Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Extrakce dat z webu pomocí web scrapingu

Diplomová práce

Autor: Bc. Jan Thér

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Martina Husáková, Ph.D.

Hradec Králové Srpen 2022

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 14.08.2022 Jan Thér

Poděkování:

Děkuji vedoucí diplomové práce Ing. Martině Husákové, Ph.D. za metodické a svědomité vedení práce, a především za notnou dávku trpělivosti.

Anotace

Cílem diplomové práce je seznámit čtenáře s problematikou získávání dat z internetu, následně popsat způsoby jejich získávání a dále vytvořit nástroj který bude těchto přístupů využívat pro efektivní a intuitivní práci s daty z webových stránek. Tento nástroj, webová aplikace, bude vytvořen na základě popsání a srovnání existujících web scrapingových nástrojů a jeho cílem bude odstranit jejich nedostatky. Následně bude tento nástroj otestován na několika ukázkových webech. Výsledky tohoto testování budou poté shrnuty a dány do souvislosti s přístupy popsanými v teoretické části práce. Návrh a implementace tohoto nástroje bude podrobně popsána v práci.

Annotation

Title: Extracting web data using web scraping

The aim of the diploma thesis is to familiarize the reader with the problem of extracting data from the internet, then to describe the approaches for obtaining them and to create a tool that will use these approaches for efficient and intuitive work with data from websites. This tool, a web application, will be created based on the analysis and comparison of existing web scraping tools and will aim to eliminate their shortcomings. Subsequently, this tool will be tested on several sample sites. The results of this testing will then be summarized and put into context with the approaches described in the theoretical part of the thesis. The design and implementation of this tool will be described in detail in the thesis.

Obsah

1 Úvod

„Kdo ovládá koření, ovládá vesmír“, tak alespoň zní citát ze slavného románu Duna od Franka Herberta. Kdyby však Frank žil dnes, jistě by svůj původní text poupravil. Vývoj posledních let totiž ukazuje na jasný trend, kdy úspěch společností či produktů dnes více než kdy jindy závisí na schopnosti získávat data a následně s nimi vhodně pracovat. V současné době je každý z nás obklopen nekonečným spektrem rozličných dat, a to nejen ve světě informatiky a internetu, ale takřka kdekoliv, protože co dnes nejsou data. Nejsou právě oním kořením?

V současnosti je možné se často setkávat s pojmem „data-driven“, tedy řízený daty. Tento přístup se nyní dostává do popředí čím dál častěji, a to jak při vývoji různých aplikací, testování, vědeckých výzkumech, tak marketingu. Správné zpracování velkého množství dat totiž umožňuje přistupovat k řešení problémů zcela odlišnými způsoby než donedávna, kdy hlavním motorem úspěchu byla intuice a zkušenost. Získávání dat je pak jen jedním z dílčích kroků většího celku, který je dnes hojně označován jako datová analýza.

Následující práce se zabývá otázkou, jak přistupovat a získávat data z internetu, především pak z webových stránek. Vzhledem k ohromnému množství dat již manuální shromažďování informací dávno není dostačující, proto jsou vyvíjeny nástroje a postupy, které tento proces zefektivňují a automatizují. Tato práce si dává za cíl seznámit čtenáře s problematikou získávání dat z internetu, způsoby, jak k získávání dat přistupovat, jejich úskalími a možnými řešeními těchto úskalí. Především je pak zaměřena na takzvaný web scraping, který je právě jedním ze způsobů, jak data efektivně získávat.

Následně dojde k popsání existujících web scrapingových nástrojů. Tyto nástroje budou srovnány a na základě vyhodnocení tohoto srovnání dojde k návrhu a posléze implementaci webové aplikace, která umožní uživatelům webu efektivně získávat data z internetových stránek, a to právě s pomocí využití web scrapingu.

Samotná webová aplikace, která je součástí praktické části práce, je rozdělena na dvě části – klientskou a serverovou. Obě tyto části budou podrobněji popsány v práci. Tento webový nástroj by měl umožňovat získávání dat z webových stránek,

ke kterým bude přistupovat způsoby aktivně bojujícími proti detekci anti-bot nástroji. Z takto obdržených dat bude následně možné volit pro uživatele zajímavé prvky. Získaná data by mělo být následně možné ukládat v podobě vhodné pro další analýzu. Vzniklý nástroj by tak měl být vhodnou alternativou k již existujícím web scraperům popsaných v rámci práce.

2 Vybrané technologie extrakce dat z webu

Ačkoliv je tato práce primárně zaměřena na extrakci dat pomocí web scrapingu, je vhodné přiblížit čtenáři i jiné technologie, které umožňují získávat data z webových stránek. Jednou z takových technologií je API neboli Application Programming Interface, která bude popsána v následující kapitole.

2.1 API

API může být ve zkratce popsáno jako sada pravidel, která umožňují snadnou komunikaci napříč aplikacemi. Současný uživatel webu se s ním dnes setkává snad na každém kroku. Lze říci, že se bez této technologie web už ani neobejde téměř všude tam, kde je vyžadována práce s externími daty. Jedním z příkladů může být mobilní aplikace, která zobrazuje počasí. Data o teplotách, vlhkosti nebo třeba rychlosti větru s největší pravděpodobností nejsou generována pomocí senzorů v daném zařízení, nýbrž získávána pomocí volání, která přistupují k jedné z nesčetných API meteorologických ústavů po celém světě. Je také důležité podotknout, že se API nevyskytují výhradně v oblasti webů, ale napříč všemi technologiemi, které vyžadují komunikaci s jinými systémy.

2.1.1 Jak API funguje?

V souvislosti s touto technologií je často možné se setkat s analogií restauračního zařízení. Většina restauračních zařízení se neobejde bez obsluhy a kuchyně. V této analogii může být kuchyně považována za jakýsi vzdálený server, který poskytuje informace. Obsluha tato data předává zákazníkovi. Zákazník, tedy uživatel aplikace, však o tyto informace musí nejdříve obsluhu požádat. Tomuto požadavku se říká API volání. Jakmile dojde k tomuto volání, obsluha předá požadavek kuchyni, tedy serveru. Server data vygeneruje, předá je obsluze a ta je předá zákazníkovi. Důležité je vědět, že uživatel (uživatelova aplikace) tedy zákazník, nemusí vědět, jakým způsobem server pracuje. Stačí, když dokáže pracovat s jeho API. Data jsou mu pak většinou předána ve srozumitelné podobě na požádání. Tím se technologie API

fundamentálně liší od web scrapingu, který k získávání dat z webu přistupuje zcela odlišně.

V souvislosti s předchozí kapitolou by se v tomto případě mohlo jednat o data o počasí, kdy je při otevření aplikace vysláno volání na API vzdálený server, který informace poskytuje. Tato data jsou pak většinou zpracována a upravena samotnou mobilní aplikací či webovou stránkou, aby byla uživateli předána v efektní a srozumitelné podobě. Ačkoliv tato úprava dat již není součástí samotného procesu práce s API, jedná se o velmi důležitou součást procesu jak práce s API, tak web scrapingu. Jen zřídkakdy jsou data využívána v podobě, v jaké byla primárně extrahována.

2.1.2 Druhy API

Jak tomu bývá ve všech oblastech výpočetních technologií, tak i API prošly v průběhu let vývojem a modifikacemi, které umožňují jejich konkrétní a efektivní užití. API se dají dělit podle mnoha kritérií. Ať už je to například podle rozsahu a přístupu k jejich užívání např. OPEN API, ke kterým má přístup každý, jelikož jsou veřejně dostupné. Dále pak je lze dělit podle způsobu, případu konkrétního užití nebo konkrétní architektury. Druhů API je nepřeberné množství, ale jako příklad může být uvedeno již zmiňované API pro počasí. V současnosti je možné se setkat se třemi hlavními architekturami, které jsou blíže popsány v následující pasáži:

REST

Tedy Representational State Transfer, je v současnosti nejpoužívanější architektura používaná napříč webovými API. V podstatě se jedná o řadu pravidel a pokynů pro snadno použitelné, škálovatelné a lightweight API. K tomu, aby mohla být API považována za REST, musí splňovat právě tato pravidla a pokyny. REST by měla vracet klientovi, tedy koncovému uživateli, data v takové podobě, která mohou být snadno zpracovatelná. Klient může pomocí REST API také často upravovat data na

serveru. Aby mohla být API považována za REST, musí splňovat následující řadu pravidel:

- **oddělení klienta od serveru:**

  Klient a server spolu mohou interagovat jen jediným způsobem, tj. klient vyšle na server požadavek a ten odpoví. Server nemůže vysílat požadavky na klienta. Stejně tak klient nemůže odpovídat. [1]

- **jednotný interface:**

  Všechny požadavky a všechny odpovědi musí odpovídat jednotnému formátu. Většina REST využívá ke komunikaci protokolu HTTP (Hypertext Transfer Protocol). Většina požadavků tak ke správnému použití vyžaduje dvě základní informace. První je informace o endpointu – cíli, odkud budou získávány informace (zpravidla URL adresa). Druhou je metoda, kterou je k cíli přistupováno. Tyto metody jsou v zásadě čtyři a jsou jimi: **GET** – pro získávání dat ze serveru, **POST** – pro vytvoření nových dat, **PUT** – pro editaci již existujících dat a nakonec **DELETE** – pro vymazání existujících dat. Po obdržení požadavku vrací server informace o zdroji. Tyto informace jsou většinou ve formátu JSON (JavaScript Object Notation). [1]

- **stateless:**

  Všechny požadavky jsou bezstavové, to jinými slovy znamená, že každý požadavek každého klienta je serverem zpracováván separátně. Z toho plyne, že server nezná předchozí požadavky klienta a ke všem tak přistupuje nezávisle na sobě. [1]

- **vícevrstvé systémy:**

  Každá komunikace mezi serverem a klientem by měla probíhat podle stejného formátu, a to bez ohledu na to, kolik se mezi klientem a konečným serverem nachází jiných vrstev. Takovými vrstvami mohou být například systémy pro zabezpečení. [1]

- **cachable:**

  Některá data, ke kterým je například přistupováno často, mohou být uložena v cache paměti na klientském zařízení. Ve chvíli, kdy jsou taková data vyžadována, nemusí dojít k vyslání požadavku na server, ale jsou namísto toho získána z lokálního uložiště. Je tím tak snížena zátěž serveru a data jsou získána rychleji. [1]

SOAP

Je zkratka pro Simple Object Access Protocol, což je na rozdíl od REST protokol. To znamená, že jeho užívání stojí oproti REST na mnohem rigidnějších pravidlech. Musí být implementován podle přesně daných požadavků a jedním z nich je například výhradní užití formátu XML (Extensible Markup Language) při zasílání zpráv. Kvůli této rigidnosti je použití SOAP často komplikovanější nežli REST. Na druhou stranu z něj však tato vlastnost činí spolehlivější a bezpečnější řešení, a ačkoliv se jedná o starší z API architektur, právě díky této vlastnosti je možné se s ní setkat i v současnosti, a to zejména v oblastech, které vyžadují spolehlivý chod. [1]

RPC

Neboli Remote Procedure Call je protokol, který využívá buď formátu XML nebo JSON. Pro získávání dat používá metodu GET a pro všechnu ostatní funkcionalitu POST. Struktura dat a komunikace napříč systémy není svazována takovými pravidly jako v případě SOAP. Oproti SOAP je tak považován za jednodušší, a to jak v oblasti výpočetní ceny, tak učící křivky pro práci s tímto protokolem. Díky těmto vlastnostem dosahuje vyššího výkonu, a právě proto se často používá například v souvislosti s technologií microservices (tj. mikroslužeb). [2]

3 Web scraping

Web scraping, často označovaný také jako screen scraping nebo web mining, je soubor rozličných technik, které slouží k získávání dat z internetových stránek za účelem jejich dalšího zpracování. Ve své podstatě se každý, kdo někdy zkopíroval a někam vložil určitá data z internetu, může považovat za web scrapera. Takový postup je však obzvlášť v době dnešního internetu, kdy jednotlivé weby tvoří gigabyty dat, značně neefektivní. Pro zefektivnění tohoto procesu proto vznikají automatické nástroje, které dokáží zpracovat tisíce stránek denně. Jedná se o tzv. web scrapery.

Vyvstává otázka, proč nevyužít nástrojů API, které přesně k těmto účelům slouží, jak bylo popsáno v předchozí kapitole. Rozdíl mezi získáváním dat pomocí API a web scrapingu je zásadní. API je možné využít pouze tam, kde autor určité aplikace jejich použití umožňuje. To znamená, že sám svá data nabízí v nějaké strukturované formě tak, aby je mohli ostatní uživatelé využít. Valná většina internetových zdrojů však touto možností nedisponuje. API má také určitá omezení, tj. ne vždy je jejich užívání zdarma, může být omezeno určitými datovými limitacemi, to znamená, že umožňuje například jen určitý počet přístupů za den, anebo jednoduše neposkytuje všechna uživatelem vyžadovaná data. Pokud by si tak uživatel například usmyslel srovnat ceny určitého produktu napříč rozličnými e-shopy, nezbývalo by mu, než manuálně navštívit všechny tyto e-shopy, nalézt konkrétní produkty a zjistit jejich ceny. Takto zjištěné informace by pak bylo nutné dále zpracovat. Například jimi naplnit uživatelem vybraný tabulkový procesor, opět manuálně. Na takovém postupu není v případě srovnání cen jednoho produktu nic špatného. V případě, kdy je potřeba srovnávat ceny většího množství produktů každý den, kdy se navíc stránky pravidelně aktualizují, například dochází k častým změnám cen, je takový postup neudržitelný.

Web scrapery tento postup automatizují. Stejně jako uživatel, tak i web scrapery navštěvují jednotlivé stránky a získávají z nich konkrétní, uživatelem zvolená data. Internetový prostor je v současnosti z valné většiny tvořen HTML kódem, který je určen k tomu, aby byl vhodně zpracováván člověkem, ale pro automat je nestrukturovaný. To znamená, že musí být dále upraven pro další zpracování. Web

scrapery zastávají i tuto činnost. Následně jsou tato data uložena v uživatelem zvoleném formátu a struktuře například v databázi či tabulkovém procesoru. Web scraping je tak možné považovat za jeden z dílčích kroků datových věd, kdy jsou na základě dalších analytických nástrojů získávány další informace.

3.1 Kde se web scraping využívá?

Na první pohled se může zdát, že web scraping je jen okrajovou záležitostí pár nadšenců, kteří si chtějí usnadnit nákupy. Pravda je však taková, že společnosti v současnosti více a více zastávají takzvaný data-driven (tj. daty řízený) přístup. To znamená, že strategická rozhodnutí činí na základě analýz velkého množství dat. Tato data však společnosti musí nějakým způsobem získat. Pokud nemají data interní, nezbývá, než se uchýlit k využití web scrapingu. Zde následuje pár příkladů možných případů užití:

- **analýza produktů e-shopů:**

  Konkurenceschopnost je v současném světě internetové prodeje velice zásadní. Mnoho společností se tak snaží neustále přicházet se způsoby, jak zvýšit svou konkurenceschopnost na trhu. Jedním ze způsobů, jak využívat web scrapingu k získání převahy na trhu, je analýza e-shopů. V takovém případě dochází k extrakci dat jednotlivých produktů. Jedná se většinou o jejich ceny, hodnocení, dostupnost, či získání produktů, které se zobrazují jako doporučené. Na základě těchto dat je možné pomocí analytických nástrojů zjistit, jaký je o produkt zájem, jaká je jeho průměrná cena či hodnocení. Společnosti poté na základě těchto zjištění upravují své marketingové snahy, mění ceny svých produktů, omezují či zvyšují jejich kapacitu.

- **umělá inteligence:**

  Většina nástrojů umělé inteligence využívá ke svému učení velkého množství dat. Mnoho z těchto dat je získáváno z webů právě pomocí web scraperů. Za příklady takových nástrojů může být uvedena například klasifikace obrázků,

kdy nástroj dokáže určit zvíře nacházející se na obrázku. Dalším příkladem mohou být chatovací nástroje, které vznikají pomocí milionů textů získaných web scrapery ze sociálních sítí.

- **predikce trhu:**

  Informace získané sledováním jak sociálních sítí, tak webového zpravodajství jsou v současnosti nezřídka využívány k predikci vývoje trhu. Dříve se jednalo převážně o akciové trhy. V současnosti se však prediktivních analýz z dat získaných web scrapery využívá například i k predikci dalšího směřování cen kryptoměn.

Všechny výše zmíněné oblasti přes marketing, analýzu konkurence a trhu, sociální a další vědecké výzkumy se dnes bez práce s velkým objemem dat neobejdou. S web scrapingem je tak možné se v současnosti setkat takřka ve všech oblastech života.

3.2 Techniky web scrapingu

Techniky web scrapingu je možné rozdělit na základě mnoha různých kritérií. Pro snadné seznámení však bude uvedeno dělení jen na základě způsobu, jakým je přistupováno k získávání dat.

3.2.1 Manuální získávání dat

Jak již bylo zmíněno výše, každý, kdo někdy zkopíroval data z webového zdroje a následně tato data uložil, prováděl web scraping. Tato technika je ve většině případů považována za velice neefektivní, a proto nevhodnou pro zpracování velkého množství dat. Je však možné setkat se i s případy, kdy se uživatel bez užití této techniky neobejde. Struktura některých webů může být mnohdy natolik komplikovaná, že neumožňuje využití žádných automatických nástrojů. Mnoho současných webů také obsahuje nástroje, které se snaží aktivně zamezit používání těchto automatických nástrojů. V takových případech je manuální získávání dat jedinou možností.

3.2.2 Regulární výrazy

Jedná se o velmi efektivní a zároveň jednoduchou techniku získávání dat z webových stránek. Regulární výrazy, zkráceně regex, jsou podporovány většinou programovacích jazyků a jejich implementace je tak často jednoduchá. U této techniky není potřeba rozkládat HTML strom, ale dochází k vyhledávání dat pomocí shody výrazů. Jedná se o techniku, která využívá vyhledávacích vzorů. Tyto vzory využívají dvou typů znaků, literálů a tzv. metaznaků, speciálních znaků, které umožňují pokročilejší možnosti vyhledávání. Jedním ze speciálních znaků je například tečka (.), ta zastupuje jakýkoliv libovolný znak. Její použití v regexu by pak mohlo vypadat následovně: **.ad** – tento vzorec vyhledává všechny řetězce, které skládají přesně ze tří písmen a končí řetězcem **ad**. Podobných speciálních znaků existuje celá řada. Dalšími důležitými znaky jsou tzv. kvantifikátory, ty určují, kolikrát je určitý řetězec obsažen v uživatelem vyhledávaném textu. Otazník (?) znamená minimálně žádné, maximálně však jedno opakování. Hvězdička (*) značí žádný nebo libovolný počet opakování. Plus (+) značí alespoň jedno a více opakování. [3]

Vzorec pro získání všech elementů odstavců včetně tagů a jejich obsahu by vypadal následovně:

```
<p>.+<\/p>
```

HTML:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Titulek</title>
   </head>
   <body>
      <p>Prvni odstavec.</p>
      <div>
         <p>Druhy odstavec.</p>
         <p>Treti odstavec.</p>
      </div>
   </body>
</html>
```

Uvedený vzor by nalezl shodu s následujícími prvky:

```
<p>Prvni odstavec.</p>
<p>Druhy odstavec.</p>
<p>Treti odstavec.</p>
```

Jak je z ukázky patrné, tato metoda zcela ignoruje HTML strukturu dat a řídí se pouze shodou řetězců. Ačkoliv se jedná o velmi efektivní přístup, bez dostatečné znalosti regulárních výrazů může docházet k získávání neočekávaných dat, a to obzvlášť na webech, které nemají jednotnou strukturu.

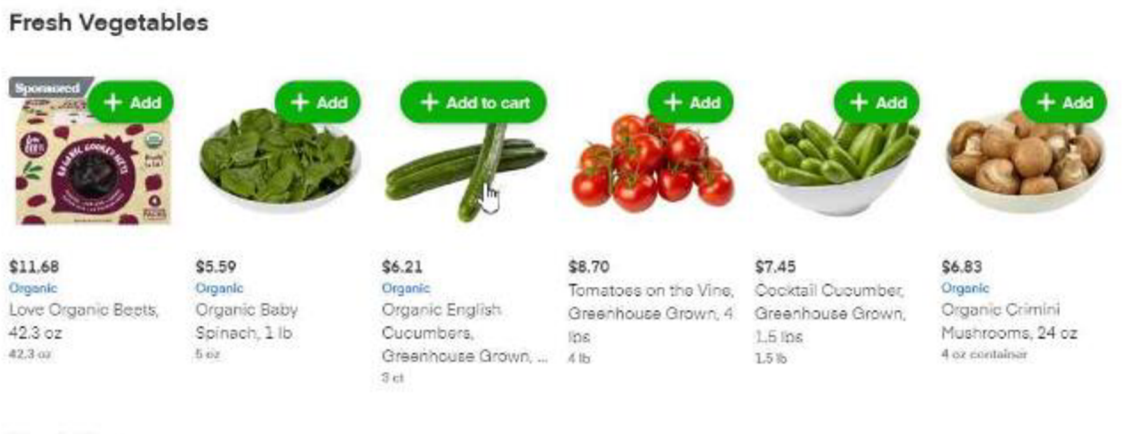
3.2.3 HTTP požadavky – HTML analýza

Tato metoda je vhodná především pro statické weby, jejichž obsah je generován již na serveru. Při této analýze dochází k vyslání HTTP požadavku a obsah webu je následně stažen v takové podobě, v jaké byl v danou chvíli vytvořen serverem. Následné zpracování dat pak probíhá procházením HTML pomocí tagů, elementů a dalších prvků. Tato metoda není nejvhodnější pro práci s dynamickými weby, jejichž obsah se mění na základě interakce uživatele. Nicméně vhodným využitím HTTP požadavků, nebo hlubší znalostí dané domény, je možné využít tuto metodu i pro dynamické weby. Výhodou této techniky je její relativní rychlost a snadná použitelnost. Na druhou stranu je náchylnější pro detekci botů. Je tak nutné mít na paměti, že opakované používání této metody může často vést k omezení přístupu k webu.

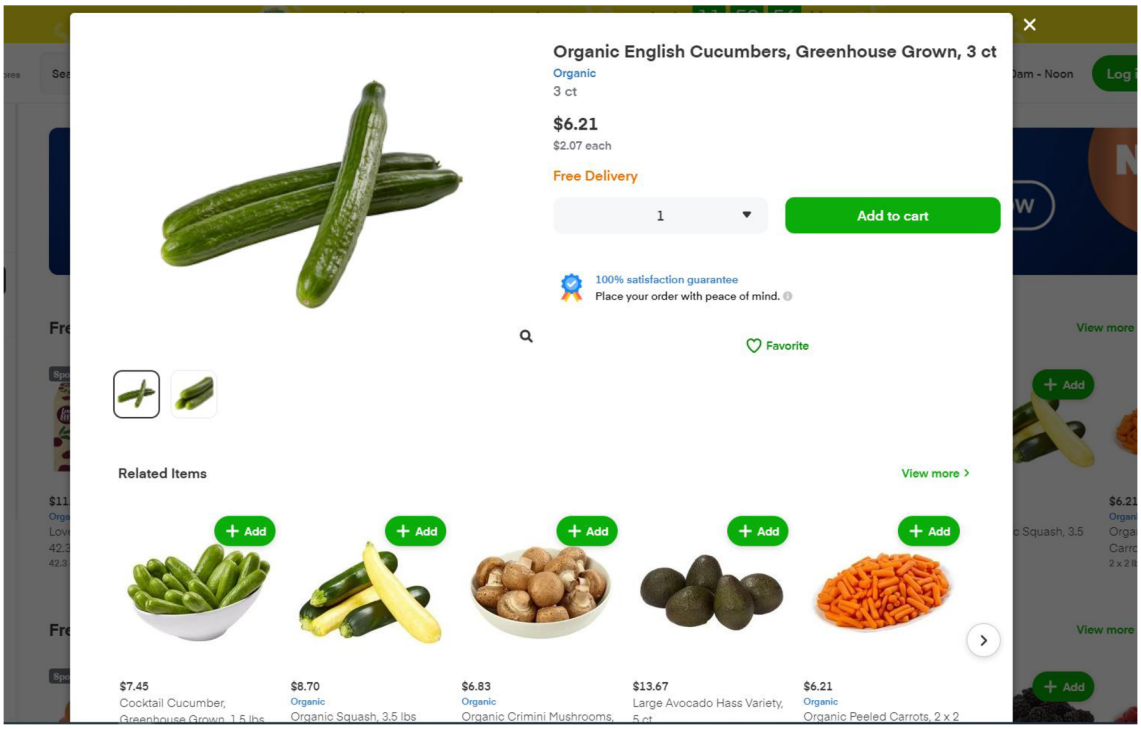
3.2.4 DOM analýza

Na rozdíl od předchozí metody je analýza DOM (Document Object Model) vhodná i pro dynamické weby. Obsah takových webů může být z velké části generován až na straně klienta, a to zejména za použití JavaScriptu a asynchronních volání pomocí technologie AJAX. Takové weby často při prvotním přístupu obsahují jen základní informace, které jsou následně generovány. Jako jeden z příkladů dynamického

získávání dat může být zmíněno zobrazení modálního okna s detaily produktu. S těmito okny je možné se setkat na mnohých e-shopech. Při kliknutí na obrázek produktu se na server vyšle požadavek, který získá dodatečné informace o produktu:

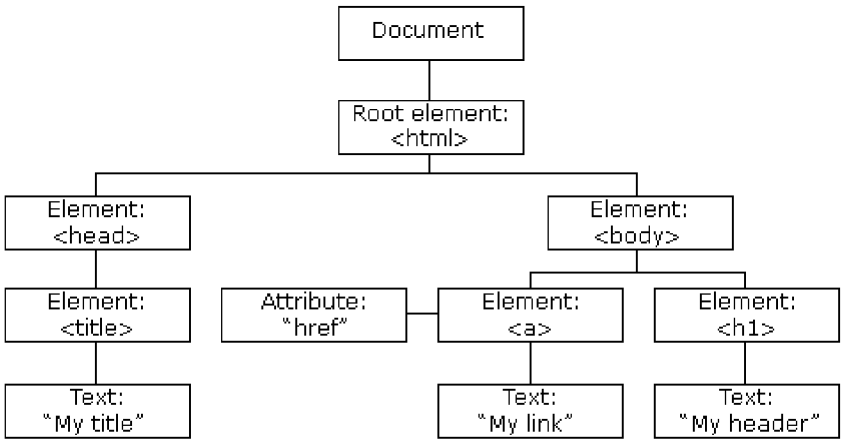


Obrázek 1 - výběr produktu (vlastní zpracování)



Obrázek 2 - modální okno (vlastní zpracování)

K použití této metody je tak potřeba prohlížeč, který umožňuje zpracovávat skripty na straně klienta. Takovými prohlížeči může být například Chrome, Firefox, Edge, nicméně pro hromadné zpracování dat není používání klasických prohlížečů ideální. Proto se v souvislosti s web scrapingem často využívá takzvaných headless prohlížečů, tedy prohlížečů bez grafického rozhraní. Zde se na řadu dostává již samotný DOM – jedná se o standard, který popisuje, jakým způsobem snadno přistupovat k HTML a měnit jej. V DOM je celé HTML převedeno do objektů. Tyto objekty pak mají své atributy a metody, kterými je možné k jejich vlastnostem přistupovat. DOM je využíván právě JavaScriptem k dodávání dynamického obsahu. Výhodou této metody je tak oproti HTML analýze schopnost získávat takřka veškerý obsah webu, to však za cenu vyšší komplexnosti řešení. Ukázka rozložení prvků do DOM struktury:



Obrázek 3 - HTML DOM [4]

3.3 Omezení web scrapingu

Ačkoliv získávání veřejně dostupných dat z webu není nelegální, jak je možné se dočíst ve směrnici 2019/790, kterou přijal Evropský parlament a Rada v roce 2019 [5]. je vhodné při používání těchto metod dostát určitým etickým zásadám. Existuje celá řada důvodů, proč provozovatelé webu nechtějí, aby byla jejich data zpracovávána web scrapery. Jako první důvod se nabízí soukromí – mnozí uživatelé jednoduše nechtějí, aby jejich data byla dále zpracována, ačkoliv mohou být veřejně dostupná. Tím hlavním důvodem však většinou bývá vysoká zátěž serveru, kterou mohou

způsobit stovky dotazů, přicházejících na server během scrapingu. Z těchto důvodů by mělo být ke každému webu přistupováno individuálně a před tím, než dojde ke scrapingu, danou doménu důkladně prozkoumat. V další části následuje výčet nejobvyklejších způsobů, kterými se weby brání před boty.

3.3.1 Robots.txt a případy užití

Nejjednodušším způsobem, jak návštěvníkům webu předat pravidla, jakým způsobem přistupovat k webu, je pomocí případů užití. V angličtině je možné se s nimi setkat pod zkratkou TOS (Terms Of Service) a dále pak souboru robots.txt.

Případy užití se často objevují v patičkách, modálních oknech anebo na samostatných stránkách k tomu určených. Ačkoliv je z etických důvodů správné řídit se pravidly v nich popsanými, pro web takřka neexistuje způsob, jak dodržování těchto pravidel vymáhat. Tato pravidla jsou pak většinou psána v lidsky čitelné podobě, jsou tak určena převážně běžným lidským uživatelům internetu, nejsou tedy vhodná pro zpracování boty.

Tento problém je řešen souborem robots.txt. Následující text čerpá z oficiální dokumentace robots [6]. Jedná se o soubor, který udává pravidla, jak a ke kterým částem webu mohou boti přistupovat. Ten je běžnou součástí mnoha webů - `www.google.com/robots.txt`

Jedná se o text, který by měl dodržovat takzvaný Robots Exclusion Standard proto, aby jej bylo možné snadno zpracovat právě boty.

Pravidla pro vytváření robots.txt:

- soubor se musí jmenovat robots.txt,
- web může mít jen jeden tento soubor,
- soubor musí být umístěn v kořenovém adresáři,
- soubor musí být kódován formátem UTF-8.

Každý soubor může obsahovat skupinu nebo více skupin pravidel. Každá z těchto skupin začíná výrazem `User-agent` a obsahuje výčet souborů, ke kterým bot může a nemůže přistupovat. `User-agent` je součást HTTP požadavku, která nese

informace o zařízení, které tento požadavek vyslalo. Pravidla jsou pak trojího druhu:

- `disallow` – tímto pravidlem se popisují soubory, které by bot neměl navštěvovat
- `allow` – tímto pravidlem se popisují soubory, které bot může navštěvovat
- `sitemap` – tyto soubory by bot měl navštívit. Uživatel touto direktivou botům sděluje, jaké informace sám považuje za důležité a umožňuje tím efektivnější zpracování webu.

Následující sekce je věnována ukázkám základních nastavení pravidel robots.txt.

- Toto pravidlo zakazuje všem typům robotů procházet celý web:

```
User-agent: *
Disallow: /
```

- Pravidlo zakazující všem typům robotů procházet obsah konkrétně zmíněných adresářů:

```
User-agent: *
Disallow: /skola
Disallow: /zprávy/domov/
```

- Následující kód ukazuje více skupin pravidel:

```
User-agent: *
Disallow: /skola
Disallow: /zprávy/domov/

User-agent: Googlebot
Allow: /
```

- Pravidla je možné také nastavit tak, aby ovlivňovala přístup jen k určitým typům souborů:

```
User-agent: *
Disallow: /*.jpg$
```

Stejně, jako je tomu u případů užití, ani robots.txt nedokáže tato pravidla vymáhat. Ačkoliv existuje mnoho knihoven pro scraping, které defaultně tato pravidla dodržují, lze toto nastavení ve většině z nich změnit. Případně použít vlastní nástroj, který může soubor zcela ignorovat. V rámci webové etiky je však důrazně doporučováno tato pravidla dodržovat. Dále už následují způsoby, určené k odhalování botů. Ty dokážou jejich práci skutečně ovlivnit.

3.3.2 Kontrola hlavičky

Každý dotaz na webový server obsahuje hlavičku . Dotazy odesílané z běžného prohlížeče dodržují určitý formát. Anti-bot techniky vyhledávají dotazy, které těmto běžným formátům neodpovídají a tyto dotazy blokují. Jednoduchým způsobem, jak tuto ochranu obejít, je upravit hlavičku dotazu tak, aby odpovídala některému z těchto běžných formátů. Dalším způsobem je využití headless prohlížeče, tato technika však bývá výpočtově mnohem náročnější.

3.3.3 Kontrola user-agenta

Jak už bylo zmíněno výše, user-agent je textový řetězec, který obsahuje informace o zařízení odesílajícím dotaz na server. Většinou obsahuje informace o prohlížeči, jeho verzi a operačním systému, na kterém je prohlížeč spuštěný. Tento řetězec je součástí hlavičky dotazu. Skriptem odesílané dotazy přirozeně tyto informace defaultně neobsahují. Dotazy, v nichž se vyskytují neobvyklé user-agenty, jsou detekovány anti-bot systémy a blokovány. Jednoduchým způsobem, jak toto omezení obejít, je vlastním nastavením tohoto řetězce v kódu scraperu. Takto vypadá současná podoba běžného user-agenta Chrome prohlížeče pro Windows 10:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/102.0.5005.61/63 Safari/537.36
```

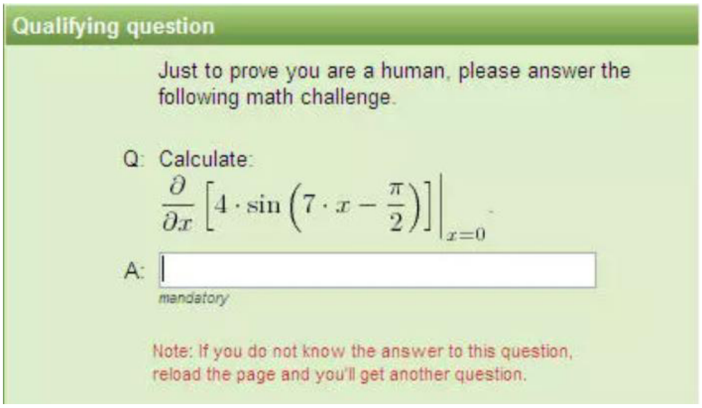
Ještě efektivnějším způsobem vyhýbání se detekci tohoto typu je takzvaná rotace user-agentů, kdy dochází k častému obměňování tohoto řetězce a server se tak domnívá, že dotazy přicházejí z rozličných zařízení.

3.3.4 Kontrola IP adresy

Dalším způsobem, jakým může dojít k odhalení bota, je pomocí kontroly IP adresy. Přesněji řečeno kontroly chování zařízení s danou IP adresou. Scrapovací nástroje nezřídka vysílají stovky až tisíce dotazů na jeden server. To buď ve velmi krátkém časovém úseku nebo v pravidelných cyklech. Anti-bot systémy posuzují, zdali takové chování může odpovídat lidskému uživateli. Pokud odhalí, že tomu tak není, adresu, která vykazuje známky neobvyklého chování, blokuje. Pro scraper tak vzniká problém. Existuje několik způsobů, jak se takové detekci vyhnout. Nejjednodušším způsobem je snížení počtu dotazů, které jsou na server odesílány. Pokud si tvůrce programu dává za cíl získat data o stovkách produktů, bez stovek dotazů se neobejde. V takovém případě nezbývá než upravit práci scraperu tak, aby více odpovídal chování lidského uživatele. Jedním ze způsobů, jak toho docílit, je snížit frekvenci odesílání dotazů. Namísto deseti dotazů za sekundu, tak posílat jen dva. I právě touto pravidelností dotazů může být ale anti-bot nástroji scraper odhalen. Dalším způsobem, jak zabránit této detekci, je nastavit nepravidelnou délku časového úseku mezi odesláním dotazů. Pokud dochází k odhalení i v tomto případě, nezbývá než pravidelně měnit IP adresu zdrojového zařízení. K tomu existuje celá řada nástrojů – přes VPN, proxy služby až po cloudová řešení. Kombinace výše popsaných metod drasticky snižuje možnost blokace scraperu.

3.3.5 Captcha

Je zkratkou pro Completely Automated Public Turing test to tell Computers and Humans Apart. Jak už anglický název napovídá, jedná se o nástroj, který slouží k automatickému odlišování lidí od počítačů. Vyskytuje se v mnoha podobách, od jednoduchých vyskakujících oken vyžadujících potvrzení o člověčenství až po složité mini-hry, jejichž úspěšné splnění slouží jako dostatečný důkaz lidského návštěvníka webu.



Obrázek 4 – Ukázka captchy [7]

Captcha se však většinou objevuje až jako důsledek předchozí detekce. Pokud jsou tak dodržovány výše popsané postupy, neměla by se na webu objevit. V opačném případě i tak existují řešení, jak se s tímto nástrojem vypořádat. V současnosti je možné se setkat s celou řadou knihoven, které se specializují na úspěšné splnění captchy. Jejich užívání bývá však většinou placené, případně může být jejich implementace složitá. Proto je vždy lepší pokusit se této formě blokace vyhnout.

3.3.6 Přihlášení

Další způsob, jakým weby omezují možnost botů získávat data, je jednoduše pomocí nutnosti registrace, kdy jsou všechny podstatné informace zobrazovány až přihlášeným uživatelům na základě cookies, které jsou používány k identifikaci uživatele. I tento zdánlivě neřešitelný problém má však své řešení. Jedním ze způsobů, jak toto omezení obejít, je simulací přihlášení. Dnes existují desítky nástrojů, které uživateli umožňují vytvořit posloupnost kroků, které jsou následně vykonány tak, jak jim uživatel zadal. Tyto technologie jsou například schopné simulovat vyplnění formulářů a kliknutí na tlačítko odeslat, tak aby došlo ke skutečnému přihlášení. Jedním z takových nástrojů je například Selenium, které se používá především pro automatické testování webových aplikací, nicméně našlo své využití i na poli web scrapingu. Těchto technik se také často využívá ve spojení s již zmiňovanými headless prohlížeči. Nevýhodou tohoto řešení je však nutnost dobré znalosti zkoumané domény a jeho značně omezená znuvupoužitelnost. Při

scrapingu webů, které vyžadují přihlášení, případně jakoukoliv jinou aktivitu uživatele, je nutné přistupovat ke každému webu individuálně. Často je nutné vytvářet jiné posloupnosti kroků pro každý web. Dalším řešením nutnosti přihlášení, je získání cookies standardní cestou – tvůrce scraperu se jednoduše skutečně na web přihlásí. Jeho zdrojové zařízení pak obdrží informace cookies pro autentizaci. Tyto cookies následně umístí do svého scraperu. Při každém dotazu pak dochází k autentizaci pomocí těchto cookies tak, jako kdyby se jednalo o dotazování běžného lidského uživatele. Informace z podobného webu pak mohou být po určitou dobu platnosti cookies získávány bez omezení. I pro tuto metodu však platí omezení, obdobná těm, která platí pro simulační nástroje. Ke každému webu musí být přistupováno individuálně, tato metoda pak navíc vyžaduje interakci reálného uživatele.

3.4 Analýza existujících řešení

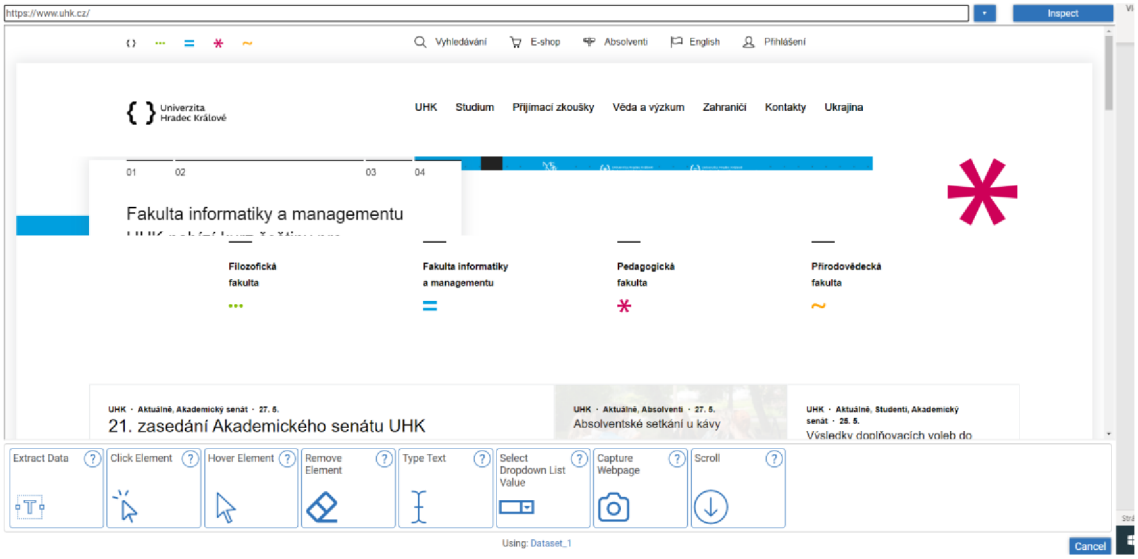
Software pro web scraping se dá v zásadě rozdělit do dvou kategorií. Jedná se o buď o nástroje, které jsou spuštěny na lokálním zařízení a využívají lokálních zdrojů, anebo nástroje cloudové, ke kterým je přistupováno například pomocí webového prohlížeče, případně jeho rozšířeními. Při výběru vhodného nástroje je třeba mít na paměti celou řadu kritérií, která ovlivňují výběr nástroje, a to zejména na základě oblasti jejich využití. Jedná se například o možnosti překonávání bot detektorů, které byly popsány v předchozí kapitole. Existuje celá řada web scraperů, které nabízí sofistikované postupy, jak detekci předcházet. Tato služba je však často vykoupena vyšší cenou. Dalším kritériem výběru softwaru je pak tedy cena. Nástroje se pohybují od těch, které jsou zadarmo, přes nástroje za jednorázový poplatek, až po nástroje fungující na bázi měsíčních poplatků. Dále se nástroje liší v množství stránek a času, za který jsou schopné tyto stránky zpracovat. Některé umožňují jen jednorázový scraping, další možnost periodického procházení stránek. Jedním z důležitých aspektů web scraperů je také schopnost exportovat data v takové podobě, která koncovému uživateli vyhovuje. Kritérií, na základě kterých vybírat nástroj, je mnoho. Nejdůležitější je tak především určit, pro jaké účely bude software používán. Například nástroj, který je vhodný pro získávání e-commerce dat, nemusí

být efektivní pro stahování mailů. V následující analýze bude vždy popsán zástupce z jedné ze tří kategorií nástrojů. Těmito kategoriemi jsou webové aplikace, programy vyžadující lokální instalaci a rozšíření pro prohlížeče.

3.4.1 Grabzit

Je platforma, která nabízí jak API, kterou lze intergrovat do vlastního scraperu, tak přímo webovou aplikaci, ke které lze přistupovat běžným prohlížečem. Nástroj je ve svém základním provedení zdarma. Nicméně existuje i v placeném provedení, které na rozdíl od neplaceného umožňuje procházet větší množství webů měsíčně a také například přidání vlastní proxy adresy. V následující analýze bude popsána především webová aplikace v provedení zdarma. [8]

Před použitím tohoto nástroje je nutné se zaregistrovat na oficiálních stránkách výrobce. Program totiž umožňuje mimo jiné nastavit pravidelné procházení webů a získávání dat z nich na základě uživatelem stanoveného harmonogramu. Tato data jsou následně ukládána na server a přihlášený uživatel k nim může kdykoliv přistupovat. Nástroj pak nabízí možnost zvolit si mezi různými formáty exportu. Namátkou je možné zmínit CSV, XML nebo JSON. Pro nastavení samostatného scrapování slouží sofistikovaný nástroj, který nabízí rozsáhlou škálu možností pro manipulaci s webovou stránkou. Grafické rozhraní stránky se zobrazuje ve speciálním okně, ve kterém lze následně vybírat jednotlivé prvky pouhým kliknutím. Což je pro uživatele bez hlubších znalostí programování a fungování webových stránek značné plus. Nicméně výběr prvku ve většině případů vede k nekonečnému proklikávání oken nastavení a práce s tímto nástrojem se tak po čase stává velmi unavující. Dalšími možnostmi práce s webem je pak možnost interakce s prvky, které vyžadují uživatelovu akci. Interakce s těmito prvky, může následně přinášet další obsah, k jehož načtení při prvním dotazu na server webové stránky nemuselo dojít.



Obrázek 5 - Prostředí Grabzit (vlastní zpracování)

Aplikace umožňuje vytvářet na základě uživatelova výběru prvků, ze kterých chce vytvářet data, šablony. Tyto šablony lze následně aplikovat na více zvolených webových stránek bez nutnosti upravovat je pro každou stránku zvlášť. Šablony je tak možné využít například na hromadné získávání cen produktů v jednom konkrétním internetovém obchodě.

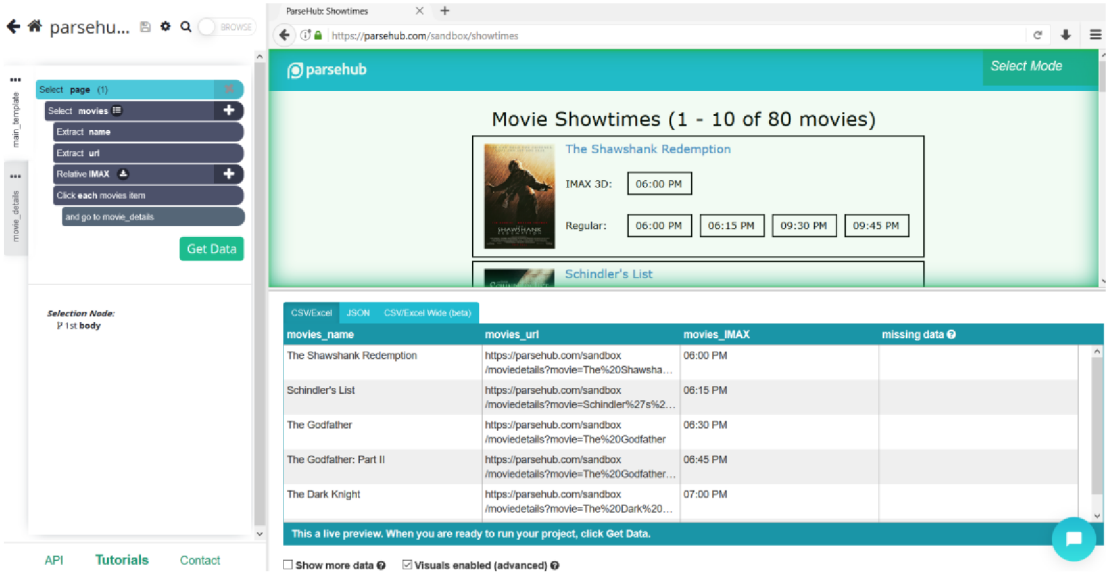
Nevýhodou tohoto nástroje je bezesporu nemožnost nastavení proxy adres v základní verzi, jak již bylo zmíněno výše. Další metody pro předcházení detekce anti-bot nástroji také nejsou implementovány, jelikož většina testovaných webů namísto běžného obsahu vracela jako výsledek captchu, jak ukázal test několika e-commerce webů. Placené verze již některé z metod popsaných v předchozí kapitole nabízejí. Za další velkou nevýhodu pak autor považuje relativní složitost nástroje. Ačkoliv by měl být tento nástroj určen i pro začátečníky, pochopení některých mechanismů nutných pro provedení i základního získávání dat může zabrat i desítky minut.

3.4.2 ParseHub

Stejně jako předchozí zmíněný nástroj, i ParseHub nabízí jak API, které je možné integrovat do vlastního řešení, tak samotný scrapingový nástroj. Na rozdíl od

předchozího řešení je však ParseHub lokálně běžící aplikace. Aplikace je nabízena v několika variantách, cenově odstupňovaných na základě nabízených funkcí. Verze se liší zejména v možnostech překonávání detekce botů a počtu a trvání scrapingu. Nejzákladnější verze nenabízí žádnou z možností obcházení botů, umožňuje procházet maximálně 200 stránek pro jeden projekt a jejich průchod trvá 40 minut. Základní verze pak oproti předchozímu nástroji nenabízí ani možnost naplánovat pravidelný scraping. Naopak nejdražší, profesionální verze, touto možností disponuje. Proti detekci se dokáže bránit například rotací IP adres. Pro jeden projekt umožňuje procházet neomezené množství stránek s tím, že průchod každých 200 stránek vyžaduje 2 minuty. Nejzákladnější verze je opět zdarma, a právě s tou se autor práce seznámil. [9]

Před použitím tohoto nástroje je opět nutné zaregistrovat se na oficiálním webu aplikace. Následně musí být aplikace stažena a nainstalována. Při jejím spuštění je uživatel proveden krátkým tutoriálem, který vysvětluje základní funkcionalitu systému. Tou je obdobně jako u předchozí aplikace, jak možnost získávat data na základě výběru kliknutím, tak možnost simulace procházení webu pro získávání dalších dat. Tato vybraná data je možné seskupovat do kategorií na základě charakteristik zvolených uživatelem. Data je opět možné volit ze speciálního grafického rozhraní, ve kterém je vyrenderována webová stránka na základě zadané adresy. Oproti předchozímu řešení je však tento nástroj značně intuitivnější a svižnější. Ve chvíli, kdy je uživatel se svým výběrem spokojen a vytvořil šablonu pro výběr požadovaných dat, dochází k procházení zvolených stránek. Výsledky této operace je pak opět možné uložit v několika formátech. Těmi základními jsou CSV a JSON.



Obrázek 6 - Prostředí ParseHub (vlastní zpracování)

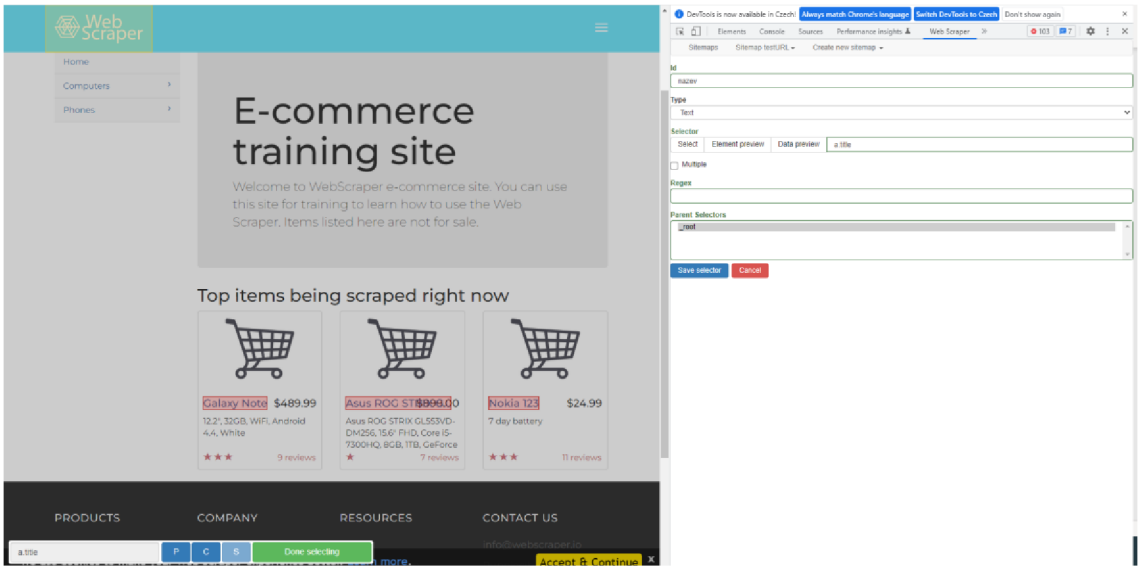
Na rozdíl od dříve zmíněného nástroje implementuje tento software i ve své verzi zdarma základní metody pro obcházení detektorů botů. Nicméně při vyšším počtu dotazů na cílový server i zde dochází k odhalení a omezení přístupu. Placené verze pak tento problém odstraňují například implementací rotace IP adres. Za nevýhodu může být považována nutnost instalace aplikace.

3.4.3 Web Scraper

Dalším nástrojem pro extrakci dat z webu je Web Scraper. V základní verzi se jedná o rozšíření do prohlížeče, které pro svou práci využívá lokálních zdrojů počítače. Nástroj se opět vyskytuje v několika verzích. A stejně jako u předchozích představených scraperů se počet funkcí každé verze zvyšuje společně s výší měsíčních poplatků. [10]

Základní verze, na kterou byla tato analýza zaměřena, nabízí vytváření šablon pro procházení stránek pomocí výběru kliknutím přímo v prohlížeči a spouští se v prostředí vývojářské konzole. Při vytváření šablony nejdříve uživatel zadá URL požadované stránky a zvolí prvek, jehož data mají být získána. Takto získaná data pak mohou být v základní verzi uložena ve formátu CSV. Nicméně placené verze

nástroje umožňují volit mezi více formáty, resp. CSV, XLSX a JSON. Tím ve své podstatě výčet funkcí základní verze končí. Pro překonávání detekce botů nenabízí základní verze téměř žádné nástroje. Pokud je tak šablona aplikována na více stránek, detekci se nevyhne. Tento nedostatek lze obdobně jako u předchozích nástrojů vyřešit přechodem na některou z placených verzí. Předcházení detekcí je v jejich případě řešeno pomocí využívání proxy adres. Placená provedení navíc nevyužívají při scrapingu pouze lokálních zdrojů, ale využívají cloudu. Tyto vyšší verze pak obdobně jako předchozí zmíněné nástroje umožňují využívat API, které je možné implementovat do vlastního kódu.



Obrázek 7 - Prostředí Web Scraperu (vlastní zpracování)

Ve srovnání s již popsanými řešeními, vítězí tento jednoznačně svou jednoduchostí a intuitivností používání. Není vyžadována žádná registrace. Stačí několik málo kliknutí a aplikace je součástí prohlížeče. Samotný výběr prvků je velice přímočarý. Při vyšším počtu zvolených prvků a zejména pak na komplexnějších webech, však může být poněkud nepřehledný. Nemožnost pustit nástroj mimo prostředí vývojářského rozhraní pak mnohdy vede k nešťastným situacím, jelikož vykreslování prvků závisí na velikosti a dalších dispozicích okna, ve kterém jsou zobrazovány.

3.4.4 Výsledky analýzy

Ačkoliv jsou v předchozí analýze hlouběji popsány pouze tři nástroje, analýze bylo podrobeno mnoho dalších z nich podrobněji pak především Octoparase a Data Miner. Pro každou kategorii byl vždy vybrán pouze jeden nástroj, jelikož většina z nástrojů v každé kategorii sdílí jak stejnou funkcionalitu, tak stejné nedostatky. Pouze v oblasti čistě webových aplikací nebyl nalezen další nástroj zdarma, který by se svou funkcionalitou alespoň vzdáleně přibližoval popsanému Grabzit. Alternativou k ParseHub je pak například Octoparse, který se svým hlouběji popsaným konkurentem nabízí takřka stejnou funkcionalitu, jež je vykoupena téměř totožnými nedostatky, tedy nízkou schopností vyhýbat se detekci botů. Ta opět přichází až s placenými verzemi. Na poli rozšíření pro prohlížeče existuje plejáda zástupců, které si s výše podrobně popsaným softwarem v ničem nezadají, patří mezi ně například Data Miner.io, Instant Data Scraper nebo Scraping Agent od společnosti Agenty, ten se však nevyskytuje ve verzi zdarma. Zde pro shrnutí následuje srovnávací tabulka existujících nástrojů, všechny ve své základní verzi. Základní předcházení detekce znamená, že nástroje nevyužívá proxy, ani jiných metod pro skrytí zdrojové IP adresy:

NÁZEV	TYP	PŘEDCHÁZENÍ DETEKCI	PROXY	VÝSTUP
GRABZIT	Webová aplikace	Ne	Ne	CSV, XML, JSON a další
PARSEHUB	Lokální instalace	Základní	Ne	CSV, JSON
OCTOPARSE	Lokální instalace	Základní	Ne	CSV, HTML, JSON, XLSX
WEBSCRAPER	Plugin prohlížeče	Ne	Ne	CSV, JSON, XLSX
DATA MINER	Plugin prohlížeče	Základní	Ne	CSV, XLSX

Tabulka 1 - Přehled existujících řešení (vlastní zpracování)

Během analýzy existujících řešení bylo učiněno několik závěrů. Nástroje, které jsou zdarma a zároveň umožňují kvalitní scraping čistě pomocí webové aplikace, v

podstatě neexistují. Kvalitnější alternativou k nim mohou být rozšíření do prohlížečů. Snadnost jejich uvedení do provozu je však většinou vykoupena sníženou funkcionalitou. Z analýzy vyplývá, že v současnosti nejvhodnějšími nástroji pro kvalitní scraping jsou lokálně instalované aplikace. Právě nutnost instalace a využívání lokálních zdrojů však pro mnohé z uživatelů není vhodná a například v zaměstnáních, kdy je využíváno firemních zdrojů, které jsou často omezovány v možnostech instalace softwarů, takřka nemožná. Dále pak žádný z testovaný nástrojů, a to napříč všemi kategoriemi, nenabízí kvalitní ochranu před detekcí botů, tedy alespoň ve svých základních verzích. Analýza existujících řešení tak dokázala, že na poli web scrapingu je nadále co zlepšovat.

3.4.5 Vývojové prostředky

Pro pokročilejší uživatele, programátory nebo ty, pro které jsou existující popsaná řešení nedostačující, budou dále představeny vývojové prostředky v podobě knihoven, frameworků a dalších rozšíření, které je možné implementovat do programovacího procesu a vytvořit tak vlastní řešení na míru.

- **BeautifulSoup** je knihovna pro jazyk Python, která dokáže modifikovat předaný HTML nebo XML řetězec do podoby objektu. Nad tímto objektem je možné provádět celou řadu převážně filtrovacích a vyhledávácích operací. Jelikož je tato knihovna využita v rámci praktické části práce, je detailněji popsána v následujících kapitolách.
- **Scrapy** je tentokrát již framework jazyka Python, který nabízí komplexní a úplné řešení pro web scraping. Na rozdíl od BeautifulSoup, které slouží jen k práci se samotným kódem webu, řeší Scrapy samo i celou řadu problémů popsaných v předchozích kapitolách. Nabízí mnoho funkcí pro vytváření bezpečných HTTP požadavků aktivně předcházejících detekci. Takto získaná data umožňuje dále zpracovat podobným způsobem jako BeautifulSoup, ten je mimochodem možné zakomponovat do pipeline zmíněného frameworku. Scrapy se navíc samo dokáže postarat i o tvorbu strukturovaného výstupu například ve formátu CSV. [11]

- **Selenium** je další z frameworků hojně využívaných v souvislosti s web scrapingem. Tento framework byl původně navržen pro automatické testování webových aplikací. Umožňuje totiž automatizovat práci webového prohlížeče, tedy například napodobit chování běžného uživatele. Právě díky této vlastnosti je mocným nástrojem web scrapingu. Je možné jej použít s celou řadou programovacích jazyků, může být tak považován za platformě nezávislý. Obdobně jako BeautifulSoup je možné integrovat tento nástroj do vývojového procesu Scrapy. [12]

4 Vlastní implementace web scraperu

Před tím než dojde k popsání samotné implementace, je třeba čtenáře seznámit s cíli a případy užití, pro které bude aplikace navržena.

4.1 Nedostatky existujících nástrojů

Na základě zjištění během analýzy existujících řešení bylo odhaleno několik nedostatků dostupných nástrojů. Jedním z největších nedostatků je podle autora nutnost instalace těchto nástrojů. A to ať formou rozšíření pro prohlížeče, tak formou lokální instalace. Navrhovaný nástroj by tak měl být intuitivní webovou aplikací, kterou bude možné spustit pomocí prohlížeče.

Dalším velkým nedostatkem je bezesporu neschopnost předcházet detekci scrapingu anti-bot nástroji. Autorem navržené řešení by tak mělo právě tento nedostatek odstranit. Vytvořený nástroj by měl využívat technik, které byly popsány v části o omezeních web scrapingu. Řešení by tak mělo využívat především možnost používání proxy adres. Tyto adresy by měly být v nepravidelných cyklech náhodně přepínány tak, aby nebylo možné web scraper odhalit ani pomocí pokročilejších nástrojů analýzy chování uživatele. Ze stejného důvodu by dále mělo dojít k implementaci přepínání user-agentů.

4.2 Nástroje aplikace

Pro zachování co možná nejjednoduššího používání nástroje a jeho přehlednosti se bude jednat o jednostránkovou webovou aplikaci. Tato aplikace by měla poskytovat v zásadě tři hlavní funkce: scraping uživatelem zvoleného webu (návštěvu zvolené URL a získání její struktury) a parsing (extrakci uživatelem zvolených prvků). Pro extrakci bude autorem vytvořen jednoduchý parsovací nástroj, s jehož pomocí bude moci uživatel díky selektorům volit konkrétní prvek, a to ať už celý element včetně tagů, atributů a jeho obsahu, nebo jednotlivé části elementu. Dále by měla aplikace umožňovat export. Tato funkcionalita by měla uživateli dovolit získávat data ve dvou formátech, těmi jsou CSV a JSON.

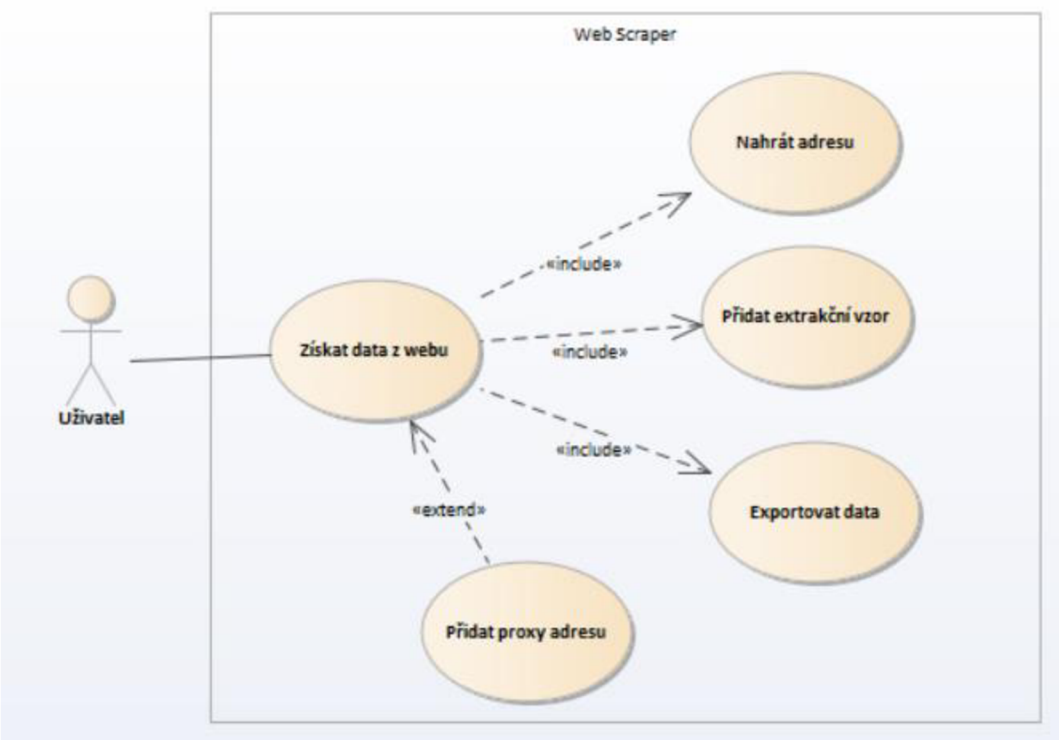
4.3 Používání aplikace

Aplikaci bude možné používat následujícím způsobem. Po načtení základního rozložení webové stránky, které sestává ze čtyř částí, se uživateli zobrazí ukázková struktura webové stránky společně se základním vzorem pro získávání jednotlivých prvků. Po odeslání zmíněného vzoru jsou uživateli předloženy výsledky v podobě jednotlivých získaných prvků. Toto základní zobrazení by mělo sloužit jako návod k použití aplikace.

Po tomto základním seznámení by měla práce uživatele probíhat takto. Do formuláře pro adresy je zadána jedna až libovolné množství webových adres. Při každém zadání je uživateli zobrazena struktura zadaného webu. Mezi adresami je možné libovolně přepínat.

Dalším krokem by pak mělo být zadání vzorce pro získání prvků webu. Po jeho odeslání se uživateli zobrazí výsledky. Je-li vzorec zadán korektně a jsou-li s jeho pomocí získány nějaké prvky, jsou uživateli zobrazeny, v opačném případě se zobrazí chybová hláška. Takto získané výsledky jsou uživateli předány ve formátu JSON, který se dá jednoduše zkopírovat. Případně je možné tyto výsledky exportovat ve formátu CSV. Toto řešení však není vhodné pro hromadné získávání dat, slouží totiž především pro kontrolu získávaných částí webu.

K získávání většího množství dat jsou určena dvě modální okna. První z nich se sestává z formuláře pro nahrání souboru ve formátu CSV, který obsahuje webové adresy, ty budou postupně scrapovány a parsovány. V tomto okně je také možné naleznout vstupní pole pro zadání mailové adresy, na níž budou získaná data zaslána. Další modální okno obsahuje formulář pro přidání proxy adres. Mezi těmito adresami pak bude následně náhodně přepínáno. Zadání proxy adres je nepovinné. Nicméně vynecháním tohoto kroku uživatel riskuje detekci scraperu a následné omezení získávání dat.



Obrázek 8: Use Case diagram základního používání aplikace (vlastní zpracování)

4.4 Architektura

Aplikace bude využívat modelu klient-server. Klient bude využívat lokálních zdrojů uživatele a ke komunikaci se serverem bude využíváno komunikace přes síť pomocí HTTP dotazů. Klientská část pak bude reprezentována pomocí single-page aplikace spuštěné ve webovém prohlížeči. Server bude zprostředkovávat REST API, tedy technologii popsanou v jedné z předchozích kapitol práce.

4.5 Použité technologie

V následující sekci jsou podrobněji popsány technologie, které budou použity při implementaci nástroje.

4.5.1 Python

Python je interpretovaný, objektově orientovaný jazyk. Díky své univerzálnosti se dá použít takřka na cokoliv, od jednoduchých webů až po složité neuronové sítě.

Jedná se o open source projekt, který je podporován většinou běžných platforem – Windows, macOS, Android, Unix. V mnoha žebříčcích se umísťuje na prvních příčkách oblíbenosti programovacích jazyků. Stojí za ním rozsáhlá a velmi aktivní komunita, díky níž umožňuje používat velké množství knihoven a balíčků, které značným způsobem usnadňují vývoj rozličných aplikací. Právě díky těmto vlastnostem byl autorem práce vybrán jako ideální jazyk pro tento projekt. [13]

4.5.2 Flask

Jedná se o webový microframework napsaný v jazyce Python. Micro znamená, že obsahuje jen nezbytně nutnou funkcionalitu pro vytváření webových aplikací, díky tomu je lehký a jednoduchý. Pro funkcionalitu vyšších úrovní spoléhá na rozšíření pomocí balíčků a knihoven. Skládá se ze dvou hlavních komponent. První je Werkzeug, knihovna poskytující rozhraní pro základní komunikaci na webu, routování, debugging a další webovou funkcionalitu. Druhou je Jinja2, která slouží jako šablonový procesor k usnadnění vytváření dokumentů [14]. Další informace je možné naleznout na webu nástroje [15]. Flask byl vybrán zejména pro svou podporu RESTful API. Serverová část práce tak bude fungovat především jako API, která zpracovává a předává data. Prezentační část aplikace tedy bude probíhat zejména v režii klienta. Balíček je instalován pomocí následujícího příkazu: `pip install Flask`

Pro ukázku kód toho, jak málo stačí pro vytvoření jednoduché webové aplikace pomocí Flasku:

```
from flask import Flask
app = Flask(__name__)
@app.route("/"   )
Def pozdrav():
Return "Ahoj, svete!"
```

Zdrojový kód 1: Založení Flask aplikace (vlastní zpracování)

4.5.3 BeautifulSoup

Jedná se o python knihovnu, která slouží k získávání dat z HTML a XML souborů a to tak, že daný řetězec převádí do stromové struktury objektů, mezi kterými umožňuje

snadno procházet, vyhledávat, a dokonce je upravovat. Ke své práci využívá uživatelem zvolený parser. Pokud není uživatelem žádný zadán využije defaultního python parseru, nicméně pro nejvyšší efektivitu této knihovny se doporučuje použití v kombinaci s lxml parserem. BeutifulSoup je v současnosti pravděpodobně nejpopulárnějším nástrojem pro zjednodušení práce s HTML a XML soubory. Stojí za ním aktivní a rozsáhlá komunita. V současnosti se vyskytuje ve verzi 4.8.1, která byla také použita v popisovaném projektu. [16]

Nástroj pracuje s již existujícím HTML nebo XML souborem. Je nutné upozornit, že sám nevytváří dotazy na server. Pro účely web scrapingu je tak nutné jej zkombinovat s nástrojem, který volání na požadovaný server umožňuje. Výsledek tohoto volání je pak BeautifulSoup předán v podobě textového řetězce typu string. Řetězec je následně převeden na strom python objektů, a to pomocí následujícího příkazu:

```
soup = BeautifulSoup(retezec)
```

Tento konstruktor přijímá dva argumenty, prvním je právě textový řetězec obsahující HTML nebo XML, je povinný. Druhým je již zmíněný parser, který je nepovinný.

Knihovna pracuje v zásadě se čtyřmi typy objektů. Těmi jsou Tag, BeautifulSoup, NavigableString a Comment. Pro tuto práci jsou nejdůležitější především první dva zmíněné typy, proto budou krátce představeny.

Objekty typu Tag odpovídají tagům tak, jak se objevují v HTML nebo XML dokumentu. BeautifulSoup umožňuje přistupovat k jednotlivým objektům tohoto typu pomocí názvu tagu. Zde následuje ukázka toho, jak objekt zmíněného typu vytvořit, konkrétně pak bude obsahovat první element typu odstavce:

```
soup = BeautifulSoup(html,"lxml")
tag = soup.p
```

Tagy nabízejí celou řadu metod a atributů, pomocí nichž je možné přistupovat či dále upravovat jejich vlastnosti. Jedním z často používaných atributů je name, ten

umožňuje měnit název tohoto objektu, tedy tag v samotném HTML. Dalším užitečným atributem je attrs. Ten umožňuje přistupovat k atributům tagu. Tyto atributy je pak možné měnit tak, že je k nim přistupováno jako ke klasickému python slovníku.

V této práci dochází především k použití objektů třídy BeautifulSoup. Objekty tohoto typu reprezentují HTML nebo XML jako celek. Slouží tak především k jednoduchému vyhledávání napříč celým dokumentem. K tomu jsou určeny metody *find_all()* a *.find()*. Tyto metody používají jako parametr takřka libovolný textový řetězec, který je obsažený ve zpracovávaném kódu, typicky název tagu, atributu či vnitřního textu elementu. Metody se liší v zásadě jen v návratové hodnotě. *Find()* vrací objekt typu Tag, který odpovídá prvnímu nalezenému tagu na základě zadaného parametru. *Find_all()* vrací set, který obsahuje všechny objekty typu Tag, které odpovídají hledání.

Pro tento projekt je však nejdůležitější metoda *select()*. Ta vrací obdobně jako *find_all()* set objektů třídy Tag, a je tomu tak i v případě, kdy je nalezen jediný unikátní element. Jako parametru využívá řetězce reprezentujícího CSS selektor, to umožňuje jednoznačně přistupovat k jednotlivým elementům, případně ke všem těm, které odpovídají danému selektoru. Díky této vlastnosti je tato metoda vhodná pro použití v kombinaci s nástroji pro vývojáře, které jsou dnes součástí takřka každého běžného prohlížeče. Pro lepší představu bude uvedeno několik příkladů, které se objeví i v samotném projektu.

Předpokládejme následující HTML kód:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
<p>My second paragraph.</p>
<p>My third paragraph.</p>
<div class="block">
    <p>Block of code</p>
    <a href="127.0.0.1">Link</a>
</div>
</body>
</html>
```

Zdrojový kód 2: Základní HTML struktura (vlastní zpracování)

Kód pro získání všech elementů odstavce (<p>) by vypadal následovně:

```
soup.select("p")
```

Tento kód obsahuje následující set objektů typu Tag:

```
[<p>My first paragraph.</p>, <p>My second paragraph.</p>, <p>My
third paragraph.</p>, <p>Block of code</p>]
```

Jak je z ukázky patrné, pomocí toho kódu je možné získat tak, jak říkají pravidla kaskádových stylů, všechny elementy odstavců, a to bez ohledu na jejich zanoření. V tomto případě se tak jedná o určitou obdobu funkce *find_all()*, která byla popsána výše.

Názvy tagů, třídy, identifikátory, atributy a jejich posloupnosti je pak po vzoru CSS selektorů možné libovolně řetězit, tato funkce navíc umožňuje využívat i pseudo selektory, například *:contains()*, s jejich pomocí se pak jedná o opravdu univerzální nástroj.

Element odstavce obsahující text „Block of code“ je možné naleznout několika způsoby. Jedním z nich je využití znalosti jeho rodičovského elementu, tedy blokového elementu s tagem :

```
soup.select(" div > p")
```

V případě jednoduchého HTML, jakým je například to v ukázce, je výše uvedený selektor dostačující. V případě komplexnější stránky by však zřídka podobný selektor nalezl unikátní element a došlo by tak k několikanásobné selekci. Pro zajištění unikátnosti je tak vhodné používat vyhledávání pomocí tříd či identifikátorů. Jejich použití by vypadalo následovně:

```
soup.select(".block > p")
```

K identifikaci byla zvolena třída .block, kterou je označen již zmíněný blokový element.

Pro ukázku pak ještě získání elementu na základě znalosti atributu:

```
soup.select("[href]")
```

A výsledek:

```
[<a href="127.0.0.1">Link</a>]
```

V této ukázce je k identifikaci elementu použit jen název samotného atributu. K přesnějšímu určení elementu je však možné použít i hodnotu daného atributu.

Knihovna BeautifulSoup pak nabízí celou řadu dalších funkcí. Pro účely této práce je však popsaná funkcionalita dostačující a k používání schopností nad rámec těch popsaných v ukázkách nedochází.

4.5.4 Javascript

Je multiplatformní, interpretovaný, objektově orientovaný skriptovací jazyk. Nejčastěji se používá při vývoji webových stránek, kdy je jeho intepretace prováděna pomocí webového prohlížeče na straně klienta. V takovém případě slouží především pro práci s grafickým prostředím, nicméně v současnosti je možné setkat se s jeho použitím i na straně serveru, a to především v souvislosti s Node.js. Je však ale možné využít jej i při tvorbě desktopových, či mobilních aplikací. [17]

4.5.5 React

Je open-source javascriptová knihovna pro efektivní a flexibilní vytváření grafického rozhraní. Jedná se o nástroj vyvíjený společností Meta (dříve Facebook). Celé grafické rozhraní je v tomto případě rozděleno do menších komponent, které spolu navzájem komunikují a jsou převáděny do DOM struktury. K vykreslování těchto komponent obvykle využívá jazyka JSX, který se velmi podobá HTML. Komponentou může být takřka jakákoliv část webové stránky. Od jednoduchých tlačítek až po komplexní formuláře. Právě tato technologie bude použita při tvorbě uživatelského rozhraní webové aplikace. [18]

4.6 Implementace serverové části

Jak bylo zmíněno již v předchozích sekcích. Serverová část aplikace je naprogramována pomocí jazyka Python, a to zejména s využitím microframeworku Flask. Jelikož serverová část funguje především jako REST API, bude následující text věnován především popisu přístupových bodů a třídám, k jejichž vytvoření došlo během práce. V některých částech bude pro ukázku přiložen zdrojový kód. Ve většině případů se jedná o speciální typ API tříd, které zajišťují správnou obsluhu požadavků přijatých serverem.

4.6.1 Url

Jedná se o třídu, která slouží k práci s webovými adresami společně s HTML strukturami daných webových stránek. Tato třída obsahuje tři metody.

První z nich je metoda typu GET. Cílem této metody je především předat uživateli data o zadaných webových stránkách. Při prvním spuštění je uživateli předána ukázková webová stránka. Naopak při dalších spuštěních již metoda předává informace o uživatelem zadaných webech. K uchování těchto dat využívá dlouhotrvající session, která je schopna data udržet až na několik dní. Metoda tak slouží pro obnovení dat při znovuspuštění aplikace. K metodě je přistupováno HTTP dotazem typu GET.

Při volání této metody dochází ke kontrole obsahu session s názvem htmls, pokud je tato session prázdná, je použita funkce *addSampleData()* pro naplnění session ukázkovou webovou stránkou. V opačném případě je uživateli předán již existující obsah session. Tato data jsou následně převedena do formátu JSON, pro tyto účely je použita funkce *jsonify()*, která navíc tato funkce zabalí do objektu typu Response. Response slouží k přidání informací o hlavičce, statusu, typu dat a další informace nutné pro vytvoření validní odpovědi pro dotaz.

Další metodou této třídy je metoda typu POST. Tato metoda už v tomto případě, na rozdíl od předchozí zmíněné, přijímá uživatelem zadané parametry. Tyto parametry jsou dva, url, který je typu string a id, ten je typu integer. Jedná se o parametry povinné.

Po úspěšném předání parametrů dochází k volání funkce *rotateHeaders()*. Tato funkce obsahuje několik základních user-agentů. Agenti jsou uloženy v pythonovském listu, z toho je následně náhodně zvolen user-agent. Funkce má v zásadě dva cíle, jak bylo popsáno v kapitole „omezení web scraperů“ jejich obměňování slouží k zamezení detekce anti-bot systémy. Nicméně během práce došlo ke zjištění, že bez jejich použití mnohdy nedochází ke korektnímu zobrazení webových stránek, jsou tak zcela nezbytné pro správné fungování scaperu. Náhodně zvolený agent je následně předán do hlavičky požadavku, který je zaslán pomocí funkce *get()* Python modulu Requests na uživatelem zvolenou adresu.

V tuto chvíli již dochází k použití knihovny BeautifulSoup. Odpověď získaná pomocí modulu Requests je předána jako textový řetězec. K parsování dochází pomocí knihovny lxml. V tomto případě však ještě nedochází k využití žádných filtrovacích funkcí nástroje, které byly popsány v předchozí kapitole. Cílem je v tuto chvíli převést získaný HTML kód na vhodným způsobem modifikovaný textový řetězec tak, aby byl správně čitelný pro koncového uživatele. K tomu je použita funkce *prettify()*, ta je také součástí balíčku BeautifulSoup.

Následně již přichází na řadu dobře známa session htmls. Dochází ke kontrole její existence. Pokud session neexistuje, je vytvořena. V opačném případě je na konec listu, který je obsažen v této session, přidán datový objekt typu slovník. Ten v sobě

nese informace o identifikátoru záznamu, url získané webové stránky, a právě HTML stránky převedené do čitelného formátu.

Uživateli je následně jako odpověď zasláno zmíněné upravené HTML, opět zabalené pomoci funkce *jsonify()*. K metodě je přistupováno HTTP dotazem typu POST. Zde následuje ukázka zmíněného kódu:

```
@cross_origin(supports_credentials=True)
def post(self):
    args = url_put_args.parse_args()
    url = args["url"]
    id = args["id"]
    header = headerRotation.rotateHeaders()
    try:
        result = requests.get(url, headers=header)
        html = BeautifulSoup(result.text, "lxml")
        html = html.prettify()
        if "htmls" in session:
            session["htmls"].append({"html": html, "id": id, "url":
url})
        else:
            session["htmls"] = []
            session["htmls"].append({"html": html, "id": id, "url":
url})
        response = jsonify(str(html))
    except:
        response = "Adresa neodpovídá", 500

    return response
```

Zdrojový kód 3: Metoda POST třídy Url (vlastní zpracování)

DELETE je další metodou této třídy. Tato metoda je určena, jak již název napovídá, k mazání. Je použita pro mazání záznamu o adresách. V tomto případě spíš však záznamu ze session htmls. I tato metoda přijímá parametr, tentokrát identifikátor záznamu v podobě integeru. Po úspěšném předání parametru dochází znovu ke kontrole existence session. Pokud tato session existuje, dochází k její filtraci. Nově je do ní uložen list, který neobsahuje uživatelem zadaný identifikátor, který byl předán v parametru. K metodě je přistupováno HTTP dotazem typu DELETE. Přístupovou adresou pro metody třídy Url je **/url**.

4.6.2 Patterns

Tato třída je svou funkcionalitou a použitím velice podobná předchozí zmíněné třídě, zejména její GET metodě. Za pomoci session s názvem patterns jsou udržována

data o uživatelem vytvořených extrakčních vzorech pro selekci prvků webové stránky. Data z této session jsou pak obdobně jako v předchozím případě předána HTTP dotazovou metodou typu GET, a to na URL **/getPatterns**.

Stejně jako v předchozím případě dochází ke kontrole obsahu a existence session s názvem patterns. Pokud tato session neexistuje, případně je-li prázdná, je naplněna ukázkovým vzorem. V opačném případě je naplněna vzorem, který byl již zadán uživatelem. I návratový typ je stejný, jedná se o JSON zabalený do objektu Response, dochází tedy opět k použití funkce *jsonify()*.

4.6.3 Proxy

Proxy je další API třídou. Má dvě metody. První z nich je typu POST, ta přijímá jako parametr list s adresami a případnými autentizačními údaji. Tento řetězec je ukládán do session proxys. Dále obsahuje metodu typu GET, která slouží k načtení uživatelem zadaných adres. K metodám této třídy je přistupováno pomocí adresy **/proxy**.

4.6.4 ProxyChecker

Je API třídou, která slouží k ověření dostupnosti proxy adresy. Obsahuje jednu metodu typu GET, ta jako vstupní parametr přijímá datový typ slovníku – dict. Z těchto dat následně pomocí funkce *getProxy()* získává proxy adresu ve vhodném formátu. V případě, kdy se jedná o adresu vyžadující ověření, musí dojít k úpravě adresy, tak aby vyhovovala vstupním kritériím používaného proxy serveru. Pro ověření dostupnosti je následně vyslán dotaz na náhodný server pomocí funkce *get()* knihovny Requests.

4.6.5 Parser

Parser již slouží k samotnému získávání jednotlivých informací z webové stránky. Využívá k tomu metodu typu POST. Tato metoda přijímá parametr, kterým je vzorec

zadaný uživatelem. Tento vzorec je nejdříve uložen do session patterns, jejím cílem je především zabránit ztrátě dat při nechtěném zavření stránky. Následně opět dochází ke kontrole session htmls. Pokud je prázdná, klient obdrží odpověď ve formě prázdného listu. V opačném případě dochází k využití funkce *getAllPatterns()*, která přijímá parametr v podobě vzorce zadaného uživatelem a jejím cílem je modifikovat jej do podoby vhodné pro použití ve spojitosti s knihovnou BeautilfulSoup. Vrací set objektů třídy Pattern, ta bude podrobněji popsána v další podkapitole.

Následuje cyklus, který informace z každého prvku tohoto setu využívá pro získání dat z každé adresy uložené v session htmls. Jedná se tak o cyklus zanořený v cyklu. K získávání těchto dat se používá třídy Result. Ta bude také popsaná později. Návratová hodnota metody této třídy je následně uložena do slovníku společně s uživatelem zloveným názvem a identifikátorem získaných dat. Identifikátor je zvolen na základě dat ze session htmls. Název je získán právě pomocí třídy Patterns.

Výsledek je opět předán ve formátu JSON a to opět za použití funkce *jsonify()*. Tato funkcionalita je dostupná na adrese **/parser**.

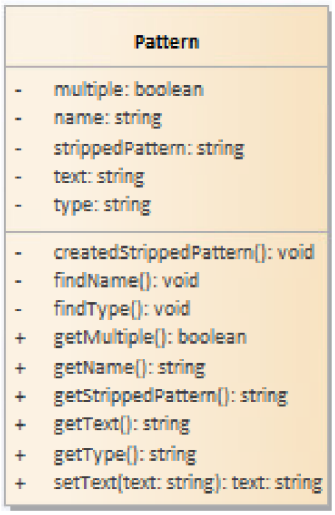
4.6.6 Pattern

V tomto případě se nejedná o žádnou z API tříd, nepřijímá tak žádné HTTP dotazy jako předchozí zmíněné. Jejím úkolem je zpracovat uživatelem zadaný vzorec a připravit jej pro předání knihovně BeautifulSoup. Konstruktor přijímá jediný parametr, tím je textový řetězec obsahující právě daný vzorec neboli pattern.

Atributy této třídy jsou text, name, type, strippedPattern a multiple. Všechny tyto atributy jsou datového typu string až na multiple, který v sobě nese boolean.

- Text je řetězcem vzorce zadaným přímo uživatelem, je následně zpracováván.
- Name slouží k pojmenování uživatelem vybraného prvku.
- Type určuje, o jaký typ prvku se jedná, může to být například vnitřní text elementu, celý element včetně tagů nebo například hodnota atributu třídy.

- StrippedPattern je samotný CSS selektor připravený pro předání pro BeautifulSoup.
- Multiple značí, zdali se jedná o jeden či více prvků webové stránky.



Obrázek 9: Model třídy Pattern (vlastní zpracování)

Ve této třídě je následně několik metod, které slouží k naplnění již zmíněných atributů. Jsou to privátní metody *findName()*, *findType()* a *createStrippedPattern()*, volají se již při vytváření objektu. Pomocí těchto metod je uživatelem zadaný řetězec rozdělen na menší části, ze kterých jsou následně získány nezbytné informace. Pro ukázku je přiložen kód metody *findName()*:

```
def __findName(self):
    if self.__text.find("==>") > -1:
        self.__name = self.__text.split('==>',1)[1].strip()
        self.__multiple = False
    elif self.__text.find("===") > -1:
        self.__name = self.__text.split('===', 1)[1].strip()
        self.__multiple = True
    else:
        self.__name = "ERROR: Vzor nebyl zadán ve správném formátu"
```

Zdrojový kód 4: Metoda findName() (vlastní zpracování)

Tato metoda slouží k získání jména a typu prvku. V atributu text dochází nejdříve pomocí metody *find()* k hledání řetězce „===“ nebo „==>“, očekává se, že za těmito řetězci bude uveden název prvku. Na základě toho, o jaký z řetězců se jedná, se také určuje typ prvku. Pokud se jedná o „==>“ prvek je jen jeden, naopak pokud je

řetězcem „===“ očekává se větší množství prvků stejného typu. Pokud není žádný z těchto řetězců v atributu přítomen, je jméno změněno na chybovou hlášku.

4.6.7 Zápis extrakčních vzorců

V tuto chvíli je vhodné popsat, jakým způsobem funguje speciální zápis pro vybírání prvků webové stránky. V ideálním případě by měl být každý prvek (či skupina prvků) popsán na jediném řádku. Začátek vzoru pro každý prvek je označen direktivou „select:“, po ní již následuje CSS selektor. K získání tohoto selektoru je vhodné využít nástrojů pro vývojáře, které jsou součástí většiny moderních prohlížečů. Je doporučeno používat webový prohlížeč Google Chrome. Ve chvíli, kdy je prvek zvolen, je dále možné určit, jaká hodnota bude extrahována. Může se jednat buď o vnitřní text, hodnotu atributu či celý element včetně tagů. Tuto volbu je možné určit následujícími způsoby:

- Chce-li uživatel získat vnitřní text elementu, následuje po CSS selektoru direktiva „text“, vzorec pro takový výběr vypadal takto:
  ```
  select: title >>> text ==> titulek;
  ```
- Je-li cílem získat hodnotu konkrétního atributu prvku, je nutné použít direktivu atr(), která jako vstupní parametr přijímá název atributu, kód vypadá následovně, v tomto případě se jedná o atribut třídy:
  ```
  select: title >>> atr(class) ==> titulek;
  ```
- K získání celého elementu včetně tagů dojde pomocí vynechání celé této části:
  ```
  select: title ==> titulek;
  ```

Tento zápis navíc dále umožňuje prvky seskupovat. Ve chvíli, kdy je zvolen prvek i jeho typ, je možné určit, zdali dojde ke zpracování všech prvků, které odpovídají hledanému selektoru, případně zdali bude vybrán jen první z nich. K tomu slouží řetězec „==>“, respektive „===“. První zmíněný řetězec značí zvolení jen jediného prvku. Druhý řetězec hledá všechny prvky, které odpovídají danému selektoru. Jejich použití:

```
select: title ==> titulek;
```

a

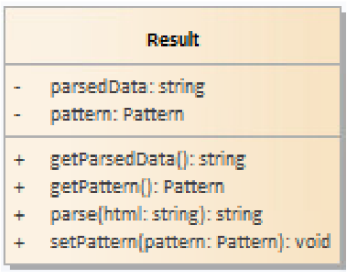
```
select: title === titulek;
```

Po této přiřazovací části je nutné zadat název, kterým bude prvek označen. Následně je nutné celý vzorec ukončit středníkem.

4.6.8 Result

Druhou třídou, která nepatří mezi API je Result. Tato třída již slouží k samotnému vyhledávání prvků na webové stránce. Má dva atributy. Jedním z nich je objekt typu Pattern, ten je předáván již při vytváření Resultu, je tedy konstruktorovým parametrem. Druhým atributem je parsedData, jeho hodnota je získávána pomocí metody, která je popsána níže.

Metoda *parse()* přijímá jeden vstupní parametr typu string, tím je html. Html je kód webové stránky ze session htmls. Z proměnné html je pak již v rámci samotné metody vytvářen objekt třídy BeautifulSoup. Na základě atributu type poskytnutého objektem Pattern je pak určeno, jaký typ prvku uživatel zvolil, a podle toho jsou zvoleny funkce knihovny BeautifulSoup, které budou použity. V zásadě však dochází převážně k použití funkce *select()*, která může být podrobena další filtraci. Výsledkem této operace je vždy list prvků. Pomocí atributu multiple je dále určeno, kolik prvků tohoto listu je vráceno. Dále je již tato hodnota uložena do proměnné parsedData.



Obrázek 10: Model třídy Result (vlastní zpracování)

Kód metody *parse()*:

```
def parse(self, html):
    html = BeautifulSoup(html, "lxml")
    try:
        if (self.__pattern.type != ""):
            if (self.__pattern.type == "text"):
                self.__parsedData = [a.text.strip() for a in
html.select(self.__pattern.strippedPattern)]
            else:
                self.__parsedData = [a.get(self.__pattern.type) for a
in

html.select(self.__pattern.strippedPattern)]

                if (len(self.__parsedData) > 0 and
self.__parsedData[0]):
                    if isinstance(self.__parsedData[0], list):
                        newList = []
                        for r in self.__parsedData:
                            if (r is not None):
                                if (r[0]):
                                    newList.append(r[0])
                                    self.__parsedData = newList
        else:
            self.__parsedData = [str(a) for a in
html.select(self.__pattern.strippedPattern)]

        if not self.__pattern.multiple and self.__parsedData:
            self.__parsedData = self.__parsedData[0]
    except Exception as ex:
        self.__parsedData = "ERROR: Vzor nebyl zadán ve správném
formátu"

    if self.__parsedData is None:
        self.__parsedData = []

    return self.__parsedData
```

Zdrojový kód 5: Metoda parse() (vlastní zpracování)

4.6.9 File

File je pravděpodobně nejdůležitější třídou celého projektu. Je tomu tak především protože spojuje většinu výše popsaných tříd. Obsahuje jednu metodu typu POST. Tato metoda přijímá tři parametry, těmi jsou textový řetězec reprezentující mail uživatele, datový soubor formátu CSV a informace o formátu, v jakém bude proveden export. Metoda je přístupná na adrese **/uploadFile**.

Prvním krokem metody po načtení parametrů je získání webových adres z nahraného souboru. Tyto adresy jsou následně předány funkci *getAllResults()*,

která je určena ke zpracování většího množství adres. Nejdříve načte vzory ze session patterns. Následně, pokud existují, jsou načteny proxy adresy ze session proxys. Tyto proxy se při každém požadavku na server zvolené webové stránky náhodně střídají. Celá funkce pak funguje podobným způsobem jako ta popsaná v kapitole Parser. S tím rozdílem, že v tomto případě dochází při každém zavolání této metody k vytvoření nového vlákna, jelikož se jedná o dlouhotrvající proces, který by mohl bránit ve zbylé funkcionalitě aplikace. Na každou adresu jsou aplikovány uživatelem zadané vzory. Výsledky této operace jsou pak ukládány do setu. Daný set je následně vrácen zpátky původní metodě v objektu typu JSON.

Zmíněný objekt je dále předán funkci *newMail()* , která slouží k vytvoření mailu. Přijímá dva parametry, již zmíněný JSON a dále pak mailovou adresu uživatele. Na tuto adresu je následně odeslán JSON řetězec a CSV soubor s výsledky dané operace. K vytvoření zmíněného CSV souboru slouží knihovna Pandas.

4.6.10 Seznam přístupových bodů

K serveru je možné přistupovat pomocí těchto adres a metod:

ZDROJ	ADRESA	METODY
URL	/url	GET, POST, DELETE
PARSER	/parser	POST
PATTERNS	/getPatterns	GET
FILE	/uploadFile	POST
PROXY	/proxy	POST, GET
PROXYCHECKER	/checkProxy	GET

Tabulka 2: Seznam přístupových bodů (vlastní zpracování)

4.7 Implementace klient části

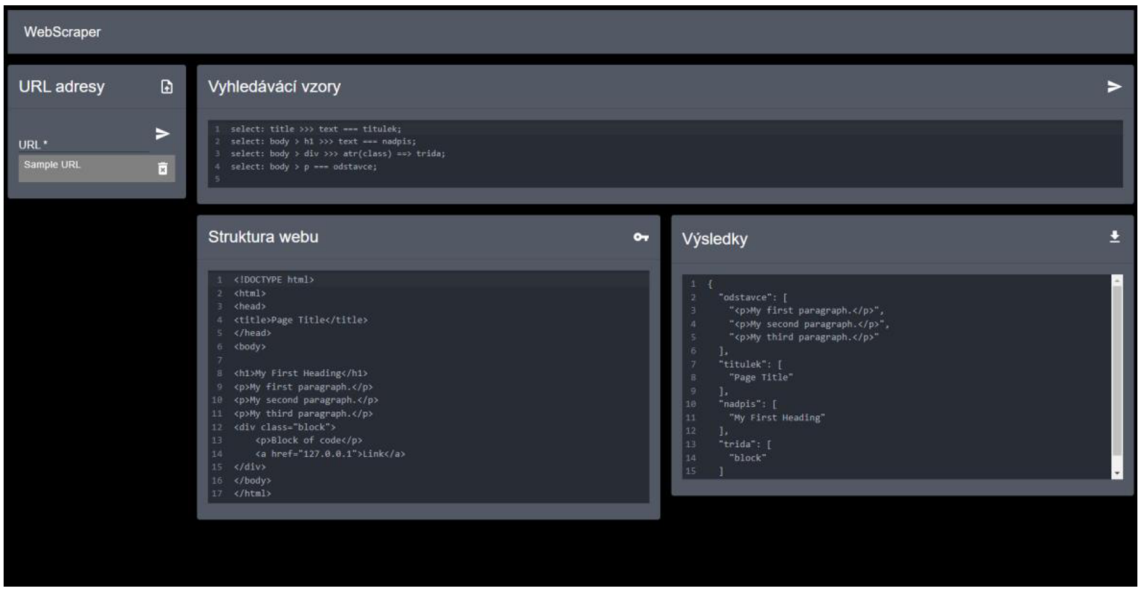
Část klienta je celá psána pomocí jazyka JavaScript, a to za významného přispění knihovny React. Jak bylo popsáno v kapitole popisující tuto knihovnu, grafické rozhraní se v jejím podání skládá z komponent. Tyto komponenty jsou v podstatě

speciálním typem funkce. Jelikož se jedná převážně o kód popisující vzhled dané komponenty, nedojde k jejich detailnímu popisu a blíže přiblíženy budou jen zajímavé části.

4.7.1 Základní rozložení aplikace

Jak již bylo zmíněno v předchozích kapitolách, klient je single page aplikace, to znamená, že její obsah je tvořen jedinou stránkou. Data jsou zobrazována dynamicky bez nutnosti překreslování celé webové stránky. Rozhraní aplikace je rozloženo do základních čtyř částí:

- První částí je levý panel, který slouží k přidávání a zobrazení adres zadaných uživatelem. Mezi těmito adresami lze libovolně přepínat. Přepnutí vede k zobrazení výsledků a struktury dané webové stránky. Tento panel umožňuje zároveň spustit modální okno, to slouží jako formulář pro zadání mailové adresy a nahrání CSV souboru s webovými adresami.
- Druhá část slouží k zadávání extrakčních vzorů. Tyto vzorce jsou určeny k identifikaci prvků webové stránky. Po jejich odeslání jsou vybrané prvky zobrazeny v části s výsledky.
- Třetí část zobrazuje strukturu webu právě vybrané webové adresy. Zároveň také slouží k otevírání modálního okna pro přidávání proxy adres.
- Čtvrtá část zobrazuje prvky nalezené pomocí zadaného vzorce. Její obsah se mění na základě právě zvolené webové adresy. Tato část zároveň umožňuje stáhnout CSV soubor se všemi nalezenými prvky.



Obrázek 11 - Rozložení aplikace (vlastní zpracování)

4.7.2 Komponenta App

Komponenta App může být považována za základní stavební kámen celé aplikace, viz příklad níže. Zastřešuje totiž všechny čtyři základní části aplikace a slouží k předávání dat mezi nimi. Definuje rozložení celého layoutu, a to pomocí komponenty Grid, která je součástí grafické knihovny Material UI. Tato knihovna je následně používána napříč celou aplikací. Umožňuje totiž snadně a rychle upravovat vzhled webové stránky, a to za přispění celé řady komponent. Zároveň komponenta obsahuje funkci *useEffect()* zjišťující připravenost webové aplikace. Na server je vyslán dotaz pro nahrání dat. Dokud klient nezíská odpověď na tento dotaz, je zobrazena načítací obrazovka. Následuje kód této komponenty.

```
function App() {
  const [response, setResponse] = useState("");
  const [urls, setURLs] = useState("");
  const [patterns, setPatterns] = useState("");
  const [loading, setLoading] = useState(false);

  const handleUpdate = (response) => {
    setResponse(response);
  };
  const handleUpdateURLList = (list) => {
    setURLs(list);
  };
  const handleSetPatterns = (patterns) => {
    setPatterns(patterns);
  };

  useEffect(() => {
    setLoading(true);
    fetch("/url", {
      method: "GET",
      credentials: "include",
      headers: {
        "Content-Type": "application/json",
      },
    })
      .then((res) => {
        if (!res.ok) {
          // error coming back from server
          throw Error("could not fetch the data for that resource");
        }
        return res.json();
      })
      .then((data) => {
        setLoading(false);
      })
      .catch((err) => {
        setLoading(false);
      });
  }, []);

  return (
    <Box sx={{ flexGrow: 1 }} p={1}>
      {loading ? (
        <LinearProgress />
      ) : (
        <Grid container spacing={2}>
          <Grid item xs={12} md={12}>
            <Navbar />
          </Grid>
          <Grid item xs={12} md={2}>
            <Leftbar
              handleUpdate={handleUpdate}
              handleUpdateURLList={handleUpdateURLList}
              patterns={patterns}
            />
          </Grid>
          <Grid item xs={12} md={10}>
            <Main
              HTMLresponse={response}
```

```
              listURLs={urls}
              handleSetPatterns={handleSetPatterns}
            />
          </Grid>
        </Grid>
      )}
    </Box>
  );
}

export default App;
```

Zdrojový kód 6: Komponenta App (vlastní zpracování)

4.7.3 Komponenta Leftbar

Tato komponenta popisuje v podstatě celou první část aplikace, tedy levý panel. Krom vzhledu levého panelu definuje několik základních funkcí pro komunikaci se serverem. Tyto funkce budou krátce popsány:

- *useEffect()* je speciálním typem funkce přímo knihovny React, která slouží k používání takzvaných side-effects, sleduje tedy akce, ke kterým dochází mimo rozsah funkce. Zde konkrétně vyčkává na načtení stránky a je spuštěna právě jednou. V tomto případě je *useEffect()* použit ve spojení s funkcí *fetch()*, která slouží ke komunikaci pomocí HTTP protokolu. Tentokrát přistupuje k serverovému bodu **/url** za použití metody GET. Získává informace o webových adresách a jejich strukturách.
- *handleSubmit()* je funkce pro zaslání uživatelem zadané adresy na server. Opět je k tomu využita funkce *fetch()*. Tato adresa je zaslána jako parametr na přístupový bod **/url** pomocí metody POST. Vrací strukturu webu.
- *handleDelete()* s využitím *fetch()* komunikuje se serverem. Tentokrát slouží k mazání zvolené adresy, a to pomocí parametru identifikátoru. Požadavek vysílá na adresu **/url** pomocí metody typu DELETE.

4.7.4 Komponenta Main

Tato komponenta se svým použitím velice podobá komponentě App, a to především protože je určena k definici rozložení dalších tří základních částí aplikace. Zároveň

je použita k předávání dat mezi těmito komponentami. Sama pak navíc obsahuje několik funkcí. Většina z nich slouží právě k filtraci předávaných dat. Obsahuje však jednu funkci komunikující se serverem:

- *handleSubmit()* je funkce sloužící k zasílání vzorů pro výběr komponenty. Jak je již v této práci zvykem, ke komunikaci se serverem využívá funkce *fetch()*, která jako parametr předává textový řetězec obsahující vzory. Komunikuje se serverem pomocí adresy **/parser**.

4.7.5 Komponenta Pattern

Komponenta určená převážně k definici vzhledu druhé základní části webové aplikace, udává tedy převážně informace o vstupním formuláři pro zadávání extrakčního vzoru. Obsahuje však také jednu funkci pro komunikaci se serverem:

- *useEffect()* je tu stejně jako v případě komponenty Leftbar použit pro získání dat při načtení aplikace. Provede se opět právě jednou. Komunikuje se serverem přes adresu **/getPatterns** a jejím cílem je načtení vzorů, kterými je následně naplněn vstupní formulář. Následuje ukázka:

```
useEffect(() => {
  fetch("/getPatterns", {
    method: "GET",
    credentials: "include",
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((res) => {
      if (!res.ok) {
        throw Error("could not fetch the data for that resource");
      }
      return res.json();
    })
    .then((data) => {
      if (data) {
        setPattern(data);
      }
    })
    .catch((err) => {});
}, []);
```

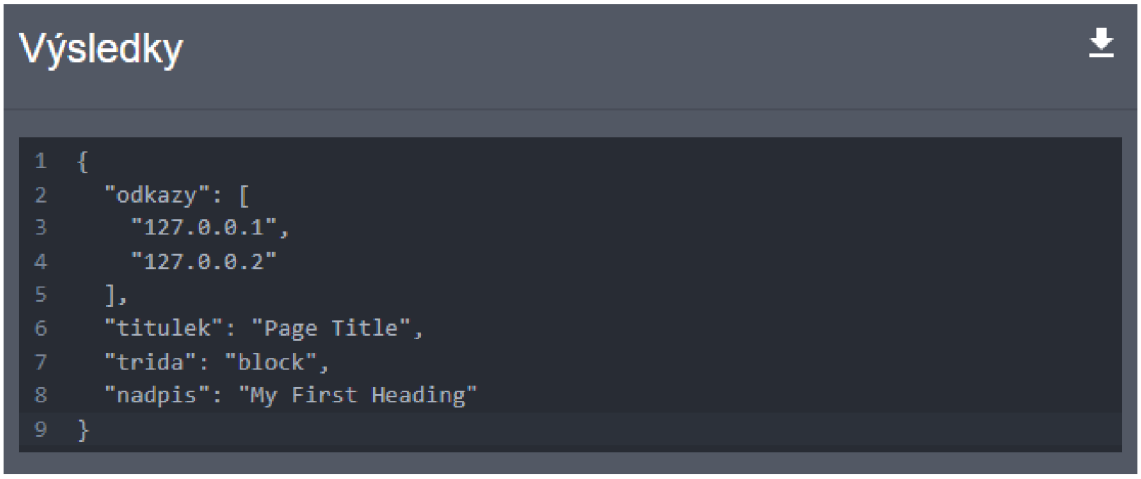
Zdrojový kód 7: Funkce komponenty Pattern (vlastní zpracování)

4.7.6 Komponenta Structure

Jedná se o menší komponentu, neobsahuje totiž žádnou zajímavější funkci a vnitřní logiku. Popisuje vzhled třetí základní části aplikace a k zobrazuje strukturu získané webové stránky. Je zde tak především uvedena právě protože popisuje základní část. Zároveň však ale zastřešuje komponentu ProxyModal představující modální okno pro přidávání proxy adres.

4.7.7 Komponenta Results

Svými vlastnostmi se tato komponenta velmi podobná předchozí popsané komponentě Structure. Je určena především k popisu vzhledu čtvrté základní části aplikace a zobrazení výsledků vyhledávání – prvků webové stránky. Také pak zastřešuje jednu z významnějších komponent, tou je CsvLink, která slouží ke stažení CSV souboru s výsledky zobrazenými v Results.



Obrázek 12: Část aplikace s výsledky (vlastní zpracování)

4.7.8 Komponenta ProxyModal

Jedná se o komponentu reprezentující modální okno sloužící k přidávání proxy adres. Krom definice vzhledu a funkcí sloužících k ovládání modálního okna obsahuje tři funkce spolupracující se serverem, všechny využívají funkce *fetch()*:

- *useEffect()* je zde obdobně jako v předchozích případech určen k načtení existujících adres již zadaných uživatelem. Spouští se vždy při otevření modálního okna a přistupuje k adrese **/proxy** pomocí dotazu typu GET.
- *handleSubmit()* je funkcí určenou k zasílání proxy adres na server. Přistupuje na adresu **/proxy** pomocí HTTP dotazu typu POST. Adresy nejsou na server přidávány po jednom, ale jako pole.
- *checkProxy()* je funkce, která slouží k ověření dostupnosti zadané proxy adresy. Na přístupový bod **checkProxy/** je zaslán GET požadavek. Na serveru pak následně dochází k ověření platnosti adresy. Pokud server vrací chybnou odpověď, adresa není zařazena do seznamu a uživatel je o tomto neúspěchu informován.

Obrázek 13: Modální okno pro vložení proxy adres (vlastní zpracování)

4.7.9 Komponenta UrlModal

Svou funkcionalitou komponenta velice podobná komponentě ProxyModal, opět definuje vzhled a funkcionalitu modálního okna. Tentokrát však slouží k nahrávání CSV souboru, který obsahuje adresy webových stránek, na server. Společně s tímto souborem je na server nahrána také mailová adresa, na kterou jsou následně zaslány nalezené prvky, a dále pak informace o formátu, v jakém budou výsledky uloženy. K tomu je využito funkce *handleSubmit()*:

- *handleSubmit()* funguje jako stejnojmenná funkce předchozí komponenty. Obdobně tak využívá metody POST. Tentokrát však komunikuje s adresou **/uploadFile**. Na tento přístupový body jsou zaslány tři parametry. Řetězec s mailovou adresou, řetězec určující typ exportu a CSV soubor.

5 Vyhodnocení výsledků

Na začátku předchozí kapitoly bylo vytyčeno několik cílů, kterých by měla aplikace dosáhnout. V první řadě se mělo jednat o funkční webovou aplikaci určenou k web scrapingu. Ta se nachází na následující webové adrese: https://webscraper2022.netlify.app/, tento cíl je tudíž splněn. Dalším z cílů bylo vytvořit nástroj, který by měl svou funkcionalitou aktivně předcházet detekci boty, a to za pomocí praktik již zmíněných v této práci. Pro připomenutí by se mělo jednat zejména o rotaci hlaviček dotazů cílených na daný server a možnost využívání proxy adres. Implementace těchto technik je také zmíněna v předchozí kapitole, nicméně nakolik tyto praktiky opravdu fungují? V této sekci bude provedeno několik testů, které by měly na tuto otázku odpovědět.

5.1 Metodika testování

Testy budou provedeny pomocí hromadného scrapingu na několika vybraných e-shopech, nejdříve bez použití proxy adres a následně za jejich použití. Je nutné mít dále na paměti, že některé z praktik aktivního předcházení detekce jsou defaultně spuštěny, jedná se právě o rotaci hlaviček HTTP dotazů a aktivního zpomalování vysílání dotazů na vybraný server. V kódu je nastavena půlsekundová prodleva mezi každým dotazem. Tato praktika nejenom snižuje šanci detekce boty, ale zároveň vede k zachování alespoň minima web scraperova kodexu, zásadně totiž snižuje zátěž na dotazovaný server. Testování bude prováděno pomocí aplikace umístěné na lokálním serveru autorova počítače.

Na každém webu dojde k extrakci tří základních informací z dvou set různých produktů. Bude se jednat o cenu, název produktu a URL obrázku produktu. Testovanými weby budou https://www.lidl.co.uk/, https://www.alza.cz/ a https://www.pilulka.cz/. Při testu s proxy adresami, dojde k náhodné rotaci mezi sedmi adresami poskytnutými platformou Webshare.

5.2 Výsledky testování

V případě prvního testovaného webu, tedy britské obdoby e-shopu Lidl, byl implementovaný nástroj úspěšný ve sto procentech případů, a to jak za použíti proxy adres, tak při jejich absenci. U všech zadaných adres tak došlo k úspěšné extrakci názvu produktů, ceny a URL adres obrázků.

Při scrapingu e-shopu Alza však došlo k zajímavému zjištění. Při použití proxy adres byla extrakce úspěšná jen u šesti produktů z celkové množiny dvou set. Zatímco při defaultním nastavení, tedy rotaci hlaviček dotazů a zasílání dotazů s prodlevou, došlo k extrakci všech požadovaných informací. Je nutné zmínit, že jako proxy byly použity adresy veřejného poskytovatele, proto je možné, že jsou tyto adresy předem vytipovány, a tím pádem efektivněji detekovány anti-bot systémy. Scaper byl při použití těchto adres přesměrován na stránku s captchou. I tento problém je možné vyřešit, nicméně by bylo za potřebí použití individuálních a zejména komplexnějších přístupů.

Stejně jako v případě britského Lidlu i Pilulka dosáhla sto procentní úspěšnosti v obou testech, jak s defaultním nastavením, tak s proxy adresami.

WEB	BEZ PROXY	S PROXY
LIDL.CO.UK	200/200	200/200
ALZA.CZ	200/200	6/200
PILULKA.CZ	200/200	200/200

Tabulka 3: Úspěšnost web scraperu (vlastní zpracování)

Na základě výše popsaného testování lze označit implementovaný nástroj za funkční a efektivní.

V rámci implementace došlo i k otestování rozšíření nástroje o možnost extrakce dat z dynamických webů vyžadujících interakci uživatele, a to za pomoci již zmíněného nástroje Selenium. Tato funkcionalita tak byla zaměřena například na stránky vyžadující přihlášení, či zobrazení rozšiřujícího obsahu, který se objeví jen v případě kliknutí na odkaz, či tlačítko. Umožňoval tak například zacílit na konkrétní webové

prvky a vyvolat jejich akci. Například vyplnění a odeslání formuláře, či udělení souhlasu k poskytnutí cookies. Uživatel tam měl možnost nastavit několik kroků, k jejich provedení by došlo v rámci zpracovávání stránky. Během práce na tomto rozšíření však brzy vyšlo najevo, že vytvořit intuitivní a zároveň univerzálně funkční nástroj, navíc ve formě webové aplikace, je značně komplikované, především však v době, kdy značná část webových stránek prezentuje svůj obsah za pomoci pop-up vyskakovacích oken a dalších prvků mimo DOM strukturu. A ačkoliv přineslo jeho použití na několika testovaných webech úspěšné výsledky, bylo od něj záhy ustoupeno, jelikož většina stránek vyžadovala individuální přístup na úrovni kódu. Autor práce může navíc tyto poznatky doplnit o zkušenosti z pracovního prostředí, jelikož se sám již několik let pohybuje v oblasti datové analýzy a s web scrapingem přichází do styku na denním pořádku. Ke kvalitní extrakci dat u dynamických webů je ve valné většině případů zapotřebí použití kódu přesně cíleného na konkrétní web.

Práce by pak dále mohla být rozšířena mimo výše popsaný nástroj například o možnost vytváření harmonogramu scrapování. Kdy by například docházelo k pravidelné extrakci informací z uživatelem zadaných stránek ve zvolený čas. Dalším zajímavým rozšířením by pak mohla být možnost volení prvků pomocí grafického rozhraní bez nutnosti znalosti jejich CSS selektorů.

6 Závěry a doporučení

V rámci práce došlo k představení různých typů přístupů k extrakci dat webů, a to především za použití API a web scrapingu. Dále došlo k seznámení se s existujícími nástroji podporujícími tyto přístupy. Tyto nástroje byly následně podrobeny kritickému testování a srovnání.

Na základě tohoto testování a srovnání byl navržen a následně implementován nástroj, který měl za cíl odstranit nedostatky popsaných nástrojů a přijít s vlastním přístupem k řešení dané problematiky. Za největší problémy existujících řešení byla považována téměř naprostá absence webových aplikací řešících problematiku web scrapingu a neefektivní aktivní předcházení detekce boty na stránkách extrahovaných webů.

Celý nástroj byl rozdělen do dvou částí, funguje na architektuře klient-server. Serverová část je psána především za pomocí jazyka Python a do za výrazného přispění frameworku Flask a knihovny BeautifulSoup. Ve své podstatě funguje jako API sloužící k ovládání scraperu pomocí HTTP dotazů. Klientská část je pak napsána v jazyce JavaScript s použitím knihovny React.

Tento nástroj byl následně nasazen na veřejně přístupný server a podroben testování na několika moderních webech. Data z těchto webů byla dále extrahována dle požadavků, jak je popsáno v páté kapitole. Proto lze nástroj považovat za úspěšný.

Zásadním poznatkem této práce je pak zjištění, že automatická extrakce dat z webových stránek vyžaduje v současnosti čím dál komplexnější řešení, zejména kvůli více a více sofistikovaným nástrojům pro detekci botů. Za nejlepší a nejefektivnější řešení pro extrakci dat je tak zcela jasně považovat individuální přístup ke každému webu. Každému scrapingu by měla především předcházet důkladná analýza problematické domény, a to jak webové stránky, tak vytyčení cílů, ke kterým budou získávaná data použita. Nejvhodnějším řešením pro komplexnější problémy se tak jednoznačně jeví individuální přístup až na úrovni kódu, a to například právě za použití jazyka Python, který nabízí celou řadu knihoven

(BeautifulSoup, Pandas), frameworků (Scrapy) a dalších již hotových řešení pro práci s daty velkého rozsahu a datovou problematikou obecně.

Seznam použité literatury

[1] JULIVER, Jamie. REST APIs: How They Work and What You Need to Know [online]. 2020 [cit. 2022-05-18]. Dostupné z: https://blog.hubspot.com/website/what-is-rest-api

[2] ALTEXSOFT. Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC [online]. 2020 [cit. 2022-05-18]. Dostupné z: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/

[3] GOYVAERTS, Jan a Steven LEVITHAN. Regulární výrazy: kuchařka programátora. Brno: Computer Press, 2010. ISBN 978-80-251-1935-8.

[4] REFSNES DATA, JavaScript HTML DOM. W3Schools [online].[cit. 2022-05-20]. Dostupné z: https://www.w3schools.com/js/js_htmldom.asp

[5] Směrnice Evropského parlamentu a Rady 2019/790 ze dne 17. dubna 2019 o autorském právu a právech s ním souvisejících na jednotném digitálním trhu a o změně směrnic 96/9/ES a 2001/29/ES, [cit. 2022-05-24]. Dostupné z https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32019L0790&from=EN

[6] GOOGLE. Create a robots.txt file [online]. In: . 2022 [cit. 2022-05-24]. Dostupné z: https://developers.google.com/search/docs/advanced/robots/create-robots-txt

[7] ROZI, Fatkhur. Captcha Dikti Emang Bikin Pusing, Tapi 10 Captcha Ini Juga Nggak Kalah Bikin Sengsara. Hipwee [online]. 2017 [cit. 2022-05-20]. Dostupné z: https://www.hipwee.com/feature/bukan-cuma-captcha-dikti-yang-bikin-sakit-hati-ini-10-captcha-yang-dijamin-bakal-buat-kamu-emosi/

[8] GRABZIT. GrabzIt [online]. [cit. 2022-06-02]. Dostupné z: https://grabz.it/

[9] PARSEHUB. ParseHub [online]. [cit. 2022-08-13]. Dostupné z: https://www.parsehub.com/

[10] WEB SCRAPER. Web Scraper [online]. [cit. 2022-06-02]. Dostupné z: https://webscraper.io/

[11] VANDEN BROUCKE, Seppe a Bart BAESENS. Practical Web Scraping for Data Science: Best Practices and Examples with Python. 1. Apress Berkeley, CA, 2018. ISBN 978-1-4842-3582-9.

[12] SELENIUM. The Selenium Browser Automation Project. Selenium [online]. 2022 [cit. 2022-06-118]. Dostupné z: https://www.selenium.dev/

[13] PYTHON. Python [online]. 2022 [cit. 2022-06-18]. Dostupné z: https://www.python.org

[14] RELAN, Kunal. Building REST APIs with Flask: Create Python Web Services with MySQL. 1. Apress Berkeley, CA, 2019. ISBN 978-1-4842-5021-1

[15] PALLETS. Flask: User's Guide [online]. [cit. 2022-08-12]. Dostupné z: https://flask.palletsprojects.com/en/2.2.x/

[16] RICHARDSON, Leonard. Beautiful Soup Documentation [online]. 2022 [cit. 2022-07-18]. Dostupné z: https://beautiful-soup-4.readthedocs.io/en/latest/

[17] PEHLIVANIAN, Ara a Don NGUYEN. JavaScript okamžitě. 2. vydání. Přeložil Ondřej BAŠE. Brno: Computer Press, 2021. ISBN 978-80-251-5025-2.

[18] META PLATFORMS, INC. React: Getting Started [online]. 2022 [cit. 2022-07-18]. Dostupné z: https://reactjs.org/docs/getting-started.html

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Přílohy

Obsah přiloženého souboru

Přiložený soubor obsahuje dvě hlavní složky se zdrojovými kódy popsaného nástroje. Adresář server obsahuje zdrojové kódy serverové části. Adresář klient obsahuje zdrojové kódy klientské části. Dále je v souboru přiložený textový soubor odkazující na git implementovaného nástroje.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2020/2021

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení:	**Bc. Jan Thér**
Osobní číslo:	**I1900290**
Adresa:	Královec 61, Královec, 54203 Královec, Česká republika
Téma práce:	Extrakce dat z webu pomocí webscrapingu
Téma práce anglicky:	Data extraction from web sources with webscraping
Vedoucí práce:	Ing. Martina Husáková, Ph.D. Katedra informačních technologií

Zásady pro vypracování:

Student se seznámí se současnými trendy a technologiemi v oblasti extrakce dat a to zejména s využitím technologie webscrapingu. Následně navrhne a implementuje vlastní webscrapingový nástroj, jehož použitelnost otestuje a vyhodnotí.
Osnova:

1. Úvod
2. Vybrané technologie extrakce dat z webu
3. Využitelnost extrakce dat z webu
4. Webscraping
5. Vlastní implementace webscraperu
6. Vyhodnocení výsledků
7. Závěry a doporučení

Seznam doporučené literatury:

SLÁNSKÝ, David. *Data a analytika pro 21. století*. [Praha]: Professional Publishing, 2018. ISBN 978-80-88260-25-7.
BROUCKE, Seppe vanden a Bart BAESENS. *Practical Web Scraping for Data Science*. 1. Berkeley, CA: Apress, 2018. ISBN 978-1-4842-3582-9.
HAJBA, Gábor László. *Website Scraping with Python: Using BeautifulSoup and Scrapy*. 1. Apress, 2018. ISBN 978-1-4842-3924-7.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: