VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

METÓDY EXTRAKCIE DÁT Z WEBOVÝCH STRÁNOK

METHODS OF DATA EXTRACTION FROM THE WEB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ PERINA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Perina Lukáš**
Program: Informační technologie
Název: **Metody extrakce dat z webových stránek**
Methods of Data Extraction from the Web
Kategorie: Web

Zadání:

1. Seznamte se se současnými serverovými i klientskými technologiemi pro implementaci webových aplikací v JavaScriptu.
2. Prostudujte současné přístupy pro extrakci dat (scraping) z webových dokumentů.
3. Navrhněte architekturu aplikace pro extrakci dat z webových stránek na základě předem definovaných pravidel. Svá rozhodnutí konzultujte s vedoucím.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte vyhodnocení funkčnosti vytvořené aplikace na vhodné sadě webových dokumentů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Alarte, J.; Insa, D.; Silva, J.; et al.: Main Content Extraction from Heterogeneous Webpages. In Web Information Systems Engineering - WISE 2018. Cham: Springer International Publishing. 2018. ISBN 978-3-030-02922-7. pp. 393-407.
- Burget, R.: Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: 15th International Conference on Web Information Systems and Technologies. Vienna: SciTePress - Science and Technology Publications, 2019, s. 326-333. ISBN 978-989-758-386-5.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Cieľom tejto bakalárskej práce je návrh architektúry a následná implementácia aplikácie, ktorá je určená na extrakciu dát (web scraping) z webových dokumentov. Na rozdiel od konvenčných metód sa jedná o extrakciu založenú na definovaní dátových typov a regulárnych výrazov hľadaných prvkov. Extrakcia prebieha tak, aby nebolo potrebné poznať detailnú štruktúru daného webového dokumentu, a aby bolo umožnené použitie jednej definície na detekciu hľadaných prvkov na rôznych webových stránkach. Algoritmus dosahuje priemernú presnosť 88,26% a recall 82,85%. Týmto prístupom sa umožní zredukovať čas, ktorý je potrebný na analýzu jednotlivých stránok na minimum a nebrať štruktúru kódu, ako určujúci faktor pri vytváraní požiadavkov na webscraping.

Abstract

The purpose of this bachelor thesis is to design an architecture and subsequent implementation of an application designed for data extraction (web scraping) from web documents. Unlike conventional methods, it is an extraction based on defining data types and regular expressions of requested elements. Extraction is executed in such a manner, where it is not necessary to know the detailed structure of given web document, and the possibility of using just one definition to detect requested elements on different web pages. Algorithm is able to achieve overall accuraccy of 88,26% and recall 82,85%. This approach can reduce the time required for analysis significantly, and not to take the structure of the code as a determining factor in creating webscraping requests.

Kľúčové slová

Web scraping, Javascript, Node.js, Google Chrome, Json, Extrakcia dát, scraping, web, DOM, CSS, HTML, Puppeteer

Keywords

Web scraping, Javascript, Node.js, Google Chrome, Json, data extraction, scraping, web, DOM, CSS, HTML, Puppeteer

Citácia

PERINA, Lukáš. *Metódy extrakcie dát z webových stránok*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Metódy extrakcie dát z webových stránok

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Burgeta Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Lukáš Perina
13. apríla 2021

Poďakovanie

Moje poďakovanie patrí vedúcemu práce pánovi Ing. Radkovi Burgetovi Ph.D. za dôsledné vedenie bakalárskej práce a konzultácie ktoré mi pomohli pri práci a určili ten správny smer.

Obsah

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je návrh architektúry, a následná implementácia aplikácie určenej na extrakciu dát z webových stránok. V dnešnej dobe je pojem extrakcia dát používaný stále častejšie, a to práve v spojitosti s webovými technológiami a webom samotným. Je to tak pravdepodobne preto, že sa internet ako taký, a hlavne jeho obsah v podobe webových stránok, neustále rozširuje. Objem dát, ktorý sa na webe nachádza, sa zväčšuje neúprosnou rýchlosťou, a preto je zber údajov z rôznych webstránok stále zložitejší. A to nielen čo sa zložitosti štruktúry webu týka, ale vzhľadom na veľký počet webov sa zvyšuje aj časová náročnosť analýzy a následnej extrakcie dát. Takejto extrakcii sa hovorí web scraping. Web scraping môže prebiehať aj manuálne, kde ho vykonáva určená osoba, avšak v dnešnej dobe je už táto procedúra automatizovaná.

Práve pre spomínanú expanziu internetu sa tejto téme venuje čoraz viac spoločností, so snahou uľahčiť prístup k web scrapingu pre každodenných užívateľov. Pre neustály vývoj webu je avšak nutné vyvíjať aj aplikácie, ktoré sú na extrakciu určené.

Pri rozlišovaní takýchto aplikácií sa berie na vedomie hlavne požadovaný vstupný formát a jeho zložitosť, požadovaný výstupný formát a jeho využiteľnosť v praxi, a v neposlednom rade rýchlosť aplikácie z pohľadu extrakcie dát.

Táto práca je zameraná hlavne na časť týkajúcu sa požadovaného vstupu aplikácie. Hlavným cieľom aplikácie je umožniť vyjadrenie požadovaných vstupných parametrov tak, aby neboli závislé na štruktúre webovej stránky, ale na jej obsahu. Na dosiahnutie tohto cieľa je potrebné vstupné údaje správne definovať, pretože práve na týchto vstupných údajoch bude závislá úspešnosť následnej extrakcie dát. Tento vstup pozostáva z adries webových stránok, dátových typov, rozloženia objektov na webstránke a požadovaných názvov extrahovaných informácií. Aplikácia sa následne pozerá na každú stránku zvlášť a určí výsledné údaje na základe poskytnutých relevantných informácií o štruktúre. Týmto spôsobom je možné definovať jeden vstup raz, a použiť ho na viaceré webové dokumenty bez nutnosti ďalšieho zásahu. Zároveň by malo byť umožnené extrahovať akékoľvek údaje z akejkoľvek stránky, bez nutnosti analýzy danej webovej stránky alebo poznania jej vnútornej štruktúry. Vstupné a výstupné údaje budú taktiež užívateľský prívetivé tak, aby nebol kladený dôraz na skúsenosti s programovaním a programovacími jazykmi.

Práca je rozdelená do 7 kapitol, z ktorých každá predstavuje časť vývoja finálnej aplikácie, od predstavenia aktuálnej problematiky a technológií, ktoré sú v súčastnosti dostupné až po finálne testovanie aplikácie a zhodnotenie výsledkov.

Prvá časť práce sa v kapitole 2 zaoberá aktuálnym prístupom k problematike, ako sa web scraping využíva v súčasnosti, aké sú štandardy a dostupné technológie a porovnanie týchto prístupov z hľadiska využiteľnosti.

V druhej časti tejto práce sú v kapitole 3 popísané technológie, ktoré je treba brať v úvahu pri návrhu, tvorbe a analýze aplikácie určenej na extrakciu dát. Jedná sa hlavne o technológie, ktoré sa stali základným kameňom navrhnutej aplikácie, a o technológie na ktorých aplikácia stavia.

Kapitola 4 sa zameriava na analýzu a návrh architektúry aplikácie. Je založená na analýze špecifikácie, požiadavkov aplikácie a popisuje postup návrhu aplikácie.

V 5. kapitole sú popísané jednotlivé detaily implementácie riešenia a hlavná logika aplikácie.

6. kapitola sa následne venuje testovaniu aplikácie za pomoci niekoľkých datasetov, následnej analýze a vyhodnotení výsledkov ktoré aplikácia dosiahla.

Kapitola 2

Aktuálny prístup k problematike Web Scrapingu

V tejto kapitole sú uvedené a popísané aktuálne prístupy ktoré sa používajú na extrakciu dát z webových dokumentov. Takáto extrakcia môže prebiehať online alebo offline, v závislosti na tom aké dáta a za akým cieľom chceme extrahovať.

Web scrapingu ako takému sa v súčasnosti venuje stále viac a viac spoločností, a preto nieje prekvapením že sa rozširujú nielen možnosti web scrapingových aplikácií, ale aj prístup takýchto aplikácií k priamej extrakcii dát.

Zároveň sa rozširuje pole pôsobnosti, a možnosti využitia takto extrahovaných dát. Preto je táto kapitola určená hlavne rozboru týchto metód. Poznatky získané z tohto rozboru zároveň pomôžu určiť smer ktorým sa návrh a vývoj aplikácie môžu uberať.

2.1 Web scraping

Web scraping je jednou z foriem získavania údajov z webových stránok. Takéto získavanie údajov je možné buď priamo - *World Wide Web* za pomoci *Hypertext Transfer Protokolu*, alebo pomocou webového prehliadača (napr. Google Chrome). Za web scraping sa dá považovať aj manuálny zber dát za využitia ľudskej sily, ale všeobecne sa tým myslí využitie dedikovaného počítača ktorý túto prácu automatizuje. Tu sa môže jednať napríklad o jednoduché kopírovanie dát, alebo zložitejšie generovanie výstupných štruktúr napríklad vo formáte JSON.

Automatizované získavanie informácií z webu sa dostalo do vývoja krátko po zavedení World Wide Web. Kvôli neustálemu vývoju technológií je však vo vývoji doteraz, a jeho možnosti sa neustále rozširujú. Prvé formy automatizovaných web scraperov boli určené primárne pre tvorbu databázy vyhľadávacieho indexu pre vývoj World Wide Web Vyhľadávače. Za jeden z prvých takýchto scraperov je preto považovaný World Wide Web Wanderer, ktorý bol vyvinutý v roku 1993 za účelom zmerania veľkosti celého internetu, a neskôr na jeho priamu indexáciu. [11]

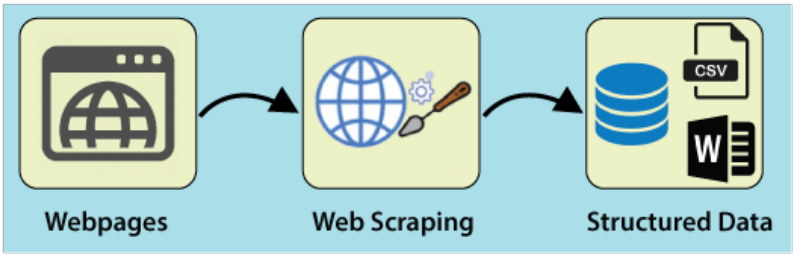
2.2 Súčasné postupy

Existujú rôzne prístupy k danej problematike, kde rozdiely sú jednoznačné hlavne pri spôsobe a type analýzy a extrakcie finálnych dát. Hlavným cieľom pri tvorbe aplikácie na takúto extrakciu dát je užívateľská prívetivosť, a jednoduchosť definovania vstupných požiadavkov.

Postup ktorý aplikácie na extrakciu dát využívajú je prevažne rovnaký, a skladá sa z troch primárnych krokov:

1. Odoslanie GET požiadavku na webový server a následné obdržanie odpovedi
2. Analýza extrahovaného HTML kódu na základe stromovej štruktúry
3. Použitie zvoleného postupu na extrakciu dát a spracovanie hľadaného obsahu

Tieto kroky sú potom pri každej aplikácií mierne prispôsobené určeniu a typu implementácie. [11]



Obr. 2.1: Všeobecný postup ktorým sa aplikácia typu web scraper riadi. Prevzaté z Javatpoint [6]

Medzi najpopulárnejšie web scrapery patria aplikácie s jednoduchou point-and-click politikou, kde používateľ po inštalácií takejto aplikácie jednoducho zvolí webové dokumenty z ktorých chce extrakciu prevádzať, a následne interaktívne vyznačí dáta ktoré ho zaujímajú. Pre užívateľov ktorý nemajú príliš veľké skúsenosti s informačnými technológiami a nechcú platiť za túto službu nemalé peniaze rôznym korporáciám je to skrátka jediná možnosť.

Spomínaný prístup má samozrejme svoje výhody, medzi ktoré sa radí najmä spomínaná užívateľská prívetivosť a relatívne nízka cena, avšak v prípade potreby automatizácie takejto činnosti existujú na trhu lepšie riešenia. V prípade že je potrebná vysoká automatizácia, alebo sa jedná o veľké množstvo dát a stránok ktoré sú na extrakciu určené, je výhodnejšie použiť aplikáciu ktorá je ladená skôr na tento spôsob. Pri firmách ktoré sa venujú extrahovaniu údajov z webových stránok na profesionálnej úrovni sa počet prenesených dát v súčasnosti pohybuje už v petabajtoch, a počet extrahovaných webových stránok sa pohybuje v miliardách mesačne na jednu takúto firmu.

Pri takto veľkých číslach narážajú spomínané firmy na rôzne problémy, ktoré rádový používateľ web scrapingovej aplikácie riešiť nemusí. Medzi takéto problémy sa radia v neposlednom rade napríklad:

- Náročnosť na hardvérový čas
- Limitovanie počtu možných GET dotazov
- Veľká diverzifikácia architektúry webových stránok
- Objem dát potrebný na prenesenie údajov a následné uloženie extrahovaných údajov

Príklad ako sa takýmto problémom brániť alebo im priam predchádzať je v prípade limitovania počtu možných GET dotazov na jednu webovú stránku meniť IP adresy z ktorých sú GET dotazy odosielané. Ani to však nie vždy môže byť riešením, keďže stránky v dnešnej dobe môžu využívať rôzne ochrany proti web scrapingu, ako napríklad priamu detekciu človeka od robota alebo tzv. CAPTCHA[1] .[17]

Web scraping má teda svoje výhody, ale aj nevýhody. Tieto sú popísané podrobnejšie v nasledujúcej časti tejto kapitoly.

2.3 Výhody, nevýhody a využitie web scrapingu

Web scraping má v súčasnosti viac možností a oblastí využitia ako tomu bolo v minulosti. Toto je dané hlavne neustálim rozširovaním sa internetu ako celku. V počiatkoch internetu sa aplikácie typu web scraper nazývali skôr crawler, a využívali prevažne na indexáciu obsahu, a nie na zber, extrakciu a analýzu dát tak ako je tomu teraz.

Web scraper je ale aplikácia ktorá je založená presne na tých fundamentálnych základoch ako spomínaný crawler, s jediným rozdielom a to tým, že jeho úlohou je zber dát. Takýto zber môže prebiehať napríklad kopírovaním dát do databázy. V mnohých prípadoch sú aplikácie typu scraper využívané veľkými korporáciami ako napríklad Microsoft, Amazon alebo Google na cielenie zobrazovaných reklám tak, aby boli pre užívateľa ako jednotlivca relevantné.[11]

Jednotlivé prípady využitia takejto aplikácie sa neustále rozširujú, obecne ale platí že jeho využitie spadá do nasledovných kategórií:

- Monitorovanie ceny produktov
- Získavanie údajov o nových produktoch na trhu
- Analýza konkurencie
- Evidovanie údajov o konkrétnej doméne podľa druhu záujmu
- Monitorovanie ceny leteniek

A mnohé ďalšie prípady. Údaje extrahované touto metódou môžu byť následne využité na širšie pochopenie vývoja udalostí napríklad na trhu, alebo u konkurencie.

Web scraping má teda jasné výhody čo sa kolekcie dát týka. Patria medzi ne už spomínané monitorovanie konkurencie, cien produktov a analýza rôznych dát. V neposlednom rade ale treba zmieniť aj výhody z pohľadu technológie samotnej. Medzi ne patria napríklad:

- Rýchlo sa rozvíjajúce technológie
- Automatizovanie monotónnych činností
- Zber obsiahleho množstva dát, ktorý by bol manuálne nemožný
- V porovnaní s manuálnym zberom dát exponenciálne rýchlejšie výsledky

[1] Completely Automated Public Turing test to tell Computers and Humans Apart.

Každá technológia a jej využitie má však aj svoje nevýhody. V porovnaní s manuálnym zberom dát, ktoré môže byť rozumné ak sa jedná o ojedinelú prípadne unikátnu udalosť, kedy je takýto zber potrebné vykonať jeden krát za dlhé obdobie, sa tvorba alebo priame využitie web scrapera očakáva až v prípade že sa daná akcia musí vykonávať automatizovane a v určitých časových intervaloch. [6]

V takom prípade je potreba zvážiť aj rôzne prvky pri nielen prevádzke, ale aj samotnom vytváraní a následnom používaní web scrapera. Medzi najhlavnejšie nevýhody sa preto radí najmä:

- Nutná znalosť kódu a programovania
- IP Detekcia a CAPTCHA
- Ku každej stránke je potreba pristupovať individuálne

Zároveň treba spomenúť fakt, že dynamické stránky môžu meniť svoju štruktúru, a preto nemôžme zabúdať na údržbu web scrapera nielen z pohľadu úrovne bezpečnosti programu, ale aj z pohľadu údajov ktoré web scraper prijíma. V prípade že webová stránka zmení štruktúru tak, že aktuálna konfigurácia web scrapera nedokáže pokračovať v extrahovaní údajov tak ako predtým, je potrebné v mnohých prípadoch stránku znova analyzovať, a v prípade že je web scraper neschopný efektívne využiť algoritmy ktoré v minulosti fungovali, v niektorých prípadoch aj dodatočne zanalyzovať prístupy web scrapera a upraviť jeho logiku tak, aby mohol v extrahovaní pokračovať.

2.4 Metódy využívané v súčasnosti

Aktuálne používané metódy sa vo všeobecnosti dajú rozdeliť do dvoch kategórií:

- Pripravené aplikácie
- Platformy a knižnice

Pripravené aplikácie budú obľúbenejšie najmä pri širokej verejnosti, ktorá má čoraz častejšie o web scraping záujem. Tieto aplikácie predstavujú rýchlu a často jednoduchú cestu k splneniu toho čo často požadujeme najviac - výsledky rýchlo a kvalitne.

V mnohých prípadoch však chceme priniesť aj svoje nápady na vylepšenia. Nastavenie takýchto aplikácií býva často limitované, a ich kód nemusí vždy byť verejne dostupný. V takom prípade je lepšie využiť jednu z dostupných knižníc, na ktorých väčšina týchto aplikácií stavia.

2.4.1 Hotové riešenia

V prípade že nás zaujíma len priama extrakcia dát, v dnešnej dobe je na výber z mnohých aplikácií ktoré sú pripravené na použitie, a tak nieje treba programovať aplikácie znova. Každá má však svoje vlastné postupy pri extrakcii, a z toho plynie fakt že nie každá takáto aplikácia dokáže splniť požadované zadanie.

Medzi najznámejšie aplikácie v tomto odvetví však patria hlavne[6]:

- Scrapping-bot
- Octoparse
- Import.io
- Dexi.io
- Outwit

V niektorých prípadoch nám stačí jednoduché rozšírenie do prehliadača (napr. Web Scraper[2]) ktoré sú samozrejme efektívne hlavne pri menšom počte webstránok a dát určených na scraping.

Spomenuté aplikácie sú vo svojom odvetví veľmi populárne, avšak každá z nich má svoje limity. Pre skúsenejších používateľov ktorý majú záujem web scraping prispôsobiť svojim potrebám preto žiadna z nich nemusí byť ideálna cesta.

2.4.2 Platformy a knižnice

Pokiaľ však chceme vytvoriť vlastný program alebo aplikáciu typu web scraper, je potrebné rozlišovať na úrovni technológií dostupných pri programovaní. V prvom rade je treba vziať do úvahy programovací jazyk. V takom prípade je na výber rovno z niekoľko volne dostupných programovacích jazykov. Medzi dva najpopulárnejšie však patria hlavne Python, a Node.js. Každý má však znova svoje výhody a nevýhody, a preto je treba zvážiť na aké účely bude web scraper používaný. Vo všeobecnosti je populárnejší práve prvý spomínaný Python, vďaka jeho popularite z pohľadu jednoduchosti písania čistého kódu, komunity, a podpory ktorá je mu venovaná. Python ponúka veľké množstvo frameworkov, z ktorých tie najpopulárnejšie zahŕňajú hlavne Selenium[3] a Scrapy[4], ktoré som samozrejme zohľadnil pri výbere vhodného frameworku na vytvorenie web scrapera.

Na druhej strane Node.js stavia na základoch Javascriptu, ktorý je vyvíjaný priamo pre prácu s web stránkami. Kým Python je lepší čo sa týka podpory a rozšírenosti frameworkov, Javascript má výhodu v lepšej integrácií nielen s webovými stránkami ako takými, ale aj s webovými prehliadačmi samotnými. Python v mnohých prípadoch vyžaduje driver na komunikáciu s prehliadačom, a podpora dynamických stránok je znova jednoznačne lepšia pomocou Javascriptu, práve kvôli jeho spomínanej lepšej integrácií a faktom, že Javascript je jednou z hlavných súčastí dynamických moderných webových stránok. Javascript je jednou z hlavných prvkov modernej webovej stránky, ktorá sa skladá ešte z HTML a CSS. Jeho úlohou je hlavne vytváranie dynamického obsahu vytvárajúceho dojem že webová stránka s užívateľom interaguje. Vo väčšine prípadov beží iba na strane klienta, takže dynamickosť samotná nezaťažuje server. Zároveň je v mnohých prípadoch využívaná asynchrónnosť[5] tohto programovacieho jazyka, na zlepšenie odozvy a interakcie. Práve tento fakt sa stáva rozdielovým faktorom pri extrakcii dát z webových stránok, keďže obsah webovej stránky sa v mnohých prípadoch môže načítavať asynchrónne a nezávisle od ostatných častí webu.

[2] https://webscraper.io/

[3] https://www.selenium.dev/

[4] https://scrapy.org/

[5] Asynchrónnosť znamená v tomto prípade neblokovanie prehliadačových prostriedkov a načítavanie obsahu za behu webstránky.

Z môjho vlastného rozboru a výskumu som zistil, že dynamickosť a asynchrónnosť stránok je prvok ktorý je lepšie zvládaný za pomoci Javascriptu. [12][8]

Rozhodnutie používať Javascript som učinil zároveň s predpokladom využitia jedného z jeho volne dostupných frameworkov, a zároveň som zohľadnil svoje predošlé skúsenosti práve so spomínaným jazykom. K tomuto rozhodnutiu prispel aj fakt, že v dnešnej dobe je dynamika stránok takmer štandard, a tu hrá úloha Javascriptu veľkú rolu. V procese výberu frameworku som zohľadňoval hlavne integráciu daného frameworku s prehliadačom, výbornú podporu dynamických stránok a asynchrónnosť ktorá je pri takejto aplikácii kľúčová.

Ako prvý framewrok spomeniem určite Apify SDK, za ktorým stojí Česká firma Apify Technologies[6].[17] Práve Apify ma totiž inšpirovalo v rôznych odvetviach vývoja. Medzi najznámejšie technológie ktoré používajú Javascript patria najmä:

- Apify SDK
- Puppeteer
- Cheerio

Z pomedzi týchto technológií ktoré zdieľajú niektoré základné funkcie bola na zostavenie výslednej aplikácie použitá technológia Puppeteer. Puppeteer je zároveň jedna z technológií využívaných práve spomínanou Českou firmou Apify.

V porovnaní s technológiou Cheerio má Puppeteer svoje výhody. V niektorých faktoroch vyhráva ale aj Cheerio, a preto je treba zvážiť hlavne nasledovné:

- Možnosť scrapingu dynamických stránok
- Funkcionality a možnosti
- Kompatibilita webových technológií
- Rýchlosť analýzy a extrakcie

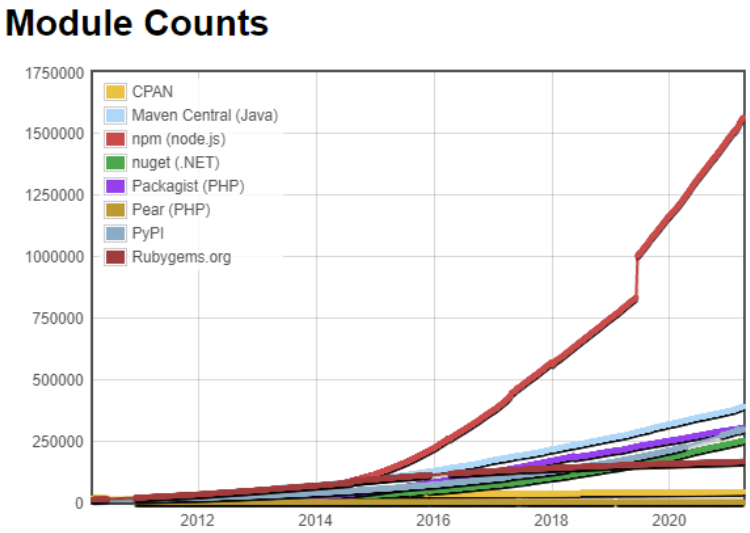
V prvom rade treba zvážiť aktuálnu situáciu a technológie ktoré sa na webe vyskytujú. 95% webových stránok v súčasnosti používa Javascript, takže predpoklad že moderné weby budú obsahovať dynamický obsah je prevažne jasný, a preto treba s takýmto obsahom počítať. Node.js ktorý je postavený na Javascripte je zároveň najrýchlejšie sa rozširujúcou technológiou ktorú web využíva. V čase písania má Node.js viac ako 1.5 milióna dostupných rozširujúcich balíčkov, a jeho náskok je v súčasnej dobe jednoznačný.[5]

Pri porovnávaní technológií Puppeteer a Cheerio je na prvý pohľad jednoznačné čo majú spoločné, a v čom naopak každá z nich vyniká. Tieto poznatky pomôžu jednoznačne určiť technológiu na základe požiadavkov ktoré vznikli zároveň s výberom témy tejto práce.

Cheerio a Puppeteer sú technológie založené na Node.js, a patria medzi najpopulárnejšie technológie v tomto odvetví. Ich jednoznačné vlastnosti však definujú nielen ich možné prípady použitia, ale zároveň schopnosti a rýchlosť akou sú schopné dosiahnuť očakávané výsledky.

Puppeteer je technológia ktorá je vyvíjaná za účelom automatizácie webového prehliadača, zatiaľčo hlavným určením Cheeria je web scraping samotný. Tým že je Cheerio určené

[6] https://apify.com/



Obr. 2.2: Porovnanie najpopulárnejších technológií a vývoj počtu ich modulov. Zdroj [3]

priamo na požadovanú úlohu je v tomto ohľade rýchlejšie a výhodnejšie čo sa nášho požiadavku týka. Cheerio je zároveň multiplatformné, a dokáže pracovať ako s prehliadačom Firefox, tak s prehliadačom Google Chrome. Na druhú stranu Puppeteer je viazaný iba na prehliadač Google Chrome vo forme Chromium. Puppeteer má priamo kontrolu nad prehliadačom, takže je možné ovládať prehliadač ako celok. Tento fakt sa dá brať aj ako výhoda aj ako nevýhoda, keďže je to pravdepodobne jeden z dôvodov prečo je Puppeteer pomalší vo vykonávaní funkcie Web Scrapera. Puppeteer však prináša schopnosť plynulej práce s Javascriptom na webovej stránke, dokáže parsovať aj webové stránky postavené na moderných technológiách ako React alebo Angular, ktoré sú v dnešnej dobe na vzostupe pre ich moderné prvky. Z priamej integrácie s prehliadačom vyplýva aj fakt že Puppeteer dokáže vytvárať snímky obrazovky počas načítavania a pracovania s webovou stránkou. Zároveň podporuje technológiu XML, a nerobí mu problém ako čítanie, tak ani parsovanie a priame spúšťanie Javascriptových funkcií. [14]

Práve podpora dodatočných funkcií a možnosť parsovania dynamických stránok s Javascriptom ma presvedčili o rozširovaní poznatkov technológie Puppeteer, a jej následné využitie pri návrhu architektúry a implementácie výslednej aplikácie.

Z uvedených informácií je pomerne jasné aké výhody a nevýhody dané riešenia majú, a čo ponúkajú. Ako už bolo spomenuté v úvode tejto práce, jej cieľom je zjednotiť hlavne vstupné požiadavky tak, aby sa jednotlivé časti týchto požiadavkov neviazali na jednotlivé webstránky. Popis návrhu je predstavený v nasledujúcej časti tejto kapitoly.

2.5 Navrhované riešenie

Cieľom tejto práce je vytvorenie aplikácie určenej na web scraping s unikátnymi vlastnosťami. Kľúčové vlastnosti aplikácie budú tvorené hlavne vstupným a výstupným formátom, postupom ktorý aplikácia bude na extrakciu dát využívať, a dôrazom na dodržaní univerzálnosti vstupných pravidiel.

Univerzálnosť vstupných pravidiel je v tomto prípade myslená v zmysle využiteľnosti raz definovaných požiadaviek na viaceré webové dokumenty a datasety. Definícia vstupných pravidiel bude rozdelená na hlavné a vedľajšie pravidlá tak, aby bolo možné odlíšiť vnútornú konfiguráciu aplikácie a užívateľské pravidlá za účelom jednoznačne určiť ich prioritu a typ využitia.

Finálna aplikácia bude vyvíjaná pomocou technológie Puppeteer najmä vďaka jeho jednoznačnej integrácií s webovým prehliadačom. To sa na prvý pohľad nemusí zdať dôležité. Jedná sa predsa o aplikáciu na extrakciu dát, nie aplikáciu ktorá slúži na automatizáciu prehliadača ako takého. Údaje z webovej stránky sa predsa dajú získať jednoduchšie, bez využitia webového prehliadača, ktorý celý proces len spomaľuje.

To je síce pravda, avšak spomínaná podpora dynamických stránok, a stránok postavených na moderných technológiách ako napríklad React a Angular je značne limitovaná. Načítanie týchto stránok tak aby ich zobrazenie zodpovedalo realite je prehliadačom Chrome zvládnuté na výbornú, a protokol Chrome DevTools dokáže zabezpečiť komunikáciu na vysokej úrovni, ktorá dané dáta prenesie tak ako bolo zamýšľané. Po naviazaní spojenia má Puppeteer plnú kontrolu nad prehliadačom, a extrakcia dát môže začať.

Postup ktorý bude aplikácia na analyzovanie a detekciu obsahu používať bude popísaný v nasledovných kapitolách. Mal by byť ale efektívny, s dôrazom na presnosť a rýchlosť akou dokáže rozpoznať že sa jedná o falošné alebo pravé výsledky. To bude jednoznačne určené porovnaním s definovanými vstupnými pravidlami.

Už samotný protokol DevTools používa technológiu JSON kvôli jeho jednoznačným výhodám v oblasti prenostiteľnosti medzi nielen aplikáciami ale aj programovacími jazykmi. Komunikácia výslednej aplikácie z pohľadu vstupných a výstupných dát by preto mala tiež využívať práve túto technológiu. Definovanie pravidiel bude v tomto formáte pre človeka jednoduché na zápis a pre stroj jednoduché na čítanie.

Formát JSON bude zároveň použitý aj na spätnú komunikáciu kde sa v tomto formáte budú po úspešnom dokončení procesu nachádzať vygenerované výsledky. Tieto výsledky budú tak pripravené na ďalšie využitie v praxi.

Pri návrhu riešenia boli brané v úvahu aktuálne dostupné aplikácie a frameworky.

Kapitola 3

Technológie

Táto kapitola sa zaoberá popisom technológií, modulov a prvkov, na základe ktorých bude následne zostavená aplikácia tak, aby bolo čo najefektívnejšie nielen jej využívanie, ale aj údržba. Zároveň treba brať v úvahu aj prívetivosť a jednoduchosť užívateľského vstupu, a formát, využiteľnosť a prenositeľnosť výstupných dát.

K výberu týchto technológií boli využité poznatky z Kapitoly 2, ktoré ďalej určovali smer návrhu a vývoja aplikácie.

3.1 HTML

Základným stavebným blokom každej webovej stránky je nepochybne HTML. HTML je skratka pre *Hypertext Markup Language*, a už svojím názvom napovedá akú úlohu vo webových technológiách zohráva.

HTML dokument sa skladá z elementov, ktoré sú predstavené v podobe takzvaných *značiek*. Tieto značky sú reprezentované pomocou znakov <> a dovoľujú určovať typ a rozsah uvedeného obsahu. Tieto elementy môžu obsahovať zároveň rôzne ďalšie atribúty, ako napríklad trieda elementu, alebo identifikátor. Takéto atribúty sú potom využívané v technológiách ktoré s HTML dokumentami spolupracujú, ako napríklad CSS a Javascript.

Takéto dokumenty sú následne reprezentované vo webovom prehliadači, ktorý HTML dokumenty zobrazí zároveň s doplnkovými dokumentami ktoré určujú štylizáciu a správanie sa danej webstránky. HTML dokumenty samotné neobsahujú žiadnu štylizáciu, avšak typ elementu ktorý použijeme pre realizovaní určitého prvku je kritický, pretože udáva základné rozloženie obsahu ktorý element predstavuje.[16]

3.2 DOM

Jazyk HTML nieje určený na programovacie účely, a preto je potrebné pri práci s webovými stránkami zvážiť aj technológiu DOM. DOM je skratka ktorá reprezentuje *Document Object Model*. Predstavuje akési prepojenie medzi HTML a programovacím jazykom ktorý chce s webovou stránkou komunikovať. [7]

Pri komunikácií s programovacím jazykom DOM reprezentuje stránku v dvoch hlavných podobách:

- Uzly
- Objekty

Práve tento fakt nám ďalej dovoľuje s webovou stránkou interagovať a meniť jej obsah. Príkladom takéhoto využitia jazyk Javascript, ktorý bol vyvinutý za účelom vytvorenia dojmu interakcie užívateľa s webovou stránkou.

DOM je zároveň veľmi dôležitý pri web scrapingu, keďže práve komunikácia s webovou stránkou na úrovni programovacieho jazyka je kľúčová pri extrahovaní požadovaných údajov z webových dokumentov. Takáto extrakcia je možná vďaka reprezentácií webovej stránky v objektovo orientovanej podobe.

3.3 CSS

CSS predstavuje istý jazyk, ktorý je používaný za účelom popisu štylizácie HTML dokumentov. CSS znamená *Cascading Style Sheets*, kde Cascading predstavuje jedno z hlavným pravidiel tohto jazyka - štylizačné pravidlá majú určenú prioritu podľa ktorej budú aplikované, a zároveň umožňujú na jednej webovej stránke používať viacero CSS súborov.

Primárne určenie je už spomínaná štylizácia a definovanie rozloženia elementov na webovej stránke. Zároveň môžu definovať vzhľad v závislosti od zariadenia ktoré webovú stránku v danom momente navštívilo.

HTML samotné nebolo vyvinuté za účelom definovania vzhľadu, a preto bolo nutné vyvinúť technológiu ktorá tento nedostatok kompenzovala. CSS pravidlá sú zväčša aplikované na atribút triedy, ktorý bližšie špecifikuje skupinu prvkov v HTML dokumente. V niektorých prípadoch sa však pravdilá aplikujú aj na celé elementy, prípadne identifikátory.[15]

Pri vytváraní CSS pravidiel rozlišujeme 3 hlavné miesta kde sa tieto pravidlá môžu nachádzať:

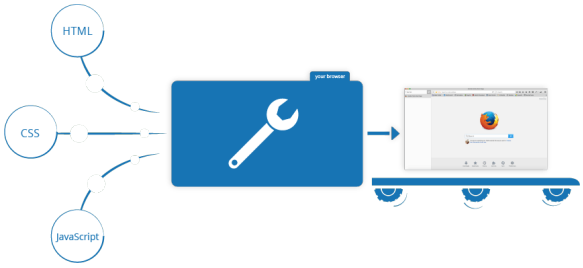
- Riadkové
- Interné
- Externé

Rozlišovanie medzi týmito typmi je dôležité, pretože z názvu *Cascading* vyplýva že každé miesto môže mať inú prioritu finálneho vykonania daného pravidla.

3.4 Javascript

Programovací jazyk Javascript bol vyvinutý za účelom vytvárania dojmu interakcie pri webových stránkach. Javascript je jednou z hlavných súčastí webových stránok, a je používaný na viac ako 95% stránkach celosvetovo.[5] Dá sa preto predpokladať že bude jeho integrácia s webovými prehliadačmi a webovými stránkami na vysokej úrovni.

Práve kombinácia HTML, CSS a Javascriptu totiž dokáže docieliť výsledky ktoré sa pri moderných štandardoch vyžadujú. Či už sa jedná o dynamickú zmenu obsahu alebo animovanie niektorých elementov, je to práve Javascript ktorý túto interakciu docieľuje. [8]



Obr. 3.1: HTML, CSS a JavaScript sú 3 hlavné zložky ktoré spolu vytvárajú web ako ho poznáme dnes. Zdroj [8]

Javascript je založený na programovacom jazyku Java, ktorý bol ďalej modifikovaný a prispôsobený na prácu priamo vo webovom prehliadači. Javascript dokáže nielen reagovať na vstupy od užívateľa, ale reagovať aj na niektoré udalosti ktoré sa stanú na pozadí, v prehliadači. [8]

Táto technológia ďalej stavia aj na asynchrónnosti, teda prístupu za behu. To dovoľuje stránkam načítať Javascript až po načítaní webového obsahu tak, aby nevznikali zbytočné dlhé čakacie doby na načítanie webstránky. Zároveň sa dá asynchrónnosť využiť pri odosielaní formulárov, kde nieje nutné po odoslaní znova načítavať stránku. Tým sa zároveň zlepšuje responzívnosť, keďže nieje nutné čakať na vykonanie špecifickej akcie.

Jednou z najpopulárnejších vlastností tohto programovacieho jazyka je podpora API[1]. Podpora API umožňuje využitie naprogramovaných rozhraní bez toho aby si tieto rozhrania užívateľ programoval sám. Tento fakt urýchľuje vývoj webových aplikácií a pomáha udržiavať štandard. API pri Javascripte sa primárne rozdeľujú do dvoch kategórií:

- Prehliadačové API
- Externé API

Kde prehliadačové API predstavuje napríklad aj spomínaný DOM, ktorý javascriptu poskytuje HTML elementy vo forme objektov, s ktorými javascript dokáže pracovať. Čo sa externých API týka, sem patria najmä API tretích strán, ako napríklad Google Maps alebo Twitter.

V mnohých prípadoch sú to práve API, ktoré môžu naraziť na väčšie množstvo problémov pri načítavaní webovej stránky. Prispieva k tomu aj asynchrónnosť, kde v prípade že sa HTML dokument načíta skôr ako Javascriptové súbory, môže nastať problém kde vykonávanie skriptu zlyhá. Asynchrónnosť zároveň súvisí s postupnosťou vykonávania kódu. V prípade že je využitá asynchrónna funkcia, jej výsledok je reprezentovaný v podobe *Promise*. Výsledok Promise nemusí byť dostupný okamžite, a preto v prípadoch že sa na túto skutočnosť neberie ohľad môže vykonávanie funkcie zlyhať.[8]

[1] Application Programming Interfaces

3.4.1 JSON

Objekty sú v Javascripte popísané pomocou formátu JSON. JSON je skratka pre *JavaScript Object Notation* kde už z názvu vyplýva jeho pôvod. Je to jednoduchá forma popisu objektov tak, že je vo výsledku ľahko čitateľná človekom, a zároveň jednoducho parsovatelná počítačom. [2]

Aj keď je súčasťou názvu *Javascript*, JSON je univerzálny spôsob zápisu objektov a je to formát adaptovaný mnohými programovacími jazykmi súčasnosti. Práve tieto vlastnosti robia z tohto formátu multiplatformný spôsob prenosu informácií.[2]

JSON ponúka dve základné štruktúry:

- Objekt
- Pole

Tieto základné štruktúry sú podporované modernými prehliadačmi a programovacími jazykmi ktoré sa v tomto zmysle používajú.[2]

3.4.2 Node.js

Node.js ako runtime Javascritu bol vytvorený hlavne za účelom spracovávania asynchrónnych aplikácií. Beží na Javascriptovom rozhraní V8 ktoré bolo pôvodne vytvorené pre webový prehliadač Google Chrome. Práve vytvorenie tohto rozhrania viedlo k neskoršiemu vytvoreniu Node.js, pretože toto rozhranie dokázalo urýchliť Javascript kód pomocou priamej interpretácie tohto kódu do jazyku počítača. Node.js ako runtime predstavoval možnosť spúšťania Javascriptového kódu mimo webového prehliadača, priamo na fyzickom počítači. To viedlo k vytváraniu aplikácií priamo spustiteľných či už na počítači alebo smartfóne. Neskôr sa pridali rôzne moduly, ako napríklad HTTP, ktoré umožňovali ďalšie rozšírenie pôsobenia, a to napríklad aj na servery tak, aby mohol Node.js pôsobiť ako samostatné prostredie na strane servera. Node.js je v dnešnej dobe využívané zároveň pri technológiách ako napríklad React a Angular ktoré predstavujú frameworky založené a fungujúce práve vďaka Node.js.[9]

Node.js ponúka zároveň možnosť využitia rôznych modulov ktoré boli vyvinuté za účelom rozšírenia vlastností a funkcionality. Node.js sa aj vďaka týmto modulom a faktu že s jeho príchodom je možné Javascript využívať ako na backende tak aj na frontende stáva viac a viac populárnym. O tom svedčia aj jeho štatistiky z pohľadu vývoja týchto modulov. Moduly sú ďalej pre užívateľov distribuované pomocou NPM.

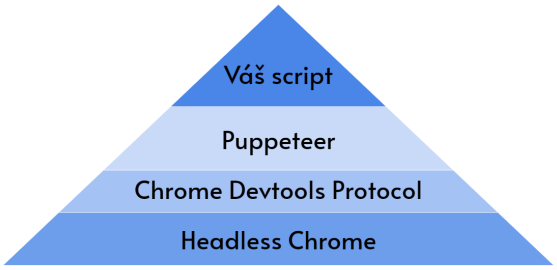
3.4.3 NPM

Jednou z najdôležitejších súčastí Node.js je NPM(Node Package Manager). Jedná sa o najväčšie a najpopulárnejšie úložisko doplnkových modulov a projektov pre Node.js. Jeho popularita stále stúpa a aktuálne sa v ňom nachádza už viac ako 1.5 milióna rôznych projektov 2.2[10]. A práve tu sa nachádza aj knižnica Puppeteer.

3.4.4 Puppeteer

Hlavnou súčasťou výslednej aplikácie je Puppeteer. Je to jedna z mnoha knižníc vytvorených pre Node.js distribuovaná pomocou NPM. V aplikácií zohráva dôležitú úlohu, keďže práve technológia Puppeteer je spôsob akým aplikácia komunikuje a extrahuje údaje z prehliadača.

Spojenie s prehliadačom je naviazané pomocou protokolu DevTools vyvinutého spoločnosťou Google, a používa prehliadač Google Chrome alebo Chromium v špeciálnom Headless móde, kedy nieje zobrazená vizuálna inštancia prehliadača, avšak všetky ostatné funkcie sú plne dostupné. [13]



Obr. 3.2: Hierarchia výslednej aplikácie pri použití technológie Puppeteer.

Z uvedenej hierarchie je na prvý pohľad jasné, na ktorých technológiách Puppeteer stavia:

1. **Headless Chrome** je spôsob spustenia prostredia Chromium v takzvanom headless prostredí, kedy sa inštancia programu Google Chrome alebo Chromium spustí v režime server, bez užívateľského prostredia. Tento spôsob je pri používaní Puppeteeru nastavený ako základný, avšak je možné ho vypnúť a používať Chromium v headfull móde, teda so zobrazeným užívateľským prostredím.[1]

2. **Chrome DevTools protokol** bol založený za účelom automatizácie prehliadača Google Chrome, Chromium a ostatných webových prehliadačoch založených na prostredí Blink. Predstavuje komunikačný prvok medzi programom a webovým prehliadačom. Komunikačné kanály sú rozdelené podľa domén využitia a komunikácia medzi prehliadačom a aplikáciou ďalej prebieha využitím formátu JSON.[4]

3. **Puppeteer** ďalej nadväzuje na protokol DevTools v podobe API ktoré týmto protokolom dokáže komunikovať s inštanciou prehliadača Chrome.

4. **Váš script** následne využíva knižnicu Puppeteer ktorá prichádza s radom revolučných riešení umožňujúcich automatizovať takmer čokoľvek.

Puppeteer bol vytvorený za účelom kompletnej automatizácie správania prehliadača Google Chrome. Toto zahrňuje aj zber štatistík o používaní, a vykonávaní testov rôznych aplikácií.

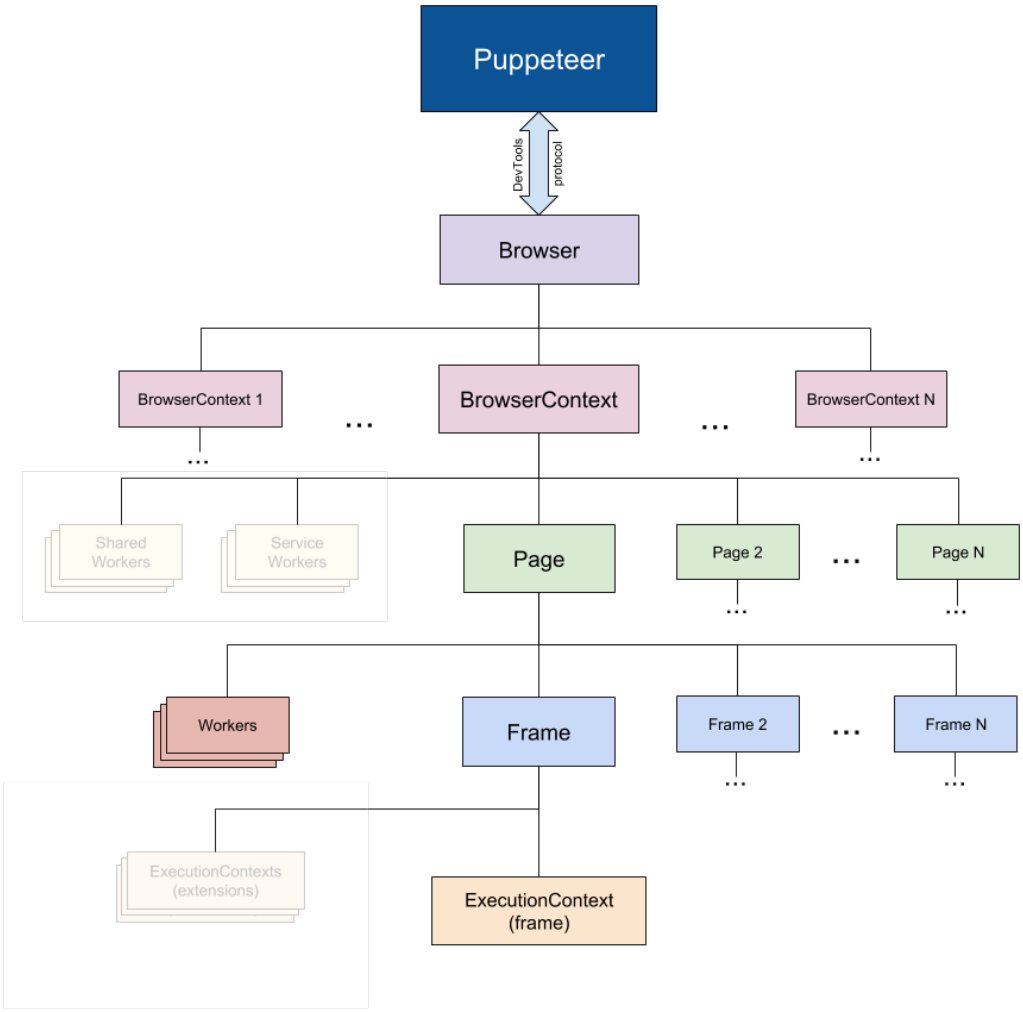
Niektoré z ďalších prípadných využití tejto knižnice zahŕňajú aj:[13]

- Vytváranie snímky obrazovky z webových stránok
- Automatizácia formulárov
- vytvorenie testovacieho prostredia pre akúkoľvek aplikáciu

Online verzia Puppeteeru je zároveň dostupná v sandbox prostredí[2] pre jednoduché vyskúšanie jeho možností.

Vo výslednej aplikácii je Puppeteer využívaný v rámci hlavného procesu, kde sa ako prvé vytvorí inštancia puppeteeru za pomoci príkazu *await puppeteer.launch();* a následnom vytvorení inštancie stránky v podobe *about:blank* za pomoci príkazu *await browser.newPage();* ktorá následne slúži ako prostredie pre načítanie a následnú extrakciu dát z danej webovej stránky. Práca s knižnicou Puppeteer je napriek širokej škále funkcií jednoduchá.

[2] https://try-puppeteer.appspot.com/



Obr. 3.3: Diagram knižnice Puppeteer. Zdroj: https://github.com/puppeteer/puppeteer/blob/main/docs/api.md

Kapitola 4

Návrh architektúry aplikácie

Úlohou tejto kapitoly je oboznámenie s návrhom architektúry finálnej aplikácie. Architektúra sa skladá z definovaní požiadavkov na vstupné dáta, výstupné dáta a vnútorné chovanie aplikácie. V úvahu je samozrejme braná aj aktuálna situácia na trhu, a s tým spojená analýza ktorá určovala smer vývoja pri dôležitých otázkach ohľadne vnútornej štruktúry.

4.1 Špecifikácia požiadaviek

Ako už názov práce predpovedá, jedná sa o extrakciu dát z webových stránok. Avšak dáta sa z webových stránok dajú extrahovať mnohými spôsobmi, tie však niesu ekvivalentné. A to nielen z pohľadu presnosti, ale aj z pohľadu rýchlosti extrakcie, spôsobu extrakcie, a ďalších možnostiach ktoré sú viazané na spôsob aký je vo finálnej fáze zvolený.

Základom extrakcie je zároveň vedieť o aké dáta máme záujem. Tieto dáta teda musia byť jednou z hlavných častí ktoré budú definované na vstupe aplikácie. To v akom formáte a v akej podobe budú tieto dáta reprezentované bude popísané v nasledujúcich častiach tejto kapitoly. Už teraz je však isté, že pre dosiahnutie pomerne vysokej presnosti bude potrebná špecifikácia rôznych pravidiel. Tieto pravidlá potom budú udávať mernú jednotku na porovnávanie nájdených potencionálnych výsledkov s definovanými pravidlami.

Úlohou týchto pravidiel bude určenie relevantnosti nájdených dát. Aplikácia bude preto tieto vstupné pravidlá overovať z pohľadu správnosti formátu. Pri definícií týchto pravidiel bude treba pravdepodobne zohľadniť diverzitu a premenlivosť výsledného prvku, napríklad premenlivú dĺžku reťazcov. Dá sa teda s určitosťou predpokladať že správna definícia vstupných pravidiel bude mať jednoznačný vplyv na úspešnosť výsledkov z hľadiska nášho očakávania. Preto je treba tomuto návrhu venovať dostatočne veľký časový priestor.

Pri návrhu je treba samozrejme zohľadniť fakt, že finálny používateľ bude chcieť špecifikáciou týchto pravidiel stráviť čo najmenej času. To je samozrejmé, a je preto odlíšiť časti pravidiel ktoré nieje nutné prepisovať tak často od pravidiel ktoré sa môžu meniť s príchodom nových požiadaviek.

4.2 Vstupné pravidlá

Pojmom vstupné pravidlá rozumieme pravidlá ktoré sú definované ako základné vstupné údaje pri spustení aplikácie. Tieto údaje budú mať za úlohu definovať presne to, čo v danom konkrétnom prípade užívateľ aplikácie očakáva ako aplikačný výstup. Sú to zároveň pravidlá pre aplikáciu, ktoré určujú objekty vo webovom dokumente určené na extrakciu.

Ako formát týchto pravidiel bol vybraný už spomínaný JSON, vďaka jeho jednoduchej zrozumiteľnosti ako z pohľadu človeka, tak aj z pohľadu počítača. Súbory vo formáte JSON sú zároveň veľmi jednoducho prenositelné aj medzi viacerými programovacími jazykmi, čo je samozrejme ďalšia výhoda. S týmto formátom mám zároveň osobné skúsenosti, takže to bola pomerne jasná voľba.

4.2.1 Hlavné pravidlá

V prvom rade treba začať prvotným upresnením, a teda z čoho sa vlastne tieto pravidlá budú skladať. S ohľadom na užívateľskú jednoduchosť, boli hlavné a vedľajšie pravidla rozdelené do dvoch konfiguračných súborov, pre ich jednoduchšiu správu.

Medzi hlavné pravidlá na extrakciu boli vybraté:

- Zoznam webových stránok v podobe URL
- Štruktúra hľadaných prvkov
- Voliteľný atribút blacklist

Kde **zoznam webových stránok** určuje všetky webové stránky ktoré sú určené na extrakciu. Budú to webové stránky ktoré budú ako atribút posielané aplikácii a výsledná aplikácia ich bude prechádzať v takom poradí, v akom boli zadané. V prípade že je niektorá zo stránok nedostupná, aplikácia o tom užívateľa informuje, a bude pokračovať nasledovnou webovou stránkou. Pokračovanie namiesto ukončenia aplikácie je zvolené z dôvodu neustáleho vývoja webu. V prípade že jedna z preddefinovaných stránok zmení adresu, a táto adresa už nebude dostupná, by toto nemalo ovplyvniť vygenerovanie výsledkov ostatných adries.

Štruktúra hľadaných prvkov by mala byť definovaná ako objekt určujúci požadovaný výsledný názov elementu, teda jeho jednoznačný identifikátor, a jeho „pseudo“ dátový typ. Kde jeho identifikátor bude použitý zároveň pri finálnom výpise výsledných údajov, a dátový typ použitý podľa pravidiel ktoré budú popísané neskôr v tejto kapitole. Štruktúra je vo formáte objektu, a teda jednotlivé identifikátory a dátové typy sú k sebe priradené tak aby aj pre človeka bolo na prvý pohľad jasné v akom vzťahu tieto údaje sú.

Blacklist je voliteľný atribút (nemusí byť vo finálnom vstupnom súbore reprezentovaný). Predstavuje pole prvkov vo formáte reťazcov, ktoré užívateľ definuje ako také, ktoré by sa vo výslednom súbore nemali nachádzať. Definovanie takýchto reťazcov pomocou Blacklistu bude samozrejme mať za následok ovplyvnenie aplikácie pri rozhodovaní o pozícii a formáte výsledných údajov.

4.2.2 Vedľajšie pravidlá

Ako vedľajšie pravidlá boli zvolené také pravidlá, ktorých prenositeľnosť aj medzi rôznymi typmi definovaných domén je pravdepodobnejšia, a preto by nemuselo byť vždy nutné tieto pravidlá meniť. Zmena týchto pravidiel sa teda berie ako menej častá udalosť, a vo väčšine prípadov môže stačiť definovať hlavné pravidlá.

Medzi tieto pravidlá boli zaradené:

- Maximálny pomer zlyhania
- Formát
- Primárny prvok

Ako **maximálny pomer zlyhania** je určený výsledný pomer nesprávnych prvkov ku všetkým nájdeným prvkom z daného testovacieho okruhu. V prípade že je tento pomer vyšší ako definovaný je aktuálny testovací okruh zahodený a test prebieha nanovo s novým testovacím okruhom. Tento testovací okruh je reprezentovaný dátami, ktoré sú porovnávané s primárnym prvkom. Ten určuje či aktuálny kontrolovaný prvok spĺňa regulárny výraz primárneho prvku. V prípade že ho spĺňa je prvok nezmenený, v prípade že ho nespĺňa je prvok nahradený výrazom *NULL* ktorý v následnom výpočte poslúži ako súčet prvkov ktorých kontrola „zlyhala“.

V prípade že je aktuálny pomer zlyhania menší ako maximálny pomer zlyhania definovaný vo vstupnom súbore, je daný testovací okruh braný za správny, uloží sa identifikátor okruhu a aplikácia pokračuje ďalšími krokmi ako napríklad generovanie výsledkov.

Formát predstavuje objekt ktorý sa skladá z názvu definovaného dátového typu a jeho korešpondujúcej reprezentácie v podobe jednoznačného regulárneho výrazu. Na dátové typy v tomto objekte naväzuje štruktúra prvkov definovaných v hlavných pravidlách. Štruktúra využíva práve dané regulárne výrazy na overenie správnosti zaradenia výsledného prvku.

Dá sa predpokladať že správne definovaný regulárny výraz bude mať dramatický vplyv na výsledné dáta. V prípade že test daného prvku na predpokladaný regulárny výraz neprejde, nieje možné určiť jeho dátový typ a teda výsledný jednoznačný identifikátor. V takom prípade nieje možné garantovať zaradenie takéhoto prvku do výsledného súboru dát.

Primárny prvok je taký prvok, ktorý je určený ako prvok ktorý sa v objekte určenom na extrakciu opakuje, a je teda možné sa na jeho prezenciu spoliehať. Tento prvok definuje objekt, a je možné nejakú jeho deriváciu nájsť v každom objekte ktorého obsah chceme extrahovať.

Jeho reprezentácia je znova v podobe regulárneho výrazu, avšak on samotný sa nemusí nachádzať v definovanej štruktúre ani formáte vstupných údajov. Aplikácia tento prvok využíva na prvotnú analýzu stránky ako takej, a na základe jeho prezencie v danom testovacom okruhu je posúdené či daný testovaný okruh je kandidátom na definíciu správnych výsledných objektov.

4.3 Výstupný formát

Čo sa výstupného formátu týka, mal by byť znovu pokiaľ možno v univerzálnom formáte. Keďže sa dá predpokladať že extrahované dáta budú v prípade potreby použité, je vhodné zvoliť formát ktorý je podporovaný rôznymi aplikáciami a programovacími jazykmi.

Z toho dôvodu tu bol znovu zvolený formát JSON. Návrh počíta s jedným výstupným súborom pre jednu doménu tak, aby výstupný súbor niesol názov domény z ktorej extrahované údaje pochádzajú. Umiestnenie týchto výsledných súborov je už však na užívateľovi, avšak pri nešpecifikovaní parametra určujúceho cestu k výstupným súborom bude použitá základná zložka s názvom *output*.

Štruktúra v akej budú jednotlivé extrahované objekty reprezentované je zhodná so štruktúrou definovanou v užívateľskom vstupnom súbore. Táto štruktúra je potom zachovaná pre všetky objekty. Ak však nejaký kľúč z daného objektu nieje dostupný alebo nebol extrahovaný, je tento kľúč vynechaný. V niektorých prípadoch ak sa dátový typ daného prvku nepodarí overiť, a konfigurácia aplikácie je spustená bez parametru ktorý zaručuje odstránenie nedefinovaných prvkov z objektu, sa v objekte môžu nachádzať kľúče so špeciálnym názvom *undefined*. Tento názov značí prvok, ktorého dátový typ nebolo možné na základe špecifikovaných vstupných požiadavkov definovať, avšak patrí do vyhľadávanej podskupiny.

Formát JSON ako výstupný formát je striktne dodržaný a riadne formátovaný tak, aby bol jednoducho čitateľný aj pre človeka. Bezproblémové využitie dát z takto štruktúrovaného výstupu je preto garantované.

Sekundárnym výstupným súborom sú takzvané metadáta. Sú to dáta ktoré predstavujú rôzne vlastnosti popisujúce priebeh extrakcie ktorá prebehla pri danom spustení aplikácie. Metadáta sú znova pre jednoduchú prenositeľnosť vo formáte JSON, a jeho objekty obsahujú tieto údaje:

- Url adresa webového dokumentu
- Počet testovaných okruhov
- Odkaz na prvý nájdený objekt v podobe *document.querySelectorAll*
- Počet výsledných objektov
- Čas potrebný na otestovanie okruhov
- Čas potrebný na uloženie výsledkov
- Čas potrebný na načítanie webového dokumentu

Tieto metadáta môžu poslúžiť ako kontrola výsledných súborov a porovnanie časov potrebných na vykonanie určitých akcií. Metadáta sú generované pri každom spustení aplikácie, a ich obsah je prepisovaný tak, aby obsahoval vždy údaje o poslednom spustení.

Konzolový výstup aplikácie

Aplikácia zároveň pri každom spustení a jej behu informuje užívateľa o aktuálne prevádzanej akcii. V režime bez vypisovania explicitných informácií aplikácia informuje o začatí exekúcie formou vypísania *Execution stared*. Následne po skontrolovaní vstupných údajov informuje o začatí extrakcie vypísaním *BEGIN PRIMARY EXTRACTION*. Následne sa vypisujú aktuálne URL, na ktorých extrakcii sa v danom momente pracuje.

Na záver aplikácia informuje o úspešnosti extrakcie, o lokácii vygenerovaného súboru s metadátami a o finálnej dĺžke trvania celého procesu v milisekundách.

V prípade spustenia aplikácie s prepínačom *-v* je na výstup vypísaný podrobnejší obsah. V tomto prípade sa na konzolovom výstupe objavia časy jednotlivých akcií, informácie o počte výsledných objektov a zoznam testovaných okruhov.

4.4 Architektúra aplikácie

V tejto časti práce bude popísaná architektúra aplikácie tak, ako bola aplikácia navrhnutá, a odôvodnenie týchto krokov.

4.4.1 Jadro programu

Hlavný zdrojový kód sa bude nachádzať v jednom súbore ktorý pre jednoznačnosť v rámci tejto práce nazvem *extractor.js*. Koncovka js napovedá že sa bude jednať o Javascriptový súbor. Pri spúšťaní aplikácie sa preto bude tento súbor volať pomocou príkazu *node* ktorý značí Node.js.

Aplikácia bude naväzovať na dva prídavné súbory vo formáte JSON, ktoré budú predstavovať vstupné informácie vo formáte popísanom v predošlej časti tejto kapitoly.

Hlavný komunikačný kanál aplikácie a webového prehliadača bude zabezpečený pomocou technológie Puppeteer a protokolu Chrome DevTools. Toto spojenie bude nadviazané automaticky po spustení aplikácie. Pred naviazaním tohto spojenia však dôjde ku kontrole vstupných dát zadaných používateľom. V prípade že je formát vstupných dát poškodený alebo nedodržaný, alebo vo vstupných súboroch chýbajú niektoré dôležité položky, je o tom používateľ informovaný a vykonávanie je ukončené.

4.4.2 Hlavný proces

Úlohou hlavného procesu bude vymedzenie finálneho okruhu výsledkov. Čo sa týchto okruhov týka, ich výber prešiel rôznymi iteráciami, avšak nakoniec bol zvolený spôsob určovania týchto okruhov na základe atribútu *class*. Aplikácia teda najskôr analyzuje stránku tak, aby bolo možné vygenerovať zoznam používaných atribútov. Na základe tohto zoznamu následne aplikácia generuje výsledné okruhy ktoré sú naplnené údajmi z daného okruhu definovaného *class* atribútom.

Aby bolo možné určiť správnosť údajov, je každý údaj porovnávaný zvlášť. Údaje sú porovnávané s reprezentujúcim pravidlom regulárneho výrazu, v tomto prípade primárnym prvkom zo vstupných pravidiel.

Analýza výsledkov

Po určení správneho okruhu, bude následne aplikácia pracovať s daným *class* atribútom ktorý daný okruh dát definuje. Aplikácia berie daný údaje daného okruhu ako potencionálnu súčasť výsledkov, nemusí to byť ale tak vždy. Preto aplikácia kontroluje susedné prvky z pohľadu DOM, a snaží sa získať čo najviac informácií nielen o susedných, ale aj rodičovských uzloch a detských potomkoch daného prvku.

Po zoskupení týchto informácií a uložení textových reprezentácií týchto prvkov aplikácia prechádza k ďalšiemu kroku, ktorý predstavuje finálnu definíciu „pseudo“ dátového typu určeného užívateľom.

Určenie tohto typu nieje vždy jednoduché, a preto v prípade že špecifikácia regulárneho výrazu nieje dostatočne presná, alebo je z iného dôvodu časť dát nedefinovateľná, je takýto prvok označený výrazom *undefined*. Nedefinované položky však vo výsledku užívateľovi ani inému programu ktorý by dáta využíval nemôžu byť užitočné. Práve z toho dôvodu má užívateľ týmto prípadom predchádzať, a to dvoma spôsobmi.

Keďže aplikácia analyzuje výsledky ako celok, vie presne určiť aký dátový typ bol na danom mieste definovaný v predošlých prípadoch objektov, ktorých dáta testom na regulárny výraz prešli. Týmto sa dá jednoducho predísť strate dát pri anomáliách. V niektorých prípadoch avšak ani toto nieje riešenie, v tom prípade má aplikácia možnosť jednoducho takéto nedefinované výsledky vynechať a užívateľovi vôbec nezobrazovať.

Generovanie výsledných údajov

Po analýze výsledkov a ich zaradení k dátovým typom prichádza na rad generovanie výsledných údajov. Aplikácia v tomto prípade generuje výsledky pre každý webový dokument zvlášť, a to tak, že výsledný súbor nesie názov v podobe URL stránky, z ktorej údaje pochádzajú.

Po vytvorení výsledného súboru sú analyzované údaje transformované do podoby JSON, ktorý je zároveň riadne formátovaný pre jednoduchú čitateľnosť ako človekom, tak aj počítačom. Po vygenerovaní všetkých potrebných dát je súbor uzavretý a výsledky sú pripravené na použitie. Pred samotným ukončením aplikácie však prichádza na rad ešte záverečné generovanie metadát, ktorých úlohou je informovať o celkových výsledkoch extrakcie.

Metadáta predstavujú akési dáta o dátach. Ich generovaním dávame používateľovi informácie ktoré môže pokladať za užitočné, jasný a jednoduchý prehľad o extrakcii z pohľadu počtu extrahovaných elementov a dĺžky trvania tejto akcie.

Kapitola 5

Implementácia riešenia

V nasledujúcej kapitole bude popísaný postup implementácie výsledného riešenia. Aplikácia typu web scraper bude implementovaná na operačnom systéme Windows. Jej využiteľnosť je samozrejme multiplatformná a žiadna z jej súčastí nieje na operačný systém Windows viazaná. Aplikácia bola vo finálnej fáze otestovaná aj na operačnom systéme Manjaro, pod ktorým bežala tak ako bola navrhnutá.

Implementácia bola z pohľadu web scrapingu rozdelená do viacerých častí:

- Konfigurácia
- Vstupné dáta
- Jadro aplikácie

Postup práce pri implementácií bol vďaka vytvoreniu návrhu a konzultáciám s vedúcim práce pánom Radkom Burgetom jednoznačný. Najskôr bola implementovaná aplikačná logika, a následne bola táto logika rozvedená. Počas implementácie boli zároveň vyskúšané dva rôzne prístupy k problematike. Tieto prístupy sa týkali hlavne rozlíšenia prístupu k vytváraní dátových okruhov. Po skúmaní som pristúpil k vytváraní týchto okruhov dát na základe atribútu *class*. Bližšie informácie o tom ako aplikácia dané okruhy vytvára a analyzuje budú popísané v rámci tejto kapitoly.

5.1 Konfiguračný súbor

Externá konfigurácia dostupná pre užívateľa je uložená v konfiguračnom súbore, ktorý môže užívateľ volne upravovať a dokonca súbor úplne vymeniť. Základný konfiguračný súbor je k výslednej implementácií pripojený, a aplikácia ho očakáva implicitne ako súbor *config.json*.

Jeho úlohou je definícia základných pravidiel ktoré budú pri extrakcii dát z webových stránok používané. Formát a obsah tohto súboru je preto jednoznačne daný, a musí byť dodržaný. V opačnom prípade hrozí že aplikácia takýto súbor odmietne a nebude možné v extrakcii dát pokračovať. Aplikácia v takomto prípade užívateľa zároveň informuje o danej situácii chybovým hlásením zobrazeným v príslušnom termináli.

Štruktúra konfiguračného súboru je daná nasledovne:

- **maxFailRatio** ako hodnota v rozsahu 0-100 ktorá definuje maximálny pomer prvkov ktoré pri určovaní výsledného okruhu neboli úspešné. Tieto prvky sú vo výslednom okruhu dát predstavené hodnotou *NULL*. Ak je výsledný pomer týchto prvkov väčší ako hodnota udaná v *maxFailRatio* je takýto okruh vyradený, a neberie sa pri ďalšom určovaní v úvahu. Táto hodnota je úzko viazaná s hodnotou *primary* ktorá bude popísaná neskôr. Zmenu hodnoty na číslo blížiace sa 0 sa zväčší výsledná presnosť definície, avšak v mnohých prípadoch môže výsledný okruh so správnymi hodnotami obsahovať takzvané *anomálie*, a preto je vhodné túto hodnotu udávať v rozsahu 40-60. Pri číslach blížiacich sa 100 je znova možné že výsledný okruh bude *false positive*, a nebude možné takýto okruh dát brať ako relevantný.

- **format** predstavuje súbor formátov na základe ktorých aplikácia definuje dátové typy výsledných prvkov. Tento súbor formátov je definovaný nasledovne:

  názov_dátového_typu : reprezentácia_regulárneho_výrazu

  Formáty sú reprezentované ako pravidlá určené regulárnymi výrazmi. Jeden „pseudo“ dátový typ je preto definovaný jedným jednoznačným regulárnym výrazom. Tieto výrazy sú užívateľovi prístupné a môže ich definíciu kedykoľvek meniť, a pridávať vlastné dátové typy definované vlastnými výrazmi. Ich formát je po takomto definovaní v konfiguračnom súbore dostupný v dátovom súbore na následnú špecifikáciu požadovaných vstupných údajov. Z určenia tohto formátu je možné jednoznačne povedať, že správna definícia týchto formátov bude viesť k úspešnejšej extrakcii.

- **primary** je hodnota regulárneho výrazu, ktorá sa viaže s hodnotou *maxFailRatio*, a to takým spôsobom, že práve táto hodnota je používaná na prvotnú detekciu výsledných dátových okruhov. Tieto okruhy sú definované pomocou atribútu *class*. Jedná sa o hodnotu definovanú užívateľom ako primárnu v danom vyhľadávanom prípade. Môže sa jednať napríklad o reprezentáciu skóre pri vyhľadávaní športových výsledkov alebo formát času typický pri daný súbor dát.

5.2 Vstupné dáta

Aplikácia ďalej rozoznáva súbor so vstupnými dátami definovanými užívateľom. Tieto dáta sú reprezentované znova vo formáte JSON, a dodržanie štruktúry je preto kritické.

Súbor so vstupnými dátami slúži užívateľovi na presnú definíciu danej extrakčnej úlohy. Samostatný súbor je preto vhodné vytvoriť pre každý dataset(skupinu domén z ktorej budú následne dáta extrahované) zvlášť. Prípadne je možné meniť vlastnosti a obsah daného dátového súboru tak, aby bol v daný moment jeho obsah relevantný.

Keďže sa jedná o jeden z primárnych súborov aplikácie, je jeho výskyt podmienený. Aplikácia pomocou neho dokáže určiť o aké dáta má v danej chvíli používateľ záujem, a tieto dáta budú v rovnakej forme reprezentované vo výstupných súboroch.

Formát výstupného súboru je preto so súborom obsahujúcim vstupné dáta úzko viazaný, kedže práve vstupné dáta určujú jeho následnú reprezentáciu. Výstupný súbor dát je založený na formáte JSON, a jeho štruktúra odpovedá štruktúre definovanej na vstupe.

Vstupné dáta sú preto formované nasledovne:

- **urls** predstavuje zoznam URL adries webových dokumentov. Tento zoznam je zložený z adries konkrétnych webových dokumentov, o ktoré má v danom momente užívateľ

záujem. Jeho poradie je kľúčové, a pri spustení aplikácie striktne dodržané. Nemalo by však v žiadnom prípade ovplyvniť žiadnu z vlastností aplikácie, pretože aplikácia sa na každý webový dokument pozerá samostatne. V prípade že sa jedná o webovú stránku stiahnutú do offline podoby, je potrebné určiť absolútnu cestu k danému dokumentu. V takom prípade je zároveň odporúčané aplikáciu spúšťať s parametrom *--offline*. Aplikácia zároveň kontroluje dostupnosť daného webového dokumentu a ak daný webový dokument nieje dostupný, je o tom užívateľ riadne informovaný. Zároveň však nieje ukončené vykonávanie programu a aplikácia pokračuje nasledovnými URL adresami zo zoznamu. Toto rozhodnutie bolo učinené hlavne z dynamickej povahy internetu, kde URL adresy sa v mnohých prípadoch môžu často meniť, a zastavenie vykonávania programu by bolo príliš obmedzujúce. Z toho dôvodu je o danej situácií užívateľ iba informovaný. Počet adries nieje limitovaný, avšak z dôvodu častej diverzifikácií webových dokumentov je pre používateľa výhodnejšie zoznam rozdeliť do viacerých súborov, čo následne môže značne ovplyvniť kvalitu výsledkov.

- **structure** slúži na definíciu štruktúry extrakčnej úlohy a jej objektov záujmu. Štruktúru definuje používateľ tak, aby odpovedala jeho požadovanému výstupu. Jej formát je definovaný nasledovne:

  výsledný_názov : názov_dátového_typu

  kde výsledný názov je názov, ktorý užívateľ požaduje vo výstupnom súbore, a názov dátového typu je dátový typ daného prvku. Dátový typ musí byť zhodný s jedným z dátových typov v konfiguračnom súbore, v opačnom prípade vyhodnotenie takéhoto prvku nebude umožnené. Počet prvkov v štruktúre nieje obmedzený, avšak jeho štruktúra je obmedzená iba na jednoduché výrazy. Zároveň treba brať na vedomie fakt, že sa jedná o štruktúru objektu ktorý nás v danej extrakčnej úlohe zaujíma. To znamená že nemá zmysel do štruktúry zahrnúť iné prvky ako sú pri danom objekte očakávané. Budú pravdepodobne zahodené alebo vôbec nebudú detegované.

  Vo výslednom súbore sa môžu objaviť len prvky ktoré tu boli zadefinované. Avšak v niektorých prípadoch výsledný súbor môže obsahovať prvok „undefined“. Takýto prvok značí, že sa pri extrakcii dát z daného objektu objavil prvok, ktorý aplikácia nedokázala podľa definovaných pravidiel zaradiť. Tento jav sa avšak dá vypnúť spustením aplikácie s parametrom *--noundef*. Pri takomto spustení je každý výsledný objekt kontrolovaný na prítomnosť nedefinovaných prvkov, a takéto sú následne z objektu vyradené. Na druhej strane prítomnosť všetkých definovaných prvkov nieje podmienená. Ak niektorý z prvkov pri danom objekte chýba, je jednoducho vynechaný, a jeho reprezentácia nieje vo výslednom súbore uvedená ani vo formáte prázdneho reťazca znakov. Takáto situácia môže nastať napríklad pri extrakcii dát z obchodných reťazcov a použitia dátového typu reprezentujúceho „zľavu“. Zľava nemusí byť pri každom produkte určená, a preto aplikácia pri produktoch bez zľavy daný prvok vynechá.

- **blacklist** ako voliteľný argument predstavuje zoznam reťazcov znakov, ktoré užívateľ rozpoznal ako neočakávané alebo nechcené. V takom prípade užívateľ daný reťazec zaradí do zoznamu blacklist, ktorý aplikácií slúži ako zoznam nežiaducich reťazcov. Pri spúšťaní aplikácie je zároveň nutné aplikáciu spustiť s parametrom *-b*. V opačnom prípade bude blacklist ignorovaný. Pri použití blacklistu nezáleží či sa daný reťazec znakov nachádza v inom reťazci, existuje ako samostatný reťazec alebo predstavuje jediný obsah daného elementu. V každom prípade je takýto reťazec vyradený, a aplikácia ho určí ako prvok určený na vyradenie z výsledného objektu. Využitie blacklistu

je na rozhodnutí užívateľa, a preto nieje jeho existencia v súbore so vstupnými dátami podmienená. V prípade ale že ho užívateľ bude chcieť využiť, je treba zachovať jeho jednoduchú štruktúru v podobe pola reťazcov. Pole nieje obmedzené, a môže obsahovať nekonečne veľa výrazov.

5.3 Štruktúra aplikácie

Výsledná aplikácia je štrukturovaná do niekoľkých logických častí. Po prvotnom spustení aplikácie prebehne kontrola vstupných parametrov, konfigurácie a vstupných dát. Tým sa zadefinujú vlastnosti programu pri danej extrakčnej úlohe.

Asynchrónny proces následne očakáva vykonanie hlavného procesu. Ten je zložený z funkcií ktoré definujú správanie programu.

Ako prvé aplikácia zabezpečí vytvorenie inštancie API Puppeteer. To znamená že sa vytvorí inštancia prehliadača Chromium s parametrom *headless* ktorý prehliadač spustí v špeciálnom móde, kde je užívateľské prostredie skryté. Táto forma bola vybraná hlavne kvôli urýchleniu procesu a zachovania jednoty pri vykonávaní danej extrakčnej úlohy.

Po vytvorení spojenia je vytvorená inštancia pre webovú stránku, ktorá bude v danom bode úlohy analyzovaná. Stránka je následne načítaná a prebehne zber dát v podobe všetkých dostupných triednych atribútov jednotlivých prvkov. Tieto triedne atribúty sú následne zoradené podľa početnosti. Algoritmus následne rozdelí obsah webovej stránky do okruhov, ktoré sú definované práve spomínanými triednymi atribútami. Tieto triedne atribúty sú viazané na štylizáciu webovej stránky, a preto sa dá určiť že ich výskyt bude rovnaký práve pre jednu skupinu dát. Zároveň sa dá predpokladať, že ak sa jedná o dáta ktoré sa na webovej stránke opakujú, tak bude výskyt týchto triednych atribútov väčší ako výskyt iných atribútov. Preto je zoradenie zároveň kritické pre rýchlosť analýzy dát. To ako analýza prebieha bude ďalej popísané v tejto kapitole.

Po analýze prebehne extrakcia dát. Tá prebieha v troch krokoch:

1. Evaluácia
2. Analýza
3. Príprava výsledkov

Po vykonaní prípravy výsledkov sú výsledky zapísané vo formátovanej podobe do výsledného súboru. Ten následne nesie názov domény alebo súboru, z ktorého boli dané dáta extrahované.

Literatúra

[1] *Headless Chromium* [https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md]. (Accessed on 04/03/2021).

[2] *JSON* [https://www.json.org/json-en.html]. (Accessed on 04/01/2021).

[3] *Modulecounts* [http://www.modulecounts.com/]. (Accessed on 03/31/2021).

[4] CHROMEDEVTOOLS. *Chrome DevTools Protocol* [https://chromedevtools.github.io/devtools-protocol/]. (Accessed on 04/03/2021).

[5] ELLIOTT, E. *How Popular is JavaScript in 2019? | by Eric Elliott | JavaScript Scene | Medium* [https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1]. May 2019. (Accessed on 03/31/2021).

[6] JAVATPOINT. *Web Scraping Using Python - Javatpoint* [https://www.javatpoint.com/web-scraping-using-python]. (Accessed on 03/31/2021).

[7] MOZILLA. *Introduction to the DOM - Web APIs | MDN* [https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction]. (Accessed on 04/01/2021).

[8] MOZILLA. *What is JavaScript? - Learn web development | MDN* [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript]. (Accessed on 03/31/2021).

[9] NODEJS. *About | Node.js* [https://nodejs.org/en/about/]. (Accessed on 04/02/2021).

[10] NODEJS. *What is npm? | Node.js* [https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/]. August 2011. (Accessed on 04/03/2021).

[11] OCTOPARSE. *What is Web Scraping and How Does It Work | Octoparse* [https://www.octoparse.com/blog/web-scraping-introduction#]. October 2018. (Accessed on 03/30/2021).

[12] PROWEBSCRAPER. *The 5 Best Programming Languages for Web Scraping – ProWebScraper* [https://prowebscraper.com/blog/best-programming-language-for-web-scraping/]. (Accessed on 03/31/2021).

[13] Puppeteer. *Puppeteer v8.0.0* [https://pptr.dev/]. (Accessed on 04/03/2021).

[14] tarun007. *Difference between cheerio and puppeteer - GeeksforGeeks* [https://www.geeksforgeeks.org/difference-between-cheerio-and-puppeteer/]. July 2020. (Accessed on 03/31/2021).

[15] w3schools. *CSS Introduction* [https://www.w3schools.com/css/css_intro.asp]. (Accessed on 04/01/2021).

[16] w3schools. *What is HTML* [https://www.w3schools.com/whatis/whatis_html.asp]. (Accessed on 04/01/2021).

[17] Wolf, K. *Jan Čurn (Apify): Z webu stahujeme už miliardu stránek měsíčně - Lupa.cz* [https://www.lupa.cz/clanky/jan-curn-apify-z-webu-stahujeme-uz-miliardu-stranek-mesicne/]. January 2020. (Accessed on 03/30/2021).