



M Ű E G Y E T E M 1 7 8 2

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

I. Tartalomjegyzék

1 Bevezetés	7
2 Irodalomkutatás.....	9
2.1 PowerShell és képességei	9
2.1.1 Invoke-WebRequest, Invoke-RestMethod	9
2.1.2 Internet Explorer object	10
2.2 Front end	11
3 Tervezés	12
3.1 Funkcionalitás (felhasználói szint)	13
3.1.1 Back end	13
3.1.2 Front end	19
3.2 Funkcionalitás (rendszer és komponens szint)	21
3.2.1 Back end	21
3.2.2 Front end	22
3.3 Architektúra	24
3.3.1 Front end	26
3.3.2 Back end	27
4 Megvalósítás	32
4.1 Back end	32
4.1.1 Link gyűjtő.....	32
4.1.2 REST végpont.....	32
4.1.3 Scraper	33
4.1.4 Comparator	35
4.1.5 Tesztelés.....	36
4.2 Front end	40
4.2.1 Graphical User Interface	40
4.2.2 Proxy.....	42
4.2.3 Teszt.....	42

II. Ábrajegyzék

ábra 1: V- modell	13
ábra 2: Back end összefoglaló Use Case diagram	14
ábra 3: Back end alapvető Use Case diagramja	14
ábra 4: Back end Use Case diagramja	15
ábra 5: Back end Use Case diagramja 2	15
ábra 6: Front end használati esetek diagramja	19
ábra 7: Hibaüzenet példa.....	20
ábra 8: Front end – back end interakció.....	21
ábra 9: Entitások kapcsolatai	24
ábra 10: Rendszer-kommunikáció szekvencia diagram.....	25
ábra 11: Parse-olási sebesség eredmények	29
ábra 12: Oldal lekérdezés átlagsebességek	30
ábra 13: Példa a rossz paraméterezésre.....	34
ábra 14: Futtatás eredménye egy autó adataira	37
ábra 15: Futtatás eredménye 2 autóra	38
ábra 16: Futtatás eredmény egy része 150 autóra	39
ábra 17: GUI kezdőképernyő	40
ábra 18: GUI tesz pozitív eset.....	42
ábra 19: GUI teszt azonos bemenetek.....	43
ábra 20: GUI teszt hibás bemenet	43
ábra 21: GUI teszt üres bemenetek	43

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. 12. 08.

.....
Deák Zsolt

III. Kivonat

Adva van egy erőteljes eszköz, a PowerShell, aminek első számú célja a rendszeradminisztráció automatizálása, megkönnyítése. Szintén adott a probléma, hogy az Interneten strukturáltanul 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 nyerek 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 a rangsorolás alapját egy általam kidolgozott egyszerű (és determinisztikus) algoritmus adja, amely a nagyjából azonos korú és értékű autók összehasonlításakor láthatóan értékes információval szolgálhat. 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.

IV. Abstract

Given one of the most powerful system administration tools available, the PowerShell, capable of solving hundreds of problems with ease. Not only it can automate the everyday processes, but also, it can automate gathering information from the World Wide Web. In this work I made a proof of concept to prove that PowerShell can be used efficiently for tasks very different from its original purpose, thanks to the variety of features it has.

The proof of concept is about processing data of Hasznáلتó.hu's car pages (which is called web scraping in general). This processing is focused on the car comparison (ranking) 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. A very similar tool 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 features, I needed to develop my own simple (and deterministic) algorithm. This gives the basis 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 written in PHP and JavaScript in place of the PowerShell command line interface.

1 Bevezetés

A scraping arra szolgál, hogy adatokat tudjunk kinyerni website-okból. [1] Több fajtája és felhasználása létezik. Egyik legelterjedtebb felhasználása a keresőmotorok adatgyűjtő mechanizmusa, egy másik hatalmas terület a data mining. Ugyanakkor a scraping nem kizárólagosan automatizált adatgyűjtést jelent, lehet részben, vagy teljesen manuális is. Részben manuálisan lehet egy scraper tool felhasználásával, vagy mondjuk kézi weblap letöltéssel adatot gyűjteni, majd kiértékelni programmal. A diplomatervben általam vázolt megoldás teljesen automatikus, csak a célpont webcímet kell megadni neki.

Módszertől függően szükség lehet az adatok feldolgozására is, amelynek szintén több módja van. Természetesen történhet kézzel, vagy HyperText Markup Language (HTML) [2] parse-olással, pattern matchinggel és még sok más eszközzel. A módszerek kiválasztása nyílt kérdés még a területen, mivel nem sok elterjedt keretrendszer áll rendelkezésre. Például a legnagyobb online enciklopédia a Wikipedia csak egyet sorol fel, ez Python nyelven van megírva. A Python nyelv eszközeiben és tömör script mivoltában tökéletesen alkalmas ilyen feladatok megvalósítására. A PowerShell ezen a téren még nem kapott akkora nyilvánosságot, mint a Python, pedig hasonlóan tömör, kifejező nyelv, szintén jelentős eszköztárral (a teljes .NET keretrendszer kis megszorításokkal).

Jelen projekt fókuszában a scrapingben rejlő lehetőségek PowerShelles kiaknázásának egy aspektusa van. Az irodalomkutatásban elsősorban a technológia megszorításait járom körbe. Ezután a tervezésben végig veszem a felhasználási eseteket, meghatározom a követelményeket és megtervezek egy architektúrát a megvalósításhoz. Végül a megvalósításban bemutatom az implementáció kihívásait, fontosabb pontjait és a szükséges teszteseteket.

Akkor van értelme a scrapinggel foglalkozni, ha van egy terület, amin lehet alkalmazni az ötleteket, emellett kell legyen motiváció is a fejlesztésben. Mivel az autós világ trendjeit, árait, magas színről tekintett technológiáit követem a mindennapokban, ez a téma megfelelő motivációt nyújt, hogy a maximumot igyekezzek kihozni az alkalmazásból és a technológiából is.

Egy technológia felderítése öncélú, ha nincs mögötte semmilyen felhasználói igény, vagy üzleti szükség. Ugyanúgy ahogy az autók értékét igazából nem a paramétereik határozzák meg, hanem a vásárlók érdeklődése a paraméterek iránt, egy szoftver értékét is a használata tölti meg valós tartalommal, nem a gondos tervezés és megvalósítás. A használat persze nem garantált, de javítani lehet a valószínűségét, ha nagy a potenciális használók köre. A projekt természetesen jelen tárgyalt formájában még nem a valós felhasználást célozza, de alapot teremthet egy olyan alkalmazásnak, ami már célkozhatja. Fontos tehát felmérni a felhasználók körét, értelmet adva a programnak. Mivel egy magyar oldalt céloz az alkalmazás, ez már erősen korlátozza a felhasználók körét. Ezen belül azonban jelentős a köre azoknak, akiknek szüksége lehet a programra. Az első felhasználói kör a hirdető, akik el akarják adni autóikat, mivel érdekelheti őket, hogy a hasonló autókhoz képest milyen értéket képvisel az övé. Az oldalon lévő autóhirdetésekből kiindulva ez 95 000-nél is több, ami közt nyilván vannak kereskedők is, de megfelelő nagyságrendi alap. Emellett a Használtautó.hu indexe szerint [3] havi több mint 50 000 autó cserél gazdát Magyarországon (és a Használtautó.hu a legnagyobb magyar használt autó hirdető portál). Ők képezik azt a felhasználói csoportot, amely a legjobb döntést igyekeznek meghozni vásárláskor. Ilyen jelentős kiadások előtt feltehető, hogy minden megfontolást figyelembe vesznek a vásárlók. Ez teremti tehát alapot a projektnek.

2 Irodalomkutatás

Az irodalomkutatás célja megvizsgálni a technológiákat és felmérni a lehetőségeket, a rendelkezésre álló dokumentációk, cikkek alapján. Ez első sorban a témából adódóan a PowerShell releváns funkcionalitásainak kutatása, másod sorban a front end létjogosultságának és működési elvének bemutatása.

2.1 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. [4]

Alapvető kérdés, hogy az egyszerű, 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 [5] és Invoke-RestMethod [6] függvények. Régebbi verzió esetén az Internet Explorer object [7] segítségével lehet elérni azonos eredményt. [8]

2.1.1 Invoke-WebRequest, Invoke-RestMethod

Az Invoke-WebRequest és az Invoke-RestMethod nagyon hasonló metódusok, szembeszökő, hogy paraméterezésük megegyezik. Az első különbség, ami észrevehető, hogy az Application Programming Interface (API) [9] dokumentációk szerint [5][6] az Invoke-WebRequest és az Invoke-RestMethod névleg másféle szolgáltatásokkal kommunikálnak. Az előbbinél az szerepel, hogy web page-ekkel és web service-ekkel kommunikál, míg az utóbbinál, hogy RESTful szolgáltatásoknak [10] küldhet kéréseket. Ezen belül az egyes metódusok által támogatott protokollok a dokumentáció alapján:

Protokoll	Invoke-WebRequest	Invoke-RestMethod
HTTP	Támogatja	Támogatja
HTTPS	Támogatja	Támogatja
FTP	Támogatja	Nem támogatja
FILE (URI séma)	Támogatja	Nem támogatja

Ennek ellenére az Invoke-RestMethod Uri paraméterének leírásánál mind a négy féle protokoll fel van tüntetve.

Különbség lehet még, hogy Invoke-WebRequest esetén a UseBasicParsing paraméter dokumentációjából kiderül, hogy enélkül a paraméter nélkül a metódus a háttérben az Internet Explorert (vagy annak modulját) használja 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 a dokumentáció csak egy másik paraméter leírásának másolatát tartalmazza.

A fentiek alapján a hivatalos dokumentáció nem elégséges annak eldöntésére, hogy melyik függvényt érdemes használni, így a 3.3.2 szakaszban az irodalomkutatáson túlmutató kísérletek segítségével választom ki a megfelelőt. A kiválasztás során végül a legfontosabb szempont nem a paraméterezés (mivel a mindkettő által támogatott HTTP protokollt használom), hanem a sebesség.

2.1.2 Internet Explorer object

Az Internet Explorer (IE) ComObject [11] egy valódi IE példány programozott irányítását teszi lehetővé. A Visible nevű tulajdonság segítségével állítható, hogy a folyamat közben látható legyen-e a böngésző, vagy sem. Ugyanakkor kézenfekvő hátránya egy valódi böngésző automatizálásának, hogy a böngészőket eredetileg nem erre fejlesztették. Az IE esetében probléma lehet az oldalak betöltésének hatékonysága.

A navigációs API [12] 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.

2.2 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) [13], 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 fejlesztői szemszögből csak egy egyszerű weboldalra és a legelterjedtebb [14] webes front end keretrendszer megértésére és használatára.

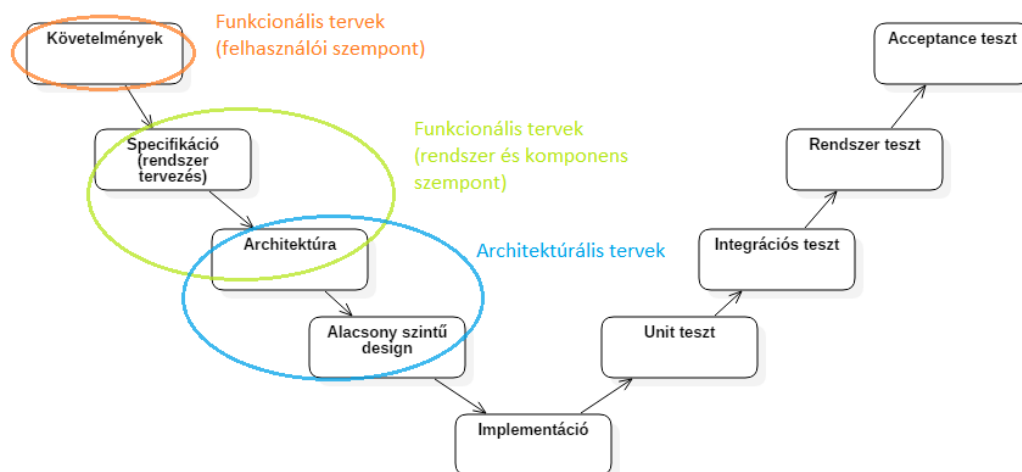
3 Tervezés

A back end három fő és egy mellék komponensre osztható. Ezek a scaper, a REST service, a comparator és a funkcionalításban részt nem vevő autó linkgyűjtő script.

Először ezek funkcionális képességeit ismertetem a pontos architektúra és belső felépítés bemutatásának mellőzésével, külön tárgyalva a felhasználói elvárásokat - az ennek való megfelelés kritériumait - és az egyéb funkcionális követelményeket, amelyek nem érintik közvetlenül a felhasználót (rendszer- és komponens szintű követelmények). A szétválasztás nem kizáró, vannak átfedések. Ezen felül felhasználói követelmények megvalósulásának szükséges feltétele a rendszer- és komponens szintű követelményeknek való megfelelés, ámde ez utóbbiak az itt tárgyalttól eltérő rendszerrel is képesek lehetnek kielégíteni az előbbit.

Ez után következik az architektúrális tervezés, amiben a hangsúly a komponensek és a rétegek összekapcsolásán van. Színén itt kerülnek bemutatásra az alacsony szintű tervezési döntések.

A használt stratégia leginkább a V-modell [15] bal szárához (fejlesztői életciklus) hasonlítható, azzal a különbséggel, hogy nem minden tervezési szint jelenik meg elkülönülten (1. ábra), így egy logikusabb és koherensebb terv készülhet. Mivel ez egy kis projektnek tekinthető, így az egyes perspektívák még nem válnak szét olyan élesen, mint egy nagyobb alkalmazásnál válhatnak. Még ez utóbbi esetben is sokszor mutatja a tapasztalat, hogy az ajánlástól való eltérés szükséges lehet a jobb átláthatóság érdekében. Így elkerülhetők a fontos részek kihagyása, vagy a sokszori előre és visszautalások más dokumentumokra, illetve esetemben fejezetekre. Ugyan magánál a fejlesztésnél az inkrementális iterációk módszerét használtam, tehát egyfajta agilis irányvonalat, viszont dokumentációs szempontból sokkal praktikusabbnak tűnt ez a fajta csoportosítás. Megvalósítástól függően változó lehet az egyes szintek definíciója, de talán a leginkább használtak a követelmények, specifikáció, magasszintű design (architektúra), alacsony szintű design.



ábra 1: V- modell

Az 1. ábrán látható, hogy az általam vázolt tervezési lépések hogyan feleltethetők meg a gyakorlatban használt egyik fajta V-modell szintjeinek.

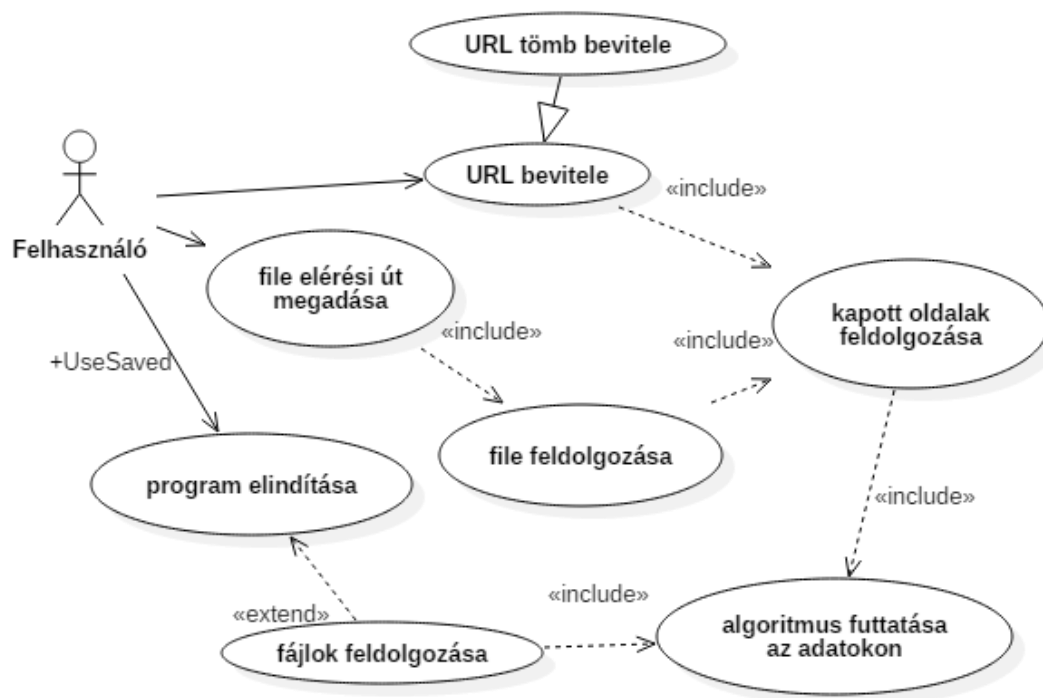
3.1 Funkcionalitás (felhasználói szint)

A funkcionalitás tervezésénél a használati eseteknek való megfelelés szempontjait járom körbe. Ehhez szükséges első sorban a használati esetek feltérképezése, majd az egyes komponensekre vetített követelmények meghatározása, melyek később az architektúra alapját adják.

3.1.1 Back end

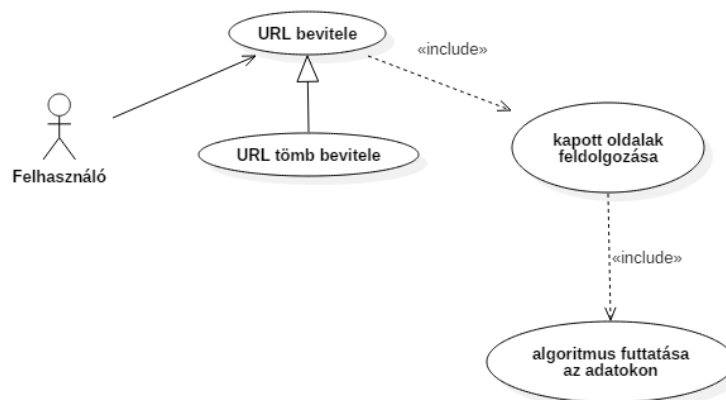
Mivel a back end teljes egészében script alapú, ezért kétféleképpen lehet rá tekinteni: mint kiszolgáló a front endnek, vagy mint önálló applikáció. A back end tervezésénél bemutatom mindkét szempontot. Ez a szakasz a felhasználói szempontok tervezését hivatott bemutatni, így a két réteg interakciója nem itt kerül tárgyalásra.

A 2. ábrán együtt szerepel az összes standalone use case egy diagramon.



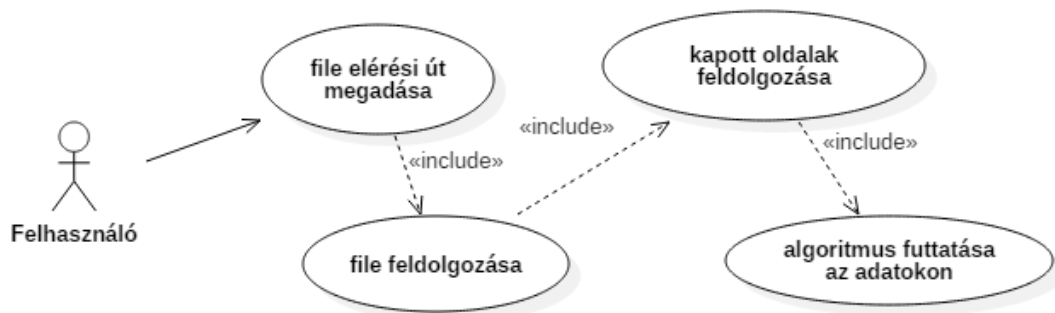
ábra 2: Back end összefoglaló Use Case diagram

Ebből az ábrából kiindulva a Unified Modeling Language (UML) 2 [16] Use Case diagramok segítségével bemutatom az egyes részeset-sorozatait az összefoglaló használati eset ábrának. Fontos látni, hogy a részesetek száma a tárgyalni kívánt részletességtől függ, definíció szerint lehet use case bármely folyamat vagy állapot sorozat, amely a felhasználó(k) számára értékes információt szolgáltatathat. [16] Így tehát funkcionális követelményeinek legjobb meghatározása végett egy viszonylag részletest diagramból indultam ki.



ábra 3: Back end alapvető Use Case diagramja

Az 3. ábra diagramján látható a felhasználó, mint aktor. Az aktor képes egy vagy több URL-t [18] megadni az alkalmazásnak (scriptnek), ezt jelképezi az „URL bevitele” és ennek a leszármazottja az „URL tömb bevitele” use case-ek. Az előbbi a „kapott oldalak feldolgozása” használati esettel van függőségben, ami a magja a szolgáltatásnak. Ennek a magnak részegysége az „algorithmus futtatása az adatokon” használati eset, mely minden esetben lefut (ezt jelöli az „include” stereotype a dependecia jel mellett), mégis kiemelendő, pont mint a „kapott oldalak feldolgozása” eset.



ábra 4: Back end Use Case diagramja

A back end front endtől független működésének egyik Use Case diagramja látható az 3. ábrán. A feldolgozási rész itt is ugyan úgy kell működjön, mint az első esetben, ha az oldalak címei azonos formában rendelkezésre állnak. 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 5: Back end Use Case diagramja 2

A back end harmadik használati esete, mikor a felhasználó a scraper scriptet közvetlenül, az 4. ábrán ábrázolt „UseSaved” paramétert megadva indítja (ez a Felhasználó és a program indítása közötti asszociáció neve is). Ebben az esetben, mint

látható, a kívánt URL lista helyett közvetlenül az algoritmus futásához szükséges adatok kell képezzék a „fájlok feldolgozása” use-case kimenetét, hiszen ez közvetlenül az „algoritmus futtatása az adatokon” esettel áll kapcsolatban, még hozzá részesete neki.

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.

3.1.1.1 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. Ehhez 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) [17] kiterjesztésű 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 harmadik esetben (mikor xml fájlokat használ a program) 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 tovább adódik a hívónak.

3.1.1.2 A Comparator

A comparator service (compare.ps1) túlmutat a web scrapingen, mivel nem csak kigyűjti a weben strukturálatlanul jelen levő adatokat, hanem a bemeneten érkező adatokat feldolgozza, majd visszatér a feldolgozás eredményével. A bemeneten egy kulcs-érték párokból álló objektumot vár (hash-táblák tömbje), kimenetként pedig egy ember által olvasható HTML szöveget ad vissza és ment el a lokális állományok közé is. 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.

3.1.1.3 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 lehet. Azt a használati esetet fedi le, amikor 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. 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 kimentti egy fájlba az összes hirdetés linkjét, amely fájl közvetlen bemenetként átadható a scraper-nek. A scraper lefuttatása után pedig kapunk egy rangsort a listázott autókról. Természetesen ez a rangsor is

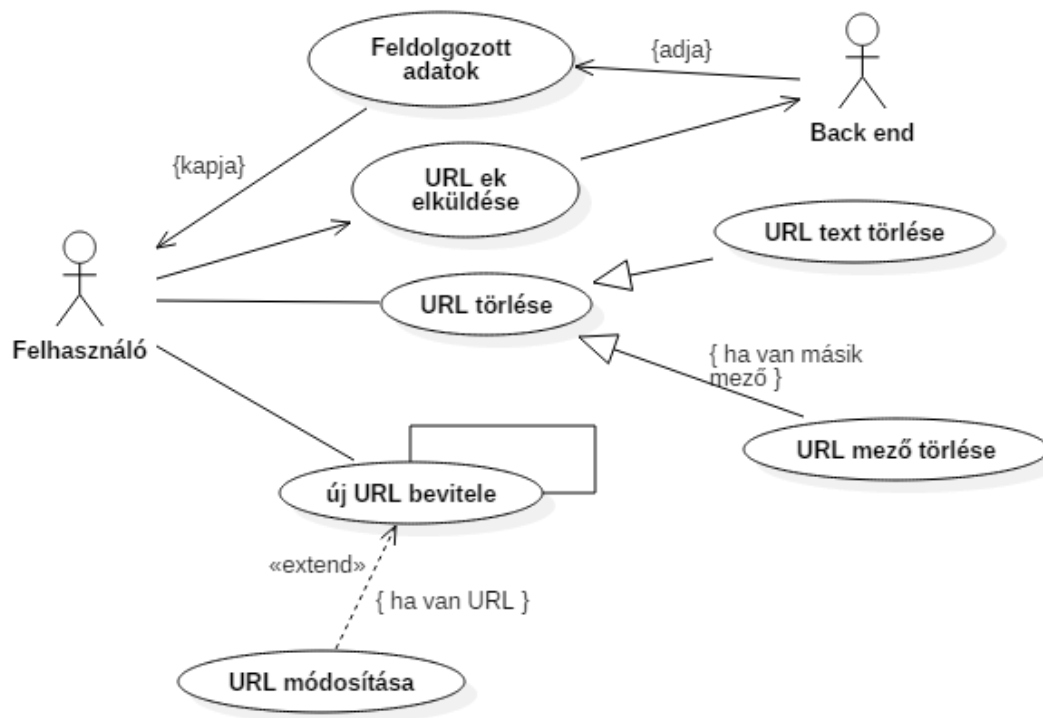
tartalmazhat azonos elemeket, viszont egyfajta szűrést mindenképp jelent, már csak a hirdetésekben található adatok mennyiségét tekintve is, mivel negatívabban bírál el olyan hirdetéseket, ahol valamely adatok hiányoznak.

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

3.1.2 Front end

A front end feladata kiszolgálni a fent részben már tárgyalt funkcionalitások egy részhalmazát, egy sokkal könnyebben használható, és ami még fontosabb, sokkal könnyebben elérhető módon. A program használatakor a felhasználó kizárólag a felhasználói felülettel (ez esetben Graphical User Interface, GUI [19]) áll interakcióban. Ezen kívül van még egy front end komponens, egy köztes réteg, egy front end domain-beli back end, amivel csak közvetve áll kapcsolatban a felhasználó.

Ez a közvetett kapcsolat az 5. ábrán az „URL ek elküldése” és a „Feldolgozott adatok” use case-ekben jelenik meg működési szinten. A „Felhasználó” nevű aktor el tudja küldeni az általa megadott URL-ek listáját a „Back end” aktornak, aki képes a „Feldolgozott adatok” használati eseten keresztül eredményt szolgáltatni. A felhasználó szintén ezen utóbbi használati eseten keresztül tud hozzáférni az eredményekhez. A fent említett négy kapcsolat irányított asszociációkkal van jelképezve az ábrán.



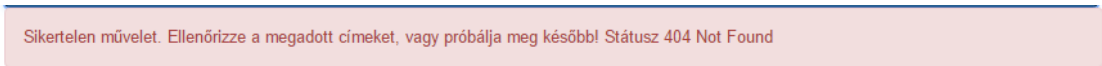
ábra 6: Front end használati esetek diagramja

Ahhoz, hogy az URL-eket el lehessen küldeni a back endnek, lehetőséget kell biztosítani a felhasználónak, hogy bevigyen URL-eket. Ezt jelképezi értelemszerűen az „új URL bevitele” use case, aminek van egy „URL módosítása” extensionje is, vagyis

nem minden bevitt követ (vagy része) egy módosítás, de van rá lehetőség, amennyiben teljesül a függőség mellé írt megszorítás, vagyis van már bevitt és nem törölt URL. Van tehát lehetőség törlésre is, két formában: lehet törölni az URL-t, vagy az egész mezőt, amely az URL-t tartalmazza. Utóbbi csak akkor lehetséges, ha legalább két mező van a képernyőn.

3.1.2.1 GUI

A GUI felhasználói szempontból képes kell legyen URL-eket beolvasni (maximum tízet) és lehetőséget kell biztosítson ezek feldolgozásra küldésére, módosítására, törlésére, ahogy ez az előző pontban felvázolt használati esetekben mint front end feladat szerepelt. Ezután a feldolgozás eredményéről képes kell legyen értesíteni a felhasználót. Ez az eredmény lehet a feldolgozásból származó adathalmaz, vagy hibaüzenet.



Sikertelen művelet. Ellenőrizze a megadott címeket, vagy próbálja meg később! Státusz 404 Not Found

ábra 7: Hibaüzenet példa

Mivel az egyetlen művelet, ami nem elhanyagolható eséllyel hibát eredményez, az az URL-ek elküldése, így csak ennek eredményeképp kaphat a felhasználó hibaüzenetet a GUI-n. Ez tartalmaz egy segítség részt lehetséges okokkal, illetve a művelet során visszakapott státuszt.

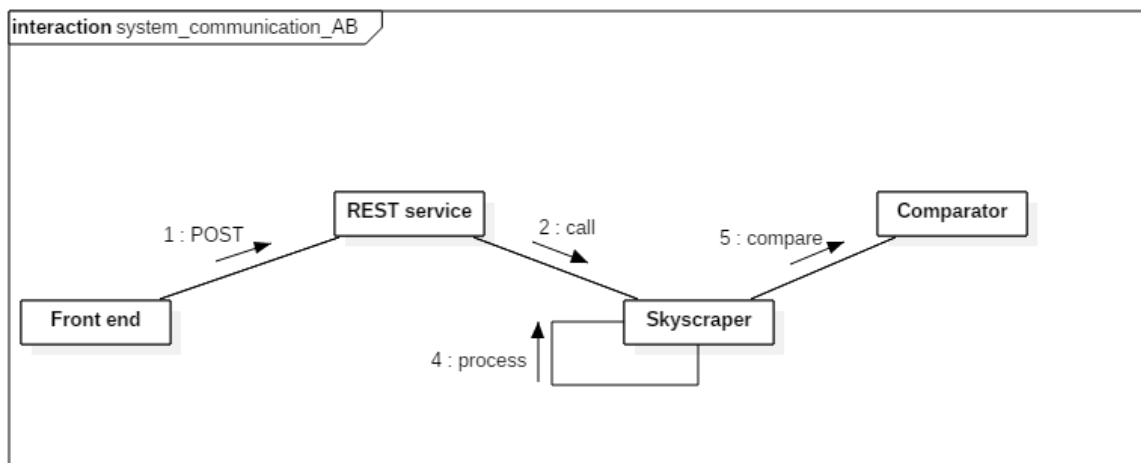
A GUI képes kell legyen a bevitt adatokat tárolni egészen addig, amíg a felhasználó be nem zárja a böngészője azon fülét, amiben a webalkalmazás fut. Erre user experience (UX) [20] szempontból van szükség, mivel ha a felhasználó mondjuk nem elég türelmes és az algoritmus futása közben a frissítésre kattint, elvesznének az addig bevitt adatai, vagyis legrosszabb esetben tíz URL-t is újra be kéne vinnie. Ráadásul, mivel ezek a szövegek nem túl olvasmányosak, nehezen megjegyezhetők, a felhasználó valószínűleg másolni fogja őket, tehát lehet, hogy ezzel az URL-ek újra kikeresését is szükségessé tenné az alkalmazás. Failure Mode and Effect Analysis (FMEA) [21] szempontból ez a veszély viszonylag gyakori (Probability), design tekintetében kritikus a bekövetkezés hatásának súlyossága (Severity), így ez a hiba kritikus (Risk Level), tehát a program késznek tekintésének (Minimum Viable Product, MVP [22]) szükséges feltétele ennek a hibalehetőségnek a kiküszöbölése (Mitigation/Requirements).

3.2 Funkcionalitás (rendszer és komponens szint)

Ez a fejezet hivatott összefoglalni a rendszerrel szemben támasztott mindazon funkcionális követelményeket, melyek nem felhasználói igényeket elégítenek ki, vagy nem közvetlenül. Ezek a követelmények többnyire jóval technikaibbak, mivel a rendszer és az egyes komponensek belső működését specifikálják.

3.2.1 Back end

A back end funkcionalitás, ha nem a felhasználó szempontjából tekintjük, rendszer szinten csak a front enddel kommunikál, két (és fél) rétegű alkalmazásról lévén szó.



ábra 8: Front end – back end interakció

A 7. ábrán látható ez a kommunikáció magas szinten ábrázolva, ez szemlélteti a funkcionalitást, amelyet a back end rendszer szinten hivatott ellátni. A Front end határát ebben az esetben a Proxy képezi. Mivel egy domainben vannak és ezen az ábrán nincs külön jelentősége a szétválasztásnak, ezért csak a „Front end” mint egyfajta ő, vagy összefogó entitás lifeline-ja szerepel. Egy POST request teremti meg a kapcsolatot a Back enddel, egész pontosan a REST service komponenssel. A PowerShell script révén inkább feladat orientált, mint objektum orientált, ezért is vannak az egybe tartozó nagyobb részek külön scriptek egységébe foglalva. Ezért van, hogy az ábrán a back end egyes egységei egy-egy entitásként szerepelnek. Szintén ezért van, hogy a „call” hívás nem a Skyscraperen meghívott call metódust jelenti, ahogy az hagyományos objektum orientált esetben jelentené, hanem, hogy a REST service meghívja magát a Skyscraper, elindítja azt. A Skyscraper önmagán belül elvégzi a process elnevezésű műveletet,

amely a 2. ábrán látott „kapott oldalak feldolgozása” use-case megvalósításának része, végül a Skyscraper meghívja a Comparator szolgáltatást.

3.2.1.1 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 különválhasson 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.[23] A REST service a 8089-es porton várja a kérések beérkezését. Ennek a működéséhez az összes tűzfalnak és hálózatzbiztonsági 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 (szinkron) működik. Ez az elv működőképességének bizonyításához elegendő, viszont valódi termékben nem alkalmazható módszer, mivel gyakorlatilag kizárja a többfelhasználós rendszert. Ennek ellenére a továbbfejlesztés lehetősége adott, mivel a központi funkcionalitás skálázható újabb PS processek indításával. Viszont ekkor figyelembe kell venni olyan megfontolásokat, mint load balancing [24], DDoS [25] támadások elleni védekezés, konkurens működés kezelése, közös erőforrás használat (a teljesség igénye nélkül), egy valóban jól működő multitenancy[26] rendszer kialakításához.

3.2.2 Front end

Az alkalmazásra, mint rendszerre tekintve a front end elsődleges szerepe a felhasználó számára absztrahálni a belső, bonyolultabb működést. Ennek a megvalósításához szükség van arra, hogy eljuttassa a GUI-ról gyűjtött inputokat a back endnek és vice versa. Ezt a rendszer szintű funkcionális követelményt valósítja meg a front end proxy (proxy.php) komponens.

3.2.2.1 Front end proxy

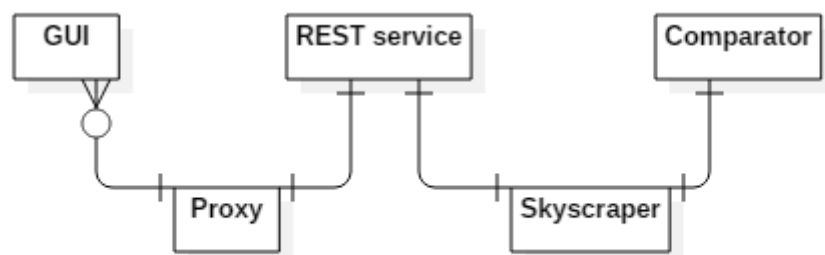
A front end proxy teszi lehetővé a front end és a back end különválasztását. A ma használt böngészők szigorú biztonsági protokollokkal dolgoznak, ezek közül az egyik a Same-origin policy [27]. Ez többek közt nem engedélyezi a különböző domainek közötti Asynchronous JavaScript and XML (AJAX) [28] hívásokat, melyek alapját képezik a back end – front end kommunikációnak. A probléma feloldására vannak szabványosított kikapuk, mint például a Cross-Origin Resource Sharing (CORS) [29] mechanizmus, ahol a szerver engedélyezheti egy header mezőben a same-origin policy-t sértő requesteket megadott domainek felől, viszont esetemben nem éreztem szükségét a biztonsági szabályok enyhítésének. Így szükségessé vált egy GUI-val azonos domainen lévő back end komponens, amely egyben az első szűrő, amit az elküldött adatokra alkalmazunk.

3.3 Architektúra

Ebben a részben a technikai megvalósíthatóság szempontjait és követelményeit járom körbe. Az eddig tárgyalt felhasználói igények teljesüléséhez feltétlen szükséges a megfelelő architektúra és a lehetőségekhez képest a legmegfelelőbb technológiák használata. A felhasználó igényeit maradéktalanul kielégítő szolgáltatás mellett törekednünk kell az egyszerű megvalósíthatóságra is. A back end esetében a témaválasztás miatt adott a technológia.

A front end technológiáinál fontos szempont, hogy a tanulási folyamat gyors legyen, mivel a fejlesztési időnek nagyobb az értéke a PS fejlesztésekor (és tanulásakor), lévén ez a munka központi eleme, ez a magasabb priorítás. Szintén elsődleges fontossággal bír, hogy egy megszokott felhasználói felület legyen az eredmény, így elérje a front end a célját, a hatékony absztrakciót. Egy nehezen használható rendezetlen felhasználói felület épp akkora hátrány egy programnál, mint ha csak parancssorból irányítható: mindkettőt meg kell tanulni, meg kell szokni, energiát kell befektetni ahhoz, hogy hozzáférjen a felhasználó a termék érdemi részéhez. A web mai állása szerint viszont a felhasználók az első néhány másodperc alapján eldöntik, hogy használni akarják-e a weblapot amire navigáltak, vagy sem. [30] A magas igények és a viszonylag alacsony ráfordítás (költségoptimalizálás) eredménye egy minimalista, letisztult design kell legyen.

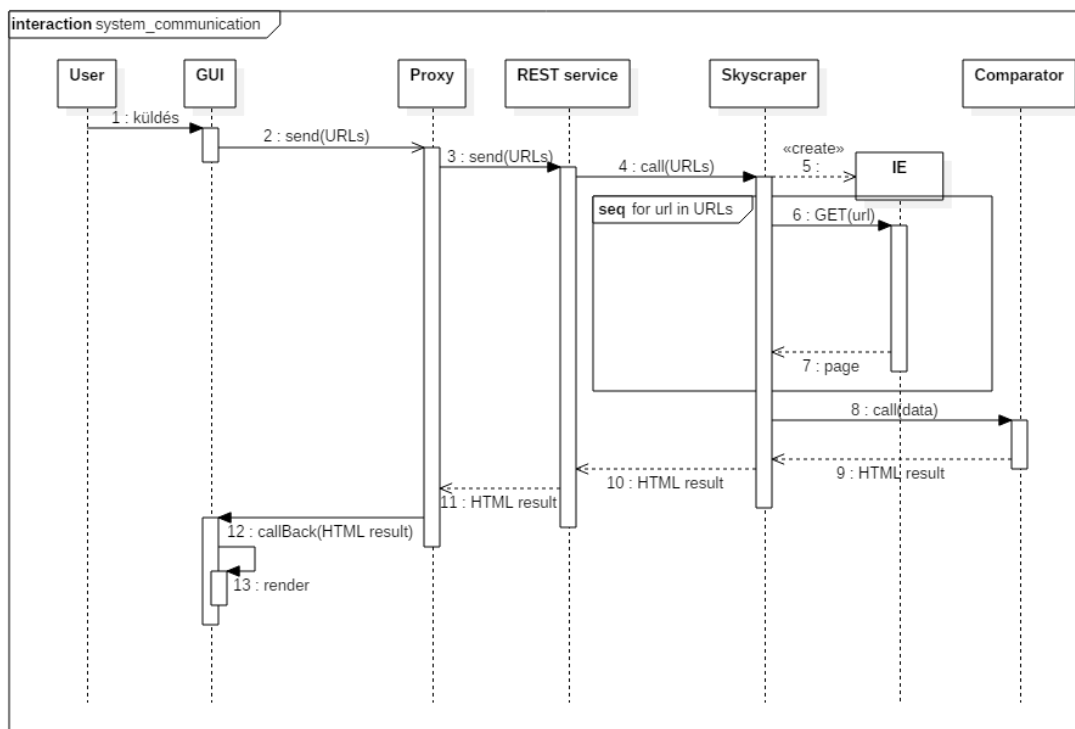
A rendszer működésének bemutatása előtt egy Entity Relationship diagramon vázolom a funkcionális követelményekben megfogalmazott entitások tervét, ezek kapcsolataival együtt.



ábra 9: Entitások kapcsolatai

Mivel a követelmények megengedik a szinkron működést, ezért az entitások között egy az egyhez hozzárendelés van kivétel a GUI és a Proxy között.

Az architektúra magas szinten a rendszer belső kommunikációinak kiszolgálására kell fókuszáljon. A funkcionális leírásból is viszonylag jól kivethető kommunikációs szekvencia megvalósításának tervét szemlélteti a 10. ábra. Ahogy már ott szerepelt a felhasználtól indul a folyamat a GUI-n kattintással („Elküldés”). A GUI elküldi egy POST requestben AJAX-on keresztül az URL-eket a Proxy-nak. Innen egy másik POST kérésben továbbítódnak az URL-ek a back end REST szolgáltatásához. A szolgáltatás a saját PS processén belül meghívja a Skyscraper-t, a paraméterek még mindig az URL-ek („Uri” nevesített paraméter). A szekvencia diagramon jól látszik, hogy ekkor következik a legidőigényesebb feladat, az oldalak adatainak lekérdezése GET requestekkel.



ábra 10: Rendszer-kommunikáció szekvencia diagram

Az oldalakból ismét nem elhanyagolható, ám a GET requestek futásánál jóval rövidebb idő alatt felépül az adathalmaz, ezt kapja meg a Comparator, még mindig ugyanannak a PS processnek egy újabb szintjén. Ez előállítja a kimenetet az adatokból, lefuttatva rajta az algoritmust. Az eredmény visszaszivárog a hívási láncon a Proxy-ig. Mivel az eredeti hívás a Proxy felé aszinkron volt, ezért a GUI egy callbacken keresztül értesül az eredményről, amit megjelenít (akár valós, akár error). Ez utóbbi két folyamat a használt diagram editor technikai akadályai miatt ilyen hosszú, ezzel szemben a

valóságban attól függ a hosszuk, hogy milyen gépen fut a böngésző. Normál körülmények között olyan elhanyagolható időt vesz igénybe, ami nem is megjeleníthető az ábra többi folyamatának méretéhez viszonyítva. Ugyanez igaz a hívások között eltelt időkre, csak a szemléletesség kedvéért vannak helyek hagyva. A lényegi információ a sorrendiség, az IE-rel való kommunikáció és az algoritmus futásának időigénye, a többi híváshoz képest, mivel itt több nagyságrend különbség is lehet.

Fontos kérdés, hogy a kommunikáció során milyen formátumban terjednek az üzenetek (az ábrán zárójelben írt paraméterek). Az URL-ek a lehető legegyszerűbb módon szöveg payload formájában utaznak végig a hívásokon. Egy sorban egy URL, maximum tíz URL. Visszafele pedig egy weblapon megjeleníthető valid HTML részlet jön, szintén text payloadként.

3.3.1 Front end

Az architektúra részleteit az adat áramlásával megegyező irányból közelítem meg. Az adatok a front enden születnek meg, mikor a felhasználó beviszi őket a szövegmezőkbe.

A felület az internetes trendeknek megfelelően HTML(5) alapú. Ez a nyelv strukturáltságával, széleskörű támogatottságával ideális alap, jól kombinálható technológiák egész tárházával. A felület szerkezete tehát ezen a nyelven kell elkészülni. Ezzel a funkcionális elvárások lefedhetők lennének, ez azonban igaz a command line alapú felhasználói felületre is. A front end létrehozásának egyik legfontosabb célja az alkalmazás használatának megkönnyítése. Ha a felhasználók számára a legegyszerűbben használható és leginkább kellemes élményt nyújtó felület a minden designt nélkülöző HTML oldal lenne, a vállalatok nem ölnének milliárdokat újabb designok és effektek kitalálására. Az általános architektúrális követelményeknek való megfelelést ez az egy specializált alkövetelmény tehát nem teszi lehetővé. A felületnek szüksége van dinamikára és elfogadható kinézetre.

A dinamika technológiája részben szintén adja magát. Manapság egy HTML weboldal szinte kizárólagosan JavaScriptet (JS) [31] használhat a kliens oldali műveletekhez. JS-en belül igen sok keretrendszer áll rendelkezésre, különböző módokon téve egyszerűbbé a kódot és a fejlesztést. Egy ilyen vékony réteg esetén ezekre külön időt fordítani overengineering, architektúrális szempontból tehát irrelevánsak.

Nagyjából hasonlóak igazak a kinézetre is. HTML külső formázásra egyértelműen Cascading Style Sheets-et (CSS) [32] használ a webfejlesztő társadalom. Az erre épülő keretrendszerek szintén nagy számban vannak jelen az interneten. Sok közülük szabadon használható, jól dokumentált, egyszerű példákkal illusztrált, elterjedten használt. Ezek közt vannak kiemelkedők, mint a jQuery UI [33], ami főleg a felület dinamikusságát könnyíti meg és előre definiált elemeket (widget) nyújt, a Foundation (pl Pixar weblap designja), vagy a Bootstrap [14] (Twitter fejlesztés). Ezek közül a Bootstrap tűnik legjobban támogatottnak és a legegyszerűbbnek is használat szempontjából. Követve a leírásokat, egy ilyen vékony front endet ezzel lehet a leghatékonyabban felhasználó-barát kinézetűvé alakítani, miközben látható a referencia oldalakon, hogy a jövőbeni terjeszkedésnek, fejlesztésnek is teret enged. Mindezt úgy teszi lehetővé, hogy magát a CSS-t jóformán nem is kell használni, csupán az általuk definiált kinézeti elemeket kell megfelelően kombinálni az elemek class attribútumain keresztül.

Az adatáramlás következő megállója a Proxy komponens. Ez a komponens egy viszonylag egyszerű funkcionalitást lát el, így tetszőleges back end technológián megvalósítható. A kiválasztás szempontja ezek alapján a támogatottság és elterjedtség kell legyen, így a választás egyértelműen a PHP-ra [35] esett, elsősorban az elsöprő többségű piaci részesedése [36] miatt, másodsorban amiatt, hogy ha valóban a nyilvánosság számára elérhető weblap a cél, a PHP gyakorlatilag minden PaaS [37] szolgáltatónál elérhető (future-proof design).

3.3.2 Back end

A back endnél a technológia adott, a PowerShell. Néhány architektúrális szempontot azonban itt is figyelembe kell venni. A legfontosabb, hogy egyáltalán képes legyen helyes működésre a back end, ha egy PS-lel (Windows Management Frameworkkel) rendelkező operációs rendszeren, megfelelő beállításokkal próbálják futtatni. Az Irodalomkutatásban már megjegyeztem, hogy a REST hívás csak PS 3-tól került be a standard API-ba. Ennek következményeképp lehetséges, hogy egy Windows 7-es operációs rendszer alapértelmezésben még nem rendelkezik a megfelelő könyvtárral (például a clouds.bme.hu virtuális gépe [2016.11.07]), így az IE objektumot használja a dokumentum lekérdezéshez. A visszakapott dokumentumnak ez esetben más

a típusa, mint a WebRequest által visszaadottnak. Ezen kérdések kezelésére a programot fel kell készíteni.

Egy másik fontos szempont a biztonság. Habár az International Software Testing and Qualification Board (ISTQB) [38] szerint a biztonság funkcionális jellegű, ettől a standardtól eltérnek, mivel a funkcionális terveknek inkább az egyes elemek használata volt előtérbe helyezve és ezek nehezen egyeztethetők össze egy ilyen technikai jellegű követelménnyel. Ha a felhasználó mint alkalmazást használja a scripteket, akkor csak magának tud ártani ráadásul rendelkezik a kóddal, bármilyen biztonsági intézkedést felülbírállhat. Tehát csak a front end felől és felé kell garantálni a biztonságos működést. A funkcionális követelmények garantálják a minimális szükséges hozzáférést engedélyezését a külvilág felé. Az adatok egy ponton érkezhettek be és ugyanott küldhetők ki. A rendszer szinkron működik, így temérdek biztonsági kockázat megszűnik, többek közt a párhuzamos folyamatok biztonságos izolációja, vagy a DDoS támadások. Ugyan egy rosszindulatú harmadik fél le tudja foglalni a számítási kapacitást így is, ennek kiszűrése azonban jóval túlmutat a projekt keretein. A rendszer szintén nincs felkészítve a man in the middle jellegű támadásokra. Fel kell legyen készülve azonban az injection jellegű támadásokra, mivel ezek nem csak a szolgáltatás kimaradását, vagy egy-egy felhasználónak adott rossz válaszokat eredményezhet, hanem az egész back end környezet meghibásodását, rosszabb esetben véglegesen. Ennek elkerülésére a bemeneten érkező URI-kat [39] whitelistelni [40] kell, vagyis csak a megengedett típusú (potenciálisan hasznaltauto.hu autó weblap) címek érkezhetnek meg feldolgozásra.

Harmadrészt fontos nemfunkcionális szempont a rendszer sebességét leginkább szolgáló megoldások használata. Az architektúra áttekintésénél a szekvencia diagramon látszott, hogy a lapok lekérdezése és parse-olása a sebesség szempontjából legkritikusabb szakasz. Ezek optimumának megtalálására tehát külön mini projekteket kell létrehozni, a megfelelő tervezés lefolytatásához. Az egyik ilyen a parseSpeedTest.ps1.

A parse-olási sebességek összehasonlításánál az első lépésben letöltöttem egy dokumentumot, az új verziókban használatos Invoke-WebRequestrel. Meghívhattam volna bármelyik hasonló, rendelkezésre álló metódust is, de ezt nyújtja az újabb API és ez van a legjobban ledokumentálva (mi a visszatérési érték, mik a paraméterek).

A betöltött dokumentumot először a pipeline használata nélkül dolgozom fel:

```

$tables = $doc.ParsedHtml.GetElementsByTagName("TABLE")
ForEach($item in $tables){
    if($item.className -eq "hirdetesadatok"){
        $elements = $item
    }
}

```

Ez a kódrészlet először leszűkíti a lap elemeit (node-jait) táblázatokra, majd ezek közt megkeresi azt a táblázatot, amely a hirdetés adatait tartalmazza. Ezt a folyamatot tízszer megismételve az időt átlagolva kijön egy nagyságrendi viszonyítási pont.

Ezután a dokumentumot ismét tízszer beparse-olja a pipeline-t használó verzió is:

```

$elements = $doc.ParsedHtml.GetElementsByTagName("TABLE") | Where-Object
className -eq "hirdetesadatok"

```

Végül csak az összehasonlítás végett az IE parse-olását kikapcsolva is lefut ugyanaz, azzal az egy szembetűnő különbséggel, hogy itt nekem kell gondoskodni a „parse-olásról”, amit egy egyszerű reguláris kifejezéssel valósítok meg.

```

$doc = Invoke-WebRequest -Uri $uri -Method Get -UseBasicParsing
[regex]$regex = '<table class="hirdetesadatok">(.*?|\r|\n)*</table>'
For($i = 0; $i -lt $maxIter; $i++){
    ...
    $elements = $regex.Match($doc.Content).Value
    ...
}

```

Itt is átlagolódik az idő, majd kilép az alkalmazás. Meglepő módon a következő eredményeket kaptam:

```

Non-pipe parsing average time = 0.07359195
Pipe parsing average time = 0.20445136
Non-pipe parsing average time with BasicParsing ON = 0.00080205
Done
3.9226779 seconds elapsed

```

ábra 11: Parse-olási sebesség eredmények

Az első két módszer között közel háromszoros az eltérés, továbbá látható, hogy a parse-olás a rosszabbik esetben is csak néhány tizedmásodperces nagyságrendbe esik. A harmadik eset szembeszökően gyorsabb volt a többinél.

A másik sebességet mérő script a `getPageSpeedTest.ps1`. Ennek hivatása eldönteni melyik web lekérdező megoldás a leggyorsabb. A mérési elv hasonló az előzőhöz, annyi különbséggel, hogy itt nem szeretném, ha esetleg torzítaná az eredményeket, ha valamelyik megoldás a háttérben cache-elne az adatokat, így itt annyiszor fut le a ciklus, ahány URL-t kap a script a bemenetén. Szintén különbség,

hogy jelentős késleltetés tettem az egyes lekérdezések közé, hogy véletlenül se tűnjön rossz szándékú próbálkozásnak a teszttem.

Az első a megszokott WebRequest:

```
$doc = Invoke-WebRequest -Uri $uri -Method Get
```

A második a RestMethod:

```
$doc = Invoke-RestMethod -Uri $uri -Method Get
```

Ennek az előbbi kettőnek a dokumentációjuk alapján nemigen kéne eltérniük. A harmadik módszer azonban egy teljesen más stratégiával dolgozza fel az adatokat, így itt már várható különbség.

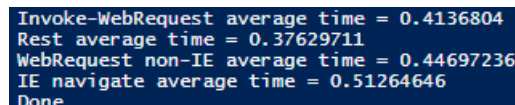
```
$doc = Invoke-WebRequest -Uri $uri -Method Get -UseBasicParsing
```

Végül pedig az IE objektum használatával töltöm le a dokumentumot:

```
$ie.Navigate($uri)
$i = 0
while ($ie.busy) {
    Start-Sleep -Milliseconds 10
    $i++
    if($i -ge 300) {
        Write-Host "Navigation timed out" -ForegroundColor Red
        Continue
    }
}
$doc=$ie.Document
```

A pollingos várakozásra aszinkron callback hiányában van szükség. Enélkül akkor is el lehet kérni a Documentet, ha az még üres.

Ez a kísérlet hasonló futási időket eredményezett a három módszer esetén, így azok sorrendje kevésbé egyértelműen eldönthető, mint az előző kísérletben:



```
Invoke-WebRequest average time = 0.4136804
Rest average time = 0.37629711
WebRequest non-IE average time = 0.44697236
IE navigate average time = 0.51264646
Done
```

ábra 12: Oldal lekérdezés átlagsebességek

Erről az ábráról is azt lehet leolvasni, ami többszöri futtatás után is kijött. Az átlagos sebesség hozzávetőleg egy tizedmásodpercen belül van minden módszer esetén, tehát nem jelentős a különbség (a példában 5 URL volt a bemenet). Annál inkább igaz, hogy az egyes módszerek átlagos futási ideje is nagyjából egy tizedmásodperces sávon belül változik.

A fenti két kísérlet alátámasztja a szekvencia diagramon látott időviszonyokat, valamint az is kiderül belőlük, hogy még a parse-olás ideje se szignifikáns a weblapok letöltéséhez képest. A praktikai szempontokat figyelembe véve az Invoke-Webrequest a legjobb választás a REST hívásokhoz. A parse-oláshoz az átlagos futási idők alapján a UseBasicParsing-ot használó verzió lenne a legjobb. Ellene szól azonban, hogy túlzottan specializálja a scrape-elést erre az egy weboldalra, illetve hatalmas regressziós lehetőségeket rejt magában és plusz komplexitást visz a rendszerbe. Nem elég ugyanis egy reguláris kifejezéssel kinyerni a megfelelő TABLE tag kódját, ebből a kapott HTML részletből ki kell bontani a belső szöveges tartalmat. Ha automatikus parse-olást alkalmazok, csak néhány elemre kell hagyatkoznom a HTML Document Object Model struktúrát tekintve. Ehhez hozzátéve, hogy habár sokkal gyorsabb a „kézzel” parse-olás, a teljes futási időhöz képest nem jelentős a nyereség. A koncepció, hogy ez a nyelv is alkalmas a scrapingre elvesztené a lényegét, ha egy nyelvi szinten generális (más nyelveken is könnyen megvalósítható), viszont csak egy adott feladatra alkalmazható megoldást választanék. Így a felhasznált technika a második leggyorsabb parse-olás, a pipeline-t nem felhasználó feldolgozás lett az új PowerShellt támogató futtatás esetén.

4 Megvalósítás

Adva vannak az irányelvek, a felhasználónak való megfeleléshez. Meg van határozva az MVP eléréséhez szükséges feature lista. Meg vannak tervezve a kommunikációs csatornák (GUI – Proxy, illetve Proxy – REST service), csakúgy, mint a kommunikáció tartalmi sémája. Ki vannak találva a folyamatok, a sorrendiség és az együttműködő komponensek. Szükséges még a kivitelezés fontosabb részleteinek tárgyalása és a teszt megfontolások.

4.1 Back end

Először a kisegítő funkciókat ismertetem, majd a hívási láncon lefele haladva a back end funkcionalitás lényegi elemeire is kitérek.

4.1.1 Link gyűjtő

A link gyűjtő segédalkalmazás a bemenetén egy URL-t vár és opcionálisan egy számot, aminek alapértelmezése 5. Belül egy ciklus fut annyiszor, ahányat a felhasználó ebben a paraméterben megadott, kivéve, ha nincs annyi találati oldal. A ciklus belsejében egy a tervezésből már ismert WebRequest lekérdezi az aktuális oldalt, majd ebből egy egyszerű reguláris kifejezés (`http://www.hasznaltauto.hu/auto/(.*/.)*`">`) kigyűjti a használt autó linkeket és elmenti egy előre meghatározott fájlba. A ciklus következő futása előtt a program átírja az aktuális találati lista linkjét. Itt tehát tamperinget alkalmazok a találati lista oldalak közötti navigáláshoz. Ennek előnye, hogy a háttérben a linkek elnevezési struktúrája sokkal kevésbé valószínű, hogy megváltozzon, mint mondjuk a node-ok struktúrája, amelyek tartalmazzák a linket a következő oldalra. Hátránya, hogy validálni kell, valóban létezik-e következő oldal. Ezt a válaszban kapott link és a lekért link összehasonlításával teszem meg, tehát legrosszabb esetben egy felesleges lekérdezés történik.

4.1.2 REST végpont

A végpont dolga a beérkező kérések továbbítása a szolgáltatásnak és a megfelelő válaszok konstruálása. Ehhez egy .NET osztályt, a HttpListenert használja. A kérések feldolgozásakor ellenőrzi, hogy a használt metódus POST-e, van-e payload az üzenetben, illetve ha van, ezt maximum 10 sorra korlátozza. Ha valamelyik feltétel nem

4.1.3 Scraper

```
Param
(
    [ValidatePattern('^http:\\\\www.hasznaltauto.hu\\auto\\([a-zA-Z]|\\d|\\|_|-|\\.)+$')]
    [Parameter(Mandatory=$true, ParameterSetName = "Online")]
    [String[]]
    $Uri,
    [Parameter(Mandatory=$true, ParameterSetName = "MultiOnline")]
    [string]
    $Path,
    [Parameter(ParameterSetName = "Offline")]
    [switch]
    $UseSaved
)
```

33

```
PS D:\bes> .\skyscraper_ie.ps1 -UseSaved -Uri http://www.hasznaltauto.hu/auto/test1
D:\bes\skyscraper_ie.ps1 : Parameter set cannot be resolved using the specified named
parameters.
```

ábra 13: Példa a rossz paraméterezésre

UseSaved futtatási esetben az előre definiált (beégetett) kimeneti mappa fájljain hívunk egy Import-Clixml metódus, ami egyből a kulcs – érték párokba rendezett adatokkal tér vissza. Ennek haszna, hogy az algoritmus tesztelését lehet akár offline is végezni, ráadásul jelentősen gyorsabban, akár sok adaton is.

Path és Uri paraméter használatakor is az első lépés annak megállapítása, hogy milyen verziójú PS-ben fut az alkalmazás. A kritikus pontokon később ez alapján disztigvál a rendszer.

```
If($PSVersionTable.PSVersion.Major -ge 3){
    $data = ScrapeWebPages
} Else{
    $data = ScrapeWebPages -compatibilityMode
}
```

A ScrapeWebPages funkció tartalmazza innentől a lényegi részeket a scriptben. Először az alapján, hogy melyik paramétert használjuk, épít egy URI tömböt, majd a tömb minden elemére elvégzi a következőket:

- Whitelisteli a címet, csak akkor dolgozza fel, ha a jó domainre mutat. Erre azért van szükség, mert nem feltétlen az Uri paraméterből jönnek a címek.
- Megnézi a mentett adatokat, és ha megtalálja köztük az aktuálisan kiértékelendő autóét, és ez nem régebben lett mentve egy napnál, akkor eltárolja az adatokat az aktuális indexhez egy másik tömbben (dataTable) és tovább lép a következő URL-re. A keresés egy lépésben történik, mivel az URL-ekből generálódik az adatokat tároló XML fájlok neve.
- Meghívja a spike-okban [42] kiválasztott letöltő és parse-oló műveleteket annak megfelelően, hogy compatibility módban fut-e, vagy sem.
- Hozzáadja a dataTable tömbhöz az adatokból és a címből álló hashmapet
- Elmenti az adatokat XML-ben (cache-eléshez)

Innentől a Comparator szolgáltatás veszi át az adatokat a Scraper közvetlenül ennek kimenetét adja tovább.

4.1.4 Comparator

A Comparator szolgáltatás felépít egy HTML szöveget, ami egy táblázatot tartalmaz az autók adataival. Táblázat sorai az autók paraméterei, az oszlopai az autók. Az utolsó sor egy számított értéket tartalmaz minden autóhoz, ez alapján vannak sorrendbe rakva az oszlopok. Az autók adatainak valódiságát, helyességét a program nem kérdőjelezi meg. Az értékek egy hét tulajdonságot tartalmazó determinisztikus algoritmus futtatásával állnak elő. Az egyes tulajdonságok és súlyaik:

1. Kilowattban mért teljesítmény. Az autónál megadott érték tizennegyedével kerül számításba. Ha nincs megadva, értéke egy.
2. Állapot. Ha bármilyen módon sérül, vagy nincs megadva, értéke mínusz húsz. Minden más esetben értéke nulla.
3. Literben mért csomagtartó méret. Ha meg van adva, a száznegyvened részével kerül számításba, egyébként értéke nulla.
4. Saját tömeg, kilogrammban kifejezve. Ha meg van adva, a mínusz ötszázadával kerül számításba, egyébként értéke mínusz három.
5. Kilométerben mért futásteljesítmény. Az első százezer kilométer mínusz tíz pont, egyenletesen elosztva (ha még nem ment volna annyit). Száz- és kétszázezer kilométer között további öt pont a levonás, szintén egyenletes eloszlással, valamint minden további százezer kilométer további kettő egész öt tized pont mínusz, egyenletesen.
6. Ha meg volt adva teljesítmény és meg van adva az ár, akkor a következő képlet szerint kerül számításba: $\max\left(\frac{\text{teljesítmény} \cdot 500\,000}{\text{ár}}, 10\right)$. Egyéb esetekben az értéke egy.
7. Ha meg van adva az autó gyártási ideje, akkor Edmunds [43]-et alapul véve, a következőképp kerül számításba:

$$f(kor) = \begin{cases} 0, & kor < 3 \text{ hónap} \\ M + 11 * (kor - 1) + 19, & kor \leq 5 \text{ év} \\ M + \log_{10}(kor - 3) * 10 + 60, & kor \leq 30 \text{ év} \\ M + \log_{10}(kor - 3) * 10 + 60 - 0.0006 * kor^3, & kor > 30 \text{ év} \end{cases}$$

Ahol M a gyártás hónapja. Az így kapott szám additív inverzének harmada kerül a végső összegbe.

Ezen hét érték összege plusz hetven egy autó értéke. A konstans eltolásra azért van szükség, hogy ne zavarja össze a felhasználót, hogy az érték sokszor negatív. Az egyes tulajdonságok pozitív vagy negatív előjele az érték meghatározásból származtathatók. Mivel egy használati eszköz valós értékét igyekszem meghatározni, ezért az egyes tulajdonságokról el kell dönteni, hogy valóban értéket képviselnek-e, vagy pont hogy csökkentik az értéket. A számok a személyes heurisztikáim az értékmeghatározáshoz, mivel évek óta figyelem a mainstream autó médiát és használt autó piacot, ezért ez egy hozzávetőleg szakértői heurisztikának tekinthető. Ugyanígy, az érték, vagy nem érték kérdés is szubjektív megítélésű, például, hogy az autó tömege egy látszólag neutrális paraméter, de valójában egy súlytalan autó, vagy egy versenytársaihoz képest fele súlyú nagyon kelendő lenne. A kevésbé fontos karakterisztikák, mint tömeg és csomagtartó méret kisebb, míg a fontosak, mint futásteljesítmény, ár, kor, teljesítmény, jóval nagyobb súllyal kerülnek számításba. Az állapot kilóg a sorból. Ennek legfontosabb információtartalma akkor van, ha véletlenül hiányoznak a hirdetésből, de természetesen jelentős levonást eredményez az értékből, ha sérült, csupán nincs megkülönböztetés a sérülés vagy hiányosságok mértékei között. Ezt a logikát követve: minden hiányzó információ valamilyen formában rontja, vagy legalábbis nem növeli az értéket, típusától függően. Az egyes paraméterek valós értékeire a tesztelésben lehet példákat találni.

4.1.5 Tesztelés

Mivel a scriptben nagyon közvetlen módon jelennek meg a felhasználói, magas szintű követelmények, ezért a back end tesztelésénél is ezeknek lefedésén volt a hangsúly. A back end mint rendszer a Scraper komponensen keresztül tesztelhető. Ennek bemenetein részben whitelistinget, részben validációt alkalmazunk. Ha rosszul paraméterezi fel a felhasználó a scriptet, nem kap eredményt. Mivel ebben az esetben a forráskód rendelkezésre áll, más failure módok tárgyalása értelmetlen, mivel nem lehet rájuk felkészülni. A performance kérdéseket a tervezési szakaszban leteszteltem. Amit lehet és érdemes tesztelni, hogy hogy működik a back end egy adaton, néhány adaton, és sok adaton. Az elvárás, hogy soha, amikor komolyabb változás van a kódban, ne lépjen fel olyan regresszió, amely a bemeneti paraméterek helyes megadása esetén sikertelen futtatáshoz vezet.

1. *Teszt egy autó adatain:*

Az autó eredeti adatainak egy részlete (+ nincs csomagtartó méret és tömeg):

Teljesítmény	320 kW, 435 LE
Kilométeróra állása	17 078 km
Vételár	29.900.000 Ft
Állapot	Sérülésmentes
Évjárat	2014/1

Az ebből kapott eredmények:

Mass	-3.00
AgeLoss	-13.33
Price	10.00
Speedometer	-1.71
Condition	0.00
Power	22.86
Trunk	0.00
Worth	84.82

ASTON MARTIN

Teljesítmény	320 kW, 435 LE
Hengerűrtartalom	4735 cm³
Kilométeróra állása	17 078 km
Kivitel	Coupe
Okmányok jellege	Érvényes magyar okmányokkal
Üzemanyag	Benzin
Állapot	Sérülésmentes
Évjárat	2014/1
Henger-elrendezés	V
Ár (EUR)	€ 96.545
Klíma fajtája	Digitális többzónás klíma
Vételár	29.900.000 Ft
Szállítható szem. száma	2 fő
Calculated value	85

ábra 14: Futtatás eredménye egy autó adataira

2. Teszt két autó adatain:

A második autó adataival a bemenet releváns része:

Teljesítmény	320 kW, 435 LE	92 kW, 125 LE
Kilométeróra állása	17 078 km	2 600 km
Vételár	29.900.000 Ft	6.599.000 Ft
Állapot	Sérülésmentes	Megkímélt
Évjárat	2014/1	2015/3
Tömeg		1 215 kg
Csomagtartó		

A futtatás eredménye:

	ASTON MARTIN	AUDI A3
Teljesítmény	320 kW, 435 LE	92 kW, 125 LE
Hengerűrtartalom	4735 cm ³	1395 cm ³
Kilométeróra állása	17 078 km	2 600 km
Kivitel	Coupe	Sedan
Okmányok jellege	Érvényes magyar okmányokkal	Érvényes magyar okmányokkal
Üzemanyag	Benzin	Benzin
Állapot	Sérülésmentes	Megkímélt
Évjárat	2014/1	2015/3
Henger-elrendezés	V	Soros
Ár (EUR)	€ 96.545	€ 21.316
Klíma fajtája	Digitális többzónás klíma	Digitális klíma
Vételár	29.900.000 Ft	6.599.000 Ft
Szállítható szem. száma	2 fő	5 fő
Sebességváltó fajtája		Manuális (6 fokozatú) sebességváltó
Szín		Fehér
Kárpit színe (1)		Szürke
Műszaki vizsga érvényes		2019/3
Saját tömeg		1 215 kg
Csomagtartó		425 liter
Ajtók száma		4
Nyári gumi méret		205/55 R 16
Össztömeg		1 765 kg
Hajtás		Első kerék
Calculated value	85	78

ábra 15: Futtatás eredménye 2 autóra

Látható, hogy a második autónál jóval több alapadat meg volt adva. Ez természetesen nem okoz hibát, minden kiírásra kerül.

3. Teszt 150 adaton:

Ennél a tesztnél már meglévő xml fájlba mentett adatokat használtam. Az algoritmus ilyen körülmények között 1,6 másodperc alatt futott le. Az adatok lementése csak teszt üzemmódban történik meg, rendes futáskor csak cache-elés céljából készül átmeneti fájl.

	FIAT X1	SUZUKI SJ	DACIA SANDERO	NISSAN JUKE	FORD MUSTANG
Henger-elrendezés	Soros	Soros Álló	Soros	Soros	Soros
Szállítható szem. száma	2 fő	2 fő	5 fő	5 fő	4 fő
Sebességváltó fajtája	Manuális (5 fokozatú) sebességváltó	Manuális (5 fokozatú) sebességváltó felező váltóval	Manuális (5 fokozatú) sebességváltó	Manuális (6 fokozatú) sebességváltó	Automata (6 fokozatú tiptronic) sebességváltó
Teljesítmény	175 kW, 238 LE	73 kW, 99 LE	55 kW, 75 LE	140 kW, 190 LE	231 kW, 314 LE
Hengerűrtartalom	1998 cm ³	1600 cm ³	1390 cm ³	1618 cm ³	2300 cm ³
Kilométeróra állása	200 km	1 km		5 km	5 500 km
Kivitel	Sport	Terepjáró	Ferdehátú	Városi terepjáró (crossover)	Coupe
Saját tömeg	620 kg	650 kg	895 kg	1 300 kg	
		●			
		●			
		●			
Calculated value	151	130	96	91	88

ábra 16: Futtatás eredmény egy része 150 autóra

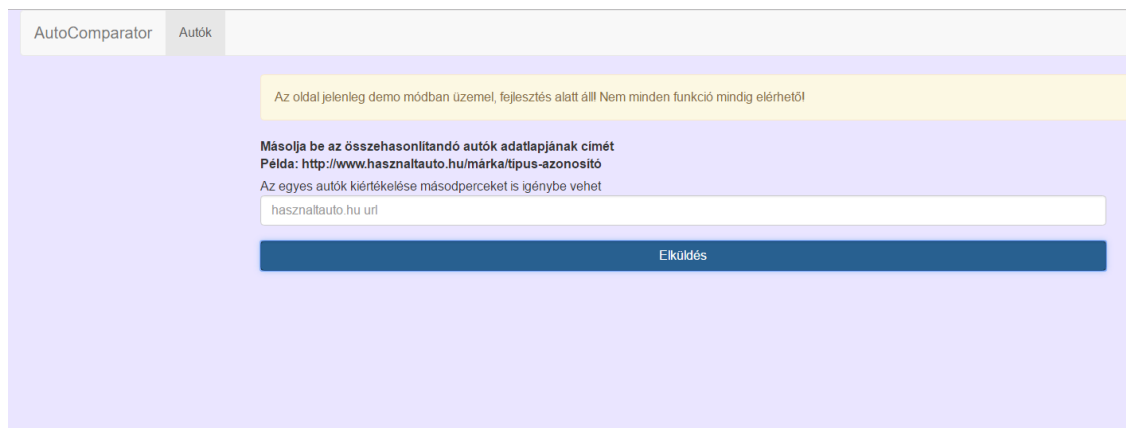
4.2 Front end

A front end kialakításához nem használtam fel külső modularizálást támogató frameworkot. A legnagyobb része a front endnek „natív” JS, néhol jQuery-vel [44] kiegészítve. Az legfontosabb szempontok, ahogy a tervezésben is szerepelt, az egyszerűség és a használhatóság.

A front end kiszolgálója egy tetszőleges webszerver, amely támogatja a PHP futtatást. Tetszőleges alatt értem az apache mockot, amelyet egy WAMP(Windows Apache MySQL PHP) stack formájában használok teszteléshez, épp úgy, mint egy PaaS szolgáltató (kívülről nem feltétlen látható) szerver szolgáltatását, amely PHP futtatást biztosít, mondjuk egy Git szinkronizáción keresztül. A PHP index oldal semmi mást nem csinál, mint meghivatkozza a valódi index HTML oldalt, ezt kapja meg a browser.

A megoldás könnyedén hordozható szolgáltatók között. Példának okán, ahhoz, hogy egy Heroku nevű PaaS providertől átmigráljam AWS (Amazon Web Services) Elastic Beanstalkra, a kódon nem kellett módosítani, csak konfigurációs fájlokon, melyek az adott környezet jellemezték.

4.2.1 Graphical User Interface



ábra 17: GUI kezdőképernyő

A UI modularizálását natív módon oldottam meg, valamilyen strukturálásra még egy ilyen egyszerű felület esetén is szükség volt. Rendelkezésre áll az alapvető működést támogató index.js, ez kezeli a lap betöltődését és a kommunikációt a PHP proxy-val.

A navigációs bar felépítéséért a NavbarBuilder.js felel. Single-page applikációnál nincs kifejezett szükség ennek a különválasztására, viszont praktikus, ha több oldal is lenne egyszer.

Végül egy InputElementBuilder.js fájl végzi a szövegmezők létrehozását és törlését. Mikor a felhasználó ráklikkel az aktuális utolsó szövegmezőre, létrejön egy új üres mező ez alatt, ha még nem volt összesen tíz beviteli mező felvéve. Az első kivételével minden mező mellé felvesz egy gombot is, amivel a mezőt ki lehet venni a listából.

Az aktuális sessionben a felhasználó bevitt értékei nem vesznek el. Ezt a JS sessionStorage objektuma biztosítja, mely az oldal elhagyásakor elmenti a szövegmezők tartalmát. Mikor a felhasználó a lapra navigál ismét, többek közt a következő kódrészlet fut le:

```
if( !sessionStorage.inputUriCount || sessionStorage.inputUriCount > 10 ||
sessionStorage.inputUriCount < 1 ) {
    sessionStorage.clear();
    var textDiv = createTextDiv();
    formElement.insertBefore(textDiv, sendButtonDiv);
} else {
    var savedUris = JSON.parse(sessionStorage.inputUriTextContent);
    for(var i = 0; i < Number(sessionStorage.inputUriCount); i++) {
        var textDiv = createTextDiv(savedUris[i]);
        formElement.insertBefore(textDiv, sendButtonDiv);
    }
}
```

Először ellenőrzi a storage tartalmát, majd ha ez megfelel az elvártnak, az elemeket átadja egyesével az InputElementBuilder createTextDiv metódusának, amely létrehozza a beviteli mezőt. Ezután a mező és a gombja elhelyezésre kerülnek a küldés gomb felett.

A futtatás eredménye a back endnél már tárgyalt HTML alapú táblázat, így ez közvetlenül betöltődik az oldal HTML kódjába, amikor megérkezik a válasz.

```
if (xhttp.readyState == 4 && xhttp.status == 200) {
    document.getElementById("resultTable").innerHTML =
    xhttp.responseText;
}
```

Ezzel szemben, ha sikertelen a művelet egy piros szövegdobozban egy hibaüzenet tájékoztatja a felhasználót a hiba jellegéről és a potenciális megoldási lehetőségekről, ahogy ez a 6. ábrán is látható volt.

4.2.2 Proxy

A proxy.php teszi lehetővé, hogy a front end tetszőleges back end service-szel tudjon kommunikálni. Enélkül a back end válaszaiban mindig jelen kéne legyen egy megfelelő whitelist, amely tartalmazza a front end aktuális domainjét. Ez a tervezésben tárgyaltaikon kívül hordozhatóság szempontjából is praktikus. A proxy a kapott adatokkal felépít egy curl hívást (ha kapott adatokat, ezt az egyet ellenőrzi), amelyet egy előre definiált cím felé indít.

4.2.3 Teszt

A felhasználói felület tesztelésénél a legfontosabb szempont, hogy helyes bemenetre működik-e a rendszer. Csak egy helyes scenario van, minden más esetben hibát kell jelezzen a felület. Minden scenariot nem lehet tesztelni az ISTQB exhaustive testingről szóló része szerint, ebből kifolyólag csak a fontosabb negatív eseteket fedem le.

1. Pozitív eset

Egyedi, valid url-eket megadva, az eredmény az „Elküldés” gomb alatt megjelenő táblázat az autók adataival, legalul, ahogy már feljebb is látható volt a számított értékkel. (Egy az egyben megkapja a back enden generált táblázatot.)

Az oldal jelenleg demo módban üzemel, fejlesztés alatt áll! Nem minden funkció mindig elérhető!

Másolja be az összehasonlítandó autók adatlapjának címét
Példa: <http://www.hasznaltauto.hu/márka/típus-azonosító>
Az egyes autók kiértékelése másodpercekig is igénybe vehet

	TOYOTA YARIS	SKODA OCTAVIA
Okmányok jellege	Érvényes magyar okmányokkal	Érvényes magyar okmányokkal
Műszaki vizsga érvényes	2018/8	2017/6
Klíma fajtája	Digitális klíma	Digitális kétzónás klíma
Hengerűrtartalom	1798 cm³	1896 cm³
Akciós ár	2.149.000 Ft	

ábra 18: GUI teszt pozitív eset

2. Azonos URL-ek

Ha azonos URL-eket ad be a felhasználó, az autó adatai csak egyszer jelennek meg. Az üres mezők nem kerülnek elküldésre.

TOYOTA YARIS	
Okmányok jellege	Érvényes magyar okmányokkal
Műszaki vizsga érvényes	2018/8
Klíma fajtája	Digitális klíma
Hengerűrtartalom	1798 cm³
Akciós ár	2.149.000 Ft
Henger-elrendezés	Soros

ábra 19: GUI teszt azonos bemenetek

3. *Hibás URL megadása*

Ha a felhasználó nem jó URL-t ad meg (nem hasznaltauto.hu-ra mutat, vagy nem autóra mutat ...), hibaüzenetet kap vissza.

Sikertelen művelet. Ellenőrizze a megadott címeket, vagy próbálja meg későbbi Státusz 404 Not Found

ábra 20: GUI teszt hibás bemenet

4. *Üres bemenet*

Ha a felhasználó nem ad meg egy URL-t se, szintén hibaüzenetet kap.

Sikertelen művelet. Ellenőrizze a megadott címeket, vagy próbálja meg későbbi Státusz 418 No content, I'm a teapot

ábra 21: GUI teszt üres bemenetek

V. Összefoglalás

A munka elején az Irodalomkutatásban lehetőségem volt megismerni a megvalósításhoz rendelkezésemre álló eszközöket, elméleti háttérrel nyerni a feladat elvégzéséhez. Ennek végére kialakult bennem egy átfogó terv, egy vision, hogy mik pontosan az elérendő célok és körvonalazódtak az utak is, amelyeken el lehet érni ezeket.

Ezek után a tervek első lépéseként meghatároztam minden elvárást a rendszerrel szemben, ami egy felhasználó oldaláról felmerülhet az adott business scope-on belül (MVP). Tettem ezt már egyben két komponens csoport definiálásával, hogy minimálisan ugyan de konkrét tárgyai legyenek a követelményeknek.

Miután már világosak voltak a legmagasabb szintű követelmények, meghatároztam, hogy ezek milyen belső folyamatok mellett kell teljesüljenek. A folyamatok meghatározása azon funkcionális követelmények körét jelenti, melyek nem kapcsolhatók közvetlenül a felhasználóhoz.

A követelmények rendelkezésre állásával lehetőség nyílt az architektúra megtervezésére és a konkrétan használt technológiák körének meghatározására. Itt nyert struktúrát az addig csak implicit összerendelt komponensek halmaza. Ez alapján a terv alapján lehetett egy elvárásoknak megfelelő implementációt készíteni.

A tervekből és megvalósításból látható, hogy az MVP teljesíti a vele szemben támasztott elvárásokat. A felhasználók képesek általa egy találati listát tovább szűkíteni, így megkönnyíti a döntési folyamatot. A folyamat teljesen automatizált, egyszerűen használható. A megvalósítás hordozható és kihasználja a technológia adta lehetőségeket a maximális lehetséges teljesítmény eléréséhez. Az algoritmus képes felderíteni nagyjából azonos hirdetések különbségeit.

Mindezek mellett rengeteg fejlesztési potenciálja is van a programnak. A keresést a találati listában ki lehetne ajánlani a GUI-ra. Lehetne a keresések eredményét szűrni, sorrendbe állítani más szempontok szerint, mint ahogy alapértelmezésben van. Az egyes tulajdonságok szerint a legjobb értéket ki lehetne emelni a találati táblázatban. Ezekhez a változtatásokhoz érdemes lenne megfontolni egy modularizálást használó front end keretrendszer használatát, mivel ekkor már a front end is sokkal nagyobb komponens lenne, így a fenntarthatóság érdekében megfelelően kellene strukturálni.

Lehetne kialakítani felhasználói profilokat, amelyekhez hozzá lehetne rendelni keresési előzményeket és preferenciákat, kedvenceket, és a statikus súlyokkal dolgozó algoritmus kaphatná a preferenciákat a felhasználótól. A mostani megvalósítás helyett a jobb értékmeghatározáshoz lehetne használni külső értékeléseket, új autó árfigyelést és ebből adott típusra értékcsökkenést, lehetne fenntartási költségeket számolni, megbízhatóságot figyelni. Ki lehetne bővíteni az algoritmushoz használt tulajdonságok listáját az extrákkal (tolatóradar, bőrülés, klíma...), lehetne felhasználónként profilt analízálni. Összefoglalóan, lehetne az algoritmusban valamilyen mesterséges intelligenciát alkalmazni, tanulóvá tenni a folyamatot, miközben kiegészíteni a rendelkezésre álló adatok listáját. Ezen többnyire back end átalakításokhoz már nyilvánvalóan kellene egy adatbázis is, esetleg ha a számításokhoz kell egy in-memory adatbázis, akkor kettő. Teljesítmény szempontjából, ha a PS technológiánál maradva kellene fejleszteni, akkor az aszinkronitás kérdéskörét és az ezzel kapcsolatos biztonsági feladatokat kéne megoldani.

Érdemes megjegyezni, hogy a PowerShell új verziói is további lehetőségeket nyújtanak majd az alkalmazás továbbfejlesztéséhez, hiszen a fejlesztői az elmúlt néhány évben is publikáltak olyan új könyvtárakat, amelyek segítik a scrapelést.

Szakdolgozatomban tehát bemutattam egy kész Proof of Conceptet, amely megmutatja a PowerShell képességét scraping feladatok ellátására, ezen belül pedig egy konkrét felhasználási mód példázza ennek hasznosságát.

VI. Irodalomjegyzék

- [1] Frances Zlotnick: *Web scraping with Beautiful Soup*, http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html (revision 20:19, 8 December 2016)
- [2] W3C: *HTML5*, <https://www.w3.org/TR/html5/> (revision 19:29, 8 December 2016)
- [3] Hasznaltauto.hu: *Trendforduló előtt a használtautó-piac*, http://hasznaltauto-index.hu/cikk/trendfordulo_elott_a_hasznaltauto-piac-81 (revision 15:28, 2 December 2016)
- [4] TechNet: *Scripting with Windows PowerShell*, <https://technet.microsoft.com/en-us/library/bb978526.aspx> (revision 15:03, 4 July 2016)
- [5] TechNet: *Invoke-WebRequest*, <https://technet.microsoft.com/en-us/library/hh849901.aspx> (revision 09:57, 10 September 2016)
- [6] TechNet: *Invoke-RestMethod*, <https://technet.microsoft.com/en-us/library/hh849971.aspx> (revision 09:58, 10 September 2016)
- [7] MSDN: *InternetExplorer object*, [https://msdn.microsoft.com/en-us/library/aa752084\(v=vs.85\).aspx#properties](https://msdn.microsoft.com/en-us/library/aa752084(v=vs.85).aspx#properties) (revision, 14:25, 12 September 2016)
- [8] 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)
- [9] Brian Cooksey: *An introduction to APIs*, <https://zapier.com/learn/apis/> (revision 19:14, 7 December 2016)
- [10] Roy Thomas Fielding: *Representational state transfer (REST)*, http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (revision 21:11, 7 December 2016)
- [11] 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)
- [12] MSDN: *Navigate method*, [https://msdn.microsoft.com/en-us/library/aa752093\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa752093(v=vs.85).aspx) (revision 20:26, 14 September 2016)
- [13] TechNet: *Using the Set-ExecutionPolicyCmdlet*, <https://technet.microsoft.com/en-us/library/ee176961.aspx> (revision 18:14, 17 September 2016)
- [14] Bootstrap: *Bootstrap index page*, <http://getbootstrap.com/> (revision 19:10, 17 September 2016)

- [15] K.V.K.K. Prasad: *Istqb Certification Study Guide: Iseb, Istqb/ Itb, Qai Certification, 2008 Ed.* DreamTech Press, 2006
- [16] Object Management Group: *Unified Modeling Language: Superstructure*, <http://doc.omg.org/formal/2005-07-04.pdf> (revision 21:36, 6 December 2016)
- [17] W3C: *Extensible Markup Language (XML)*, <https://www.w3.org/XML/> (revision 21:32, 7 December 2016)
- [18] WHATWG: *URL*, <https://url.spec.whatwg.org/> (revision 21:17, 7 December 2016)
- [19] The Linux Information Project: *GUI Definition*, <http://www.linfo.org/gui.html> (revision 19:25, 7 December 2016)
- [20] Marc Hassenzahl: *User experience and Experience Design*, <https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/user-experience-and-experience-design> (revision 21:26, 7 December 2016)
- [21] ASQ.org: *Failure Mode Effect Analysis(FMEA)*, <http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html> (revision 20:56, 7 December 2016)
- [22] RahimHirji: *DEFINING YOUR MINIMUM VIABLE PRODUCT*, <http://www.edukwest.com/minimum-viable-product/> (revision 20:58, 7 December 2016)
- [23] 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 2016)
- [24] Thomas Decker: *Dynamic Load Balancing and Scheduling*, <http://www2.cs.uni-paderborn.de/cs/ag-monien/RESEARCH/LOADBAL/> (revision 21:10, 25 September 2016)
- [25] Network Working Group: *Internet Denial-of-Service Considerations*, <https://tools.ietf.org/html/rfc4732> (revision 21:06, 25 September 2016)
- [26] Google: *About multitenancy*, <https://cloud.google.com/appengine/docs/java/multitenancy/?csw=1> (revision 19:05, 29 October 2016)
- [27] W3C: *Same-Origin Policy*, https://www.w3.org/Security/wiki/Same-Origin_Policy (revision 19:40, 29 October 2016)
- [28] Aaron Swartz: *A Brief History of Ajax*, <http://www.aaronsw.com/weblog/ajaxhistory> (revision 19:11, 7 December 2016)
- [29] W3C: *Cross-Origin Resource Sharing*, <https://www.w3.org/TR/cors/> (revision 19:17, 7 December 2016)

- [30] Peep Laja: *First Impressions Matter: The Importance of Great Visual Design*, <http://conversionxl.com/first-impressions-matter-the-importance-of-great-visual-design/> (revision 19:22, 30 October 2016)
- [31] David Flanagan: *JavaScript: The Definitive Guide*. O'Reilly Media Inc., 2011
- [32] W3C: *Cascading Style Sheets (CSS) Snapshot 2010*, <https://www.w3.org/TR/css-2010/> (revision 19:20, 7 December 2016)
- [33] jQuery UI: *About jQuery UI*, <http://jqueryui.com/about/> (revision 21:40, 7 December)
- [34] Foundation: *Foundation is a responsive front-end framework.*, <http://foundation.zurb.com/showcase/about.html> (revision 21:44, 7 December 2016)
- [35] PHP: *What is PHP?*, <http://php.net/manual/en/intro-whatis.php> (revision 21:07, 7 December)
- [36] W3Techs: *Usage of server-side programming languages for websites*, https://w3techs.com/technologies/overview/programming_language/all (revision 10:08, 7 November 2016)
- [37] Amazon: *Types of Cloud Computing*, <https://aws.amazon.com/types-of-cloud-computing/> (revision 21:01, 7 December)
- [38] ISTQB: *International Software Testing and Qualifications Board*, <http://www.istqb.org/> (revision 20:42, 7 December 2016)
- [39] W3C: *Uniform Resource Identifier*, <https://www.w3.org/Addressing/URL/uri-spec.html> (revision 21:14, 7 December 2016)
- [40] Laurie Williams: *Input Validation Vulnerabilities*, http://agile.csc.ncsu.edu/SEMaterials/3_InputValidation.pdf (revision 20:45, 7 December 2016)
- [41] The Scripting Guys: *Backwards Compatibility in PowerShell*, <https://blogs.technet.microsoft.com/heyscriptingguy/2015/09/14/backwards-compatibility-in-powershell/> (revision 18:04, 7 November 2016)
- [42] Bill Ambrosini: *Spikes and the Effort-to-Grief Ratio*, <https://www.scrumalliance.org/community/articles/2013/march/spikes-and-the-effort-to-grief-ratio> (revision 18:23, 21 November 2016)
- [43] Edmunds: *Depreciation Infographic: How Fast Does My New Car Lose Value?*, <http://www.edmunds.com/car-buying/how-fast-does-my-new-car-lose-value-infographic.html> (revision 19:47, 21 November 2016)
- [44] jQuery: *What is jQuery?*, <https://jquery.com/> (revision 21:36, 7 December 2016)

VII. Rövidítések jegyzéke

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
ISTQB	International Software Testing and Qualification Board
JS	JavaScript
MVP	Minimum Viable Product
PaaS	Platform as a Service
PHP	PHP: Hypertext Processor
PS	PowerShell
REST	Representational State Transfer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UX	User eXperience
XML	Extensible Markup Language