5.3.2. Automatikus importálás 36](#_Toc512721508)

[5.4. Tőzsdei tranzakciók importálása 39](#_Toc512721509)

[5.4.1. Felhasználó által deklarált importálás 39](#_Toc512721510)

[5.4.2. Automatikus importálás 40](#_Toc512721511)

[5.4.3. Nyereség-veszteség kiszámítás 45](#_Toc512721512)

[5.5. Tárolt tranzakciók megjelenítése 49](#_Toc512721513)

[5.5.1. Banki adatok 50](#_Toc512721514)

[5.5.2. Tőzsde adatok 51](#_Toc512721515)

[5.6. Részvény árfolyamok megjelenítése 51](#_Toc512721516)

[5.6.1. Internetről történő lekérés 51](#_Toc512721517)

[5.6.2. Adatbázisban eltárolt adatok 53](#_Toc512721518)

[6. Összefoglalás 53](#_Toc512721519)

# Bevezetés

Mivel a személyes vagyon nyilvántartása pénzügyi alkalmazások nélkül hosszú és rengeteg odafigyelést igénylő folyamat, így az emberek többsége nem is veszi a fáradságot e nyomon követések elkészítésére. Tovább bonyolítja a helyzetet, ha egy adott személy, aki szeretné nyomon követni a pénzügyeit, de tegyük fel, hogy több banknál is rendelkezik fiókkal, vagy egy család tagjai nem ugyan annál a banknál rendelkeznek számlával mégis egyszerűen és gyorsan szeretnék megtekinteni az összesített pénzügyi kimutatásaikat. Az effajta problémák áthidalására jöttek létre a személyes pénzügyi alkalmazások.

Tehát személyes pénzügyi szoftverek egyszerűbbé teszik a felhasználók számára pénzügyeik nyomon követését és bizonyos funkciók nyújtásával segítsék beosztani, esetleg jobbá tenni azt. Ezek a funkciók többek között lehetnek:

* Ha a banki tranzakciókat tekintjük, például számon tarthatjuk mennyi pénzünk van jelenleg a bankszámlánkon, mennyi pénzt tudtunk félretenni bizonyos hónapokban, milyen kategóriájú termékekre költöttünk a legtöbbet, hány tranzakció volt a számlánkon bizonyos hónapokban, napokban.
* Ha tőzsdei tranzakciók nyomon követéséről beszélünk az alkalmazás segítségével meg tudjuk jeleníteni a jelenleg birtokolt részvényeket és azokhoz tartozó darabszámokat illetve a jelenlegi részvényárfolyamot is. Le tudjuk kérni a birtokolt részvényekhez tartozó jelenlegi és egy bizonyos intervallumra visszamenő legmagasabb, legalacsonyabb, nyitó és záró árakat. Valamint a felhasználó által eladott részvényekhez nyereség-veszteség kiszámolására is van lehetőség.

Ezekből az adatokból olyan következtetéseket tud levonni a felhasználó, amelyek segítik a sikeres pénzbeosztását, esetleges spórolást illetve a tőzsdei nyereségek esélyét. De elsősorban segítik a pénzügyi tudatosság kialakulását.

Bizonyos applikációk felajánlanak pénzspórolási tanácsokat is, amiket a felhasználó tranzakcióinak statisztikai elemzéseiből nyernek ki. Habár e tanácsok többsége fizetős funkció.

A legnépszerűbb pénzügyi szoftverek legfőbb tulajdonsága hogy a felhasználók egy helyen láthatják az összes bankszámlájuk tranzakcióját, költési szokásukat és egyéb kimutatásokat. Így egy átfogó képet kapva a személyes pénzügyeikről, ezáltal segítve annak nyomon követését és esetleges jobbá tételét is.

Fontos megjegyezni, hogy manapság az alkalmazások nagy része elérhető telefonos applikáció formájában is. Mivel a mai emberek azonnali elérhetőséget kívánnak a nap bármely pontjában, főként, ha a személyes pénzügyeik alakulásáról van szó. Tehát a sikeres pénzügyi alkalmazások mind elérhetőek telefonos készülékre is.

Az első fejezetben a feladat rövid leírása található, leírva a fő problémát, annak okát, illetve tisztázva azokat a funkciókat, amivel a programnak rendelkeznie kell.

A második fejezetben bemutatásra fog kerülni pár elterjedtebb pénzügyi alkalmazás, azok funkcióival, pozitív illetve negatív tulajdonságukkal egyaránt. Illetve ezen alkalmazások összehasonlítása táblázat formájában.

A harmadik fejezetben bemutatásra kerülnek a szoftver képernyőtervei, illetve az azokhoz kötött funkcionális követelményeiket is, figyelembe véve a második fejezetben olvasottakkal.

A negyedik fejezetben leírásra kerülnek a program megírása alatt használt szoftvereket és e szoftverek ismertetése, illetve hogy miért ezeket az alkalmazásokat használtam a szakdolgozatok megírásához.

Az ötödik fejezetben felsorolásra kerülnek a jövőbeli tervek, fejlesztési irányok a jelenleg meglévő szoftvert illetően.

# Feladat leírása

A jelenleg piacon lévő pénzügyi alkalmazások mind nagyon különbözőek, mind megjelenítésben, mint nyújtott funkciók terén. De egy közös tulajdonsággal mind rendelkeznek, méghozzá azzal hogy mind képesek feldolgozni és megjeleníteni a felhasználó által beimportált tranzakciókat. Egy számlaaggregátor applikáció kifejlesztését megkönnyíti az a tény, hogy a felhasználó által beimportált fájlok banktól függetlenül ugyan azzal a formátummal rendelkeznek. Sajnos a Magyarországon jelenlévő bankoktól és tőzsdei programoktól letölthető tranzakciós fájlok felépítése nincs egységesítve. Tehát minden fájl felépítése, struktúrája más és más.

Célom egy olyan importáló szoftver írása, amely képes feldolgozni bármely magyar banktól, illetve Magyarországon elérhető tőzsdei programból exportált számlakivonatot tartalmazó fájl adatait. Majd ezen adatokat eltárolni egy egységes formátumban lévő adatbázisba. Majd azokat egy táblázat formájában megjeleníteni.

Valamint az eladott tőzsdei tranzakciók esetén megjeleníteni a nyereség illetve veszteség jelenlegi állását. Ahol a kiszámolási módszert a felhasználó választja ki.

A FinTech (pénzügyi technológia) egy fejlődésben lévő iparág, mely hatékonyabbá teheti a pénzügyi szektort a digitális kor innovációinak felhasználásával. Ennek hatására, kiszorítva azon pénzügyi szektorban jelenlévő, úgymond kevésbé informatikai szemléletben gondolkozó szereplőket.

A fintech cégek között megkülönböztetnünk két alkategóriát.

* Enabler: a hagyományos pénzügyi rendszer résztvevőit (bankok, biztosítok stb.) támogatják, segítik annak érdekében, hogy a FinTech cégek által támasztott kihívásoknak meg tudjanak felelni.
* Disruptor: hagyományos pénzügyi intézmények (bankok, biztosítok stb.) versenytársainak tekinthetők.

A bankszektor alapvetően konzervatív jellege miatt sokáig ellenkezett a technológiai újításokkal szemben. De a mai digitalizált világban a konzervatív nézéspont nem túl jó tulajdonság.

Senkinek sem okoz nagy meglepetést, hogy a banki szektor a leginkább szabályozott intézmények közé tartozik. Tehát egy-egy újítás nehezen építhető be a fennálló jogi környezetbe. Viszont új típusú fogyasztói igények megjelenése és a gyorsan fejlődő technológia azt eredményezi, hogy a FinTech iparág az egyik legnépszerűbb új technológia lett a befektetők körében (főként külföldön). Tehát a bankok most azzal a választással szembesülnek, hogy vagy alkalmazkodnak a digitális forradalomhoz, vagy elveszítik a piaci részesedésüket.

A FinTech előnyei annyira ígéretesek hogy szinte elfeledkezünk az esetleges hátrányaikról, ami nagy hiba lenne.

* Az online elérhető adatok könnyen ellophatók harmadik féltől. Más jövedelmező célokra vagy akár személyazonosság-lopáshoz is felhasználhatók

A FinTech cégek évtizedes kriptográfiai és biztonsági kutatások csúcspontjára épülnek és bár az SHA-256 titkosítást jelenleg nagyon erősnek tartják, még mindig vannak bizonyos sebezhető területek. Ami egy nagy hátrány, mivel az emberek hajlandóak szkeptikusak kezdetben a technológiai újításokkal szemben, főleg ha a pénzügyeikről van szó. De mentségükre szóljon, hogy veszélymentes rendszer nem létezik.

## Blokklánc technológia

A blokklánc technológia (blockchain) egy főként FinTech cégek által használt újítás, ami elsősorban a pénzügyi válság után vált ismerté. Fő előnye a nyílt forráskódú fejlesztés, ami lehetővé teszi a tranzakciók elszámolását közvetítők bevonása nélkül. Ebből következik, hogy töredékére csökkennek az elszámolási idők. Tehát ami eddig napokba telt, az a blokklánc technológiával közel valós időben teljesül.

## FinTech Magyarországon

Magyarországon a tavalyi év (2017) szeptemberében benyújtásra került egy törvényjavaslat a bankok és a FinTech cégek számára. Ha a PSD2 nevű irányelv elfogadásra kerül az a két fő dolgot jelent.

* Nagyobb versenyt a bankok számára.
* A bankolást megújító innovatív cégeknek pedig könnyebb piacra lépést.

A PSD2 irányelv elsődleges célja a belső piac további erősítése. Ami előnyös, mind a fogyasztók, mind a kereskedők és a vállalkozók számára is.

Az irányelv új típusú szolgáltatók megjelenését is lehetővé teszi. Az úgynevezett külső szolgáltatók kétféle új szolgáltatást nyújthatnak: számlainformációs és fizetéskezdeményezési opciót. Ezekhez az ügyfelek engedélyével banki adatokat lehet lekérdezni, vagyis a FinTech szolgáltatók hozzáférést kapnak a banki rendszerekhez.

Ez a gyakorlatban olyan számlaaggregátor szolgáltatók megjelenését jelenti, melyek alkalmasak a személyes pénzügyi alkalmazások által nyújtott funkciók megvalósítására.

# Irodalmi áttekintés

Mivel a személyes pénzügyek a legtöbb ember számára fontos, így nagy igény van ilyen szoftverekre, főleg napjainkban, ahol az emberek kényelemben szeretik élni mindennapjaikat. Tehát egy át fogó képet szeretnének látni pénzügyeikről, mindössze pár kattintás segítségével.

Természetesen számos ilyen alkalmazás létezik. Nem szokatlan hogy egy felhasználó egyszerre használ több pénzügyi alkalmazást, mivel amit az egyik változat tud, azt nem biztos, hogy a másik is.

Habár, a pénzügyi applikációk mindegyike rendelkezik Tranzakciók Importálása menüponttal, ahol pár kattintás segítségével új tranzakciókat tudunk felvinni a rendszerbe, valamint Táblázat menüponttal, ami tartalmazza a beimportált tranzakciókat. Illetve könnyedén előfordulhat, hogy egy család használja ugyan azt az alkalmazást és a közös kasszájukat követik nyomon. Tegyük fel hogy, a család minden tagja külön banknál rendelkezik számlával (ami könnyedén előfordulhat). Így egy ilyen szoftvernek képesnek kell lennie több bank adatait is feldolgozni. Ha egy alkalmazás nem rendelkezik e funkciókkal, nem lesz túl népszerű a felhasználók körében.

Természetesen egy pénzügyi szoftvert népszerűvé tehet az általa nyújtott funkciók sokasága is. Például hogy nem csak bankoktól származó adatok, hanem tőzsdei tranzakciók kezelésére is képes. Valamint nyereség-veszteség kiszámolására is használható. Gyakori a Pénztárca funkció is, ami segítségével a felhasználók saját kedvűkre megszabhatnak egy úgymond képzeletbeli korlátot bizonyos terméktípusokra, amit nem szabad túllépniük egy általuk megszabott ideig.

Habár ezen applikációk többségének kipróbálása ingyenes, természetesen léteznek fizetős változatok is. Fizetős funkciók között szerepelhet a számlabefizetés, szöveges értesítés telefonra bizonyos tranzakciók esetén stb..

Tehát rengeteg tulajdonság tehet egy pénzügyi alkalmazás sikeressé, avagy kevésbé népszerűvé. A következőekben négy darab szoftvert mutatok be azok pozitív illetve negatív tulajdonságait bemutatva.

## Mint.com

A Mint.com egy olyan alkalmazás, amely többek között segít beosztani a pénzünket, emlékeztet a számlák befizetésére. Sőt, egy régi frissítésnek köszönhetően, szigorúan csak a felhasználó engedélyével, már be is fizethet helyettünk számlákat. Természetesen képes több bankszámlához származó adat eltárolására is. Csak a számlaforgalmat látja, azt kezelni nem tudja (számlabefizetésen kívül, ha a felhasználó igény tart erre a funkcióra).

A szoftver az első importálás után egy algoritmus segítségével megvizsgálja a felhasználó korábbi tranzakcióit, ezáltal megtudva költési szokásait. Így adva esetleges spórolási illetve költési tippeket a felhasználó számára. Az alkalmazás rendelkezik egy pontrendszerrel, ahova minden felhasználó összesített értékelése kerül be (spórolás, pénztárca korlát betartása stb. alapján kerül kiszámításra).

A Mint.com létezik telefonos alkalmazás formájában is. Sajnos a magyar bankok kezelésére nem képes, viszont a PayPal fiókokat támogatja, ami Magyarországon is elérhető.



1. ábra. Mint.com kezdőoldala.

Az ábrán látható menüpont a Mint.com kezdőoldala. Láthatjuk, a szoftver e menüpontja tartalmazza az éppen aktuális Pénztárca állását, felhasználó pontozását. Továbbá láthatjuk az esedékes számlabefizetések dátumát (négy hétre előretekintve) illetve azok pénzösszegeit is megjelenítve. Valamint az elmúlt fél év statisztikáit jövedelem szempontjából egy grafikonon megjelenítve.

**Pozitív tulajdonságok**

* Könnyen átlátható, rengeteg oktató videó található az interneten.
* Többlépcsős beléptető rendszerrel rendelkezik.
* Használta ingyenes.
* Rendelkezik tőzsdei tranzakció importálás funkcióval.
* Létezik telefonos alkalmazás formájában is.

**Negatív tulajdonságok**

* Csak az Amerikai bankkártyák adatait tudja kezelni.
* Csak ellenösszeg fejében kapunk tanácsokat a spóroláshoz.

## YNAB (You Need A Budget)

A YNAB a nevéhez hűen, csak egy pénztárca létrehozásához és kezeléséhez jó. De arra tökéletesen megfelel. Feltéve, ha az ember elég időt szán az alkalmazás megismerésére. A program, regisztrálástól számítva 34 napig ingyenes. Majd havi 5 dollár az előfizetés. Főként telefonos applikációnak tervezték, de elérhető asztali alkalmazásként is.

Habár a felhasználók többsége szerint a szoftver hatékony használatának megtanulása sok időt emészthet fel, segítségünkre lehetnek az útmutató videók, amiből az interneten rengeteg található. A YNAB a pénzünk jelenlegi helyzetével foglalkozik, és nem tekint előre esetleges jövőbeli számlákra, fizetés stb. (a Mint.com-al ellentétben). Valamint nem rendelkezik tőzsdei tranzakciók importálásának lehetőségével.



2. ábra: a YNAB kezdőoldala.

Az ábrán láthatjuk a felhasználó által létrehozott Pénztárca tulajdonságait. Ahol láthatjuk a pénztárcánk maximális korlátait, az ebből már elköltött pénzösszeget illetve a kategóriánkénti aktivitásunkat.

Bal oldalt láthatjuk a fontosabb adatokat:

* Jelenleg rendelkezünk 3,362.01 $-al
  + Amiből jövedelem 1,740.00 $
  + A spórolt pénzösszeg összesen 5,247.23 $
  + A tartozásunk (hitelkártyán) összesen $3,625.22$

**Pozitív tulajdonságok**

* Személyes „pénztárca” alkalmazás.
* Spórolásra koncentrál.
* Rendelkezik ingyenes próbaidővel.
* Sok oktató tartalom található róla az interneten.

**Negatív tulajdonságok**

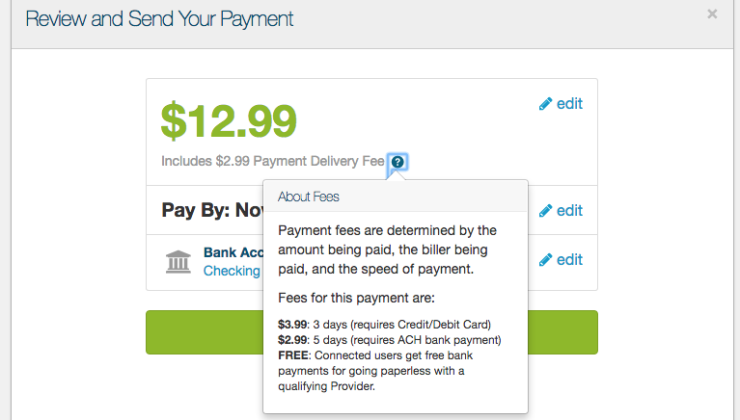
* Sok idő, míg a felhasználó meg tanulja hatékonyan használni a funkciókat.
* Nincs többlépcsős belépés.
* Főként költésre van kiélezve.
* Nem képes tőzsdei tranzakciók kezelésére

## Doxo

A Doxo egy olyan pénzügyi alkalmazás, ami főként a számla befizetésre és a banki számlához tartozó dokumentumok tárolására van kiélezve. Egy kifejezetten jó alkalmazás azon emberek számára, akik nem szeretnének többé papír alapú tranzakciós dokumentumokat (például számlákat) tárolni.

Habár az alkalmazás használata alapvetően ingyenes, rendelkezik fizetős funkciókkal is például számlabefizetés. Ami a Doxo egyik meghatározó funkciója.

Sajnos amilyen jól hangzik a számlabefizetés funkció, annyira rosszul működik. Mivel nagyon lassú. Illetve, esetleges lefagyások is bekövetkezhetnek, ami azt eredményezheti, hogy nem megy végbe az átutalás. Illetve az alkalmazáson keresztül való befizetésért számláktól függően külön kezelési költséget számol fel az alkalmazás. Valamint a kezelési költség csak a számla befizetésének végbemenetele után jelenik meg.



3. ábra: Doxo számla befizetés funkciója.

**Pozitív tulajdonságok**

* Rendelkezik számla-befizetési opcióval.
* A használata nagy részben ingyenes.
* Tud tárolni blokkokat és egyéb dokumentumokat, amik a bankszámláinkhoz tartoznak.

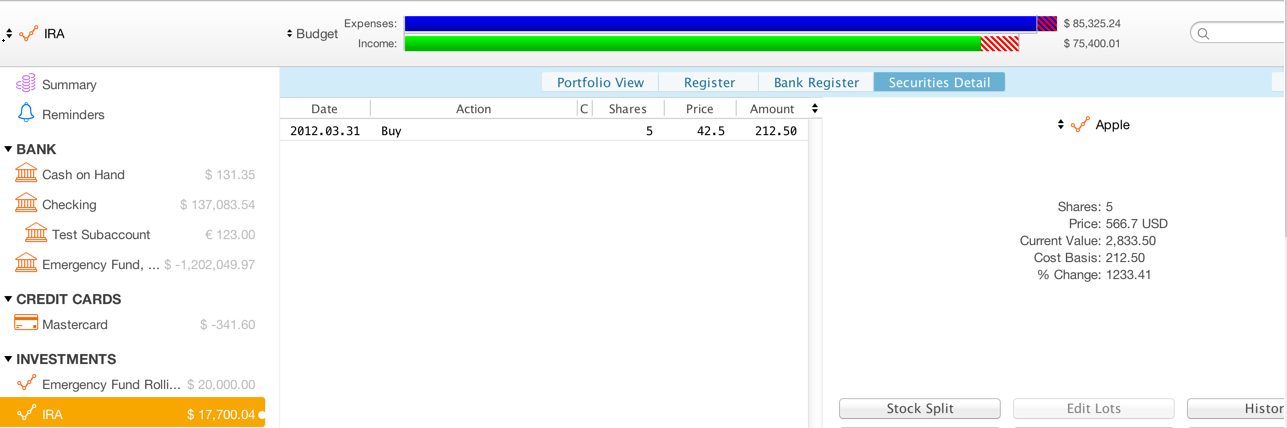
**Negatív Tulajdonságok**

* Lassú lehet bizonyos funkciók esetén például számla befizetés, tranzakciók importálás (bank függő).
* Sok negatív vélemény (felhasználók befizették a számláikat, de a pénz sosem érkezett meg).

## Quicken Premier

A Quicken Premier az egyik legrégebb óta piacon lévő pénzügyi alkalmazás. Legfőképp azon felhasználók számára ajánlott, akik már ismeretesek egyéb számlamenedzselős programokkal és tudják mit keresnek. A többi alkalmazáshoz hasonlóan mutatja a befizetéseket, a számla jelenlegi állását, pénztárca jelenlegi helyzetét.

Habár a felület talán kicsit barátságtalanabbnak tűnhet a többi piacon lévő alkalmazáshoz képest, több funkcióit, megjelenítési módot tartalmaz, mint konkurens társai. Valamint rendelkezik kiforrott befektetés követéssel is.



4. ábra: Quicken Premier befektetés kezelés.

Az fenti ábrán láthatjuk, hogy a felhasználó 2012.03.31.-én vásárolt öt darab Apple részvényt, méghozzá darabonként 42,5 dollárért. Majd az oldal jobb szélén láthatjuk, hogy ha ezt az öt darab részvényt eladná 1233,41%-os nyereséget könyvelhetne el.

**Pozitív tulajdonságai**

* Rendelkezik befektetés követő funkcióval.
* Gyakran jönnek ki új verziói.
* Elsősorban asztali alkalmazásnak tervezték, de már elérhető telefonra is.
* Az egyik legrégebb piacon lévő személyes pénzügyi alkalmazás.

**Negatív tulajdonságok**

* Nincs ingyenes változata
* Nem kimondottan kezdők számára fejlesztették ki.

## Összehasonlító táblázat

A négy alkalmazás alapján összeállítottam egy táblázatot, ami szemlélteti az alkalmazások funkcióit, illetve egyéb tulajdonságait a többi alkalmazással szemben.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alkalmazás neve | Univerzális importálás | Telefonos változat | Nyomtatás funkció | Fizetős funkció | Befektetések kezelése |
| Mint.com | + | + | + | Tanácsok | + |
| YNAB | + | + | - | csak 34 napig ingyenes használata | - |
| Doxo | + | + | - | számla befizetés | - |
| Quicken Premier. | + | + | + | éves díja 80$ | + |

# Személyes pénzügyi importáló tervezése

A szoftver tervezési fázisának elején fontos döntéseket kellet meghoznom. Elsősorban hogy milyen platformra tervezem létrehozni a szoftvert, illetve hogy milyen programozási nyelvet válasszak a fejlesztés közben fellépő problémák hatásos megoldásához. Mivel célom volt a platformfüggetlenség. Valamint egy könnyen változtatható, látványos grafikus felület létrehozása, ezért a C# programnyelvet választottam. Így csupán NET keretrendszerre van szükség a célszámítógépen, ami pedig rendelkezésre áll a felhasználók által leggyakrabban használt operációs rendszerek mindegyikén (Windows, Linux, Mac).

Illetve a C# platform előnyei között szerepel:

* Objektum orientált programozási nyelv, amely megkönnyíti a programozó dolgát bizonyos problémák könnyű megoldásában, illetve program struktúrájának átlátásában.
* Támogatja a többszálú alkalmazások létrehozását.
* Rengeteg ingyenes könyvtárra rendelkezik.
* Számtalan oktató illetve ismeretterjesztő tartalom található az interneten.

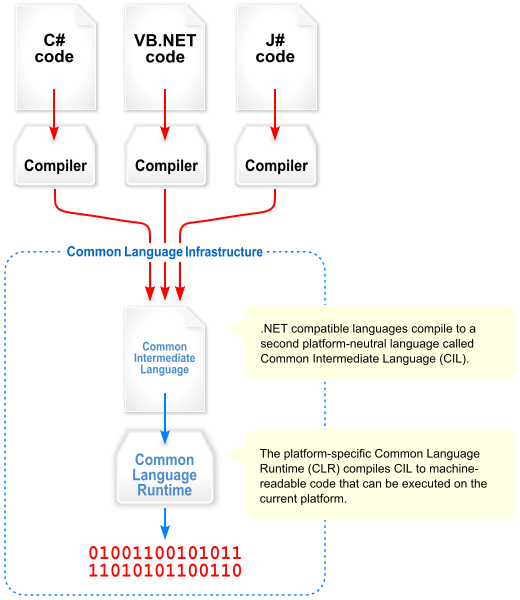
Valamint az sem volt az utolsó szempont hogy egy eddig ismeretlen programozási nyelvet tanuljak és sajátítsak el a szakdolgozatom megírása által.

## Használt technológiák

Ebben a fejezetem bemutatom a szoftver elkészítése során használt technológiákat.

### .NET keretrendszer

A Microsoft .NET rendszere többféle programozási nyelven is programozható - feltéve, ha az adott programozási nyelvhez van olyan fordítóprogram, amely alkalmazza a Common Language Specification elveit és ajánlásait, valamint képes a .NET keretrendszer virtuális kódjára fordítani. Sok ilyen programozási nyelv létezik. Például a Microsoft által direkt támogatott nyelvek a Visual Basic, J# , C#, Visual C++.



5. ábra: A .NET programkódok fordítási folyamata.

(A Common Language Specification olyan szabályok halmaza, melyek leírják azokat az alapvető normákat, amelyre a programozási nyelveknek illeszkedniük kell.)

Mivel a .NET keretrendszerben mindegy, hogy milyen nyelven programozunk, az egyik programozási nyelven megírt eljárást a másik nyelvből is meg lehet hívni és használni. Fontos hogy ez a funkció csak abban az esetben valósítható meg, ha a nyelvek megegyeznek bizonyos alapvető elvekben. Ilyen elvek közé tartozik a tömbök és rekordok, ábrázolása és használata, alapvető típusok (Integer, Double, String, stb.) reprezentációja a memóriában.

A Common Language Runtime (CLR) egy végrehajtási környezet. Ami egy rétegként működik az operációs rendszer és a .NET nyelveken írt alkalmazások között, (azon nyelvek, melyek megfelelnek a Common Language Specification (CLS) szabványnak). A Common Language Runtime (CLR) fő funkciója a kezelt kód natív kóddá való átalakítása, majd a program végrehajtása. A program végrehajtása során a CLR a memóriát, szálkezelés, memória felszabadítást (Garbage Collection), kivételkezelést, jogosultsági rendszert kezelést és egyéb rendszerszolgáltatásokat kezel.

### C# programozási nyelv

A C# az a programozási nyelv, ami a legközvetlenebb módon tükrözi az alatta működő, minden .NET keretrendszeren futtatható programot. A nyelv adattípusai az objektumok. Több korlátozást és továbbfejlesztést is tartalmaz a C és C++ nyelvekhez képest. Például:

* A legtöbb objektum-hozzáférés csak biztonságos hivatkozásokon keresztül tehető meg.
* Rendelkezik Garbage Collection funkcióval, ami automatikusan felszabadítja azon memória területeket, amire már nincs hivatkozás.
* A C++-tól eltérően C# csak egyszeres öröklődést enged meg. Viszont egy osztály több interfészt is megvalósíthat
* A C# sokkal típus biztosabb, mint a C++. Az egyetlen implicit konverzió a biztonságos konverzió. Úgy, mint az egészek tágabb intervallumba konvertálása vagy a leszármazott osztályok alaposztályba konvertálása. Nem lehetséges az implicit konverzió a logikai (boolean) és az egész típusok között, vagy a felsorolás tagok és az egészek között.

Mivel a C# kétféle grafikus felhasználói felület létrehozását is támogatja:

* WPF (Windows Presentation Foundation)
* WinForm (Windows From Designer)

Meg kellet vizsgálnom, hogy melyik felületet választásával tudnám elérni a legjobb grafikus felületet létrehozását.

### WPF (Windows Presentation Foundation) vagy WinForm (Windows From Designer)

A fő különbség a két felület között hogy a WinForm csak egy felületet biztosít az alapvető képernyőelemek fölé (például egy Szövegdoboz). Viszont a WPF et használva felépíthetjük akár a semmiből saját készítésű képernyőeleminket. Egy jó példa erre a különbségre egy Gomb elem készítése, amiben van egy kép és egy Szövegdoboz is, ez nem egy beépített képernyőelem, így a WinForm nem ajánlja fel nekünk ezt a lehetőséget. Tehát nekünk kell implementálnunk az ezt megvalósító metódust, ami a következőképpen lehetséges

* Létre kell hoznunk egy olyan gombot, ami támogatja a képek megjelenítését.
* Vagy importálunk kell egy harmadik féltől származó kiegészítést.

Ellenben a WPF felületen egy Gomb komponens tartalmazhat bármit, mivel ez az elem végülis egy keret, ami bizonyos állapotokkal rendelkezhet (például érintetlen, kattintott, letiltott stb.).

Ez a tulajdonság nem csak a Gombról mondható el, hanem az összes WPF-ben létrehozható elemről is. Ha az akarjuk megoldani WPF felületen, hogy egy Gomb tartalmazzon egy képet és egy Szövegdobozt nincs más dolgunk, mint a Gomb tartalmába beilleszteni ezeket az elemeket. Viszont ennek a kötetlenségnek és a semmiből való felépítésnek az a hátránya, hogy bizonyos esetekben több időt kell fordítanunk olyan elemek létrehozására, amik a WinForm korlátozott felületén alapvető képernyőelemként elérhetőek.

#### XAML

Az XAML az XML nyelv egy változata, amit a Microsoft fejlesztett ki a grafikus felhasználó felület leírására WPF megjelenítési felület esetén. A WinForm grafikus elemei és azok tulajdonságai, változásai ugyan azon a nyelven íródtak, mint maga a program. Viszont XAML esetén külön van választva a megjelenítésért felelő kód és a programkód, ami segíti elkülöníteni a programozók munkáját.

#### Databinding

Röviden, a Databinding egy olyan technológia, ami a képernyőn megjelenített elem tulajdonságát összeköti egy programkódban szereplő objektum értékével.



6. ábra: Databinding fajtái WPF esetén.

A cél adat jelenti a grafikus felületlen megjelenített adatot, míg a forrás a programkódban lévő objektumot. A cél és a forrás adat összekötése databinding használatával van megvalósítva.

* Egyszeri: (Az ábrán nincs illusztrálva) E kötés esetén, ha a cél adat megváltozik, arról a forrás nem fog tudomást szerezni.
* Egyszeri (Az ábrán OneWayToSource): Az Egyszeri kötés fordítottja, tehát akkor frissíti egyszer a forrás értéket, mikor a cél érték megváltozik.
* Egy irányú: (Az ábrán OneWay) Az egy irányú összerendelés a cél érték automatikus megváltoztatását eredményezi, abban az esetben, ha megváltoztatjuk a forrás adatot, de a célobjektum tulajdonságainak módosítása nem hat vissza a forrás értékre.
* Kétirányú: (Az ábrán TwoWay) a Kétirányú összerendelés megváltoztatja a forrásadatot vagy a céladatot a másik frissítése esetén. Tehát ha a kódban változtatunk egy tulajdonságot, az kihatással lesz a felhasználói felületen megjelenő elem tulajdonságára is, valamint ugyan ez fordítva is igaz.

**Összehasonlítás**

WPF előnyei:

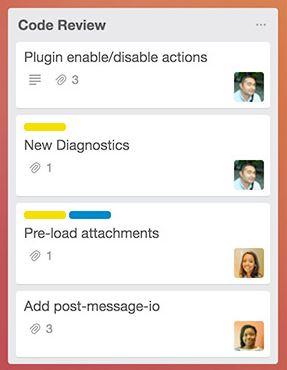
* Újabb, tehát jobban szinkronban van az új technológiákkal.
* Rugalmas, több dolgot csinálhatunk anélkül, hogy egyéb, harmadik féltől származó kiegészítőket implementálnunk kellene.
* XAML segítségével könnyen el tudjuk különíteni a megjelenítését és a programozás részét a programunknak.
* Rendelkezik Dabinding-el, ami segítségével össze tudjuk kötni a programkódunkban lévő objektumokat a grafikus felületen megjelenített elemek tulajdonságaival.

WinForm előnyei:

* Régebbi technológia, ezáltal többet tesztelték és próbálták.
* Sok harmadik féltől származó kiegészítés létezik, amit ingyen letölthetünk.

### Feladatkezelés

A Trello egy web alapú projektmenedzsment alkalmazás, aminek segítségével nyomon tudjuk követni, hogy állunk a számunkra kitűzött célok elérésében. A Trello alapvetően egy weboldal, amely vízszintesen felsorolt ​​listákat tartalmaz, így egy madártávlatos képet kaphatunk a projektről, amin dolgozunk. A listákon belül kártyák helyezhetőek el, amik a részfeladatok és az ahhoz tartozó információkat tartalmazzák. Ezen kártyák mozgathatok, szerkeszthetőek.



7. ábra: Trello folyamatok állapotának áttekintése.

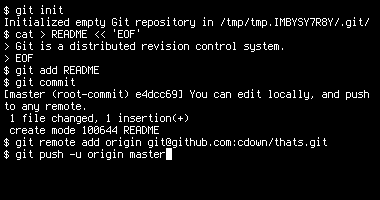
Az egyes kártyák tartalmazhatnak ellenőrző listákat, képeket, csatolmányokat, határidő dátumokat, színes címkéket és vitafelvételeket másoktól, akik osztoznak a projecten. Annyi sajátkészítésű táblát csinálhatunk, amennyit csak szeretnénk. Az egyik például "Otthoni teendők" a másik "Szakdolgozat" és így tovább.

Ha több fős a project csapattal rendelkezünk, akkor különböző emberekhez rendelhetünk bizonyos részfeladatokat, erről ők is értesítést kapnak email-en

### Verziókövetés

A Git röviden egy nyílt forráskódú verziókezelő rendszer, számítógépes fájlok módosításainak nyomon követéséhez. Valamint segít koordinálni a munkát e fájlokon több ember között. Elsősorban a forráskód kezeléshez használják szoftverfejlesztésben, de felhasználható fájlok bármely változatának nyomon követésére is.

A konkurens verzióvezérlő rendszerhez hasonlóan a Git esetében is minden számítógépen megtalálható a Git könyvtár teljes körű archívuma, a teljes változtatási előzményekkel és verziókövetési lehetőségekkel, függetlenül a hálózati hozzáféréstől.



8. ábra: verziókövetés Git használatával

### Képernyőtervek

A Pencil Project nevű mockup lehetővé teszi, hogy gyorsan és egyszerűen hozzunk létre szórakoztató és interaktív grafikus felületet, maketteket. Az alkalmazás rengeteg beépített elemmel rendelkezik és nagyon könnyű kezelni. Valamint egy egyszerű behúzós felülettel, segítségével tudunk új elemeket hozzáadni a készülő felülethez.

A Pencil Project természetesen engedi, hogy saját képeinket is beimportáljunk. Letölthető asztali felületre, de használható böngészőn keresztül is. Habár a telepített verziósokkal több képernyő elemmel rendelkezik.

### Grafikonok

Mivel a WPF grafikus felülete nem rendelkezik alapértelmezett grafikon elemmel. Viszont egy ilyen grafikus elem sajátkezű implementálása túl sok időt emésztett volna fel, ezért a legcélszerűbb opciónak azt találtam, hogy egy harmadik féltől származó könyvtárt fogok használni a grafikonok megjelenítésére. A választásom a Live-Charts könyvtárára esett.

Azért erre a könyvtárra esett a választásom mivel nagyon szépen kidolgozott, könnyen skálázható elemekkel bővíti ki a grafikus elemek listáját. Valamint WPF felületen történő megjelenítést és a DataBindigot is támogat. Illetve remek oktató anyagok találhatóak a projekt publikus GitHub oldalán.

### Részvény árfolyamok lekérése

A bizonyos időintervallumra visszamenő részvényekhez tartozó adatok (nyitó, záró, legmagasabb, legalacsonyabb ár) lekéréséhez találnom kellet egy API-t (Application Programming Interface***)*** amely lehetővé tesz ezen adatok egyszerű és gyors lekérését.

Ezen feltételeket figyelembe véve, az IEX alkalmazást választottam. Ami lehetővé teszi a részvényadatok lekérését csv fájl formátumban. Egy lekéréshez két adatot kell szolgáltatnunk az alkalmazás számára, a tőzsdei szimbólumot, például az Apple Inc. esetén a tőzsdei szimbólum AAPL. Valamint egy intervallumot, ami az aktuális naptól számított időtől számított

* 5,2,1 év
* 6,3,1 hónap
* 1 nap

Valamint nagyszerű dokumentáció és példa kódok szerepel az alkalmazás hivatalos oldalán. Az API-n keresztül való adatlekérés egy böngészőn beírt linken keresztül érhető el.

## Követelmények

A következő alapvető követelményeket állítottam fel a programmal szemben a tervezési fázis elején.

### Alapvető követelmények

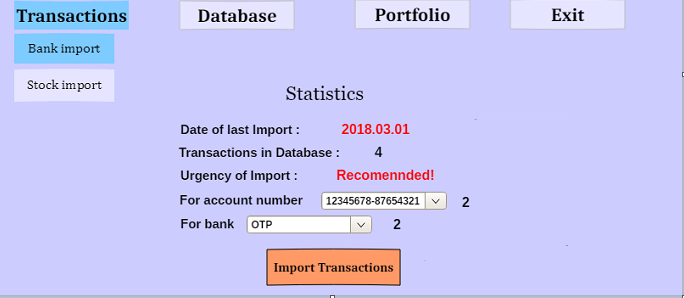
* A szoftver rendelkezzen egy felhasználói adatbázissal, ahonnét a bejelentkezés során ellenőrzésre kerülnek a beírt adatok. Illetve a felhasználónak legyen lehetősége regisztrálni.
* Importálás esetén a felhasználónak jelenjen meg egy fájl ablak, ahol a saját számítógépén levő fájlokat tudja kiválasztani.
* A program támogassa az Excel, valamint a CSV kiterjesztésű fájlok beolvasását.
* A felhasználónak legyen lehetősége egyszerre több fájlt importálni. Ezalatt az értendő, hogy az import fájl kiválasztásra során a control billentyű lenyomásával több fájl is kijelölhető egyszerre.
* Banki tranzakciók esetén egy felhasználóhoz tartozhasson több számlaszám is.
* Banki tranzakciók esetén egy beolvasott tranzakció csak abban az esetben lesz eltárolva, hogyha legalább az alábbiakban felsorolt információk szerepelnek benne:
  + Tranzakció dátuma
  + Tranzakcióhoz tartozó összeg
  + Tranzakciós fájlhoz tartozó bankszámlaszám
* Ezen felül az alábbi adatok kiolvasására és eltárolására lesz lehetőség:
  + Tranzakcióhoz tartozó leírás
  + Számlaegyenleg
* A tőzsdei tranzakciók esetén egy beolvasott tranzakció csak abban az esetben lesz eltárolva, hogyha az alábbiakban felsorolt információk szerepelnek benne:
  + Tranzakció dátuma
  + Termék neve (Vásárolt/Eladott részvény neve)
  + Tranzakcióhoz tartozó összeg
  + Tranzakció típusa (Eladás vagy Vásárlás)
  + Vásárolt/eladott mennyiség
* A hatékonyság érdekében, minden menüből a program futása során csak egy példány jöjjön létre. Amit Singleton tervezési minta segítségével oldok meg.

### Grafikus felület

Kezdetben a grafikus felület megtervezésére fókuszáltam, mivel az, az alkalmazás egyik legmeghatározóbb része. Az alábbi követelményeket állítottam fel a grafikus felülettel szemben.

A Bindingot részesítem az EventHandlerrel szemben előnyben. Mivel ha EventHandlert használok egy bizonyos esemény kezelésére, akkor az EventHandler függvénynek mindenképpen abban az osztályban kell lennie ahol az elem létre van hozva (Az elemek az osztály adattagjai képezik), így egy olyan kód keletkezik ahol keveredik az üzleti, és a megjelenítésért felelő kód.

* Ahol lehet ott Databinding segítségével fogom megoldani az események bekövetkezését, adatok lekérését, változtatását. Például a gombokhoz rendelt parancsok, táblázatokhoz tartozó adatok, legördülő lista elemei, aktuálisan kiválasztott elem stb.
* Ha nem lehetséges megoldani a Databinding segítségével egy bizonyos esemény kezelésér, akkor EventHandlert fogok használni.
* A rendszer bizonyos funkciók elérésére tartalmazzon gombokat, amikre rákattintva a felhasználó meg tudja jeleníteni a kattintott menü tartalmát.
* A felhasználó a fontosabb eseményekről az oldal közepére kapjon egy felugró szövegdoboz (Message Box) értesítést.



9. ábra: korai képernyőterv a banki tranzakciók importálásához.

### Tranzakciók importálása

Mivel az új tranzakciók importálása volt a fő funkciója a készülő programnak ezért annak tervezésével töltöttem el a legtöbb időt. Az alábbi követelményeket állítottam fel az importálás funkcióval szemben:

* Abban az esetben fog eltárolásra kerülni egy importált banki tranzakció, ha annak adatai még nem szerepelnek az eltárolt adatok között. Kivéve, ha a felhasználó mindenképpen importálni szeretné a tranzakciót.
* .A tőzsdei tranzakciók esetében a felhasználó választhassa ki a nyereség - veszteség kiszámolási módszert.
* A felhasználó kapjon egy felugró szöveges értesítőt mind banki, mind tőzsdei fájl importálásának végén, hogy melyik fájlból hány darab tranzakciót sikerült felvinnie, illetve ha volt egyezés az eltárolt adatok és az importált adatok között azt is tudassuk vele.
* Mind tőzsdei, illetve baki tranzakciók esetén a rendszer adatbázisában ugyan abban a sorrendben kerüljenek eltárolásra a tranzakciók, ahogyan az importált fájlban voltak.
* Az alkalmazás rendelkezzen egy automatizált, illetve egy felhasználó által deklarált importálási lehetőséggel.

#### Automatikus importálás

Ha a felhasználó az automatikus importálási folyamatot választja, a szoftver olyan algoritmusokat használjon, ahol az importált fájlt megvizsgálva megpróbálja beazonosítani azon oszlopokat illetve cellákat, amelyek egy érvényes tranzakció beolvasásához szükségesek. Ha az algoritmusok által nem sikerül beolvasni az importált fájl adatait, akkor a felhasználó kap egy értesítést, hogy használja a Felhasználó által deklarált importálást. Sikeres importálás esetén a kiolvasott tranzakciók kerüljenek elmentésre a rendszer adatbázisba.

#### Felhasználó által deklarált importálás

Ha a felhasználó szeretné vezérelni az importálás folyamatát, akkor legyen lehetősége a rendszerbe felvinni az általa beazonosított oszlopokat illetve cellákat. Miután az összes adatot felvitte, egy gombra kattintva jelzést adhat a szoftvernek, hogy az elkezdheti a tranzakciók adatainak kiolvasását a felhasználó által megadott információkat használva. Sikeres importálás esetén a kiolvasott tranzakciók kerüljenek elmentésre a rendszer adatbázisba.

### Tárolt adatok megtekintése

* A tárolt adatok megjelenítésé táblázaton keresztül történjen, ahol az oszlopok a tranzakcióhoz tartozó tulajdonságok.
* Ha a táblázat egy cellájának maximális mérete kisebb, mint a megjelenítendő adat hossza, akkor legyen lehetőség egy szövegbuborék segítségével megjeleníteni a teljes adatot.
* Tőzsdei tranzakciók esetén, ha nyereséges volt az adott részvény eladása, akkor azt jelezzük a táblázatban zöld színnel, ellenkező esetben piros színnel.
* Ha szükséget táblázat jobb oldalán jelenjen meg egy görgető csúszka.
* Mind banki és tőzsdei tranzakciókat tartalmazó táblázat esetén minden adatnál jelenjen meg az importálás dátuma.

### Részvényadatok lekérése Internetről

* A felhasználónak legyen lehetősége beírni az általa lekért részvény szimbólumát. Majd megadnia, hogy milyen dátumra visszamenőleg szeretné lekérni az adatokat.
* A lekért adatok jelenjenek megy egy gráfon, ahol a metszéspontok tartalmazzák árat.
* A lekért adatok kerüljenek elmentésre a rendszer adatbázisába.

### Adatok tárolása

* Fontos alapkövetelménye az adatok mentésnek, hogy a felhasználó a folyamatból semmit ne érzékeljen. Ezen feltétel alatt azt értem, hogy külön szálon fusson az adatbázisba való írás, hogy ne akadályozza a felhasználó tevékenységét eseteleges ,,megfagyásokkal" .
* Az alkalmazás egy belső MSSQL adatbázisba tárolja el az importált adatokat. Amit külső felhasználó is eltudjon érni.
* Külön adatbázis kell a felhasználói adatok, a tranzakciókhoz tartozó információk, Illetve az internetről lekért részvényadatok tárolásához is.
* A felhasználókhoz tartozó jelszavak titkosítva legyenek.

# Személyes pénzügyi importáló szoftver megvalósítása

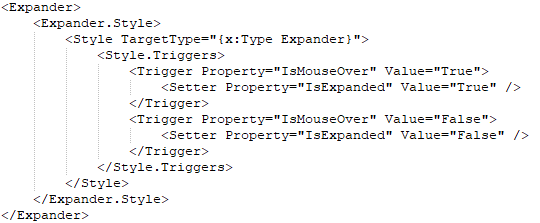
Ebben a fejezetben be fogom mutatni az elkészült Személyes pénzügyi importáló szoftver megvalósítását, illetve a bizonyos részekhez tartozó implementációkat.

## Grafikus felület

Mint ahogy a tervezés fejezetében leírtam, kezdetben a grafikus felület megvalósítására fókuszáltam.

1. Fontosnak tartottam a modern megjelenést, így egy olyan előugró menüt dolgoztam ki, ami abban az esetben lesz teljesen látható, ha a felhasználó ráviszi az kurzort. Az implementációt nagymértékben megkönnyítette XAML-ben lévő Tag-ek sokasága.

* Természetesen gondoltam az almenükre is, az almenük abban az esetben jelennek meg egy főmenü alatt, ha a felhasználó ráviszi az egerét a főmenüre-ikonjára, viszont ha egy új menüpontra húzza át az egeret, akkor automatikusan bezáródik a régi almenü és megnyílik az új.
* Az almenü abban az esetben is bezáródik, ha a felhasználó elviszi az egeret az alkalmazás menüjéről.



10. ábra: Az almenük megjelenése és eltűnéséért felelős XAML kód

Az Expander egy olyan XAML Tag, amit használva az elemek tartalmazhatnak magukban több elemet is. Létrehoztam egy egyedi stílust (Style) ,aminek megadtam egy célt, hogy mire hajtsa végre a stílusváltozást (TargetType), aminek megadtam magát az Expandert (hogy az egész elemre kihatással legyen).

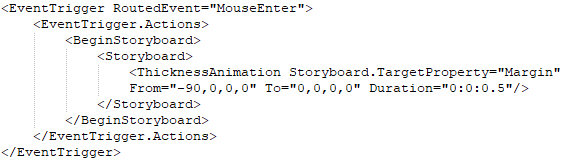
Ezek után meg kell adnom, hogy a stílusváltozás egy esemény bekövetkezésekor lépjen életbe (Style.Trigger). Ez az esemény nem más, mint az IsMouseOver állapot, ami akkor lesz igaz, ha az adott elem, amiben az expanderben van, a kurzor alatt helyezkedik el.

Ha az IsMouserOver igaz, automatikusan kibontjuk az Expander elemet, úgy, hogy az elem IsExpanded tulajdonságát (Property) expliciten igazra (True) állítjuk. Így láthatóvá válnak a benne elhelyezkedő elemek is.

Ha az IsMouseOver hamis, tehát a kurzor alatt már nem az aktuális elem helyezkedik el. Annak IsExpanded tulajdonságát expliciten hamisra (False) állítjuk, ezzel bezárjuk az Expandert.

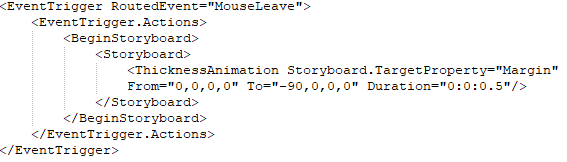
1. A következő feladatom volt az előugró menü kódjának leimplementálása.

* A menü elmeit beletettem egy Panel-be. A panel alapértelmezetten -90 pixel távolságra helyezkedik el a bal margótól, tehát a felhasználó számára láthatatlan. Fontos megjegyezni, hogy ha a Panel-ben elhelyezkedik egy elem, és a Panel tulajdonságát változtatom, változni fog vele a benne lévő elem is. Ezt a tulajdonság változást kihasználva, elég a Panel-hez hozzárendelni egy stílust, ami kihatással lesz a benne lévő elemekre is.
* A stílus változás előidézéséhez az EventTrigger Tag-et választottam, ami akkor lét életbe ha a paraméterben átadott esemény bekövetkezik.
* MouseEnter, ami akkor lép életbe, ha a jelen esetben a Panel-be történik egy kurzor belépés. Ilyenkor elkezdődik egy Storyboard. A Storyboard egy olyan Tag, amelyben eseményeknek időkorlátot, késleltetést állíthatunk be. Végül, a Storyboardon belül eltolom a Panel bal margóját 90 pixellel jobbra, méghozzá fél másodperces késleltetéssel.



11. ábra: a menü kicsúszásáért felelő XAML kód

* MouseLeave, ami akkor lép életbe, ha az elemről ,,lehúzzuk" a kurzort. Ilyenkor ismét elkezdődik egy Storyboard, csak most a bal margót 0 pixeltől eltolunk -90 pixelig, itt is fél másodperces késleltetéssel.



12. ábra: menü visszacsúszásáért felelő XAML kód.

## Bejelentkezés és regisztráció

A program indítása során elsőként a bejelentkező felület jelenik meg. Ahol a felhasználónak két opcióra van lehetősége.

### Regisztráció

Ha a felhasználó nem rendelkezik fiókkal, akkor a regisztrálás opciót kell választania, hogy létrehozhasson egy új fiókot.

A regisztráláshoz a felhasználónak meg kell adnia a létrehozandó fiókhoz tartozó felhasználónevet. Majd a fiókhoz tartozó jelszót.

A létrehozandó fiókhoz tartozó követelmények:

* Regisztrálás során nem választhatunk olyan felhasználónevet, ami már szerepel a rendszer adatbázisában.
* A jelszónak legalább 4 karakter hosszúságúnak kell lennie.
* Biztonsági okokból a jelszó beírására kétszer van szükség. A két jelszónak egyeznie kell.

Sikeres regisztráció esetén a fiókhoz tartozó jelszó az adatbázisban titkosítva kerül elmentésre. A titkosításra a C# alapkönyvtár által biztosított osztály alkalmaztam. A felhasználó kap egy szöveges értesítést a fiók létrejöttéről.

### Bejelentkezés

A bejelentkezés folyamatát egy gombra kattintva van lehetőség elkezdeni. A felhasználónak ehhez, ki kell töltenie a felhasználónév illetve a jelszó mezőket. Ha ezt nem teszi meg, kap egy szöveges értesítést ezen adatok kitöltéséről. A bejelentkezés a következő folyamatokból áll:

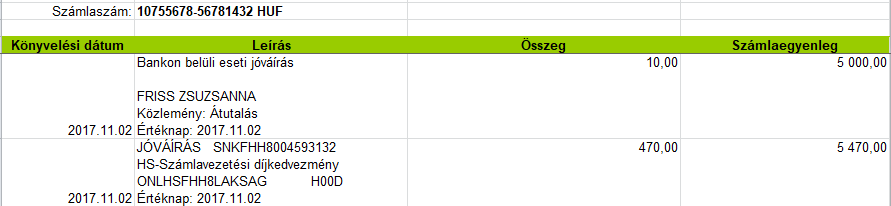
* Az alkalmazás ellenőrzi, hogy létezik-e a felhasználó által beírt fióknév a rendszer adatbázisában.
* Ha létezik a fióknév, akkor az eltárolt, titkosított jelszót visszafejtésre kerül, majd megvizsgáljuk, hogy egyezik-e a beírt jelszóval.
  + Ha igen, akkor a felhasználó számára megjelenik az alkalmazás főmenüje, a Banki tranzakciók importálása.
* Ha nem sikeres a bejelentkezés, akkor a bejelentkezni kívánó személy kap egy értesítést, hogy eltárolt adatok nem egyeznek az általa beírt adatokkal.
  + Sikertelen belépésre háromszor van lehetősége a felhasználónak, ha ezt a három lehetőséget túlhaladja, harminc másodpercig nem tud újrapróbálkozni.
  + A harminc másodperc leteltével ismét három lehetősége van a bejelentkezésre.

## Banki tranzakcióik importálása

A tranzakciók importálását először a banki tranzakciók beolvasásával kezdtem el megvalósítani. Első feladatom volt, hogy minél több számlakivonatot tartalmazó fájlt szerezzek. Ami nem volt egyszerű feladat. Egyrészt azért nem, mivel az emberek bizalmatlanok a pénzügyi dolgaikat érintően (ami érthető). Másrészt mivel célom volt minél többféle banktól származó fájl megszerzése ezért olyan embereket kellet találnom, akik más és más bankoknál rendelkeznek fiókkal. Egy idő után gyakori volt az a probléma hogy nem találtam olyan embert, aki olyan banknál lenne, ahonnét még nem rendelkezek, tranzakciós fájlal.

A fájlok szerkezete nagyon különbözőnek bizonyult, egyes fájlok nem rendelkeztek egyenleg oszloppal, vagy két oszlopban helyezkedett el a tranzakció összege (Jóváírás, Terhelés). Mindegyik fájltípus más formában jelenítette meg a tranzakció dátumát, több olyan oszlop volt, ami megfelel tranzakció leírásának (Közlemény, Leírás, Információ stb.).

A Következőekben bemutatok két különböző banktól származó fájlt (a bankszámlaszámok természetesen nem valódiak).



13. ábra: CIB banktól letölthető tranzakciós fájl.

Bátran kijelenthetem, hogy az egyik legbarátságosabb formátummal a CIB banktól letölthető fájl rendelkezik, több okból kifolyólag.

A legpozitívabb tulajdonságnak az bizonyult, hogy összesen négy oszloppal rendelkezik a fájl, plusz egy cella, amiben a számlaszám helyezkedik el. Ahogy a korábbiakban leírtam fájlok nagy részében, több oszlop tartalma is megfelelhet tranzakció leírásának, itt nem lép fel ez a probléma mivel csak egy Leírás nevű oszlop van. Valamint egy oszlopban helyezkedik el a tranzakció összege, illetve a fájlban megtalálható számlaegyenleg oszlop is. Továbbá pozitívum hogy a tranzakciós sorok között nincs üres sor.

Az előbb felsorolt pozitív tulajdonságok sajnos nem mondhatóak el az FHB banktól letölthető fájlról.



14. ábra: FHB banktól letölthető tranzakciós fájl.

Amit elsőként észre lehet venni a fenti ábrát nézve, hogy jóval, több oszloppal rendelkezik, mint CIB-es fájl. Több oszlop tartalma is megfelelhet tranzakció leírásnak, például Tranzakció típusa, Közlemény, Leírás.

Látható hogy a tranzakció összege két oszlopra van szétszedve (Jóváírás, Terhelés). Valamint az egyenleg oszlopban csak bizonyos sorokban szerepel érték, ami elég összezavaró. Valamint ami a beolvasást jelenti, nagy problémát jelent, hogy rengeteg üres cella szerepel bizonyos sorokban.

### Felhasználó által deklarált importálás

**Koncepció**

Egy olyan felület létrehozása, ahol a felhasználónak lehetősége van beírnia az importált fájlhoz tartozó oszlopszámokat és egyéb adatokat, grafikus elemeket (szövegdoboz, legördülő lista) használva. Amik segítségével az alkalmazás kiolvassa számára a tranzakciókat az ő általa megadott értékek szerint.

Természetesen a felhasználónak nem kell minding újra és újra beírnia azon importálandó fájlok adatait, amit már egyszer megtett. Ezt megoldva, a rendszer adatbázisában tárolni fogom a beírt oszlopszámokat és egyéb adatokat. Ami mindig frissülni fog új importálás esetén.

Minden Felhasználó által deklarált importálás esetén az összes adatbázisban eltárolt fájlokhoz tartozó adat betöltődik egy legördülő listába. A legördülő lista aktuálisan kiválasztott elemét változtatva, a felhasználó által kitöltendő adatok is változnak.

Alapértelmezetten a legjobban illeszkedő értékek szerint vannak kitöltve a felhasználó által bekért adatok értékei (szövegdoboz, legördülő lista).

Ha a felhasználó kitöltötte ezen adatokat egy gombra kattintva jelzést ad az alkalmazásnak, hogy elkezdheti kiolvasni a tranzakciókat az általa beírt által.

Mivel a felhasználó egyszerre több fájlt is tud importálni a jobb felső sarokban láthatja majd az aktuális fájl nevét.

#### Felület

**Implementáció**

Fontosnak tartottam, hogy ha a felhasználónak egy oszlopot kell azonosítania, írhassa be azt egész számként, illetve karakter formájában is (Excelben az oszlopok karakterekkel vannak jelölve). Természetesen, ha a felhasználó karakter formátumban írja be, azt kódban át kell alakítani egy számra, amit a következőképpen oldottam meg:

public static int ExcelColumnNameToNumber(string columnName){

columnName = columnName.ToUpper();

int sum = 0;

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

sum \*= 26;

sum += (columnName[i] - 'A' + 1);

}

return sum;

}

A felhasználónak a következőekben felsorolt adatokat kell kitöltenie minden beolvasandó fájlt importálásánál (ahol több lehetőség van egy legördülő listából kell kiválasztania a számára megfelelő opciót)

* Tranzakciók kezdősora, egyszerű egész szám.
  + Azt a sort jelöli az importált fájlban ahol a tranzakciók adatainak felsorolása kezdődik.
* Számlaszám (az importáló számlaszáma), itt három lehetősége van a felhasználónak:
  + egy cellában helyezkedik. Például B3 ahol a B az oszlopot azonosítja (2. oszlop), 3 pedig a sort azonosítja.
  + a számlaszám oszlopban helyezkedik el (minden tranzakció esetén szerepel a felhasználó számlaszáma).
  + az Excel munkafüzet neve tartalmazza a számlaszámot.
* Tranzakció dátumát tartalmazó oszlop
  + Egy oszlop azonosítóját kell beírnia a felhasználónak.
* Tranzakció összegét tartalmazó oszlop, két lehetősége van a felhasználónak
  + Egy oszlopban tartalmazza a fájl az összeget, egy oszlop szám beírására van lehetőség.
  + Két oszlopban tartalmazza a fájl az összeget (Jóváírás, Terhelés), két oszlop szám beírására van lehetőség.
* Számlaszám egyenlegét tartalmazó oszlop, két lehetősége van a felhasználónak
  + Ha szerepel a fájlban ilyen oszlop, akkor az oszlop azonosítóját írja be.
  + Nem szerepel ilyen oszlop, nem szükséges beírni ezt az adatot.
* Tranzakcióhoz tartozó leírás oszlop
  + Itt fontosnak tartottam azt, hogy a felhasználónak több oszlop beírására is legyen lehetősége. Tehát felvihet akár több oszlop azonosítót is, azokat vesszővel elválasztva (például A,C,E).
* Bank neve
  + Mivel ezt az adat a fájlokban nem szerepel, ezért a felhasználónak saját kézzel kell ezt az adatot beírnia egy szövegdobozba.

#### Eltárolt adatok betöltése

**Implementáció**

A korábban importált fájlok eltárolt adatai következőképpen kerülnek betöltésre:

* Először is a belső adatbázisból lekérésre kerül az importált fájlokhoz tartozó összes bejegyzés.
* A felhasználó által importált fájl megnyitásra kerül a program által, majd egyesével megnézi minden eltárolt rekordra az illeszkedés mértékét. Például:
  + a számlaszám pozícióján elhelyezkedő adat illeszkedik-e egy bankszámlaszám formátumára (huszonnégy vagy tizenhat szám)
  + Dátum oszlop celláiban tényleg dátumok szerepelnek?
  + Összeg oszlop celláiban szám formátumú értékek szerepel?
  + Rendelkezik-e egyenleg oszloppal, ha igen egész számok szerepelnek a celláiban?
* Egy egyszerű maximumkeresést alkalmazva az a tárolt rekord fog betöltésre kerülni ahol a legtöbb egyezés volt.
* Ha nulla egyezés van az importált fájl és az eltárolt adatok között, nem töltünk be alapértelmezett adatokat a felhasználó számára.

### Automatikus importálás

**Koncepció**

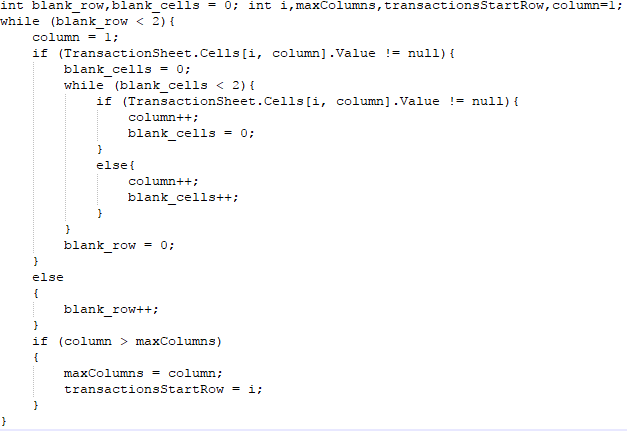
Mivel minden szoftverben fontos az automatizált működés, ezért fontosnak tartottam egy olyan importálás implementálását ahol az alkalmazás algoritmusokat használva megpróbálja beazonosítani a tranzakciók kiolvasásához szükséges oszlopokat illetve cellákat. Majd ezen adatokat felhasználva, az alkalmazás kiolvassa az importált fájlban lévő adatokat.

**Implementáció**

Mint ahogy korábban bemutattam a fájlok formátuma, struktúrája nagyon különbözik. Az egyetlen közös dolog az, hogy minden fájl ugyan azon adatokat hivatott megjeleníteni (tranzakciókhoz tartozó ár, dátum, bankszámlaszám, leírás, egyenleg). Tehát arra döntésre jutottam, hogy reguláris kifejezéseket fogok használni a bizonyos oszlopok illetve cellák beazonosításához.

Ehhez elsősorban meg kell adnom az algoritmus számára, hogy melyik sorban kezdje az oszlopok vizsgálatát. Tehát meg kell találni azt a sort, ahol a tranzakciók adatainak felsorolása kezdődik. Amit a következőképpen oldottam meg:

* Mivel a tranzakciók abban a sorban kezdőnek ahol a fájlban a legtöbb oszloppal rendelkezik egy sor, így nem volt más dolgom, mint végig menni a fájlon és meg találni ezt a sort, ami egy egyszerű minimum keresés segítségével megoldható.



15. ábra: a leghosszabb sor megtalálása

* + Ahol a TransactionSheet objektum jelöli az aktuális Excel munkafüzetet, Cells[i,j] hivatkozása pedig a munkafüzet egy cellája, ahol i a cella sorát azonosítja, j pedig az oszlopot.

Mivel már tudjuk melyik sorban kell elkezdenünk az adatok vizsgálatát, elkezdjük a reguláris kifejezések használatát.

* A bankszámlaszámra beazonosítására többek között a következő reguláris kifejezések felelnek:
  + \d{8}-\d{8}, kétszer nyolc darab decimális számot (0-9) egymástól kötőjellel elválasztva.
  + \d{8}-\d{8}-\d{8}, háromszor nyolc darab decimális számot (0-9) egymástól kötőjellel választva.
  + \d{16} tizenhat darab decimális számot keresünk.
  + \d{24] huszonnégy darab decimális számot keresünk
* Az összeg oszlop beazonosítására többek között a következő reguláris kifejezések felelnek:
  + Összeg, Tranzakció összege, ha illeszkedik a szöveg, mint regulás kifejezés megvizsgálásra kerül az oszlop celláinak értéke is. Hogy egész számot tartalmaznak-e.
  + Terhelés, Jóváírás, ez esetben két oszlop felel a tranzakció összegének azonosítására. Ha illeszkedik a szöveg mint regulás kifejezés megvizsgálásra kerül az oszlop celláinak értéke is. Hogy egész számot tartalmaznak-e.
* A dátum oszlop beazonosítására többek között a következő reguláris kifejezések felelnek:
  + \d{4}.\d{2}.\d{2}
    - \d{4}. jelenti, hogy négy darab decimális (0-9) számot várunk utána egy ponttal (ez azonosítja az évet)
    - \d{2}.\d{2} jelenti, hogy két darab decimális számot várunk (első a hónap, második a nap)
    - például 2018.04.20
  + ^\d{4}.\s\d{2}.\s\d{2}
    - \d{4}\s jelenti, hogy négy darab decimális számot (0-9) várunk White Space karakterekkel elválasztva például szóköz, tabulátor stb. (ez azonosítja az évet)
    - \d{2} jelenti, hogy két darab decimális számot (0-9) várunk (az kifejezés a hónapot, második a napot)
    - például 2018 .04 .20
* Leírás oszlop keresése alatt több oszlop száma is eltárolásra kerülhet. többek között a következő reguláris kifejezések felelnek a Leírás oszlop beazonosítására:
  + ^Közlemény$
  + ^Típus$
  + ^Leírás$
  + ^ jelzi hogy az érték elejétől nézzük az illeszkedést
  + $ jelzi a vizsgált érték végét
* Egyenleg oszlop keresése alatt felmerülhet, hogy nem sikerül beazonosítani, mivel az importált fájl nem rendelkezik vele. Többek között a következő reguláris kifejezések felelnek az Egyenleg oszlop beazonosítására:
  + Egyenleg$"
  + ^Számlaegyenleg$
  + ^ jelzi hogy az érték elejétől nézzük az illeszkedést
  + $ jelzi a vizsgált érték végét

A tranzakciók kiolvasása az importált fájlban abban csak az esetben kerülnek kiolvasásra hogyha sikerült beazonosítani legalább a számlaszámhoz, dátumhoz, összeghez tartozó adatokat.

## Tőzsdei tranzakciók importálása

### Felhasználó által deklarált importálás

**Koncepció**

Hasonlóan a Banki tranzakciók esetében itt is implementálásra került egy grafikus felület ahol a felhasználó saját kézzel irányíthatja a tranzakciók beolvasásának menetét.

Természetesen itt is betöltésre kerülnek az adatbázisban már eltárolt, importált fájlokhoz tartozó adatok. Illetve a legjobban egyező rekord értékei az alapértelmezettek grafikus felületen elhelyezkedő szövegdobozokban illetve legördülő listákban.

**Implementáció**

Természetesen itt is van lehetősége a felhasználónak betű illetve szám formátumban megadni az oszlopok azonosítóját.

A felhasználó által kitöltendő adatok a következők:

(Ahol több lehetősége van a felhasználónak egy legördülő listából kell kiválasztania az aktuálisan megfelelő opciót)

* Tranzakciók kezdősora, egyszerű egész szám.
  + Azt a sort jelöli az importált fájlban ahol a tranzakciók adatainak felsorolása kezdődik.
* Termék név, oszlopazonosító
  + Azt az oszlopot jelöli ahol szerepel a cég neve, aki kibocsájtja a felhasználó által vásárolt/eladott részvényt.
* Összeg, oszlopazonosító
  + Azt az oszlopot jelöli ahol szerepel az eladott/vásárolt részvény árfolyama.
* Mennyiség, oszlopazonosító
  + Azt az oszlopot jelöli ahol szerepel a tranzakcióhoz tartozó mennyiség.
* Dátum, oszlopazonosító
  + Azt az oszlopot jelöli ahol szerepel a tranzakcióhoz tartozó dátum.
* Típus, oszlopazonosító
  + Azt az oszlopot jelöli ahol szerepel a tranzakció típusa (Adás/Vétel).

### Automatikus importálás

**Koncepció**

Mint a banki tranzakciók beolvasásánál, itt is fontosnak tartottam egy olyan algoritmusimplementálást, ami automatikusan **megpróbálja** felismerni a tranzakció adatainak beolvasásához szükséges oszlopokat.

Alapvetően arra a tényre alapoztam, hogy bár különböznek a tőzsdei programokból letölthető tranzakciós fájlok mind struktúrában és elrendezésben, mégis ugyan azon adatokat tartalmazzák.

Erre a tényre alapozva megpróbáltam olyan reguláris kifejezéseket írni, amik segítségével egységesíthető a kellő oszlopok felismerése.

**Implementáció**

1. A tranzakció dátumának felismerése

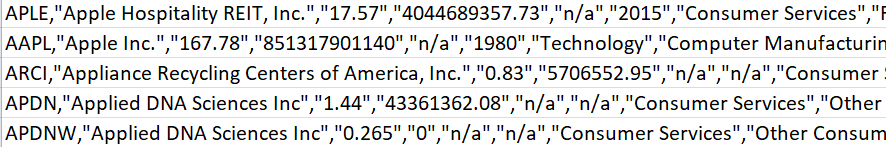
* ^ kifejezés jelenti, hogy a vizsgált elem elejétől nézzük az illeszkedést
* ^\d{4}.\d{2}.\d{2}
  + \d{4}. jelenti, hogy négy darab decimális számot várunk utána egy ponttal (ez azonosítja az évet)
  + \d{2} jelenti, hogy két darab decimális számot várunk (az kifejezés a hónapot, második a napot)
  + például 2018.04.20
* ^\d{4}.\s\d{2}.\s\d{2}
  + \d{4}\s jelenti, hogy négy darab decimális számot várunk White Space karaktert például szóköz, tabulátor stb. (ez azonosítja az évet)
  + \d{2} jelenti, hogy két darab decimális számot várunk (az kifejezés a hónapot, második a napot)
  + például 2018 .04 .20

1. A tranzakcióhoz tartozó részvény nevének felismerése

* Minden tőzsdén jelenlévő cég nevében szerepel a cég típusa. Például:
  + Nyrt (Nyílt részvény társasság)
  + Inc (Incorporation )
  + Company,
  + AG (Aktiengesellschaft)
* Így nem volt más dolgom, mint olyan reguláris kifejezéseket írni, amely az ilyen végződésű cellákat ismeri fel.

1. A tranzakcióhoz tartozó eladási vagy vételi ár felismerése

* Itt viszont már komplikációk merültek fel, mivel nem alkalmazhattam olyan beazonosítást, hogy megvizsgálom egy cella tartalmát, és ha az egy szám, akkor az lesz a részvényhez tartozó ár, mivel semmi nem garantálja, hogy ha egy cella értéke valóban egy szám, akkor az a szám a részvény vételi vagy eladási árát takarja. A problémát megoldva, többlépéses vizsgálatot alkalmaztam.
  1. Első lépés, sikeresen beazonosítottuk a dátum és részvény nevéhez tartozó oszlopokat.
  2. Második lépés, minden különböző céghez elmentjük a legrégebbi tranzakció dátumát.
  3. Harmadik lépés, az IEX API segítségével le lehet kérni tőzsdei szimbólumokhoz tartozó árakat, bizonyos idő intervallumra visszamenőleg. Viszont, jelenleg csak a cég nevekkel rendelkezünk. Tehát, meg kell találnunk a cég nevéhez tartozó tőzsdei szimbólumokat. Amit a NASDAQ oldaláról letölthető csv formátumú fájlból teszek meg.



16. ábra: NASDAQ oldaláról letölthető csv fájl tartalma

* A fenti ábráról láthatjuk, hogy az első oszlopban szerepel a tőzsdei szimbólum, a második oszlopban pedig a szimbólumhoz tartozó cég neve.
* A NASDAQ oldaláról letölthető fájl beolvasása után egyszerűen csak futtattam egy keresést az adott cég nevére, és kiolvastam az előtte lévő oszlopban szereplő tőzsdei szimbólumot.
* Mivel gyakorinak találtam azt az eshetőséget, hogy a tőzsdei tranzakciókat tartalmazó fájlban illetve a NASDAQ oldaláról lekért adatok között egy vessző vagy pont eltérés szerepel ezért, egy karakter eltérését megengedtem a fájl és a lekért cégnevek között. Amit Levenshtein távolság függvény implementálásával oldottam meg:
  + Amely vár két sztringet paraméterként, majd visszatér a paramétereként átadott két szöveg közötti eltérések számával.

int levenshteinDist(string s,string t){

int n = s.Length;

int m = t.Length;

int[,] d = new int[n + 1, m + 1];

if (n == 0){

return m;

}

if (m == 0){

return n;

}

for (int i = 0; i <= n; d[i, 0] = i++);

for (int j = 0; j <= m; d[0, j] = j++);

for (int i = 1; i <= n; i++){

for (int j = 1; j <= m; j++){

int cost = (t[j-1] == s[i-1])?0:1;

d[i,j]=Math.Min(

Math.Min(d[i-1,j]+1 d[i,j-1]+1),

d[i-1,j-1]+cost);

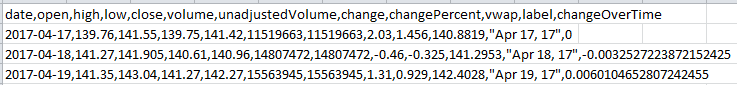
}

}

return d[n, m];

}

* 1. Negyedik lépés, a negyedik lépéshez teljesülnie kell a harmadik lépésnek.
* A második lépés során elmentett legrégebbi dátumokat besoroljuk az alábbi kategóriákba:
  + legfeljebb egy hónapja történt
  + legfeljebb három hónapja történt
  + legfeljebb hat hónapja történt
  + legfeljebb egy éve történt
  + legfeljebb három éve történt
  + legfeljebb öt éve történt
* Erre azért van szükségünk, mivel az IEX API számára ebbe a hat időintervallumba eső dátumok lekérése van lehetőségünk.
* Mivel már rendelkezünk a cég nevéhez tartozó tőzsdei szimbólummal és a dátumot besoroltuk a fent látható intervallumokba. A következő dolgom volt IEX API-t használva lekérni a bizonyos dátumhoz tartozó legmagasabb illetve legalacsonyabb részvényárat.



17. ábra: IEX API-tól lekért, tőzsdei információkat tartalmazó fájl

* A lekért adatok közül szükségünk lesz a ,,High" értékre, ami az adott naphoz tartozó legmagasabb árat jelenti és a ,,Low" értékekre ami az adott naphoz tartozó legalacsonyabb árat jelenti.
* Az először is írtam egy reguláris kifejezést, hogy megtaláljam a dátumokat, mivel nem áll szándékunkban minden elemet megvizsgálni. Majd megvizsgálom, hogy egyezik-e az általunk keresett dátummal (a részvényhez tartozó legrégebbi dátum az importált fájlból). Ha egyezik, akkor lekérjük a mellette jobbra elhelyezkedő elemet, ez lesz a dátumhoz tartozó legmagasabb ár, illetve a tőle néggyel jobbra elhelyezkedő elemet, ami a dátumhoz tartozó legalacsonyabb ár.
  1. Ötödik lépés, megvizsgáljuk, hogy a tranzakciós fájl aktuális sorában, ahonnét a dátumot, illetve a céghez tartozó tőzsdei szimbólumot kiolvastuk, létezik-e olyan cella, aminek az értéke nagyobb vagy egyenlő, mint a kiolvasott legkisebb érték, és kisebb egyenlő, mint a kiolvasott legnagyobb érték.
* Ha létezik ilyen érték, akkor megtaláltuk az árhoz tartozó oszlopot, ha nem jártunk sikerrel, akkor megvizsgáljuk a következő céget ugyan ezzel a módszerrel.

1. A tranzakcióhoz tartozó mennyiség felismerése

* Az oszlop nevére illeszkedő reguláris kifejezés segítségével.
* Pár példa a vizsgált illeszkedésekre:
  + Mennyiség
  + Quantity

1. A tranzakció típusának megtalálása

* Az oszlop nevére illeszkedő reguláris kifejezés segítségével.
* Pár példa a vizsgált illeszkedésekre:
  + Vásárolt, Buy, Bought
  + Eladott, Sell, Sold

1. A megtalált adatok alapján történő kiolvasás

* Ha minden oszlopot sikeresen megtaláltunk, akkor kezdődhet a tranzakciók tényleges kiolvasása az importált fájlból.
* Ha a kellő oszlopok beazonosítása nem sikeres, akkor a felhasználó kap egy értesítést, hogy az importált fájlra alkalmazzon Felhasználó által deklarált importálást.

### Nyereség-veszteség kiszámítás

A nyereség-veszteség kiszámításának három módszerét tudja kiválasztani a felhasználó az általa importált tőzsdei tranzakciókra. Az importálás megkezdése előtt.

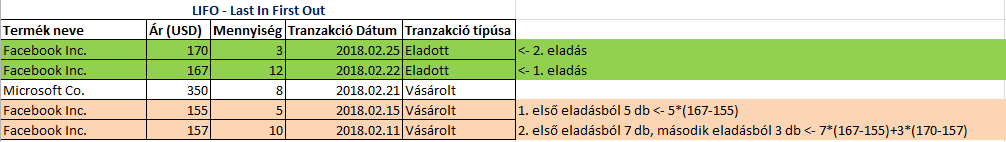
**FIFO (First In-First Out)** azaz amit időrendi sorrendben előbb vásároltunk meg, azt fogjuk először felhasználni tehát eladni. Az alábbi ábrán, egy egyszerű példával szemléltetem a FIFO elv működését.



18. ábra: FIFO elv működése

* Mivel FIFO elvet alkalmazunk, ezért a vásárolt tranzakciókat alulról-felfelé dolgozzuk fel. A fenti ábrából láthatjuk, hogy 2018.01.22.-én vásároltunk tíz darab Apple Inc. részvényt, darabonként 170 dollár értékben, hívjuk el ezt a tranzakciót Első tranzakciónak. Majd 2018.01.25.-én vásároltunk öt darab Apple Inc. részvényt, darabonként 165 dollár áron hívjuk el ezt a tranzakciót Második tranzakciónak, illetve 2018.02.02.-án vásároltunk nyolc darab Coca Cola Co. részvény darabonként 350 dollár áron, mivel ezeket a részvényeket nem adtuk el ezért a későbbiekben nem foglalkozunk vele. Majd láthatjuk, hogy eladtunk tizenkét darab Apple részvényt 2018.02.02.-án méghozzá 180 dollár értékben. Ha FIFO elvet alkalmazunk a nyereség-veszteség kiszámolására, akkor azt a részvényt adtuk el, amit legelőször megvettünk. Mivel FIFO elvet alkalmazunk, ezért az Első tranzakcióból adtunk el tíz darab részvényt, aminek a nyeresége a következő: 10\*(180-170) ahol a 10 jelenti az eladott részvények számát, a 180 az eladási árat és 170 a vételi árat. Még maradt két eladott részvényünk az első eladásból, amit a Második tranzakcióból adtunk el, mivel az Első tranzakcióból eladtuk minden részvényt. Ennek az eladásnak a nyeresége a következő: 2\*(180-165) ahol kettő az eladott részvények száma, 180 a részvények eladási ára, 165 a Második tranzakcióból származó vételi ár.
* Tehát a 12 darab eladott Apple Inc. részvény nyeresége összesen 10\*(180-170)+2\*(180-165) ami egyenlő száz-harminccal.
* Az első eladás nyereség-veszteség kiszámolása után van összesen három darab Apple Inc. részvényünk a Második tranzakcióból. De a második eladás során pont három darabot adunk el. Így a második eladáshoz tartozó nyereségét a következő módon számoljuk ki: 3\*(182-165) ahol három az eladott részvények száma, 182 a részvények eladási ára 165 a részvények vételi ára.
* Tehát második eladáshoz tartozó nyereség egyenlő ötveneggyel.

**LIFO (Last In-First Out)** azaz amit időrendi sorrendben később vásároltunk meg részvényt, azt fogjuk először "felhasználni" tehát eladni. Az alábbi ábrán, egy egyszerű példával szemléltetem a LIFO elv működését is.



19. ábra: LIFO elv működése

* Az ezelőtti példához hasonlóan itt is nevezzük el a vásárolt tranzakciókat. De mivel most LIFO elvet alkalmazunk a vásárolt tranzakciókat fordított időrendi sorrendben fogjuk feldolgozni. Az ábráról leolvashatjuk, hogy 2018.02.21.-én vásároltunk nyolc darab Microsoft Co. részvényt háromszáz-ötven dollár értékben, mivel ezt nem adtuk el, nem foglalkozunk vele a későbbiekben. Láthatjuk 2018.02.15.-én vásároltunk Facebook Inc. részvényt, öt darabot, darabonként száz-ötvenöt dollár áron, nevezzük el Első tranzakciónak. Valamint 2018.02.11.-én is vásároltunk tíz darab Facebook Inc. részvényt, darabonként száz-ötvenhét dollár áron, nevezzük el Második tranzakciónak. Észrevehetjük, hogy 2018.02.22.-én eladtunk tizenkét darab Facebook Inc. részvényt. Mivel tizenkét darab részvényt adtunk el, és az Első tranzakció öt darab részvényt tartalmaz. Ezért az első öt darab eladott részvényhez számolt nyereség a következő: 5\*(167-155) ahol 5 az eladott részvények száma, 167 az eladott részvények ára és 155 a vásárolt részvények ára. Mivel tizenkét darabot adtunk el és eddig csak az első ötre számoltuk ki a nyereséget a Második tranzakcióból kell kiszámolnunk maradék hét eladott részvény nyereségét. Az első eladásból vett hét részvény és a Második tranzakcióból számolt nyereség a következő: 7\*(167-157) ahol hét az eladott részvények száma, 167 az eladott részvények ára, 157 a vásárolt részvények ára.
* Tehát az első eladáshoz tartozó nyerség egyenlő 5\*(167-155)+7\*(167-157) ami egyenlő százharminccal.
* A második eladáshoz tartozó nyereség kiszámolása a következő módon történik: mivel a Második tranzakcióhoz már csak három részvény tartozik, és a második eladás során három részvényt adtunk el így a következő képletet alkalmazhatjuk: 3\*(170-157) ahol három az eladott részvények száma, 170 az eladott részvények ára és 157 a vásárolt részvények ára.
* Tehát a második tartozó nyerség egyenlő harminckilenccel. Illetve rendelkezünk még nyolc darab Microsoft Co. részvénnyel.

**CUSTOM** nyereség-veszteség, Ha a felhasználó a CUSTOM nyereség-veszteség kiszámítási módot választja az alábbi táblázat és a gombok segítésével a saját íze-kedve szerint köthet össze eladott és vásárolt tranzakciókat.

* Az oldalon láthatunk majd egy táblázatot, ami tartalmazza a frissen beimportált tranzakciókat. Az oszlopok a tranzakciókhoz tartozó adatokat jelenítik meg.
* A táblázat felett elhelyezkedik egy legördülő lista, amely az összes különböző cégekhez tartalmazó részvény nevét tartalmazza, amelyeket a szoftver a felhasználó által beimportált fájlból olvas ki.
* A legördülő lista mellet van egy gomb, amin az "Restart this" feliratot
* láthatja a felhasználó. A gomb, megnyomás esetén, a legördülő listából aktuálisan kiválasztott részvényhez tartozó mennyiségek visszakerülnek az eredeti állapotukba. A nem kiválasztott nevekhez tartozó mennyiségek maradnak a jelenlegi állapotukban.
* A "Restart this" gomb mellet az "Restart all" gomb helyezkedik el. A gomb, megnyomás esetén, az összes részvényhez tartozó mennyiség a kezdeti állapotba vált vissza
* A részvényeket tartalmazó táblázat mellet láthatunk majd egy gombot, ami a "Calculate" feliratot viseli. Ez a gomb alapértelmezetten kikapcsolt állapotban van. A gomb engedélyeséhez a felhasználónak ki kell jelölnie a táblázatban egy eladott és egy vásárolt részvényt. Amit az egér kattintása és CONTROL billentyűzet lenyomása segítségével tud megtenni. Erre arra van szükség mivel így a felhasználó tudja kijelölni melyik eladást melyik vásárlással akarja összekötni.
* A "Calculate" gomb alatt megjelenik egy "Export preview" feliratú gomb, amire rákattintva a megjelenik egy új táblázat az importált tranzakciókat tartalmazó táblázat helyén. Az új táblázat tartalmazza a már kiszámolt részvényeket azon esetleges nyereség illetve veszteségét.
* Az "Export preview" gomb alatt helyezkedik a "Export transactions" gomb, amire rákattintva a felhasználó véglegesíti a nyereség-veszteség kiszámolást, és a tranzakciók, azon adataival elmentésre kerülnek a rendszer adatbázisába.

## Tárolt tranzakciók megjelenítése

**Koncepció**

Az alkalmazásba beimportált tranzakciós fájlok eddig különböző formátumú illetve struktúra szerint voltak rendezve. Így, fontosnak tartottam egy egységes táblázat formájában megjeleníteni a rendszerben eltárolt adatokat.

**Implementáció**

* A táblázat adatforrása (cél adat) Egy irányú kötéssel (OneWay binding) van összekötve a programkód egy osztályokat tartalmazó lista objektumával (forrás adat).
* Az oszlopok jelenítik meg a lista objektumainak (forrás adat) adattagjait.
* Ahogy már korábban leírtam, program futása során minden menüből csak egy példány fog létrejönni. Viszont, ha a táblázat elemeihez új tranzakciókat adunk hozzá (importálás), akkor tudatni kell a grafikus felületet, hogy a táblázat forrás adatának összetétele megváltozott. Így frissüljön a cél adat is. Ami a következőképpen lehetséges:
  + Először is, a forrás adat osztályában implementálni kell az értesítésért felelő interfészt (INotifyPropertyChanged), ami megköveteli egy változó létrehozását.

event PropertyChangedEventHandler

PropertyChanged;

* + Majd létre kell hoznunk egy függvényt, ami értesítést küld a grafikus felület számára a következő módon

void OnPropertyChanged(string s)

{

PropertyChanged(this

,new ertentArgs(s));

}

* + Természetesen az adatösszekötés során ezt XAML szinten is tudatni kell, méghozzá hogy az UpdateSoruceTrigger tulajdonság beállításával.

UpdateSourceTrigger=PropertyChanged

* Mivel a táblázatban lévő cellák változó hosszúságú adatokat tartalmazhatnak, fontosnak tartottam, hogy ha egy cella tartalma hosszabb, mint amit az általa maximálisan megjeleníthető adatmennyiség, akkor a felhasználónak legyen lehetősége rávinni kurzort a cellára, hogy egy szövegbuborék segítségével kiolvashassa annak teljes taralmát. Ami a ToolTip tulajdonság állításával érhető el XAML szinten.

<Setter Property="ToolTip" Value="{Binding

cellValue}" />

* + Ahol cellValue jelöli a cellához tartozó teljes értéket.

### Banki adatok

Ahogy már korábban említettem, fennállhat az az eshetőség, hogy egy banki import fájl nem rendelkezik egyenleg oszloppal vagy megfelelő leírás oszloppal. Ebben az esetben a táblázat aktuális sorában szereplő cella üres lesz.

A banki adatok megjelenítéséhez egy hatoszlopú táblázat jelenítek meg a felhasználó számára. Az oszlopok az alábbi adatokat jelenítik meg:

* Tranzakció dátuma
* Tranzakció leírása
* Tranzakció összege
* Importált fájlhoz tartozó számlaszám
* Importált fájlhoz tartozó bank neve
* Importálás dátuma

### Tőzsde adatok

A tőzsdei adatok megjelenítéséhez egy hatoszlopú táblázat jelenítek meg a felhasználó számára. Az oszlopok az alábbi adatokat jelenítik meg:

* Tranzakció dátuma
* Termék neve (Részvény kibocsájtója)
* Tranzakció összege
* Tranzakció típusa (Eladás/Vásárlás)
* Mennyiség
* Profit
* Importálás dátuma

Ha a felhasználó profitált az egyes részvények eladásából, akkor az aktuális profit cella háttérszíne zöldre vált, ellenkező esetben piros színt vesz fel.

Mivel nem akartam túlzsúfolni a táblázatot, ezért az eladott tranzakciók esetén a felhasználó úgy tudja megnézni, hogy milyen nyereség-veszteség módszer által lett kiszámolva a megjelenített profit, hogy ráviszi a kurzort az adott cellára. Ahol megjelenik egy szövegbuborék, ami tartalmazza a kiszámítási módszer nevét.

## Részvény árfolyamok megjelenítése

A részvényekhez tartozó adatokat a felhasználó számára az IEX API segítségével töltöm le, majd lementem a belső adatbázisba. Az eltárolt adatok a következők:

* dátum (ÉÉÉÉ-HH-NN) formátumban
* lekért tőzsdei szimbólum
* dátumhoz tartozó legmagasabb, legalacsonyabb ár.
* dátumhoz tartozó nyitó, záró ár.

### Internetről történő lekérés

**Koncepció**

A lekért adatokat a Live-Charts könyvtárat használva megjelenítem meg egy grafikon formájában. Ahol a grafikon X tengelye jelzi a dátumokat, az Y tengely pedig az aktuális árakat. Így a metszéspontok tartalmazzák az egyes naphoz tartozó árakat.

A részvényárfolyamok lekérésénél fontosnak tartottam, hogy a felhasználó láthassa a lekért tőzsdei szimbólumhoz tartozó legmagasabb és legalacsonyabb árakat is. A gráfon megjelenített két árhoz tartozó statisztika különböző színekkel jelenik meg. Ahol értelemszerűen az alul elhelyezkedő adatcsík jelzi az aznap legalacsonyabb, míg a felső a legmagasabb áron eladott részvény árat.

A felhasználónak lehetősége van beírnia az általa lekérni kívánt tőzsdei szimbólumot (például TSLA, ami a Tesla Inc.-hez tartozó tőzsdei szimbólum). Valamint kiválaszthatja, hogy az aktuális naptól számítva milyen időintervallumra visszamenőleg szeretné megjeleníteni az adatokat. Majd egy gomb megnyomásával jelzést ad az alkalmazás számára az adatok letöltésére és megjelenítésére.

**Implementáció**

Az adatok lekérése a következő módon kerül sor:

* a lekért adatokat ,,felszeletelem” a sztring osztály split metódusával.
* A felszeletelt adatok minden eleme közül a legmagasabb illetve legalacsonyabbat beleteszem két különböző listába.
* A Live-Charts könyvtárat használva megjelenítem külön-külön a listák tartalmát, a következő módon:

SeriesCollection Series = new SeriesCollection{

new LineSeries{

Title = "Highest price: ",

Values = highestValues

},

new LineSeries{

Title = "Lowest price: ",

Values = lowestValues

}

};

* + Ahol a SeriesCollection a Live-Charts által biztosított osztály.
  + highestValues tartalmazza a dátumhoz tartozó legmagasabb árat
  + lowestValues tartalmazza a dátumhoz tartozó legalacsonyabb árat
* Az adatok megjelenítésével párhuzamosan fut egy másik szál, ami a lekért adatok adatbázisba történő elmentéséért felel.

### Adatbázisban eltárolt adatok

**Koncepció**

A felhasználónak lehetősége van az adatbázisban eltárolt tőzsdei adatok megjelenítésére egy táblázat formájában, ahol a táblázat oszlopai az adott részvényhez tatozó eltárolt tulajdonságokat jelenítik meg. A megjelenített adatok alapértelmezetten dátum szerint csökkenő sorrendben vannak rendezve.

A felhasználó számára egy legördülő lista tartalmazza az adatbázisban eltárolt tőzsdei szimbólumokat. A legördülő lista kiválasztott elemének változtatásával lehet módosítani a táblázat tartalmát.

**Implementáció:**

* Először is beolvasom az adatbázisban eltárolt adatokat, feltéve ha a tőzsdei szimbólum egyezik a legördülő lista aktuálisan kiválasztott elemével.
* A beolvasott elemek dátumai szöveges formában vannak eltárolva, tehát dátumra kell átkonvertálni, mivel a sztring mint osztály, nem rendelkezik nagyobb, kisebb operátorral.
* Az átalakítást követően, a dátumok alkotta listára gyorsrendezés algoritmust alkalmaztam.
* Végül, a táblázat elemeit egyenlővé tettem a rendezett listával, természetesen a táblázat elemei XAML szinten össze vannak kötve a programkódban levő listával (binding).

# Összefoglalás

Munkám során kidolgoztam módszereket, amik segítségével lehetővé válik a különböző formátumú és struktúrájú banki, illetve tőzsdei tranzakciós export fájlok beolvasása. Ezen beolvasott adatok elmentésre kerülnek a szoftver adatbázisában, valamint egy egységes formátumú táblázatban megjeleníthetőek.

Mindez azért fontos, mivel mind a banki és tőzsdei tranzakció történetet tartalmazó fájlok formátuma nagyon mértékben különbözik, mind a megjelenített adatok, illetve elrendezés tekintetében. Ez okból könnyen előfordulhat, hogy bizonyos felhasználók nem szánnak időt a pénzügyi adataik nyomon követésére. Különösképp abban az esetben ha az adott felhasználó több banknál is rendelkeznek fiókkal vagy tőzsdei programban is kereskedik részvényekkel

Viszont az általam tervezett és implementált szoftver használatával a felhasználóknak lehetősége nyílik pénzügyi nyilvántartásuk gyors elkészítésére és nyomon követésére egyaránt. Valamit létrehoztam olyan funkciókat, amik segítséget nyújthat a felhasználók számára portfóliójuk javításában.

Nyereség-veszteség kiszámítása három különböző módon.

Adott tőzsdei szimbólumhoz tartozó, felhasználó által választott idő intervallumig visszamenőleg, a részvényárfolyamok nyitó, záró, legmagasabb, legalacsonyabb árának megjelenítésének lehetősége grafikon, illetve táblázat formájában.

Sajnos, sok bank elérhető Magyarországon és nem mindegyik export fájlhoz volt lehetőségem hozzájutni mivel a felhasználók féltik kiadni egy külső személynek a pénzügyi nyilvántartásukat, ami érthető. Valamint egyes bankok nem szolgáltatnak Excel, illetve csv kiterjesztésű fájlokat, hanem a PDF formátumot részesítik előnyben. Illetve a tőzsdei programok nagy része fizetős és csak nagyon kevés rendelkezik demó számlával azon felhasználók számára akik szeretnék letesztelni a szoftvert.