VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INTEGRACE WEBOVÝCH ZDROJŮ DAT DO INFORMAČNÍCH SYSTÉMŮ

INTEGRATION OF WEB DATA SOURCES TO INFORMATION SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ERIK HRUBÝ**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154241

Ústav: Ústav informačních systémů (UIFS)
Student: **Hrubý Erik**
Program: Informační technologie
Název: **Integrace webových zdrojů dat do informačních systémů**
Kategorie: Web
Akademický rok: 2023/24

Zadání:

1. Prostudujte možnosti prezentace strukturovaných dat ve webových dokumentech.
2. Seznamte se s existujícími knihovnami a nástroji pro extrakci dat z webových dokumentů (web scraping). Zaměřte se na platformu Java.
3. Navrhněte architekturu knihovny pro snadnou integraci dat z konkrétních webových zdrojů do informačního systému.
4. Implementujte navrženou knihovnu. Po dohodě s vedoucím implementujte i podporu pro vhodné autentizační metody.
5. Proveďte testování vytvořené knihovny na vhodných příkladech.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Ament, J. D., McCright, A., Finnigan, K., Szynkiewicz, M.: REST Client for MicroProfile, version 3.0, Eclipse Foundation, 2021
- Sahin, K.: Scraping the web with Playwright, online: https://www.scrapingbee.com/blog/playwright-web-scraping/ [cit. 10.10.2023]

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3

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

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Cieľom tejto práce je vytvoriť knižnicu pre integráciu dát z webových zdrojov, typu HTML dokument, do informačných systémov. Knižnica je implementovaná v jazyku Java a programátor ju bude môcť použiť na rýchle a jednoduché mapovanie dát z dokumentu HTML na dátové štruktúry jazyku Java (objekty), ktoré bude môcť ďalej voľne využívať vo svojom informačnom systéme fungujúcom na platforme Java. Od programátora bude vyžadované aby dodal knižnici vlastnú implementáciu, v ktorej bude pomocou anotácií popísané akým spôsobom sa majú dané hodnoty vyhľadať pomocou selektoru CSS alebo výrazu XPath. Na stiahnutie webového dokumentu je použitá knižnica Jsoup.

Abstract

The goal of this work is to create a library for the integration of data from web resources, such as HTML document, into information systems. The library is implemented in the Java language and the programmer will be able to use it for quick and easy mapping of data from the HTML document to Java data structures (objects), which he will be able to freely use in his information system operating on the Java platform. The programmer will be required to supply the library with his own implementation, in which the annotations will describe how the given values should be searched using the CSS selector or the XPath expression. The Jsoup library is used to download the web document.

Kľúčové slová

Java, HTML, CSS, Jsoup, XPath, Informačný systém, Web scraping, Java anotácie, Maven, Java Reflection API

Keywords

Java, HTML, CSS, Jsoup, XPath, Information system, Web scraping, Java annotations, Maven, Java Reflection API

Citácia

HRUBÝ, Erik. *Integrace webových zdrojů dat do informačních systémů.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Integrace webových zdrojů dat do informačních systémů

Prehlásenie

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

.....
Erik Hrubý
6. mája 2024

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce doc. Ing. Radkovi Burgetovi, Ph.D za pravidelné konzultácie, pomoc s návrhom a rady pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Za najväčšie úložisko informácií na Zemi sa často považuje internet. Internet obsahuje obrovské množstvo údajov a znalostí vrátane webových stránok, databáz a iných digitálnych zdrojov, ktoré sú prístupné ľuďom na celom svete. Úspech podnikania dnes výrazne závisí od schopnosti organizácie robiť rozhodnutia založené na údajoch a rozvíjať stratégiu založenú na analýze údajov. Pri tejto analýze údajov sa často používa technika nazývaná web scraping, ktorá sa používa na extrahovanie údajov z webových stránok.

Existuje rada nástrojov, ktoré nám umožňujú realizovať web scraping. Pre jednoduché prípady existujú dokonca rozšírenia prehliadača, ktoré sú ale rozsahom obmedzené. Pri pokuse použiť tieto nástroje pre integráciu v informačných systémoch, narazíme na problém, kde nám dané nástroje ponúkajú rozsiahle možnosti, ako s webovými stránkami manipulovať, vyhľadávať, prechádzať, ale programátor si musí stále manuálne spracovať dané dáta, aby s nimi mohol pracovať v doméne daného systému.

Cieľom tejto bakalárskej práce je vytvoriť knižnicu pre jednoduchú integráciu dát z webových zdrojov do informačných systémov so zameraním na platformu Java. Knižnica bude teda poskytovať spôsob, ako rýchlo a jednoducho previesť dáta z webového dokumentu do domény informačného systému.

Práca je logicky rozdelená na jednotlivé kapitoly. V kapitole 2 je popísané čo sú to webové dokumenty a akými spôsobmi v nich je možne reprezentovať rôzne dáta. Ďalej sa v kapitole 3 zoznámime s nástrojmi, ktoré sa využívajú na extrakciu dát z webových dokumentov, a to, akým spôsobom fungujú. Spomenuté budú aj nástroje, ktoré je možné použiť pre platformu Java. Kapitola 4 obsahuje návrh knižnice pre integráciu dát z webových dokumentov, ktorá je následne implementovaná v kapitole 5. Demo aplikácie, slúžiace na ukážku fungovania implementácie a testovanie, je popísané v kapitole 6. Posledná kapitola 7 obsahuje záverečné zhrnutie.

Kapitola 2

Možnosti prezentácie štruktúrovaných dát vo webových dokumentoch

Dáta na webových stránkach sú obvykle štruktúrované a prezentované pomocou kombinácie HTML[1], CSS[2] a niekedy aj JavaScriptu. Pre správne pochopenie techniky web scrapingu, ktorá bude nasledovať v kapitole 3, je doležité pochopiť, čo sú to webové dokumenty, ako vyzerá ich štruktúra a prvky, ktoré obsahujú.

2.1 Webové dokumenty

V tejto práci sa bude často vyskytovať slovo webový dokument (skrátene dokument), ktoré sa často zamieňa za slovo webová stránka. Webový dokument preto definujeme ako súbor, ktorý obsahuje zdrojový kód HTML. Webová stránka je následne zložená z niekoľkých webových dokumentov prístupných cez URL[3]. Na zobrazenie webových stránok sa používa webový prehliadač, ktorý získava tieto dokumenty pomocou protokolu HTTP[4] alebo HTTPS[5] [1].

2.1.1 HyperText Markup Language

HyperText Markup Language alebo v skratke HTML je najzákladnejším stavebným kameňom webu. Definuje význam a štruktúru webového obsahu. Obsah môže byť napríklad štruktúrovaný v rámci súboru odsekov, zoznamu bodov s odrážkami, alebo pomocou obrázkov a tabuliek. Iné technológie, okrem HTML, sa vo všeobecnosti používajú na opis vzhľadu/prezentácie webovej stránky, ako sú napríklad CSS, alebo funkčnosti/správania, na čo sa využíva jazyk JavaScript. *Hypertext* označuje odkazy, ktoré navzájom spájajú webové stránky, či už v rámci jednej lokality stránky alebo, medzi rôznymi webovými stránkami. HTML používa značky (*markup*) na označenie textu, obrázkov a iného obsahu na zobrazenie vo webovom prehliadači. Kód HTML obsahuje špeciálne prvky, ako sú `<head>`, `<title>`, `<body>`, `<header>`, `<p>`, `<div>`, `<span>`, `<img>`, a mnoho ďalších [10].

[1] HyperText Markup Language
[2] Cascading Style Sheets
[3] Uniform Resource Locator
[4] Hypertext Transfer Protocol
[5] Hypertext Transfer Protocol Secure

2.1.2 Štruktúra prvku HTML

Prvky HTML sú odlíšené od ostatného textu v dokumente ”tagmi”, ktoré pozostávajú z názvu prvku obklopeného znakmi `<` a `>`. V názve prvku sa nerozlišujú veľké a malé písmená. To znamená, že môže byť napísaný veľkými písmenami, malými písmenami, alebo kombináciou oboch. Teda tag `<title>` môže byť napísaný ako `<Title>`, `<TITLE>`, alebo akýmkoľvek iným spôsobom. Konvenciou a odporúčanou praxou je však písať značky malými písmenami [10].

Hlavné časti prvku:

- **otvárací tag** – pozostáva z názvu prvku zabaleného do otváracích a zatváracích lomených zátvoriek. Označuje, kde prvok začína.
- **uzatvárací tag** – je rovnaký ako otvárací tag, ale obsahuje ešte lomku pred názvom prvku. Označuje, kde prvok končí.
- **obsah** – obsah prvku, ktorý sa nachádza medzi otváracím a uzatváracím tagom.
- **prvok** – obsah spolu s otváracím a uzatváracím tagom tvoria prvok.

```
<title>Title</title>
```

Výpis 2.1: Ukážka prvku <title> obsahujúci text "Title".

Prvky môžu mať aj **atribúty**, ktoré sa nachádzajú vždy v otváracom tagu. Atribúty obsahujú ďalšie informácie o prvku, ktoré nechceme, aby sa zobrazovali v skutočnom obsahu. Zvyčajne sa vyskytujú v pároch meno/hodnota vo formáte `meno="hodnota"`, ale atribúty môžu byt aj **bezhodnotové**.

```
<button type="button" disabled>Click me!</button>
```

Výpis 2.2: Ukážka prvku <button> obsahujúci atribúty type s hodnotou ”button” a bezhodnotový atribút disabled.

Prvky je možné vložiť aj do iných prvkov. To sa nazýva **vnorenie** (*nesting*). To nám umožňuje vytvorenie zložitých štruktúr a rozložení na webových stránkach. Každý vnorený prvok je obsiahnutý vo svojom nadradenom prvku a tvorí hierarchickú štruktúru. Prvky musia byt správne otvorené a uzatvorené, aby bolo jasné, či sú vnútri, alebo mimo seba. Ak sa prekrývajú, webový prehliadač sa pokúsi čo najlepšie odhadnúť, ako majú byť prvky usporiadané, čo môže viesť k neočakávaným výsledkom.

```
<head>
    <title>
        <span>Main Title</span>
    </title>
</head>
```

Výpis 2.3: Ukážka vnorenia prvkov.

Niektoré prvky nemajú **žiadny** obsah a nazývajú sa **prázdne prvky** (*void elements*). To znamená, že nemôžu mať žiadne vnorené prvky a obsahujú iba počiatočný tag. Prázdne prvky **nesmú** obsahovať koncový tag.

```
<input type="text" id="username" name="username">
```

Výpis 2.4: Ukážka prázdneho prvku input.

2.1.3 Základná štruktúra webového dokumentu HTML

Týmto boli zhrnuté základy jednotlivých prvkov HTML. Teraz si priblížime, ako sa tieto prvky kombinujú do výsledného dokumentu.

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>Simple HTML Page</title>
</head>
<body>
    <header><h1>Simple HTML Page header</h1></header>
        <ul>
          <li>Section 1</li>
          <li>Section 2</li>
        </ul>
    <main>
        <section id="section2">
            <table border="1">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Age</th>
                        <th>Country</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Emma</td>
                        <td>25</td>
                        <td>USA</td>
                    </tr>
                    <tr>
                        <td>Alexander</td>
                        <td>30</td>
                        <td>Spain</td>
                    </tr>
                </tbody>
            </table>
        </section>
    </main>
</body>
</html>
```

Výpis 2.5: Ukážka jednoduchej stránky HTML obsahujúcej bodkovaný zoznam a tabuľku.

Hlavné prvky, ktoré obsahuje každý dokument [10]:

- `<!DOCTYPE html>` – povinná preambula. Pozostatok času, keď bolo HTML ešte mladé (okolo rokov 1991/92), `doctypes` mali fungovať ako odkazy na súbor pravidiel, ktoré musela HTML stránka dodržiavať, aby bola považovaná za dobrý HTML, čo mohlo znamenať automatickú kontrolu chýb a iné užitočné veci. V dnešnej dobe však toho veľa nerobia a sú v podstate potrebné len na to, aby sa dokument správal správne.
- `<html>` – tento prvok obaľuje všetok obsah na celej stránke. Označuje sa aj ako koreňový prvok (`root element`).
- `<head>` – tento prvok funguje ako kontajner pre všetky veci, ktoré chceme zahrnúť na stránku HTML a ktoré nie sú obsahom, ktorý je zobrazovaný používateľom stránky. To zahŕňa veci ako kľúčové slová a popis stránky, ktoré sa majú zobrazovať vo výsledkoch vyhľadávania, CSS na úpravu štýlu obsahu, deklarácie znakovej sady a ďalšie.
- `<meta charset="utf-8">` – nastavuje znakovú sadu, ktorú by mal dokument používať, na UTF-8, ktorý obsahuje väčšinu znakov z veľkej väčšiny písaných jazykov.
- `<title>` – nastaví názov danej stránky, ktorý sa zobrazí na karte prehliadača, na ktorej je stránka načítaná.
- `<body>` – obsahuje všetok obsah, ktorý sa má zobraziť používateľom webu, keď navštívia stránku, či už ide o text, obrázky, videá, alebo čokoľvek iné.

2.2 Typy dát

Dáta môžu byť rozdelené do rôznych typov na základe ich štruktúry. Každý typ má svoje vlastné charakteristiky a je vhodný pre rôzne typy aplikácií a prípady použitia.

2.2.1 Štruktúrované dáta

Štruktúrované dáta sú akékoľvek informácie **usporiadané** do formátu, ktorý tradičné počítače ľahko nájdu – je na nich možné robiť efektívnu analýzu. Dokumenty so štruktúrovanými údajmi majú **pevné rozloženie** a zvyčajne obsahujú polia s textom, hodnotami a inými usporiadanými údajmi, ako sú pevné formuláre, tabuľky, alebo databázy. Najväčšou výhodou dokumentov so štruktúrovaným rozložením je, že informácie sú už navrhnuté a optimalizované pre rýchle a jednoduché spracovanie počítačovými systémami. Vďaka tomu je tiež možné údaje ľahko vyhľadávať pomocou tradičných automatizačných nástrojov, založených na pevne daných pravidlách, ako napríklad tabuľku relačnej databázy [25].

Last name	First Name	Email	Country
Johnson	Emma	emma.johnson@example.com	United States
García	Alexander	alexander.garcia@example.com	Spain
Lee	Mia	mia.lee@example.com	Australia
Müller	Lucas	lucas.muller@example.com	Germany

Obr. 2.1: Ukážka štruktúrovaných dát.

2.2.2 Polo-štruktúrované dáta

Polo-štruktúrované údaje sú informácie, ktoré sami o sebe neexistujú v štruktúrovanom alebo pevnom formáte, ako je databáza alebo tabuľka, ale obsahujú značky a metadáta na oddelenie sémantických prvkov, a vytvorenie hierarchie záznamov, a polí. Takéto dáta je možné nájsť napríklad v dokumentoch XML[6]/HTML, súboroch JSON[7], alebo databázach NoSQL [25].

```
{
  {
    "firstName": "Emma",
    "lastName": "Johnson",
    "email": "emma.johnson@example.com",
    "country": "United States"
  },
  {
    "firstName": "Alexander",
    "lastName": "Garcia",
    "email": "alexander.garcia@example.com",
    "country": "Spain"
  },
  {
    "firstName": "Mia",
    "lastName": "Lee",
    "email": "mia.lee@example.com",
    "country": "Australia"
  },
  {
    "firstName": "Lucas",
    "lastName": "Muller",
    "email": "lucas.muller@example.com",
    "country": "Germany"
  }
}
```

Výpis 2.6: Ukážka polo-štruktúrovaných dát vo formáte JSON.

2.2.3 Neštruktúrované dáta

Neštruktúrované dáta sú informácie, ktoré nie sú usporiadané do žiadneho konkrétneho formátu a môžu mať úplne **voľnú formu**. Tento typ dát býva plný textu a chýba mu akákoľvek organizácia alebo metadáta, ktoré by umožnili jednoduchú analýzu alebo spracovanie počítačmi [25]. Príklady neštruktúrovaných údajov môžu zahŕňať textové dokumenty, fotografie, videá, e-maily, knihy a mnohé iné. Analýza neštruktúrovaných dát obvykle vyžaduje pokročilé techniky, ako je spracovanie prirodzeného jazyka, rozpoznávanie obrázkov (image recognition), alebo strojové učenie (machine learning).

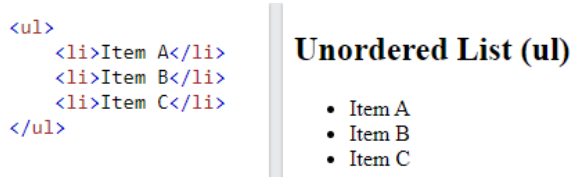
[6] Extensible Markup Language
[7] JavaScript Object Notation

2.3 Štruktúry v dokumentoch HTML

HTML, ako primárne značkovací jazyk na štruktúrovanie obsahu na webe, nemá tradičné dátové štruktúry, aké nájdeme v programovacích jazykoch. Poskytuje však prvky na organizáciu a prezentáciu rôznych typov obsahu.

Zoznamy

HTML zoznam sa používa na zobrazenie údajov alebo akýchkoľvek informácií na webových stránkach v očíslovanej (`<ol>`), neočíslovanej (`<ul>`), alebo popisnej (`<dl>`) forme.



Obr. 2.2: Ukážka neočíslovaného zoznamu a jeho zobrazenia v prehliadači.

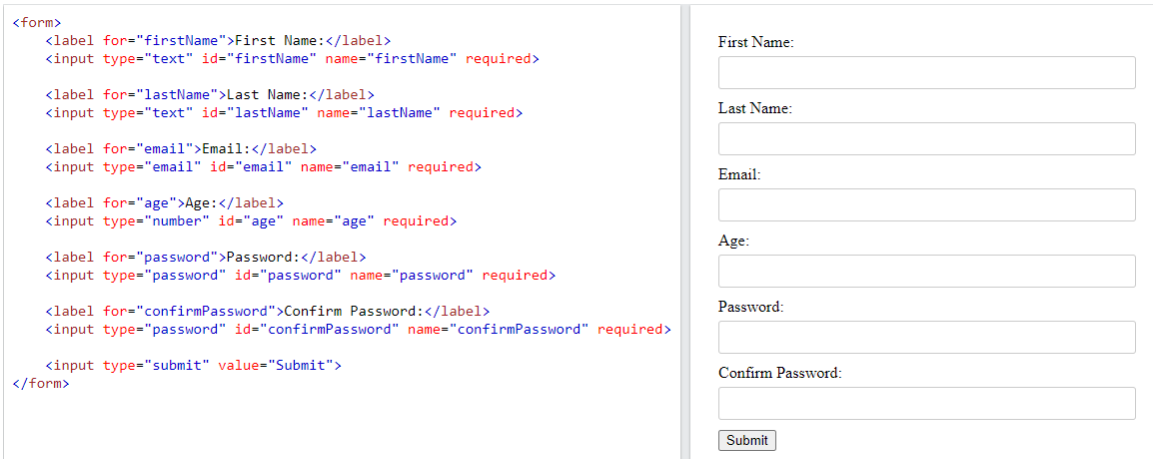
Tabuľky

Predstavujú tabuľkové údaje – to znamená informácie prezentované v dvojrozmernej tabuľke zloženej z riadkov a stĺpcov buniek obsahujúcich údaje.

Obr. 2.3: Ukážka jednoduchej tabuľky a jej zobrazenia v prehliadači.

Formuláre

Prvok `<form>` sa používa na časť dokumentu obsahujúcu interaktívne ovládacie prvky na odosielanie informácií. Tento prvok obsahuje rôzne vstupné (`<input>`) prvky, ako textové polia, začiarkavacie políčka (`checkbox`), tlačidlá a podobne.



Obr. 2.4: Ukážka formulára a jeho zobrazenia v prehliadači.

2.4 Štruktúrované dáta v dokumentoch HTML

Zatiaľ čo ľudskí používatelia internetu môžu vyvodiť, že nádpis sa má chápať ako nádpis a podnádpis je meno autora atď., programy dokážu interpretovať iba informácie, ktoré boli označené v kóde HTML (nádpis `<h1>`, podnádpis `<h2>`, kurzíva `<i>`). Takéto problémy sú relevantné pre webové prehľadávače (*web crawlers*) vyhľadávacích nástrojov (napr. *Google*[8]), ktoré sú zodpovedné za určenie relevantnosti webovej stránky na základe vyhľadávacích požiadavkov. Mnohí majitelia webových stránok preto obohacujú svoje HTML dokumenty o **strojovo čitateľné** sémantické informácie, ktoré definujú význam jednotlivých obsahov. Toto je známe v kontexte webových stránok ako **štruktúrované dáta** [7].

Existuje niekoľko štandardných formátov, ktoré vlastníci stránok môžu použiť, aby zabezpečili, že obsah so štruktúrovanými údajmi bude strojovo čitateľný. Patria sem formáty založené na sémantickom značkovaní, ako `microformats`, `RDFa` a `Microdata`, kde v závislosti od formátu je možné priamo v HTML kóde použiť tradičné atribúty HTML, alebo nové prvky označovania. Ďalšou možnosťou je formát `JSON-LD`, ktorý umožňuje označiť webovú stránku v rámci skriptu [7].

2.4.1 microformats

`microformats` sa používajú na označenie bežných typov informácií, ako sú ľudia, udalosti, miesta, recenzie a iné, pomocou tried a atribútov v rámci značiek HTML. `microformats` dodržiavajú princípy *"reuse the web"*, čo znamená, že ich cieľom je stavať na existujúcich webových konvenciách a postupoch, a nie zavádzať úplne nové štandardy. Vďaka tomu sa dajú ľahko implementovať a integrovať do existujúcich webových stránok [8].

[8] https://www.google.com/

Niektoré bežne používané mikroformáty zahŕňajú:

- **hCard** – kontaktné informácie ľudí alebo organizácií.
- **hReview** – recenzie produktov, služieb alebo iných položiek.
- **hRecipe** – recepty na varenie.
- **adr** – informácie o umiestnení adresy.
- **geo** – geografické súradnice (zemepisná šírka a dĺžka).

```
<div class="geo">
    <span class="latitude">49.1950</span>,
    <span class="longitude">16.6068</span>
</div>
```

Výpis 2.7: Ukážka použitia microformatu geo na reprezentáciu geografických súradníc.

2.4.2 RDFa

`RDFa`[9] poskytuje syntax na priradenie sémantických značiek priamo k prvkom a atribútom HTML dokumentov. To umožňuje vytvorenie metódy na zverejňovanie štruktúrovaných dát na webe spôsobom, ktorý umožňuje ich vzájomné prepojenie a spracovanie ľuďmi aj strojmi. `RDFa` umožňuje vkladať štruktúrované údaje pomocou atribútov ako `property`, `typeof` a `resource`. Tieto štruktúrované údaje predstavujú entity a vzťahy skutočného sveta. V `RDFa` je odporúčané používanie URI[10] na jedinečnú identifikáciu zdrojov, čo umožňuje ich vzájomné prepojenie v rámci rôznych súborov údajov a webových stránok [17].

```
<div typeof="geo:Point">
    <span property="geo:lat">49.1950</span>,
    <span property="geo:long">16.6068</span>
</div>
```

Výpis 2.8: Ukážka použitia RDFa na reprezentáciu geografických súradníc.

2.4.3 Microdata

`Microdata` sa pokúšajú poskytnúť jednoduchší spôsob na označenie prvkov HTML strojovo čitateľnými značkami (tags), ako podobné prístupy použitia `RDFa` a `microformats`. Zvyčajne sa používajú v spojení so **schémami**, ktoré definujú slovníky výrazov na popis konkrétnych typov údajov. Jednou z najpoužívanejších schém je `Schema.org`[11], ktorá poskytuje komplexný súbor výrazov na popis širokého spektra entít a vlastností. Microdata na vysokej úrovni pozostávajú zo skupiny párov **názov-hodnota**. Skupiny sa nazývajú položky a každý pár názov-hodnota je vlastnosť. Položky a vlastnosti sú reprezentované štandardnými prvkami. Na vytvorenie položky sa používa atribút `itemscope`. Pre pridanie vlastnosti k položke sa použije atribút `itemprop` na jednom z potomkov položky [12].

[9] Resource Description Framework in Attributes
[10] Uniform Resource Identifier
[11] https://schema.org/

```
<div itemscope itemtype="http://schema.org/Place">
    <span itemprop="geo" itemscope itemtype="http://schema.org/GeoCoordinates">
        <span itemprop="latitude">49.1950</span>,
        <span itemprop="longitude">16.6068</span>
    </span>
</div>
```

Výpis 2.9: Ukážka použitia Microdata na reprezentáciu geografických súradníc.

2.4.4 JSON-LD

JSON-LD[12] je formát na výmenu údajov, ktorý sa používa na serializáciu[13] takzvaných **Linked Data**. Linked Data je metóda na publikovanie a prepojenie štruktúrovaných dát na webe. Riadi sa súborom princípov a osvedčených postupov tak, aby bolo zabezpečené, že údaje zverejnené na webe budú dostupné, vyhľadateľné a interoperabilné [21].

JSON-LD používa známu syntax JSON, ktorá je široko podporovaná a zrozumiteľná. To umožňuje definovať kontext pre údaje, ktorý (kontext) poskytuje mapovanie medzi kľúčmi JSON a URI. JSON-LD je zvyčajne súčasťou prvkou `<script>` v hlavičke dokumentu HTML. Zahrnutím JSON-LD do prvkou `<script>` ho možno oddeliť od hlavného obsahu dokumentu HTML, čím sa zlepší čitateľnosť a udržiavateľnosť [20].

```
<head>
    <script type="application/ld+json">
    {
      "@context": "https://schema.org/",
      "@type": "Place",
      "geo": {
        "@type": "GeoCoordinates",
        "latitude": 49.1950,
        "longitude": 16.6068
      }
    }
    </script>
</head>
```

Výpis 2.10: Ukážka použitia JSON-LD na reprezentáciu geografických súradníc v hlavičke dokumentu HTML.

[12] JavaScript Object Notation for Linked Data
[13] https://developer.mozilla.org/en-US/docs/Glossary/Serialization

Kapitola 3

Nástroje na extrakciu dát z webových dokumentov

Extrakcia dát z webových dokumentov, taktiež nazývaná **web scraping**, web harvesting, alebo data mining, je mocným nástrojom na získavanie údajov vo veľkom rozsahu a extrahovanie z nich cenných informácií, bez ohľadu na to, či ide o osobné alebo obchodné účely [3].

Web scraping má mnoho výhod, napríklad nám umožňuje rýchlo získať veľké množstvo informácií. Predstavme si, že chceme získať recenzie produktov z rôznych webových stránok ako Amazon a Google. S web scrapingom to je zvládnuteľné za pár minút, zatiaľ čo manuálne by na to mohli byť potrebné hodiny alebo dokonca dni. Okrem toho nám web scraping pomáha automatizovať opakujúce sa úlohy a môže byť veľmi užitočný aj vo firemnom prostredí. Mnohé spoločnosti ho využívajú na výskum trhu, kde zbierajú informácie o produktoch a cenách od svojich konkurentov, alebo agregujú údaje z viacerých zdrojov [3].

3.1 Web Scraping verzus API

Existuje viacero spôsobov ako získať údaje z webových stránok a web scraping nie je jediným. V skutočnosti to nie je ani najčastejší prístup. Preferovaná metóda je využitie API[1], čo znamená rozhranie pre programovanie aplikácií, ktoré poskytuje **programový spôsob**, ako komunikovať s konkrétnou webovou stránkou alebo aplikáciou. Týmto spôsobom je možné získať údaje presnejšie a štruktúrovanejšie ako pri použití web scrapingu [3].

3.1.1 Výhody a nevýhody extrakcie dát pomocou API

Z hľadiska jednoduchosti použitia poskytuje API zvyčajne priamočiare metódy na požiadavky údajov, typicky prostredníctvom HTTP požiadavku. Okrem toho často poskytujú odpovede vo štruktúrovaných formátoch ako JSON alebo XML, čo zjednodušuje ich programové spracovanie.

Jednou z častých nevýhod je **chýbajúce** API. Niektoré webové stránky nepotrebujú rozhranie API, pretože primárne slúžia na statický obsah, alebo nevyžadujú externú integráciu. S použitím API môžu prísť aj určité komplikácie s autentizáciou a autorizáciou. Niektoré API vyžadujú autentizáciu, napríklad prostredníctvom API kľúčov alebo OAuth[2]

[1] Application Programming Interface
[2] https://oauth.net/2/

tokenov, čo zvyšuje úroveň zložitosti pri zabezpečenom riadení týchto overovacích údajov. Dôležité je taktiež podotknúť, že nie všetky údaje sú dostupné cez API. Poskytovatelia môžu obmedziť rozsah prístupných údajov z dôvodu obáv o súkromie, obchodných politík, alebo technických obmedzení. Ďalej API často stanovujú **limity** na frekvenciu požiadaviek a kvóty, čo obmedzuje objem údajov, ku ktorým možno pristupovať v určitom časovom období. To môže skomplikovať projekty, ktoré pristupujú k veľkému množstvu údajov [2]. API taktiež poskytujú štruktúrované možnosti vyhľadávania, ktoré umožňujú používateľom presne špecifikovať ich požiadavky na údaje. Táto vlastnosť zabezpečuje, že sa získavajú len relevantné údaje, čo v konečnom dôsledku šetrí čas a zdroje. Navyše, API umožňujú nastavenie rôznych parametrov, ako je filtrovanie, triedenie a stránkovanie, čo poskytuje používateľom väčšiu kontrolu nad spôsobom, akým sú údaje zobrazené a spracované [2].

3.1.2 Výhody a nevýhody extrakcie dát pomocou web scrapingu

Web scraping musí efektívne zvládať rôzne štruktúry webových stránok. Zahŕňa analýzu HTML a v niektorých prípadoch aj vykonávanie JavaScriptu, čo zvyšuje komplexitu vzhľadom na dynamickú povahu štruktúr webových stránok. Na extrahovanie takéhoto dynamického obsahu môžu byť potrebné sofistikovanejšie prístupy, ako je použitie headless prehliadačov (sekcia 3.2.3). Údržba je ďalším aspektom, ktorý treba zvážiť pri web scrapingu. Web scraper často potrebuje **pravidelné aktualizácie** na prispôsobenie sa zmenám v štruktúre alebo dizajne webovej stránky, čo ho robí náročnejším na údržbu [2].

Jednou z výhod web scrapingu je možnosť prístupu k údajom, ktoré by nemuseli byť dostupné cez API. Toto ho robí užitočným na extrahovanie informácií z webových stránok bez verejného API. To znamená, že web scraping nám ponúka širšiu škálu potenciálnych zdrojov údajov, s ktorými môžeme pracovať. Okrem toho, nie je podrobený rovnakým obmedzeniam ako použitie API, ako sú limity na frekvenciu požiadaviek, alebo obmedzenia rozsahu prístupných údajov. Tieto vlastnosti poskytujú väčšiu slobodu vzhľadom na zbieranie údajov. Navyše nám web scraping poskytuje flexibilitu vzhľadom na frekvenciu zbierania údajov, čo nám umožňuje zbierať údaje tak často, ako ich potrebujeme, pokiaľ cieľová webová stránka zostane prístupná [2].

3.2 Proces extrahovania dát pri web scrapingu

Všeobecný proces, ktorý spĺňa každý web scraper:

1. Identifikovanie cieľovej web stránky
2. Získanie adresy URL cieľovej stránky
3. Odoslanie požiadavku na danú adresu URL pre získanie zdrojového dokumentu
4. Extrakcia informácií zo zdrojového dokumentu
5. Uloženie dát, alebo ďalšia práca s dátami

Pri získaní zdrojového dokumentu môžeme rozdeliť web scraping na dve rôzne typy v závislosti od toho, ako je obsah dostupný na webovej stránke.

3.2.1 Statický web scraping

Obsah webovej stránky je **statický** a priamo doručený zo servera webovej stránky do prehliadača, takže kód HTML zostáva v takýchto prípadoch **nezmenený**. Vďaka tomu je statický web scraping porovnateľne rýchlejší ako dynamický, pretože načítanie údajov na webovej stránke je rýchlejšie. Tento prístup funguje dobre pre tradičné webové stránky, blogové príspevky, alebo statické stránky, ako je dokumentácia bez AJAX[3] alebo JavaScriptu pre dynamické načítanie obsahu [5].

3.2.2 Dynamický web scraping

Tento prístup extrahuje údaje z webových stránok, ktoré sa pri dynamickom načítavaní obsahu spoliehajú na AJAX a JavaScript. Obsah na takýchto webových stránkach sa načítava a **mení za behu**, čo sťažuje priame extrahovanie dát. Dynamický web scraping sa používa väčšinou vtedy, keď je potrebné zhromažďovať údaje z interaktívnych platforiem, online obchodov, alebo stránok sociálnych médií [5].

3.2.3 Headless browser

Hlavným dôvodom, prečo sa na web scraping používajú takzvané headless browsers, je to, že čoraz viac webových stránok je vytvorených pomocou SPA[4] frameworkov ako React.js, Vue.js, alebo Angular. Ak sa z takéhoto webu pokúsime extrahovať dáta bežným HTTP klientom, dostaneme prázdnu HTML stránku, pretože je postavená na front-end kóde Javascript. Headless browsers riešia tento problém spustením kódu Javascript, rovnako ako bežný prehliadač na počítači, ale bez grafického používateľského rozhrania [19].

3.3 Zabezpečenie proti web scrapingu

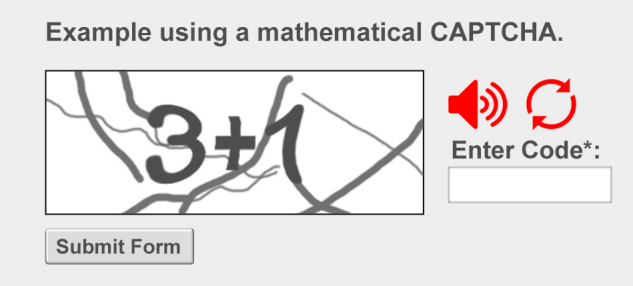
Z hľadiska bezpečnosti je web scraping netriviálnym problémom, keďže ide o pomerne často zaužívaný postup v mnohých spoločnostiach a má aj zákonité využitie. Spoločnosti môžu používať web scraping na webových stránkach pre veci, ako sú vyhľadávacie nástroje (search engines), alebo poskytovanie porovnávania cien spotrebiteľom. Je takmer nemožné zabrániť 100% všetkým pokusom a v konečnom dôsledku, je cieľom zvýšiť úroveň obtiažnosti web scrapingu [18].

[3] https://developer.mozilla.org/en-US/docs/Glossary/AJAX
[4] Single Page Application

Negatívne dopady spôsobené web scrapingom:

- opätovné zverejňovanie marketingového obsahu s cieľom odlákať používateľov od konkurencie.
- prístup k citlivým alebo súkromným informáciám, ako sú užívateľské profily, osobné údaje, alebo vlastnícke informácie.
- môže spotrebovať značné zdroje servera, ako je šírka pásma a výpočtový výkon, najmä ak sa vykonáva vo veľkom rozsahu, alebo so zlým úmyslom.
- môže porušovať zmluvné podmienky alebo zásady používania webových stránok.

Na ochranu pred web scrapingom existujú rôzne opatrenia, ako monitorovanie správania užívateľov, implementácia CAPTCHA[5] na overenie ľudských užívateľov a odradenie automatizovaných programov, alebo blokovanie IP adries s podozrivou aktivitou.

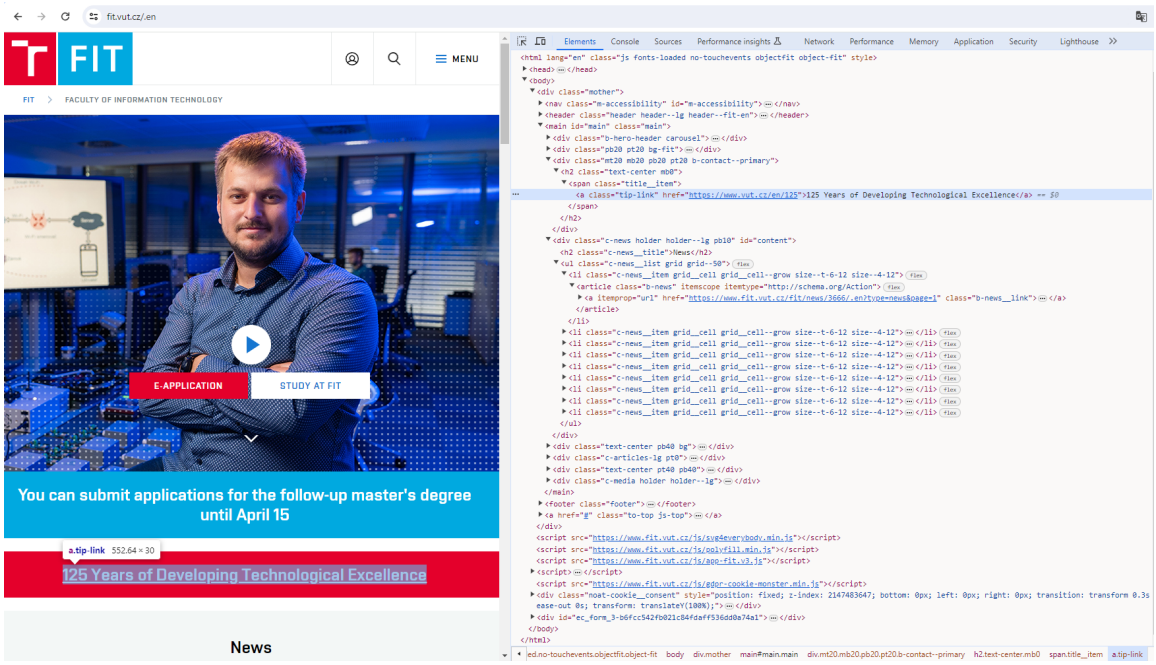


Obr. 3.1: Ukážka zabezpečenia pomocou CAPTCHA, prevzaté z [16]. Predstaví používateľom matematický problém, ktorý musia vyriešiť, aby dokázali, že sú ľudia a bol im umožnený vstup na stránku.

3.4 Extrakcia informácií z dokumentu HTML

Pri pokuse o extrakciu dát (parsing) zo zdrojového dokumentu HTML, býva prvým krokom použitie webového prehliadača a vývojárskych nástrojov na zobrazenie HTML kódu danej stránky. To umožňuje programátorovi **porozumieť** štruktúre HTML kódu a lokalizovať konkrétne prvky, ktoré chce extrahovať, ako napríklad text, obrázky, alebo odkazy. Následne pomocou zvoleného parsera **zozbiera** konkrétne prvky, ktoré v predchádzajúcom kroku lokalizoval. Rôzne parsery ponúkajú rôzne metódy, ktorými je možné konkrétny prvok vybrať. Tieto metódy dostávajú ako parameter takzvané **selektory**, čo sú vlastne reťazce, ktoré ukazujú na polohu daného prvku v HTML kóde. Medzi často používané selektory patria napríklad CSS, XPath alebo JQuery. Iným spôsobom ako prechádzať a manipulovať s dokumentom je s využitím aplikačného rozhrania DOM.

[5] http://www.captcha.net/

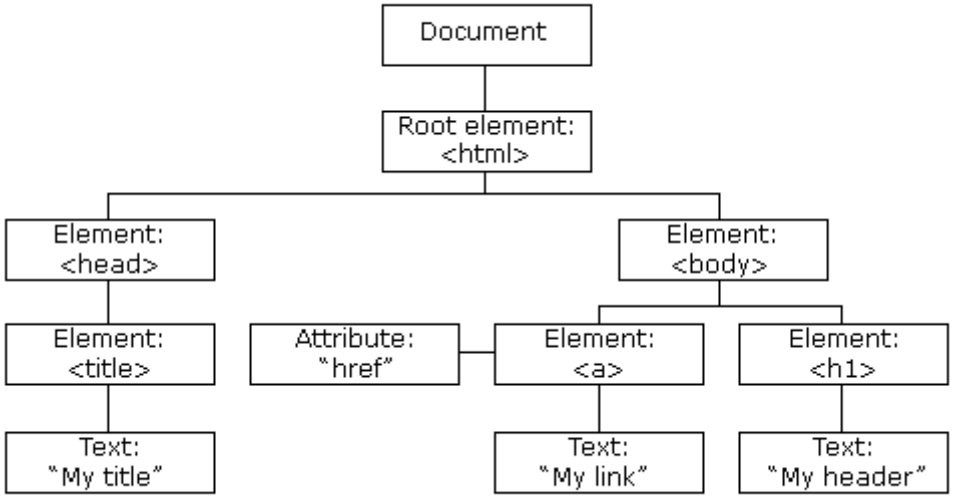


Obr. 3.2: Ukážka použitia vývojárskych nástrojov v prehliadači Google Chrome na analýzu štruktúry kódu HTML.

3.4.1 Document Object Model

Document Object Model (DOM) spojuje webové stránky so skriptami alebo programovacími jazykmi tým, že v pamäti reprezentuje štruktúru dokumentu, ako je napríklad HTML, ktorý reprezentuje webovú stránku. DOM reprezentuje takýto dokument pomocou **logického stromu**, kde každá vetva končí v uzle a každý uzol obsahuje objekty. Metódy DOM umožňujú programovo pristupovať k tomuto stromu pomocou skriptovacích jazykov, ako je napríklad JavaScript, a umožňujú zmeny v štruktúre, štýle, alebo obsahu dokumentu. Všetky vlastnosti, metódy a udalosti určené na manipuláciu a vytváranie webových stránok sú organizované do objektov, vrátane objektu dokumentu, ktorý reprezentuje samotný dokument, a objektov tabuliek, ktoré implementujú rozhranie DOM HTMLTableElement pre prístup k HTML tabuľkám [11].

DOM bol navrhnutý tak, aby bol nezávislý na konkrétnom programovacom jazyku. To umožňuje dostupnosť štruktúrovaného zobrazenia dokumentu cez jedno konzistentné API. Implementácie DOM-u môžu byť teda vytvorené pre akýkoľvek jazyk [11].



Obr. 3.3: DOM reprezentácia dokumentu HTML, prevzaté z [24].

Základné dátové typy DOM

- **Document** – tento objekt je samotný koreňový objekt dokumentu.
- **Node** – každý objekt nachádzajúci sa v dokumente je určitým typom uzla. V HTML dokumente môže byť objekt `Element` uzlom, ale tiež textovým uzlom, alebo uzlom atribútu.
- **Element** – objekty elementov implementujú rozhranie `DOM Element` a tiež základné rozhranie `Node`, ktoré sú oba zahrnuté spolu v tejto referencii.
- **NodeList** – pole prvkov `Element`. K položkám v `NodeList` je možné pristupovať pomocou indexu.
- **Attr** – je to odkaz na objekt, ktorý odhaľuje špeciálne (hoci malé) rozhranie pre atribúty.

3.4.2 CSS selektor

V CSS[6] sa používajú vzory, alebo inak nazývané selektory, na nájdenie zhody alebo výber prvkov, ktoré chce programátor naštýlovať. Selektory sa tiež používajú na vybratie uzlov DOM. Selektory, či už používané v CSS alebo inom programovacom jazyku, umožňujú **zamerať** HTML prvky na základe ich typu, atribútov, aktuálneho stavu, alebo dokonca aj pozície v DOM [9].

Selektory CSS sú pomerne jednoduché a intuitívne na použitie, vďaka čomu sú vhodnejšie pre jednoduché úlohy výberu prvkov v dokumentoch HTML. Existuje mnoho rôznych selektorov a spôsobov, ako dané selektory skladať. Spomenuté budú ale najčastejšie používané a relevantné pre použitie pri zameraní a výbere prvku v HTML dokumente.

[6] Cascading Style Sheets

- **Typový selektor** – vyberie všetky prvky, ktoré majú daný názov uzla, napríklad selektor `input` nám vyberie všetky prvky `<input>`. Patrí tu aj univerzálny selektor označený symbolom hviezdička (*), ktorý vyberá všetky prvky z dokumentu.
- **Selektor triedy** – vyberie všetky prvky, ktoré majú daný atribút triedy označený názvom triedy a znakom bodka (.) ako predponu. Takže selektor `.red` nám vyberie všetky prvky, ktoré obsahujú atribút `class="red"`.
- **Id selektor** – vyberie prvok na základe hodnoty jeho atribútu id. Selektor má tvar id s predponou znaku mriežky (#). Napríklad selektor `#testId` nám vyberie všetky prvky obsahujúce atribút `id="testId"`. Id je v dokumente globálny atribút a mal by sa v dokumente nachádzať iba jeden prvok s daným id. Ak je ich viac, tak sa zvyčajne vyberú všetky prvky s daným id, ale u niektorých nástrojov sa môže stať, že takúto funkcionalitu nepodporujú a zahlásia chybu.
- **Jednoduchý selektor** – Každý selektor, ktorý obsahuje jeden základný selektor, alebo selektor atribútu, je jednoduchý selektor.
- **Zložený selektor** – skladá sa z postupnosti jednoduchých selektorov, ktoré nie sú oddelené kombinátorom. Zložený selektor predstavuje súbor súbežných podmienok na jednom prvku. Takže zložený selektor `input#testId` by vybral prvok `<input>` s atribútom id nastaveným na testId.
- **Komplexný selektor** – Komplexný selektor je sekvencia jedného alebo viacerých jednoduchých a/alebo zložených selektorov, ktoré sú oddelené kombinátormi[7].

3.4.3 XPath

XPath je skratka pre jazyk XML[8] Path. Používa sa na poskytnutie flexibilného spôsobu adresovania rôznych častí XML dokumentu. Môže sa tiež použiť na testovanie adresovaných uzlov v dokumente na zistenie, či sa zhodujú s daným vzorom, alebo nie. XPath sa používa hlavne v XSLT[9], ale dá sa použiť aj ako oveľa mocnejší spôsob navigácie cez DOM akéhokoľvek dokumentu v jazyku XML pomocou XPathExpression, ako je napríklad HTML, alebo nájdenie konkrétneho prvku alebo atribútu v dokumente pomocou XPath predikátu [13].

XPath je preferovaný pred CSS selektormi pre svoju presnosť a pokročilé možnosti. Vyniká v detailnom výbere prvkov z hlboko vnorených webových stránok a ponúka veľké množstvo funkcií, ako napríklad funkcia *last()*, ktorá vyberie posledný uzol z aktuálnej množiny vyberaných uzlov. Na rozdiel od CSS selektorov umožňuje aj prístup k atribútom a textovým uzlom. Zatiaľ čo CSS selektory sú ideálne pre základné úlohy, XPath exceluje v zložitých scenároch. Keďže XPath je pomerne rozsiahly jazyk, spomenuté budú základné konštrukcie a výrazy pre výber prvkov.

Xpath výrazy

- `//` – vyberie uzly v dokumente z aktuálneho uzla, ktoré zodpovedajú výberu bez ohľadu na to, kde sa nachádzajú. Napríklad, ak chceme vybrať všetky prvky `<title>` v rámci celého dokumentu bez ohľadu na to, kde sa nachádzajú, môžeme použiť výraz `//title`.

[7] https://www.w3schools.com/css/css_combinators.asp
[8] Extensible Markup Language
[9] Extensible Stylesheet Language Transformations

- `/` – používa sa na určenie priameho vzťahu medzi uzlami. Vyberie uzly, ktoré sú bezprostrednými potomkami aktuálneho uzla. Ak chceme vybrať iba prvky `<title>`, ktoré sú priamymi potomkami prvkov `<movie>` v rámci prvku `<cinema>`, môžeme použiť vyraz `/cinema/movie/title`.
- `@` – používa sa na výber atribútov prvkov. Napríklad, ak chceme vybrať atribút `category` prvkov `<movie>`, môžeme použiť výraz `/movie/@category`.
- **predikát** – je výraz XPath napísaný v hranatých zátvorkách. Vzťahuje sa na obmedzenie vybraných uzlov v množine uzlov pre určitú podmienku.

  Príklady predikátov

  - `/input[4]` – vyberie v poradí štvrtý prvok `<input>`
  - `/input[last()]` – vyberie posledný prvok `<input>`
  - `/car[price = 80]` – vyberie prvok `<car><price>80</price></car>`

3.4.4 jQuery

Selektory jQuery sú metódy používané na výber prvkov HTML na základe rôznych kritérií, podobne ako selektory CSS. Umožňujú zacieliť na konkrétne prvky v DOM a vykonávať akcie ako manipulácia, prechod, spracovanie udalostí a ďalšie. Použitie jQuery nám poskytuje výkonný a flexibilný spôsob interakcie s prvkami na webovej stránke. Selektory jQuery sú výkonnejšie a flexibilnejšie ako selektory CSS, pretože podporujú ďalšie funkcie, ako napríklad filtrovanie na základe atribútov prvkov alebo obsahu, a výber prvkov na základe ich stavu [14].

jQuery ponúka niekoľko funkcií, ktoré nie sú dostupné v selektoroch CSS, ako napríklad:

- **:visible/:hidden** – umožňujú zacieliť na prvky na základe ich viditeľnosti na stránke. Pre vybratie všetkých viditeľných `<input>` prvkov by sme použili selektor `input:hidden`.
- **:eq()** – umožňuje zacieliť na prvky v konkrétnom indexe v rámci zhodnej množiny prvkov. Selektor `input:eq(3)` nám vyberie v poradí štvrtý prvok (indexovanie od nuly) `<input>`.
- **:has()** – umožňuje vybrať prvky, ktoré obsahujú aspoň jeden zodpovedajúci prvok. Selektor `div:has(p)` nám vyberie všetky prvky `<div>`, ktoré obsahujú aspoň jeden prvok `<p>`.

3.5 Extrakcia dát na platforme Java

Java je populárnou voľbou pre web scraping vďaka svojej všestrannosti a rozsiahlemu zoznamu dostupných knižníc. Dokáže zvládnuť zložité úlohy, automatizovať procesy a spracovávať veľké množstvo údajov. Silná podpora multithreadingu v jazyku Java umožňuje paralelné spracovanie viacerých webových stránok. Keďže zameranie tejto prace je na integráciu dát do informačných systémov, stojí za to spomenúť použiteľnosť Javy ako platformy. Java ponúka niekoľko výhod pre riadenie projektov IS[10], ako je spoľahlivosť, efektívnosť

[10]Information system

a všestrannosť. Java je spoľahlivá vďaka vysokej úrovni zabezpečenia, stability a robustnosti, ako aj funkciám, ako je automatická správa pamäte a spracovanie výnimiek, ktoré zabraňujú úniku pamäte, chybám a pádom.

3.5.1 Playwright

Playwright je knižnica na automatizáciu prehliadača pre `Node.js`, ktorá umožňuje spoľahlivú, rýchlu a efektívnu automatizáciu prehliadača pomocou niekoľkých riadkov kódu. Jednoduchosť a výkonné možnosti automatizácie z tejto knižnice robia ideálny nástroj na scrapovanie webu a data mining. Dodáva sa aj s podporou takzvaného headless browseru, čo umožňuje spracovanie dynamických webových stránok. Poskytuje bežné akcie, ako je interakcia s webovými prvkami a snímanie snímok. Podporuje aj paralelné vykonávanie a správu zdrojov, čo umožňuje rýchlejšie spracovanie viacerých stránok súčasne [19].

3.5.2 Selenium

Selenium je obzvlášť užitočný na web scraping dynamických webových stránok, ktoré sa pri generovaní obsahu alebo interakcii s používateľom vo veľkej miere spoliehajú na JavaScript. Mnoho moderných webových stránok načítava údaje asynchrónne alebo aktualizuje svoj obsah dynamicky, čím sú tradičné techniky web scrapingu menej efektívne. Vďaka schopnosti interagovať s webovými prvkami a spúšťať JavaScript, je veľmi vhodný na spracovanie takýchto webových stránok. Selenium podporuje niekoľko programovacích jazykov vrátane Pythonu, Java, C# a ďalších [22].

3.5.3 HtmlUnit

HTMLUnit je "GUI-Less"[11] webový prehliadač, ktorý poskytuje spôsob, ako simulovať interakcie prehliadača a automatizovať testovanie webových stránok. Umožňuje interakciu s webovými stránkami, odosielanie formulárov, klikanie na odkazy a programovo extrahovať obsah. Dokáže analyzovať/zobrazovať obsah HTML dokumentu a umožňuje manipulovať s DOM - to umožňuje úlohy, ako je extrahovanie textu, atribútov a vlastností prvkov z webových stránok [4].

3.5.4 Jsoup

Jsoup je Java HTML parser. Inak povedané, Jsoup je Java knižnica, ktorá umožňuje analyzovať akýkoľvek dokument HTML. Pomocou Jsoup je možné analyzovať lokálny súbor HTML alebo stiahnuť vzdialený dokument HTML z adresy URL. Jsoup tiež ponúka širokú škálu metód na prechádzanie DOM-u. Jsoup podporuje používanie selektorov CSS a metódy podobné Jquery, na výber a extrahovanie údajov z prvkov HTML [6].

[11] bez grafického užívateľského rozhrania

Kapitola 4

Návrh knižnice

Táto kapitola obsahuje upresnenie požiadavkov knižnice, popisuje jej architektúru a vybrané technológie, ktoré budú použité pri implementácii.

4.1 Špecifikácia zadania

Hlavnou úlohou knižnice bude integrácia dát z webových dokumentov do informačného systému, inak povedané, bude to prevedenie hodnôt z dokumentu na hodnoty v doméne daného informačného systému. Knižnica bude zameraná na platformu Java, presnejšie Java 17. Knižnica bude podporovať spracovanie webových dokumentov typu HTML. Bude zameraná na integráciu polo-štruktúrovaných dát, ktoré sú základom webových dokumentov (informácie viditeľné pri zobrazení stránky vo webovom prehliadači), narozdiel od štruktúrovaných, ktoré v dnešnej dobe ešte mnoho stránok nevyužíva. Štruktúrované dáta sú taktiež obmedzujúce v tom, aké informácie sú dostupné, čo je závisle na tvorcovi danej stránky.

Od programátora, ktorý bude túto knižnicu používať, bude vyžadované, aby dodal vlastnú časť implementácie, ktorá bude popisovať akým spôsobom, a ktoré údaje je potrebné z dokumentu spracovať. Taktiež bude popisovať, ako má vyzerať výsledná štruktúra Java objektu, ktorý bude výstupom tejto knižnice.

Po dohode bude knižnica podporovať minimálne autentizáciu pomocou jednoduchého formulára.

4.2 Návrh implementácie dodanej programátorom

Spôsob akým programátor popíše, ako sa má dokument spracovať, je inšpirovaný Java knižnicou Jackson[1], presnejšie, to akým spôsobom sa používajú anotácie na prispôsobenie procesu serializácie a deserializácie[2] JSON na objekty Java. V tejto práci nebudeme spracovávať JSON, ale môžeme sa na to pozerať, ako kvázi na deserializáciu HTML na objekt Java. Programátor teda dodá **popis triedy Java** s využitím anotácií, ktoré bude poskytovať knižnica.

Jackson je široko používaný v rôznych doménach vrátane vývoja webu, mikro služieb, spracovania údajov a ďalších. To, že bude formát popisu triedy podobný už existujúcim

[1] https://github.com/FasterXML/jackson
[2] https://www.geeksforgeeks.org/serialization-in-java/

nástrojom, by malo zjednodušiť použitie, bez nutnosti hlbšieho popisu nejakého nového formátu.

4.2.1 Java objekt a Java trieda

V Jave sú trieda a objekt úzko súvisiace pojmy, ale slúžia na rôzne účely, preto je dôležité pochopiť základné rozdiely.

Java trieda

Java trieda slúži ako plán alebo šablóna na vytváranie objektov. Definuje **vlastnosti** (polia) a **správanie** (metódy), ktoré budú mať objekty danej triedy. Triedy sú základnými stavebnými blokmi objektovo-orientovaného programovania (OOP) v Jave. Triedy sú deklarované pomocou kľúčového slova `class`, za ktorým nasleduje názov triedy, a ich obsah je uzavretý v zložených zátvorkách `{}`.

```
public class MyClass {
    private String myField;

    public void myMethod() {
        // some method implementation
    }
}
```

Výpis 4.1: Ukážka Java triedy.

Java objekt

Objekt je inštanciou triedy. Predstavuje **konkrétny výskyt** alebo **realizáciu** triedy, do ktorej patrí. Objekty majú stav (hodnoty svojich polí) a správanie (metódy). Objekty sa vytvárajú pomocou kľúčového slova `new`, za ktorým nasleduje názov triedy a parametre konštruktora.

```
MyClass myObject = new MyClass();
```

Výpis 4.2: Ukážka vytvorenia objektu pomocou triedy MyClass z výpisu 4.1.

4.2.2 Anotácie Java

Anotácie Java sú formou metadát, ktoré možno pridať do prvkov kódu Java, ako sú triedy, metódy, polia, parametre a ďalšie. Anotácie poskytujú dodatočné informácie o kóde pre nástroje, rámce a prostredie runtime. Používajú sa na prenos inštrukcií, konfigurácií alebo obmedzení do kompilátora runtime, alebo iných nástrojov.

Anotácie sa definujú pomocou symbolu `@`, za ktorým nasleduje názov anotácie, za ktorým nasledujú parametre v zátvorkách. Anotácie môžu byť vstavané (poskytované platformou Java) alebo vlastné (definované používateľmi alebo knižnicami tretích strán) [15].

```java
public class MyObject {
    // Date field with @JsonFormat annotation
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "UTC")
    private Date date;
}
```

Výpis 4.3: Ukážka popisu triedy obsahujúcej anotáciu @JsonFormat pre knižnicu Jackson. Keď je tento objekt serializovaný do JSON, pole dátumu bude naformátované podľa zadaného vzoru a časového pásma.

4.2.3 Návrh popisu triedy Java

Ako už bolo spomenuté, programátor dodá knižnici vlastnú implementáciu, čo bude vlastne popis jednoduchej Java triedy. Tento popis bude doplnený o vlastné anotácie nad jednotlivými poliami alebo triedami, ktoré bude knižnica poskytovať. Anotácia bude obsahovať parameter, podľa ktorej bude hodnota vyhľadaná v dokumente HTML.

Návrh vlastných anotácii:

- **@CssSelect** – hodnota tohto poľa bude naplnená hodnotou prvku z dokumentu HTML, ktorý bude zameraný použitím daného CSS selektoru.
- **@XPathSelect** – hodnota tohto poľa bude naplnená hodnotou prvku z dokumentu HTML, ktorý bude zameraný použitím daného XPath selektoru.
- **@ObjectSelect** – použitie pri vlastnej definovanej triede, bližšie popísanej v časti 4.2.3.
- **@DateTimePattern,** – pre špecifikovanie formátu, ktorý bude použitý pri konvertovaní hodnôt opísaných v časti 4.2.3.

Pri návrhu bola zvažovaná pôvodne aj anotácia `@ByClass`, kde bude prvok zameraný podľa daného atribútu *class*, a anotácie ako `@BySpan` a `@ByTitle` a iné, ktoré by zamerali prvok podľa nazvú prvku v dokumente. Vďaka tomu, ako sú údaje štruktúrované v dokumente HTML, je vo väčšine prípadov hľadaná hodnota zanorená vo veľkom množstve nadradených prvkov a použitie týchto anotácií nie je dostačujúce, a vyžaduje zameranie prvku pomocou CSS/XPath selektoru. V prípade potreby je takéto vyhľadanie prvku ľahko dosiahnuteľné pomocou CSS/XPath selektoru. Ak chceme vyhľadať prvok `<span>`, môžeme jednoducho použiť CSS selektor `span`, alebo XPath selektor `//span` pre dosiahnutie rovnakého výsledku.

```
public class MovieInfo {
   @CssSelect("title")
    private String title;

    @XPathSelect("div/span[1]")
    private Integer year;

    @XPathSelect("div/span[2]")
    private String duration;

    @XPathSelect("div/span[3]")
    private String ageRating;
}
```

Výpis 4.4: Ukážka popisu triedy s použitím anotácií nad poľom triedy.

Ak je anotácia pridaná nad poľom ako v príklade 4.4, bude po spracovaní dokumentu HTML tento objekt obsahovať nájdené hodnoty z dokumentu, ak sa tam takéto hodnoty nachádzali.

Podporované typy polí

Anotácie nad poľom budú fungovať len pre určité typy. Dokument HTML je textovo založený formát, dátový typ `String` je preto základný typ, ktorý bude knižnica podporovať. Často sa v dokumentoch vyskytujú aj číselné a časové hodnoty. Tieto hodnoty je možné pomerne ľahko prekonvertovať do ich príslušnej reprezentácie v jazyku Java, preto budú podporované aj typy `Integer`, `Double`, `LocalDate`, `LocalTime` a `LocalDateTime`. Pri numerických typoch (`Integer`, `Double`) a časových typoch (`LocalDate`, `LocalTime`, `LocalDateTime`) sa nájdená textová hodnota z dokumentu prekonvertuje na daný typ v jazyku Java ak je to možné (reťazec ”23” je možné previesť na `Integer` číslo 23, ale reťazec ”abc” nie).

Pole časového typu

Časové hodnoty sa v dokumentoch vyskytujú v rôznych formátoch (12-hodinový, 24-hodinový, ISO 8601 a pod.). Pri konverzii je dôležité vedieť aký formát použiť, aby bola výsledná hodnota poľa správna. Na túto funkcionalitu slúži anotácia `@DateTimePattern`, ktorú je možné použiť nad poľami typu `LocalDate`, `LocalTime` a `LocalDateTime` pre špecifikovanie vzoru, ktorý bude použitý pri konvertovaní tejto hodnoty.

```
public class ExampleClass {
    @CssSelect("div[class="date"]")
    @DateTimePattern("yyyy-MM-dd")
    private LocalDate date;

}
```

Výpis 4.5: Ukážka použitia anotácie @DateTimePattern na špecifikovanie formátu dátumu.

Pole typu List

Často nastane situácia, kedy sa v dokumente podarí nájsť viacero výskytov pre daný selektor. V takomto prípade sa priradí do poľa hodnota vždy prvého nájdeného výskytu. Ak budeme chcieť, aby pole obsahovalo hodnoty všetkých nájdených výskytov, bude potrebné definovať typ daného poľa ako `List`.

```
public class MovieInfo {
    @XPathSelect("span")
    private List<Integer> year;
}
```

Výpis 4.6: Ukážka popisu triedy.

Vďaka tomu, ako sú dáta usporiadané v HTML dokumente, často nastane situácia, kedy potrebujeme, aby bol výsledok spracovania zoznamom vlastnej definovanej triedy, napríklad zoznam triedy `MovieInfo` z príkladu 4.4. V takomto prípade bude možné použiť anotáciu `@ObjectSelect`. Triedu `MovieInfo` bude taktiež treba doplniť o anotáciu nad triedou, ktorá bude popisovať, ako rozdeliť dokument na jednotlivé celky, v ktorých sú údaje obsiahnuté.

```
<body>
    <moviesList>
        <movie>
            <title>Movie A</title>
            <div>
                <span class="year">1998</span>
                <span class="duration">1h 30m</span>
                <span class="ageRating">15</span>
            </div>
        </movie>
        <movie>
            <title>Movie B</title>
            <div>
                <span class="year">2005</span>
                <span class="duration">1h 55m</span>
                <span class="ageRating">PG-13</span>
            </div>
        </movie>
        <movie>
            <title>Movie C</title>
            <div>
                <span class="year">1974</span>
                <span class="duration">52m</span>
                <span class="ageRating">R</span>
            </div>
        </movie>
    </moviesList>
</body>
```

Výpis 4.7: Ukážka tela dokumentu obsahujúceho informácie o filmoch, ktoré potrebujeme spracovať.

V tomto prípade chceme, aby sme vo výsledku dostali zoznam objektov `MovieInfo`, kde bude každý objekt `MovieInfo` (reprezentujúci jeden film) obsahovať jednotlivé údaje pri-

slúchajúc danému filmu. V príklade 4.7 môžeme vidieť, že každý film je obsiahnutý v prvku `<movie>`. Do triedy `MovieInfo` teda pribudne anotácia nad triedou, ktorá popisuje, že dokument bude rozdelený po častiach podľa prvku `<movie>`. To dosiahneme nasledujúcim spôsobom:

```
@CssSelect("movie")
public class MovieInfo {
    @CssSelect("title")
    private String title;

    @XPathSelect("div/span[1]")
    private Integer year;

    @XPathSelect("div/span[2]")
    private String duration;

    @XPathSelect("div/span[3]")
    private String ageRating;
}

public class MovieList {
    @ObjectSelect
    private List<MovieInfo> movies;
}
```

Výpis 4.8: Ukážka popisu tried na spracovanie hodnôt z výpisu 4.7.

4.3 Vybranie nástroja pre prácu so zdrojovým dokumentom

Z knižníc spomenutých v 3.5, sa po preskúmaní bližších funkcionalít a možností každej knižnice, rozhodlo pre Jsoup. Jsoup je ako jediný zo spomínaných možností primárne zameraný na HTML parsing. Knižnice ako Playwright, Selenium a HtmlUnit obsahujú naviac mnoho funkcionalít, ako podpora pre automatizované testovanie stránok, mocking, zaznamenávanie snímok a videí, ktoré v tejto práci nebudeme potrebovať a vďaka ním sú tieto knižnice zložitejšie na použitie, a pomalšie pri spracovaní dokumentu. Vďaka tomu, že Jsoup nepatrí medzi headless browser-i 3.2.3, nevyžaduje ďalšie závislosti a eliminuje potrebu inštalácie a konfigurácie prehliadača, a príslušných WebDriver-ov. To značne uľahčuje jeho zahrnutie do projektu.

Nevýhoda knižnice Jsoup je možnosť spracovávať iba statické webové dokumenty. Je tu ale možnosť dynamicky dokument získať pomocou inej knižnice a následne iba predať tento (už statický) dokument knižnici Jsoup na spracovanie. To bude vyžadovať hlbšie zapojenie programátora, ktorý si bude musieť zvoliť vlastný nastroj na dynamické získanie dokumentu a konfiguráciu prehliadača.

4.4 Nástroj na zostavenie projektu

Nástroje na automatizáciu zostavovania, alebo nástroje na zostavovanie, sú aplikácie používané na automatizáciu kompilácie programu. Automatizácia zostavovania je dôležitým aspektom vývoja softvéru. Vzťahuje sa na proces automatizácie úloh potrebných na pre-

menu zdrojového kódu na spustiteľné programy. Najpopulárnejšie nástroje na tvorbu pre vývoj v jazyku Java sú Maven a Gradle [23].

Maven aj Gradle sú bezplatný softvér s open-source kódom, distribuovaný pod licenciou Apache 2.0. Oba sú vysoko prispôsobiteľné a podporované rôznymi Java IDE[3]. Gradle zostavovací skript je vo svojej podstate všestrannejší a výkonnejší ako Maven. Je to preto, že Gradle je založený na programovacom jazyku (Groovy), zatiaľ čo Maven je založený na značkovacom jazyku (XML). Na veľkých projektoch môže Gradle fungovať lepšie a rýchlejšie ako Maven, pre menšie projekty je rozdiel vo výkone Maven zanedbateľný. Je tiež známe, že Gradle má strmú krivku učenia, dokonca aj pre skúsených programátorov [23].

Vo výsledku bol pre tento projekt zvolený Maven, ktorý je vhodnejší a dostačujúci pre projekty malého rozsahu ako je tento.

4.5 Návrh testovania

Na otestovanie základných zameraní prvkov a konvertovaní hodnôt, bude knižnica obsahovať jednoduché testy. Zložitejšie testovanie bude prebiehať pripravením testovacích scenárov obsiahnutých v demo aplikácii. Demo bude obsahovať testovacie scenáre, kde bude na rôznych stránkach ukázané, že knižnica funguje správne. Jednotlivé stránky a prípady použitia/testovacie scenáre by mali byť dostatočne odlišné, aby bolo preverené, že knižnica je dostatočne flexibilná a vhodná pre širokú škálu použití. Pretože výstupom použitia knižnice bude Java objekt, ktorý existuje v pamäti, bude potrebné nájsť vhodný spôsob, ako takýto objekt zobraziť v jednoducho pochopiteľnej forme. Na toto zobrazenie bude použitá knižnica Jackson, ktorá serializuje Java objekt do JSON formátu, ktorý bude následne vypísaný na štandardný výstup.

[3]Integrated development environment

Kapitola 5

Implementácia

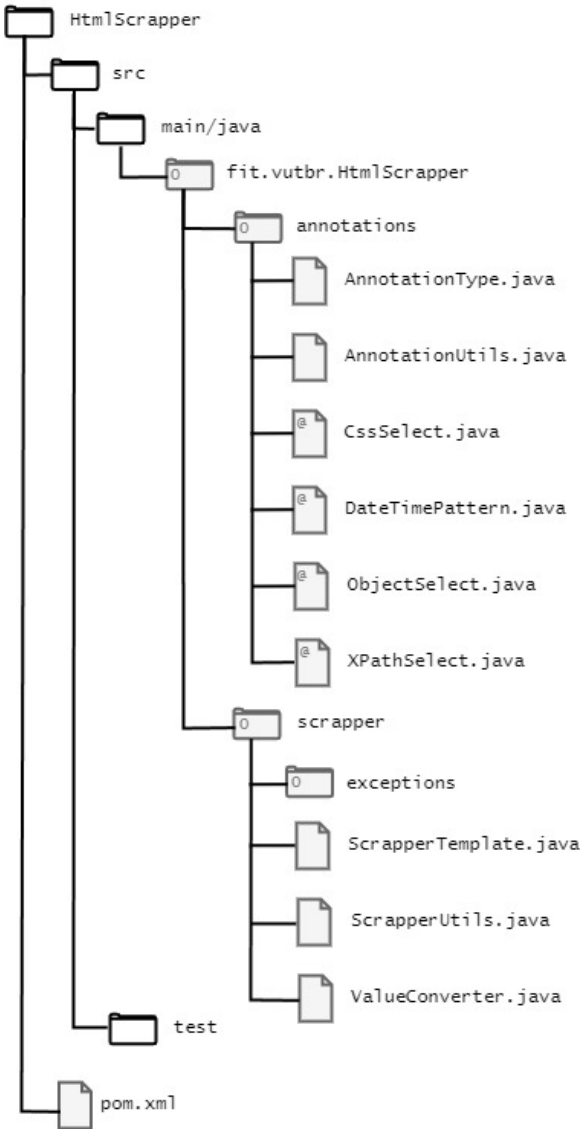
V tejto kapitole sú popísané implementácie jednotlivých častí navrhnutej knižnice. Knižnica obsahuje dve hlavné časti, ktorými sú definícia anotácií a hlavná logika programu, ktorej vstupným bodom je trieda `ScrapperTemplate`.

5.1 Štruktúra projektu

Nástroje použité pri implementácii:

- Java Development Kit 17
- vývojove prostredie IntelliJ IDEA 2023
- Maven 3
- Jsoup 1.17.2
- JUnit 4.13.1
- distribuovaný verzovací systém Git

Implementácia prebiehala s použitím vývojového prostredia IntelliJ IDEA. Projekt bol v IDEA-i založený ako Maven projekt, čo vygenerovalo základnú štruktúru a konfiguračný súbor pom.xml.



Obr. 5.1: Výsledná adresárová štruktúra projektu

Zdrojové súbory Java sú organizované do balíčkov (java packages). Balíček je mechanizmus na organizovanie tried a rozhraní do menných priestorov a pomáha predchádzať konfliktom v názvoch, uľahčuje modularizáciu a poskytuje hierarchickú štruktúru na organizovanie súvisiacich tried a rozhraní. Každý balíček má jedinečný názov a triedy/rozhrania v rámci balíka sú identifikované svojim plne kvalifikovaným menom, ktoré zahŕňa meno balíka, za ktorým nasleduje bodka (.) a názov triedy/rozhrania.

V projekte sa nachádzajú všetky zdrojové súbory v balíčku `fit.vutbr.HtmlScrapper` v zložke `src/main/java` koreňového adresára.

5.1.1 Konfiguračný súbor pom.xml

Súbor pom.xml (Project Object Model) je konfiguračný súbor, ktorý definuje štruktúru, závislosti a ďalšie nastavenia pre projekt Maven. Slúži ako centrálny konfiguračný súbor pre projekt Maven a poskytuje základné informácie potrebné na zostavenie, testovanie

a nasadenie projektu. Súbor pom.xml sa zvyčajne nachádza v koreňovom adresári projektu Maven a má formát XML.

Prvou dôležitou častou je sekcia `<properties>`, kde sa špecifikujú zdrojové a cieľové úrovne kompatibility pre kompilátor Java na verziu 17. Pre vyhnutie sa kompilačným chybám, vyžaduje táto knižnica Java verziu 17 alebo novšiu.

Ďalej je vhodné spomenúť sekciu `<dependencies>`, ktorá sa používa na deklarovanie závislostí pre projekt. Závislosti sú externé knižnice alebo moduly, ktoré projekt vyžaduje na kompiláciu, spustenie alebo testovanie kódu. Maven spravuje tieto závislosti automaticky stiahnutím potrebných súborov JAR zo vzdialených úložísk (v niektorých prípadoch lokálneho úložiska) a ich pridaním do cesty triedy projektu. Pre tento projekt je definovaná závislosť Jsoup vo verzii 1.17.2 a JUnit verzia 4.13.1, ktorá bude použitá pre základné testy opísané v časti 6.1.

Poslednou častou sú sekcie `<groupId>`, `<artifactId>` a `<version>`. Tieto prvky spolu tvoria súradnice artefaktu Maven. Používajú sa v systéme Maven na identifikáciu a lokalizáciu závislostí, riešenie konfliktov a stiahnutie požadovaných artefaktov z repozitárov. Pri deklarácii závislostí v `pom.xml` projektu, je potrebné zadať groupId, artifactId a verziu závislosti na označenie, od ktorej knižnice alebo modulu závisí daný projekt, a ktorá verzia by sa mala použiť.

5.2 Balíček annotations

V balíčku annotations sa nachádzajú zdrojové súbory potrebné pre definovanie vlastných anotácií, opísaných v častiach 4.2.3, a pomocné metódy na prácu s nimi.

V Jave sa definujú vlastné anotácie pomocou kľúčového slova `@interface`. Pri definícii je potrebné špecifikovať aj anotácie `@Retention` a `@Target`. Všetky anotácie v projekte budú obsahovať `@Retention(RetentionPolicy.RUNTIME)`. To určuje, že anotácia by mala byť zachovaná počas behu, čo k nej umožňuje prístup prostredníctvom reflexie. Túto vlastnosť budeme potrebovať pri spracovaní triedy v časti 5.4.3.

`@Target` určuje cieľ, na ktorý možno použiť túto anotáciu. Ako svoju hodnotu berie pole konštánt typu `ElementType`. V tomto projekte budú použité `ElementType.FIELD`, čo označuje, že anotáciu možno použiť na polia, a `ElementType.TYPE`, čo označuje, že anotáciu možno použiť na triedy, rozhrania, enumy a typy anotácií.

Vo vnútri anotácie je možné deklarovať metódy. Tieto metódy definujú atribúty anotácie a ich návratové typy definujú typy atribútov anotácie. Existuje tu špeciálny prípad metódy pomenovanej `value()`. Takto pomenovaná metóda umožňuje vynechať názov prvku pri zadávaní jeho hodnoty v deklarácii anotácie. Výraz `@Annotation(value = "someValue")` je možné potom jednoducho zapísať ako `@Annotation("someValue")`. Táto funkcia poskytuje stručnejšiu syntax pri používaní anotácií, najmä ak má anotácia iba jeden prvok.

5.3 Balíček exceptions

V jazyku Java je výnimkou (exception) udalosť, ktorá naruší normálny priebeh vykonávania programu. Predstavuje neočakávaný stav alebo chybu, ktorá sa vyskytne počas behu programu. Keď nastane výnimočný stav, Java vytvorí objekt známy ako ”objekt výnimky”, ktorý predstavuje špecifický typ chyby.

Existujú dve hlavné typy výnimiek:

- **strážené výnimky (checked exceptions)** – je potrebné ich explicitne spracovať. Predstavujú podmienky, ktoré môže metóda predvídať a je možné sa z nich zotaviť.
- **nestrážené výnimky (unchecked exceptions)** – výnimky, ktoré nevyžadujú aby boli explicitne spracované a predstavujú zvyčajne nejakú chybu v logike programu.

V balíčku `exceptions` sa nachádza definícia dvoch strážených výnimiek používaných v tejto knižnici:

- **ScrapperTemplateException** – predstavuje všeobecnú chybu kniznice, ktorá nastáva zvyčajne pri chybách v spracovaní triedy s použitím reflection API.
- **ScrapperTemplateSourceDocumentDownloadException** – chyba pri získavaní zdrojového dokumentu.

5.4 ScrapperTemplate

Typicky obsahuje Java aplikácia funkciu `main()`, ktorá je vstupným bodom pre aplikáciu Java. Je to počiatočný bod, z ktorého Java Virtual Machine (JVM) spúšťa program. Tento projekt takúto funkciu **neobsahuje**, keďže je určený ako knižnica, ktorá bude použitá ako závislosť v iných aplikáciách a informačných systémoch Java.

Vstupným bodom pre túto knižnicu je trieda `ScrapperTemplate`, nachádzajúca sa v balíčku `scrapper`, a jej metóda `scrapeData()`. Táto trieda obsahuje hlavnú logiku knižnice, ktorú je možné rozdeliť do nasledujúcich krokov:

- získanie/stiahnutie zdrojového dokumentu HTML
- vytvorenie potrebných objektov podľa popisu triedy predanej metóde `scrapeData()`
- zameranie hodnôt v dokumente podľa zadaných selektorov
- naplnenie polí vytvorených objektov nájdenými hodnotami
- vrátenie Java objektu ako výsledku

5.4.1 Získanie zdrojového dokumentu

Po zavolaní metódy `scrapeData()` je prvým krokom získanie zdrojového dokumentu z URL predaného do metódy v parametri *url*. Pre získanie dokumentu je využitá knižnica Jsoup a jej metódy `connect()` a `get()`. Výsledkom je dokument ako Java objekt typu `Document`, s ktorým dokáže ďalej knižnica Jsoup jednoducho pracovať. Pri neúspešnom pokuse o získanie dokumentu nám vyhodí knižnica výnimku popísanú v sekcii 5.3.

5.4.2 Možnosti autentizácie

Existuje veľké množstvo spôsobov, ktoré môže webová stránka využívať na autentizáciu, ako použitie cookies, tokenov, alebo nástroje tretích strán (napr. OAuth). Knižnica preto poskytuje spôsob, ako jednoducho zahrnúť rôzne spôsoby autentizácií v prípade potreby.

@Override

Anotácia `@Override` sa používa na označenie toho, že metóda v podtriede prepisuje metódu svojej nadtriedy. Inými slovami, podtrieda poskytuje špecifickú implementáciu metódy, ktorá bola deklarovaná jednou z jej rodičovských tried. Tento mechanizmus môžeme použiť na to, aby si každý programátor dokázal vytvoriť vlastnú implementáciu autentizácie ktorú potrebuje, vytvorením podtriedy z triedy `ScrapperTemplate` a prepísaním metódy `getDocumentFromUrl(String url)`.

```
public class ScrapperTemplateWithLogin extends ScrapperTemplate {
    @Override
    protected Document getDocumentFromUrl(String url) {
        // some method implementation with authentication that returns
            Document object
    }
}
```

Výpis 5.1: Ukážka prepísania metódy.

5.4.3 Vytvorenie potrebných objektov podľa popisu triedy

Výstupom knižnice je Java objekt vytvorený z popisu triedy predanej metóde `scrapeData()` v parametri *clazz*. Na spracovanie tejto triedy je použité Java Reflection API. Okrem vytvorenia objektov je použitá reflexia taktiež na spracovanie jednotlivých polí triedy a ich anotácií obsahujúce selektory, ktoré budú ďalej použité pri zameraní hodnôt v dokumente.

Java Reflection API[1]

Java Reflection API je mechanizmus, ktorý umožňuje manipulovať s triedami, metódami, poľami a ďalšími komponentami programu Java dynamicky za behu. Poskytuje spôsob, ako preskúmať štruktúru, správanie a metadáta tried a objektov Java, aj keď kód nie je dostupný v čase kompilácie. Reflexia sa často používa v scenároch, kde štruktúra tried alebo objektov nie je vopred známa, ako napríklad v rámcoch, knižniciach a nástrojoch, ktoré vyžadujú úpravu kódu za behu.

5.4.4 Zameranie hodnôt v dokumente

Na získanie hodnôt z dokumentu sa používajú metódy knižnice Jsoup. Na vyhľadanie prvku sú použite metódy `select()` alebo `selectXpath()`, podľa anotácie definovanej nad príslušným poľom. Keďže sa snažíme získať z webového dokumentu hodnoty, ktoré sú reálne zobrazené vo webovom prehliadači, hodnoty sa budú vyhľadávať iba v časti `<body>` webového dokumentu. Ďalej môžeme predpokladať, že naša hodnota sa bude vo výsledku nachádzať vždy vo vnútri prvku (<prvok>*hľadaná hodnota*<prvok>). Pre získanie vnútornej hodnoty z prvku je použitá Jsoup metóda `html()`.

5.4.5 Naplnenie polí vytvorených objektov nájdenými hodnotami

Pri získaní prvku pomocou metódy `html()`, je výsledok vždy objekt typu `String`. Pre konverziu na iné typy je vytvorená trieda `ValueConverter`.

[1] Application Programming Interface

Pri číselných hodnotách bývajú vo webových dokumentoch použité rôzne formáty, kde môže byť oddeľovač desatinných miest bodka (.) alebo čiarka (,). Rôzny zápis nastáva aj pri oddeľovaní tisícok, kde sa môže použiť medzera alebo čiarka. Konverzia z reťazového typu do číselného typu preto vyžaduje, aby bol reťazec najprv prevedený do normovaného tvaru bez oddeľovačov tisícok a s bodkou ako oddeľovačom desatinných miest.

Pri časových typoch sa na konverziu použije vzor definovaný v anotácii `@DateTimePattern`, alebo štandardný ISO-8601[2] formát, ak táto anotácia chýba. Vzor odpovedá formátom definovaným pre Java triedu `DateTimeFormatter`[3].

Po úspešnej konverzii je hodnota priradená poľu objektu pomocou reflexie. Dáta na webových stránkach sú veľmi rôznorodé, často bude treba výsledný objekt ďalej manuálne spracovať, aby sme z jednotlivých reťazcov dostali podrobnejšie dáta.

[2] https://www.iso.org/iso-8601-date-and-time-format.html
[3] https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

Kapitola 6

Testovanie

Knižnica obsahuje jednoduché testy, napísané pomocou knižnice JUnit[1], na overenie základných zameraní prvkov a konvertovaní hodnôt podľa vzoru. Na ďalšie testovanie knižnice je vytvorená demo aplikácia, obsahujúca testovacie scenáre pre rôzne stránky, ktoré majú overovať, že knižnica funguje správne a je vhodne použiteľná pre širokú škálu stránok.

6.1 Základné testy knižnice

Testy sa nachádzajú v zložke `src/test` koreňového adresára v triede *HtmlScrapperTest*. Na testovanie bol vytvorený jednoduchý HTML dokument `test.html` nachádzajúci sa v zložke `src/test/resources`. Testy využívajú upravenú implementáciu v triede *ScrapperTemplateLocal*, kde sa namiesto stiahnutia zdrojového dokumentu z internetu, načíta testovací dokument `test.html`.

Tieto testy sú zamerané na overenie zamerania prvkov pomocou Css/Xpath selektorov, konvertovanie hodnôt na numerické a časové typy, overenie fungovania použitia vzoru v anotácii `@DateTimePattern`, a overenie výnimky pri neúspešnom pokuse o získanie zdrojového dokumentu.

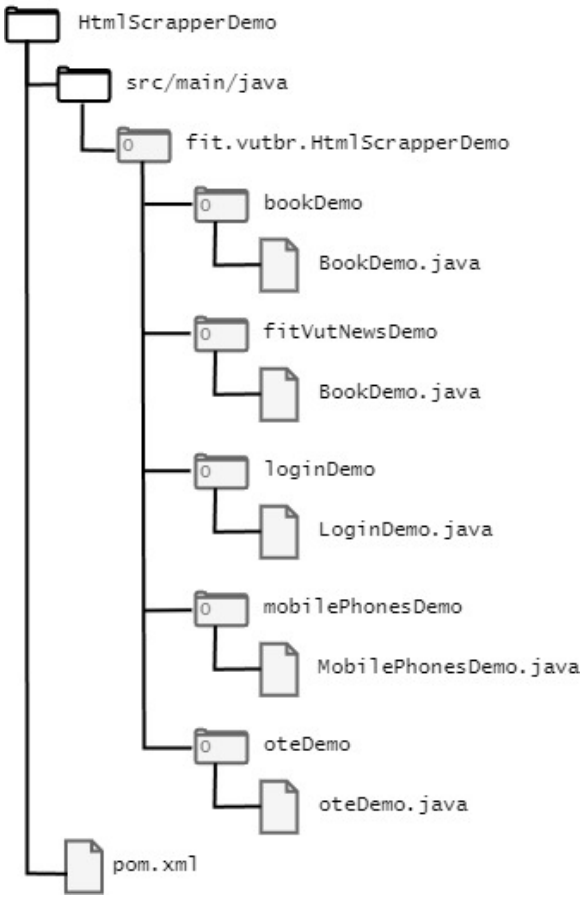
6.2 Demo aplikácia

Demo aplikácia obsahuje zložitejšie testovacie scenáre na reálnych webových stránkach. Keďže výsledkom použitia knižnice je Java objekt v pamäti, je použitá knižnica Jackson na vhodné zobrazenie reprezentácie tohto objektu, ktorý bude následne vypísaný v JSON formáte na štandardný výstup. Dáta na stránkach sa môžu často meniť, preto nie je možné stav takéhoto objektu overiť voči pevne zadaným hodnotám, ako to bolo v testoch 6.1, a musia sa overiť vizuálne podľa výpisu na štandardný výstup.

6.2.1 Štruktúra demo aplikácie

Aplikácia je, rovnako ako implementácia knižnice, Maven projekt. Súbor `pom.xml` má pridanú závislosť implementácie knižnice 5, čo nám umožní ju použiť v tejto aplikácii. Na to, aby Maven tuto závislosť rozoznal, je nutné ju nahrať do lokálneho repozitára Maven spustením príkazu `mvn install` v adresári, ktorý obsahuje `pom.xml` súbor. Testovacie scenáre sú rozdelené do jednotlivých balíčkov a každý obsahuje spustiteľnú triedu s `main()` funkciou.

[1] https://junit.org/junit4/



Obr. 6.1: Adresárová štruktúra demo aplikácie s hlavnými triedami obsahujúcimi main() funkciu.

bookDemo

Cieľom tohto scenára je zozbierať údaje o aktuálne novo vydaných knihách. Scenár testuje použitie anotácie `@ObjectSelect` a extrakciu hodnoty atribútu z prvku pomocou funkcie `extractAttributeValue`.

fitVutNewsDemo

Tento scenár zozbiera údaje z noviniek stránky. Testuje sa tu viacnásobné volanie knižnice po sebe. Prvé volanie zozbiera odkazy na jednotlivé články a následne sa spracovávajú údaje z každého zozbieraného odkazu. Testuje sa tu aj konverzia dátumu na typ `LocalDate`.

loginDemo

Scenár slúži na otestovanie prihlásenia na fórum. Obsahuje upravenú implementáciu metódy `getDocumentFromUrl()` na prihlásenie pomocou formulára.

mobilePhonesDemo

Scenár zozbiera parametre o mobilných telefónoch stránky. Slúži na ukážku ďalšieho spracovania objektu, ktorý je výstupom knižnice. Vďaka tomu, ako sú údaje formátované na danej stránke, je nutné nad nimi previesť manuálnu syntaktickú analýzu.

oteDemo

Na otestovanie konverzie do číselných typov `Integer` a `Double`, slúži tento scenár, ktorý zozbiera ceny elektriny za daný deň.

Kapitola 7

Záver

Cieľom bakalárskej práce bolo vytvoriť knižnicu v programovacom jazyku Java, pre integráciu dát z webových zdrojov do informačných systémov. Po preskúmaní možností prezentácie dát vo webových dokumentoch a technológií používaných na extrakciu dát, bola pri implementácií použitá knižnica Jsoup.

Knižnica ponúka programátorom možnosť, ako rýchlo a jednoducho popísať objekty Java a priradiť im hodnoty z webového dokumentu. Na zameranie hodnôt v dokumente je možné použiť selektory CSS alebo XPath. V popise objektu je možné používať aj podporované numerické a časové typy, kde bude reťazcová hodnota z dokumentu prekonvertovaná na daný typ. Knižnica poskytuje možnosť prepísať časť implementácie, čo umožňuje programátorovi definovanie vlastného spôsobu autentizovania voči webovej stránke. Pri implementácii vznikla aj demo aplikácia využitá pri testovaní, ktorá slúži zároveň aj na ukážku použitia knižnice.

Keďže sa dáta na webových stránkach vyskytujú v rôznych formátoch, býva často potreba výsledný objekt ešte dodatočne manuálne spracovať. Prípadným rozšírením by preto mohlo byť zakomponovanie tohto spracovania v rámci knižnice. Ďalším rozšírením by bolo pridanie ďalších anotácií na spracovanie hodnôt atribútov v prvkoch HTML.

Novo vyvinutá Java knižnica uľahčuje integrovanie webových dát do informačných systémov a ponúka programátorom zjednodušený proces definovania Java objektov, a extrakcie hodnôt z webových dokumentov. Okrem schopnosti prispôsobiť metódy autentizácie, knižnica nielen zjednodušuje integráciu, ale otvára aj cesty pre budúce rozšírenia, ako je zahrnutie manuálneho spracovania údajov a zavedenie dodatočných anotácií na spracovanie atribútov.

Literatúra

[1] BITFARM INFORMATIONSSYSTEME GMBH. *Web Documents: Definition Usage* [online]. [cit. 2024-05-05]. Dostupné z: https://www.bitfarm-archiv.com/document-management/glossary/web-documents.html.

[2] DATAHUT. *Web Scraping vs API: What's the best way to extract data* [online]. [cit. 2024-05-05]. Dostupné z: https://www.blog.datahut.co/post/web-scraping-vs-api.

[3] DUBOIS, A. *What Is Web Scraping: The Ultimate Beginner's Guide* [online]. [cit. 2024-05-05]. Dostupné z: https://proxyway.com/guides/what-is-web-scraping.

[4] GARGOYLE SOFTWARE INC.. *HtmlUnit* [online]. 2024 [cit. 2024-05-05]. Dostupné z: https://htmlunit.sourceforge.io/.

[5] GUPTA, V. *How To Perform Web Scraping with Selenium Java* [online]. [cit. 2024-05-05]. Dostupné z: https://www.lambdatest.com/blog/web-scraping-with-selenium-java/.

[6] HEDLEY, J. *How to Use Selenium to Web Scrape* [online]. [cit. 2024-05-05]. Dostupné z: https://jsoup.org/.

[7] IONOS. *Structured data: an introduction* [online]. [cit. 2024-05-05]. Dostupné z: https://www.ionos.com/digitalguide/websites/website-creation/structured-data-an-introduction/.

[8] MICROFORMATS.ORG. *Microformats* [online]. 2024 [cit. 2024-05-05]. Dostupné z: https://microformats.org/.

[9] MOZILLA CORPORATION. *CSS selectors* [online]. [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_selectors.

[10] MOZILLA CORPORATION. *HTML: HyperText Markup Language* [online]. [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[11] MOZILLA CORPORATION. *Introduction to the DOM* [online]. [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[12] MOZILLA CORPORATION. *Microdata* [online]. [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML/Microdata.

[13] MOZILLA CORPORATION. *XPath* [online]. [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/XPath.

[14] OpenJS Foundation. *Jquery* [online]. 2024 [cit. 2024-05-05]. Dostupné z: https://jquery.com/.

[15] Oracle Corporation. *Lesson: Annotations* [online]. 2022 [cit. 2024-05-05]. Dostupné z: https://docs.oracle.com/javase/tutorial/java/annotations/index.html.

[16] radware. *What is a CAPTCHA and How Do CAPTCHAs Work?* [online]. [cit. 2024-05-05]. Dostupné z: https://www.radware.com/cyberpedia/bot-management/captcha/.

[17] RDFa. *RDFa* [online]. 2024 [cit. 2024-05-05]. Dostupné z: https://rdfa.info/docs.

[18] Rogin, C. *Understanding and Preventing Website Content Scraping* [online]. [cit. 2024-05-05]. Dostupné z: https://fingerprint.com/blog/website-content-scraping-prevention/.

[19] Sahin, K. *Scraping The Web With Playwright* [online]. [cit. 2024-05-05]. Dostupné z: https://www.scrapingbee.com/blog/playwright-web-scraping/.

[20] Sporny, M. *JSON-LD: Core Markup* [online]. [cit. 2024-05-05]. Dostupné z: https://www.youtube.com/watch?v=UmvWk_TQ30A.

[21] Sporny, M. *What is JSON-LD?* [online]. [cit. 2024-05-05]. Dostupné z: https://www.youtube.com/watch?v=vioCbTo3C-4.

[22] Varatiya, M. *How to Use Selenium to Web Scrape* [online]. [cit. 2024-05-05]. Dostupné z: https://www.scaler.com/topics/selenium-web-scraping/.

[23] Vartanian, E. *Java build tools: Maven vs Gradle* [online]. [cit. 2024-05-05]. Dostupné z: https://www.educative.io/blog/java-build-tools-maven-vs-gradle.

[24] W3Schools. *JavaScript HTML DOM* [online]. [cit. 2024-05-05]. Dostupné z: https://www.w3schools.com/js/js_htmldom.asp.

[25] Woolard, C. *What's the difference between structured, semi-structured, and unstructured data?* [online]. [cit. 2024-05-05]. Dostupné z: https://automationhero.ai/blog/whats-the-difference-between-structured-semi-structured-and-unstructured-data/.

Príloha A

Obsah pamäťového média

- document.pdf - písomná časť bakalárskej práce
- HtmlScrapper.zip - zdrojový kód knižnice
- HtmlScrapperDemo.zip - zdrojový kód demo aplikácie
- latex.zip - zdrojový kód latexu
- README.md - návod na spustenie demo aplikácie