SZAKDOLGOZAT

Tóth Patrik Csaba

2018

Pannon Egyetem

Műszaki Informatikai Kar

Rendszer és Számítástudományi Tanszék

Programtervező informatikus BSc

SZAKDOLGOZAT

Importáló megvalósítása személyes pénzügyi nyilvántartáshoz

Tóth Patrik Csaba

Témavezető: Dr. Heckl István

2018

**SZAKDOLGOZAT TÉMAKIÍRÁS**

**Tóth Patrik Csaba**

Programtervező informatikus BSc szakos hallgató részére

**Importáló tervezése és megvalósítása pénzügyi nyilvántartáshoz**

Témavezető: Dr. Heckl István

**Feladat leírása:**

A személyes vagyon nyilvántartása, a pénzügyi tudatosság azzal a lépéssel kezdődik, hogy valaki rendszeresen feljegyzi a bevételeit és a kiadásait. Ezzel a személyes nyilvántartással az illető nyomon tudja követni vagyona változását. Habár a bankunktól letudjuk kérni a számla forgalmunkat bizonyos időre visszamenőleg, e kimutatások felépítése bankonként más és más. A cél egy olyan alkalmazás készítése, amely fel tudja dolgozni a különböző bankoktól lekért adatokat (számlaforgalmak, árfolyamadatok, értékpapíradatok), ezáltal a felhasználó számára egy áttekinthető eredményt adni a pénzügyeiről, ezzel segítve a pénzügyi tudatosságának kialakulását.

**A diplomázó feladata:**

1. Téma irodalmának áttekintése, hasonló alkalmazások megismerése.
2. Szoftver specifikációk, képernyő tervek, használati esetek készítése.
3. A számlakezelő rendszer implementálása
4. A megvalósított rendszer tesztelése, továbbfejlesztési irányok meghatározása

Dr. Heckl István Dr. Süle Zoltán

egyetemi docens egyetemi docens

témavezető szakvezető

Nyilatkozat

Alulírott *Tóth Patrik Csaba* diplomázó hallgató, kijelentem, hogy a szakdolgozatot a Pannon Egyetem Rendszer és Számítástudományi tanszékén készítettem programtervező informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a szakdolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2011. október 30.

aláírás

Alulírott *Dr. Heckl István* témavezető kijelentem, hogy a szakdolgozatot *Tóth Patrik Csaba* a Pannon Egyetem Rendszer és Számítástudományi tanszékén készítette mérnök informatikus BSc szak (BSc in Computer Engineering) megszerzése érdekében.

Kijelentem, hogy a szakdolgozat védésre bocsátását engedélyezem.

Veszprém, 2018. május 04.

aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani azon személyek számára, akik segítették munkámat ötleteikkel, egyéb ismereteikkel. hogy a szakdolgozatom elkészülhessen.

Elsősorban témavezetőmnek Dr. Heckl István tanár úrnak, szeretnék köszönetet mondani hasznos tanácsaiért, segítségéért, ami nagymértékben segítette szakdolgozatom elkészülését.

Továbbá azon emberek számára, akik tanácsokkal, segítőkészségükkel, valamint bizalmukkal hozzájárultak, hogy a programom létrejöhessen.

**TARTALMI ÖSSZEFOGLALÓ**

A szakdolgozatom keretében megterveztem és megvalósítottam egy olyan univerzális importálót, amely segíti azon személyeket, akik Magyarországon jelenlévő banknál rendelkeznek fiókkal és a bankok által nyújtott eltérő formátumot áthidalva szeretnék nyomon követni pénzügyeiket.

Az első fejezetekben bemutatom az alapproblémát, valamint e probléma megoldása milyen eredményekkel járhat a felhasználók számára. Szót ejtek pénzügyi intézményekhez felvetett törvényjavaslatról, illetve az változtatás jövőbeli hatásairól. Valamint név szerint bemutatok pár hasonló, habár nem hazánkban működő személyes pénzügyi alkalmazást.

Ezt követően bemutatom a programmal szemben felalított követelményeket. Mind megjelenítés, mind belső működés szempontjából. Majd bemutatásra kerülnek a program megvalósításhoz használt technológiák. Ezután részletesen bemutatom a bizonyos funkciókhoz tatozó koncepciót, illetve az ahhoz tartozó megvalósítást is.

Végül összefoglalom az elért eredményeket, valamint bemutatom az elkészült alkalmazást futás közben készült képernyőképek segítségével. Illetve leírom a továbbfejlesztési irányokat a jövőre tekintve.

**ABSTRACT**

For my theis, I designed and implemented a universal importer which can help users, who has bank accounts from any bank available in Hungary, to track their personal finances, despite the fact that these files, provided from banks are very different in all meanings.

In the first chapters, I present the basic problem and I show other functions that can be implemented after the main problem is solved. I also show what law proposals the government will bring in he near future to change the current situantions of financial institutions, and what kind of impact that will have. Then I demonstrate other personal finance softwares, however they can not be used to import transactions from banks available in Hungary.

After these chapterrs I introduce what requirements I made regardig for the graphical interface and for the inner operations. Also I present the technologies that I used to make the application. After that, I go in depth how i implemented functions and solved specific problems that came up druing the implementation.

Finally, I summarize the results, also i show how the software works by help of pictures that were taken during running time. Also I show ways how the apploication can be improved.

**Tartalomjegyzék**

[Bevezetés 10](#_Toc513023227)

[1.Feladat leírása 12](#_Toc513023228)

[1.1.Blokklánc technológia 13](#_Toc513023229)

[1.2.FinTech Magyarországon 14](#_Toc513023230)

[2.Irodalmi áttekintés 14](#_Toc513023231)

[2.1.Mint.com 15](#_Toc513023232)

[2.2.YNAB (You Need A Budget) 17](#_Toc513023233)

[2.3.Quicken Premier 19](#_Toc513023234)

[2.4.Összehasonlító táblázat 20](#_Toc513023235)

[3.Személyes pénzügyi importáló tervezése 20](#_Toc513023236)

[3.1.Használt technológiák 21](#_Toc513023237)

[3.1.1..NET keretrendszer 21](#_Toc513023238)

[3.1.2.C# programozási nyelv 23](#_Toc513023239)

[3.1.3.Grafikus megjelenítés 23](#_Toc513023240)

[3.1.4.Adattárolás 26](#_Toc513023241)

[3.1.5.Feladatkezelés 26](#_Toc513023242)

[3.1.6.Verziókövetés 27](#_Toc513023243)

[3.1.7.Képernyőtervek 28](#_Toc513023244)

[3.1.8.Grafikonok 28](#_Toc513023245)

[3.1.9.Részvény árfolyamok lekérése 29](#_Toc513023246)

[3.2.Követelmények 29](#_Toc513023247)

[3.2.1.Alapvető követelmények 29](#_Toc513023248)

[3.2.2.Grafikus felület 30](#_Toc513023249)

[3.2.3.Tranzakciók importálása 31](#_Toc513023250)

[3.2.4.Tárolt adatok megtekintése 33](#_Toc513023251)

[3.2.5.Részvényadatok lekérése Internetről 33](#_Toc513023252)

[3.2.6.Adatok tárolása 33](#_Toc513023253)

[4.Személyes pénzügyi importáló szoftver megvalósítása 34](#_Toc513023254)

[4.1.Grafikus felület 34](#_Toc513023255)

[4.2.Bejelentkezés és regisztráció 37](#_Toc513023256)

[4.2.1.Regisztráció 37](#_Toc513023257)

[4.2.2.Bejelentkezés 37](#_Toc513023258)

[4.3.Banki tranzakcióik importálása 38](#_Toc513023259)

[4.3.1.Felhasználó által deklarált importálás 40](#_Toc513023260)

[4.3.2.Automatikus importálás 43](#_Toc513023261)

[4.4.Tőzsdei tranzakciók importálása 46](#_Toc513023262)

[4.4.1.Felhasználó által deklarált importálás 46](#_Toc513023263)

[4.4.2.Automatikus importálás 47](#_Toc513023264)

[4.4.3.Nyereség-veszteség kiszámítás 52](#_Toc513023265)

[4.5. Tárolt tranzakciók megjelenítése 56](#_Toc513023266)

1. [4.5.1. Banki adatok 58](#_Toc513023267)
2. [4.5.2. Tőzsde adatok 58](#_Toc513023268)

[4.6.Részvény árfolyamok megjelenítése 59](#_Toc513023269)

[4.6.1.Internetről történő lekérés 59](#_Toc513023270)

[4.6.2.Adatbázisban eltárolt részvény adatok 60](#_Toc513023271)

[4.7.Adatok tárolása adatbázisban 61](#_Toc513023272)

[4.7.1.Adatbázisba történő írás 62](#_Toc513023273)

[4.7.2.Adatbázisból történő olvasás 62](#_Toc513023274)

[Összefoglalás 63](#_Toc513023275)

[Irodalomjegyzék 64](#_Toc513023276)

[Mellékletek 64](#_Toc513023277)

# 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 adott egy személy 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 megoldására jöttek létre a személyes pénzügyi alkalmazások.

Tehát személyes pénzügyi szoftverek egyszerűbbé teszik a felhasználók számára pénzügyeik nyomon követését és egyéb funkciókkal segítik beosztani, esetleg jobbá tenni azt. Egyéb funkcióra pár példa:

* Ha a banki tranzakciókat tekintjük, számon tarthatjuk mennyi pénzünk van jelenleg a bankszámlánkon, mennyi pénzt sikerült félretennünk 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, azok darabszámaival, illetve a jelenlegi részvényárfolyamokat is. Le tudjuk kérni a bizonyos részvényekhez tartozó jelenlegi, és egy bizonyos intervallumra visszamenő legmagasabb, legalacsonyabb, nyitó és záró árakat. Valamint a felhasználó által importált eladott részvények nyereség-veszteség kiszámolására is van lehetőség.

Ezen 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 bankszámlájukról származó tranzakciókat, egy egységes táblázat formájában. Így egy átfogó képet kapva a személyes pénzügyeikről.

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.

Az első fejezetben a feladat rövid leírása található, leírva a fő problémát. Valamint bemutatom a közelmúltban benyújtott, pénzügyi intézményeket célzó parlamenti törvényjavaslatot, annak esetleges következményeivel együtt.

A második fejezetben bemutatásra kerülni pár személyes pénzügyi alkalmazás, azok funkcióival, pozitív illetve negatív tulajdonságukkal egyaránt. Majd a bemutatott alkalmazások összehasonlításra kerülnek egy táblázat segítségével.

A harmadik fejezetben felsorolásra kerülnek a szoftver megvalósításához használt technológiák, azok rövid ismertetésével. Valamint a megvalósítandó funkciókhoz kötött funkcionális és nem funkcionális követelmények.

A negyedik fejezetben bemutatom a szoftver funkcióhoz tartozó koncepciókat , illetve az implementálás folyamatát. Illetve leírásra kerülnek a megvalósítás közben fellépő problémákra hozott megoldások is.

Az ötödik fejezetben olvasható az összefoglalás, ami tartalmazza az elért eredményeket, majd felsorolásra kerülnek fejlesztési irányok az elkészült szoftver 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 az eltárolt tranzakciókat egy táblázat formájában megjeleníteni. Valamint az eladott tőzsdei tranzakciók esetén megjeleníteni a nyereség-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 személyes pénzügyi alkalmazást, mivel amit az egyik változat tud, azt nem biztos, hogy a másik is.

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 azt az alkalmazást és a közös kasszájukat követik nyomon. Tegyük fel hogy, a család minden tagja külön banknál rendelkezik számlával. Így egy ilyen szoftvernek képesnek kell lennie több bank adatait is feldolgozni majd ezeket bankszámlák segítségével elkülöníteni. Ha egy alkalmazás nem rendelkezik e funkciókkal, nem lesz túl népszerű a felhasználók körében.

Egy pénzügyi szoftvert népszerűvé tehet az általa nyújtott funkciók sokasága, például hogy nem csak banki számlakivonatok, hanem tőzsdei tranzakciók feldolgozására is használható. Valamint részvényárfolyamok lekérésére is alkalmas. 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 bemutatásra kerül három személyes pénzügyi alkalmazás azok pozitív illetve negatív tulajdonságaival.

## 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 bankszámlától 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 tippeket a felhasználó számára. Az alkalmazás rendelkezik egy pontrendszerrel, ahova minden felhasználó összesített értékelése bekerül (spórolás mértéke, 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, bankszámlák egyenlegét. Továbbá láthatjuk az esedékes számlabefizetések dátumát (négy hétre előretekintve) 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, 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.

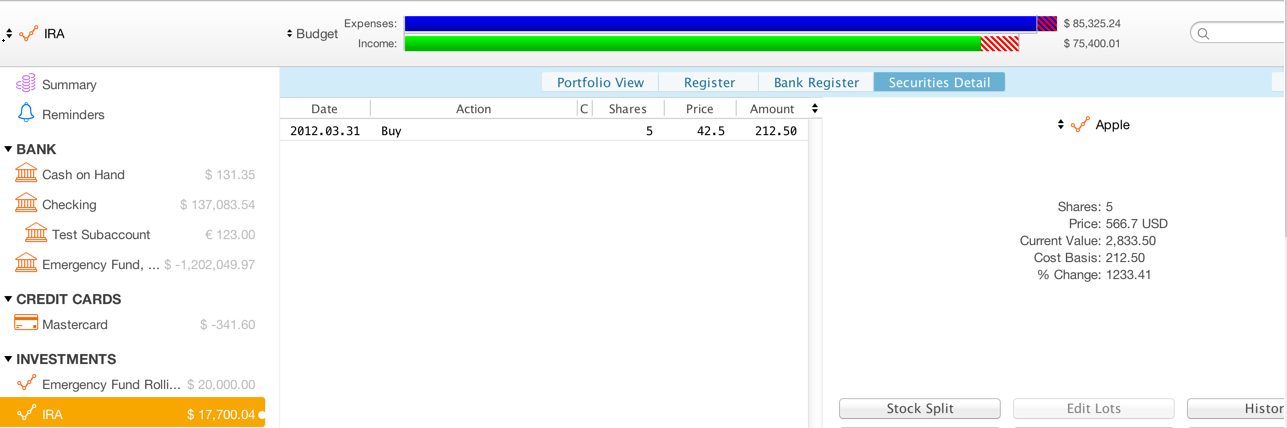
**Negatív tulajdonságok**

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

## 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, darabonként 42,5 dollárért. Majd az oldal jobb szélén láthatjuk, hogy az öt darab részvény jelenleg 1233,41%-al többet ér, mint mikor megvásárolta.

**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

Az előbbiekben bemutatott három alkalmazás alapján összeállítottam egy táblázatot, ami szemlélteti az alkalmazások funkcióit a többi alkalmazással szemben.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Alkalmazás neve | Univerzális importálás | Telefonos változat | Nyomtatás funkció | Fizetős funkció | Befektetések kezelése |
| Mint.com | + | + | + | pénzügyi tanácsok | + |
| YNAB | + | + | - | 34 napig ingyenes használata | - |
| 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 hatékony 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 könyvtárral rendelkezik.
* Számtalan oktató illetve ismeretterjesztő tartalom található az interneten.

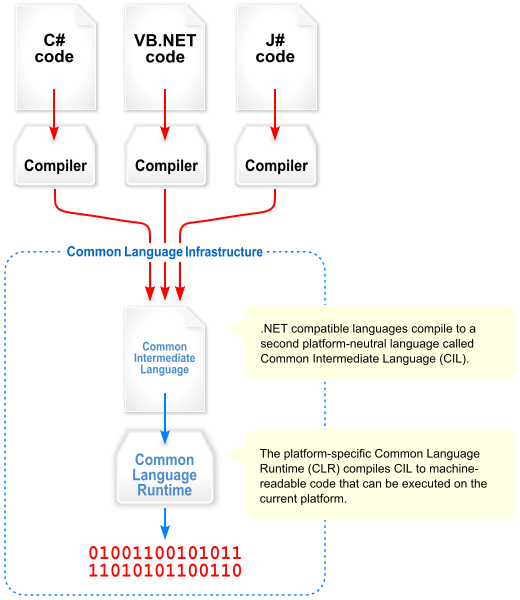
Valamint az sem volt az utolsó szempont hogy egy eddig ismeretlen programozási nyelvet tanuljak és sajátítsak el a szakdolgozatom megírása által.

## Használt technológiák

Ebben a fejezetem bemutatom a szoftver elkészítése során használt technológiákat, azok rövid ismertetésével.

### .NET keretrendszer

A Microsoft .NET rendszere többféle programozási nyelven is programozható. Azzal a feltétellel, hogy az adott programozási nyelvhez van olyan fordítóprogram, amely alkalmazza a Common Language Specification elveit, 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. Ez a funkció csak abban az esetben valósítható meg, ha az adott nyelvek megegyeznek bizonyos alapvető elvekben. Például 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. A szintaxisa leginkább a Javára és a C++-ra emlékeztet, bár sok olyan eleme van, ami egyikre sem jellemző. 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++. 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 eredmény.

### Grafikus megjelenítés

A fő különbség a két felület között hogy a WinForm (Windows Form Designer ) csak egy felületet biztosít az alapvető képernyőelemek fölé (például egy Szövegdoboz). Viszont a WPF-et (Windows Presentation Foundation) 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 azt akarjuk megoldani WPF felületen, hogy egy gomb tartalmazzon egy képet és egy szövegdobozt nincs más dolgunk, mint a gomb tartalmába beilleszteni ezeket az elemeket.

#### 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 azon 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 jelképezi a grafikus felületlen megjelenített adatot, míg a forrás adat a programkódban lévő objektumot. A cél és a forrás adat összekötése Databinding használatával van megvalósítva.

* Egyszeri: (Az ábrán nincs illusztrálva) E kötés esetén, a céladat egyszer megváltozhat, viszont arról a forrás nem fog tudomást szerezni.
* Egyszeri (Az ábrán OneWayToSource): Az Egyszeri kötés fordítottja, tehát akkor frissíti egyszer a forrásértéket, mikor a célérték megváltozik.
* Egy irányú: (Az ábrán OneWay) Az egy irányú összerendelés a célérték automatikus megváltoztatását eredményezi, abban az esetben, ha megváltoztatjuk a forrásadatot, viszont a célobjektum tulajdonságainak módosítása nem hat vissza a forrásértékre.
* Kétirányú: (Az ábrán TwoWay) a Kétirányú összerendelés megváltoztatja a forrásadatot vagy a céladatot a másik frissítése esetén. Tehát ha a programkódban 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 lehetőségünk van elválasztani a megjelenítésért felelő kódot a programkódtól.
* 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.

Összességében az XAML és a Databinding adta lehetőségeket tekintve a WPF grafikus felületen való fejlesztést választottam.

### Adattárolás

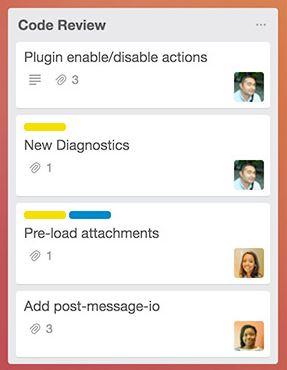
Mivel az importált tranzakciók és a rendszer működéséhez szükséges egyéb adatok egy SQL adatbázisban kerülnek eltárolása. Fontosnak tartottam, hogy a szoftver használói egyszerűen és gyorsan tudjanak kapcsolódni az adatbázishoz. Ezen feltételek figyelembe véve a döntésem az SQLite adatbázisra esett.

Az SQLite motor egy olyan adatbázis-kezelő rendszer, amely a program egy részét képezi méghozzá egy hozzálinkelt programkönyvtár lévén. A kliens-szerver architektúrájú adatbázis-kezelő rendszerekkel ellentétben, amik a programtól függetlenül működnek. A SQLite adatbázis használata esetén a program függvényhívások által férhet hozzá az adatbázis által nyújtott funkciókhoz. Fontos megjegyezni, hogy ez csökkenti az adatbázis eléréséhez szükséges időt, mivel a függvényhívások gyorsabbak, mint a folyamatok közötti kommunikáció.

Mivel az adatbázis a program egy részét képezi, az MSSQL adatbázis-kezelő rendszerrel ellentétben az SQLite adatbázis eléréséhez nem kell egyéb szolgáltatásokat telepítenünk a számítógépre, amelyen az adott programot futtatjuk.

### 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.

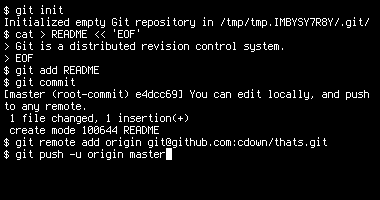
Az egyes kártyák tartalmazhatnak ellenőrző listákat, képeket, csatolmányokat, határidő dátumokat, színes címkéket és vitafelvételeket másoktól, akik osztoznak a projecten. Annyi sajátkészítésű táblát csinálhatunk, amennyit csak szeretnénk. Az egyik például "Otthoni teendők" a másik "Szakdolgozat" és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

### Verziókövetés

Minden szoftverfejlesztési projekt jelentős részét képezi a verziókövetés. Ez nem volt másképp az szakdolgozatomnál sem, ahol a verziókövetésre a Git adta lehetőségeket használtam.

A Git röviden egy nyílt forráskódú verziókezelő rendszer, számítógépes fájlok módosításainak nyomon követéséhez. 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 készülő rendszer képernyőterveinek elkészítésére a Pencil Project nevű mockup-ot használtam. Ami lehetővé teszi, hogy gyorsan és egyszerűen hozzunk létre szórakoztató és interaktív grafikus felületet, maketteket. Az alkalmazás rengeteg beépített elemmel valamint egy egyszerű behúzós felülettel rendelkezik és nagyon könnyű kezelni.

A Pencil Project természetesen engedi, hogy saját képeinket is beimportáljuk a készülő felületre. Létezik asztali alkalmazás formájában, de használható böngészőn keresztül is. Habár az asztali verzión több beépített képernyő elem közül választhatunk.

### Grafikonok

Mivel a WPF grafikus felülete nem rendelkezik alapértelmezett grafikon elemmel és 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, 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 nyilvános 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 teszi ezen adatok egyszerű és gyors lekérését.

Ezen feltételeket figyelembe véve, az IEX alkalmazást választottam. Ami lehetővé teszi a részvényadatok lekérését egy csv fájl formájában. 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 számolódik.

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 tervezési fázis elején a következő alapvető funkcionális illetve nem funkcionális követelményeket állítottam fel a programmal szemben.

### Alapvető követelmények

* Az alkalmazás rendelkezzen egy felhasználói adatbázissal, ahonnét a bejelentkezés során ellenőrzésre kerülnek a beírt adatok. Illetve a felhasználónak legyen lehetősége regisztrálni.
* Tranzakciók importálása esetén a felhasználónak jelenjen meg egy 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 a tranzakció importálás folyamata alatt a 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 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-objektumból a program futása során csak egy példány jöjjön létre. Amit Singleton tervezési minta segítségével oldok meg.

### Grafikus felület

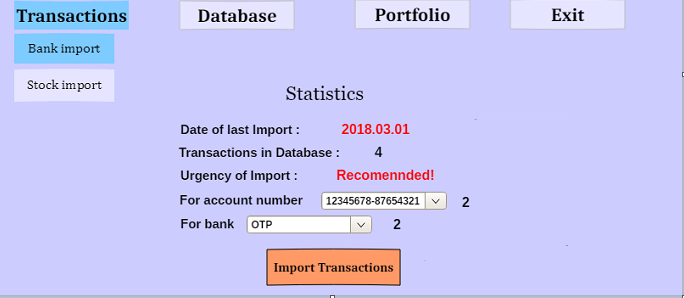
Kezdetben a grafikus felület megtervezésére fókuszáltam, mivel az, az alkalmazás egyik legmeghatározóbb része. Az alábbi követelményeket állítottam fel a grafikus felülettel szemben.

A grafikus felületen bekövetkezett eseményeket két féle módon lehet lekezelni:

* Databinding
* EventHandler

A Databinding-ot részesítem előnyben az EventHandler-el szemben. Mivel ha EventHandler-t 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 logikáért felelő kód, és a megjelenítésért felelő kóddal.

* A grafikus felületen szerepeljenek menük, amik tartalmazzák az adott funkcióhoz tartozó almenüket.
* 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 szövegdobozt, mint értesítés.



9. ábra: korai képernyőterv a banki tranzakciók importálása menüről.

### Tranzakciók importálása

Mivel a tranzakciók importálása az alkalmazás fő funkciója 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 használ, 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

Ha a felhasználó sikeresen importált tranzakciókat, akkor e tranzakciók legyenek megtekinthetőek egy egységesített táblázat formájában, mind banki és tőzsdei tranzakciók esetén.

* A tárolt adatok megjelenítése 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 több adat szerepel a táblázatban, mint annak magassága, akkor a táblázat jobb oldalán jelenjen meg egy görgető csúszka, amit mozgatva a felhasználó láthatja az összes táblázatban szereplő adatot.
* 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 grafikonon, 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.

* Az alkalmazás egy a program részét képező SQLite adatbázisba tárolja el a rendszer működéséhez szükséges adatokat.
* Külön tábla kell a felhasználói adatok, a tranzakciókhoz tartozó információk, Illetve az internetről lekért részvényadatok tárolásához is.
* A felhasználókhoz tartozó jelszavak biztonsági okokból titkosított formátumban legyenek eltárolva.

# Személyes pénzügyi importáló szoftver megvalósítása

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

## Grafikus felület

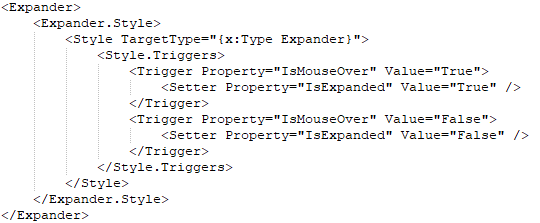
Mint ahogy a tervezés fejezetében leírtam, kezdetben a grafikus felület megvalósítására fókuszáltam.

**Koncepció**

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

**Implementáció**

Az implementációt nagymértékben megkönnyítette XAML-ben lévő Tag-ek sokasága. Elsőként az almenük megjelenítését oldottam meg.



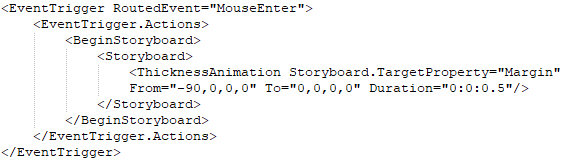
10. ábra: Az almenük megjelenéséért és eltűnéséért felelős XAML kód

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.

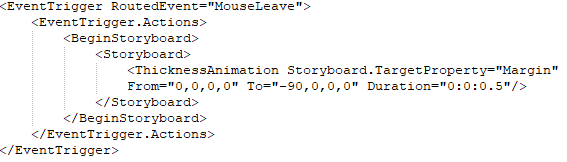
* Ha az IsMouserOver igaz, automatikusan kibontjuk az Expander elemet, úgy, hogy az elem IsExpanded tulajdonságát (Property) expliciten igazra (True) állítjuk. Így láthatóvá válnak a benne elhelyezkedő elemek is.
* Ha az IsMouseOver hamis, tehát a kurzor alatt már nem az aktuális elem helyezkedik el. Annak IsExpanded tulajdonságát expliciten hamisra (False) állítjuk, ezzel bezárjuk az Expandert.

Az almenük kezelése után a következő feladatom az előugró menü kódjának leimplementálása volt, amit a következőképpen oldottam meg:



11. ábra: a menü kicsúszásáért felelő XAML kód

* 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 megváltoztatjuk, 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.
* 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 van lehetősége bejelentkezni illetve regisztrálni.

### 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 hosszúnak kell lennie.
* Biztonsági okokból a jelszó beírására kétszer van szükség. A két jelszónak egyeznie kell.

Ha a felhasználó által beírt adatok nem felelnek meg az előbbiekben leírt követelményeknek a regisztráció érvénytelen lesz, amikről szöveges értesítést kap a felhasználó.

Sikeres regisztráció esetén a fiókhoz tartozó jelszó az adatbázisban titkosítva kerül elmentésre, majd a felhasználó kap egy szöveges értesítést a fiók létrejöttéről..

### Bejelentkezés

A bejelentkezéshez, a felhasználónak ki kell töltenie a felhasználónév illetve a jelszó mezőket. Ha ezt nem teszi meg, kap egy szöveges értesítést ezen adatok kitöltéséről. A bejelentkezés gombra nyomva a következő folyamatok bekövetkezésére kerül sor:

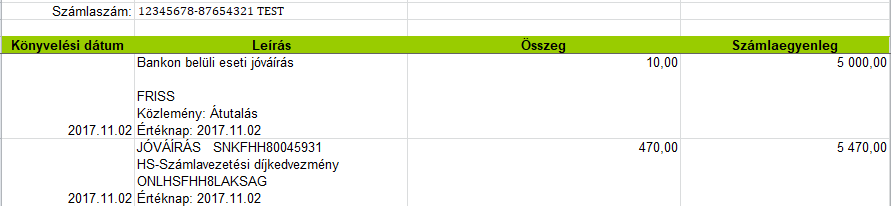
* Az alkalmazás ellenőrzi, hogy létezik-e a felhasználó által beírt fióknév a rendszer adatbázisában.
* Ha létezik a fióknév, akkor az eltárolt, titkosított jelszót visszafejtésre kerül, majd megvizsgáljuk, hogy egyezik-e a beírt jelszóval.
  + Ha igen, akkor a felhasználó számára megjelenik az alkalmazás főmenüje, a Banki tranzakciók importálása.
  + Ha nem, akkor a bejelentkezés sikertelennek minősül.
* Ha sikerestelen a bejelentkezés, akkor a bejelentkezni kívánó személy kap egy szöveges értesítést, hogy az 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.
  + Harminc másodperc elteltével ismét három lehetőség 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). Minden 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 fájlokban szereplő adatok a nem valódiak).



13. ábra: CIB banktól letölthető tranzakciós fájl.

Bátran kijelenthetem, hogy az egyik legbarátságosabb formátummal a CIB banktól letölthető fájl rendelkezik. 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 banktól származó 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 az adatok beolvasást jelenti, nagy problémát jelent, hogy rengeteg üres cella szerepel bizonyos sorokban.

### Felhasználó által deklarált importálás

**Koncepció**

A koncepció egy olyan felület létrehozása, ahol a felhasználónak lehetősége van beírnia az importált fájlhoz tartozó, tranzakció adatait azonosító oszlopszámokat és egyéb adatokat, grafikus elemeket használva (szövegdoboz, legördülő lista). Amik által az alkalmazás az importált fájlból kiolvassa a tranzakciók adatait.

Természetesen a felhasználónak nem kell minding újra és újra beírnia azon importálandó fájlok adatait, amit már egyszer megtett. Ezt megoldva, a rendszer adatbázisában eltárolásra kerülnek a korábban beírt, bizonyos fájltípusokhoz tartozó oszlop és cellaazonosítók. Ami mindig frissülni fog új Felhasználó által deklarált importálás esetén.

Minden Felhasználó által deklarált importálás esetén az összes adatbázisban eltárolt, fájlokhoz tartozó oszlopszám és cellaazonosító betöltődik egy legördülő listába. A legördülő lista aktuálisan kiválasztott elemét változtatva, a felhasználó által kitöltendő adatok értékeihez (szövegdoboz, legördülő lista) beillesztésre kerülnek az eltárolt adatok. Alapértelmezetten a legjobban illeszkedő értékek szerint vannak kitöltve a felhasználó által bekért adatok (szövegdoboz, legördülő lista).

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

Mivel a felhasználónak lehetősége van egyszerre több tranzakciós fájl importálására is, Felhasználó által deklarált importálás esetén a jobb felső sarokban megjelenik egy szövegmező, ami megjeleníti 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;

}

Ami a paraméterben megadott szöveget átalakítja a neki megfelelő számmá. Például „A” betű átadása esetén a visszatérített érték 1 lesz.

A felhasználónak a következőekben felsorolt adat mezőket 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.

#### Tárolt fájl adatok betöltése

**Implementáció**

A korábban importált fájlok eltárolt adatai (oszlopszám, cellaazonosító) a következőképpen kerülnek betöltésre:

* Először is, az adatbázisból lekérésre kerül az összes korábban importált fájlhoz tartozó oszlop és cellaazonosító.
* A felhasználó által importált fájl megnyitásra kerül a program által, majd egyesével megvizsgálja minden eltárolt rekordra az illeszkedés mértékét. Például:
  + a számlaszám pozícióján elhelyezkedő adat illeszkedik-e egy bankszámlaszám formátumára.
  + 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 alapértelmezetten betöltésre kerülni ahol a legtöbb egyezés volt.
* Ha egyáltalán nincs egyezés importált fájl és az eltárolt adatok között, nem töltünk be alapértelmezett adatokat a felhasználó számára.

### Automatikus importálás

**Koncepció**

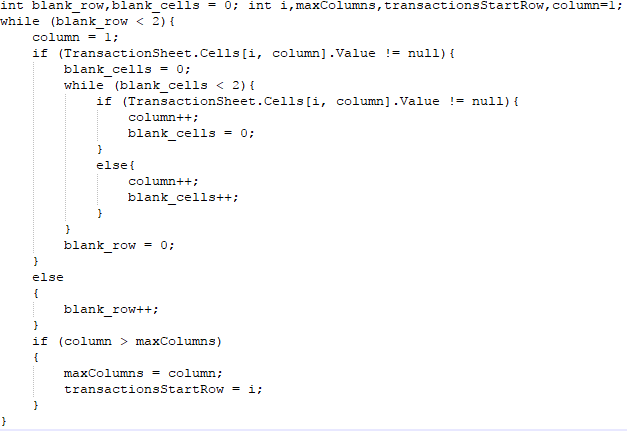
Mivel minden szoftverben fontos az automatizált működés, ezért fontosnak tartottam egy olyan importálás implementálását ahol az alkalmazás algoritmusokat használva **megpróbálja** beazonosítani a tranzakciók kiolvasásához szükséges oszlopokat illetve cellákat. Majd ezen adatokat felhasználva, az alkalmazás 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 fájl sorain és meg találni azt, ami a legtöbb oszloppal rendelkezik, ami egy egyszerű maximum 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.

* ^ jelzi hogy az érték elejétől nézzük az illeszkedést
* $ jelzi a vizsgált érték végét
* 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 szám.
  + ^\d{24]$ huszonnégy darab decimális szám
* 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 valóban 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 (0-9) számot várunk ponttal elválasztva (első két számjegy a hónapot azonosítja, második két számjegy a napot).
    - például 2018.04.20
  + ^\d{4}.\s\d{2}.\s\d{2}$
    - \d{4} jelenti, hogy négy darab decimális számot (0-9) várunk (ez a négy számjegy azonosítja az évet).
    - \d{2}.\s\d{2} jelenti, hogy két darab decimális számot (0-9) várunk White Space karakterrel (szóköz, tabulátor stb.) elválasztva (első két számjegy a hónapot azonosítja, második két számjegy 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$
* Egyenleg oszlop keresése alatt felmerülhet, hogy nem sikerül beazonosítani, mivel az importált fájl nem rendelkezik ilyen oszloppal. Többek között a következő reguláris kifejezések felelnek az Egyenleg oszlop beazonosítására:
  + Egyenleg$
  + ^Számlaegyenleg$

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 alapértelmezett értékek a 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

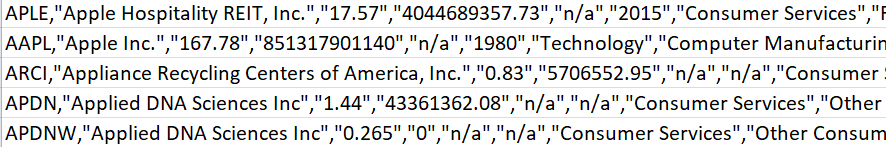
* ^ kifejezés jelenti, hogy a vizsgált elem elejétől nézzük az illeszkedést
  + ^\d{4}.\d{2}.\d{2}$
    - \d{4}. jelenti, hogy négy darab decimális (0-9) számot várunk utána egy ponttal (ez azonosítja az évet)
    - \d{2}.\d{2} jelenti, hogy két darab decimális (0-9) számot várunk ponttal elválasztva (első két számjegy a hónapot azonosítja, második két számjegy a napot).
    - például 2018.04.20
  + ^\d{4}.\s\d{2}.\s\d{2}$
    - \d{4} jelenti, hogy négy darab decimális számot (0-9) várunk (ez a négy számjegy azonosítja az évet).
    - \d{2}.\s\d{2} jelenti, hogy két darab decimális számot (0-9) várunk White Space karakterrel (szóköz, tabulátor stb.) elválasztva (első két számjegy a hónapot azonosítja, második két számjegy 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);

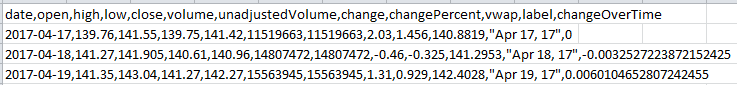
}

}

return d[n, m];

}

* 1. Negyedik lépés elkezdéséhez előfeltétel a harmadik lépés teljesülése.
* 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 az 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

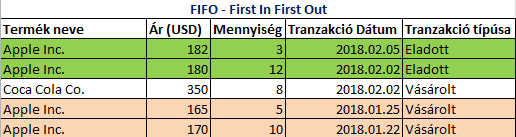
1. A megtalált adatok alapján történő kiolvasás

* 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 felhasználó tőzsdei tranzakciók importálása esetén három különböző módszerre. számítathatja ki az általa eladott részvények esetleges nyerségét, illetve veszteségét. Amit még az importálandó fájl kiválasztása előtt meg kell adnia. A módszerek a következőek:

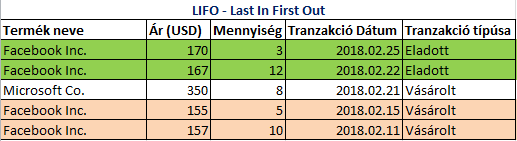
**FIFO (First In-First Out)** azaz amit időrendi sorrendben előbb vásároltunk meg, azt fogjuk először felhasználni tehát eladni. Az alábbi ábrán, egy egyszerű példával szemléltetem a FIFO elv működését.



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, ezt hívjuk el **Első tranzakciónak**. Majd 2018.01.25.-én vásároltunk öt darab Apple Inc. részvényt, darabonként 165 dollár áron ezt hívjuk el 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.
* 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 vásárolt részvényt.
* Ennek az eladásnak a nyeresége a következő: 2 \* ( 180 – 165 )
  + Ahol 2 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 százharminccal.
* Az első eladás nyereség-veszteség kiszámolása után marad összesen három darab Apple Inc. részvényünk a **Második tranzakcióból**.
* A második eladás során pont három darabot adunk el. Így a második eladáshoz tartozó nyereségét a következő módon számoljuk ki: 3 \* ( 182 – 165 )
  + Ahol 3 az eladott részvények száma, 182 a részvények eladási ára 165 a részvények vételi ára.
* Így 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 350 dollár értékben, mivel ezt nem adtuk el, nem foglalkozunk vele a későbbiekben.
* Láthatjuk 2018.02.15.-én vásároltunk Facebook Inc. részvényt, öt darabot, darabonként 150 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**.
* Láthatjuk, 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 darab részvényt 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 7 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.
* Már csak második eladáshoz tartozó nyereség kiszámolása a következik. Mivel a **Második tranzakcióhoz** már csak három részvény tartozik, és a második eladás során három részvényt adtunk el így a következő módon számolhatjuk ki a nyereség-veszteséget : 3 \* ( 170 – 157 )
  + ahol 3 az eladott részvények száma, 170 az eladott részvények ára és 157 a vásárolt részvények ára.
* Tehát a második tartozó nyerség egyenlő harminckilenccel.

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

* Az oldalon láthat majd egy táblázatot, ami tartalmazza a frissen beimportált tranzakciókat. Ahol 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 azért 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 nyerség-veszteség értékeket.
* 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 PropertyChangedEventArgs(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 részvény 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).

## Adatok tárolása adatbázisban

Ahogy a használt technológiák fejezetben is említettem az adatok tárolására SQLite adatbázist használok. A követelmények fejezetben alapvető követelménynek szabtam meg azt, hogy ha az alkalmazás használata során az adatbázisba írunk adatokat, azt a felhasználó ne érzékelje esetleges megfagyásokkal. Ezen feltételt teljesülését egy mellékszál létrehozásával oldottam meg. Amit a következőképpen hozok létre:

ThreadStart threadStart=delegate{ writeToDB(neededTransactions);

};

Thread sqlThread = new Thread(threadStart);

sqlThread.IsBackground = true;

sqlThread.Start();

* Ahol ThreadStart és Thread beépített osztályok.
* writeToDB pedig létrejövő szál által futtatott függvény.

Jelen esetben a külön szálon futó tevékenyég egy függvény, amit egy különálló folyamatként futtatunk. A Thread objektumunk IsBackground objektumának igaz -ra állításával tudatjuk a programmal hogy az induló szál csak egy háttér folyamat. Majd az objektumunkra meghívjuk a Start függvényt, ezzel elindítva azt.

### Adatbázisba történő írás

Az SQLite adatbázisba a következőképpen tudunk írni:

string insertQuery = "insert into [UserInfo](Felhaszn, Jelszo)values('test1','test2')";

SQLiteCommand insercommand = new

SQLiteCommand(insertQuery, mConn);

insercommand.CommandType = CommandType.Text;

insercommand.ExecuteNonQuery();

* Ahol először egy sztring formájában létrehozzuk a végrehajtani kívánt parancsot, amiben szerepel a tevékenység (beszúrás, frissítés stb.).
* A tábla neve, amit írni szeretnénk ( jelen esetben UserInfo).
* A táblában elhelyezkedő oszlopok nevei (jelen esetben Felhaszn, Jelszo),
* Az adatbázisba írandó új értékek (jelen esetben test1,test2).
* Majd azt az SQLiteCommand parancskezelő osztály segítségével, amit az SQLite könyvtára biztosít, végrehatjuk a létrehozott parancsot.

### Adatbázisból történő olvasás

Az SQLite adatbázisból a következőképpen tudunk olvasni:

string usernameInUseQuery = "select \* from [UserInfo]

where Felhaszn= 'test1';

SQLiteCommand command =

new SQLiteCommand(usernameInUseQuery, mConn);

DataTable DT = new DataTable();

SQLiteDataAdapter adapter =

new SQLiteDataAdapter(command);

* Ahol először egy sztring formájában létrehozzuk a végrehajtani kívánt parancsot, amiben szerepelnek a kiolvasandó információk, szűrők.
* A tábla neve, amiből olvasni szeretnénk (jelen esetben UserInfo).
* Majd azt az SQLiteCommand parancskezelő osztály segítségével, amit az SQLite könyvtára biztosít, létrehozott parancsot.
* Majd a parancsot végrehatjuk az SQLiteDataAdapter osztály segítségével, ami visszatéríti az olvasás parancs eredményét.

# 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 nagymértékben 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.

Összességében elmondható hogy az eredmény megfelel a felállított követelményeknek. A rendszer fejlesztésére, eredményesebb működésére úgy lenne lehetőség, hogy minél több Magyarországon elérhető banktól származó tranzakciós export fájlal tesztelni lehetne.

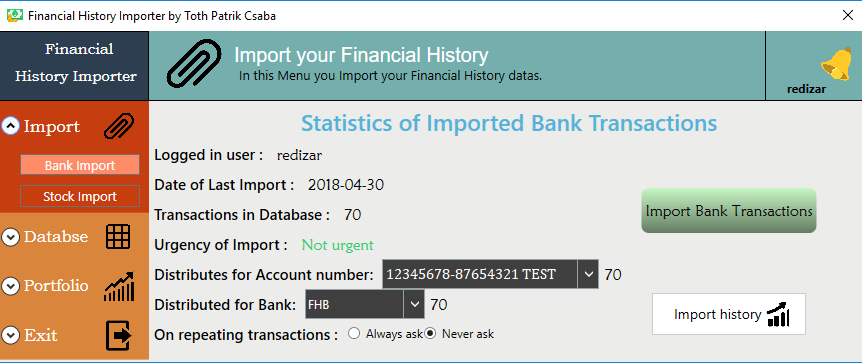
# Irodalomjegyzék

1. *https://index.hu/gazdasag/2017/09/28/megnyilnak\_a\_kapuk\_a\_bankrendszer\_felforgatoi\_elott Tóth Patrik Cs. (letöltés dátuma 2017. október 10.)*
2. *https://iextrading.com/developer/docs/#getting-started Tóth Patrik Cs. (letöltés dátuma 2018, március 9)*
3. *http://www.nasdaq.com/screening/companies-by-industry.aspx?render=download Tóth Patrik Cs. (letöltés dátuma 2018, március 1)*
4. *https://github.com/Live-Charts/Live-Charts Tóth Patrik Cs. (letöltés dátuma 2018, január 8)*
5. *https://www.mnb.hu/letoltes/konzultacios-dokumentum.pdf Tóth Patrik Cs. (letöltés dátuma 2018, január 15)*
6. *https://www.sqlite.org/docs.html Tóth Patrik Cs. (letöltés dátuma 2018, április 15)*

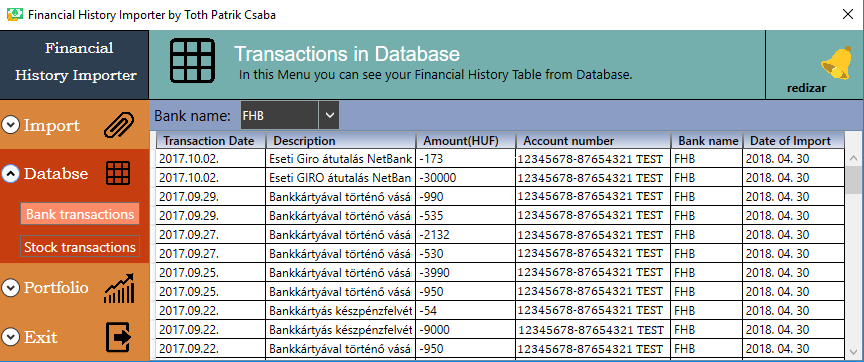
# Mellékletek

Program futtatásához szükséges teendők, követelmények:

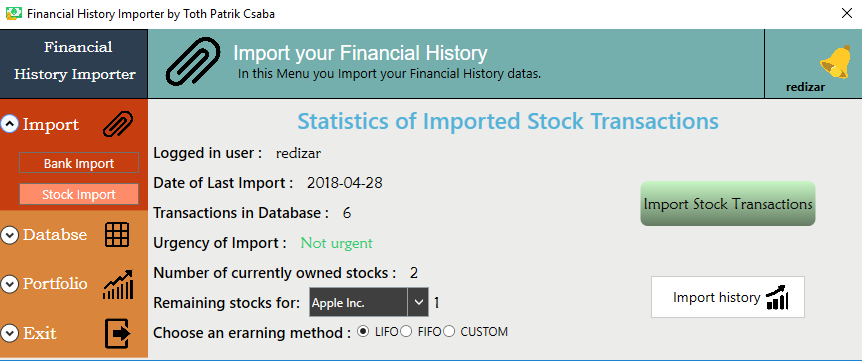
* A CD mellékleten található Deployment mappa lemásolása sajátgépre.
* A Deployment mappában található Setup.exe futtatása.
* A program telepítése és futtatás közben rendszergazda jogok szükségesek.
* Automatikus tőzsdei tranzakciók beolvasása esetén internet elérés szükséges.
* Tőzsdei adatok lekérése esetén internet elérés szükséges.



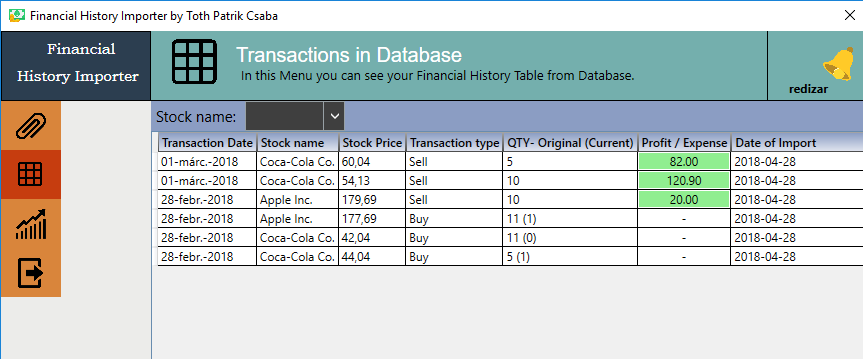
20. ábra: banki tranzakciók importálása menü.



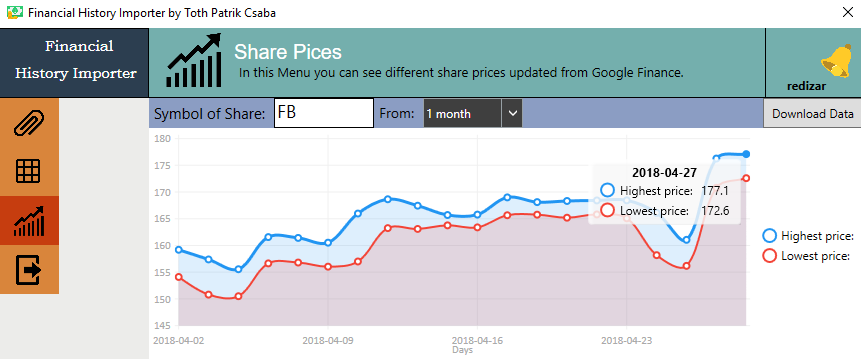
21. ábra: importált banki tranzakciók megjelenítése egységesített táblázatban.



22. ábra: banki tranzakciók importálása menü.



23. ábra: importált tőzsdei tranzakciók megjelenítése egységesített táblázatban.



24. ábra: Facebook Inc. részvények legmagasabb, legalacsonyabb árfolyama napokra lebontva, egy hónapra visszamenőleg..



25. ábra: korábban grafikon formájában lekért részvényadatok részletesebb elmentett adatai egy táblázat formájában megjelenítve.