Pannon Egyetem   
Műszaki Informatikai Kar  
Projekt Alapú Szoftverfejlesztés (VEMISAM233P/2020/21/2)  
Programtervező informatikus MSc

**Publikációs listák szinkronizálása**

2021

(RSZT 3. téma)

Témavezető: Szekér Szabolcs

Csapatattagok:

Szászfai Júlia (GO45XL)  
 Csizmazia Máté (XI32IS)  
 Pataki Miklós (BQPM6L)

**Tartalom**

[**1.** **A projekt munkaterve** 3](#_Toc71553201)

[**2.** **Bevezetés** 5](#_Toc71553202)

[**3.** **MTMT és Google Scholar rendszerek ismertetése** 6](#_Toc71553203)

[**3.1.** **MTMT rendszer** 6](#_Toc71553204)

[**3.2.** **Google Scholar rendszer** 6](#_Toc71553205)

[**4.** **Felhasználóhoz tartozó publikációs lista lehívása MTMT és Google Scholar rendszerekből** 8](#_Toc71553206)

[**4.1.** **Google Scholar API vizsgálata, kiértékelése** 8](#_Toc71553207)

[**4.1.1.** **Scraper API** 8](#_Toc71553208)

[**4.1.2.** **SERP API** 9](#_Toc71553209)

[**4.1.3.** **SerpWow** 9](#_Toc71553210)

[**4.1.4.** **Scale SERP** 10](#_Toc71553211)

[**4.1.5.** **Publish or Perish** 10](#_Toc71553212)

[**4.1.6.** **scholarly** 12](#_Toc71553213)

[**4.2.** **Web scraping technológia bemutatása** 13](#_Toc71553214)

[**4.2.1.** **Web scraping fogalma** 13](#_Toc71553215)

[**4.2.2.** **Web scraping története** 14](#_Toc71553216)

[**4.2.3.** **Technikák** 14](#_Toc71553217)

[***Emberi másolás-beillesztés*** 14](#_Toc71553218)

[***Mintaszöveg egyezés*** 15](#_Toc71553219)

[***HTTP programozás*** 15](#_Toc71553220)

[***HTML elemzés (parsing)*** 15](#_Toc71553221)

[***DOM elemzés*** 15](#_Toc71553222)

[***Függőleges összesítés*** 15](#_Toc71553223)

[***Szemantikai annotáció felismerése*** 16](#_Toc71553224)

[***Számítógépes látás – weboldal analízis*** 16](#_Toc71553225)

[***Selenium*** 16](#_Toc71553226)

[**5.** **Azonos feltételű kereső megvalósítása MTMT és Google Scholar rendszerekben** 18](#_Toc71553227)

[**6.** **Adatok kigyűjtése, találatok megjelenítése** 29](#_Toc71553228)

[**7.** **Találatok összehasonlítása és különbségek keresése** 32](#_Toc71553229)

[**8.** **Automatikus/szoftverrel történő összehasonlítás előnye a manuálissal szemben** 34](#_Toc71553230)

[**9.** **Összegzés, további fejlesztés** 35](#_Toc71553231)

[**10.** **Irodalomjegyzék** 36](#_Toc71553232)

1. **A projekt munkaterve**

* MTMT és Google Scholar rendszerek elemzése
* Technológiák kiválasztása
* Rendszerterv elkészítése
* C# Forms alkalmazás skeleton létrehozása
* Webscraper (Selenium) implementálása
* Custom Google Scholar API (scholarly, flask) fejlesztése
* Custom Python API integrálása C# környezetbe
* Szerzőnkénti lekérdezés megvalósítása
* Felhasználói felület fejlesztése
* Kigyűjtött adatok közös adatszerkezetre alakítása
* Lekért adatok összekötése a felülettel
* Találatok összehasonlítása
* Találatok kiértékelése: különbségek, egyezések megjelenítése

***Munka felosztása***

Szászfai Júlia:

* MTMT és Google Scholar rendszerek elemzése
* Osztályok kialakítása
* Adatok összehasonlítása, különbségek keresése
* Felhasználói felület
* Dokumentáció

Csizmazia Máté:

* API-ok elemzése
* Google Scholar-ból adatok lekérdezése
* Flask REST API létrehozása
* Felhasználói felület
* Dokumentáció

Pataki Miklós:

* Webscraping technológia elemzése, kiértékelése
* MTMT rendszerből adatok letöltése
* MTMT és Google Scholar adatok integrációja
* Felhasználói felület kialakítása
* Dokumentáció

1. **Bevezetés**

Az MTMT publikációs lista Google Scholar alapján történő manuális szinkronizációja időigényes feladat, ezért egy olyan framework elkészítése a cél, ami lehetővé teszi ezek automatizálását (adatok kigyűjtése, adatok összehasonlítása) és ellenőrzését egy felhasználóbarát felhasználói felület segítségével. A megvalósítandó feladatok az alábbiak:

* + Kereső létrehozása
  + Adatok kigyűjtése az MTMT oldaláról
  + Adatok kigyűjtése a Google Scholar oldaláról
  + Lekért adatok listázása GUI-ba
  + listák közötti különbségek felfedezése és megjelenítése

A szoftver elkészítését egy C# Forms alkalmazásban tervezzük elkészíteni. A forms segítségével könnyedén és gyorsan létrehozható egy olyan grafikus felület, amivel látványosan megjeleníthetők a publikációk. A két felületről lekért adatokat betöltjük egy egymáshoz közel lévő, egymás mellet elhelyezkedő grid elembe, így listaként tudjuk kezelni ezeket a publikációkat. Az itt megjelenő publikációk kiegészülnek további részletes információkkal, ami attól függ, hogy melyik felület, milyen publikációs részletekkel rendelkezik. Az MTMT-ről web scraping segítségével kérjük le az adatokat, amíg a Google Scholar felületéről egy API segítségével tervezzük az adatlekérést. Miután a program megkapta a két felület adatát, ezt egy közös szerkezetre hozzuk, így könnyítve az összehasonlítás menetét. Így a továbbiakban, az észlelt eltéréseket könnyedén tudja javítani manuálisan a szerző.

1. **MTMT és Google Scholar rendszerek ismertetése**
   1. **MTMT rendszer**

Az MTMT rendszer, másnéven a Magyar Tudományos Művek Tára, egy olyan webalkalmazás, ami kifejezetten azért jött létre, hogy legyen egy sok célra felhasználható nemzeti bibliográfiai adatbázis Magyarországon. Az öt alapító intézmény egyetértett abban, hogy a hazai tudományos kutatómunkák nyilvántartására és bemutatására meg kell valósítani ezt az alkalmazást, a rendszerhez csatlakozó szervezetek segítségével. Ennek a közös rendszernek a megalkotása több előnnyel járt:

* Egységes, országszintű rendszerben vehetnek részt azt intézmények
* Üzemeltetése és fejlesztése jelentősen olcsóbb, mint a lokális fejlesztés és működtetés
* Sok célra felhasználhatóak az adatok
* Külső minőségbiztosítás integrálva van
* Nagyobb szervezetek adatigényeivel kompatibilis

Az MTMT rendszer emellett természetesen a kutatók érdekeit is szolgálja, mivel a személyes tudományos bibliográfia összeállítása egyszerű, kis munkaigénnyel karbantartható, segíti a pályázatok és jelentések készítését, személyes honlapra beilleszthet és nem utolsó sorban az MTA, ODT, MAB, OTKA elfogadja, preferálja.

Az MTMT az alábbi szolgáltatásokat nyújtja:

* Hiteles tudományos eredmény-nyilvántartást biztosít, és bemutatja az eredményeket, magyar és részben angol nyelven.
* Nyilvános, hiteles, átvehető és hivatkozható publikációs listákat ad. Ezek felhasználhatók az intézményi akkreditációban, a doktori iskolákkal kapcsolatos követelmények teljesítése során, MTA, OTKA és NKTH pályázatoknál.
* Bibliográfiai adatok révén átmeneteket biztosít a teljes szövegű közleménytárhoz.
* Az adatot szolgáltató intézmények tudományos teljesítményéről összesített képet ad.
* A rendszer automatikus adatfeltöltést és megfelelő előkészítéssel adatkonverziót is végez, meglévő adattárakból és könyvtári rendszerekből.

Ezen szolgáltatások korszerűsítésén folyamatosan dolgozik az MTMT csapata.

* 1. **Google Scholar rendszer**

A Scholar, a Google anyavállalat egy specifikusan tudományos irodalom keresésére létrehozott felülete. Ez a felület egy egyszerű módszert kínál a keresésre, akár több tudományterület vagy forrás között, a Google Scholar mindig segít megtalálni a releváns munkát a tudományos kutatások világában.

A Scholar az alábbi pontok miatt játszik nagy szerepet a publikációk nyilvántartásában:

* Bármilyen tudományos irodalom keresésére ad lehetőséget egy felületen.
* Jól láthatóan jelzi a kapcsolódó műveket, idézeteket, szerzőket és publikációkat.
* Visszakereshető a forrása a dokumentumnak
* Szerzői profil létrehozásának lehetősége
* Jól követhető és ellenőrizhető a kiadott tanulmányok idézései

A Google Scholar célja, hogy rangsorolja a dokumentumokat a kutató módjára, mérlegelve a publikációk teljes szövegét, megjelenésének helyét, azt, hogy hol készült a kutatás, másrészt azt is, hogy milyen gyakorisággal idézte más tudományos szakirodalom a művet a közelmúltban.

Alapjában ugyan úgy működik, mint a Google kereső, tehát ha valami kulcsszóra keresünk rá, legyen az a szerző neve, intézmény neve, kutatói terület, vagy csak simán egy szó, a Scholar mindig fog tudni találatot visszaadni.

1. **Felhasználóhoz tartozó publikációs lista lehívása MTMT és Google Scholar rendszerekből**

Ezen rendszerekből az adatkigyűjtés nem egy egyszerű feladat, ezért a fejlesztés kezdete előtt szükséges volt feltérképezni a lehetőségeket a megvalósítással kapcsolatban. Az MTMT oldal egy jól rendszerezett weboldal, ahonnan rengeteg információ kinyerhető, viszont alapjában nem rendelkezik bármilyen API-val, ami egyszerűbbé tenné az ilyesféle adatok kinyerését. Ebből az okból kifolyólag találni kellett egy megoldást ennek kiküszöbölésére, ami nem más, mint a Web Scraping technológia alkalmazása. A Google Scholar esetében több lehetőség is adott volt, mivel egy nagy rendszerről beszélünk, ezért már több lekérdezéseket segítő API-t is készítettek hozzá. A fejlesztők nagyrésze jobban szereti használni ezeket az API-kat, mivel jelentősen le tudja rövidíteni egy fejlesztés menetét, ha egy jól működőt sikerül találnunk, viszont ezen szolgálatások használatáért több helyen jelentős összegeket is elkérhetnek. A további alfejezetekben bemutatásra kerülnek azok az API-k, amiket a fejlesztők a leginkább használnak ehhez.

* 1. **Google Scholar API vizsgálata, kiértékelése**

A Google Scholar (Tudós) a tudományos cikkek és kutatási folyóiratok milliárdjaival az egyik legjobb online forrás a magas színvonalú tudományos kutatáshoz. Mivel azonban a Google nem nyújt API-t, amely lehetővé teszi a szükséges adatok nagy mennyiségű megkeresését és kinyerését, más számos módon kell megszerezni a szükséges adatokat. Esetleg külső felek API-ján keresztül, vagy webscraping technikával, kimásoljuk az adatokat. A következő részben néhány lehetőséget tárgyalunk ezek közül, és megadjuk előnyeit és hátrányait.

* + 1. [**Scraper API**](https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/#scraperapi)

A Scraper API egy proxy API, amelyet arra terveztek, hogy az internetről az adatokat a lehető legnagyobb mértékben lementse.

A Google-hoz hasonlóan nehezen scrap-elhető webhelyekhez készült Scraper API csökkenti a saját proxy infrastruktúrájának kiépítésének és fenntartásának szükségességét. Egyszerűen elküldi a lescrapelni kívánt URL-t az API-nak, és ez kezeli a változó proxykat, az automatikus újrapróbálkozásokat, a CAPTCHA-kat és a blokkolásokat, a hibákat elrejti és csak sikeres eredményeket hozza. Innentől kezdve a parancsfájlnak csak elemeznie kell a HTML-válaszból szükséges adatokat.

Ha egy előre elkészített Google Scholar scrap-elő könyvtárral kombináljuk, mint például a Scholarly, akkor néhány órán belül könnyedén létrehozhatunk egy egyedi Google Scholar API-t, amely megfelel az adatigényeinek.

99 dollárért havonta 1 000 000 Google Scholar oldalt is kinyerhetünk. A Scraper API-kat rendelkezik ingyenes próbaverzióval, ami 5000 ingyenes kérést tud.

***Előnyök***

A legolcsóbb lehetőség azok számára, akik megbízhatóan szeretnék kinyerni a Google Scholar adatait kutatási projektjeikhez. Ráadásul nagyvonalú ingyenes próbaverzió is rendelkezésre áll.

***Hátrányok***

Alapvető ismeretekre van szüksége a webes scrapelésről.

* + 1. [**SERP API**](https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/#serpapi)

A következő a SERP API, amely remek lehetőség azok számára, akik csak a Google Scholar adatait szeretnék lekérni, és nem akarják, hogy saját webscrapereket készítsenek.

A SERP API egy kifejezetten a Google Scholar számára létrehozott API, amely a tipikus keresési eredményekben található összes adatot visszaadja; beleértve: cím, link, töredék, idézetek, kiadvány stb.

Ennek a nagyszerű API-nak az egyetlen hátránya a költség. A kötségmodell 50 dollárnál kezdődik 5000 keresés után, és 250 dollárra nő 30 000 API-hívás esetén, ez a Google Scholar API nagyon költséges lehetőség lehet, ha sok Google Scholar-adatra van szüksége.

***Előnyök***

Kiváló minőségű és könnyen használható Google Scholar API, amely visszaadja az összes szükséges információt.

***Hátrányok***

Drágább lehet nagyobb projekteknél, és nem testreszabható a saját projektje követelményeinek megfelelően.

* + 1. [**SerpWow**](https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/#serpwow)

Egy másik, harmadik féltől származó Google Scholar API-t biztosító vállalat a SerpWow. Bár nincs annyira dokumentálva, mint a SERP API Google Scholar API-ja, ez az API sok ugyanolyan funkcionalitással rendelkezik, valamivel olcsóbb áron.

Egyszerűen el lehet küldeni a keresési lekérdezést az API -juknak, és az összes Google Scholar keresési eredményt JSON formátumban adja vissza.

Az 5000 API-hívásért 45 dollárral kezdődő költségmodell nagyszerű megoldás, ha gyors és egyszerű módszert szeretnénk kinyerni a Google Scholar adataiból. Azonban a részletezett API megoldásokhoz hasonlóan ez is nagyon drága lehet, ha nagyobb mennyiségű adatra van szükség - 100 000 API hívás havonta 500 dollár.

***Előnyök***

Az adatokat JSON formátumban adja vissza, és valamivel olcsóbb, mint a SERP API.

***Hátrányok***

Nincs külön Google Scholar dokumentáció, és nagyon költséges.

* + 1. [**Scale SERP**](https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/#scaleserp)

A Scale SERP a következő Google Scholar API a listán. Bár a SerpWow-hoz hasonlóan néz ki, alacsonyabb áron kínálnak nagyon hasonló terméket.

Mivel a tervek havi 4 dollárért kezdődik 250 API-hívásért, és 10 millió API-hívássá növekednek 8500 dollárért, a Scale SERP mindenki számára kínál valamit.

A SerpWow-hoz és a SERP API-hoz hasonlóan ők is JSON formátumban adják vissza az adatokat, azonban adataik nem annyira részletesek. Csak a csupasz alapvető dolgokat részesíti előnyben, például a címet, a linket, a szerzőt és a részleteket. Bizonyos adatokat nem ad vissza, például az idézetek száma stb.

***Előnyök***

A legolcsóbb Google Scholar specifikus API, de még mindig legalább háromszor drágább, mint a Scraper API.

***Hátrányok***

Nem ad olyan részletes adatokat, mint a többi API, és nem testre szabható.

* + 1. [**Publish or Perish**](https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/#publishorperish)

A Publish or Perish kifejezetten a Google Scholar számára tervezett adatkivonó eszköz, amely lehetőséget nyújt saját Google Scholar API létrehozására.

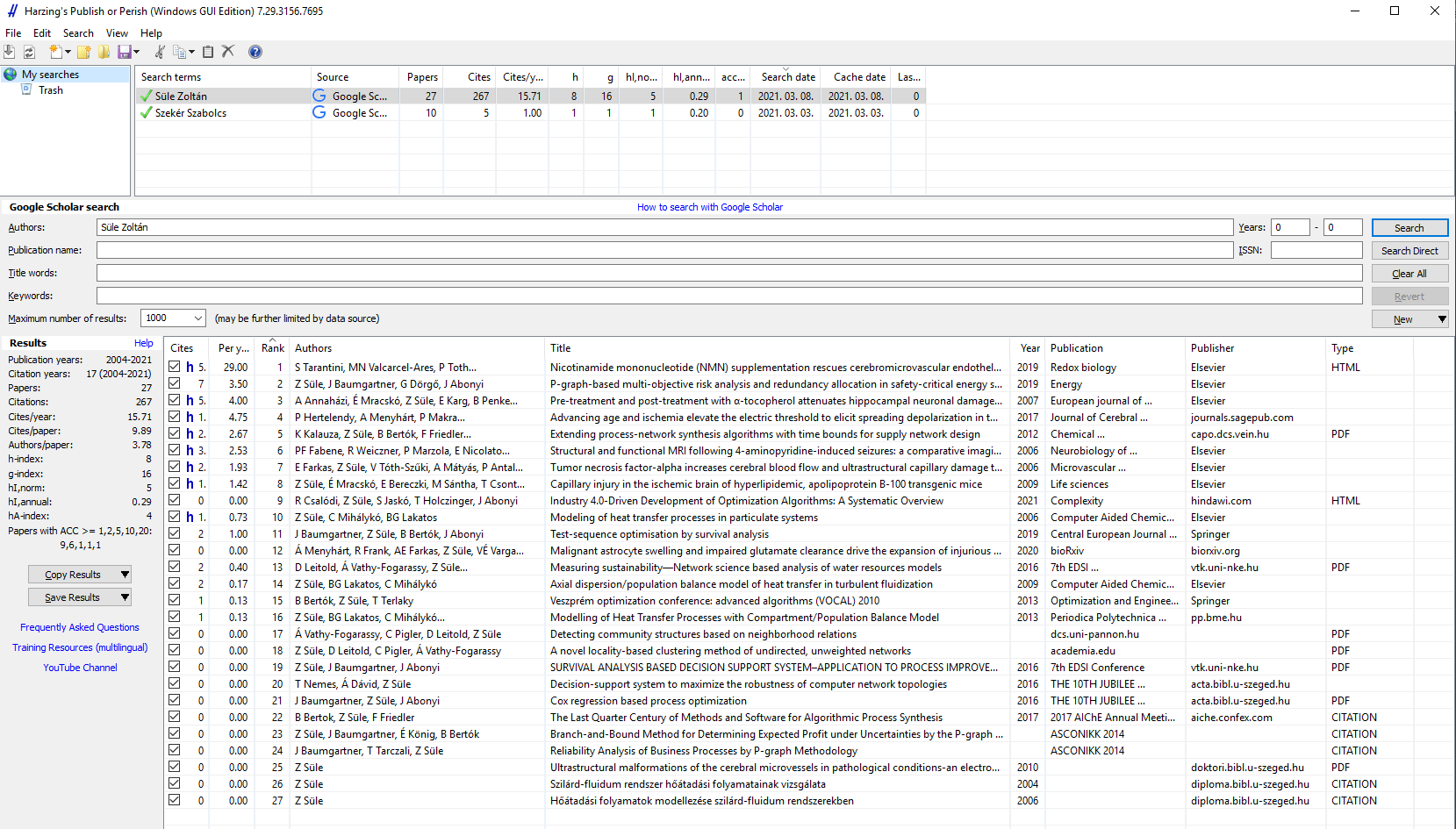
Vigyázni kell azonban, mivel a szoftver kéréseket küld a Google Scholar-nak a számítógép IP-címének felhasználásával, így a Google letilthatja IP-címét, ha túl sok adatot próbálunk meg kinyerni ezzel az eszközzel. Tehát, ha több mint pár száz keresési eredményt kell kinyernünk a Google Scholarból, akkor mindenképpen használnunk kell egy olyan proxy megoldást, mint pl. a Scraper API.

***Előnyök***

Teljesen ingyenes és könnyen használható.

***Hátrányok***

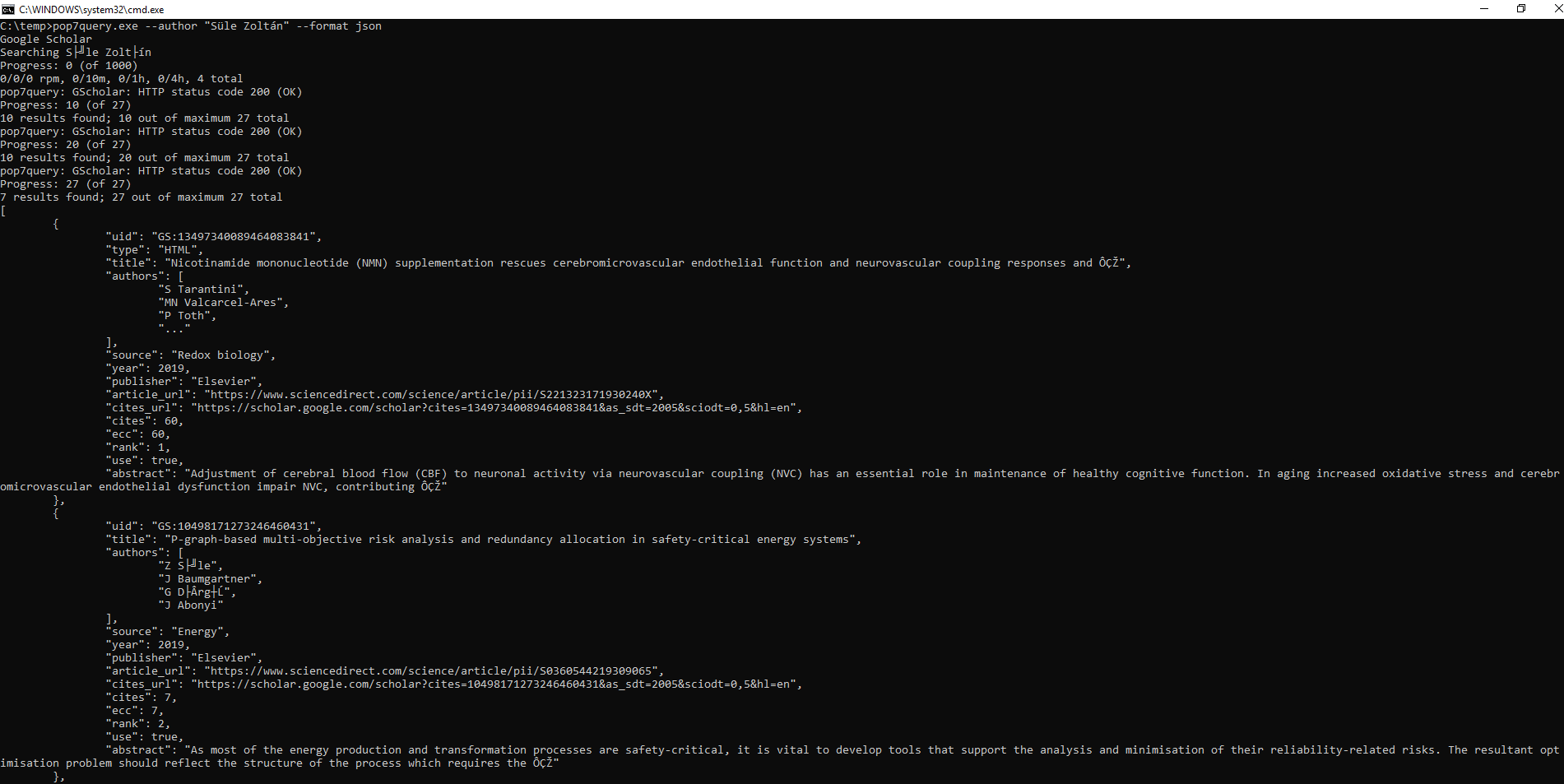
Fennáll annak a kockázata, hogy IP-címét letiltják, ha proxy használata nélkül használjuk.



**1**. ábra - Publish or Perish asztali változata

Publish or Perish nem csak grafikus, hanem parancssoros felületet is biztosít. Ez a felület is elérhető több platformon is. Az eredményt képes fájlba is irányítani.

Különböző formátumokat támogat: csv, json, rtf, tsv. A következő képernyőképen látható egy keresés eredménye a parancssoros futtatásról:



**2**. ábra - Publish or Perish konzolos változata

* + 1. **scholarly**

A scholarly egy Python nyelvhez íródott könyvtár, ami segíti a Google Scholar oldalon tárolt adatok lekérdezését. Mivel a python nyelvhez íródott, ezért a használata roppant egyszerű, akár két sor megírásával már lekérdezhetjük egy szerző vagy publikáció adatait. Természetesen, ha részletesebb adatok lekérése a cél, ezeket is végre lehet hajtani, viszont nagyszámú publikációval rendelkező szerzők esetében ez egy hosszadalmas művelet is lehet. A lekérdezések megbízhatóak, ezért az esetek nagyrészében mindig ad eredményt. Szolgáltat külön kereső metódusokat szerzőre, publikációra vagy akár csak kulcsszavakra. Ezen keresések eredményei később kiegészíthetők további kitöltő metódusokkal, aminek paraméterébe megadható, hogy milyen további információk lekérése a cél. Ez a technológia eltér az eddig bemutatottaktól, mivel ez nem egy nyelvfüggetlen technológia, tehát nehezen integrálható egy C# nyelven íródott alkalmazásba.

***Előnyök***

Ingyenesen használható, Proxy beállítási lehetőségeket biztosít, egyszerű használat, biztonságos

***Hátrányok***

A legrészletesebb lekérdezések hosszú futási időt tudnak generálni. Nem JSON formátumban adja vissza az adatokat

* 1. **Web scraping technológia bemutatása**
     1. **Web scraping fogalma**

Az angol szótár szerint a scraping jelentése kaparás. A szó szerinti jelentés nem is áll nagyon messze a technika lényegétől. A webscraping technológia az adatok kinyerését jelenti webhelyekről. A webscraper szoftver közvetlenül hozzáférhet a világhálóhoz a Hypertext Transfer Protocol vagy a webböngésző segítségével. Míg a webscraping-nek a felhasználója egy szoftver, webes adatok kinyerését manuálisan is elvégezhetünk. A webscarping kifejezés tipikusan robotok segítségével megvalósított automatizált folyamatokra utal. Ez egy olyan másolási forma, amelyben meghatározott adatokat gyűjtenek és másolnak az internetről, jellemzően egy központi vagy helyi adatbázisba vagy táblázatba későbbi visszakeresés vagy elemzés céljából.

A webscraping magában foglalja annak beolvasását és kibontását. A letöltés egy oldal letöltése, amelyet a böngésző akkor csinál, amikor a felhasználó megnéz egy oldalt. Ezért az internetes feltérképezés az internetes scraping egyik fő eleme, az oldalak későbbi feldolgozás céljából történő letöltéséhez. A letöltés után kivonás történhet. Az oldal tartalma elemezhető, kereshető, újra formázható, adatát táblázatba másolhatja, és így tovább. A webscraper-ek általában kivesznek valamit az oldalból, hogy más célra felhasználják valahol másutt. Példa lehet a nevek és telefonszámok, vagy a vállalatok és azok URL-jeinek megkeresése és egy listára másolása (névjegy-scraping).

A weboldalak szöveges jelölőnyelvek (HTML és XHTML) felhasználásával készülnek, és gyakran rengeteg hasznos adatot tartalmaznak szöveges formában. A legtöbb weboldal azonban az emberi végfelhasználók számára készült, és nem az automatizált használat megkönnyítése érdekében. Ennek eredményeként speciális eszközöket és szoftvereket fejlesztettek ki a weboldalak scrap-elésének megkönnyítésére.

A webscraping újabb formái magukban foglalják a webszerverekről érkező adatcsatornák meghallgatását. Például a JSON-t általában az ügyfél és a webkiszolgáló közötti szállítási tárolási mechanizmusként használják.

Vannak olyan módszerek, amelyeket egyes webhelyek megakadályoznak az internetes scrapingben, például észlelik és letiltják a robotok oldalainak feltérképezését. Válaszul vannak olyan webscraping rendszerek, amelyek a DOM-elemzés, a számítógépes látás és a természetes nyelv feldolgozásának technikáira támaszkodnak az emberi böngészés szimulálásához, hogy lehetővé tegyék a weboldal tartalmának összegyűjtését offline elemzéshez.

A webes scraping az a folyamat, amely során automatikusan bányásznak adatokat, vagy információkat gyűjtenek a világhálóról. Ez egy olyan terület, amelynek aktív fejlesztései közös célt szolgálnak a szemantikus webes jövőképpel, egy ambiciózus kezdeményezés, amelyhez még mindig áttörésekre van szükség a szövegfeldolgozásban, a szemantikai megértésben, a mesterséges intelligenciában és az ember-számítógép interakciókban. A jelenlegi webscraping megoldások az ad-hoc-tól kezdve, emberi erőfeszítéseket igényelnek, egészen a teljesen automatizált rendszerekig, amelyek képesek teljes weboldalakat strukturált információvá alakítani, korlátozásokkal.

* + 1. **Web scraping története**

A világháló 1989-es születése után 1993 júniusában hozták létre az első webrobotot, a World Wide Web Wanderert, amelyet csak a web méretének mérésére szántak.

1993 decemberében elindult az első robot, amely webalapú keresőmotort, a JumpStation-t jelentette. Mivel nem volt olyan sok weboldal elérhető az interneten, a keresőmotorok abban az időben az emberi webhelyek rendszergazdáira támaszkodtak, hogy összegyűjtsék és szerkesszék a linkeket egy adott formátumban. Ehhez képest a JumpStation nagy előrelépést hozott, ez volt az első olyan WWW keresőmotor, amely egy webrobotra támaszkodott.

2000-ben jött az első webes API (Application Programming Interface) és API robot. Ez egy olyan felület, amely sokkal könnyebbé teszi a programfejlesztést építőelemek biztosításával. 2000-ben a Salesforce és az eBay elindította saját API-ját, amely lehetővé tette a programozók számára, hogy hozzáférjenek és letöltsék a nyilvánosság számára elérhető adatok egy részét. Azóta számos webhely kínál webes API-kat, hogy hozzáférjenek nyilvános adatbázisukhoz.

* + 1. **Technikák**

***Emberi másolás-beillesztés***

A webscraping legegyszerűbb formája az adatok manuális másolása és beillesztése egy weboldalról egy szöveges fájlba vagy táblázatba. Néha még a legjobb webscraping technológia sem helyettesítheti az emberi vizsgálatot és a másolás és beillesztés funkciót, és néha ez lehet az egyetlen működőképes megoldás, amikor a kaparó weboldalak kifejezetten akadályokat állítanak a gépi automatizálás megakadályozása érdekében.

***Mintaszöveg egyezés***

Az információk weblapokról történő kinyerésének egyszerű, ugyanakkor hatékony megközelítése a UNIX grep parancson vagy a programozási nyelvek rendszeres kifejezés-illesztési szolgáltatásain (például Perl vagy Python) alapulhat.

***HTTP programozás***

A statikus és dinamikus weboldalakat úgy is le lehet kérni, hogy HTTP kéréseket küldünk a távoli webszerverre socket programozással.

***HTML elemzés (parsing)***

Sok webhely nagy oldalgyűjteménnyel rendelkezik, amelyek dinamikusan generálódnak egy mögöttes strukturált forrásból, például egy adatbázisból. Az azonos kategóriába tartozó adatokat tipikusan egy közös szkript vagy sablon kódolja hasonló oldalakra. Az adatbányászatban wrappernek (burkolónak) nevezzük azt a programot, amely egy adott információforrásban felismeri az ilyen sablonokat, kinyeri annak tartalmát és relációs formába fordítja. A wrappergeneráló algoritmusok feltételezik, hogy a wrapper indukciós rendszer bemeneti oldalai megfelelnek egy közös sablonnak, és könnyen azonosíthatók egy közös URL séma alapján. Sőt, néhány félig strukturált adatlekérdező nyelv, például az XQuery és a HTQL, is használható HTML-oldalak elemzésére, valamint az oldalak tartalmának lekérésére és átalakítására.

***DOM elemzés***

A teljes körű webböngésző, például az Internet Explorer vagy a Mozilla böngészőbe ágyazásával a programok lekérhetik az ügyféloldali szkriptek által generált dinamikus tartalmat. Ezek a böngészők vezérlik a weboldalakat DOM fává is, amelyek alapján a programok lekérhetik az oldalak egy részét. Olyan nyelvek, mint az Xpath, használhatók a kapott DOM-fa elemzésére.

***Függőleges összesítés***

Néhány vállalat kifejlesztett vertikum specifikus adatgyűjtési platformot. Ezek a platformok készítenek és figyelnek nagy mennyiségű botokat bizonyos különböző vertikumra ember nélkül és nem egy megjelölt céloldalra. Az előkészítés magában foglal egy tudásbázis felépítését az egész vertikumra és ezután a botok elkészülnek automatikusan. A platform robusztusságát a kinyert információk minőségével mérik (általában a mezők száma) és a skálázhatóságával (milyen gyorsan skálázható száz vagy ezernyi oldalakra). Ez a skálázhatóság gyakran az oldalak „hosszú farkát” (statisztikában az előfordulási grafikon domináló jobb oldalát) célozzák meg, a közös felhalmozást bonyolult vagy túl munkaigényes, hogy a tartalmat kibányásszák.

***Szemantikai annotáció felismerése***

A scrap-elt oldalak tartalmazhatnak metaadatokat, szemantikai jelöléseket és annotációkat, amik jelölhetnek bizonyos adattöredékeket. Ha az annotációk az oldalba vannak ágyazva (ahogy a Microformat-nál), akkor ez a technika a DOM elemzésnek egy különleges eseteként tekinthető. Más esetekben az annotációkat egy szemantikus rétegbe szervezik és ezeket a weboldalaktól külön tárolják és szervezik, tehát a scraper-ek le tudják kérdezni az adatok sémáját ebből a rétegből.

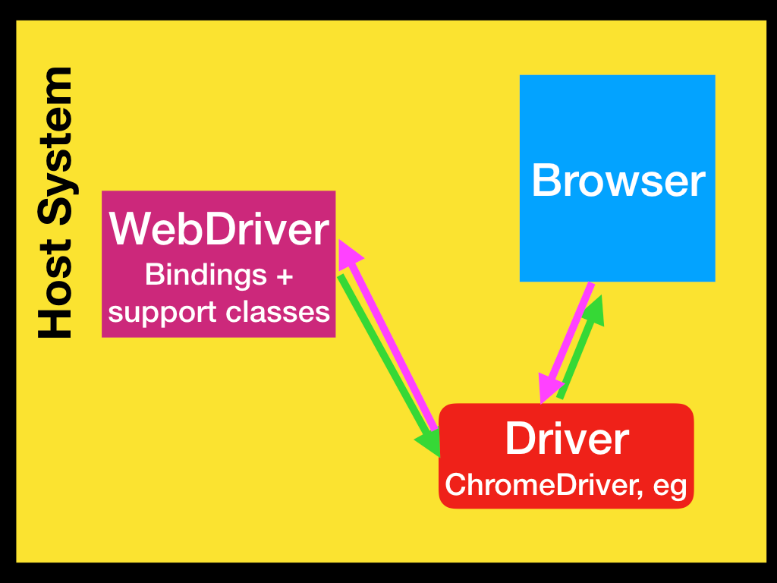
***Számítógépes látás – weboldal analízis***

A gépi tanulás és a számítógépes látás alkalmazásával olyan erőfeszítéseket tesznek, amelyek megkísérlik azonosítani és kinyerni az információkat a weboldalakról azáltal, hogy az oldalakat vizuálisan értelmezik, mint ember.

***Selenium***

A Selenium egy összefoglaló „ernyő” projekt számos eszköz és könyvtár számára, amelyek lehetővé teszik és támogatják a webböngészők automatizálását. Bővítményeket kínál a felhasználói böngészőkkel való interakció utánzásához, amely lehetővé teszi kód írását az összes nagyobb webböngészőhöz. Ezt a projektet olyan önkéntes közreműködők teszik lehetővé, akik a nyílt forráskóddal mindenki számára szabadon hozzáférhetővé tették. A Selenium középpontjában a WebDriver található, amely egy utasításkészletek írására szolgáló felület, amely sok böngészőben felcserélhetően futtatható.

Felépítését a következő ábrán láthatjuk:



3. ábra - Selenium felépítése

1. **Azonos feltételű kereső megvalósítása MTMT és Google Scholar rendszerekben**

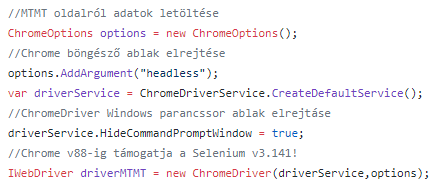
Az MTMT-ről lekérendő adatok összegyűjtésére az előzőleg bemutatott web scraping technológiát alkalmaztuk, ezek közül a Seleniumot. A Google Scholar-os adat lekérdező API-k magas költsége miatt, az említettek közül a scholarly-ra esett a választás, mivel ez a könyvtár az egyetlen ingyenes, megbízható és könnyen használható megoldás. Ennek a technológiának a választása több kérdést is felvetett, mint hogy miképp válhat integrálhatóvá a scholarly egy C# nyelven íródott alkalmazásba. Mivel a többi megoldás egy API hívás segítségével képes visszaadni az adatokat, ezért ezt az elképzelést követtük mi is a fejlesztés során.

***Selenium***

A selenium segítségével tudjuk elérni az MTMT oldalon tárolt adatokat. Ezt úgy tudjuk megvalósítani ennek a technológiának a segítségével, hogy az alkalmazásban a seleniumot használva megnyitunk egy webböngészőt a háttérben. Ezt megában a kódban tudjuk beállítani, hogy elrejtve jelenjen-e meg. A keretrendszer több böngészőt is támogat, ezekre példa:

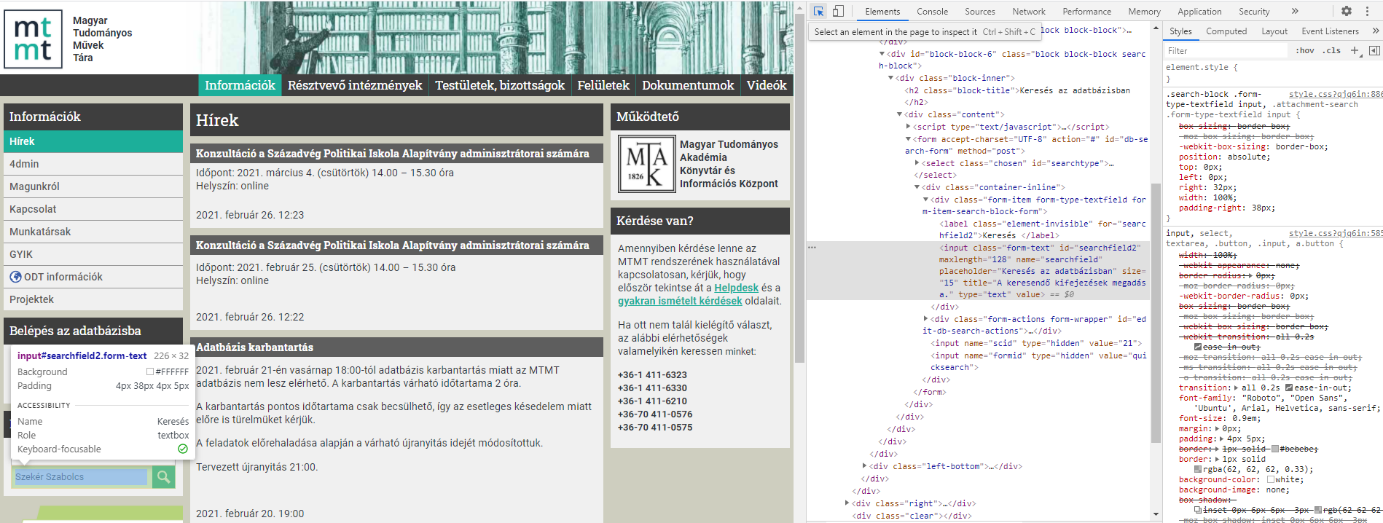
* Chrome
* Chromium
* Safari
* Firefox
* Opera
* Explorer

Mivel a Chrome a legelterjedtebb böngésző ezek közül, ezért mi is az ehhez tartozó driver-al implementáltuk ezt az alkalmazásban. A következő kód alapján tudjuk bekonfigurálni a háttérben futó web böngészőt.



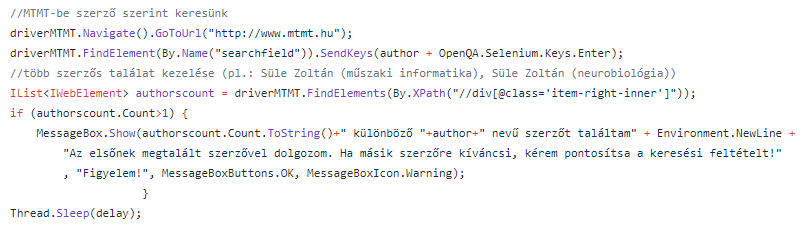
4. ábra - Selenium konfigurálása

Ezek után csak annyi dolgunk van, hogy utasítások sorozatával elnavigáljunk az MTMT oldalára. Itt már teljes mértékben hozzáférünk a weboldalhoz, mintha manuálisan rákeresnénk egy szerzőre, ezt utasításokban is kitudjuk adni az alkalmazásnak, ehhez csak arra van szükségünk, hogy megkeressük a HTML elemek közül azt a mezőt és a gombot, amire hivatkozva kiadhatjuk a parancsokat.



. ábra - Példa egy weboldal megfelelő részének megtalálásához a Chrome DevTools segítségével az MTMT oldalon

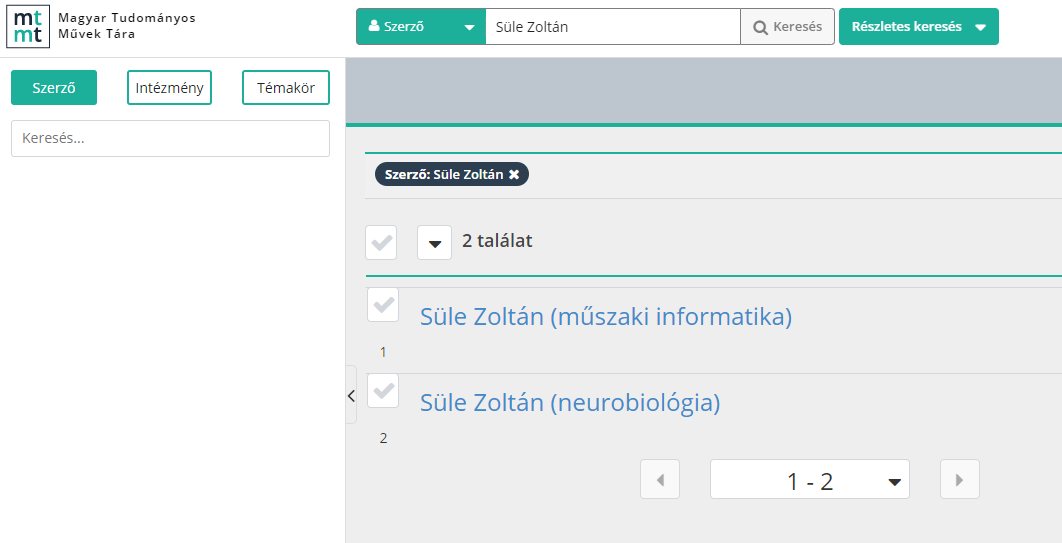
Ha megtaláltuk az általunk keresett elemeket, csak ki kell adni a parancsokat a seleniumnak.



6. ábra - Selenium működése

Látható, hogy a kódrészlet utolsó sorában a futó szálat késleltetjük. Erre azért van szükség, mivel a kiadandó parancssorozat túl gyorsan fut le. Ez több hibához is tud vezetni, akár az alkalmazás hibás futását is eredményezheti. A késleltetés mértékét egy változóban tároljuk, ennek a segítségével van finomhangolásra lehetőség. Mivel rengeteg dolog befolyásolja (hálózat, számítógép teljesítménye, stb), ezért ezt érdemes beállítani kezdetben egy magasabb értékre, így biztosan minden környezetben működni fog hiba nélkül az alkalmazás.

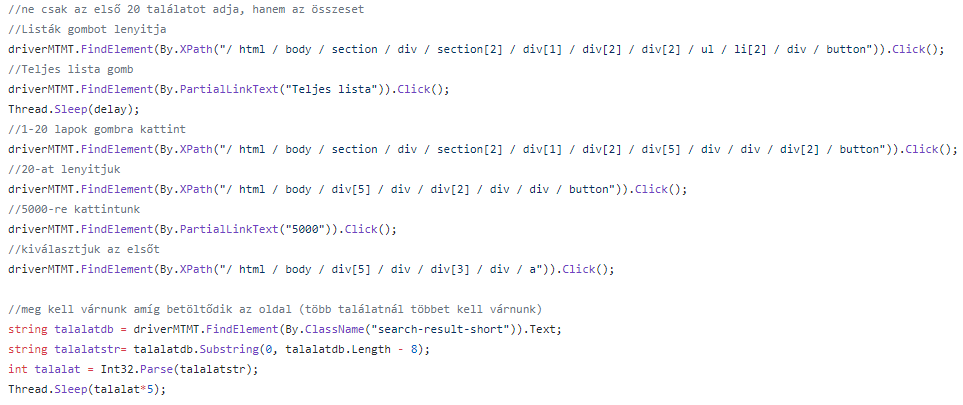
Ha eljutottunk arra a pontra, hogy elnavigáltunk a keresett szerző találataira, abba a problémába tudunk ütközni, hogy a keresett szerző nevével több találatot kapunk. Ebben az esetben az alkalmazás egy értesítő üzenetet dob, ami jelzi, hogy több találat van, ezek közül pedig az elsőt fogja kiválasztani. Amennyiben a rossz szerző adatait kérte le az alkalmazás, a keresés pontosításával könnyedén elérhető a másik szerző is.



7. ábra - MTMT azonos nevű szerzők

Alapvető hibakezelésekre is sor került az alkalmazásfejlesztés közben. Így letudjuk kezelni azokat a hibákat, amikor nincs a keresett szerzőnek profilja az oldalon. Ezesetben hibajelzéssel megáll az alkalmazás.

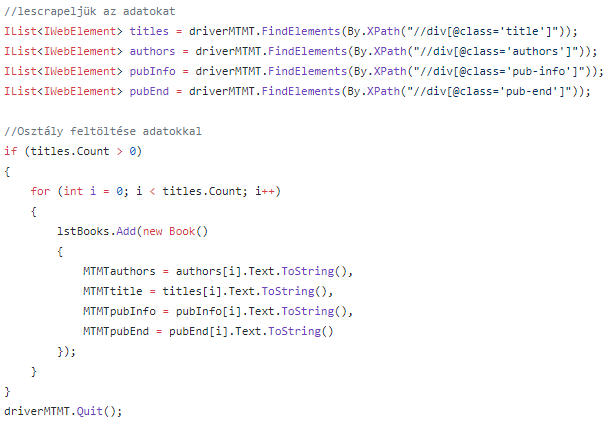
Amennyiben talált szerzőt, elnavigál a szerzőhöz tartozó publikációk listájára. Itt a következő apró probléma az volt, hogy alapvetően csak az első 20 találatot listázza ki a weboldal, ezért ezt is át kell állítani. Ezt a következő utasítássorozattal tudjuk megvalósítani.



8. ábra - MTMT listázási hibák kezelése

Ezen a ponton ismét, a futó szálat szüneteltetni kell a parancsok között, mivel itt a szerző publikációinak száma is nagyban befolyásolja a futás gyorsaságát.

Az alkalmazás ezen a ponton tudja már listázni a szerző összes publikációját. Nincs más dolgunk, mint az összes kilistázott elemből levegyük a szükséges információkat a selenium segítségével és feltöltsük a listánkat ezekkel.



9. ábra - Adatok lekérése és objektumok képzése

***Flask***

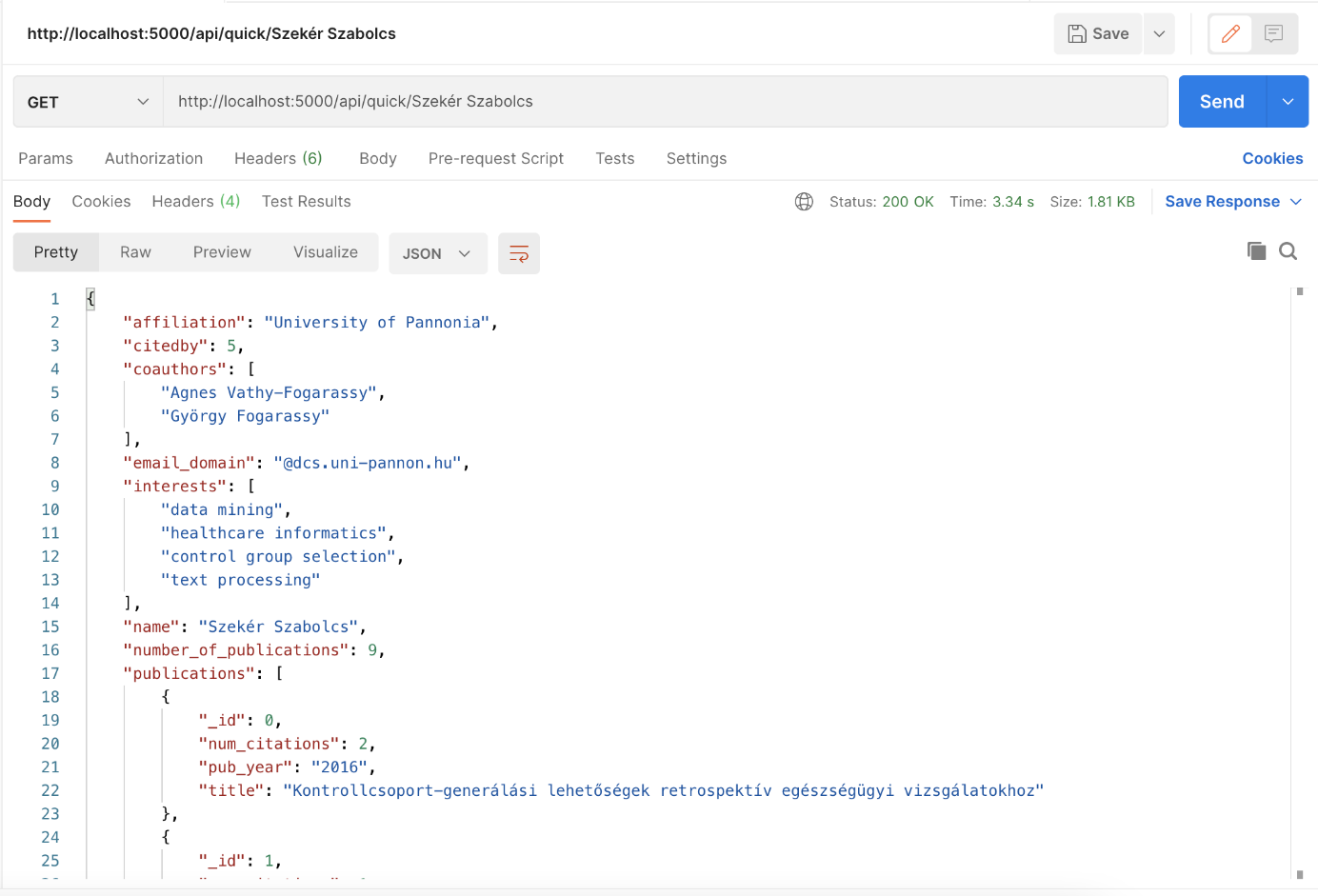
A Flask a Python nyelvhez készített webkeretrendszer, aminek a segítségével gyorsan és könnyedén lehet létrehozni webalkalmazásokat. Ennek a keretrendszernek az alkalmazásával könnyedén megvalósítható akár egy saját, lokálisan működő Google Scholar API.

Ennek a kettő könyvtárnak az összeillesztésével létre is hoztunk egy egyszerű működésű webszervert, ami három végponttal rendelkezik:

* /api/quick/author
* /api/quick/author/id
* /api/detailed/author

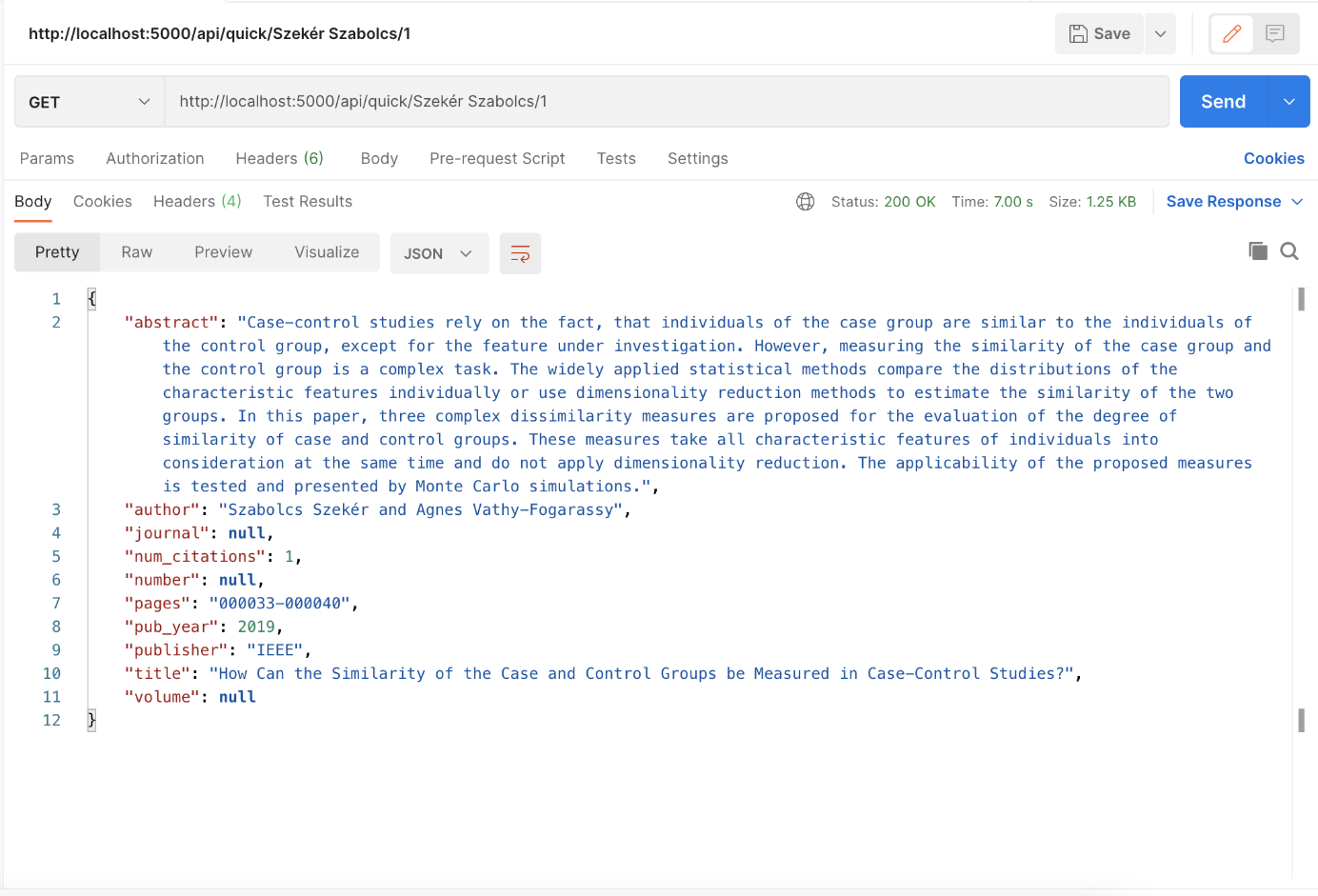
Ezen a három végpont mindegyikén egy GET kérés hívható. Az API működésének köszönhetően tetszőlegesen kérhetjük le egy szerzőnek az adatait. Ennek a megvalósításával már rendelkezünk egy olyan ingyenes API-val, ami teljesen személyre szabható. Egyetlen egy probléma maradt csak, hogy a scholarly nem JSON formátumban adja vissza a lekért adatokat, ezért ezt magában a python sciptben kell átalakítani. Az alapvető hibák kezelésére is sor került a scripten belül, mivel, ha egy olyan szerzőre keresnénk egy végponton, amire a scholarly nem talál eredményt, 500-as hibakóddal térne vissza a webszerver.

A szerver a /api/quick/Szekér%20Szabolcs kérésre az alábbi választ tudja visszaadni:



10. ábra - API GET request

Jól látható, hogy egy gyors keresés is már sok hasznos információt képes visszaadni. Egy ilyen kérés átlagos futási ideje 10 másodperc alá esik, amíg a részletes lekérdezés akár a többszörösébe is eltarthat. Természetesen az idő elteltével a lehető legrészletesebb információk érkeznek válaszként, de ennyire részletes információkra csak néhány esetben lehet szükség. A hosszú futási idők kiküszöbölésére jött létre egy harmadik végpont, amivel nem szükséges végig iterálni a kapott publikációk listáján és ezeket egyesével feltölteni, hanem lehet ID alapján lekérdezni a művek részletes információt. Egy ilyen lekérdezés az alábbi választ adja vissza:



11. ábra - API GET request publikáció részletek

***Integrálás***

Mivel egy Python webszerveres megoldás mellett döntöttünk, meg kellett vitatnunk, hogy ezt hogyan tudjuk megfelelő módon integrálni a jelenlegi C# alkalmazásunkba. Természetesen egy vállalat esetében (pl. Pannon Egyetem) nem lenne szükséges ezt integrálni magába az alkalmazásba, mivel könnyedén futtatható lehetne ezt egy belső hálózaton keresztül is, így könnyítve ennek elérését. A mi esetünkben sajnos erre nem volt lehetőség, ezért a webszervert egyben bele építettük az alkalmazásunkba. Ez egy szorosan csatolt megoldás, mivel a webszerver futtatásához előre definiált Python verzió használatára van szükség. A mi esetünkben a Python 3.9 -es verziója van használatban, de sok számítógép még a 2.7 -es verzióval lehet felszerelve, ami már jelentősen elavult, nem releváns. Tehát már kapásból szembesültünk egy függőséggel, ami mellé még csatlakoznak a package szintű függőségek. A webszerver kifejezetten a Google Scholar adatainak kinyerésére jött létre, amihez a scholarly könyvtárat használjuk. Tehát ha előzetesen nem telepítettük ezt fel, a szerver működőképtelen. Ez a package szintű függőség vonatkozik a Flask-re is, ami működteti az API-t. Ennek a könyvtárnak a használata nélkül sem tud működni a szerver, tehát ezt is előzetesen telepíteni kell.

Jól látható, hogy a szoftver ilyen téren, szorosan csatolt, mivel van több függősége is ahhoz, hogy megfelelően tudjon működni. Ezeket a hibákat a fentebb említett megoldással könnyedén ki lehet küszöbölni és ajánlott is (belső hálózaton való futtatás). Mivel nekünk nem volt lehetőségünk egy közös hálózatra kiengedni az API-nkat, ezért maradtunk az alkalmazásba integrálásnál. Mivel hárman dolgozunk egy projekten, könnyedén megtudjuk valósítani a környezeti függőségek beállítását.

Miután letöltöttük a Python legfrissebb verzióját (3.9) és feltelepítettük a nyelvhez tartozó package telepítőt, sok dolgunk már nincs, mivel a pip használatával másodpercek alatt telepíthetjük a függőségeinket. Csak a következő parancsok futtatására van szükségünk egy parancssoron belül:

* pip3 install scholarly
* pip3 install flask

Ez a két parancs fel is telepíti nekünk egyből a csomagokat.

***Integrálási alternatíva***

Alternatív megoldásként elkészült egy másik tárgy keretein belül az API-nak egy felhő alapú változata is, ami kiküszöbölné az integrálással keletkező szorosan csatolt problémákat. Mivel a felhőben fut az alkalmazás, ezért könnyedén elérhetővé válik az API használata a világ minden pontjáról. Ez a megoldás egy könnyen járható út lenne, de mivel a felhő szolgáltatások ára kifejezetten magas, ezért ez sem lenne egy költségoptimális megoldás. Tehát az alkalmazás továbbra is az alkalmazáson belül futtatja az API szervert.

***Integrálás megvalósítása***

A továbbiakban bemutatásra kerül mi módon sikerült integrálni a webszervert az alkalmazásba. Rövid kutatás után találtunk is megoldást, ami könnyen lehetővé tette ennek a megvalósítását. Mivel egy python script futtatása nem bonyolult feladat, akár egy egyszerű terminal paranccsal is végrehajtható, csak azt kellett megoldani, hogy a C# alkalmazás ezt automatikusan tegye megy. A megvalósítás egy Forms alkalmazásban kifejezetten egyszerű, mivel van egy alapból generált függvénye, ahol felépíti az alkalmazást, ebben a pontban kellene elindítani a webszervert, amit a következő metódus segítségével teszünk.



12. ábra - API szerver futtatása

Az alkalmazás ezen részében vannak „hard-code” -olva a különböző elérhetőségek. Látható, hogy van egy fix elérési útja magának a futtatandó python scriptnek, ami mellett megadjuk az elérését is a használandó python verziónak. Ha ezt a metódust megívjuk az inicializáló függvényünkben, elindítja egy háttérben futó shell-ben a webszervert. Egyetlen probléma ezzel a megoldással az volt, hogy amikor egy shell-t futtatunk ezzel a metódussal, azt várjuk, hogy a script a végére érjen, de mivel csak manuális leállítással érne végéhez a script (mivel egy szerver), ez a parancs fogja a fő szálat az alkalmazásban, tehát nem tud tovább lépni ezen a ponton. A probléma kiküszöbölésére egyszerű módon, egy külön szálon kezeljük a python szerver automatikus indítását. Ezt egy C# alkalmazásban akár egy sor kóddal is meg lehet valósítani.

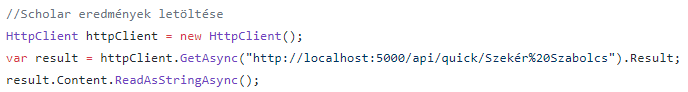


13. ábra - Párhuzamosítás

Így párhuzamosan és összhangban tud működni a C# és a Python alkalmazásunk is.

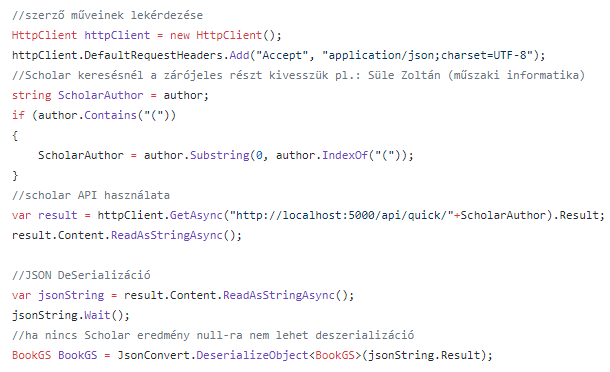
***Egységes kereső megvalósítása***

Most, hogy már automatizáltuk a webszerver indítását, egyszerű dolgunk van az adatok kinyerésével. A felület rendelkezik egy egyszerű beviteli mezővel és egy gombbal. A beviteli mezőbe bevitt értéket, ezesetben a szerző nevét fogjuk keresni az MTMT és a Google Scholar oldalakon. A gomb megnyomására indítjuk el a keresést, ami a beviteli mező értékét leszedi és tovább adja a két kereső metódusaiba. Az MTMT scrapeli egy háttérben futó böngészőben az adatokat, amíg a másik ágon ezt a beviteli értéket hozzá konkatenáljuk az API url-hez. Fent már említettem, hogy a végpontokon szerző nevére keresünk rá, tehát egy egyszerű API hívás C#-os környezetben az alábbiként néz ki.



14. ábra - C# HTTP request példa

Jól látható, hogy egy ilyen kérés megvalósítása sem bonyolult feladat, mivel a C# nyelv ezt meglehetősen jól és egyszerűen implementálta. Az így kapott result változó már tartalmazza a http kérés tartalmát JSON formátumban. A JSON adat felhasználása innentől kezdve már az alkalmazás feladata. A további API kéréseink az alkalmazásban mind hasonló módon történnek.

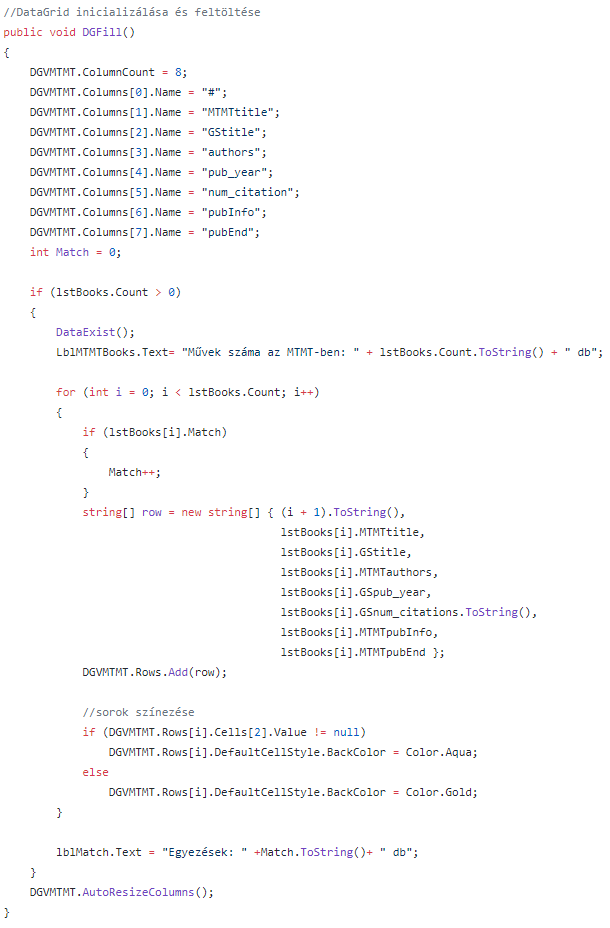


15. ábra - API request implementáció

Érdemes megemlíteni, hogy a JSON formátumban érkező adatok átalakítására és deszerializálására a Visual Studio fejlesztői környezet nagyszerű megoldásokat szolgáltat, így könnyítve a fejlesztő munkáját. Mivel általában az MTMT oldalon található publikációk a valósak, ezért a scholarly-val lekérdezett adatokat hasonlítjuk hozzá a seleniummal lekérdezett adatokhoz. A HTTP válaszban kapott adatok, ilyenkor már közös szerkezetre hozhatók az előzőlegesen kigyűjtött seleniumos adatokkal.

1. **Adatok kigyűjtése, találatok megjelenítése**

Az előző fejezetben láthattuk, hogy a publikációkat milyen módon tudjuk lekérni a két felületről, ebben a fejezetben ezek megjelenítéséről lesz szó. Mivel a lekért adatokat egy közös adatszerkezetre hoztuk a lekérést követően, így a megjelenítés kifejezetten egyszerű. Az alkalmazás felületét egy C# forms panel szolgáltatja. A felület legtetején található egy egyszerű beviteli mező, ahová a szerző nevét kell megadni. Van mellette egy kereső gomb, aminek a megnyomására hívja meg a két publikáció begyűjtő metódust a beviteli mezőbe bevitt paraméterrel. A rendszer kiegészül a lekért publikációkat tartalmazó DataGrid-el, ami egy olyan komponens, ami egy collection-t tud megjeleníteni. A publikációkat előzőlegesen egy listában tároltuk el, tehát a komponens adatforrásának ezt megtudjuk adni. Ebben táblázatosan jelennek meg a lekért adatok. A táblázatot az alábbi kód segítéségével tudjuk inicializálni és adattal feltölteni.



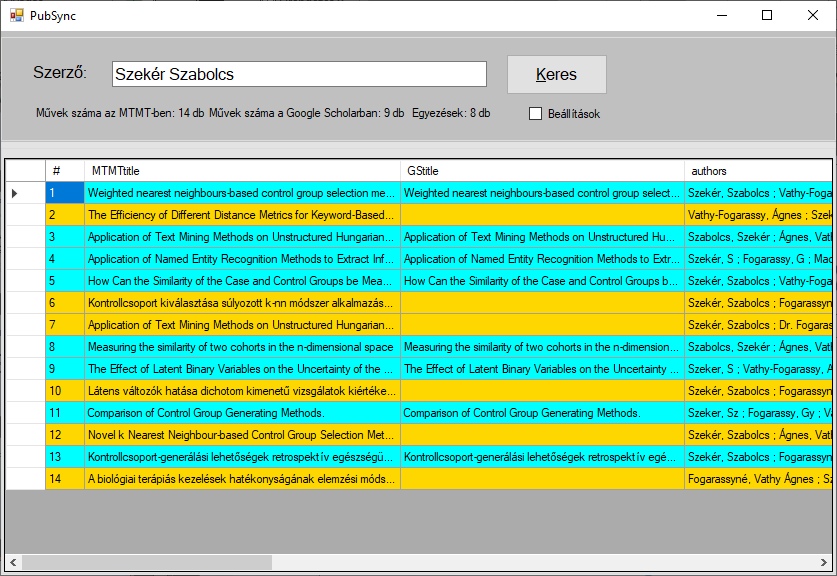
. ábra - DataGrid inicializálás és adatfeltöltés

A felület beállítások füle alatt két érték módosítására van lehetőség.

* Késleltetés
* Egyezés (kar.)

Az első beállítás felel a finomhangolásért (előzőlegesen említett szálmegállítás), a második beállítással az összehasonlítás mértékét tudjuk beállítani.

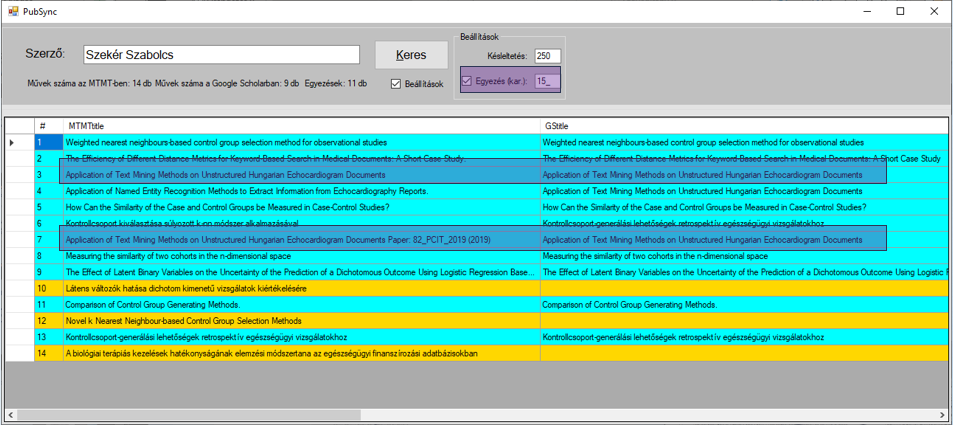
Egy label segítségével megjelenítjük szöveges formában, hogy hány művet találtunk a MTMT, és hány művet a Google Scholar felületén.



. ábra - Forms alkalmazás felülete

1. **Találatok összehasonlítása és különbségek keresése**

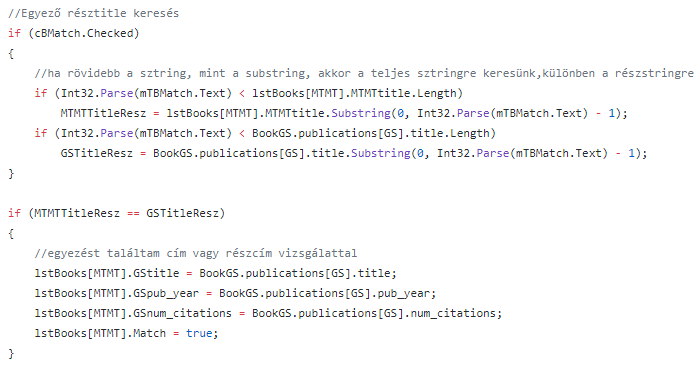
A talált elemek összehasonlítását az egyszerűség kedvéért külön színekkel tudjuk jelölni, így első ránézésre is megállapítható, hogy létezik-e eltérés a két publikációs lista között.



. ábra - Különbségek vizsgálata MTMT és Google Scholar eredmények alapján

Előző fejezetben említésre került, hogy a beállítások fül alatt állítható az egyezőség vizsgálata. Ez egy olyan egyszerű összehasonlítás, ami előre definiált értékig fogja összehasonlítani a talált publikációk címeit (az ábrán lila téglalappal jelzett területen megadható információ).

Ezt az alábbi kód segítségével hajtjuk végre.



. ábra - Cím összehasonlítás

Miután megbizonyosodtunk az eltérésről, nincs más dolga a felhasználónak, mint frissíteni manuálisan a Google Scholar profilját

1. **Automatikus/szoftverrel történő összehasonlítás előnye a manuálissal szemben**

A manuális módon történő összehasonlítás módszerével számos probléma felmerül: időigényes, költséges, körülményes és nagy a hibalehetőség. Ezzel szemben az automatizált összehasonlítás gyors, hosszú távon olcsóbb, hatékonyabb és maradéktalanul összehasonlít mindent.

1. **Összegzés, további fejlesztés**

Az alkalmazás kifejlesztése nagyban megkönnyítheti a publikációk MTMT és Google Scholar oldalak közti szinkronizációk problémáit, mivel automatikusan, jól láthatóan tudja összehasonlítani a lekért adatokat. A szoftverben való hibakezeléseknek köszönhetően folyamatos működést tud biztosítani, ezért az egymás utáni keresések is könnyen végrehajthatóak. Kellően személyre szabható az alkalmazás működése is, így apróbb hangolásokkal biztos működést tud biztosítani a végfelhasználó számára.

Mivel az alkalmazás egyben egy python webszervert is üzemel, ez szoros csatolást eredményez, aminek a kiküszöbölésére már említettünk alternatívát (Felhő szolgáltatások). Tehát egy valós projekt esetén, ez mindenképpen cél lenne. Tovább lehetne finomítani az összehasonlítást is, mivel jelenleg csak a publikációk címeit hasonlítjuk össze.

1. **Irodalomjegyzék**

<https://www.mtmt.hu/>

<https://www.mtmt.hu/az-mtmt-koncepcioja-es-korabbi-mukodesi-rendje>

<https://scholar.google.com/>

<https://www.scraperapi.com/blog/best-google-scholar-apis-proxies/>

<https://harzing.com/resources/publish-or-perish/command-line?fbclid=IwAR3JRm7kNry9EiC_oWtQZi7XRRljuzl8Vx3EddGLTlPhsNSvBNzAlgL6IP0>

<https://en.wikipedia.org/wiki/Web_scraping>

<https://www.selenium.dev/>

<https://pypi.org/project/scholarly/>