FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Deák Zsolt

Web scraping powershellel

Automatizált adatgyűjtés a világhálón

Konzulensek

Nguyen Gábor Loi

Dr. Goldschmidt Balázs

BUDAPEST, 2016

Tartalomjegyzék

[Összefoglaló 6](#_Toc462647071)

[Abstract 7](#_Toc462647072)

[1 Bevezetés 8](#_Toc462647073)

[2 Irodalomkutatás 9](#_Toc462647074)

[2.1 PowerShell, és képességei 9](#_Toc462647075)

[2.1.1 Invoke-Webrequest, Invoke-RestMethod 9](#_Toc462647076)

[2.1.2 Internet Explorer object 10](#_Toc462647077)

[2.2 Front-end 10](#_Toc462647078)

[3 Tervezés 11](#_Toc462647079)

[3.1 Funkcionalitás 11](#_Toc462647080)

[3.1.1 Back-end 11](#_Toc462647081)

[4 Összefoglalás 16](#_Toc462647082)

[5 Irodalomjegyzék 17](#_Toc462647083)

[6 Rövidítések jegyzéke 18](#_Toc462647084)

[Függelék 19](#_Toc462647085)

Ábrajegyzék

[ábra 1: Back end Use Case diagramja front end szempontból 11](#_Toc464308001)

[ábra 2: Back end Use Case diagramja 12](#_Toc464308002)

Hallgatói nyilatkozat

Alulírott **Deák Zsolt**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 10. 20.

...…………………………………………….

Deák Zsolt

Összefoglaló

Adva van egy erőteljes eszköz, a PowerShell, aminek első számú célja a rendszeradminisztráció automatizálása, megkönnyítése. Szintén adott a probléma, hogy az Interneten strukturáltalanul jelen levő információhoz hozzáférhessünk. Ebben a munkámban egy konkrét esettanulmányon keresztül bemutatom, hogy hála a PowerShell sokszínűségének egy alapfeladatától távol eső területen is hatékony megoldást lehet vele készíteni.

A példában a Használtautó.hu autóhirdetéseinek adatait nyerem ki és dolgozom fel (az ilyen eljárások gyűjtőneve a web scraping). A feldolgozás célja, hogy a site egy hiányosságát, az összehasonlító funkciót pótolja. Eredetileg egy az árukereső.hu azonos lehetőségéhez hasonló, a termékeket adataikkal együtt egymás mellett oszlopokban, táblázat formájában megjelenítő összehasonlítás volt a cél. Ehhez hasonló már elérhető a Használtautó.hu-n is. Ezt kiegészítettem egy rangsorral, amit az autók tulajdonságaiból számított érték alapján állítok fel. Mivel különböző korú és állapotú járművek összehasonlítása lineáris módszerekkel, néhány tulajdonság kiválasztásával még megközelítőleg sem ad valós képet, így ennek alapját egy általam kidolgozott egyszerű (és determinisztikus) algoritmus adja. A nagyjából azonos korú és értékű autók összehasonlítása láthatóan értékes információval is szolgálhat ennek segítségével. A példa teljessége érdekében létrehoztam egy egyszerű weblapot is, hogy online elérhető legyen a szolgáltatás. Ez utóbbi nem PowerShell nyelven van írva, hanem egy szokványos PHP és JavaScript alapú website.

Abstract

There is a strong tool for system administration given, the PowerShell. There is the problem to be able to access unstructured data and information laying around on the Internet. In this work I made a proof of concept to prove that PowerShell can be used efficiently for tasks very much different from its original usage, thanks to the variety of features it has.

The proof of concept is about processing data of Használtautó.hu’s car pages (which is called web scraping in general). This processing is focused on the car comparison functionality that is not present on the site. The idea is coming from árukereső.hu’s similar functionality, a table based, side by side comparator of products’ details. One very similar is already available at the target page. I improved this idea by ranking the cars based on their main parameters. Due to the ineffectiveness of linear methodologies in comparing cars of varied ages and conditions by a handful of data, I needed to develop my own simple (and deterministic) algorithm. This gives the base of the car ranking that can produce valuable information about cars of similar ages and prices. For the sake of completeness, I created a webpage for the service to be available online. This is a user friendly abstraction in place of the PowerShell command line, written in PHP and JavaScript.

# Bevezetés

# Irodalomkutatás

## PowerShell, és képességei

A PowerShell egyszerre képes kiváltani a hagyományos Windows Management Instrumentation Command-line-t és nyújt hozzáférést a .NET keretrendszerhez egy konvencionális script-nyelven keresztül. Lehetőség nyílik rajta keresztül objektumokat használni, függvényeket definiálni és rendelkezik a script-nyelvekre jellemző tömörséggel is. [1]

Alapvető kérdés, hogy az egyszerű, HyperText Markup Language (HTML) alapú weblapok automatikus feldolgozásához rendelkezésre állnak-e eszközök. Még pontosabban, hogy milyen lehetőségeket nyújt a PowerShell a Hypertext Transfer Protocol (HTTP) lekérdezésekhez és az általuk visszaadott adatok kezelhető struktúrába átalakításához. Verziótól függően több megoldást is nyújtanak az előre definiált könyvtárak. PowerShell 3.0 – tól elérhetőek az Invoke-Webrequest[2] és Invoke-RestMethod[3] függvények. Régebbi verzió esetén az Internet Explorer object[4] segítségével lehet elérni azonos eredményt.[5]

### Invoke-Webrequest, Invoke-RestMethod

Az Invoke-Webrequest és az Invoke-RestMethod nagyon hasonló metódusok, szembeszökő, hogy paraméterezésük megegyezik. Az első különbség, ami észrevehető, hogy a részletes leírás szerint az Invoke-Webrequest HTTP, HTTPS, FTP, FILE kérések küldésére alkalmas weblapoknak és webes szolgáltatásoknak, míg az Invoke-RestMethod vonatkozó részénél csak a HTTP, és a HTTPS kérések küldése szerepel Representational State Transfer (REST) web szolgáltatások felé. Ennek ellenére az Invoke-RestMethod Uri paraméterének leírásánál mind a négy féle kérés fel van tüntetve.

Különbség lehetne még, hogy Invoke-WebRequest esetén a UseBasicParsing paraméter dokumentációjából kiderül, hogy enélkül a paraméter nélkül a háttérben az Internet Explorer (vagy annak modulja) van használva a parse-oláshoz. Ugyanez az Invoke-RestMethodról nem mondható el biztosan, mivel a dokumentációja nem szól róla. Ez azonban azért nem döntő értékű, mivel azon a kijelentésen kívül, hogy „Indicates that the cmdlet uses basic parsing.”, vagyis hogy az alapvető parse-olás használatát jelöli ez a paraméter, azon kívül csak egy másik paraméter leírása van duplikálva itt.

A fentiek alapján a hivatalos dokumentáció nem elégséges annak eldöntésére, hogy melyik függvényt érdemes használni, így a tervezés szakaszban az irodalomkutatáson túlmutató kísérletek segítségével választom ki a megfelelőt.

### Internet Explorer object

Az Internet Explorer (IE) ComObject[7] egy valódi IE példány programozott irányítását teszi lehetővé. A Visible nevű tulajdonság segítségével állítható, hogy a folyamat közben látható legyen-e a böngésző, vagy se. Az első kézenfekvő hátránya annak, ha az ember egy valódi böngészőt akar automatizálni az, hogy nem erre fejlesztették eredetileg. Az IE esetében probléma lehet az oldalak betöltésének hatékonysága.

A navigációs Application Pogramming Interface (API)[8] egy nemblokkoló híváson keresztül szolgáltatja a funkcionalitását. Ennek következtében vagy egy eseményre való feliratkozással, vagy polling módszerrel lehet értesülni az oldal betöltésének végéről. A betöltés jelentősen lassabb lehet, mint a fent tárgyalt két esetben, mivel itt az összes forrás is betöltődik (képek, gifek, flash stb.), köszönhetően annak, hogy egy hagyományos böngészőről van szó.

## Front-end

A front-endre azért van szükség, hogy könnyebben fogyaszthatóvá, használhatóbbá tegye a programot. Kiküszöböli a parancssor használatának nehézségeit: nem kell külön engedélyezni a scriptek futtatását (vagy egy megbízható szervtől aláírást szerezni rájuk)[9], utánanézni a paraméterezésnek, elolvasni a dokumentáció egy részét. Minden úgy működik, ahogy azt a felhasználók már megszokták az évek során használt programokkal, weblapokkal. A visszakapott eredményeket is könnyebb megérteni, ha valamilyen szép vizuális megjelenítés van társítva hozzájuk, nem csak szimpla logok, vagy excel fájlok a kimenetek. Mindehhez nincs másra szükség csak egy egyszerű HTML alapú weboldalra és a legelterjedtebb[10] webes front-end keretrendszer megértésére és használatára.

# Tervezés

## Funkcionalitás

### Back-end

A back-end három fő és egy mellék komponensre osztható. Ezek a scaper, a rest service, a comparator és a funkcionalitásban részt nem vevő autó linkgyűjtő script. Először ezek funkcionális képességeit ismertetem a pontos architektúra és belső felépítés bemutatásának mellőzésével.

A funkcionalitás tervezésénél a használati eseteknek való megfelelés szempontjait járom körbe. Ehhez szükséges első sorban a használati esetek feltérképezése. Mivel a back end teljes egészében script alapú, ezért kétféle képpen lehet rá tekinteni: mint kiszolgáló a front endnek, vagy mint önálló applikáció. A back end tervezésénél bemutatom mindkét szempontot, viszont az egyes funkciókat csak egyszer, tehát a front end felé nyitott képességek nem kerülnek külön bemutatásra stand-alone szempontból.



ábra 1: Back end Use Case diagramja front end szempontból

A Unified Modeling Language (UML) 2 Use case diagramon látható a front end, mint aktor. A „valid POST request” nevű use case jelképezi a back end határát, a beérkező kérések feldolgozását. Ez a „kapott oldalak feldolgozása” használati esettel van asszociálva, ami maga a magja a szolgáltatásnak. Ennek a magnak részegysége az „algoritmus futtatása az adatokon” használati eset, mely minden esetben lefut (ezt jelöli az „include” stereotype a dependecia mellett), mégis kiemelendő. A Front end actor és a „kapott oldalak feldolgozása” logikailag össze lehetne kötve közvetlenül is, viszont a köztes használati eset hűebben tükrözi a valóságot, mivel van egy közbenső szereplő, egy proxy is a folyamatban, ami szigorúan véve a front end része, mivel ennek a domain-jében van, de nem képezi részét a kliens oldali alkalmazásnak.



ábra 2: Back end Use Case diagramja

A back end front endtől független működésének Use Case diagramja látható az ábra 2-n. A feldolgozási rész itt is ugyan úgy működik, mint az első esetben, ha az oldalak címei azonos formában rendelkezésre állnak már. Ehhez először a felhasználó meg kell adjon egy elérési utat, melyen egy text fájl található, ezt a back end feldolgozza és előállítja a szükséges formátumú URL listát.



ábra 3: Back end Use Case diagramja 2

A back end harmadik használati esete, mikor a felhasználó a scraper scriptet közvetlenül, az ábra 3-on ábrázolt „UseSaved” paramétert megadva indítja. Ebben az esetben, mint látható, a kívánt URL lista helyett közvetlenül az algoritmus futásához szükséges adatok képezik a „fájlok feldolgozása” use-case kimenetét.

A back endnek van egy kiegészítő funkciója is stand-alone használatkor: a felhasználó képes kigyűjteni általa egy, vagy több (autó) találati lista linkjeit. Ez akkor jelenthet előnyt, ha egy bonyolult preferencia rendszerrel meghatározta az elfogadható paraméterű autók körét, viszont a program segítségével meg akarja keresni a találatok közül a legjobbat, legjobbakat.

#### A scraper

Ez a script (skyscreper\_ie.ps1) képezi az egész projekt alapját, mivel ez végzi a weblapok automatikus feldolgozását, az adatok kinyerését. Háromféle teljesen szeparált működésre képes paraméterezéstől függően.

A legfontosabb, hogy képes egy URL, vagy egy URL tömb feldolgozására, melyet az Uri nevesített bemeneti paraméteren keresztül vár. A feldolgozás során vagy a megadott honlap adatait használja fel, vagy ennek egy aznap mentett (gyorsítótárazott / cache-elt) verzióját.

Másodsorban képes egy megadott elérési útvonalon lévő text állományból kiolvasni az URL-eket (soronként egy URL-t) és ezeken elvégezni a fent említett feldolgozást. Ehez a működéshez a Path paramétert kell használni a script indításakor.

A harmadik típus használatához a UseSaved paramétert kell megadni bemenő érték nélkül. Ennek a kapcsoló (switch) fajtájú paraméternek a jelenléte indikálja, hogy a mentett adatok alapján kell futtatni a programot. Így az eddigi futtatások során keletkezett adatokon fog lefutni a kiértékelés, melyek már nem HTML formában vannak tárolva, hanem xml (Extensible Markup Language) fájlokban.

Látható, hogy a három bemenetből kettő csak kényelmi szempontból szerepel, mivel ugyan azt a szerepet töltik be. Annyiban szerencsés egy fájlból beolvasó módot is alkalmazni, hogy így nem vagyunk ráutalva a pipeline használatára, ez fontos lehet a processek közötti kommunikációban, ha nem közvetlenül az URL-eket szolgáltató folyamat indítja a scriptet. Fordítva pedig hasznos a pipeline-ra hagyatkozni, ha közvetlenül indítható egy másik programból, vagy kézzel hívjuk meg a scriptet, mondjuk, ha csak egy bemeneti URL-t tartalmazna a fájl, amit beolvas. Ebben a két esetben az az elvárás, hogy a megadott webcímeken lévő adatokat beolvassa és átalakítsa a script programozottan kezelhető struktúrába. Az adatokat és a lapokat későbbi offline tesztelés céljából a script képes elmenteni.

A maradék esetben már strukturált adatok kerülnek visszaolvasásra, így nincs szükség átalakításra. Ezután mindhárom esetben véget ér a script futása, innentől az algoritmusnak kerülnek átadásra az adatok, illetve ennek visszatérési értéke továbbadódik a hívónak.

#### A REST service

A REST service (skyscraper\_rest\_service.ps1) azért lett létrehozva, hogy távoli eléréssel is lehessen futtatni a scrapert. Tehát online elérhetővé teszi a kulcsszolgáltatást, lehetővé téve, hogy a front end és a back end szeparálódjon egymástól. Szükségtelenné válik, hogy a front end szerver és a back end szerver ugyanazon a gépen, vagy virtuális gépen fusson. Ez nagy előnyt jelent az implementációban. A PoweShell 2016. augusztusig nem is volt elérhető, csak Windowson. Most már elérhető nyílt forrású GitHub projektként és használható Linuxon és OSX-en.[11] A REST service a 8089-es porton várja a kérések beérkezését. Ennek a működéséhez az összes tűzfalnak és hálózatbiztonsági berendezésnek (amely a gép és a nyilvános hálózat között található) engedélyeznie kell a bejövő és kimenő forgalmat ezen a porton.

A service egymással időben nem átlapolódó kérések kiszolgálására képes, tehát szekvenciálisan működik. Ez az elv működőképességéhez elegendő, viszont valódi termékben nem alkalmazható módszer, mivel gyakorlatilag kizárja a többfelhasználós működést. Ehhez a tényhez hozzájárul az is, hogy habár jelenleg a bejövő URL-ek száma tízben van maximálva, ezek feldolgozása akár több mint tíz másodpercet is igénybe vehet. Mivel az alapvető célkitűzés nem egy sokfelhasználós internetes szolgáltatás felépítése volt, ezért nem is lett rá hangsúly fektetve. Ennek ellenére a lehetőség adott, mivel a központi funkcionalitás könnyedén skálázható újabb PS processek indításával, tehát csak a REST service-nek kell erre, illetve a sokszoros kérések kiszolgálására. Ehhez azonban figyelembe kell venni megfontolásokat, mint load balancing[12], DDoS[13] támadások elleni védekezés, konkurens működés kezelése, közös erőforrás használat (a teljesség igénye nélkül).

#### Comparator

A comparator service (compare.ps1) tovább viszi egy lépéssel a web scraping elvét. A kinyert és immáron strukturált adatokat feldolgozza, majd visszatér a feldolgozás eredményével. A bemeneten egy kulcs-érték párokból álló objektumot vár (hashtáblák tömbje), kimenetként pedig egy ember által olvasható HTML szöveget ad vissza és ment el a lokális állományok közé. Az elmentett verzió mindig a legutolsó futtatás eredményeit tartalmazza, míg a HTML kódot azért kell vissza is adni, hogy egyszerű adatelérés legyen biztosítva a ráépülő szolgáltatásoknak.

#### Linkgyűjtő script

Ez a script ugyan nincs elérhetővé téve a REST endpointon keresztül, tehát nincs rá épülő front end szolgáltatás, viszont stand alone alkalmazásként használva a scriptcsomagot nagyon hasznos. Azt a használati esetet fedi le, mikor a felhasználó nem kifejezetten kiválasztott különböző márkájú, vagy évjáratú autókat akar összehasonlítani, amit biztosít maga a scraper és a comparator, hanem azt az esetet, mikor a felhasználó kihasználná az egymás mellé helyezés és rangsorolás adta lehetőségeket. Tehát például van egy felhasználó, akinek megvannak a preferenciái: adott márkát keres (Suzuki), azon belül adott típust (Swift), meghatározott korút (2010 és utána gyártott), és természetesen van egy büdzséje a vásárlásra (max 4 millió forint). Ezek alapján futtat egy keresést a Hasznaltauto.hu-n és kap egy eredményt (majdnem száz autóhirdetés).

Ezeknek az egyesével végigböngészése helyett a találati oldal URL-jét bemenetül adva a linkgyűjtő scriptnek az kimenti egy fájlba az összes hirdetés linkjét, amely fájl közvetlen bemenetként át lehet adva a scaper-nek. A scraper lefuttatása után pedig lesz egy rangsor a látszólag ugyanolyan, vagy nagyon hasonló autókról. Természetesen ez a rangsor is tartalmazhat azonos elemeket, viszont egyfajta szűrést mindenképp jelent, már csak a hirdetésekben található adatok mennyiségét tekintve is, mivel negatívabban bírál el olyan hirdetéseket, ahol valamely adatok hiányoznak.

A linkgyűjtő ezen kívül képes egy másik paraméterben átvenni a linkgyűjtés mélységét, vagyis, hogy hány lapozást végezzen. Lapozás alatt a találati lista következő 10 autója értendő. Ugyanígy, mikor megnyitja a kapott linket, a tíz autót megjelenítő változatot fogja értelmezni. Ha a megadott mélység túlmutat az oldalak számán, a script leáll az utolsó még valós lap linkjeinek mentését követően.

### Front end

# Összefoglalás

# Irodalomjegyzék

1. TechNet: Scripting with Windows PowerShell, <https://technet.microsoft.com/en-us/library/bb978526.aspx> (revision 15:03, 4 July 2016)
2. TechNet: *Invoke-WebRequest*, <https://technet.microsoft.com/en-us/library/hh849901.aspx> (revision 09:57, 10 September 2016)
3. TechNet: *Invoke-RestMethod*, <https://technet.microsoft.com/en-us/library/hh849971.aspx> (revision 09:58, 10 September 2016)
4. MSDN: *InternetExplorer object*, <https://msdn.microsoft.com/en-us/library/aa752084(v=vs.85).aspx#properties> (revision, 14:25, 12 September 2016)
5. PowerShell Team: *Controlling Internet Explorer object from PowerShell*, <https://blogs.msdn.microsoft.com/powershell/2006/09/10/controlling-internet-explorer-object-from-powershell/> (revision, 14:22, 12 September 2016)
6. TechNet: *about-Variables*, <https://technet.microsoft.com/en-us/library/hh847734.aspx> (revision 21:32, 8 September 2016)
7. JuanPablo Jofre (MSDN): *Creating .NET and COM Objects (New-Object)*, <https://msdn.microsoft.com/en-us/powershell/scripting/getting-started/cookbooks/creating-.net-and-com-objects--new-object-#creating-com-objects-with-new-object> (revision 19:42, 14 September 2016)
8. MSDN: *Navigate method*, <https://msdn.microsoft.com/en-us/library/aa752093(v=vs.85).aspx> (revision 20:26, 14 September 2016)
9. TechNet: *Using the Set-ExecutionPolicyCmdlet*, <https://technet.microsoft.com/en-us/library/ee176961.aspx> (revision 18:14, 17 September 2016)
10. Bootstrap: *Bootstrap index page*, <http://getbootstrap.com/> (revision 19:10, 17 September 2016)
11. PowerShell Team: *PowerShell on Linux and Open Source!*, <https://blogs.msdn.microsoft.com/powershell/2016/08/18/powershell-on-linux-and-open-source-2/> (revision 12:40, 19 September)
12. Wikipedia: *Load balancing (computing)*, <https://en.wikipedia.org/wiki/Load_balancing_(computing)> (revision 21:10, 25 September)
13. Wikipedia: *Denial-of-service attack*, <https://en.wikipedia.org/wiki/Denial-of-service_attack> (revision 21:06, 25 September)

# Rövidítések jegyzéke

|  |  |
| --- | --- |
| HTML | HyperText Markup Language |
| IE | Internet Explorer |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| REST | Representational State Transfer |
| API | Application Programming Interface |
| URL | Uniform Resource Locator |
| XML | Extensible Markup Langugage |
| PaaS | Platform as a Service |
| PS | PowerShell |

Függelék