VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÉ ZÍSKÁVÁNÍ INFORMACÍ Z WWW

AUTOMATED RETRIEVAL OF INFORMATION FROM THE WWW

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ANDREJ ŽABKA**
AUTHOR

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

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Žabka Andrej**
Program: Informační technologie
Název: **Automatizované získávání informací z WWW**
Automated Retrieval of Information from the WWW
Kategorie: Web

Zadání:

1. Seznamte se s dostupnými nástroji a knihovnami pro přístup k obsahu webových dokumentů.
2. Prostudujte současné metody extrakce informací z webových stránek.
3. Navrhněte architekturu nástroje pro automatizovaný přístup a získávání definovaných informací z většího množství webových zdrojů.
4. Po dohodě s vedoucím zvolte vhodnou platformu a implementujte navržené řešení.
5. Proveďte testování řešení na vhodné množině reálných stránek.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Perina, L.: Metody extrakce dat z webových stránek. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- Maštera, F.: Inteligentní extrakce dat ve webovém prohlížeči. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- Burget, R.: Information Extraction from the Web by Matching Visual Presentation Patterns. In: *Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia*. Lecture Notes in Computer Science vol. 10579. Kobe: Springer International Publishing, 2017

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, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 18. října 2021

Abstrakt

Táto bakalárska práca sa zaoberá extrakciou dát z webu (web scraping) a následným zobrazovaním týchto dát. Vytvorený nástroj umožňuje užívateľovi rýchlo a jednoducho vytvoriť celistvý projekt, ktorý dokáže získavať dáta z viacerých webových stránok a zobraziť ich v prívetivej podobe. Taktiež je súčasťou práce aj niekoľko príkladov, ktoré slúžia ako ukážka možností tohto nástroja a boli použité pri jeho testovaní.

Abstract

This bachelor thesis deals with data extraction from web (web scraping) and displaying this data. The created tool allows it's user to quickly and simply create a project, that can extract data from multiple web sites and display them in a user friendly fashion. The thesis also contains examples, that showcase the abilities of this tool and were used in it's testing.

Kľúčové slová

Web, scraping, zobrazovanie dát, JavaScript, Puppeteer, HTML, DOM, CSS

Keywords

Web, scraping, data presentation, JavaScript, Puppeteer, HTML, DOM, CSS

Citácia

ŽABKA, Andrej. *Automatizované získávání informací z WWW*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Automatizované získávání informací z WWW

Prehlásenie

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

.....
Andrej Žabka
5. mája 2022

Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce pánovi Burgetovi za všetku ochotu a úsilie, ktoré venoval mne a mojej práci. Taktiež ďakujem svojej rodine a svojim kamarátom, ktorí ma počas tvorby tejto práce podporovali a motivovali.

Obsah

Kapitola 1

Úvod

Automatizované získavanie informácií z internetu, tiež známe ako web scraping, je v dnešnej dobe veľmi rozšírená a hlboko preskúmaná téma. Internet obsahuje nespočetné množstvo verejne prístupných dát. Z toho je jasné, že existuje mnoho dôvodov, prečo sa venovať tejto téme. Hlavným z nich je samozrejme zisk. Najmä veľké firmy používajú rôzne nástroje na sledovanie konkurencie, monitorovanie užívateľov a reputácie firiem, porovnávanie cien produktov a mnoho ďalších. Pre tieto použitia existujú špecializované nástroje a firmy, ktoré ponúkajú automatické získavanie obrovského množstva dát a ich následnú analýzu.

Ďalším dôvodom prečo sa tejto téme venovať, síce menším, no stále relevantným, je uľahčenie a urýchlenie získavania informácií z webu pre jednotlivca, prípadne menšiu firmu. Mnoho z nás každodenne sleduje informácie na webe ako predpoveď počasia, výsledky športových zápasov, ceny akcií na trhu a podobne. Častokrát tieto dáta niesú v najprehľadnejšej podobe, alebo nám trvá dlho sa ku nim preklikať.

Práve takýmto prípadom sa zaoberá táto bakalárska práca. Existuje viacero spôsobov ako si vytvoriť vlastný web scraper, no ich použitie z pravidla nieje najjednoduchšie a ich pochopenie môže zabrať veľa času aj skúsenému informatikovi. Taktiež existujú aj komerčné produkty, ktoré ponúkajú možnosť automatizovať získavanie dát z internetu, no ani tieto produkty nemusia byť vhodné pre bežného človeka. Preto bol v rámci tejto práce vyvinutý nástroj, ktorý uľahčuje užívateľovi vytvoriť vlastný web scraper, ktorý dokáže dáta z webu získať a taktiež ich zobrazovať v žiadanej podobe.

V nasledujúcej kapitole 2 je podrobne vysvetlený pojem web scraping a technológie, ktoré nám ho umožňujú. Následne sú v kapitole 3 popísané technológie, pomocou ktorých je možné tvoriť web scrapery a už spomenuté komerčné produkty. Táto kapitola obsahuje aj zhrnutie výhod a nevýhod použitia jednotlivých prístupov a konkrétnych možností v rámci daného prístupu.

Kapitola 4 obsahuje návrh môjho riešenia a kapitola 5 popis jeho implementácie. Na koniec je v kapitole 6 zhodnotenie výsledného nástroja a jeho testovania.

Kapitola 2

Web scraping

Táto kapitola vysvetľuje čo vlastne znamená web scraping a jeho rôzne spôsoby využitia. Následne stručne opisuje technológie, ktorých základy je nutné poznať aby sme mohli lepšie pochopiť princípy web scrapingu.

2.1 Čo je to web scraping

Automatizované získavanie informácií z webu, známe hlavne ako web scraping, je proces získavania dát z webu bez použitia API[1] [10]. Za web scraping sa teda teoreticky dá považovať aj manuálne prehliadanie webu cez prehliadač vykonávané človekom, no v praxi takmer nikdy nemá tento pojem takýto význam. Web scraping je vykonávaný automatizovaným programom alebo scriptom.

Pojem web scraping začal vznikať hneď so vznikom internetu. Je to teda veľmi stará a preskúmaná oblasť, no kvôli prudkému rastu internetu a neustálemu vývoju nových technológí je čoraz dôležitejšia a stále sa vyvíja.

Prečo je scraping potrebný

Objem dát na webe sa podľa štúdií aktuálne pohybuje v desiatkach až stovkách zettabytov[2] a neustále sa zväčšuje [1]. Mnoho z týchto dát potrebujeme nejakým spôsobom ďalej spracovávať. Napríklad agregovať dáta z viacerých zdrojov, analyzovať ich, vytvárať z nich štatistiky a mnoho ďalších. Dáta na webe ale často niesú štruktúrované. Webové stránky sú vytvárané v jazyku HTML[3], ktorého hlavný ciel je vizuálna prezentácia dát a nieje určený na ďalšie spracovanie. Na ďalšie strojové spracovanie je ale potrebné mať dáta v štruktúrovanej podobe - napríklad JSON[4] alebo vo forme tabuliek relačnej databázy [2]. Práve táto transformácia dát z webu na štruktúrované dáta je hlavná úloha web scrapingu.

[1] API - Application Programming Interface
[2] zettabyte - 10^{21} Bytov = miliarda terrabytov
[3] HTML - Hypertext Markup Language
[4] JSON - JavaScript Object Notation

Využitie web scrapingu v praxi

V praxi je web scraping veľmi rozšírený. Používajú ho veľké firmy napríklad na: [13]

- Sledovanie cien nehnuteľností
- Získavanie štatistík vo svojom priemysle
- Porovnávanie cien produktov
- Získavanie údajov o potenciálnych zákazníkoch a klientoch

Existuje mnoho firiem, ktoré sa zaoberajú web scrapingom a ponúkajú ostatným firmám produkty zamerané práve na tieto spôsoby využitia. Tieto firmy a ich produkty, ich výhody a nevýhody budú bližšie popísané v kapitole 3.

Scraping ale môžu využívať aj jednotlivci alebo menšie skupiny ľudí. Samozrejme ich ciele môžu byť rovnaké ako tie veľkých firiem, no často nemajú dostatok prostriedkov na vykonávanie scrapingu v tak veľkej miere. V menšej miere sa dá scraping používať napríklad na výskum, urýchlenie bežných činností na webe alebo periodické získavanie premenlivých dát, ktorých vývin nás zaujíma.

Samotný proces web scrapingu sa dá rozdeliť na 3 hlavné časti. Tými sú: [2]

- Získanie zdrojových dát
- Nájdenie a extrakcia dát
- Uloženie (prípadne zobrazenie) výsledkov

Každá z týchto častí je rovnako dôležitá a nemôže byť zanedbaná. V praxi môžu byť tieto časti reprezentované tromi samostatnými programami, kde jeden stiahne z webu žiadaný HTML dokument, následne ďalší tento dokument rozparsuje a extrahuje žiadané dáta. Nakoniec tretí program tieto dáta uloží alebo zobrazí v žiadanej forme. Takéto delenie ale nieje nutné a pomocou existujúcich nástrojov sme schopní bez problémov obsiahnuť všetky časti v rámci jedného programu alebo skriptu.

2.2 Webové technológie

Pre lepšie pochopenie princípov web scrapingu je potrebné porozumieť štruktúre webových stránok a teda aj technológiám, pomocou ktorých sú webové stránky vytvárané. Opisom týchto technológií sa zaoberá táto podkapitola.

HTML

Hypertext Markup Language je základný stavebný blok všetkých webových stránok. Definuje obsah a štruktúru stránky. Ostatné technológie ako CSS a Javascript sú používané na definovanie výzoru a správania týchto stránok.

Slovo “Hypertext” v jeho názve znamená možnosť použitia linkov, ktoré prepájajú jednotlivé webstránky medzi sebou a sú jedným z hlavných aspektov webu.

Slovo “Markup” znamená spôsob anotácie obsahu webstránky pomocou špeciálnych elementov ako napríklad *<head>* alebo *<img>*. Jednotlivé elementy sú v rámci dokumentu definované pomocou tagov, ktoré sa skladajú z názvu elementu obkoleseného znakmi “<>”.

Elementy môžu obsahovať aj určité atribúty, ktoré definujú ďalšie vlastnosti daného elementu. Bežne používané atribúty sú napríklad *href*, ktorý definuje link na stránku, na ktorú sa má prejsť po kliknutí na daný element, alebo *id*, ktorý udáva unikátny identifikátor daného elementu. [7]

CSS

CSS alebo Cascading Style Sheets je jazyk používaný na popis dokumentu napísaného v jazyku HTML alebo XML. Definuje ako majú byť jednotlivé elementy zobrazené vo webovom prehliadači. Je to jeden zo základných jazykov webu a je štandardizovaný podľa W3C[5] špecifikácií [16]. [6]

CSS umožňuje oddelenie obsahu a výzoru web stránky tým, že CSS kód pre daný HTML dokument sa môže nachádzať v samostatnom súbore. Toto oddelenie v sebe nesie mnoho výhod. Napríklad uľahčuje prístup k dátam, umožňuje prezentovať ten istý obsah stránky v rôznych podobách na základe spôsobu zobrazenia stránky, urýchluje znovunačítanie stránok a umožňuje viacerým stránkam zdieľať rovnaký formátovací CSS súbor a teda uľahčuje aj vývoj.

Pre samotný scraping teda nieje príliš dôležitý, keďže neobsahuje žiadne podstatné dáta a všetky dáta sú extrahované z HTML dokumentov. Táto práca sa ale zaoberá aj zobrazovaním získaných dát v užívateľsky prívetivej podobe a je teda nutné opísať aspoň jeho základy.

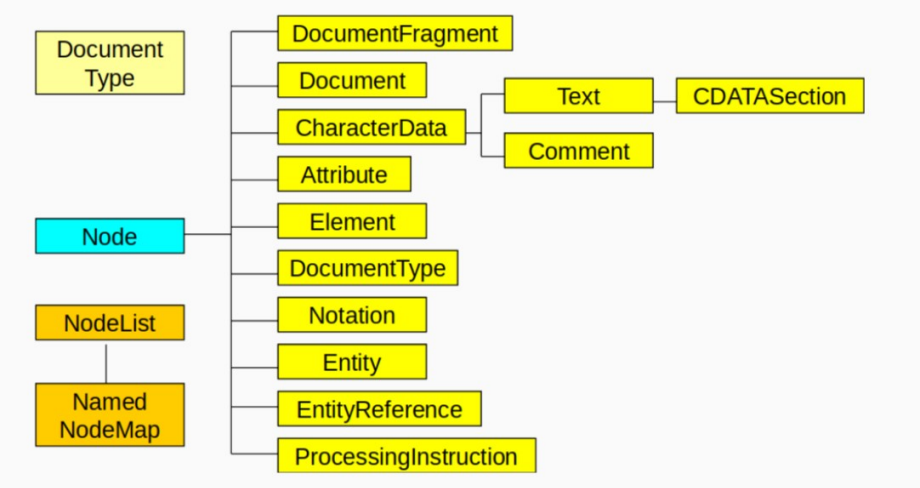
Taktiež je potrebné spomenúť, že HTML elementy bývajú na stránkach často pomenované a rozdelené do pomenovaných skupín práve pre to, aby mohli byť pomocou CSS štylizované. Toto často uľahčuje extrakciu informácií z dokumentu tým, že vieme podľa týchto pomenovaní jednoducho nájsť určitú skupinu hodnôt, ktoré nás zaujímajú. Tento postup bol používaný aj pri testovaní nástroja vytvoreného v rámci tejto práce a bude opísaný v kapitole 6.

[5] W3C - World Wide Web Consortium

Document Object Model

Document Object Model alebo skrátene DOM je programovacie rozhranie pre dokumenty vo formáte HTML alebo XML[6]. Reprezentuje dokument, aby programy mohli meniť jeho štruktúru, obsah a štylizáciu [8].

DOM je reprezentovaný stromovou štruktúrou objektov, ktoré môžu byť rôznych tried.



Obr. 2.1: Triedy DOM, prevzaté z [2]

Keďže nám DOM umožňuje pracovať s dokumentom cez programovacie rozhranie, je veľmi užitočný pri web scrapingu. Dokument je reprezentovaný v objektovo-orientovanej podobe a pomocou skriptu alebo programu dokážeme manipulovať s jeho jednotlivými elementmi. Z pohľadu scrapingu je hlavná výhoda to, že nám umožní vyhľadať všetky elementy v dokumente, ktoré majú spoločnú vlastnosť ako napríklad názov triedy.

[6]XML - Extensible Markup Language

Kapitola 3

Existujúce nástroje a riešenia

V tejto kapitole budú najskôr spomenuté alternatívy k jazyku JavaScript a zdôvodnenie, prečo bol práve on použitý pri implementácií tejto práce. Potom bude podrobne opísaný práve jazyk JavaScript a jeho najdôležitejšie súčasti. Následne budú spomenuté dve najpoužívanejšie knižnice, určené na vytváranie vlastných web scraperov a na koniec bude vymenovaných niekoľko rôznych existujúcich komerčných produktov, ktoré ponúkajú scraping, ich výhody a nevýhody.

3.1 Alternatívy JavaScriptu

Ako je v informatike bežné, aj web scraping môže byť vykonávaný pomocou širokého spektra rôznych nástrojov a rôznych programovacích jazykov. V tejto podkapitole budú stručne opísané dve alternatívy. Jednou je veľmi nízkoúrovňový prístup pomocou príkazového riadku, druhou zasa naopak vysoko abstraktný prístup cez veľmi známy jazyk Python.

Shell

Unix Shell je interpret príkazového riadku. Poskytuje rozhranie príkazovného riadku na ovládanie Unix-ového systému. Keďže nieje podstatnou súčasťou tento práce, nebudú opisované detaily ako jeho história a jeho rôzne prevedenia ako napríklad Bash alebo Korn Shell. Všetky tieto technológie sú veľmi staré, no iba samotný fakt že sa používajú dodnes, svedčí o ich dôležitosti a sile. Podrobnejší popis týchto technológií je možné nájsť napríklad tu [15].

Shell spolu s jeho mnohými nástrojmi nazývanými “utility” nám poskytuje takmer neobmedzené možnosti vo všetkých oblastiach informatiky. Výnimkou nieje ani web scraping. Ako bolo v predchádzajúcej kapitole definované, web scraping pozostáva z troch základných podúloh. Tými sú získanie zdrojových dát, nájdenie a extrakcia dát ktoré chceme a nakoniec uloženie výsledkov. Všetky tieto podúlohy sa dajú veľmi jednoducho vykonať aj cez príkazový riadok. Zdrojové dáta (v tomto prípade HTML dokument) dokážeme získať napríklad pomocou nástroja *wget*. Následne môžeme nájsť a extrahovať chcené dáta pomocou nástrojov ako sú *grep*, *sed* alebo *awk*. Na uloženie získaných dát nepotrebujeme žiaden špeciálny nástroj, túto funkcionalitu nám poskytuje samotný shell. Príklad takéhoto scrapingu pomocou príkazového riadka je tu [2].

Python

Jazyk Python je spolu s JavaScriptom najpopulárnejší programovací jazyk na svete. Takisto o ňom existuje mnoho materiálov a kníh, no keďže nieje použitý v tejto práci, tieto veci nieje potrebné riešiť podrobne. Jazyk Python je veľmi obľúbený mnohými programátormi hlavne kvôli jeho prívetivej a intuitívnej syntaxi. Toto je jeho výhoda aj v oblasti web scrapingu. Vytvoriť web scraper v Pythone, napríklad pomocou knižnice Selenium, ktorá bude neskôr spomenutá, je značne prehľadnejšie a pochopiteľnejšie ako pri použití JavaScriptu. Jednou z veľkých nevýhod Pythonu je ale jeho pomalosť a veľká náročnosť na pamäť. Toto sú dôsledky práve jeho zamerania na jednoduchosť a prívetivosť kódu.

Prečo práve JavaScript

Prečo bol teda zvolený práve jazyk JavaScript? Tak ako bude v nasledujúcej podkapitole podrobne popísané, je to jazyk, ktorý bol už od začiatku vyvýjaný na prácu s webom. Má teda v sebe zabudované veľké množstvo možností na prácu s webovými štruktúrami a dokumentami. Toto je jeho hlavná výhoda oproti jazykom ako Java, C++ alebo Python, v ktorých by táto práca mohla byť implementovaná, no práve práca s webom by v nich bola zložitejšia.

3.2 JavaScript a jeho súčasti

Jazyk JavaScript bol pôvodne vyvinutý ako skriptovací jazyk, ktorý dokážu interpretovať webové prehliadače a teda umožňuje programovať funkcionalitu webových stránok. Je to teda veľmi podstatná šúčasť webu popri HTML a CSS. Takmer vždy, keď sa na webovej stránke niečo pohne, aktualizuje alebo zmení, je to zásluha práve JavaScriptu.

V posledných rokoch sa ale JavaScript veľmi rozšíril. Boli vytvorené prostredia mimo webových prehliadačov, ktoré JavaScript interpretujú a umožňujú v ňom vytvárať obyčajné serverové a desktopové aplikácie. Z týcho prostredí je najpopulárnejšie NodeJS, ktoré bude bližšie opísané v nasledujúcej podkapitole. Dá sa teda povedať, že JavaScript je na pomedzí skriptovacieho a programovacieho jazyka.

Podľa definície je to vysokoúrovňový, interpretovaný alebo just-in-time kompilovaný programovací jazyk. Podporuje objektovo-orientované, imperatívne aj deklaratívne programovanie. Riadi sa štandardom ECMAScript. [9]

NodeJS

NodeJS je behové prostredie pre JavaScript, založené na V8 interprete, ktorý je jadrom prehliadača Google Chrome. Je to multiplatformný open-source projekt, ktorý v posledných rokoch výrazne nabral na popularite. Umožňuje vývojárom vytvárať ako serverové, tak aj mobilné či desktopové aplikácie v jazyku JavaScript.

Node aplikácia beží ako jediný proces, bez vytvárania ďalších vlákien. NodeJS poskytuje vo svojej štandardnej knižnici prostriedky, ktoré zabraňujú blokovaniu kódu. Väčšina Node knižníc je vytvorená za použitia ne-blokovacích paradigiem. Napríklad ak NodeJS vykonáva zápis alebo čítanie zo siete, databázy alebo disku, tieto operácie neblokujú procesor a nečakajú, kým sa čítanie alebo zápis dokončí. Namiesto toho prenechajú procesor a sú obnovené až potom, ako bola operácia dokončená.

Veľkou výhodou pre Node je aj to, že veľké množstvo webových vývojárov už ovláda JavaScript a títo vývojári môžu začať tvoriť serverové a desktopové aplikácie bez nutnosti učiť sa úplne nový programovací jazyk. [11]

NPM

NPM alebo Node Package Manager je, tak ako z názvu vyplýva, systém na správu balíčkov pre NodeJS. V podstate to je veľká databáza JavaScript knižníc vytvorených vývojármi z celého sveta. Je to najväčší zoznam softwaru na svete, obsahuje viac než milión balíčkov. Skladá sa z troch základných častí: [12]

- Webstránka[1]
- Rozhranie príkazového riadku (CLI)
- Databáza balíčkov

NPM používajú jednotlivci na sťahovanie a inštalovanie rôznych balíčkov a aplikácií, ale aj veľké firmy na zdieľanie svojich interných súborov. Umožňuje veľmi rýchlo a jednoducho nainštalovať balíček aj so všetkými jeho závislosťami a taktiež ho bez problémov aktualizovať. Alternatívy NPM sú napríklad *Yarn* alebo *pnpm*, no NPM je jednoznačne najviac používaný systém na správu JavaScript balíčkov.

JQuery

Jquery je open-source knižnica jazyka JavaScript. Poskytuje jednoduché rozhranie na prácu s webom, ako napríklad prechádzanie a manipuláciu s HTML dokumentmi, správu udalostí a podobne. Funguje takmer so všetkými prehliadačmi a bola použitá aj pri implementácií tejto práce.
Krátka ukážka možností tejto knižnice je predvedená v nasledujúcom príklade:

```
$('h1').innerText;
```

Takto jednoduchý kód nám stačí na nájdenie hlavného nadpisu stránky a získanie jeho textu.

```
$("[name='myInput']").value("asdf");
```

Tento kód nájde HTML element, ktorého meno je “myInput” a zadá doň hodnotu “asdf”. Samozrejme musíme najskôr vedieť, že pole očakávajúce vstup má nastavený tento atribút name a budeme ho môcť podľa neho nájsť. V príklade je vidieť aj charakteristickú funkciu *$*, čo je len skratka pre funkciu s názvom *JQuery*. Táto funkcia je veľmi často používaná v mnohých rôznych aplikáciach.

Tento príklad neukazuje ani zďaleka celú funkcionalitu tejto knižnice, slúži len ako krátka demonštrácia jej jednoduchosti a sily. JQuery poskytuje omnoho viac možností a rozhraní.

[1] NPM webstránka - www.npmjs.com

3.3 Knižnice na tvorbu web scraperov

Tak ako na každú oblasť informatiky, aj na web scraping existujú knižnice pre najpoužívanejšie programovacie jazyky. V tejto podkapitole budú spomenuté dve z najrozšírenejších, no existuje ich veľké množstvo. Najskôr bude podrobnejšie opísaná knižnica Puppeteer, ktorá bola použitá pri implementácií praktickej časti tejto práce. Potom bude stručnejšie opísaná aj knižnica Selenium, ktorá je taktiež veľmi populárna a jej použitie bolo zvažované pri návrhu tejto práce.

Puppeteer

Puppeteer je knižnica vytvorená pre NodeJS, distribuovaná ako balíček cez NPM. Poskytuje vysokoúrovňové rozhranie, na ovládanie prehliadača Google Chrome[2] alebo Chromium[3] pomocou protokolu DevTools. Umožňuje automatizovať takmer všetky činnosti, ktoré sa dajú vo webovom prehliadači robiť manuálne. Napríklad umožňuje: [14]

- Automatizovať vypĺňanie formulárov, testovanie užívateľského rozhrania
- Vytváranie snímkov obrazovky webových stránok
- Vytvorenie automatizovaného testovacieho prostredia pre aplikácie
- Zachytávať sieťovú aktivitu stránky a ukladať jej históriu

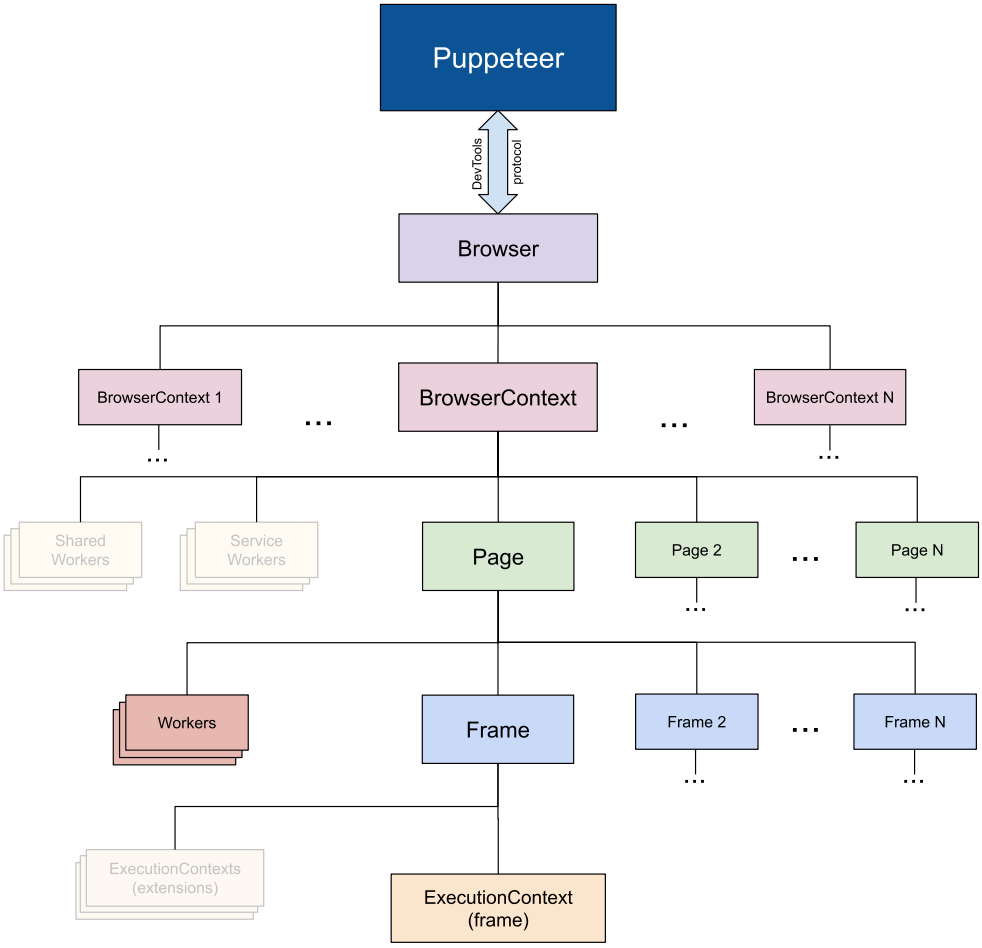
Puppeteer používa pri spustení prehliadača takzvaný “headless” mód, čo znamená, že nieje spustené žiadne grafické rozhranie prehliadača. Takýto režim prehliadača poskytuje plnú funkcionalitu a urýchluje samotný prehliadač a systém na ktorom beží.

Ako už bolo spomenuté, Puppeteer ovláda prehliadač pomocou protokolu Chrome DevTools. Tento protokol umožňuje automatizovať prácu s prehliadačmi Chrome, Chromium a ďalšími prehliadačmi založenými na prostredí Blink.

Táto knižnica, tak ako aj mnohé iné, nebola vyvinutá primárne pre web scraping. Hlavnou motiváciou automatizovania prehliadačov je testovanie užívateľského rozhrania aplikácií. No keďže tieto knižnice poskytujú takmer plnú funkcionalitu prehliadačov, sú veľmi vhodné aj na web scraping.

Architektúru knižnice Puppeteer podrobnejšie popisuje nasledujúci diagram. 3.1

[2]Prehliadač Google Chrome - www.google.com/chrome
[3]Prehliadač Chromium - www.chromium.org/Home



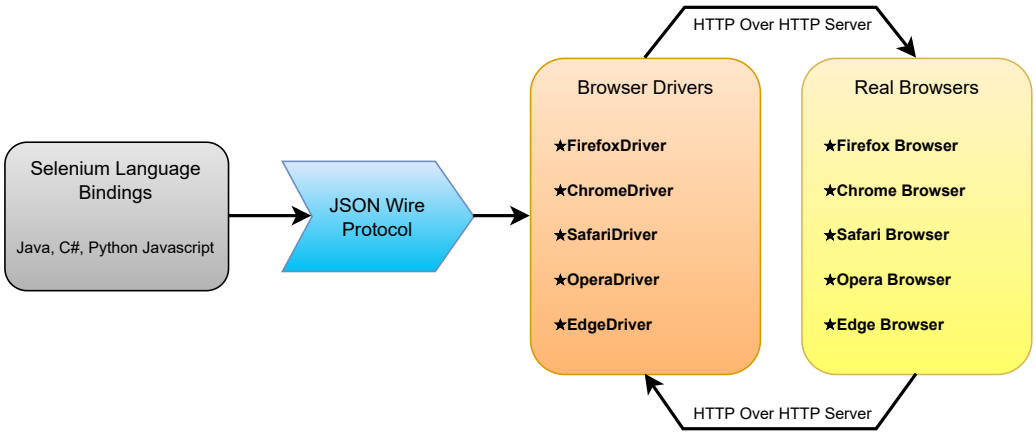
Obr. 3.1: Architektúra knižnice Puppeteer, prevzaté z [14]

Knižnica Puppeteer bola po zvážení ostatných možností použitá pri implementácií praktickej časti tejto práce. Najmä jej jednoduché a intuitívne použitie spojené so všetkými funkciami popísanými v tejto kapitole ju oddelili od ostatných alternatív. Jedna z ďalších zvažovaných alternatív, knižnica Selenium bude stručne opísaná v nasledujúcej podkapitole. Taktiež bude zdôvodnené, prečo nebola vybraná práve ona.

Selenium

Selenium WebDriver, tak isto ako Puppeteer, je knižnica vyvinutá pôvodne na automatizovanie testov užívateľských rozhraní aplikácií. Selenium bolo vyvinuté podstatne skôr a je oveľa populárnejšie ako Puppeteer. Selenium WebDriver je nasledovník systému Selenium RC, ktorý už nieje podporovaný. [4]

Prijíma príkazy pomocou JSON-Wire protokolu a predáva ich do prehliadača spusteného špecifickým ovládačom. Každá trieda prehliadačov má vlastný ovládač. Táto architektúra je znázornená na nasledujúcom diagrame. 3.2



Obr. 3.2: Architektúra Selenium WebDriver, prevzaté z www.javatpoint.com/selenium-webdriver

Ako je vidieť aj z diagramu 3.2, Selenium podporuje oveľa viac prehliadačov a rôznych programovacích jazykov než Puppeteer. Toto je jedna z najväčších výhod Selenia a taktiež dôvod, prečo je to najznámejšia knižnica v tejto oblasti. Tieto výhody sú ale výhodami iba v oblasti testovania. Pri web scrapingu nás väčšinou nezaujíma, akým prehliadačom dáta prezeráme. Preto tieto výhody nemali vyplyv na finálne rozhodnutie pri návrhu tejto práce a bola zvolená knižnica Puppeteer.

3.4 Existujúce riešenia pre web scraping

Existujúcich scraperov je príliš mnoho na to, aby sme sa mohli pokúsiť všetky vymenovať. Na internete sa ich dajú nájsť stovky. Pokrývajú celú škálu od voľnočasových projektov vytvorených jednotlivcami až po obrovské produkty vyvíjané a udržiavané korporáciami. Zoznam niekoľkých najznámejších aj s popisom ich výhod sa nachádza tu [3].

V rámci tejto práce je vhodné rozdeliť tieto produkty z hľadiska cieľovej skupiny na tri kategórie:

- Riešenia zamerané na veľké firmy a veľké objemy dát
- Všeobecnejšie riešenia, ktoré môžu využiť aj jednotlivci aj veľké firmy
- Menšie riešenia pre jednotlivcov

Pre každú z týchto kategórií bude uvedený príklad reálneho produktu a jeho výhody a nevýhody. Tiež bude medzi tieto kategórie zaradený aj produkt vyvinutý v rámci tejto práce.

Bright Data

Spoločnosť Bright Data[4], ako jedna z mnohých, ponúka riešenia tvorené na mieru pre veľké firmy a spoločnosti. Nevenuje sa iba web scrapingu, ten je len súčasťou ich riešení. Firmám ponúkajú služby ako marketing produktov, prieskumy trhu, testovanie a podobne.

Dalo by sa povedať, že toto je úplne iná sféra informatiky než web scraping jednotlivca pre vlastné použitie. Preto sa takmer nedá porovnať s riešením vyvinutým v rámci tejto práce. Pre obrovské firmy, ktoré potrebujú robiť prieskum trhu, alebo zberať veľké množstvá akýchkoľvek dát, sú riešenia ako Bright Data ideálnou voľbou.

Apify

Všeobecné riešenia pre jednotlivcov aj firmy dobre reprezentuje česká firma Apify[5]. Tiež ponúka komplexné riešenia pre veľké firmy, no okrem toho aj vytvorené scrapery pre často používané stránky a najznámejšie sociálne siete. Takisto umožňuje vytvoriť si vlastný scraper pomocou predpripraveného riešenia, bez nutnosti inštalácie akýchkoľvek prostredí a knižníc.

Táto firma, znova treba spomenúť, že aj mnohé iné, ponúka veľmi široké spektrum možností a takmer každý, kto potrebuje automatizovať nejakú činnosť na webe dokáže u nej nájsť vhodné riešenia pre svoje potreby.

V čom teda spočívajú výhody a nevýhody ich riešení? Zamerajme sa hlavne na riešenia smerované k jednotlivcom, pretože komplexné riešenia pre firmy už boli popísané vyššie. Jednotlivec si buď môže vytvoriť úplne nový vlastný scraper, alebo použiť jedno z ponúkaných existujúcich riešení.

Pri existujúcich riešeniach sú výhody celkom jasné, ušetria mnoho času, dokáže ich použiť aj osoba, ktorá nieje skúsená v informatike. Stačí len nejakým spôsobom scraper nakonfigurovať a všetko funguje. Nevýhody tohto prístupu sú ale tiež očividné. Sme limitovaní existujúcim riešením, do ktorého nesmieme zasahovať a teda je obmedzovaná jeho škálovateľnosť. Napríklad, ak zbierame dáta z jedného e-shopu a rozhodneme sa, že chceme aplikáciu rozšíriť aby zberala dáta aj z nejakého ďalšieho a ceny medzi sebou porovnávala, sme bez šance. Scraper, ktorý sme dostali je vytvorený špecificky pre jedno použitie.

Práve v takýchto prípadoch sa skôr hodí využiť možnosť vytvorenia vlastného scraperu. Táto možnosť je veľmi podobná riešeniu vytvorenému ako praktická časť tejto práce. Jeho hlavné výhody sú možnosť extportovať výsledné dáta do veľa rôznych formátov ako JSON, Excel, HTML, CSV a ďalšie. Ďalšia zaujímavá vlastnosť riešenia od Apify je aj to, že samotné scrapovanie prebieha na ich serveroch a dáta sú tiež uložené na ich úložiskách. Toto môže byť veľká výhoda, kvôli lepšiemu výkonu a nezaťažovaniu vlastného systému, ale v niektorých prípadoch je to nevýhoda a celú aplikáciu by sme chceli mať funkčnú lokálne. Taktiež je nutné spomenúť, že táto služba je spoplatnená na základe objemu získaných dát.

ParseHub

Nástroj ParseHub[6] reprezentuje skupinu scraperov, ktoré sú vyvinuté práve pre čo najjednoduchšie použitie pre hocikoho, kto chce automatizovať získavanie dát z webu. Ponúka veľmi prívetivé používateľské rozhranie, kde si používateľ interaktívne vykliká, ktoré dáta zo stránky ho zaujímajú a všetko ostatné za neho vyrieši aplikácia.

[4] Bright Data - https://brightdata.com
[5] Apify - https://apify.com
[6] ParseHub - www.parsehub.com

Nástroje tohto typu takmer presne spĺňajú popis technickej časti tejto práce. Ponúka sa teda otázka, prečo vlastne bola táto práca tvorená, keď existujú riešenia ako ParseHub, ktoré majú na prvý pohľad presne ten istý účel. Jedným z dôvodov, prečo by nám tieto riešenia nemuseli vyhovovať je to, že scraping je závislý na danej aplikácií, v prípade ParseHub desktopovej. Ak by bol náš cieľ vytvoriť serverovú aplikáciu na scraping a agregáciu dát, tieto riešenia by nám nepomohli.

Hlavným rozdielom je ale to, že táto práca sa zaoberá nielen samotným scrapingom, ale aj následným zobrazovaním získaných dát. V tomto ohľade môže v určitých prípadoch lepšie napĺňať očakávania užívateľa. Napríklad, ak si chceme pohodlne prezerať výsledky športových zápasov, dáta nechceme mať v textovom súbore ale v nejakom vizuálne príve-tivejšom formáte. Práve toto sa snažila dosiahnuť táto práca a je to jej hlavný rozdiel od bežných web scraping produktov.

Kapitola 4

Návrh riešenia

Úlohou tejto kapitoly je podrobný opis návrhu architektúry finálneho riešenia praktickej časti tejto bakalárskej práce a postupu, ktorým bol návrh dosiahnutý. Najskôr boli špecifikované požiadavky na funkcionalitu aplikácie, následne podrobne definované vstupy a žiadané výstupy. Na základe týchto informácií následne vznikol finálny návrh architektúry produktu.

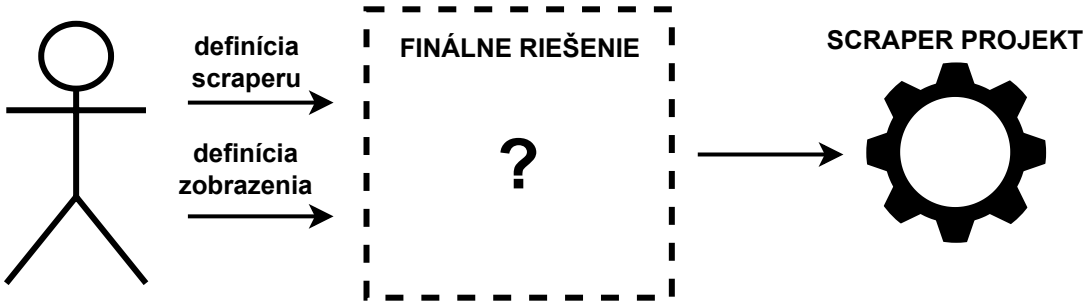
4.1 Špecifikácia požiadaviek

Cieľom tejto práce bolo vytvoriť nástroj pre automatizované získavanie definovaných informácií z väčšieho množstva zdrojov. Nástroj mal teda byť schopný nechať užívateľa nejakým spôsobom definovať, ktoré informácie z ktorých stránok ho zaujímajú a následne tieto informácie automaticky extrahovať. Dôležitou súčasťou finálneho nástroja je aj zobrazenie extrahovaných dát tak, aby boli prehľadné a vizuálne upravené. Keďže ale nevieme, o aké dáta sa bude jednať, toto zobrazenie musí byť taktiež definované užívateľom. Je to síce práca naviac pre užívateľa, no na druhú stranu mu to umožní si zobrazenie dát prispôsobiť vlastným potrebám.

Výsledný nástroj teda musí poskytovať užívateľovi veľký priestor, ako pri definícií dát, ktoré potrebuje, tak aj pri zobrazení daných dát. Táto požiadavka bola braná do úvahy už od počiatočných fází návrhu.

Keďže definícia získavania dát a aj ich zobrazovania je v roli užívateľa, hlavnou úlohou výsledného nástroja musia byť práve všetky činnosti na pozadí, teda scraping definovaných dát, ich uloženie, umožnenie ich zobrazenia a ovládanie jednotlivých modulov aplikácie, aby fungovala ako celok. Vyvinutý nástroj teda musí byť schopný po priajtí vstupov od užívateľa vytvoriť určitý samostatne funkčný celok, ktorý môžeme nazvať scraper projekt. Samozrejme môže takýchto projektov vytvoriť neobmedzené množstvo, kde každý z nich môže fungovať nezávisle na ostatných.

Podľa týchto špecifikácií bol vytvorený prvotný návrh architektúry aplikácie. Tento návrh je zobrazený v nasledujúcom diagrame 4.1.



Obr. 4.1: Prvotný návrh architektúry

Finálne riešenie teda musí prijať užívateľské vstupy a vytvoriť pomocou nich funkčný celok, ktorý dokáže samostatne extrahovať dáta z internetu a následne ich zobrazovať.

4.2 Vstupy a výstupy

Po všeobecnej špecifikácií funkcionality a architektúry riešenia bolo potrebné detailne špecifikovať očakávané vstupy a výstupy aplikácie. Tak ako pri každom produkte v informatike, aj v tejto práci bol tento krok veľmi dôležitý a bol naň kladený dôraz. Vstupy aj výstupy výsledného produktu sú už naznačené v diagrame prvotného návrhu 4.1, no je potrebné ich opísať podrobnejšie. Ich finálny popis a taktiež aj proces, akým boli navrhnuté, bude obsahom tejto podkapitoly.

Vstupy

Na začiatku návrhu akejkoľvek aplikácie je nutné presne definovať vstupy. Vstupy do aplikácie vyvinutej v rámci tejto práce sa dajú rozdeliť na dve skupiny:

- Vstupy definujúce scraping
- Vstupy definujúce zobrazenie dát

Dalo by sa povedať, že vstupy definujúce scraping sú dôležitejšie, pretože získanie a uloženie získaných dát je dôležitejšia časť výslednej aplikácie, než ich zobrazenie. V niektorých situáciach nám môže stačiť získať dáta a uložiť ich v štruktúrovanej podobe kvôli následnému ďalšiemu strojovému spracovaniu týchto dát. Aj tento fakt bol pri návrhu braný do úvahy. Vstupy definujúce scraping sú preto vyžadované a aplikácia bez nich nebude fungovať. Vstupy definujúce zobrazenie sú voliteľné a aj ak nebudú zadané, aplikácia dokáže dáta získať a uložiť, prípadne ich aj zobraziť v jednoduchej neformátovanej podobe.

Vstupy definujúce scraping zahŕňajú definíciu webstránok, z ktorých majú byť dáta získavané a následne pre každú stránku definícia samotného získania žiadaných dát. Definícia webstránok je veľmi jednoduchá, stačí aby užívateľ v akejkoľvek forme zadal URL[1] danej webstránky alebo webstránok.

[1] URL - Uniform Resource Locator

Definovať získavanie žiadaných dát je tá najzložitejšia časť, ktorú bude musieť užívateľ vykonať. Ako bolo v predošlých kapitolách spomínané, web scraping je možné vykonávať pomocou mnohých rôznych technológií a nástrojov. Dôvody, prečo bol na implementáciu tejto práce vybraný jazyk JavaScript už boli popísané v kapitole 3. Bolo teda rozhodnuté, že užívateľ bude musieť vytvoriť funkciu v jazyku JavaScript, ktorá nájde chcené dáta z webstránky a vráti ich v štruktúrovanej forme. Táto funkcia bude musieť nejakým spôsobom, napríklad cez parameter, dostať referenciu na danú stránku, aby s ňou mohla zaobchádzať. Ďalej už bude táto funkcia nazývaná "scraper funkcia".

Toto rozhodnutie samozrejme znamená, že pre človeka neskúseného v informatike bude použitie tejto aplikácie značne sťažené. Kvôli tomuto problému boli v rámci tejto práce vyvinuté aj tri reálne príklady použitia a manuál, ktorých účelom je okrem iného aj vysvetliť proces, ako vytvoriť takúto funkciu.

Jedným zo spôsobov, ako môžeme tento proces užívateľovi uľahčiť je vygenerovať kostru tejto scraper funkcie. Takto sa nebude musieť zaoberať zbytočnosťami ako správne definovanie vstupov a výstupov funkcie a podobne.

Výstupy

Ako bolo naznačené v diagrame 4.1, výstup aplikácie by mal byť funkčný scraper projekt. Samostatne fungujúci celok, ktorý dokáže získavať dáta zo zadaných webstránok a zobrazovať ich užívateľovi. Definovať tento výstup podrobnejšie bolo na začiatku návrhu veľmi zložité, no postupne pomocou experimentovania s jednotlivými technológiami bol vytvorený lepší popis tohto výstupu.

V skutočnosti nieje potrebné, aby to bol nejaký komplexný program alebo aplikácia. Stačí aby bol vygenerovaný adresár obsahujúci všetky potrebné súbory. Niektoré z týchto súborov boli definované už počas návrhu. Najdôležitejšie časti, ktoré boli známe už počas návrhu sú tieto:

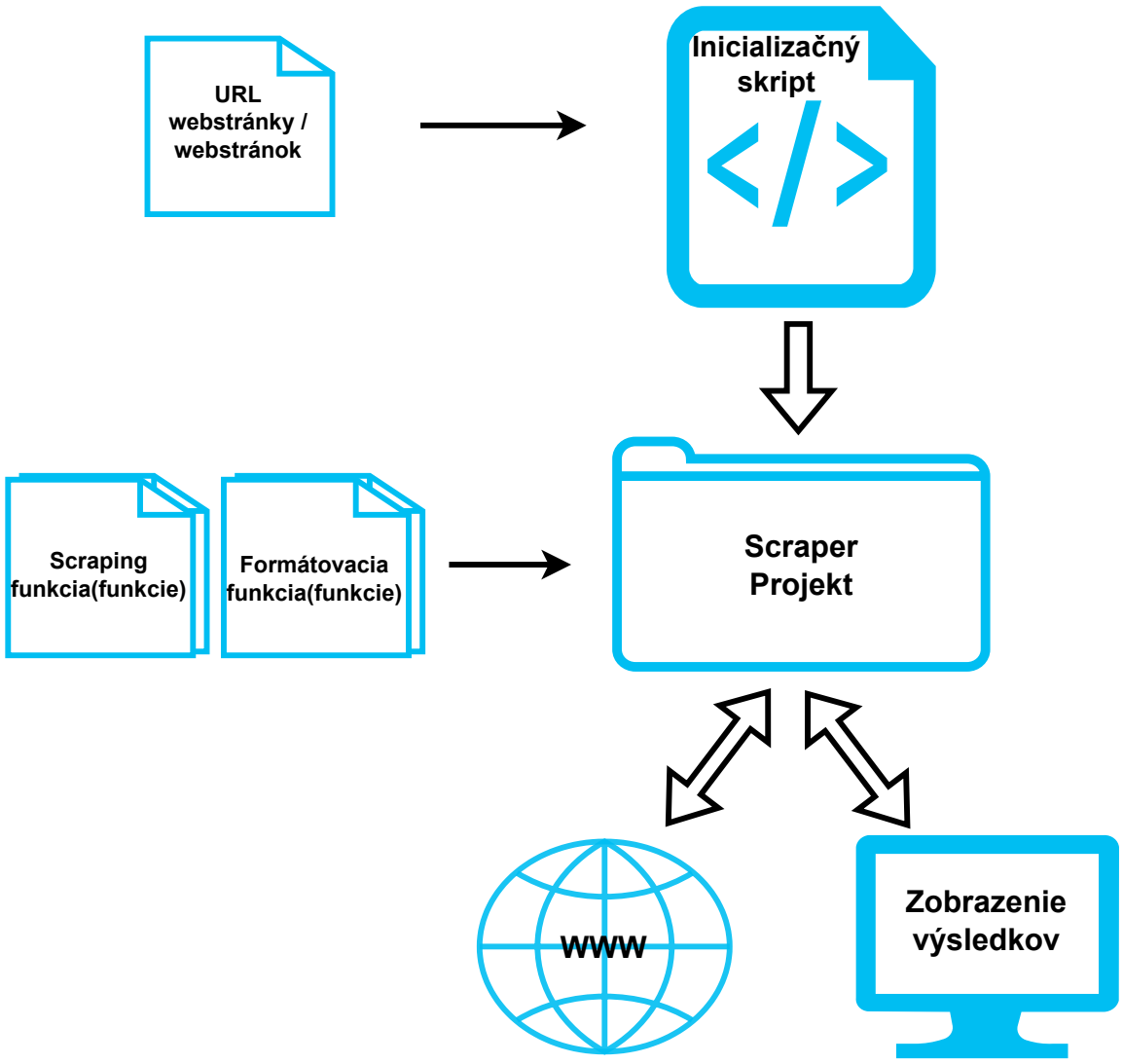
- Hlavný skript, ktorý riadi všetky súčasti
- Skript, ktorý vykonáva scraping
- Súbor index.html, obsahujúci naformátované výsledky
- Podadresár na uloženie všetkých dát z jednotlivých stránok

Samozrjeme bolo pri návrhu očakávané, že počas implementácie sa budú môcť niektoré časti pozmeniť, prípadne pribudnú nejaké nové pomocné súbory a podobne. Tento návrh popisoval iba najdôležitejšie súčasti, ktoré bude musieť výstup výsledného riešenia v nejakej forme obsahovať.

4.3 Architektúra riešenia

Po podrobnom špecifikovaní všetkých požiadaviek na výsledný produkt a experimentovaní s jednotlivými technológiami bol vytvorený finálny návrh architektúry riešenia. Jeho jednotlivé časti neboli definované úplne presne, bola určená len ich funkcionalita. Napríklad nebolo ešte známe, v akom formáte budú uložené dáta alebo akým spôsobom bude užívateľ zadávať svoje vstupy. Tieto záležitosti boli riešené až pri samotnej implementácií, ktorá bude opísaná v nasledujúcej kapitole.

Finálny návrh architektúry riešenia je znázornený nasledujúcim diagramom 4.2:



Obr. 4.2: Finálny návrh architektúry

Užívateľské vstupy sú znázornené malými vyplnenými šípkami. Ako môžme vidieť, najskôr užívateľ zadá do inicializačného skriptu URL všetkých webstránok, z ktorých chce dáta získavať. Tento krok je potrebné urobiť ako prvý, pretože ako bolo popísané vyššie v sekcii o vstupoch, užívateľ bude musieť následne vytvoriť scraping funkciu pre každú zo zadaných stránok. Kostry týchto funkcií chceme užívateľovi vygenerovať, takže potrebujeme vedieť ich počet na začiatku.

Inicializačný skript následne vygeneruje scraper projekt, ktorý ale ešte nie je funkčný. Užívateľ musí vytvoriť spomínané scraper funkcie pre každú zadanú webstránku. Po tom, ako toto užívateľ vykoná, má k dispozícii funkčný web scraper, ktorý dokáže získavať dáta z webu, ukladať ich a zobraziť ich v jednoduchej nenaformátovanej podobe.

Následne môže užívateľ definovať formátovacie funkcie pre dáta z jednotlivých webstránok. Tieto funkcie budú taktiež vygenerované inicializačným skriptom a ich výstupom bude HTML dokument, ktorý môže v sebe obsahovať aj CSS kód.

Ďalšie veci, ktoré boli pri návrhu špecifikované boli spôsob ukladania dát, frekvencia obnovovania dát a možnosť manuálneho obnovenia.

Keďže náplňou tejto práce je kombinácia web scrapingu a zobrazovania dát, nie je nutné ukladať získané dáta z minulosti. Vždy budú uložené len dáta z posledného behu skriptu riadiaceho scraping. Nebude teda možné udržiavať históriu daných dát a vytvárať z nich štatistiky alebo analýzy. Tento nástroj bude zameraný čisto na dáta, ktoré sú premenlivé a zaujíma nás len ich hodnota v aktuálnej chvíli.

Ďalšou dôležitou špecifikáciou bola frekvencia, ako často budú tieto dáta obnovované a teda, ako často bude spúšťaný scraping skript. Táto frekvencia bola určená na jedno obnovenie každú hodinu. Niekedy nás ale môžu zaujímať zmeny dát práve v danom momente a čakať hodinu na ďalšie obnovenie by bolo príliš dlho. Pre takéto prípady bola navrhnutá možnosť manuálne vynútiť toto obnovenie hocikedy užívateľ chce. Implementácia tejto funkcionality bude takisto opísaná v nasledujúcej kapitole.

Kapitola 5

Implementácia

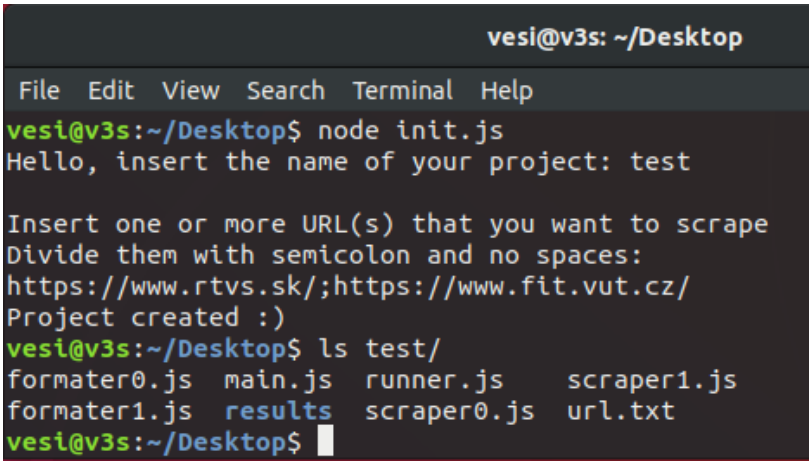
V tejto kapitole bude podrobne opísané, ako boli implementované jednotlivé časti finálneho riešenia a ako na seba nadväzujú. Implementácia riešenia sa v značnej miere prelínala s návrhom riešenia. Pri vývoji bol využívaný koncept prototypovania. Vždy, keď bola špecifikovaná nejaká časť návrhu, bol vytvorený prototyp danej časti a prípadne bol návrh upravený podľa novo zistených informácií. Tento postup umožnil veľmi dynamický a rýchly vývoj finálneho produktu, pretože po úplnom dokončení návrhu stačilo vytvorený prototyp len mierne doladiť, opraviť malé nedostatky a produkt bol dokončený.

5.1 Inicializačný skript

Inicializačný skript *init.js* je hlavná časť výsledku tejto práce. Je to skript, ktorý dokáže vytvoriť neobmedzené množstvo samostatne funkčných scraper projektov. Potrebuje na to od užívateľa prijať dva vstupy, názov scraper projektu a zoznam URL všetkých webstránok, z ktorých budú získavané dáta.

Názov projektu neznamená nič iné, než názov adresára, ktorý bude vygenerovaný a ktorý bude obsahovať vytvorený scraper projekt. Samozrejme je v skripte zahrnutá aj jednoduchá kontrola, či už adresár s rovnakým názvom v aktuálnom adresári neexistuje, aby nedochádzalo k zbytočným chybám alebo zničeniu funkčného projektu tým, že vygenerujeme nový s rovnakým názvom.

Oba tieto vstupy skript od užívateľa prijíma cez jednoduché textové rozhranie v príkazovom riadku potom, ako je spustený. Jazyk JavaScript ponúka funkcie v triede *readline.createInterface()*, pomocou ktorých sa táto funkcionalita dá veľmi jednoducho dosiahnuť. Ukážka použitia tohto skriptu je na nasledujúcom obrázku 5.1.



Obr. 5.1: Screenshot zobrazujúci funkcionalitu inicializačného skriptu

Ako je z obrázku vidieť, skript vygeneroval adresár obsahujúci niekoľko JavaScript-ových súborov, jeden pomocný súbor a podadresár na ukladanie výsledných dát. Na generovanie týchto súborov boli v inicializačnom skripte použité takzvané "multiline stringy" - viacriadkové reťazce. Je to dátový typ veľmi podobný obyčajnému reťazcu, no je možné ho rozpísať na viacero riadkov a niektoré špeciálne znaky majú vnútri neho iné chovanie. Ak teda potrebujeme vygenerovať súbor, ktorý obsahuje dlhší kód, v našom prípade skript, je to ideálna voľba oproti použitiu obyčajných reťazcov. Takýmto spôsobom sa dá vygenerovať kód so správnym odsadením a odriadkovaním, ktorý by sa síce dal vygenerovať aj pomocou obyčajných reťazcov, no bolo by to značne zložitejšie a neprehľadnejšie.

Pomocou týchto špeciálnych reťazcov sú generované všetky súbory s príponou ".js". Pre zvýšenie prehľadnosti kódu v skripte *init.js* boli tieto reťazce vybrané a umiestnené do samostatného súboru *strings.js*. Tento súbor musí byť umiestnený v rovnakom adresári ako inicializačný skript, inak sa inicializačný skript nespustí.

5.2 Výsledný scraper projekt

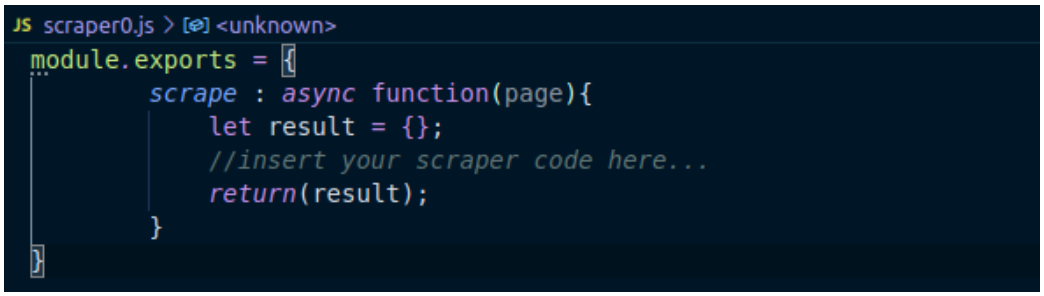
Inicializačný skript vygeneruje výsledný scraper projekt. Tento projekt je ale komplexný celok, ktorý je potrebné rozdeliť na jednotlivé časti a každú z nich popísať, aby sme mohli pochopiť ako funguje.

Scraper funkcie

Scraper funkcie, ktoré už boli spomenuté v predchádzajúcej kapitole, sú veľmi dôležitou súčasťou výsledného scraper projektu. Inicializačný skript vygeneruje ich kostru, ktorá musí byť následne vyplnená užívateľom, aby mohol projekt fungovať. Úlohou týchto funkcií je presne definovať, ktoré dáta jednotlivých webstránok chceme extrahovať a uložiť.

Pre každú zadanú webstránku je vygenerovaná práve jedna kostra scraper funkcie. Tieto funkcie sú v samostatných súboroch pomenovaných *scraper0.js*, *scraper1.js*, ... a ich poradie je dôležité, pretože sú pridelené k jednotlivým webstránkam v takom poradí, v akom boli URL adresy webstránok zadané pri behu inicializačného skriptu. Aby si užívateľ toto poradie nemusel pamätať, je uložené v pomocnom súbore *url.txt*, ktorý sa nachádza v scraper projekte.

Vygenerovaná kostra takejto scraper funkcie je zobrazená na nasledujúcom obrázku: 5.2



Obr. 5.2: Screenshot zobrazujúci vygenerovanú kostru scraper funkcie

Z obrázku kostry scraper funkcie 5.2 sa môže na prvý pohľad zdať, že je to len obyčajná prázdna funkcia a samotný obrázok je zbytočný. Toto ale nieje pravda, pretože z obrázku sú vidieť dve dôležité veci. Vstup a výstup scraper funkcie.

Vstup je objekt s názvom page, čo nieje náhoda, ale je to skutočne referencia na objekt triedy *page*, ktorý poskytuje rozhranie na prácu s jednou kartou webového prehliadača. Podrobnosti o tejto triede a všetky jej funckie sú podrobne opísané v oficiálnej dokumentácií knižnice Puppeteer [14].

Je zabezpečené, že v tejto karte bola otvorená príslušná webstránka pre danú scraper funkciu. To znamená, že pomocou objektu page máme prístup priamo ku žiadanej webstránke. Príklady vytvorených scraper funkcií za použitia rozhrania *page* budú opísané v kapitole o testovaní 6.

Výstup scraper funkcie je, ako môžme vidieť, obyčajný JavaScript objekt. Tento objekt môžme naplniť získanými dátami zo stránky a bude uložený ako výsledok scrapingu vo formáte JSON.

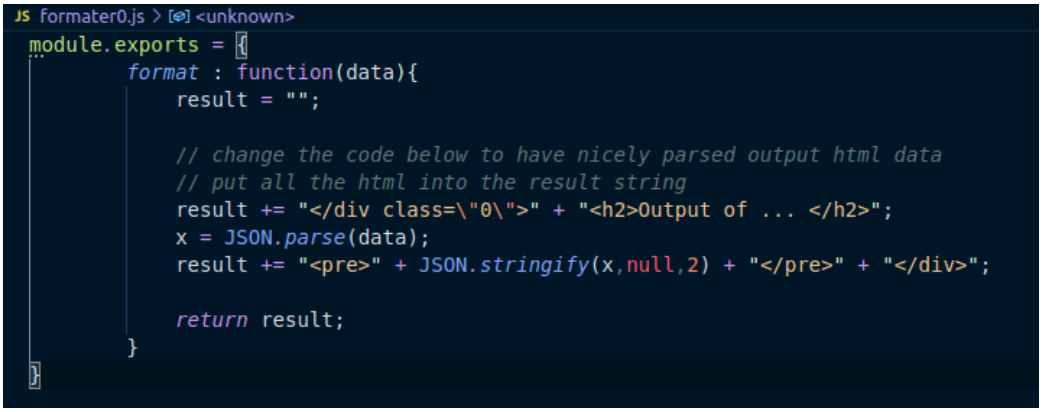
Formátovacie funkcie

Formátovacie funkcie sú takisto ako scraper funkcie vygenerované pre každú webstránku do samostatného očíslovaného súboru. Ich úlohou je vziať dáta uložené vo formáte JSON, ktoré sú výstupmi jednotlivých scraper funkcií a naformátovať ich tak, aby boli pre užívateľa prehľadné a vizuálne prívetivé.

Vstupom formátovacej funkcie sú teda dáta vo formáte JSON a jej výstupom je reťazec, obsahujúci HTML a prípadne CSS kód, ktorý ich má reprezentovať. Poskytuje teda užívateľovi takmer neobmedzené možnosti, naformátovať dáta tak, aby boli zobrazené presne ako mu najviac vyhovuje.

Výsledky všetkých formátovacích funkcií budú jeden po druhom priložené do výsledného súboru *index.html*, ktorý bude zobrazovaný užívateľovi.

Vygenerovaná kostra formátovacej funkcie je zobrazená na nasledujúcom obrázku: 5.3



Obr. 5.3: Screenshot zobrazujúci vygenerovanú kostru formátovacej funkcie

Ako môžme z obrázka 5.3 vidieť, táto funkcia nieje prázdna, tak ako scraper funkcia. To je práve preto, že vyplnenie formátovacej funkcie bolo navrhnuté ako voliteľné a scraper project funguje aj bez vykonania tohto kroku.

Kód, ktorý je na obrázku vo vnútri formátovacej funkcie vidno, robí iba to, že do výsledného HTML dokumentu budú vložené dáta v jednoduchej JSON podobe, bez akýchkoľvek úprav. Užívateľ môže tento kód prepísať, aby bol výstup podľa jeho predstáv.

Rôzne príklady formátovacích funkcií a taktiež aj situácie, kedy formátovacia funkcia nie je použitá, budú podrobne opísané v kapitole o testovaní 6.

Skript runner.js

Skript *runner.js* je ďalšou dôležitou súčasťou vygenerovaného projektu. Tentokrát je to ale skript, do ktorého užívateľ nemusí a nemal by nijak zasahovať. Je to skript, ktorý ovláda všetku prácu s knižnicou Puppeteer a teda celý web scraping.

Najskôr vytvorí inštanciu takzvaného "headless" prehliadača, ktorý už bol vysvetlený v kapitole o knižnici Puppeteer. V tomto prehliadači následne prechádza užívateľom zadané URL adresy webstránok jednu po druhej. Po otvorení každej počká, kým je stránka úplne načítaná a zavolá príslušnú scraper funkciu definovanú užívateľom. Tejto funkcií predá cez parameter referenciu na už spomínané rozhranie *page*.

Očakáva, že návratová hodnota každej zo scraper funkcií je JavaScript object, ktorý najskôr skonvertuje do formátu JSON pomocou vstavanej funkcie *JSON.stringify()* a následne ho uloží do súboru, tiež očíslovaného podľa poradia danej stránky.

Uložené dáta

Už bolo v tejto kapitole viackrát spomenuté, že dáta získané zo scraper funkcií sú uložené vo formáte JSON. Konkrétne sú uložené v podadresári *results/*. Tak isto ako scraper funkcie a formátovacie funkcie, dáta získané z každej stránky sú umiestnené v samostatnom očíslovanom súbore. Tieto súbory sú *result0.json*, *result1.json*, ... a môžu užívateľovi poskytnúť informáciu o tom, či jeho definovaná scraper funkcia funguje tak, ako očakáva.

Môže sa stať, že užívateľ bude mať vytvorenú scraper aj formátovaciu funkciu pre určitú webstránku, ale vo výslednom zobrazení neuvidí nič, prípadne uvidí dáta v inej forme, než očakával. Práve v takomto prípade je dobré vedieť kde a ako sú dáta uložené. Ak užívateľ nájde uložené dáta z danej webstránky a sú v takej podobe, ako očakával, pravdepodobne

je problém vo formátovacej funkcii. Naopak, ak dáta niesú v poriadku, formátovacia funkcia môže fungovať správne, ale chyba je určite v scraper funkcii.

5.3 Spustenie výsledného projektu

Výsledný scraper projekt sa spúšťa a je celý riadený hlavným skriptom *main.js*, ktorý je takisto vygenerovaný inicializačným skriptom vo vnútri projektového adresára.

Rovnako ako súbor *runner.js*, aj hlavný skript je už pripravený na používanie a užívateľ by doň nemal vôbec zasahovať.

Hlavný skript je ale komplexnejší ako *runner* skript, preto nebude opísaný úplne detailne. Bude ale vysvetlená jeho všeobecná funkcionalita a najdôležitejšie časti.

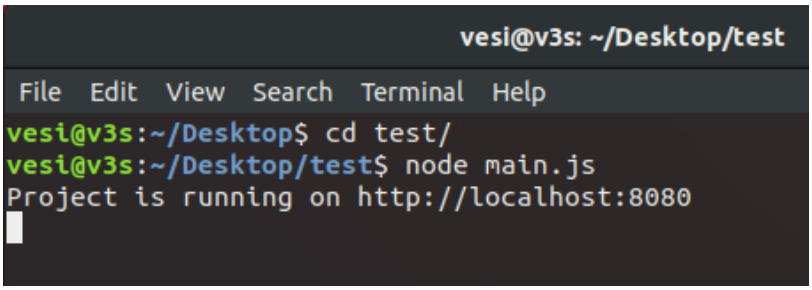
Funkcia scrape

Súčasťou tohto skriptu je dôležitá funkcia *scrape()*, ktorá v sebe zahŕňa volanie *runner* skriptu, následné zavolanie formátovacích funkcií na každý súbor s uloženými dátami a nakoniec vloženie týchto naformátovaných dát do výsledného súboru *index.html*. Táto funkcia teda získa nové dáta z webu a aktualizuje ich v zobrazení pre užívateľa.

Funkcia *scrape()* je pomocou vstavanej JavaScript funkcie *setInterval()* spúšťaná každú hodinu a teda počas behu hlavného skriptu budú každú hodinu obnovené dáta a aktualizované ich zobrazenie vo výslednom HTML súbore.

Zobrazenie dát užívateľovi

Hlavný skript scraper projektu spustíme, podobne ako inicializačný skript, cez príkazový riadok pomocou prostredia *node*. Spustenie projektu, ktorého inicializovanie bolo zobrazené na obrázku 5.1, je zobrazené na nasledujúcom obrázku 5.4:



Obr. 5.4: Screenshot zobrazujúci spustenie výsledného projektu

Ako môžme z obrázku 5.4 vidieť, hlavný skript po spustení iba vypíše hlášku, že beží a URL adresu, na ktorej môžme nájsť výsledok. Túto adresu môžme otvoriť v akomkoľvek webovom prehliadači a bude nám na nej prezentovaný výsledný HTML súbor obsahujúci dáta.

Táto funkcionalita je zabezpečená JavaScript balíčkom *http*, ktorý umožňuje vytvoriť jednoduchý HTTP[1] server. Umožňuje teda počúvať na definovanom porte a odpovedať na HTTP požiadavky odpoveďami, ktoré obsahujú náš výsledný súbor *index.html*. Taktiež umožňuje po prijatí rôznych typov požiadaviek vykonávať rôzny kód.

[1]HTTP - HyperText Transfer Protocol

Táto skutočnosť je využitá pri implementácií tlačítka, ktoré umožňuje užívateľovi vynútiť obnovenie dát bez toho, aby muselo nastať pravidelné obnovenie každú hodinu. Toto tlačítko pošle na HTTP server požiadavku typu POST, na ktorú server zareaguje tak, že najskôr zavolá funkciu *scrape()*, ktorou obnoví dáta a až potom vráti v odpovedi súbor *index.html*. Pri bežných HTTP požiadavkách, ktoré vyvolá napríklad otvorenie stránky alebo jej obyčajné obnovenie v prehliadači sa nedeje nič, iba je vrátená bežná odpoveď obsahujúca HTML súbor.

Toto tlačítko a skript, ktorý ovláda jeho funkcionalitu sú už zabudované v HTML kostre, do ktorej sú dáta vkladané. Táto kostra je taktiež súčasťou hlavného skriptu a používa už spomínaný špeciálny viacriadkový reťazec.

Kapitola 6

Testovanie

Táto kapitola opisuje priebeh a výsledky testovania nástroja, ktorý bol vyvinutý ako praktická časť tejto bakalárskej práce. V rámci testovania bola najskôr na jednoduchom príklade overená funkcionalita základných častí riešenia. Tento testovací príklad bude popísaný na začiatku tejto kapitoly.

Následne boli vyvinuté tri reálne príklady scraper projektov, ktoré získavajú a zobrazujú dáta z reálnych internetových stránok a každý z nich sa zameriava na iný problém. Tieto tri príklady budú takisto podrobne opísané v tejto kapitole, pretože ich úlohou nieje len podrobne otestovať funkčnosť riešenia, ale slúžia aj ako určitý návod pre budúceho užívateľa. Obsahujú názorné ukážky, ako môže vyzerať scraper a formátovacie funkcie pre reálne webstránky, ktoré sú bežne ľudmi používané a majú zložitú štruktúru.

Všetko testovanie prebiehalo na dvoch samostatných systémoch, pričom na jednom z nich je operačný systém Ubuntu 18.04 a na druhom Ubuntu 20.04. Na oboch bola nainštalovaná najnovšia verzia NodeJS, NPM a knižnice Puppeteer.

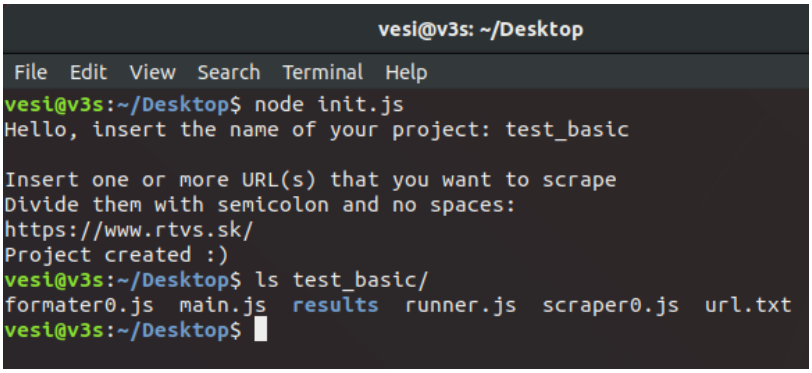
6.1 Test základnej funkcionality riešenia

Tento test mal za úlohu vytvoriť veľmi jednoduchý scraper projekt, ktorý získava dáta len z jednej webstránky a overiť základnú funkcionalitu riešenia. Teda či funguje inicializačný skript a či po vytvorení jednoduchej scraper funkcie dokáže projekt dáta automaticky získavať a zobrazovať. Taktiež bola otestovaná funkčnosť tlačítka, ktoré vynucuje obnovenie dát.

Úlohou tohto testu bude zobrazovať, aký program aktuálne vysielajú jednotlivé kanály slovenskej televízie RTVS[1].

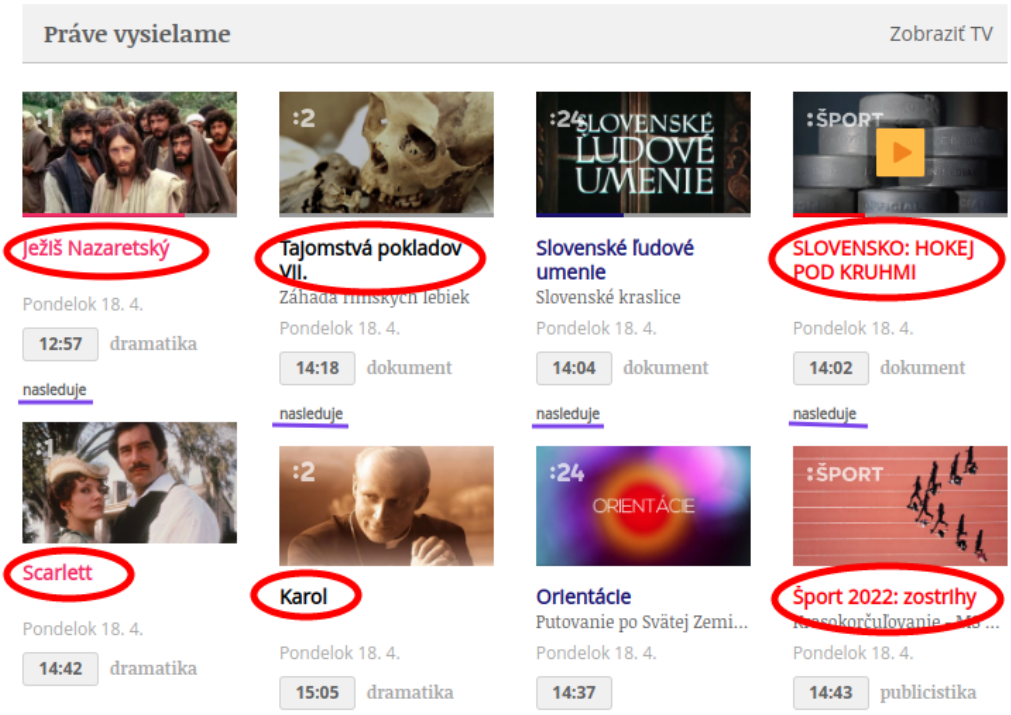
Najskôr je teda nutné projekt vytvoriť pomocou inicializačného skriptu. Tento proces bol už opísaný v predošlej kapitole, kde bol inicializačný skript vysvetľovaný. Nasledujúci obrázok ukazuje, ako toto vytvorenie prebiehalo: 6.1

[1] RTVS - www.rtvs.sk



Obr. 6.1: Screenshot zobrazujúci vytvorenie testovacieho projektu

Po vytvorení projektu a overení, či obsahuje potrebné súčasti príkazom *ls*, je potrebné vytvoriť scraper funkciu, ktorá chcené dáta z webstránky extrahuje. Povedzme, že nám bude stačiť vedieť aktuálny program a program, ktorý nasleduje. Ako môžme na nasledujúcom obrázku 6.2 vidieť, dáta ktoré chceme získať niesú blízko seba a môže sa zdať, že extrahovať ich bude ťažké.



Obr. 6.2: Screenshot zobrazujúci štruktúru webstránky a dáta, ktoré z nej chceme získať

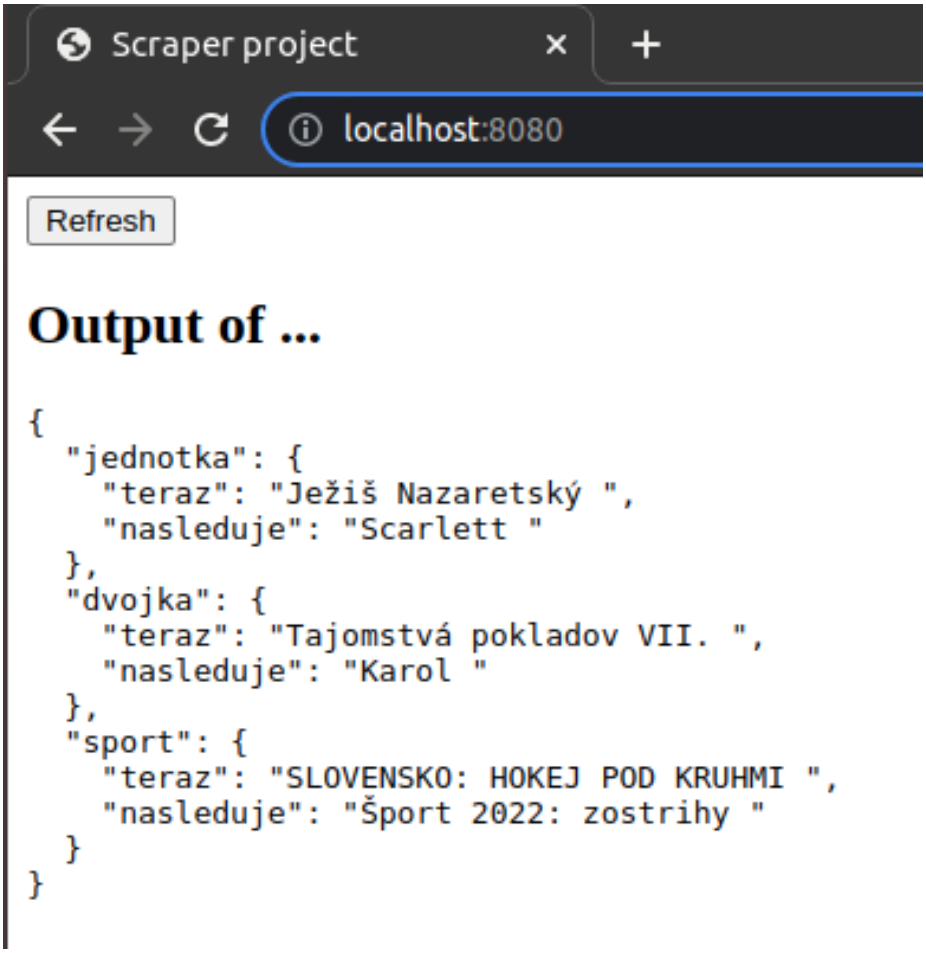
V skutočnosti je ale získanie týchto dát veľmi jednoduché. Po rýchlom prezretí dokumentu pomocou nástroja DevTools v prehliadači Chrome sa dá zistiť, že všetky dáta o programoch sa nachádzajú v jednej skupine s názvom “media___body” a názvy týchto programov, ktoré chceme získať sú HTML elementy “h6”. Teda pomocou rozhrania *page* a knižnice Jquery dokážeme získať práve tieto názvy. Celý kód použitej scraper funkcie vyzerá takto:

```
scrape : async function(page){
    let result = {};

    const results = await page.evaluate(() =>
           Array.from(document.querySelectorAll("div.media__body > h6"))
           .map( (res) => res.innerText ));
    result["jednotka"] = {}; result["dvojka"] = {}; result["sport"] = {};
    result["jednotka"]["teraz"] = results[0];
    result["jednotka"]["nasleduje"] = results[1];
    result["dvojka"]["teraz"] = results[2];
    result["dvojka"]["nasleduje"] = results[3];
    result["sport"]["teraz"] = results[6];
    result["sport"]["nasleduje"] = results[7];

    return(result);
}
```

Ako môžeme z ukážky kódu vidieť, tento scraper nebol vôbec zložitý. Všetko dôležité sa odohráva vrámci prvého príkazu, pomocou funkcie *page.evaluate()* a knižnice JQuery. Použitá konštrukcia môže byť na prvý pohľad neprehľadná, no keď ju prejdeme krok po kroku, nieje to nič svetoborné. Tento príkaz vyberie všetky elementy *h6* z danej skupiny vrámci stránky a vytvorí pole z ich textového obsahu. Toto pole je uložené do konštanty *results*, z ktorej sú následne už len do predpripraveného poľa *result* priradené hodnoty, ktoré nás zaujímajú.

Formátovacia funkcia v tomto príklade nebola použitá, pretože je hlavne zameraný na testovanie základnej funkcionality, teda či správne funguje získavanie a zobrazovanie dát. Taktiež vďaka tomu tento príklad testuje funkcionalitu projektu bez formátovacej funkcie, ktorá by nemala byť pre fungovanie projektu nutná. V ostatných testovacích príkladoch budú použité aj formátovacie funkcie a bude popísaný aj proces ich tvorby.

Keďže máme scraper funkciu hotovú a rozhodli sme, že formátovaciu funkciu používať nebudeme, môžme scraper projekt spustiť. Spustíme ho pomocou skriptu *main.js*, tak ako bolo ukázané na obrázku 5.4. Po spustení projektu môžme na adrese *localhost:8080* v ľubovoľnom webovom prehliadači vidieť výsledok zobrazený na nasledujúcom obrázku: 6.3.



Obr. 6.3: Screenshot zobrazujúci finálnu funkcionalitu testovacieho projektu

Ako môžme na obrázku 6.3 vidieť, výsledkom je jednoduchá stránka obsahujúca dáta získané scraper funkciou, bez ďalšieho formátovania. Keď necháme skript *main.js* bežať dlhšiu dobu, uvidíme, že každú hodinu sa dáta na stránke obnovia a keď hocikedy klikneme na tlačítko refresh tak sa po krátkej chvíli čakania taktiež obnovia.

Takýmto spôsobom bol tento testovací príklad otestovaný počas dvoch dní, kedy sa pravidelne stránka obnovovala a zobrazené výsledky boli porovnávané s reálnymi dátami na pôvodnej webstránke.

6.2 Testovacie scraper projekty

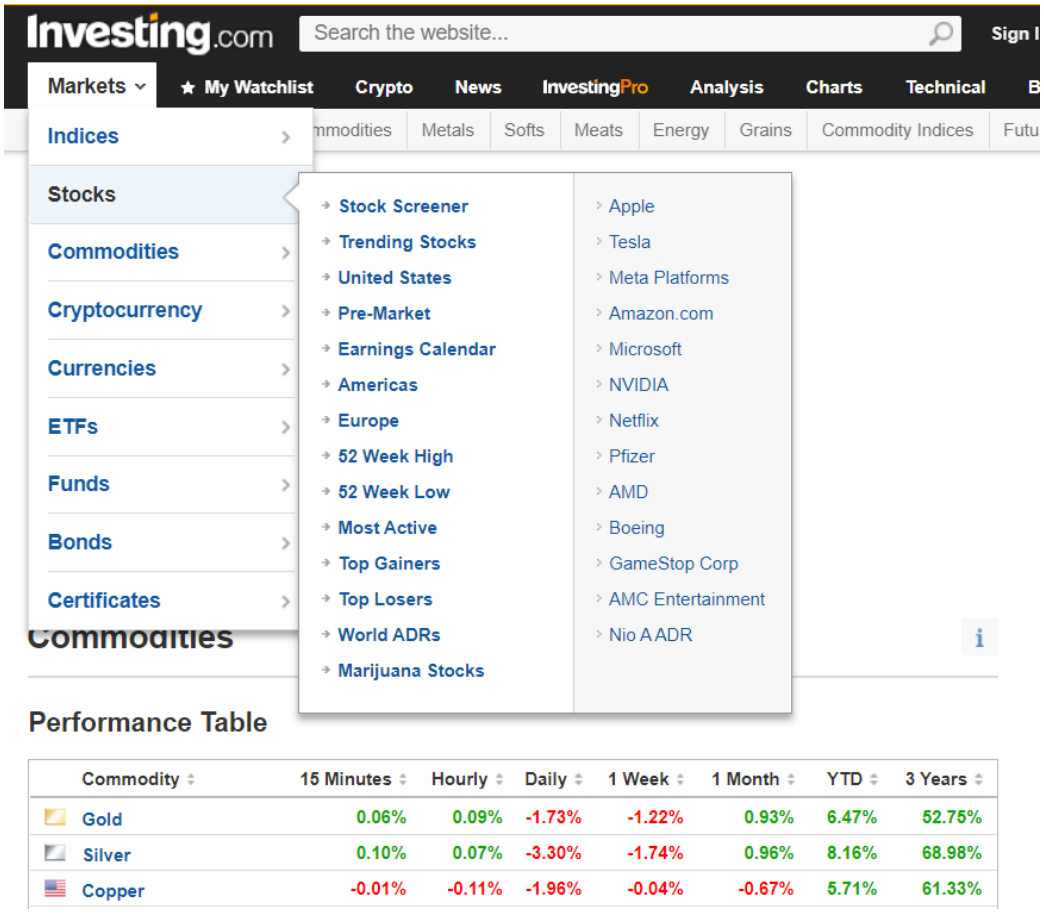
Táto podkapitola obsahuje popis troch konkrétnych scraper projektov, ktoré boli vyvinuté pomocou nástroja vytvoreného v tejto práci. Tieto projekty majú reprezentatívnu funkciu, pretože ukazujú rôzne možnosti použitia vyvinutého nástroja. Takisto ale slúžia ako návod, ako postupovať pri tvorení týchto projektov a ako sa vysporiadať s problémami, ktoré pri web scrapingu na užívateľa čakajú.

Tieto príklady môžu byť použité budúcim užívateľom ako inšpirácia, ako postupovať pri tvorení vlastného scraper projektu a akým spôsobom vytvoriť jednotlivé scraper a formátovacie funkcie. Zdrojové kódy týchto príkladov sú súčasťou aj odovzdaného média, obsa-

hujúceho zdrojové kódy tejto práce. Popis obsahu tohto média a teda aj umiestnenie týchto príkladov na ňom sa nachádza v prílohe A.

Scraper stavu trhu

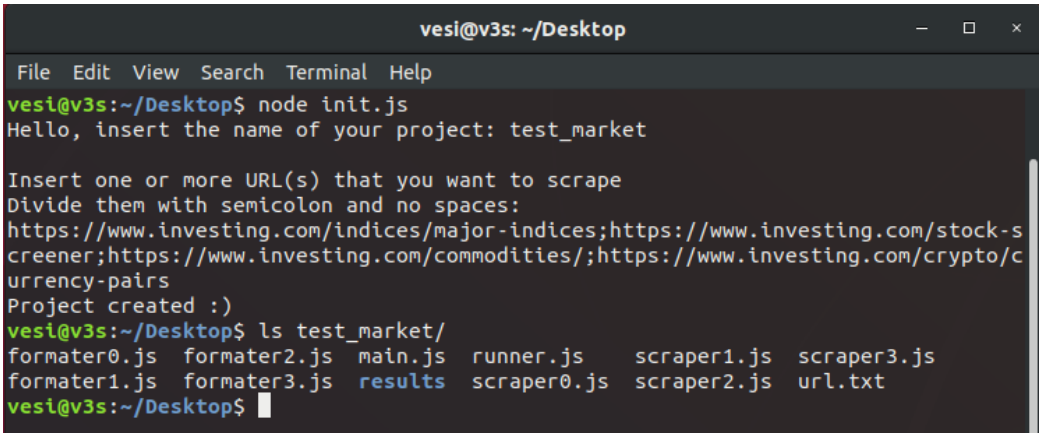
Častý problém pri automatickom získavaní informácií z webu je zložitá štruktúra niektorých webových stránok. Veľmi často dáta na danej stránke nie sú všetky vedľa seba, dokonca sa často ani nedajú naraz zobraziť. Na takýchto stránkach je potrebné preklikávať medzi ich rôznymi sekciami, aby sme získali všetky dáta, ktoré z danej stránky chceme. Príklad takejto stránky je zobrazený na nasledujúcom obrázku: 6.4



Obr. 6.4: Screenshot zobrazujúci zložitú štruktúru webstránky www.investing.com

Pri použití riešenia vyvinutého v tejto práci sa dá s týmto problémom vysporiadať dvomi spôsobmi. Buď budeme v rámci jednej scraper funkcie prechádzať jednotlivé sekcie stránky a postupne získavať všetky dáta, ktoré nás zaujímajú, alebo už pri generovaní projektu inicializačným skriptom zadáme každú sekciu webstránky ako samostatnú URL adresu a teda vytvoríme pre každú z nich samostatnú scraper funkciu.

Tento testovací príklad používa druhý spomínaný prístup. Povedzme, že chceme získať dáta o aktuálnom stave indexov, akcií, komodít a kryptomien. Každá z týchto skupín má na danej stránke svoju sekciu, na ktorej sa nachádza tabuľka s dátami. Do inicializačného skriptu teda zadáme URL adresy priamo týchto štyroch sekcií danej webstránky tak, ako je zobrazené na nasledujúcom obrázku: 6.5.



Obr. 6.5: Screenshot zobrazujúci vytvorenie scraper projektu z jednej webstránky za použitia viacerých URL adries na jej sekcie

Použitie tohto prístupu znamená, že budeme musieť vyplniť štyri samostatné scraper a formátovacie funkcie, no uľahčuje nám prácu v tom, že vôbec nieje potrebné riešiť preklikávanie medzi jednotlivými sekciami. Prepnutie mezdi jednotlivými URL zabezpečí vygenerované riešenie.

Taktiež to znamená, že jednotlivé scraper funkcie bude oveľa jednoduchšie vytvoriť, pretože nám stačí vybrať dáta ktoré chceme z jednej tabuľky. Jedna z týchto scraper funkcií, konkrétne tá, ktorá zberá dáta o komoditách, vyzerá takto:

```
scrape : async function(page){
    let result = {};

    let rows = await page.$$eval('#energy tr',(rows) =>
        rows.map((row) => row.innerText ));
    let row = rows[2].split('\t');
    result[0] = 'Crude Oil;'+ row[3] + ';' + row[8];
    rows = await page.$$eval('#metals tr',(rows) =>
        rows.map((row) => row.innerText ));
    row = rows[1].split('\t');
    result[1] = 'Gold;'+ row[3] + ';' + row[8];
    row = rows[2].split('\t');
    result[2] = 'Silver;'+ row[3] + ';' + row[8];

    return(result);
}
```

Táto funckia je podobná scraper funckií použitej v predchádzajúcom príklade. Najskôr uložíme do poľa všetok text z jednotlivých riadkov tabuľky. Potom už len vyberieme riadky, ktoré nás zaujímajú a z nich tie dáta, ktoré nás zaujímajú. Môžme vidieť, že bola použitá funkcia *page.$$eval()*, narozdiel od predtým použitej funkcie *page.evaluate()*. Táto funkcia iba uľahčuje konštrukciu príkazov tým, že v sebe zahŕňa vytvorenie poľa zo získaných dát. Podrobnosti o tejto funkcií a jej príklady použitia sa nachádzajú v oficiálnej dokumentácií knižnice Puppeteer [14].

Ostatné scraper funckie boli vytvorené rovnakým spôsobom a ich zdrojové kódy sa nachádzajú na priloženom médiu. Nasledujúci krok je teda vytvoriť formátovacie funkcie.

Tento krok je veľmi subjektívny, získané dáta vieme zobraziť nespočetne veľa spôsobmi. V tomto príklade bolo rozhodnuté, že pre každú sekciu dát bude jedna malá tabuľka obsahujúca dané dáta.

V každej z formátovacích funckií teda získané dáta rozdelíme do jednotlivých buniek v tabuľke. Formátovacia funkcia dát o komoditách vyzerá takto:

```
format : function(data){
    result = "";

    result += "<div class=\"Comodities\">" + "<h2>Comodities</h2>";
    x = JSON.parse(data);
    result += "<table><tr>";
    result += "<th>Comodity</th><th>Price</th><th>24H Change</th></tr>";
    for(let i=0; i < 3; i++){
        let row = x[i].split(';');
        result += "<tr><td>"+row[0]+"</td>";
        result += "<td>"+row[1]+"</td>";
        result += "<td>"+row[2]+"</td></tr>";
    }
    result += "</table>" + "</div>";

    return result;
}
```

Následne môžme využiť to, že jednotlivé výsledky formátovacích funkcií budú jeden za druhým vložené do HTML kostry. Do hociktorej formátovacej funkcie teda môžme pridať takýto blok s CSS kódom, ktorý formátuje celý dokument:

```
result += "<style> body{background-color: cyan;}";
result += "table{border: 1px solid black; width: 25%;margin-left: 20px;}";
result += "th{background-color: #FF0000; margin-left: 40px;}";
result += "td{border: 1px solid; padding-left: 4px;padding-right: 4px;}";
result += "</style>";
```

Scraper projekt je teda dokončený, stačí ho už len spustiť pomocou skriptu *main.js*. Výsledok tohto testovacieho scraper projektu je zobrazený na nasledujúcom obrázku: 6.6

Obr. 6.6: Screenshot zobrazujúci výsledok testovacieho scraper projektu získavajúceho dáta o aktuálnom stave trhu

Aj tento testovací príklad bol spustený na viacero dní nonstop, aby bola otestovaná jeho funkčnosť a stabilita.

Scraper športových výsledkov

Tento príklad pristupuje k problému zložitej štruktúry webstránok druhým spomínaným spôsobom. Vytvorí len jednu scraper a formátovaciu funkciu. V rámci scraper funkcie teda

bude musieť prechádzať cez jednotlivé sekcie webstránky, no keďže má k dispozícií rozhranie *page* knižnice Puppeteer, toto prechádzanie nieje žiaden veľký problém.

Tento príklad bude získavať dáta o športových výsledkoch zo stránky Flashscore [2]. Táto stránka obsahuje taktiež veľké množstvo dát a nedokážeme všetky získať naraz. Musíme prechádzať medzi tabuľkami jednotlivých športov a jednotlivých líg v rámci týchto športov.

Vytvorenie scraper projektu nieje nutné znova opisovať. Jednoducho spustíme inicializačný skript a zadáme iba jednu URL adresu "https://www.flashscore.sk/".

Ako už bolo vysvetlené pri minulom príklade, scraper funkcia bude pri použití tohto prístupu rozsiahlejšia, pretože bude zberať všetky dáta z rôznych sekcií stránky, ale taktiež bude musieť preklikávať medzi jednotlivými sekciami.

Samotné získavanie dát prebieha obdobne ako v oboch predchádzajúcich príkladoch, preto nieje nutné ho podrobne opisovať. Zaujímavejšie je ale klikanie pomocou rozhrania *page*. Nestačí nám totižto iba kliknúť pomocou funkcie *page.click()* a hneď začať extrahovať dáta. Stránka sa nemusí stihnúť za tento čas načítať. Ako teda vieme, že stránka bola po kliknutí načítaná a môžme začať s extrakciou dát ?

Na tento účel môžme použiť nasledujúcu konštrukciu:

```
await Promise.all([
    page.click('selector'),
    page.waitForNavigation({ waitUntil: 'networkidle0' })
]);
```

Táto konštrukcia zaručí, že náš prehliadač klikne na HTML prvok definovaný pomocou *selector* a počká, dokým na sieti prestane prebiehať prenos. Teda máme zaručené, že stránka bola načítaná. Niekedy sa ale môže stať, že po načítaní sa stránka ešte nejakým spôsobom mení na klientovej strane pomocou JavaScriptu. Ak stránka z ktorej sa snažíme získať dáta používa takúto funkcianalitu, neostáva nám nič iné než okrem vyššie zobrazenej konštrukcie počkať ešte ďalší fixný čas za použitia príkazu:

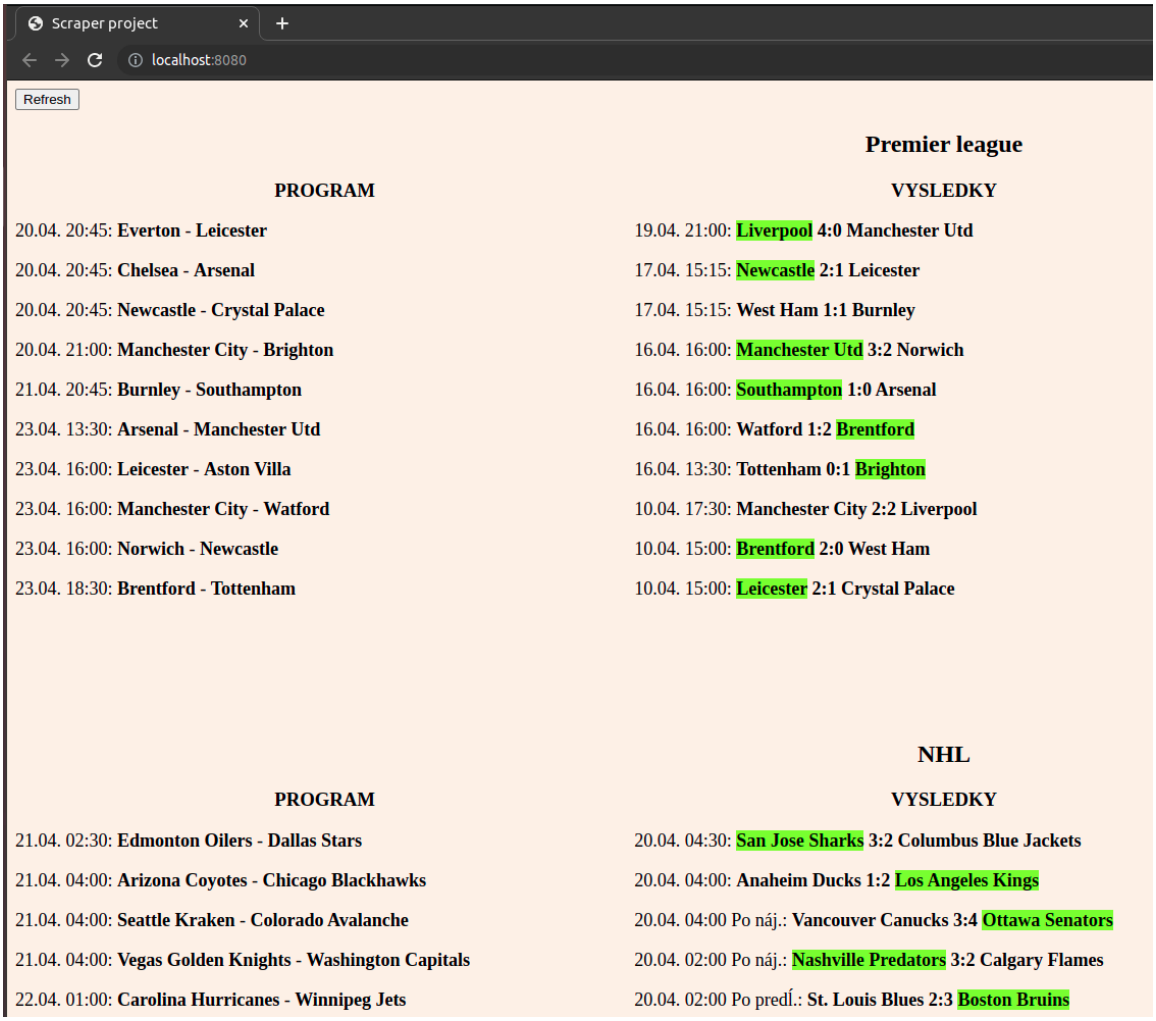
```
await page.waitForTimeout(1000); //waits one second
```

Takýmto spôsobom dokážeme vyskladať celú scraper funkciu. Preklikneme do sekcie stránky, extrahujeme dáta a pokračujeme do ďalšej. Zdrojový kód výslednej scraper funkcie tohto príkladu sa nachádza na priloženom médiu medzi ostatnými príkladmi.

Formátovacia funkcia bude taktiež dlhšia než v predchádzajúcom príklade. Neznamená to ale, že bude aj zložitejšie ju vytvoriť. Postupovať budeme úplne rovnako ako v predchádzajúcom príklade, jediná zmena je v tom, že všetky dáta musíme naformátovať v rámci jednej funkcie. Zaujímavou časťou implementácie tejto formátovacej funkcie je podfarbovanie víťaza zápasu. Dosiahnuť túto funkcionalitu je veľmi jednoduché a demonštruje, že užívateľ má takmer neobmedzené možnosti pri tvorení formátovacích funkcií. Zvyšok formátovacej funkcie je veľmi podobný formátovacím funkciám z predchádzajúceho príkladu a teda nieje nutné ho detailnejšie opisovať.

Výsledok tohto testovacieho scraper projektu je zobrazený na nasledujúcom obrázku: 6.7

[2] Flashscore - www.flashscore.sk



Obr. 6.7: Screenshot zobrazujúci výsledok testovacieho scraper projektu získavajúceho dáta o výsledkoch športových zápasov. Na screenshote nie je zobrazená úplne celá výsledná stránka kvôli lepšej viditeľnosti.

Scraper programu rádiových staníc

Ako tretí príklad bol vytvorený scraper na programy rôznych rádiových staníc. Tento príklad demonštruje použitie riešenia na viacero rôznych webstránok s rôznou štruktúrou.

V tomto príklade sa samotné scraper a formátovacie funkcie príliš nelíšia od tých, ktoré boli zobrazené v predošlích dvoch príkladoch. Preto bude v tomto príklade podrobnejšie opísané ukladanie dát do štruktúrovanej podoby. Tento krok nebol doteraz dostatočne vysvetlený.

Bolo vybraných niekoľko rádiových staníc z rôznych krajín, ktoré majú na svojej webstránke program na aktuálny deň. Program sa väčšinou skladá z názvu relácie a času jej vysielania, následne môže obsahovať nejaký bližší popis, zaradenie do určitej kategórie, účinkujúcich a prípadne aj titulný obrázok. Na každej zo stránok je program štruktúrovaný inak. My ale chceme mať dáta v štruktúrovanej podobe, ideálne z každej stránky rovnakej. Toto nám uľahčí formátovanie daných dát ale zároveň to zvýši ich prehľadnosť a čitateľnosť.

Musíme si teda vopred naplánovať, ktoré dáta z daných stránok budeme extrahovať, aby mohli mať všetky rovnakú štruktúru. V tomto prípade sme rozhodli, že každá relácia bude obsahovať názov, bližší popis a čas vysielania. Nasleduje príklad jednej zo scraper funkcií, ktorá ukladá dáta do žiadanej štruktúry:

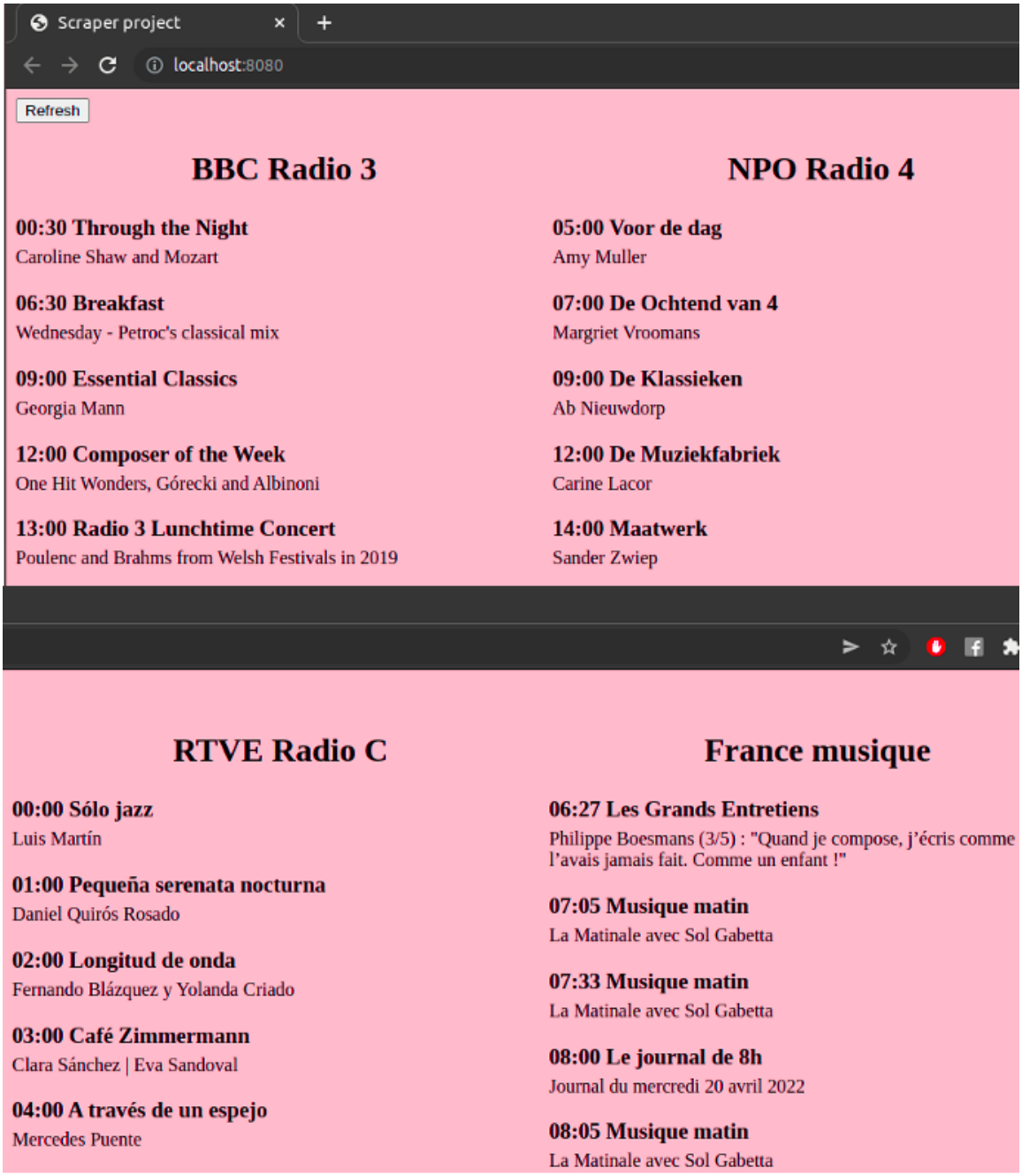
```
scrape : async function(page){
    let result = {};

    const text = await page.$$eval('.grid-wrapper',(rows) =>
        rows.map( (row) => row.innerText ));
    for(let i=1; i<text.length; i++){
        row = text[i].split('\n');
        result[i-1] = {};
        result[i-1]['time'] = row[0];
        result[i-1]['title'] = row[2];
        result[i-1]['description'] = row[3];
    }
    return(result);
}
```

Ako môžme vidieť, do objektu *result* ukladáme jednotlivé relácie ako samostatné objekty, pomenované číslami od nuly. Každý z týchto objektov obsahuje atribúty *time*, *title* a *description*. Výsledné dáta z tejto scraper funkcie teda majú takúto štruktúru:

```
{
  "0": {
    "time": "00:30",
    "title": "Through the Night",
    "description": "Caroline Shaw and Mozart"
  },
  "1": {
    "time": "06:30",
    "title": "Breakfast",
    "description": "Wednesday - Petroc's classical mix"
  },

  ...
}
```

Ak vo všetkých ostatných scraper funkciách použijeme rovnakú štruktúru dát, výrazne nám to uľahčí tvorenie formátovacej funkcie a dáta budú vo výsledku oveľa prehľadnejšie.

Výsledok tohto testovacieho scraper projektu je zobrazený na nasledujúcom obrázku: 6.8



Obr. 6.8: Screenshot zobrazujúci výsledok testovacieho scraper projektu získavajúceho dáta o programoch vysielania rôznych rádiových staníc. Screenshot je rozdelený na dve časti, ktoré sú prilepené vertikálne vedľa seba pre lepšiu viditeľnosť.

Tipy na záver

Keďže má táto kapitola slúžiť aj ako určitý návod, ako s vytvoreným nástrojom pracovať, bude ešte spomenutých zopár vecí, ktoré môžu budúcemu užívateľovi pomôcť uľahčiť prácu.

Ako prvý a najdôležitejší tip je používanie nástroja Chrome Developer Tools, alebo jeho alternatív v iných webových prehliadačoch. Počas tvorenia scraper funkcií práve tento nástroj dokáže veľmi uľahčiť prehliadanie zdrojových súborov webstránky a taktisto poskytuje

konzolu, v ktorej je možné testovať vytvorené JQuery selectory. Tento nástroj bol používaný aj pri vývoji testovacích príkladov.

Ďalší tip pre budúceho užívateľa je na začiatku scraper funkcií robiť kontroly, či daná webstránka funguje. Ak chceme aby bol náš scraper projekt stabilný a vydržal aj prípadné zlyhanie webstránky, z ktorej sú dáta extrahované, môžme implementovať jednoduchú kontrolu na začiatku scraper funkcie. Napríklad skúsime extrahovať hlavný nadpis tejto stránky a ak sa nezhoduje s očakávaním, dáta neextrahujeme a do výsledku uložíme chybovú hlášku.

Posledný tip je pre užívateľa, ktorý by chcel sprístupniť svoje riešenie celému webu, nielen na jeho stroji. Jednoduchý spôsob ako toto dosiahnuť je nástroj *lt* [5]. Tento nástroj umožňuje veľmi jednoduché zdieľanie webovej služby z lokálneho stroja na testovacie účely tak, že užívateľovi pridelí verejnú URL adresu, cez ktorú sa môže hocikto pripojiť na danú službu fungujúcu na lokálnom stroji.

6.3 Možné vylepšenia

Vytvorené riešenie má určitý priestor na vylepšenia. Napríklad by mohla byť pridaná možnosť ukladať dáta aj z minulosti a teda vytvárať históriu získaných dát, prípadne nejaké štatistiky z nich.

Taktiež by mohol byť zlepšený spôsob formátovania získaných dát. Vkladanie HTML a CSS kódu do obyčajného reťazca môže byť niekedy neprehľadné a chaotické. Mohlo by sa zaviesť používanie určitých "šablón" a možnosti dáta do týchto šablón vkladať jednoduchším spôsobom.

Ďalšie vylepšenie by mohlo byť umožnenie extrahovať z webu aj obrázky a zobrazovať ich spolu s dátami.

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo navrhnúť a implementovať nástroj, ktorý umožňuje získavať definované informácie z väčšieho množstva webových zdrojov a zobrazovať ich v užívateľsky prívetivej podobe.

Tento cieľ bol dosiahnutý a výsledný nástroj svojmu užívateľovi značne uľahčuje vytvorenie vlastného web scraperu. Odtieňuje užívateľa od všetkých implemetačných detailov, stačí iba definovať požadované dáta a požadovanú formu ich prezentácie.

Výsledný nástroj bol následne testovaný pomocou troch testovacích príkladov, ktoré slúžia na overenie jeho funkčnosti, ale tak isto aj ako ukážka jeho širokých možností. Testovanie potvrdilo, že vytvorený web scraper dokáže fungovať dlhú dobu nepretržite a bez problémov aktualizovať dáta získavané z webových stránok.

Práca mala pre mňa prínos vo viacerých oblastiach. Najviac v oblasti web scrapingu - jeho princípy, možnosti a obmedzenia. Taktiež ma zoznámila s programovacím jazykom JavaScript, ktorý je momentálne vo svete veľmi populárny.

V práci by sa dalo pokračovať viacerými spôsobmi. Jednoduchý spôsob, ako prácu vylepšiť by bolo napríklad umožniť užívateľovi udržovať históriu získaných dát, nielen ich aktuálnu podobu. Ďalšia možnosť vylepšenia by bola prepracovať užívateľské rozhranie a umožniť užívateľovi prívetivejším spôsobom definovať požadované dáta a ich výsledné zobrazenie. Toto vylepšenie by ale bolo veľmi rozsiahle a vyžadovalo by veľký objem času, prípadne samostatnú bakalársku či diplomovú prácu.

Literatúra

[1] Andre, L. *53 Important Statistics About How Much Data Is Created Every Day* [online]. [cit. 2022-31-03]. Dostupné z: https://financesonline.com/how-much-data-is-created-every-day/.

[2] Burget, R. *Extrakce dat z webu* [online]. [cit. 2022-30-03]. Dostupné z: https://www.fit.vutbr.cz/~burgetr/upa/05_webscraping/.

[3] Campbell, S. *20 BEST Web Scraping Tools for Data Extraction (Apr 2022 List)* [online]. 2022-31-03 [cit. 2022-03-04]. Dostupné z: https://www.guru99.com/web-scraping-tools.html.

[4] Gundecha, U. a Avasarala, S. *Selenium WebDriver 3 Practical Guide: End-to-end automation testing for web and mobile browsers with Selenium WebDriver, 2nd Edition*. Packt Publishing, 2018. ISBN 9781788996013. Dostupné z: https://books.google.sk/books?id=_AhnDwAAQBAJ.

[5] LocalTunnel Contributors. *LocalTunnel* [online]. [cit. 2022-06-04]. Dostupné z: https://github.com/localtunnel/localtunnel.

[6] MDN Contributors. *CSS: Cascading Style Sheets* [online]. 2022-30-03 [cit. 2022-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS.

[7] MDN Contributors. *HTML: HyperText Markup Language* [online]. 2022-18-02 [cit. 2022-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[8] MDN Contributors. *Introduction to the DOM* [online]. 2022-18-02 [cit. 2022-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[9] MDN Contributors. *JavaScript* [online]. 2021-28-07 [cit. 2022-02-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[10] Mitchell, R. *Web Scraping with Python: Collecting More Data from the Modern Web*. O'Reilly Media, Inc., 2018. ISBN 9781491985526. Dostupné z: https://books.google.cz/books?id=TYtSDwAAQBAJ.

[11] NodeJS Contributors. *Introduction to Node.js* [online]. [cit. 2022-02-04]. Dostupné z: https://nodejs.dev/learn.

[12] NPM Contributors. *About npm* [online]. [cit. 2022-02-04]. Dostupné z: https://docs.npmjs.com/about-npm.

[13] Perez, M. *What is web scraping and what is it used for ?* [online]. 2021-01-08 [cit. 2022-01-04]. Dostupné z: https://www.parsehub.com/blog/what-is-web-scraping/.

[14] Puppeteer Contributors. *Puppeteer* [online]. [cit. 2022-02-04]. Dostupné z: https://pptr.dev/.

[15] Tansley, D. a Tansley, D. *Linux and UNIX Shell Programming*. Addison-Wesley, 2000. ISBN 9780201674729. Dostupné z: https://books.google.sk/books?id=bN57Lq5yFnQC.

[16] World Wide Web Consortium (W3C). *Cascading Style Sheets home page* [online]. [cit. 2022-01-04]. Dostupné z: https://www.w3.org/Style/CSS/#specs.

Príloha A

Obsah priloženého média

Dátové médium priložené k tejto bakalárskej práci obsahuje:

- *src/* - adresár obsahujúci zdrojové kódy praktickej časti práce
- *doc/* - adresár obsahujúci zdrojové súbory textu práce
- *examples/* - adresár obsahujúci užívateľské vstupy do jednotlivých testovacích príkladov a screenshoty výsledkov
- *README.txt* - textový súbor, v ktorom je opísaný obsah priloženého média a podrobne opísaná inštalácia a používanie vytvoreného nástroja