



**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**

AUTHOR

**LUKÁŠ PERINA**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. RADEK BURGET, Ph.D.**

**BRNO 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 81,97% a recall 76,77%. Týmto prístupom sa umožní zredukovať čas, ktorý je potrebný na analýzu jednotlivých stránok na minimum, a nebrať štruktúru kódu ako určujúci faktor pri vytváraní požiadaviek 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 accuracy of 81,97% and recall 76,77%. 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.

## Klíčové slová

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

## Keywords

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

## Citácia

PERINA, Lukáš. *Metódy extrakcie dát z webových stránok*. Brno, 2021. Bakalárska 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  
20. apríla 2021

## Podakovanie

Moje podakovanie 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

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Aktuálny prístup k problematike Web Scrapingu</b>	<b>5</b>
2.1	Web scraping . . . . .	5
2.2	Súčasný postup . . . . .	5
2.3	Výhody, nevýhody a využitie web scrapingu . . . . .	7
2.4	Metódy využívané v súčasnosti . . . . .	8
2.4.1	Hotové riešenia . . . . .	8
2.4.2	Platformy a knižnice . . . . .	9
2.5	Navrhované riešenie . . . . .	12
<b>3</b>	<b>Technológie</b>	<b>13</b>
3.1	HTML . . . . .	13
3.2	DOM . . . . .	13
3.3	CSS . . . . .	14
3.4	Javascript . . . . .	14
3.4.1	JSON . . . . .	16
3.4.2	Node.js . . . . .	16
3.4.3	NPM . . . . .	16
3.4.4	Puppeteer . . . . .	17
<b>4</b>	<b>Návrh architektúry aplikácie</b>	<b>20</b>
4.1	Špecifikácia požiadaviek . . . . .	20
4.2	Vstupné pravidlá . . . . .	21
4.2.1	Hlavné pravidlá . . . . .	21
4.2.2	Vedľajšie pravidlá . . . . .	22
4.3	Výstupný formát . . . . .	23
4.4	Architektúra aplikácie . . . . .	24
4.4.1	Jadro programu . . . . .	24
4.4.2	Hlavný proces . . . . .	25
<b>5</b>	<b>Implementácia riešenia</b>	<b>27</b>
5.1	Konfiguračný súbor . . . . .	27
5.2	Vstupné dáta . . . . .	28
5.3	Štruktúra aplikácie . . . . .	30
5.4	Analýza webovej stránky . . . . .	31
5.5	Generovanie výsledkov . . . . .	33
5.5.1	Príprava výsledkov . . . . .	35

<b>6</b>	<b>Testovanie a vyhodnotenie výsledkov</b>	<b>36</b>
6.1	Testovanie počas vývoja . . . . .	36
6.2	Testovacie prostredie . . . . .	36
6.3	Spôsob testovania . . . . .	37
6.4	Testovacia sada . . . . .	38
6.4.1	Štruktúra testovacej sady . . . . .	38
6.5	Testovací skript . . . . .	39
6.6	Vyhodnotenie výsledkov . . . . .	40
	<b>Literatúra</b>	<b>42</b>
	<b>A Obsah priloženého CD</b>	<b>44</b>
	<b>B Manuál</b>	<b>45</b>
B.1	Návod na inštaláciu . . . . .	45
B.2	Štruktúra konfiguračných súborov . . . . .	46
B.3	Dostupné parametre . . . . .	47
B.4	Návratové kódy . . . . .	48

# 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 webových stránok 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 používateľov. 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 web stránke 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 používateľsky 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 súčasnosti 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 navrhovanej 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 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 protokolu *Hypertext Transfer Protocol*, 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 považovaný World Wide Web Wanderer, ktorý bol vyvinutý v roku 1993 za účelom zmerania veľkosti celého internetu a neskôr na jeho priamu indexáciu [13].

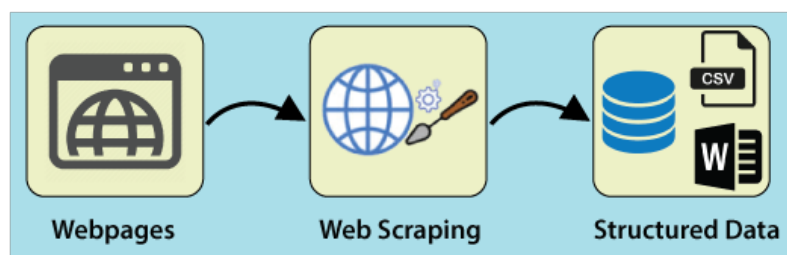
### 2.2 Súčasné postupy

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 používateľská prívetivosť a jednoduchosť definovania vstupných po-

žiadaviek. Postup, ktorý aplikácie na extrakciu dát využívajú je prevažne rovnaký, a skladá sa z troch primárnych krokov: **2.1**

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

Tieto kroky sú potom pri každej aplikácii mierne prispôsobené určeniu a typu implementácie [13].



Obr. 2.1: Všeobecný postup ktorým sa aplikácia typu web scraper riadi. Prevzaté z Javat-point [4]

Medzi najpopulárnejšie web scrapery patria aplikácie s jednoduchou point-and-click politikou. Po inštalácii takejto aplikácie používateľ jednoducho zvolí webové dokumenty, z ktorých chce extrakciu prevádzať, a následne interaktívne vyznačí dáta, ktoré ho zaujímajú. Pre používateľov, ktorí nemajú príliš veľké skúsenosti s informačnými technológiami a nechcú platiť za túto službu nemalé peniaze rôznym korporáciám je to skrátka jediná možnosť [15].

Spomínaný prístup má samozrejme svoje výhody, medzi ktoré sa radí najmä spomínaná používateľská 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. Počet extrahovaných webových stránok sa pohybuje v miliardách mesačne na jednu takúto firmu [19].

Pri takto veľkých číslach narážajú spomínané firmy na rôzne problémy, ktoré rádový používateľ web scrapingovej aplikácie riešiť nemusí. Medzi takéto problémy sa radia v nasledujúcom 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 nieje považované za optimálne riešenie. Stránky v dnešnej dobe môžu využívať rôzne ochrany proti web scrapingu, ako napríklad priamu detekciu človeka od robota alebo tzv. CAPTCHA<sup>1</sup> [19] .

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 neustálym rozširovaním sa internetu ako celku. V počiatkoch internetu sa aplikácie typu web scraper nazývali skôr *crawler* a využívali sa 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. Avšak 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. Ich účelom je prevažne cielenie zobrazovaných reklám tak, aby boli pre používateľa ako jednotlivca relevantné [13].

Jednotlivé prípady využitia takejto aplikácie sa neustále rozširujú, obecné 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

---

<sup>1</sup>Completely Automated Public Turing test to tell Computers and Humans Apart.

Každá technológia a jej využitie má však aj svoje nevýhody. Manuálny zber dát môže byť taktiež rozumná možnosť, a to hlavne v prípade, že sa jedná o ojedinelú prípadne unikátnu udalosť. V takomto prípade sa tvorba alebo priame využitie web scraperu neodporúča. A to hlavne pre časovú zložitosť nielen naprogramovania, ale aj samotnej konfigurácie a zoznámenia sa s aplikáciou. Využitie samostatnej aplikácie na takúto činnosť je rozumné až vtedy, keď sa daná akcia musí opakovať v intervaloch alebo automatizovane [4].

V takom prípade je potreba zvážiť aj rôzne prvky pri nielen prevádzke, ale aj samotnom vytváraní a následnom používaní web scraperu. 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 nemôžeme zabúdať na údržbu web scraperu 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 scraperu nedokáže pokračovať v extrahovaní údajov tak ako predtým, je potrebné v mnohých prípadoch stránku znova analyzovať.

Ak web scraper nieje schopný efektívne využiť algoritmy, ktoré v minulosti fungovali, v niektorých prípadoch je dokonca treba dodatočne zanalyzovať prístupy web scraperu 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, máme v dnešnej dobe na výber z mnohých aplikácií. Tie sú pripravené na použitie, a tak nieje treba programovať aplikácie od začiatku. Každá má však svoje vlastné postupy pri extrakcii a z toho plyní fakt, že nie každá takáto aplikácia dokáže splniť požadované zadanie.

Medzi najznámejšie aplikácie v tomto odvetví však patria hlavne<sup>[4]</sup>:

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

V niektorých prípadoch nám stačí jednoduché rozšírenie do prehliadača (napr. Web Scraper<sup>2</sup>), ktoré je samozrejme efektívne hlavne pri menšom počte webových stránok a dát určených na scraping.

Spomenuté aplikácie sú vo svojom odvetví veľmi populárne, avšak každá z nich má svoje limity. Pre skúsenejších používateľov, ktorí 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 voľne dostupných programovacích jazykov. Medzi dva najpopulárnejšie však patria hlavne Python a Javascript. Každý má však znova svoje výhody a nevýhody, a preto je treba zvážiť na aké účely bude web scraper používaný. Vo všeobecnosti je využívanější 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<sup>3</sup> a Scrapy<sup>4</sup> [2]. Tieto som samozrejme zohľadnil pri výbere vhodného frameworku na vytvorenie web scraperu.

Na druhej strane je tu však Javascript, ktorý je vyvíjaný priamo pre prácu s webovými stránkami. Kým Python je lepší čo sa týka podpory a rozšírenosti frameworkov, Javascript má výhodu v lepšej integrácii nielen s webovými stránkami, ale aj s webovými prehliadačmi samotnými. Python v mnohých prípadoch vyžaduje špeciálny ovládač na komunikáciu s prehliadačom a podpora dynamických stránok je znova jednoznačne lepšia pomocou Javascriptu. Je to tak práve kvôli jeho spomínanej lepšej integrácii 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, ktorý vyvoláva dojem, že webová stránka s užívateľom interaguje. Vo väčšine prípadov beží iba na strane klienta, takže dynamickosť samotná nezatažuje server. Zároveň je v mnohých prípadoch využívaná asynchrónnosť<sup>5</sup> tohto programovacieho jazyka, na zlepšenie odozvy a interakcie [9]. 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.

---

<sup>2</sup><https://webscraper.io/>

<sup>3</sup><https://www.selenium.dev/>

<sup>4</sup><https://scrapy.org/>

<sup>5</sup>Asynchrónnosť znamená v tomto prípade neblokovanie prostriedkov prehliadača a načítavanie obsahu webovej stránky zároveň so skriptami.

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

Rozhodnutie používať Javascript som učinil zároveň s predpokladom využitia jedného z jeho voľne 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 považovaná za 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 podporu asynchrónnosti, ktorá je pri takejto aplikácii kľúčová.

Ako prvý framework spomeniem určite Apify SDK, za ktorým stojí Česká firma Apify Technologies<sup>6</sup> [19]. 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 nielen webovou technológiou. V čase písania má Node.js viac ako 1.5 milióna dostupných rozširujúcich balíčkov a jeho náskok je v súčasnej dobe jednoznačný 2.2[3].

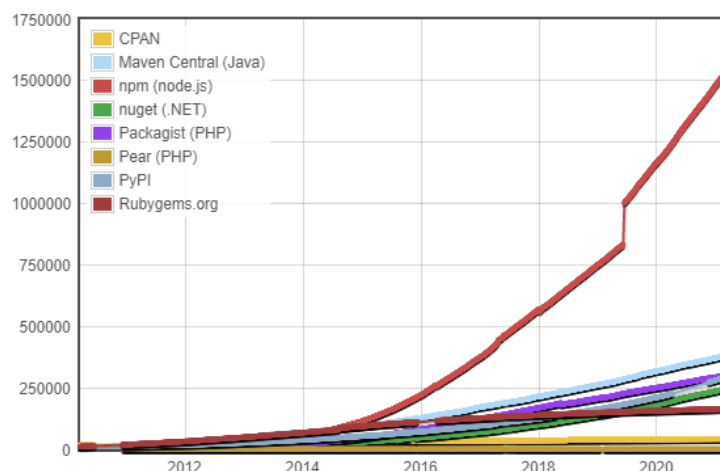
Pri porovnávaní technológií Puppeteer a Cheerio je na prvý pohľad jednoznačné, čo majú spoločné, a v čom naopak každá z nich vyniká. Tieto poznatky pomôžu jednoznačne určiť technológiu na základe požiadaviek, ktoré vznikli zároveň s výberom témy tejto práce.

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.

---

<sup>6</sup><https://apify.com/>

## Module Counts



Obr. 2.2: Porovnanie najpopulárnejších technológií a vývoj počtu ich modulov. Zdroj Moduleco <http://www.modulecounts.com/>

Puppeteer je technológia, ktorá je vyvíjaná za účelom automatizácie webového prehliadača, zatiaľčo hlavným určením Cheeria je web scraping samotný. Tým, že je Cheerio určené priamo na požadovanú úlohu je v tomto ohľade rýchlejšie a výhodnejšie pre web scraping. 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 priamu 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 Scraperu. 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 funkcií Javascriptu [16].

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 požiadaviek neviazali na jednotlivé webové dokumenty. 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 dodržanie 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 používateľské pravidlá za účelom jednoznačne určiť ich prioritu a typ využitia. Vstupné pravidlá nemôžu byť závislé na štruktúre webového dokumentu, ale na jeho obsahu.

Finálna aplikácia bude vyvíjaná pomocou technológie Puppeteer najmä vďaka jeho jednoznačnej integrácii 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. Údaje z webovej stránky sa 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 bez akýchkoľvek problémov. Protokol Chrome DevTools dokáže zabezpečiť komunikáciu na vysokej úrovni, ktorá dané dáta preniesie 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 prenositeľnosti 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 používateľského vstupu, jeho 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 `<>`. Tie nám 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. Ten HTML dokumenty zobrazí zároveň s doplnkovými dokumentami, ktoré určujú štylizáciu a správanie sa danej webovej stránky. HTML dokumenty samotné neobsahujú žiadnu štylizáciu, avšak typ použitého elementu je kritický. To hlavne z dôvodu, že udáva základné rozloženie obsahu, ktorý element predstavuje [7].

### 3.2 DOM

Jazyk HTML nieje určený na programovacie účely, a preto je potrebné pri práci s webovými stránkami zväziť aj technológiu DOM. DOM je skratka reprezentujúca *Document Object Model*. Predstavuje akési prepojenie medzi HTML a programovacím jazykom, ktorý chce s webovou stránkou komunikovať [8].

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

- Uzly
- Elementy

Práve tento fakt nám ďalej dovoľuje s webovou stránkou interagovať a meniť jej obsah. Príkladom takéhoto využitia je jazyk Javascript, ktorý bol vyvinutý za účelom vytvorenia dojmu interakcie používateľa 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 reprezentácii 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 hlavných pravidiel tohto jazyka - štylizáčne 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. Popisuje teda stav elementov pre médium na ktorom je webový dokument zobrazovaný [6].

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 pravidlá aplikujú aj na celé elementy, prípadne identifikátory.

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

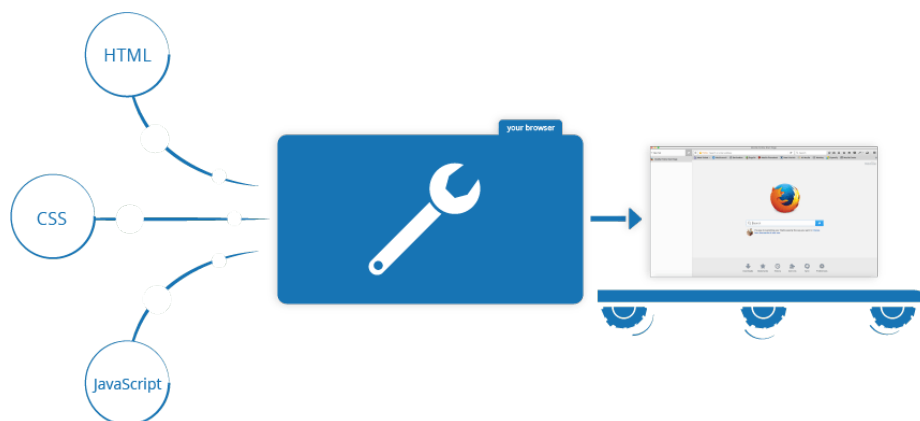
- Riadkové
- Interné
- Externé

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 [3]. 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 docieľiť výsledky, aké sa pri moderných štandardoch vyžadujú 3.1. Či už sa jedná o dynamickú zmenu obsahu alebo animovanie niektorých elementov, je to práve Javascript, ktorý túto interakciu docieľuje [9].



Obr. 3.1: HTML, CSS a JavaScript sú 3 hlavné zložky ktoré spolu vytvárajú web ako ho poznáme dnes. Zdroj [9]

Javascript je založený na programovacom jazyku Java, ktorý bol ďalej modifikovaný a prispôbený na prácu priamo vo webovom prehliadači. Javascript dokáže nielen reagovať na vstupy od používateľa, ale reagovať aj na niektoré udalosti na pozadí, v prehliadači [9].

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

Jednou z najpopulárnejších vlastností tohto programovacieho jazyka je podpora API<sup>1</sup>. Podpora API umožňuje využitie naprogramovaných rozhraní bez toho, aby si tieto rozhrania používateľ 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 API prehliadača predstavuje napríklad aj spomínaný DOM. Ten 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 súborý Javascriptu, môže nastať problém. Vykonávanie skriptu v takom prípade zlyhá. Asynchrónnosť zároveň súvisí s postupnosťou vykonávania kódu. Ak 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 [9].

<sup>1</sup>Application Programming Interfaces

### 3.4.1 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 čitateľná počítačom. 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. Podporuje reprezentáciu objektov, čísiel, reťazcov a mnohých ďalších dátových typov. Tieto môže zároveň ukladať do polí a vytvárať tak skupiny. Všetky spomínané štruktúry sú podporované modernými prehliadačmi a programovacími jazykmi, ktoré sa v tomto zmysle používajú. JSON sa najčastejšie používa pri Javascriptových a webových aplikáciách [10].

### 3.4.2 Node.js

Node.js ako behové prostredie Javascriptu bolo vytvorené hlavne za účelom obsluhovania 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 predstavoval možnosť spúšťania Javascript 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. Tie 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. Tieto technológie predstavujú frameworky založené a fungujúce práve vďaka Node.js [12].

Node.js ponúka zároveň možnosť využitia rôznych modulov. Moduly boli vyvinuté za účelom rozšírenia vlastností a funkcionality. Aj vďaka týmto modulom a faktu, že s ich 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 používateľov distribuované pomocou NPM.

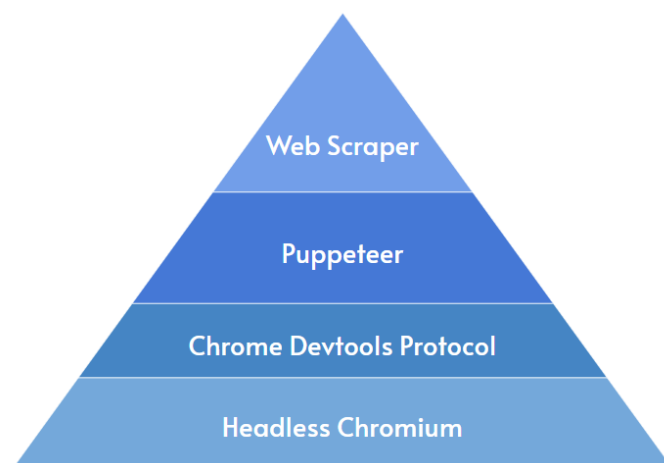
### 3.4.3 NPM

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[11]. A práve tu sa nachádza aj knižnica Puppeteer.

### 3.4.4 Puppeteer

Hlavnou súčasťou výslednej aplikácie je Puppeteer. Je to jedna z mnohých knižníc vytvorených pre Node.js distribuovaná pomocou NPM. V aplikácii zohráva dôležitú úlohu, keďže práve technológia Puppeteer je spôsob akým aplikácia komunikuje a extrahuje údaje z prehliadača [3.2](#).

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 [\[14\]](#).



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 prehliadača Chromium v takzvanom headless prostredí, kedy sa inštancia programu Google Chrome alebo Chromium spustí v režime server, bez používateľského prostredia. Ovládanie prehliadača je umožnené za pomoci terminálu. Používa sa na rôzne testovanie a automatizáciu. Režime headless je pri používaní Puppeteeru nastavený ako základný, avšak je možné ho vypnúť a používať Chromium v headfull móde, teda so zobrazeným používateľským prostredím [\[1\]](#).
2. **Chrome DevTools protokol** bol založený za účelom automatizácie prehliadača Google Chrome, Chromium a ostatných webových prehliadačov 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 [\[17\]](#).
3. **Puppeteer** ďalej nadväzuje na protokol DevTools v podobe API, ktoré týmto protokolom dokáže komunikovať s inštanciou prehliadača Chrome.
4. **Web scraper** následne využíva knižnicu Puppeteer. Tá prichádza s radom revolučných riešení umožňujúcich automatizovať takmer čokoľvek.

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

Niektoré z ďalších prípadných využití tejto knižnice zahŕňajú aj:[14]

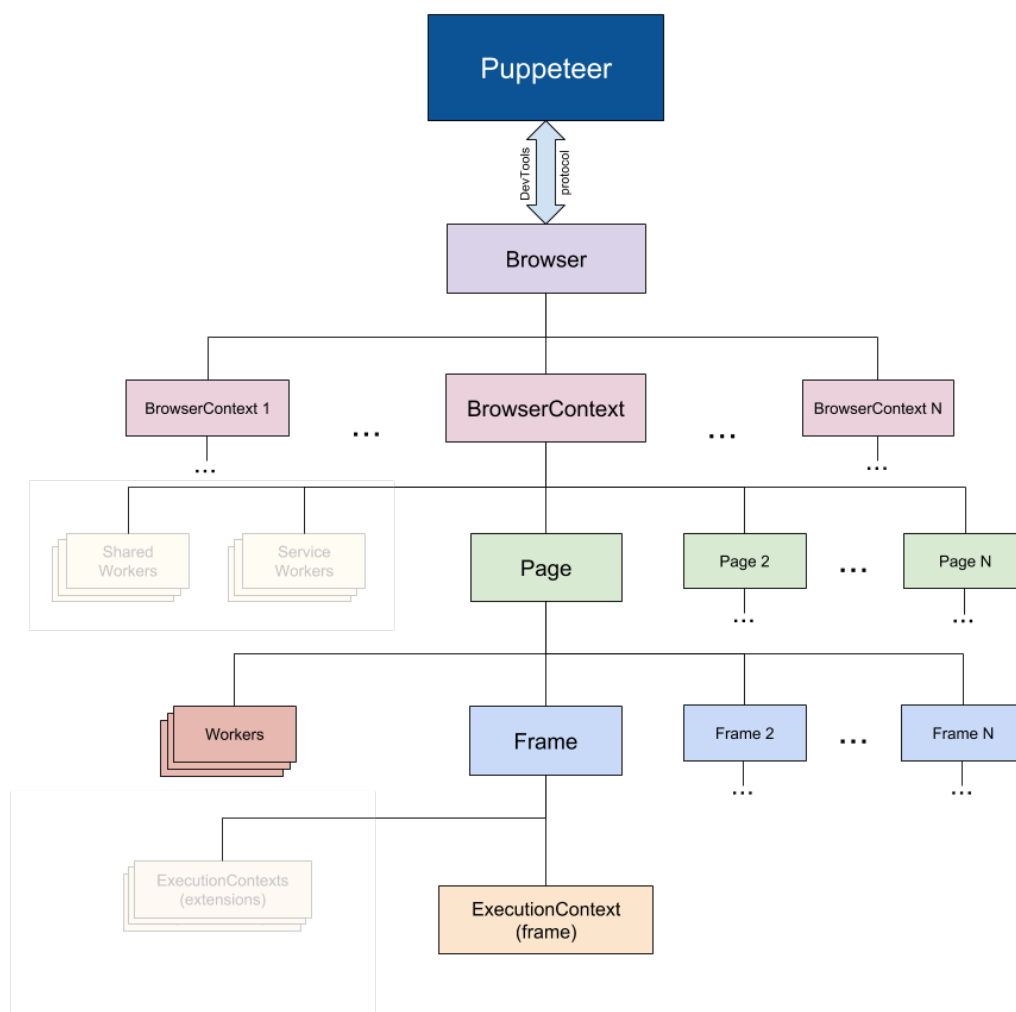
- Vytváranie snímky obrazovky z webových stránok
- Automatizácia formulárov
- vytvorenie testovacieho prostredia pre akúkoľvek aplikáciu

Online verzia Puppeteeru je zároveň dostupná v sandbox prostredí<sup>2</sup> 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 vo webovom prehliadači za pomoci príkazu *await browser.newPage()*. Vzniknutá stránka následne slúži ako prostredie pre načítanie a následnú extrakciu dát z jednotlivých webových stránok. Knižnica Puppeteer obsahuje širokú škálu funkcií. Tie dovoľujú komunikovať s prehliadačom obojsmerne, a teda prijímať aj odosielať dáta prehliadaču. Bezproblémová komunikácia medzi aplikáciou a prehliadačom je základom pre úspešnú interakciu a extrakciu dát. Nasledujúci diagram 3.3 predstavuje vzťah jednotlivých prvkov knižnice Puppeteer a spôsob akým komunikuje.

---

<sup>2</sup><https://try-puppeteer.appspot.com/>



Obr. 3.3: Diagram knihovny Puppeteer zobrazující vztah jednotlivých prvků API.  
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 definovania požiadaviek na vstupné dáta, výstupné dáta a vnútorné chovanie aplikácie. V úvahu je samozrejme braná aj aktuálna situácia a stav dostupných riešení. S tým je spojená aj analýza, ktorá určovala smer vývoja pri dôležitých otázkach ohľadne vnútornej štruktúry aplikácia.

### 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 nie sú ekvivalentné. A to nielen z pohľadu presnosti, ale aj z pohľadu rýchlosti extrakcie, spôsobu extrakcie a ďalších aspektoch viazaných na logiku aplikácie aká je vo finálnej fáze zvolená.

Základom extrakcie je zároveň vedieť o aké dáta máme záujem. Tieto dáta teda musia byť jednou z hlavných častí definovaných 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 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.

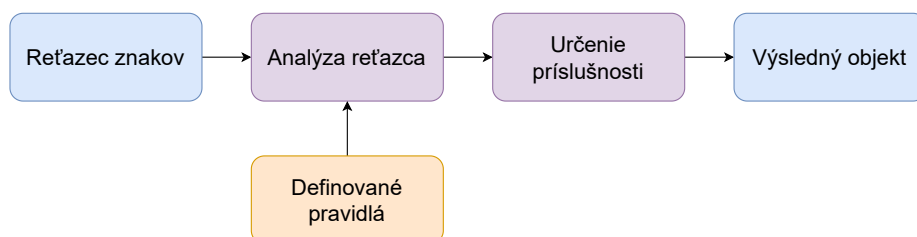
Pri návrhu je treba samozrejme zohľadniť fakt, že finálny používateľ bude chcieť špecifikáciou týchto pravidiel stráviť čo najmenej času. To je samozrejme a je preto dôležité odlíšiť časti pravidiel, ktoré nieje nutné prepisovať tak často od pravidiel, ktoré sa môžu meniť s príchodom nových požiadaviek.



## 4.2 Vstupné pravidlá

Pojmom vstupné pravidlá rozumieme pravidlá definované ako základné vstupné údaje pri spustení aplikácie. Tieto údaje budú mať za úlohu definovať presne to, čo v danom konkrétnom prípade používateľ aplikácie očakáva ako aplikačný výstup. Sú to zároveň pravidlá pre aplikáciu určujúce objekty vo webovom dokumente ako bod záujmu extrakcie 4.1.

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



Obr. 4.1: Spôsob určenia finálneho dátového typu a príslušnosti objektu

### 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 jednoduchosť definovania týchto pravidiel používateľom, boli hlavné a vedľajšie pravidla rozdelené do dvoch konfiguračných súborov, pre ich jednoduchšiu správu.

Medzi hlavné pravidlá na extrakciu boli vybrané:

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

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

**Štruktúra hľadaných prvkov** je 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. Tie 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 používateľ 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 vyradený a test prebieha nanovo s novým testovacím okruhom. Testovací okruh je reprezentovaný dátami, ktoré sú porovnávané s definíciou primárneho prvku. Tá určí či aktuálny kontrolovaný prvok spĺňa definovaný regulárny výraz. V prípade, že ho spĺňa je prvok nezmenený. V prípade, že ho nespĺňa je prvok nahradený výrazom *Null* ktorý v následnom výpočte posluží ako označenie prvkov ktorých kontrola „zlyhala“.

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

**Formát** predstavuje objekt zložený 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 prvok, ktorý sa v objektoch určených na extrakciu opakuje, a teda je 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 webovej stránky 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 znova pokiaľ možno v univerzálnom formáte. Keďže sa dá predpokladať, že extrahované dáta budú v prípade potreby použité, je vhodné zvoliť formát podporovaný rôznymi aplikáciami a programovacími jazykmi.

Z toho dôvodu tu bol znova 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 používateľovi, 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 používateľskom vstupnom súbore. Táto štruktúra je potom zachovaná pre všetky objekty. Ak však jeden z kľúčov 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 kľúč značí prvok, ktorého dátový typ nebolo možné na základe špecifikovaných vstupných požiadaviek definovať, avšak patrí do vyhľadávanej podskupiny.

Formát JSON ako výstupný formát je striktné 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 pri 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

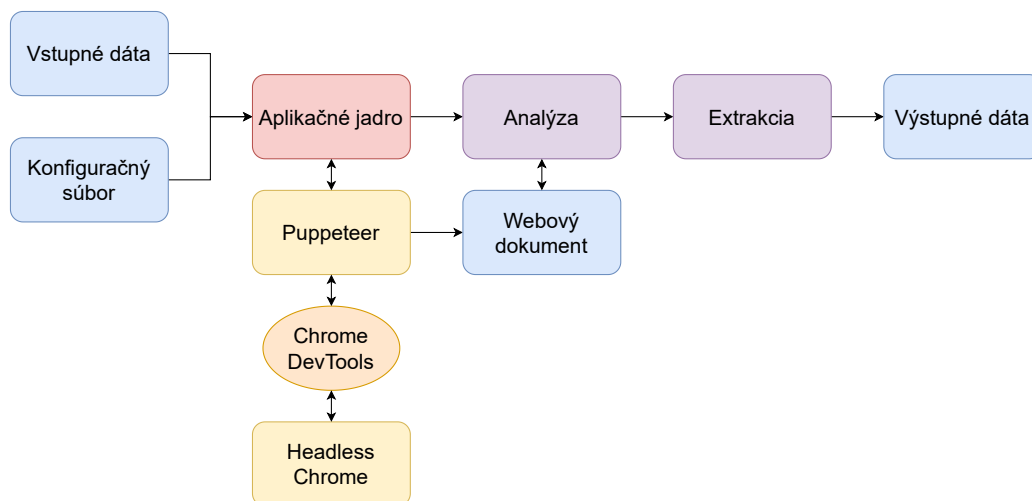
Aplikácia zároveň pri každom spustení a jej behu informuje používateľa o aktuálne prevádzanej akcii. V režime bez vypisovania explicitných informácií aplikácia informuje o začatí exekúcie formou vypísania *Execution started*. Po skontrolovaní vstupných údajov informuje o začatí extrakcie vypísaním *BEGIN 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 objavajú časy jednotlivých akcií, informácie o počte výsledných objektov, zoznam testovaných okruhov a identifikácia výsledného okruhu.

## 4.4 Architektúra aplikácie

V tejto časti práce bude popísaná architektúra aplikácie tak, ako bola aplikácia navrhnutá a odôvodnenie týchto krokov. Nasledujúci diagram reprezentuje finálny návrh architektúry aplikácie a vzťah jednotlivých prvkov.



Obr. 4.2: Diagram navrhutej architektúry

### 4.4.1 Jadro programu

Hlavný zdrojový kód sa bude nachádzať v jednom súbore, ktorý pre jednoznačnosť v rámci tejto práce nesie názov *extractor.js*. Koncovka *js* napovedá, že sa bude jednať o súbor založený na programovacom jazyku Javascript. Pri spúšťaní aplikácie sa preto bude tento súbor volať pomocou príkazu *node* ktorý značí behové prostredie 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 nadviazaním tohto spojenia však dôjde ku kontrole vstupných dát zadanych používateľom. V prípade, že je formát vstupných dát poškodený, nieje dodržaná jeho štruktúra, 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 proces výberu 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 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 údaje daného okruhu ako potencionálnu súčasť výsledkov, nemusí to tak ale byť 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 používateľom.

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 používateľovi ani inému programu, ktorý by dáta využíval, nemôžu byť užitočné. Práve z toho dôvodu by mal používateľ týmto prípadom predchádzať, a to dvoma spôsobmi.

Keďže aplikácia analyzuje výsledky ako celok, vie presne určiť aký dátový typ bol na danom indexe definovaný v predošlých prípadoch objektov, ktorých dáta testom na regulárny výraz prešli. Týmto sa dá jednoducho predísť strate dát pri anomáliách. V niektorých prípadoch však ani toto nieje riešenie. V tom prípade má aplikácia možnosť jednoducho takéto nedefinované výsledky vynechať a používateľovi 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, dĺžky trvania tejto akcie alebo časových údajov o týchto akciách.

## Kapitola 5

# Implementácia riešenia

V rámci tejto kapitoly 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 implementácii 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 vytváraniu dátových okruhov. Po skúmaní som pristúpil k vytváraniu 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 používateľa je uložená v konfiguračnom súbore, ktorý môže používateľ voľne upravovať a dokonca zvoliť svoj vlastný. Základný konfiguračný súbor je k výslednej implementácii pripojený a aplikácia ho očakáva implicitne ako súbor *config.json*.

Jeho úlohou je definícia základných pravidiel používaných pri extrakcii dát z webových stránok. 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 používateľa zároveň informuje o danej situácii chybovým hlásením zobrazeným v príslušnom termináli.

Štruktúra konfiguračného súboru je daná nasledovne:

- **maxFailRatio** ako hodnota v rozsahu 0-100. Táto hodnota definuje maximálny pomer prvkov, ktoré pri určovaní výsledného okruhu neboli úspešné. Tieto prvky sú vo výslednom okruhu dát reprezentované 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 blížiac 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 dáť 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 používateľovi 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 používateľom 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 používateľom. Tieto dáta sú reprezentované znova vo formáte JSON a dodržanie štruktúry je preto kritické.

Súbor so vstupnými dátami slúži používateľovi na presnú definíciu danej extrakčnej úlohy. Samostatný súbor je vhodné vytvoriť pre každý súbor domén(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, keďže 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 formované nasledovne:

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



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 používateľ riadne informovaný. Zároveň však nieje ukončené vykonávanie programu a aplikácia pokračuje nasledovnými URL adresami zo zoznamu. Toto rozhodnutie bolo učené 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 situácii používateľ 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 používateľ 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. Tomuto javu sa dá predchádzať spustením aplikácie s parametrom *--noundef*. Pri takomto spustení je každý výsledný objekt kontrovaný 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 „zlavu“. 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 používateľ rozpozna ako neočakávané alebo nechcené. V takom prípade používateľ daný reťazec zaradi do zoznamu blacklist, ktorý aplikáciou 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 používateľa, a preto nieje jeho existencia v súbore so vstupnými dátami podmienená. V prípade ale, že ho používateľ bude chcieť využiť, je treba zachovať jeho jednoduchú štruktúru v podobe poľa reťazcov. Veľkosť poľa nieje obmedzená a môže obsahovať nekonečne veľa výrazov.

## 5.3 Štruktúra aplikácie

Výsledná aplikácia je štruktúrovaná 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 používateľské prostredie skryté. Táto forma bola vybraná hlavne kvôli urýchleniu procesu a zachovaniu 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 atribútmi. Tieto triedne atribúty sú viazané na štylizáciu webovej stránky, a preto sa dá určiť, že ich výskyt bude rovnaký práve pre práve 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 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:

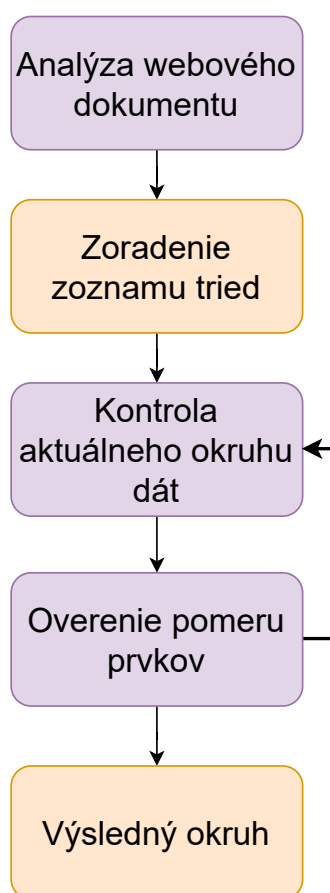
1. Evaluácia
2. Klasifikácia
3. Príprava výsledkov

Výsledky sú následne pripravené na zápis. Zapísané sú vo formátovanej podobe do výsledného súboru, ktorý nesie názov domény alebo dokumentu, z ktorého boli dané dáta extrahované. Všetky takéto súbory sú následne uložené v preddefinovanej zložke. Ak táto zložka nieje definovaná používateľom, aplikácia zvolí základnú zložku *output* v koreňovom adresári aplikácie. Súbor obsahujúci metadáta je taktiež uložený v koreňovom adresári, avšak mimo zložky s výslednými súbormi. Tým môžeme predísť nesprávnemu zaradeniu metadát a výsledných dát. Používateľ je zároveň o aktuálnej lokácii súboru s metadátami informovaný po skončení vykonávania danej extrakčnej úlohy.

## 5.4 Analýza webovej stránky

Pre získanie relevantných dát a obsahu z webového dokumentu je potrebné daný dokument analyzovať. Analýza prebieha bezprostredne po načítaní webového dokumentu a je rozdelená do niekoľkých krokov:

1. Analýza triednych atribútov
2. Zoradenie zoznamu tried na základe výskytu
3. Kontrola daného okruhu dát z pohľadu primárneho prvku
4. Overenie pomeru nevyhovujúcich prvkov



Obr. 5.1: Postup pri analýze webového dokumentu

## Analýza triednych atribútov

Ako prvý krok pri získavaní relevantných údajov o architektúre webovej stránky prebehne analýza triednych atribútov. Aplikácia analyzuje každý prvok reprezentovaný HTML značkou a uloží zoznam všetkých použitých tried, ktoré daný prvok obsahuje. Týmto spôsobom prejde aplikácia všetky značky a získa tak prehľad o všetkých triednych atribútoch na celej webovej stránke.

Keďže sa dá predpokladať, že pri extrakčnej úlohe sa bude zväčša jednať o údaje, ktoré sa na webovej stránke opakujú, sú všetky triedy zoradené do poľa na základe početnosti. Najpočetnejšie triedy budú na vrchole zoznamu, a tie ktoré sa na webovej stránke vyskytujú najmenej budú na konci zoznamu. Z toho ako fungujú triedne atribúty pri HTML značkách vieme určiť, že pre jednu skupinu prvkov, ktorá je definovaná jednou spoločnou charakteristikou, budú triedne atribúty spoločné.

## Zber údajov

Po zoradení triednych atribútov aplikácia začína so zberom údajov. Postupne od najpočetnejšieho atribútu (toho s najčastejším výskytom) sú dáta v danom okruhu určenom práve triednym atribútom uložené a následne kontrolované z pohľadu výskytu primárneho prvku. V prípade, že niektorý z prvkov danej skupiny nepredstavuje primárny prvok, je hodnota daného prvku zmenená na špeciálnu hodnotu *Null*. Táto kontrola prebieha porovnaním daného prvku s regulárnym výrazom pre primárny prvok. Hodnota *Null* pri určitom prvku ďalej aplikácií slúži ako záznam o nesplnení testu na regulárny výraz. Počet takýchto nesplnených testov je následne využitý v ďalšej časti analýzy.

Akonáhle aplikácia analyzuje všetky prvky z daného okruhu a určí, ktoré prvky nevyhovujú, postupuje na vyhodnotenie výsledného pomeru. Ten je následne porovnaný s maximálnym povoleným pomerom a v prípade, že spadá pod hranicu tohto maxima, je daný okruh prvkov určený ako výsledný.

Pri analýze sú teda kontrolované jednotlivé okruhy určené triednymi atribútmi. V prípade, že aplikácia zachytí okruh, ktorý spĺňa podmienky primárneho prvku na dostatočne dobrú hodnotu, kde krajná hodnota je určená používateľom, je daný okruh označený za výsledný a aplikácia pristúpi k extrakcii dát. Ostatné okruhy dát sú zahodené a nie sú kontrolované na prítomnosť žiadnych prvkov. Ak žiadny z okruhových dát nesplní určené požiadavky, je o tom používateľ informovaný a daný webový dokument nebude generovať výsledné údaje. Výsledný súbor ani súbor s metadátami nieje pre daný dokument generovaný. V prípade že sa jedná len o jeden dokument z kolekcie dokumentov, je generovanie výsledných údajov vynechané len pre daný dokument.

## 5.5 Generovanie výsledkov

Analýza webovej stránky určí výsledný okruh dát, v ktorom sa nachádza primárny prvok. Z jeho definície vyplýva, že je súčasťou alebo je v relatívnej blízkosti potencionálnych výsledkov. Primárny prvok je teda počiatočný bod, z ktorého bude primárne vyhľadávanie výsledných prvkov prebiehať.

Aplikácia najskôr určí počiatočný bod reprezentovaný uzlom. Pri tomto určovaní je veľmi dôležitá štruktúra webovej stránky reprezentovaná pomocou DOM<sup>3.2</sup>. Dá sa preto predpokladať, že nekvalitný alebo zložito štruktúrovaný kód webovej stránky môže ovplyvniť úspešnosť extrakcie pri danom webovom dokumente.

Následne prechádzame ku generovaní potencionálnych výsledkov. Tie sú generované nasledovným spôsobom:

1. Určenie otcovského elementu na základnú výšku 1
2. Ak je cieľový otcovský element vyššie v DOM strome, určenie otcovského elementu na danú výšku
3. Vytvorenie stromovej štruktúry obsahujúcej všetky detské prvky daného elementu
4. Odstránenie duplikátov
5. Príprava výsledkov
6. Kontrola počtu prvku vo výsledných objektoch, ak je tento počet menší ako cieľový počet, zvýšenie indexu cieľového otcovského elementu a vrátenie sa na bod 2.
7. Zápis výsledných objektov do odpovedajúcich súborov

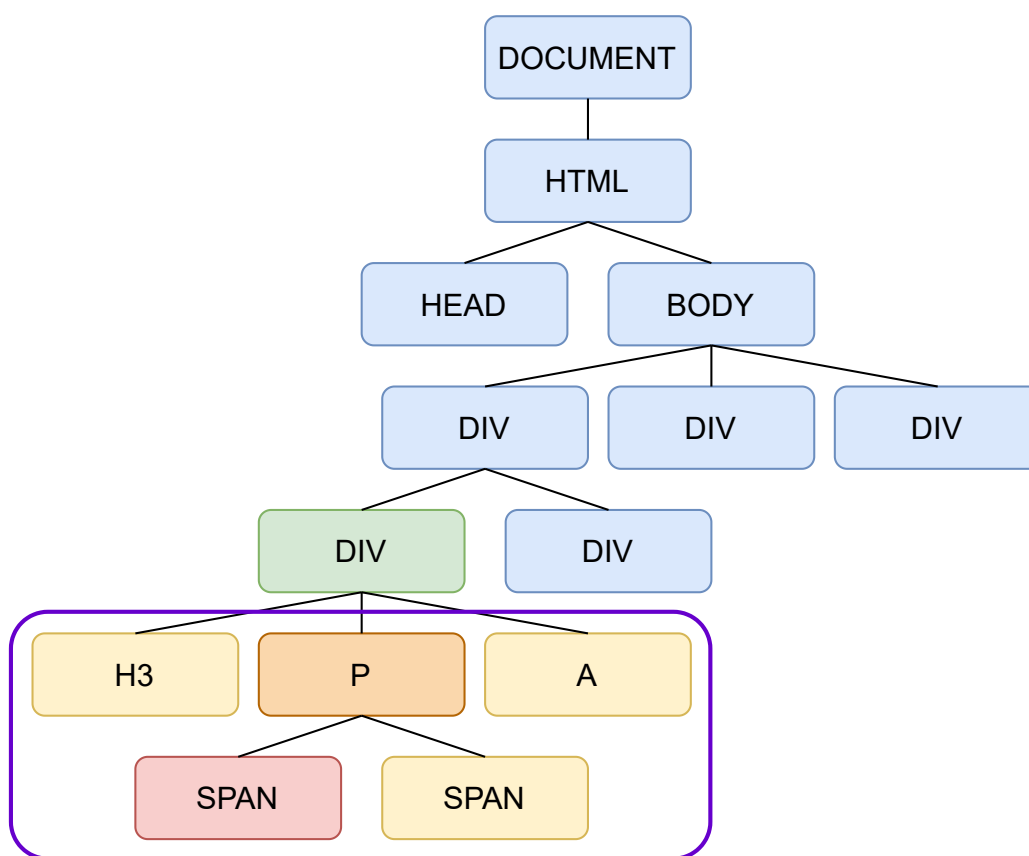
Z daného postupu je jasné, že pre správne určenie všetkých elementov je potrebné v rámci DOM stromu stúpať do správnej výšky tak, aby bolo možné určiť všetky prvky a zároveň aby stúpanie nebolo príliš vysoké. Preto aplikácia po prvom zaručení minimálneho množstva výsledných prvkov v objekte stúpanie preruší. Následne prebieha odstránenie prípadných duplikátov, ktoré pri práci s triednymi atribútmi môžu vzniknúť.

V rámci prípravy výsledkov prebieha hlavne formátovanie výsledkov a určenie ich dátového typu a reprezentácie. Detailnejší postup toho, ako to prebieha bude popísaný v rámci tejto kapitoly.

Po príprave výsledkov je na rade ich zápis. Ten prebieha ihneď po určení správneho setu výsledkov a výsledný súbor je uložený v preddefinovanom priečinku. Pre správne určenie názvu výsledného súboru je dôležité brať v úvahu aj typ extrakcie. Ak sa jedná o extrakciu z offline uložených súborov, nieje možné získať názov domény tradičným spôsobom. Názov je preto definovaný ako názov súboru, z ktorého bola extrakcia vykonávaná.

Následná vizuálna reprezentácia predstavuje typický prípad pri extrakcii a algoritmus, ktorý aplikácia pri extrakcii využíva. Jedná sa o typický príklad DOM stromu, kde sa vyskytujú aj rodičovské a detské elementy. Elementy vo fialovom rámci predstavujú elementy, ktoré sú predmetom záujmu extrakcie.

Element „SPAN“ označený červenou farbou je náš primárny element. Pri extrakcii teda aplikácia začína červeno označeným elementom „SPAN“ a označí otcovský element ako element „P“, ktorý je označený oranžovou farbou. Z daného otcovského elementu prebehne kontrola počtu detských elementov. Ich počet je však menší, ako cieľový počet prvkov zadanych pri extrakčnej úlohe. Algoritmus teda zvýši index cieľového otcovského prvku a tým nastaví na cieľový otcovský prvok element „DIV“ označený zelenou farbou. V tomto prípade je počet priamych a nepriamych detských elementov rovný počtu zadanych prvkov pri extrakčnej úlohe. Aplikácia teda našla cieľový otcovský element a elementy záujmu.



Obr. 5.2: Vizuálna reprezentácia algoritmu používaného pri extrakcii

### 5.5.1 Príprava výsledkov

V rámci generovania výsledkov prebieha aj ich finálna príprava pred zápisom. Vygenerované výsledky sú v rámci programu najskôr reprezentované len ako všeobecné prvky, ktorých dátový typ nieje určený a ich zaradenie je ekvivalentné. Následne prebehne odstránenie prípadných vzniknutých duplikátov na úrovni objektov.

Príprava výsledkov pozostáva z určenia dátového typu a príslušnosti daného prvku. V prípade, že takéto určenie nieje možné, je prvok označený ako *undefined*. Algoritmus analyzuje objekt na úrovni kľúčov a využíva špecifické pravidlá na zaradenie prvku k jeho príslušnej očakávanej reprezentácii. Ako prvé prebehne kontrola na výskyt prvku v zozname *blacklist*. Ak sa kľúč v tomto zozname nachádza, nieje ďalej určovaný jeho dátový typ a je z objektu vyradený.

V opačnom prípade prebehne kontrola na základe pravidiel definovaných vo vstupných súboroch. Tie sú reprezentované pomocou regulárnych výrazov, a tak sa prvky porovnávajú s regulárnymi výrazmi. V prípade, že prebehne zaradenie prvku splnením testu na regulárny výraz, je určený jeho dátový typ na základe príslušnosti regulárneho výrazu k definovanému dátovému typu. Následne je vytvorený nový objekt s prvkami, ktoré bolo možné zaradiť a tak im určiť príslušnosť. Zoznam všetkých takýchto objektov je potom naformátovaný do podoby JSON, a s *UTF8* kódovaním uložený v rámci výstupnej lokácie.

Po vygenerovaní výsledkov pre každý webový dokument v danej extrakčnej úlohe sú vygenerované metadáta a extrakcia je ukončená. Ak extrakcia skončí úspešne, je o tejto udalosti používateľ informovaný. Zároveň je zobrazený čas v milisekundách určujúci dĺžku extrakcie.

## Kapitola 6

# Testovanie a vyhodnotenie výsledkov

Aplikačné testovanie a vyhodnotenie dosiahnutých výsledkov je veľmi dôležitou súčasťou tvorby každého projektu. Preto sa táto kapitola zameriava práve na zhodnotenie výsledkov, ktoré vyvinutá aplikácia dosiahla.

Testovanie prebiehalo do určitej časti zároveň s vývojom aplikácie, avšak rozsiahlejšie testy boli uskutočnené až s vytvorením rôznych diverzifikovaných testovacích sád.

Testovanie pomocou týchto testovacích sád zároveň prebehlo na dvoch operačných systémoch, a to Windows 10 a Linux Manjaro. Pri testovaní a porovnávaní výsledkov medzi danými operačnými systémami nebola pozorovaná žiadna odchýlka alebo anomália.

### 6.1 Testovanie počas vývoja

K testovaniu bolo pristupované od samotného začiatku vývoja a priebežné výsledky boli konzultované s vedúcim práce. Hlavnou úlohou tohto typu testovania bolo detegovanie chýb, anomálií a neočakávaných výstupov. Zároveň slúžilo na usmernenie vývoja a tvorbu nových funkcií. To sa z väčšej časti podarilo a k prvotnému návrhu pribudli nové funkcie a vylepšenia. Niektoré z nich sa stali základnou zložkou programu a jeho neoddeliteľnou súčasťou, zatiaľ čo iné ostali zapracované ako voliteľné. V prípade voliteľných sa jednalo zväčša o funkcie a vlastnosti upresňujúce danú extrakčnú úlohu.

### 6.2 Testovacie prostredie

Ako už bolo spomenuté v úvode tejto kapitoly, testovacie prostredie z pohľadu operačného systému pozostávalo z nasledovných systémov:

- Windows 10
- Linux Manjaro



Čo sa webového prehliadača týka, použitá technológia Puppeteer nedokáže komunikovať s iným prehliadačom ako je Google Chrome<sup>1</sup>. V jej základnej verzii dokonca prináša v rámci inštalácie vlastný webový prehliadač, ktorým je práve prehliadač Chrome konkrétne vo verzii Chromium 86.0.4240.0. Pri každej inštalácii Puppeteer nainštaluje najnovšiu verziu prehliadača, a preto sa mnou používaná verzia v budúcnosti nemusí s inštalovanou zhodovať. Zachovanie kompatibility by ale malo byť zabezpečené, keďže o vývoj oboch technológií sa stará spoločnosť Google.

Obmedzenie na prehliadač Chromium sa môže zdať ako limitujúci faktor, avšak level integrácie, ktorý vďaka tomu dokáže technológia Puppeteer dosiahnuť prevyšuje všetky nedostatky.

## 6.3 Spôsob testovania

Prvotné testovanie bolo uskutočňované manuálne. Po extrakcii údajov z webovej stránky boli tieto údaje manuálne skontrolované a overená integrita dát.

V neskoršej fáze vývoja bol vyvinutý automatizačný skript pomocou programovacieho jazyka Python. Tento skript mal za úlohu porovnať očakávané údaje a údaje, ktoré aplikácia pri danej extrakčnej úlohe dokázala získať. Pri takomto porovnávaní bolo nutné zaviesť dve hlavné praktiky[18]:

- **Precision** a teda presnosť, ktorá má za úlohu odpovedať na otázku aká časť extrahovaných dát bola extrahovaná správne. Výsledná presnosť je vypočítaná nasledovným vzťahom:

$$Precision = \frac{ROR}{ROR + ROI} \quad (6.1)$$

Kde *ROR* predstavuje počet všetkých extrahovaných objektov, ktoré sú pre nás relevantné a *ROI* predstavuje počet irelevantných objektov. Týmto vzťahom vieme získať výslednú presnosť akú aplikácia dosiahla.

- **Recall**, ktorá odpovedá na otázku aká časť relevantných extrahovaných dát predstavuje očakávané dáta. Výsledná hodnota recall je preto daná vzťahom:

$$Recall = \frac{ROR}{EOR} \quad (6.2)$$

Kde *ROR* znova reprezentuje počet všetkých extrahovaných objektov, ktoré sú pre extrakciu relevantné a *EOR* reprezentuje počet všetkých relevantných očakávaných objektov.

Pomocou týchto praktík bolo následne testovanie upresnené a výsledné hodnoty určili mieru presnosti extrakcie. Webové stránky sa môžu kedykoľvek zmeniť, a preto pre nutnosť zachovania integrity dát webovej stránky bolo dôležité webovú stránku stiahnuť na lokálny disk a následne ju z neho načítavať. Webovú stránku je ale nutné stiahnuť aj so všetkými

---

<sup>1</sup>V čase vývoja a testovania aplikácie. Avšak podpora iných prehliadačov nieje v budúcnosti vylúčená.

závislostami a tak zachovať jej vzhľad a štruktúru. Stiahnutie webovej stránky konvenčnými nástrojmi alebo stiahnutie pomocou webového prehliadača nedokázalo dosiahnuť požadované výsledky. Za týmto účelom bol využitý nástroj Save Page WE<sup>2</sup>. Tento nástroj umožnil stiahnutie webovej stránky so všetkými závislosťami tak, že vzhľad a štruktúra zostali vo väčšine prípadoch zachované. Zároveň neboli pozorované žiadne výrazné anomálie a zmeny. Extrahovanie z live verzie stránky a z offline verzie bolo v konečnej fáze ekvivalentné.

Testovanie si taktiež vyžiadalo vytvorenie spoľahlivej testovacej sady, ktorá by bola schopná úspešnosť aplikácie overiť vo viacerých prípadoch použitia.

## 6.4 Testovacia sada

Vytvorenie spoľahlivej a dostatočne diverzifikovanej testovacej sady nieje jednoduchá úloha, a preto jej vytvorenie prebiehalo v spolupráci s autorom, ktorý rieši prácu na podobnú tému [5]. Táto skutočnosť bola riadne komunikovaná aj s vedúcim práce.

Finálna testovacia sada pozostávala zo 4 oblastí:

- Futbalové výsledky a program zápasov
- Eshop [www.tsbohemia.cz](http://www.tsbohemia.cz)
- Eshopy
- Správy a novinky

Kde prvé dve oblasti boli pripravené mnou, a posledné dve boli pripravené kolegom. Každá z týchto sád obsahuje 10 webových dokumentov.

### 6.4.1 Štruktúra testovacej sady

Základom každej testovacej sady je zložka s HTML súbormi, ktoré predstavujú jednotlivé webové dokumenty a ich očakávané extrahované hodnoty v druhej zložke. Očakávané hodnoty sú reprezentované JSON súbormi, kde každý nesie názov webového dokumentu, ktorého dáta predstavuje.

Zároveň sú súčasťou jednej testovacej sady súbory *config.json* a *data.json*, ktoré sú neoddeliteľnou súčasťou aplikácie a bližšie špecifikujú extrakčnú úlohu pri danej testovacej sade. Extrahované dáta sú následne uložené do zložky *output*

Jedna testovacia sada reprezentovaná práve jednou oblasťou má teda nasledovnú štruktúru, ktorá je zároveň kľúčová pre testovací skript a musí byť dodržaná:

```
Názov_oblasti
├── domains/ ..... Obsahuje HTML súbory s webovými dokumentami
├── expected/ ..... Obsahuje JSON súbory s očakávanými dátami
├── output/ ..... Obsahuje JSON súbory s vygenerovanými dátami
├── config.json ..... Predstavuje konfiguračný súbor aplikácie
└── data.json ..... Predstavuje vstupný súbor s používateľskými dátami
```

Pri kontrole výsledkov využíva testovací skript zložky *expected* a *output*, kde počet súborov ktoré obsahujú musí byť rovnaký.

<sup>2</sup><https://chrome.google.com/webstore/detail/save-page-we/dhhpefjklgkmgaeafimnjhoggjamoafof>

Po spustení aplikácie za účelom testu takejto sady, sú zároveň vygenerované metadáta do koreňového adresára tejto sady, teda vo výsledku sa budú nachádzať v rovnakom adresári ako `config.json` a `data.json`.

## 6.5 Testovací skript

Za účelom rýchlejšej a spoľahlivejšej kontroly bol vyvinutý testovací skript v programovacom jazyku Python. Nesie názov *ftest.py* a porovnáva vzájomne dva JSON súbory z jednej testovacej oblasti: Jeden zo zložky `expected`, druhý zo zložky `output`. Pri porovnávaní berie v úvahu metriky spomenuté v sekcii 6.3.

Skript berie ako parameter názov testovacej oblasti, čo je zároveň názov koreňového adresára jednej testovacej sady. Po kontrole štruktúry adresára sú načítané jednotlivé JSON súbory a skript začína s porovnávaním.

V prípade, že dva porovnávané súbory obsahujú rovnaký počet objektov, prebieha porovnávanie na úrovni kľúčov, pretože sa dá postupovať jednoznačným spôsobom a porovnávať jednotlivé kľúče objektov, čo vedie k presnejšiemu vyhodnoteniu.

Ak ale dva súbory neobsahujú rovnaký počet objektov, je veľmi zložitá priradiť jednotlivé kľúče k jednotlivým objektom. V takom prípade skript prechádza k menej presnému porovnávaniu na úrovni objektov. Vtedy je pre úspešnosť porovnania potrebná reprezentácia objektu 1:1, teda extrahovaný objekt sa musí nachádzať v očakávanom súbore.

Pri každom kľúči alebo objekte je pripočítaná hodnota k počtu irelevantných prvkov. Na záver, potom ako sú obidva súbory skontrolované, skript vygeneruje výsledné hodnoty `precision` a `recall`. Zároveň si skript drží internú pamäť týchto čiastkových výsledkov a po skončení testovaní jednej testovacej sady vygeneruje priemernú hodnotu `precision` a `recall`.

Pri spustení skriptu s parametrom `-o` sú zároveň tieto priemerné hodnoty zapísané do dočasného súboru `overall`. Po otestovaní viacerých testovacích sád je následne možné spustiť skript *overall.py*, ktorý vypočíta priemernú váženú hodnotu `precision` a priemernú váženú hodnotu `recall`.

Skript po spustení otestuje určenú testovaciu sadu a vygeneruje výsledky. Tie sa skladajú z nasledovných údajov:

- **Status:** Určuje status kontroly daného webového dokumentu
- **Precision:** Hodnota `precision` pre daný webový dokument
- **Recall:** Hodnota `recall` pre daný webový dokument
- **Domain:** Názov súboru s očakávanými výsledkami (nesie názov webového dokumentu)

Na záver sú zároveň vypísané priemerné výsledky pre testovanú sadu.

Vytvorenie tohto testovacieho skriptu bolo zároveň kľúčové pre celkové zhodnotenie výsledkov, aké aplikácia bola schopná dosiahnuť. Pri takom množstve súborov, aké sa v testovacej sade nachádza, by manuálna kontrola a vyhodnotenie prebiehala len veľmi ťažko.

## 6.6 Vyhodnotenie výsledkov

Pri vyhodnocovaní výsledkov boli využité metriky spomínané v sekcii 6.3 a testovacie sady zo sekcie 6.4. Testovacia sada „Správy a novinky“ bola pri testovaní z dôvodu lepšieho určenia požiadaviek a regulárnych výrazov rozdelená na dve. Celkové vyhodnotenie výsledkov prináša možnosť overiť kvalitu výslednej aplikácie, a preto je správne vyhodnotenie výsledkov veľmi dôležité.

Nejedná sa totiž len o jednoduché vyobrazenie výsledných dát a porovnanie dát s dátami očakávanými. Táto metrika nám síce napovie kde aplikácia dominuje a kde zlyháva, avšak je treba zväžiť aj koreláciu a kauzalitu týchto výsledkov.

V úvode tejto kapitoly bola uvedená skutočnosť, že na výsledné hodnoty nemá zmena operačného systému významný zmysel. Pri vyhodnocovaní bol teda použitý operačný systém Windows 10 vo verzii 20H2.

Webové prehliadače vyžadujú pre načítanie webového dokumentu jeho absolútnu cestu. Pre zaručenie prenositeľnosti testovacích sád boli však v týchto sádach uvádzané cesty relatívne. Pre správne spustenie testov a zaručenie nahradenia relatívnych ciest, je preto vyžadované spustenie aplikácie s parametrom `--dataset`. Použitie relatívnej cesty predpokladá umiestnenie testovacích sád v zložke `/tests/dataset/` z pohľadu koreňového adresára aplikácie.

Všetky testy sú zároveň spustiteľné pomocou priloženého súboru *Makefile*. Jeho možnosti a dostupné parametre sú popísané v *README.md* súbore priloženom v koreňovom adresári aplikácie.

### Futbalové výsledky a program zápasov

Testovacia sada zameraná na futbalové výsledky je zameraná na webové stránky zobrazujúce informácie o futbalových zápasoch. Táto sada bola zvolená pre fakt, že takéto stránky sa síce na prvý pohľad zdajú podobné, avšak nie vždy je to pravda.

Niektoré obsahujú informácií viac, niektoré menej. Niektoré obsahujú irelevantné dáta ktoré treba odfiltrovať, iné obsahujú len dáta ktoré sú očakávané.

Ako cielené dáta boli určené čas, názov domáceho tímu, skóre a názov hostujúceho tímu. Za primárny prvok bol zvolený časový údaj. Štruktúra vstupných dát preto vyzerá nasledovne:

```
"structure":{
    "time": "time",
    "team1": "string",
    "score": "score",
    "team2": "string"
}
```

Pri vyhodnocovaní výsledkov boli taktiež zohľadnené údaje, ktoré sú generované v rámci extrakcie. Tieto údaje neboli generované výsledným testovacím skriptom, a sú viazané na aktivitu aplikácie.

Z vygenerovaných údajov teda vieme vytvoriť nasledovnú tabuľku:

Tabuľka 6.1: Výsledné hodnoty testovacej sady

Doména	Precision	Recall	OC	aTime	eTime	lTime
365scores.com	100,0%	100,0%	18	114ms	5ms	995ms
777score.com	100,0%	100,0%	154	92ms	15ms	1158ms
LiveScore.com	100,0%	99,07%	81	90ms	5ms	852ms
SkySports.com	100,0%	100,0%	200	114ms	22ms	849ms
Soccerstand.com	100,0%	98,64%	110	67ms	11ms	1192ms
bbc.com	100,0%	100,0%	47	66ms	3ms	759ms
flashscore.com	100,0%	98,66%	112	76ms	8ms	1411ms
flashscore.sk	100,0%	66,95%	79	154ms	7ms	1502ms
scoreboard.com	92,84%	47,73%	110	103ms	10ms	1265ms
soccer24.com	86,14%	53,93%	80	292ms	10ms	1516ms

Kde všetky časové údaje predstavujú priemernú hodnotu z 5 pokusov, ich hodnota je zaokrúhlená na celé milisekundy a ďalej:

- **OC:** Počet extrahovaných objektov
- **aTime:** Čas potrebný na detekciu a analýzu primárneho prvku
- **eTime:** Čas potrebný na určenie príslušnosti a finálnu extrakciu údajov
- **lTime:** Čas potrebný na načítanie webového dokumentu

Vykonanie extrakcie na týchto 10 webových dokumentov priemerne trvalo 13627ms, čo nám dáva hodnotu 1362,7ms na jeden dokument.

Z tejto hodnoty je priemerne 116,8ms aTime (8,6%), 9,6ms eTime (0,7%) a 1149,9ms lTime (84,39%). Zostávajúci čas (6,31%) pripadal na réžiu aplikácie. Priemerná úspešnosť predstavovala hodnotu 97,9% a priemerná hodnota recall hodnotu 86,5%.

Z následnej analýzy výsledkov a detailnej analýzy webových dokumentov sa podarilo zistiť, že strata na hodnotách recall je daná princípom fungovania aplikácie.

Aplikácia združuje elementy na základe ich atribútu triedy. Avšak v niektorých prípadoch, ak sa časový údaj športového záznamu zmenil na „Finished“ alebo „Canceled“, sa atribút triedy zmenil tiež. Aplikácia tak tieto záznamy nezaradila do výsledkov a preto boli stratené.

# Literatúra

- [1] BIDELMAN, E. *Getting Started with Headless Chrome* [online]. Apríl 2017. Revidované 25.2.2021. (Navštívené 27.2.2021). Dostupné z: <https://developers.google.com/web/updates/2017/04/headless-chrome>.
- [2] CHAPMAN, R. *5 Best languages for web scraping* [online]. December 2019. (Navštívené 19.4.2021). Dostupné z: <https://limeproxies.netlify.app/blog/5-best-web-scraping-languages>.
- [3] ELLIOTT, E. *How Popular is JavaScript in 2019? | JavaScript Scene | Medium* [online]. Máj 2019. (Navštívené 31.3.2021). Dostupné z: <https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1>.
- [4] JAVATPOINT. *Web Scraping Using Python - Javatpoint* [online]. (Navštívené 31.3.2021). Dostupné z: <https://www.javatpoint.com/web-scraping-using-python>.
- [5] MAŠTERA, F. *Inteligentní extrakce dat ve webovém prohlížeči*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- [6] MDN, C. *CSS: Cascading Style Sheets* [online]. September 2020. Revidované 14.4.2021. (Navštívené 14.4.2021). Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [7] MDN, C. *HTML: HyperText Markup Language* [online]. September 2020. Revidované 14.4.2021. (Navštívené 15.4.2021). Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
- [8] MDN, C. *Introduction to the DOM - Web APIs | MDN* [online]. September 2020. Revidované 9.4.2021. (Navštívené 9.4.2021). Dostupné z: [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction).
- [9] MDN, C. *What is JavaScript? - Learn web development | MDN* [online]. September 2020. Revidované 19.2.2021. (Navštívené 31.3.2021). Dostupné z: [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\\_steps/What\\_is\\_JavaScript](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript).
- [10] MDN, C. *JSON* [online]. September 2021. Revidované 15.4.2021. (Navštívené 19.4.2021). Dostupné z: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON).
- [11] NODE.JS. *What is npm? | Node.js* [online]. August 2011. Navštívené 1.2.2021). Dostupné z: <https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/>.

- [12] NODE.JS. *About / Node.js* [online]. Január 2020. (Navštívené 1.2.2021). Dostupné z: <https://nodejs.org/en/about/>.
- [13] OCTOPARSE. *What is Web Scraping and How Does It Work / Octoparse* [online]. Október 2018. (Navštívené 30.3.2021). Dostupné z: <https://www.octoparse.com/blog/web-scraping-introduction>.
- [14] PUPPETEER. *Puppeteer v8.0.0* [online]. Apríl 2020. Revidované 15.3.2021. (Navštívené 17.3.2021). Dostupné z: <https://pptr.dev/>.
- [15] RUNGTA, K. *15 Best Web Scraping Tools for Data Extraction in 2021* [online]. Január 2020. Revidované 9.4.2021. (Navštívené 15.4.2021). Dostupné z: <https://www.guru99.com/web-scraping-tools.html>.
- [16] TARUN007. *Difference between cheerio and puppeteer - GeeksforGeeks* [online]. Júl 2020. (Navštívené 31.3.2021). Dostupné z: <https://www.geeksforgeeks.org/difference-between-cheerio-and-puppeteer/>.
- [17] TEAM, C. D. *Chrome DevTools Protocol* [online]. Marec 2020. (Navštívené 4.3.2021). Dostupné z: <https://chromedevtools.github.io/devtools-protocol/>.
- [18] TING, K. M. Precision and Recall. In: SAMMUT, C. a WEBB, G. I., ed. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, s. 781–781. ISBN 978-0-387-30164-8. Dostupné z: [https://doi.org/10.1007/978-0-387-30164-8\\_652](https://doi.org/10.1007/978-0-387-30164-8_652).
- [19] WOLF, K. Jan Čurn (Apify): Z webu stahujeme už miliardu stránek měsíčně - Lupa.cz. *Článok* [online]. 1. vyd. Lupa.cz. Január 2020, č. 1, [cit. 30.3.2021]. Dostupné z: <https://www.lupa.cz/clanky/jan-curn-apify-z-webu-stahujeme-uz-miliardu-stranek-mesicne/>.

## Príloha A

# Obsah priloženého CD

```
/
├── src/.....Obsahuje konfiguračné súbory a finálnu verziu aplikácie
│   ├── config.json..... Základný konfiguračný súbor
│   ├── data.json..... Základný súbor so vstupnými dátami
│   └── extractor.js..... Finálna verzia aplikácie
├── tests/.....Obsahuje testovacie skripty a datasety
│   ├── dataset/..... Zložka obsahuje jednotlivé datasety
│   ├── ftest.py..... Hlavný testovací skript
│   └── overall.py..... Skript na vyhodnotenie celkových vážených výsledkov
├── thesis/..... Adresár obsahuje technickú správu a jej zdrojové kódy
├── Makefile
├── package.json.....Obsahuje zoznam závislostí nutný pre inštaláciu
└── README.md
```



# Príloha B

## Manuál

Tento manuál obsahuje návod na inštaláciu, popis jednotlivých cieľov súboru Makefile a dostupných parametrov pri spúšťaní aplikácie.

### B.1 Návod na inštaláciu

Aplikácia po stiahnutí obsahuje len základné zdrojové kódy a preto je potrebné nainštalovať závislosti. Tie budú v prípade Node.js nainštalované do zložky `node_modules` v koreňovom adresári aplikácie.

Medzi hlavné prerekvizity patria:

- Node.js 15.12.0
- npm 7.7.0
- puppeteer 8.0.0
- css.escape 1.5.1
- minimist 1.2.5
- python3 ( Voliteľné - Iba pre účely spustenia a vyhodnotenia testov)

Inštalácia sa medzi operačnými systémami líši iba pri inštalácii Node.js. To sa v prípade operačného systému Windows 10 môže nainštalovať klasicky pomocou `.exe` súboru dostupného z oficiálneho webu Node.js<sup>1</sup>. Node.js sa dá inštalovať aj priamo v termináli, avšak v mnohých prípadoch táto inštalácia prebieha pomocou takzvaných „Správco balíkov“. Jedným z takýchto správco pre systém Windows 10 je aj Chocolatey<sup>2</sup>. Arch verzia Linuxu ponúka napríklad snapd<sup>3</sup>.

Pri inštalácii Node.js sa automaticky inštaluje aj `npm`, pretože `npm` je správca modulov pre Node.js. Nasleduje teda inštalácia modulov potrebných pre spustenie aplikácie.

Pri tejto inštalácii je možné využiť priložený súbor `package.json`, ktorý obsahuje všetky závislosti dostupné v `npm`. Príkazom `npm install` sa tieto závislosti nainštalujú a aplikácia je pripravená na spustenie.

---

<sup>1</sup><https://nodejs.org/en/download/>

<sup>2</sup><https://chocolatey.org/>

<sup>3</sup><https://snapcraft.io/>

## Makefile

Súbor `Makefile` zároveň podporuje príkazy na zjednodušenie tohto postupu. Príkazom `make windows-install` prebehne inštalácia Node.js a závislostí npm pre operačný systém Windows 10 za pomoci správcu Chocolatey. Príkazom `make linux-install` prebehne inštalácia pre operačný systém Linux.

Súbor `Makefile` podporuje nasledovné príkazy:

- **tests** - otestuje všetky testovacie sady a vypíše vyhodnotenie výsledkov.
- **dataset** - otestuje všetky testovacie sady bez vyhodnotenia výsledkov
- **testall** - vyhodnotí výsledky testovacích sád
- **football** - spustí test testovacej sady „Futbalové výsledky a program zápasov“
- **tsbohemia** - spustí test testovacej sady „Eshop [www.tsbohemia.cz](http://www.tsbohemia.cz)“
- **shops** - spustí test testovacej sady „Eshopy“
- **news** - spustí test testovacej sady „Správy a novinky“ (prvá časť)
- **news3** - spustí test testovacej sady „Správy a novinky“ (druhá časť)
- **windows-install** - inštalácia závislostí a prerekvizít pre systém Windows
- **linux-install** - inštalácia závislostí a prerekvizít pre systém Linux
- **clean** - vymaže súbory vytvorené po vyhodnotení výsledkov
- **pack** - zbalí potrebné súbory do odovzdávaného archívu

## B.2 Štruktúra konfiguračných súborov

Dôležitým faktorom je zachovanie štruktúry konfiguračných súborov. Nasledujúce príklady predstavujú konfiguračné súbory priložené v zložke s výslednou aplikáciou. Sú plne funkčné a ako ukážku extrahujú údaje z preddefinovanej webovej stránky.

Príklad konfiguračného súboru `config.json`:

```
{
  "maxFailRatio": 70,
  "format": {
    "price": "^\\d+\\s*€$",
    "string_title": "[a-zA-Z]{8,60}$",
    "string_description": "[a-zA-Z]{70,300}$",
    "location": "[a-zA-Z]{5,15}$",
    "views": "\\d+\\s*x",
    "date": "\\d+\\.\\d+\\.\\d*"
  },
  "primary": "^\\d+\\s*€$"
}
```

Príklad dátového súboru `data.json`:

```
{
  "urls": [
    "https://mobil.bazos.sk/apple/"
  ],
  "structure": {
    "price": "price",
    "title": "string_title",
    "date": "date",
    "description": "string_description",
    "location": "location",
    "views": "views"
  }
}
```

## B.3 Dostupné parametre

Aplikácia podporuje viacero parametrov pri jej spúšťaní. Úlohou týchto parametrov je bližšie špecifikovať danú extrakčnú úlohu. Pre úspešné spustenie aplikácie je však požadovaný len jeden parameter, a to `--data`. Ten špecifikuje vstupný dátový súbor.

Zoznam dostupných parametrov:

- **--data** : špecifikácia dátového súboru
- **--config** : špecifikácia konfiguračného súboru
- **--offline** : extrakcia z offline súborov
- **--noundef** : vymazanie nedefinovaných prvkov z výsledného súboru
- **--dataset** : transformuje relatívne cesty datasetov na absolútne
- **-o** : špecifikácia výstupného adresáru
- **-v** : vypisovanie podrobných informácií do konzoly
- **-h** : zobrazí pomocné info
- **-d** : pred definovaním prvku skontroluje, či nebol v minulosti definovaný
- **-g** : určí príslušnosť nedefinovaných prvkov na základe predchádzajúcich výsledkov
- **-b** : kontrola prvkov na prezenciu v blackliste
- **-u** : zjednotenie objektov (všetky objekty majú rovnaký počet prvkov)
- **-m** : striktné kontrolovanie na základe regulárnych výrazov

Použitie parametrov pri spúšťaní aplikácie:

```
node extractor.js --data=data.json [--dataset] [--config=config.json]
[-o=output/folder/] [-bdgu] [-m] [-p] [-v] [-h] [--offline] [--noundef]
```

## B.4 Návratové kódy

Aplikácia pri každom zlyhaní používateľa informuje o vzniknutej chybe. Každý návratový kód definuje typ chyby ktorá nastala.

Zoznam návratových kódov a ich popis:

- **Error 1** : nebolo možné otvoriť jeden z konfiguračných súborov
- **Error 2** : vyžadovaný parameter nebol zadaný
- **Error 3** : nebolo možné nastaviť základné parametre programu
- **Error 4** : parameter bol povolený ale neboli splnené požiadavky
- **Error 5** : nemožno vytvoriť výstupný súbor alebo zložku
- **Error 6** : nesprávna štruktúra konfiguračných súborov
- **Error 7** : interná chyba