FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání diplomové práce

| | |
|---|---|
| **Název:** | Konfigurovatelný web scraper internetového zpravodajství |
| **Student:** | Bc. Martin Thern |
| **Vedoucí:** | Ing. Jiří Šmolík |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Webové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2024/2025 |

## Pokyny pro vypracování

Cílem práce je vytvořit web scraper na stahování a extrakci dat z webových stránek zpravodajských portálů za pomoci člověkem předpřipraveného nastavení.

1. Navrhněte, implementujte a otestujte webové rozhraní pro správu nastavení scraperu.
   a) Nastavení scraperu se skládá ze seznamu zdrojů/zpravodajského portálu, který je možný upravovat. Každý zdroj obsahuje minimálně RSS URL, frekvenci stahování a nastavení extrakce dat z HTML, které je využito pro pravidelné automatické stahování obsahu.
   b) Rozhraní pro nastavení extrakce ukazuje náhled článku zpravodajského portálu, náhled na extrahovaná data dle konfigurace a možnost upravit konfiguraci extrakce.
   c) Implementujte výchozí možnosti extrakce, např. s využitím sémantických dat webu nebo často používaných tříd, případně dalších heuristik tak, aby manuální nastavení extrakce nebylo vždy nutné.
2. Navrhněte, implementujte a otestujte scraper, který bude dle výše definovaného správcovského rozhraní pravidelně stahovat články, extrahovat z nich data a ukládat do databáze.
3. Implementujte klientské webové rozhraní pro prohlížení stažených článků s jednoduchým vyhledáváním.
4. Implementujte verzování článků tak, aby bylo možné se podívat na obsah článku v historii, pokud došlo ke změně.

---

*Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 1. listopadu 2023 v Praze.*

Diplomová práca

# KONFIGUROVATEĽNÝ WEB SCRAPER INTERNETOVÉHO SPRAVODAJSTVA

**Bc. Martin Thern**

Fakulta informačních technologií
Informatika
Vedúci: Ing. Jiří Šmolík
8. mája 2024

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Zoznam výpisov kódu

# Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce Ing. Jiřímu Šmolíkovi za jeho čas, odborné vedenie a cenné rady počas riešenia mojej záverečnej práce. Rovnako by som sa rád poďakoval svojim rodičom a priateľom za ich podporu a povzbudzovanie počas celého môjho štúdia.

# Vyhlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržovaní etických princípov pri príprave vysokoškolských záverečných prác. Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplívajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, ve znění pozdějších předpisů, najmä skutočnosť, že České vysoké učenie technické v Prahe má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 ods. 1 citovaného zákona.

V súlade s ust. § 2373 ods. 2 zákona č. 89/2012 Zb., Občiansky zákonník, v znení neskorších predpisov, týmto udeľujem nevýhradné oprávnenie (licenciu) na použitie tohto autorského diela, a to vrátane všetkých počítačových programov a všetkej ich dokumentácie (ďalej súhrnne len „Dielo“), a to všetkým osobám, ktoré si želajú Dielo použiť. Tieto osoby sú oprávnené Dielo použiť akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela,a za akýmkoľvek účelom (vrátane použitia na zárobkové účely). Toto oprávnenie je časovo, teritoriálne i množstevne neobmedzené. Každá osoba, ktorá využije vyššie uvedenú licenciu, sa však zaväzuje udeliť ku každému dielu, ktoré vznikne (hoci len sčasti) na základe Diela, úpravou Diela, spojením Diela s iným dielom, zaradením Diela do diela súborného či spracovaním Diela (vrátane prekladu) licenciu aspoň vo vyššie uvedenom rozsahu a zároveň sprístupniť zdrojový kód takého diela aspoň porovnateľným spôsobom a v porovnateľnom rozsahu, ako je sprístupnený zdrojový kód Diela.

V Praze dňa 8. mája 2024

# Abstrakt

Diplomová práca sa zaoberá návrhom a implementáciou konfigurovateľného webového scrapera pre extrahovanie dát zo spravodajských portálov s využitím RSS kanálov a používateľom nastavenými pravidlami extrakcie. Ovládanie scrapera bude možné cez webovú aplikáciu.

Úvodná časť mojej diplomovej práce sa zaoberá analýzou spravodajských portálov a ich variantami publikácie dát. Súčasne sa zaoberá obecným priblížením fungovania web scrapingu.

Druhá časť práce sa zaoberá analýzou existujúcich riešení, ktoré ponúkajú web scraping ako SaaS službu. Zároveň sú analyzované aj programovacie jazyky a ich rozšírenia, ktoré sa na web scraping využívajú.

Tretia časť práce sa zaoberá vytvorením funkčných a nefunkčných požiadaviek podľa vyhodnotenia z analýzy, súčasne návrhu používateľského rozhrania.

Štvrtá časť diplomovej práce sa zaoberá implementáciou funkčných požiadaviek a následného testovania webovej aplikácie.

Webová aplikácia je implementovaná pomocou Node.js, frameworku React a databázy MongoDB.

**Klíčová slova** Web scraper, React, Node.js, MongoDB, webová aplikácia, extrahovanie dát, spravodajské portály, RSS kanály

# Abstract

The master thesis deals with designing and implementing a configurable web scraper for extracting data from news portals using RSS channels and user-set extraction rules. Control of the scraper will be possible through a web application.

The first part of my master's thesis deals with analyzing news portals and their variants of publishing data. Simultaneously, it deals with the general approach to web scraping.

The second part of the work deals with the analysis of the existing solutions, which offer web scraping as a SaaS service. At the same time also analyzes the programming languages and their expansions, which are used for web scraping.

The third part of the work deals with the creation of functional and non-functional requirements from the result of the analysis, simultaneously designing of user interface.

The fourth part of the master thesis deals with the implementation of functional and non-functional requirements and the following testing of the web application.

The web application is implemented in Node.js, the framework React and database MongoDB.

**Keywords** Web scraper, React, Node.js, MongoDB, web application, data extraction, News portals, RSS feeds

# Zoznam skratiek

| | |
|---|---|
| API | Application Programming Interface |
| CSS | Cascading Style Sheets |
| CSV | Comma Separated Values |
| DevTools | Developer Tools |
| DOM | Document Object Model |
| DQL | Diffbot Query Language |
| FTP | File Transfer Protocol |
| HTML | HyperText Markup Language |
| HTTPS | Hypertext Transfer Protocol Secure |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| JSON-LD | JavaScript Object Notation for Linked Data |
| RDFa | Resource Description Framework in attributes |
| RSS | Really Simple Syndication |
| SaaS | Software as a Service |
| SOCKS | Socket Secure |
| UI | User Interface |
| URL | Uniform Resource Locator |
| UX | User Experience |
| XML | Extensible Markup Language |

# Úvod

Prístup ku informáciám nebol nikdy jednoduchší ako v dnešnej dobe. Dnes nám k tomu stačí len smartfón a pripojenie na internet a môžeme vyhľadávať informácie, hrať hry, rezervovať si miesta vo vlaku či v kine, nakupovať, alebo mnoho iných vecí. Webových stránok je na svete veľké množstvo, podľa dostupných štatistík ku februáru 2023 ide približne o 1.13 miliardy webových stránok z ktorých je však už 82% neaktívnych. Z toho teda vyplýva, že približne 200 miliónov webových stránok je aktívnych pre používateľov [1]. To znamená obrovské množstvo dát a informácií, ku ktorým máme jednoduchý prístup. Tieto dáta a informácie môžu slúžiť rôznym účelom. Jedným z nich je naše osobné obohacovanie vedomostí, a teda snaha nájsť webové stránky, ktoré obsahujú informácie potrebné pre vyriešenie konkrétneho problému, alebo informácie ktoré rozšíria naše obzory. Hľadáme však aj dáta pre relaxáciu, či už sú to videá, hudba alebo aj literárne diela. Ďalším účelom využívania web stránok sú online obchody, ktoré ponúkajú zakúpenie rozličných produktov jednoducho aj z pohodia domova, avšak webových stránok určených pre rôzne účely je ešte mnoho. Pri obrovskom množstve webových stránok môže byť komplikované a časovo náročné nájsť tie správne informácie, alebo tú najlepšiu cenu pre hľadaný produkt, prezeraním jednej stránky za druhou. Preto sa využívajú systémy, ktoré skúmajú, získavajú, spracovávajú alebo vyhodnocujú dáta poskytnuté na webových stránkach, a poskytujú užitočné informácie o nich. Tieto systémy sa nazývajú webové crawlery alebo scrapery. Spoločnosť Google využíva niekoľko svojich webových crawlerov[2], ktoré prechádzajú jednotlivé stránky a získavajú z nich potrebné informácie, ktoré spracujú a využívajú pri indexovaní webových stránok, aby internetový vyhľadávač Google dokázal efektívne vyhľadávať potrebné záznamy. Web crawlery a scrapery získavajú rozličné dáta z webových stránok podľa ich špecifikácií, a tieto údaje sú využívané napríklad pri porovnávaní cien rovnakého produktu na rozličných stránkach, sú využívané pri analýze trhu, aktuálnych trendoch v spoločnosti, alebo sa tieto dáta spracovávajú a využívajú pri strojovom učení, či umelej inteligencii. Medzi webové stránky, ktoré poskytujú veľké množstvo informácií rozhodne patria aj spravodajské portály. Tieto portály nás denne informujú o aktuálnych dianiach doma, ale aj vo svete a obsahujú veľké množstvo informácií. Udržovať prehľad nad novými článkami však môže byť dosť komplikované, a preto je vhodné využívať web crawlery a scrapery pre získavanie potrebných informácií. Vďaka web scraperom a crawlerom vieme získavať stále aktuálne dáta z viacerých zdrojov naraz, a tieto dáta môžme využiť na ďalšie spracovanie. Napríklad pri odhaľovaní falošných a nepravdivých správ, spracovávať a vytvárať znalostné grafy, využiť ich pri trénovaní strojového učenia, ale aj na mnohé iné účely. Webové scrapery sú dnes ponúkané ako služby pre širokú verejnosť, avšak nájsť ten správny nemusí byť vôbec jednoduché a ani lacné. Navyše ponúkané služby nemusia poskytovať to, čo práve potrebujeme. Preto je hlavným cieľom mojej diplomovej práce vytvoriť nastavovateľný webový scraper, ktorý bude na základe RSS záznamov získavať dáta z jednotlivých spravodajských portálov, a bude umožňovať používateľom nastavovať pravidlá extrakcie, či už na prvý pohľad viditeľných dát na stránke ale aj sémantických.

# Kapitola 1
# Cieľ práce

Cieľom mojej záverečnej práce je vytvoriť konfigurovateľný web scraper pre automatizované sťahovanie-extrakciu dát zo spravodajských portálov. Používatelia budú mať možnosť pridávať RSS zdroje webových portálov z ktorých chcú extrahovať dáta jednotlivých článkov. Každý zdroj budú môcť jednotlivo nastaviť podľa svojich preferencií, či už výberom z predpripravených nastavení alebo definovaním vlastných nastavení extrakcie dát. Dáta budú ukladané do databázy, ktorá bude slúžiť aj na vyhľadávanie už extrahovaných článkov a ich prípadných verzií. Tento hlavný cieľ by sa dal rozdeliť na niekoľko čiastkových cieľov.

Prvým cieľom je priblížiť spôsoby, ako sú dáta uverejňované na spravodajských portáloch a ako sa k nim dá pristúpiť, a taktiež priblížiť základné metódy extrahovania dát z webu, a jeho využitie v praxi.

Druhým cieľom je analýza existujúcich webových aplikácií, ktoré ponúkajú web scraping ako službu a súčasne aj analýza programovacích jazykov a ich knižníc, ktoré sa využívajú pre tvorbu vlastných web scraperov.

Tretím cieľom je vytvorenie funkčných a nefunkčných požiadaviek podľa výsledkov z analytickej časti a podľa pokynov vypracovania práce. Rovnako vytvorenie užívateľského prostredia a architektúry webovej aplikácie podľa definovaných požiadaviek.

Štvrtým cieľom je implementácie funkčných požiadaviek a následné testovanie webové scrapera.

# Kapitola 2
# Analýza

V tejto kapitole sa budeme venovať spravodajským portálom a prístupu ku ich dátam. Následne obecnému významu web scrapingu a aj existujúcim technológiám a ich vlastnostiam, ktoré sa pre web scraping využívajú.

Ako prvým sa budeme venovať dátam ktoré spravodajské portály poskytujú, kde ich vieme nájsť, a čo nám tieto dáta určujú.

V druhej časti sa budeme venovať obecnému vysvetleniu web scrapingu, načo sa používa, a tiež rozdielom medzi manuálnym a automatickým web scrapingom a ich vlastnostiam.

V tretej časti sa budeme venovať existujúcim riešeniam ktoré poskytujú web scraping ako službu a ich vlastnostiam.

V poslednej časti sa budeme venovať vlastnostiam programovacích jazykov a ich knižníc, ktoré sa na web scraping využívajú.

Zo získaných poznatkov následne vyhodnotíme, ktorá možnosť bude najvhodnejšia pre riešenie tohto zadania.

## 2.1 Spravodajské portály a ich dáta

Spravodajské portály sú webové stránky, ktoré poskytujú informácie o aktuálnom dianí, trendoch, názore odborníkov, informácie o nedávnych udalostiach a podobne. Spravodajské portály by sa dali rozdeliť do dvoch základných skupín[3]. Prvou skupinou sú všeobecné spravodajské portály, ktoré poskytujú informácie o rôznych odvetviach ako napríklad politika, šport, ekonomika, technológie, počasie, motorizmus a iné. Tieto portály sú konštruované tak, aby zaujali široké spektrum používateľov, niektoré slúžia ako rozšírenie ku tradičným médiám ako sú napríklad noviny. Druhou skupinou sú tematické spravodajstvá. Tieto spravodajstvá rovnako poskytujú informácie o aktuálnom dianí, ale zameriavajú sa na konkrétne odvetvia alebo témy, ako napríklad motorizmus a automobilový priemysel, veda, zdravie, šport a iné. Napríklad Deník.cz[1] je všeobecný spravodajský portál a tematický spravodajský portál je napríklad Autosport[2]. Články, ktoré sú uverejňované na týchto portáloch, informujú o aktuálnom dianí vo svete, alebo sa venujú konkrétnym témam a udalostiam. Reportéri skúmajú dostupné informácie, vedú rozhovory s odborníkmi, aby poskytli čo najrelevantnejšie informácie pre čitateľov. V týchto článkoch sú okrem textových dát, ktoré popisujú udalosti a poskytujú informácie, aj iné informačné zdroje pre čitateľa, ako napríklad obrázky, videá, odkazy na iné články s podobnou tematikou. Niektoré články obsahujú možnosti hlasovania, do ktorých sa používateľ môže zapojiť, možnosti komentovania článkov a iné[3]. Avšak v dnešnej dobe už nestačí, aby boli dáta a informácie čitateľné len

[1] https://www.denik.cz/
[2] https://www.autosport.com/

pre ľudí, ale aj pre systémy, ktoré tieto dáta môžu následne spracovávať, ukladať, vyhodnocovať ich obsah a pod. Spravodajské portály používajú rôzne metódy, ako poskytovať rozličné strojovo čitateľné dáta.

Jednou z metód na sledovanie aktualít na webových stránkach sú RSS (Really Simple Syndication) kanály. RSS je formát súborov vytvorených v XML (Extensible Markup Language), ktoré poskytujú informácie o novo pridaných súboroch alebo článkoch na danej webovej stránke[4]. V RSS kanáli sú zahrnuté informácie ako titulky, popisy, obrázky, dátumy vytvorenia a ďalšie. RSS kanály sú distribuované v reálnom čase, čo znamená, že najnovšie pridané súbory sú v ich štruktúre zoradené zostupne. RSS kanály nevyužívajú všetky webové stránky ale sú časté práve pri internetových spravodajstvách a prístup ku RSS záznamu je väčšinou možný tým, že k primárnej URL webovej stránky pridáme cestu „/rss“.

Ďalšou z metód využívanou poskytovateľmi je použitie sémantických dát. Sémantické dáta sú štruktúrované dáta, ktoré poskytujú zmysluplný a špecifický kontext informáciám na webovej stránke. Tieto dáta sú navrhnuté tak, aby boli strojovo čitateľné a umožnili lepšiu interpretáciu a porozumenie pre vyhľadávacie stroje, aplikácie a iné automatizované systémy[5]. Ich podstata pramení v tvorbe Sémantického webu.

Sémantický web je rozšírením existujúceho webu WWW (World Wide Web), ktoré umožňuje informáciám byť nie len prezentovanými, ale pridáva im význam ktorý je strojovo čitateľný[5]. Na základe týchto dát vedia systémy vytvárať znalostné grafy ktoré reprezentujú sieť entít reálneho sveta ako sú objekty, situácie, udalosti a vzťahy medzi nimi. Implementácia sémantických dát do webových stránok sa často realizuje pomocou rôznych technológií a formátov ako sú microformats, microdata, OpenGraph.

Ďalšou možnosťou sú sitemapy. Simapy alebo mapy stránok sú súbory ktoré obsahujú jednotlivé podstránky webových aplikácií. Sitemapy teda obsahujú štruktúru webových aplikácií a sú napísané v XML jazyku. Sitemapy slúžia najmä ku pohybu vyhľadávacích robotov alebo crawlerov, ktoré z jednotlivých stránok získavajú informácie[6].

Webové portály poskytujú veľké množstvo dát a udržať nad nimi prehľad môže byť kompilované a nepriehľadné. Pri tomto probléme je možné využiť metódy extrakcie dát, teda web scrapingu. Táto metóda má niekoľko výhod ako napríklad získavanie aktuálnych informácií, identifikovanie rôznych rizík, spracovanie dát pre biznis účely, alebo sa získané dáta dajú ďalej spracovávať a využiť napríklad pri strojom učení.

## 2.2 Web scraping

Web scraping je efektívny proces ktorého hlavnou úlohou je získavanie veľkého množstva dát z webových stránok. Je to metóda pri ktorej získavame neštruktúrované dáta a transformujeme ich do štruktúrovanej formy, ktorú môžeme analyzovať, uchovávať a spracovávať. Web scraping sa môže vykonávať automaticky alebo manuálne.

### 2.2.1 Manuálny web scraping

Manuálny web scraping je metóda pri ktorej používateľ manuálne prechádza stránku a získava z nej dáta ktoré potrebuje[7]. Najjednoduchší manuálny web scraping je samotné kopírovanie viditeľného obsahu na webových stránkach, avšak táto metóda je obmedzená len na dáta, ktoré sú developermi sprístupnené. Existuje však širšie rozpätie nástrojov ktoré je možné využiť.

Jedným z nástrojov ktoré sa pri manuálnom web scrapingu využívajú je Developer Tooles (DevTools)[7], teda v preklade developerské nástroje ktoré sú implementované vo väčšine moderných webových prehliadačoch ako je Google Chrome alebo Mozilla Firefox a pre informácie v tejto časti budeme využívať DevTools od Google Chrome. DevTools slúžia na získavanie detailnejších informácií o štruktúre a štylizácií HTML dokumentov. Vďaka DevTools je možné zobraziť si štruktúru HTML dokumentu, kde sú viditeľné aj používané triedy či ID jednotlivých

elementov ako aj CSS štýly, ktoré je možné bezpečne upravovať a meniť, nakoľko sa tieto zmeny dejú iba v našom okne prehliadača a po znovu načítaní stránky sa všetky naše úpravy odstránia a stránka je v pôvodnom stave.

DevTools obsahuje taktiež aj možnosť nahliadnuť a analyzovať sieťovú prevádzku danej webovej stránky, kde si vieme zobraziť jednotlivé HTTP dotazy aj s príslušnými hlavičkami a informáciami obsiahnutými v týchto hlavičkách ako aj odpovede servera na tieto dotazy, čo je užitočné pri dynamicky sa vytváraných dátach[8].

Ďalšou z možností v DevTools je Recorder. Recorder je funkcia, ktorá umožňuje používateľovi si nahrať priebeh vykonávania akcií na webovej stránke. Používateľ môže nahrať svoje interakcie s webovou stránkou, ako sú kliknutia na elementy, vypĺňovanie informácií, pridávanie vecí do košíka a iné akcie. Následne si používateľ vie tento záznam prehrať a analyzovať, čo mu umožňuje možnosť efektívnejšieho testovania a prípadného ladenia interakcií na webovej stránke[8].

Pre optimalizáciu výkonu sa v DevTools nachádza funkcia Lighthouse. Lighthouse je nástroj určený na analýzu výkonnosti, dostupnosti a kvality webovej stránky. Vykonáva audit stránky a používateľovi vráti report aj s hodnotením, ktoré určuje optimálnosť webovej stránky. Toto hodnotenie je vypočítané na základe kvantitatívneho merania výkonu. Pre tento výpočet sú využívané rozličné metriky ktoré poskytujú informácie o rôznych aspektoch výkonu[9].

DevTools rovnako obsahuje konzolu vďaka ktorej vedia developeri odchytávať a zobrazovať chyby na stránke ako aj priamo v spúšťať JavaScriptové príkazy v kontexte aktuálnej webovej stránky. DevTools umožňuje zobrazenie zdrojov ktoré sú súčasťou načítanej webovej stránky a mnoho ďalších nástrojov pre prácu s webovou stránkou[8].

Hoci sa manuálny web scraping využíva najmä ak nie je potrebné spracovávať veľké množstvo dát, jeho nevýhodou je časová náročnosť a menšia škálovateľnosť. Preto sa pre získavanie objemnejších dát využíva najmä automatický web scraping.

### 2.2.2 Automatický web scraping

Automatický web scraping je proces pri ktorom využívame softvér na získavanie dát obsiahnutých na webovej stránke[7]. Web scraper ako prvé vykoná takzvaný „fetching“ na zadanú adresu. To znamená, že softvér odošle HTTP požiadavky na server na ktorom je daná webová stránka uložená. Server následne odpovie na túto požiadavku a v odpovedi pošle HTML dáta danej stránky. Následne web scraper musí tieto dáta parsovať. Parsovanie dát je proces pri ktorom software pracuje s dátami v HTML formáte a na základe definovaných pravidiel vytvára lepšie štruktúrované dáta ktoré sú jednoduchšie na prácu. Následne už záleží na samotnom web scrapery čo z danými dátami vykoná a na čo budú dáta používané.

Pri web scrapingu sa často používa aj web crawling, nazývaný tiež Web spider. Web crawler je software ktorý slúži na získavanie zdrojov ako sú URL adresy, meta tagy a ďalšie. Web crawler si udržiava zoznamy URL adries ktoré už navštívil, aby ich opätovne nenavštevoval, a tým pádom zisťuje celkovú štruktúru webovej stránky a možnosti pohybu po tejto stránke. Web crawler a web scraper sú často využívané spoločne v komplexnejších systémoch.[10]

Web scrapery sa využívajú na sťahovanie veľkého množstva dát ktoré sú následne analyzované a opierajú sa o nich analýzy finančných trhov, sledovanie cien produktov alebo analýzu sociálnych sietí, kedy sa na základe získaných dát zo sociálnych sietí vyhodnocujú najnovšie trendy pre zlepšenie marketingových kampaní.

Ďalšou oblasťou kde sú web scrapery využívané často je strojové učenie. Web scraping sa využíva pri strojovom učení kedy je potrebné poskytnúť veľké množstvo dát pre stroj ktorý ich využije ako tréningové dáta pre strojové učenie. [7]

Pri používaní webových scraperov je potrebné brať do úvahy ich legálnosť a možnosti sťahovania z jednotlivých zdrojov. Pre zistenie možností sťahovania údajov z webovej stránky pomocou scrapera slúži súbor robots.txt ktorý je definovaný pre špecifickú webovú stránku. Robots.txt je súbor obsahujúci pravidlá pre web scrapery, ktoré určujú z ktorých častí je možné obsah

```
User-agent: *
Disallow: /__esa
Disallow: /__mesa/
Disallow: /__xesa/
Disallow: /__csup/
Disallow: /__xsla/
Disallow: /__xcusp/
Disallow: /__xesa/
Disallow: /__xsla/
Disallow: /lp
Disallow: /feedback
Disallow: /langtest

Sitemap: https://www.cloudflare.com/sitemap.xml
Sitemap: https://www.cloudflare.com/fr-fr/sitemap.xml
Sitemap: https://www.cloudflare.com/de-de/sitemap.xml
Sitemap: https://www.cloudflare.com/es-es/sitemap.xml
Sitemap: https://www.cloudflare.com/pt-br/sitemap.xml
```

**Obr. 2.1** Záznam príkazov v robots.txt. Obrázok je zverejnený na [11]

sťahovať a taktiež aj časti na ktoré web scraper nemôže byť použitý. Robots.txt je možné zobraziť pri použití základnej URL webovej stránky a následne doplniť výrazom */robots.txt*. Jednou z hlavných častí v robots.txt je označenie User-agent, ktoré definuje názov špecifického robota pre ktorého sú následne definované pravidlá. Pravidlá sú definované pomocou označení „Disallow“ alebo „Allow“.

Pre zakázanie určitej časti webovej stránky sa používa „Disallow“ a pre povolenia sa používa príkaz „Allow“. Ak sa so spojením s User-agent zadá „Disallow: /“ znamená to, že pre daného robota je všetko zakázané. Naopak, ak je pri User-agent zadané „Disallow: “, znamená to, že pre daného robota je všetko povolené, teda je možné scrapovať všetky dáta na celej webovej stránke.

Ďalšie príkazy ktoré sa môžu v robots.txt vyskytnúť sú „Crawl-delay“, ktorý definuje časové pozastavenie v ktorom robot po vykonaní akcie musí ostať a až po uplynutí tohto časového pozastavenie môže robot pokračovať[11].

## 2.3 Analýza existujúcich riešení pre web scraping

V tejto časti sa zameriame na existujúce riešenia web scraperov, ktoré sú ponúkané ako SaaS (Software as a Service) z niekoľkých dôvodov. Prvým dôvodom je ich jednoduchosť a rýchlosť nasadenia. SaaS služby ponúkajú možnosť rýchleho nasadenia a používania web scraperov bez potreby správy hardwaru či nastavovania serverov, čo používateľom umožňuje sústrediť sa priamo na proces web scrapingu. Služby sú dostupné cez webové aplikácie alebo je potrebné si ich nainštalovať do svojho zariadenia. SaaS modely súčasne umožňujú prístup ku širokej škále nástrojov a funkcií, ktoré sú spravované, aktualizované a vylepšované poskytovateľom. Ďalším výhodným aspektom niektorých týchto služieb je, že na ich používanie nie je potrebná znalosť v programovaní a ich nastavovanie a spúšťanie je jednoduché. Zameraním bude porovnať akým spôsobom pristupujú ku web scrapingu z pohľadu nastavenia užívateľa, aké funkcie majú používatelia k dispozícií, a ako by sa dali tieto aspekty uplatniť pri riešení tejto diplomovej práce.

#### 2.3.0.1 Proxy server

Proxy server je nástroj, ktorý je používaný v sieťovej komunikácií ako prostredník medzi používateľom a web serverom. Význam proxy servera spočíva v sprostredkovaní komunikácie medzi používateľom a webovým serverom. Ak používateľ odošle dotaz na webový server, tak je tento dotaz najprv spracovaný proxy serverom. Ten vyhodnotí a vyfiltruje každú požiadavku od používateľa a až následne odošle požiadavku na daný webový server v mene používateľa. To zabezpečuje určitú formu ochrany komunikácie, nakoľko ak by dáta poslané z webového servera obsahovali malvér, tak ho najprv spracuje proxy server a zabráni komunikácií, takže sa malvér nedostane ku používateľovi. Výhodou proxy servera je aj možnosť obísť geografické obmedzenia, teda umožňuje pristúp ku dátam ktoré sú bežne blokované pre určitú geografickú oblasť. Proxy servery podporujú rôzne komunikačné protokoly ako napríklad FTP(File Transfer Protocol), HTTP, HTTPS, SOCKS(Socket Secure) a iné. Existuje viacero druhov proxy serverov ktoré poskytujú špecifické funkcie s rôznymi možnosťami konfigurácie a výkonnosti ako napríklad Reverse proxy, Load Balancing proxy a iné[12].

### 2.3.1 Smartproxy

Smartproxy [3] je spoločnosť ktorá od roku 2018 ponúka služby proxy serverov, ktoré umožňujú efektívne a anonymné prehľadávanie internetu. V aktuálnom období má spoločnosť Smartproxy viac ako 65 miliónov proxy serverov[4]. Smartproxy ponúka množstvo služieb ako napríklad vopred pripravené nástroje na scraping dát alebo nástroje na manažovanie proxy serverov pre široké spektrum odvetví ako sú developeri, marketingoví analytici a pod. Ku službám ktoré Smartproxy ponúka je možné pristupovať cez webovú aplikáciu alebo využiť ich API cez ktoré si užívateľ vie napríklad spravovať svoj účet, získavať reporty o sieťovej prevádzke, vytvárať podužívateľov a iné[13].

Jednou z kľúčových služieb ktoré Smartproxy ponúka sú takzvané rotujúce proxy. Rotujúce proxy sú proxy servery umožňujúce používateľom meniť svoju IP adresu v pravidelných časových intervaloch, alebo po vykonaní určitých akcií. Výhodou tejto služby je, že udržuje anonymnú komunikáciu a tiež je vhodná pri web scrapingu nakoľko pri meniacich sa IP adresách je oveľa menšia pravdepodobnosť zablokovania scraperu.

Nastaviť web scraper je možné cez ich webovú aplikáciu alebo cez API. Webová aplikácia umožňuje nastavenie scrapera bez toho aby používateľ vedel programovať. Web scraper vie používateľ nastaviť na získavanie dát zo špecifických webových stránok, získavanie dát zo sociálnych médií, scraping e-commerce stránok, SERP scraping[13]. Smartproxy ponúka prednastavené scrapery pre každú zo spomenutých kategórií. Používateľ si iba zvolí požadovanú službu a vyplní potrebné údaje ktoré od neho táto služba vyžaduje. Tieto metódy sú však obmedzené na funkcionalitu a používateľ nemá až takú voľnosť pri nastavovaní scraper. Napríklad pri zvolení si služby Web používateľ zadá iba URL z ktorej chce dáta extrahovať a následne niekoľko parametrov ako jazyk, geografickú polohu a typ zariadenia. Po spustení scraper vráti vo formáte JSON celý obsah HTML stránky. Postup nastavenia je možný vidieť aj na obrázku 2.2.

Komplexnejšou službou ktorú Smartproxy ponúka je No-Code Scraper. Táto služba funguje ako rozšírenie do webového prehliadača. Používateľ má možnosť vyberať si prvky ktoré chce extrahovať, a to jednoduchým kliknutím na požadovaný prvok. Extrahované dáta je možné stiahnuť v JSON alebo CSV formáte, alebo je možné nastaviť plánovač, ktorý bude automaticky vykonávať extrahovanie dát v určitých intervaloch. Táto služba je avšak platená a je potrebné mať nastavené mesačné predplatné.

Webová aplikácia je priehľadná a jednoduchá na používanie. V kontexte zadania diplomovej práce má však niekoľko nedostatkov. Ak by sa aj používateľ rozhodol zaplatiť si službu No-Code Scraper, musel by vyriešiť ukladanie dát, pretože služba mu povolí si dáta stiahnuť alebo odoslať

[3] https://smartproxy.com/
[4] https://smartproxy.com/about

na email, avšak nie je možnosť priameho pripojenia databázy. Pre používanie dostupného API akejkoľvek ponúkanej služby je potrebné si zaplatiť predplatné, bez toho API nie je funkčné.

Smartproxy je používané celosvetovo s veľkým množstvom proxy serverov a veľkým počtom prístupných geografických lokácií ktoré používatelia môžu využívať. Avšak tieto služby nie sú zadarmo a pre používanie Smartproxy je potrebné realizovanie mesačných platieb, ktoré sa odvíjajú od množstva využívaných GB a taktiež od kontraktných proxy služieb. Smartproxy je vhodným nástrojom pre používateľov, ktorí majú len malé znalosti v programovaní a majú radšej predpripravený systém za ktorý si búdu platiť.

### 2.3.2 Web Scraper

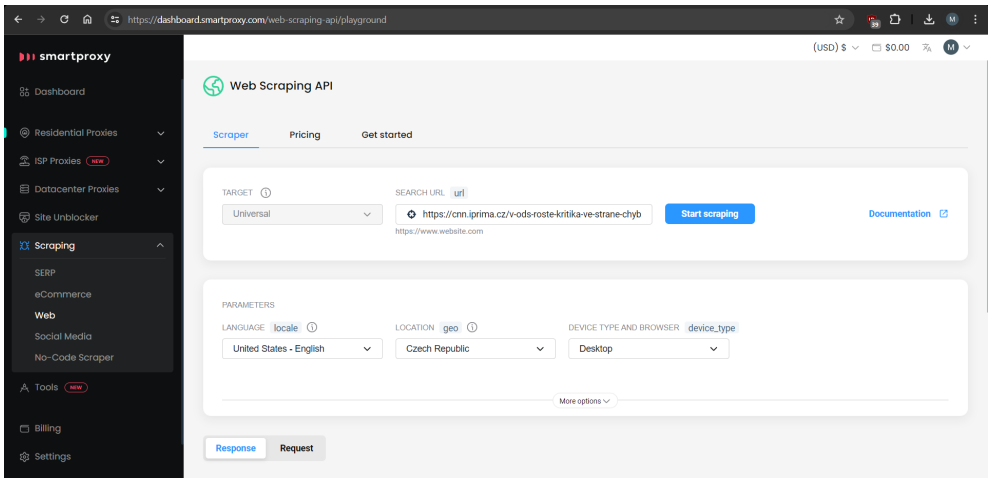
Web Scraper[5] je bezplatné rozšírenie do webového prehliadača Google Chrome od roku 2013 a v dnešnej dobe je taktiež dostupné aj pre webový prehliadač Mozilla Firefox. Po stiahnutí rozšírenia Web Scraper do prehliadača má používateľ možnosť s ním pracovať v Developer Tools. Prvým krokom pre spustenie sťahovania údajov je vytvorenie sitemapy. V sitemape používateľ definuje jej názov a url adresu alebo adresy podľa preferencie. Definovaním viacerých url adries v sitemape sa sťahovanie údajov spustí naraz pre definované zdroje. V sitemape vie používateľ jednoducho vytvárať nové selektory kliknutím na tlačidlo „Add new selector“, ktoré zobrazí nastavenia selektora. Toto nastavenie je možné vidieť aj na obrázku 2.3. V selektore je potrebné definovať jeho ID ktoré bude slúžiť ako identifikátor vo vytvorenej sitemape. Následne si používateľ musí zvoliť o aký typ dát má záujem. Tento typ definuje návratovú hodnotu v akej dáta budú sťahované[14].

Niekoľko používaných typov:

- Text : vráti text pre zvolený selektor,
- Image: vráti src atribút pre zvolený image,
- Link: vráti atribút „a href“ ako aj text pod ktorým link použitý.

Po zvolení si typu následne používateľ vyberá špecifický element stránky, ktorý chce aby scraper extrahoval. Nakoľko sa jedná o rozšírenie do prehliadača, vyberanie elementov je zabezpečené jednoduchým kliknutím na daný prvok a nie je potrebné žiadne písanie kódu[5]. Ak používateľ

[5]https://webscraper.io/




**Obr. 2.2** Nastavenie služby Web od Smartproxy, zadanie požadovanej URL a ostatných parametrov.

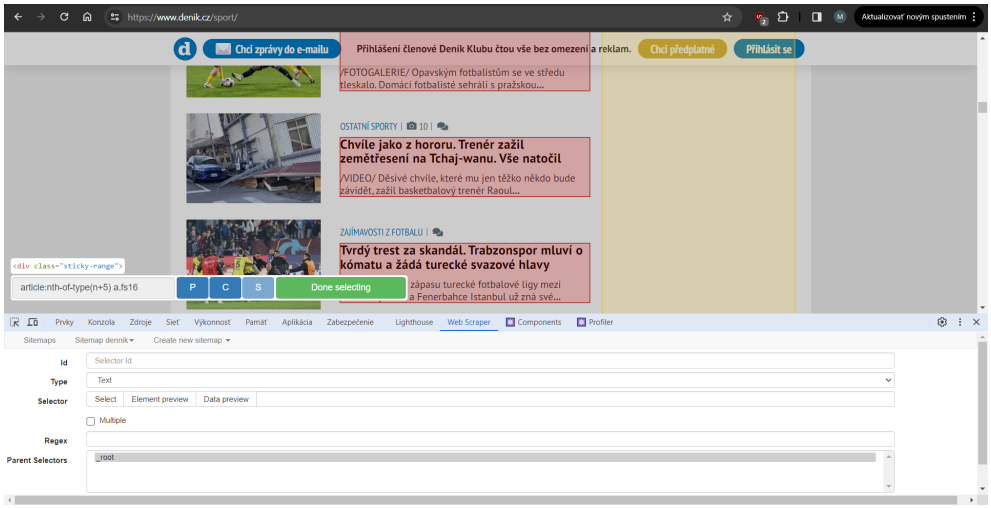
chce sťahovať viacero prvkov rovnakého typu naraz, označí si aspoň dva tieto elementy čo scraper vyhodnotí ako viacprvkový výber a označí všetky rovnaké elementy, ktoré patria do určitej klasy alebo skupiny, čo môžeme vidieť aj na obrázku 2.3. Červene označené časti textu sú označené elementy z ktorých scraper extrahuje dáta. Používateľ musí následne zaškrtnúť políčko pre výber viacerých prvkov ktoré je viditeľné na obrázku 2.3 pod názvom „Multiple“.

Pri použití typu Text môže používateľ definovať aj regulárny výraz, ktorý slúži pre získanie špecifickej podčasti textu, ktorý scraper extrahuje. Web Scraper umožňuje spájať viaceré selektory do skupín, čím sa vytvárajú komplexnejšie selektory pre extrahovanie dát. Po uložení jednotlivých nastavení je scraper pripravený na spustenie. Spustiť Web Scraper je potrebné urobiť manuálne za každým, keď chceme dané dáta extrahovať, pretože v bezplatnej verzii tohto rozšírenia nie je možné Web Scraper plánovať na opätovné sťahovanie. Avšak táto funkcia je možná cez Web Scraper Cloud.

Web Scraper Cloud je prémiová spoplatnené služba, ktorá vylepšuje funkcie samotného Web Scrapera napríklad o možnosť plánovania a automatického spúšťania, prístup a správu funkcií Web Scraper Cloudu cez API, využívanie funkcie s názvom parser, ktorá umožňuje používateľovi definovať spracovanie ukladaných dát a mnoho iného[15].Web Scraper Cloud umožňuje používateľom exportovať dáta do Dropboxu, Google Sheets, Google Drive alebo Amazon S3. Pre prácu s Web Scraper Cloud si používateľ dokupuje takzvané „Page credits“, ktoré určujú na koľko stránok môže byť scraper použitý. Extrahovanie dát z jednej URL adresy je rovné jednému kreditu, avšak nie je obmedzené množstvo dát ktoré z tejto adresy budú sťahované.

Používateľské rozhranie samotného Web Scrapera je celkom priehľadné no nie je tak dobre graficky spracované ako služba No-Code Scraper od Smartproxy. Avšak Web Scraper Cloud má jednoduché intuitívne používateľské prostredie. Pri testovaní rozšírenia Web Scraper som však narazil na pár nedostatkov. Web Scraper neposkytuje podporu na extrahovanie dát z RSS stránok a pri pokuse o ich získanie mi vrátil chybové hlásenie. Podobne nie je možné získať ani sémantické dáta a neposkytuje ani možnosť tieto dáta získať. Zároveň nie je možné pripojenie externej databázy ani v platenej verzii Web Scraper Cloud. V dostupnej dokumentácií sa nachádzali odkazy na neexistujúce video-tutoriály.

Web Scraper ako rozšírenie do prehliadača je vhodný nástroj ak chceme získať dáta jednorázovo, bez spúšťania automatického plánovača a bez toho aby sme museli písať akýkoľvek kód. Pre komplexnejšie využívanie tejto služby je potrebné využívanie Web Scraper Cloudu za ktorý však používatelia musia platiť mesačný poplatok.




**Obr. 2.3** Záznam obrazovky pri využívaní rozšírenia do prehliadača Web Scraper

### 2.3.3 Diffbot

Diffbot je spoločnosť so sídlom v Kalifornii, ktorá sa zaoberá vývojom technologických riešení v oblasti strojového učenia, web scrapingu a znalostných grafov. Spoločnosť Diffbot vytvára produkty ktoré slúžia na získavanie veľkého množstva neštruktúrovaných dát, ktoré následne pretransformujú na štruktúrované dáta. Jednou z popredných služieb ktoré ponúkajú je tvorba znalostných grafov[16].

Grafy sú vytvárané pomocou dát, ktoré spoločnosť získava z pokročilých web scrapingových technológií. Diffbot efektívne extrahuje dáta z veľkého množstva webových stránok, tie následne spracúva a vytvára alebo dopĺňa entity a vzťahy medzi nimi[16]. Aktuálny globálny znalostný graf ktorý spoločnosť Diffbot spracúva má už vyše 10 miliárd entít[17]. Používatelia majú prístup ku dátam dvoma spôsobmi. Prvý spôsob je filtrovanie v už zhromaždených dátach na základe jazyka DQL (Diffbot Query Language), ktorý slúži na vytváranie a spúšťanie dotazov nad dátami v znalostnom grafe. Druhý spôsob umožňuje nahratie vlastných dát, ktoré systém porovná s existujúcimi entitami a vyhľadá všetky dostupné informácie o zadaných entitách[17].

Ako už bolo spomenuté Diffbot využíva pokročilé technológie web scrapingu ku ktorým majú prístup aj používatelia.

Jednou zo služieb je „Extract“. Táto služba nepoužíva zoznam pravidiel, ako to býva u iných web scraperov ale používa takzvané počítačové videnie na „čítanie“ webovej stránky. Na základe toho vie obsah webovej stránky rýchlo rozkategorizovať a extrahovať zo stránky relevantné informácie. Používateľ potrebuje zadať iba URL webovej stránky z ktorej chce dáta extrahovať a služba „Extract“ sa postará o ostatné. Výhodou tejto služby je, že dokáže pracovať takmer s akoukoľvek webovou stránkou, či už sa jedná o webovú stránku obsahujúcu články, e-commerce, GitHub a mnohé iné. Po extrakcii má používateľ možnosť si dáta stiahnuť v JSON alebo CSV formáte. Výhodou tejto služby je jej rýchlosť a široký záber kategorizácie webových stránok avšak jej malou nevýhodou je, že používateľ nemá možnosť presne definovať ktoré dáta chce získať[17].

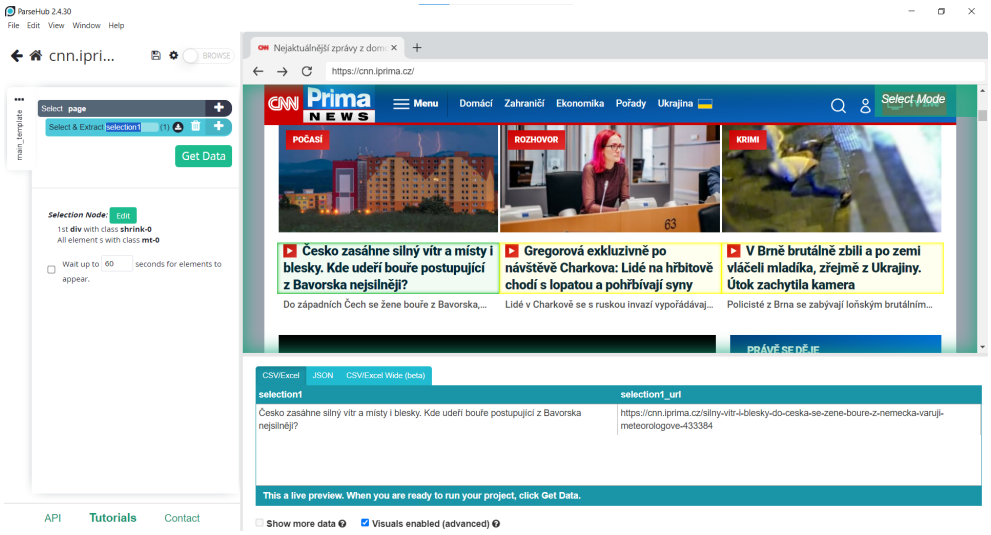
Služba „Crawlbot“ je výkonná služba ktorá slúži na web crawling. Používanie tejto služby je jednoduché a nie sú potrebné znalosti z programovania. Táto služba má hlavný parameter „seed URL“, teda hlavnú URL adresu z ktorej web crawler bude postupovať. Následne má používateľ možnosť špecifikovať počet skokov, ktorý určuje o koľko URL adries sa crawler môže vzdialiť od „seed URL“, alebo definovať parametre URL adries ktoré môže scraper navštíviť a mnohé iné parametre[17].

Využívanie služieb Diffbotu je vhodné ako aj pre jednotlivcov tak aj veľké spoločnosti. Diffbot ponúka bezplatný plán, ktorý je vhodný na osobné používanie avšak aj personalizované plány.

### 2.3.4 ParseHub

ParseHub[6] je bezplatný a voľne dostupný nástroj na web scraping, ktorý je možné nainštalovať pre rôzne operačné systémy. Pre spustenie web scrapera je v aplikácií potrebné vytvoriť nový projekt, kde sa zadá URL adresa z ktorej chce používateľ dáta extrahovať. ParseHub načíta požadovanú URL adresu a umožní definovanie elementov pre extrakciu. Pre výber elementov je potrebné iba na nich kliknúť, systém zaznamená element na ktorý používateľ klikol a uloží si daný element[18]. Pri vyberaní objektov na extrakciu systém automaticky odporúča prvky ktoré patria do rovnakých tried alebo majú spoločné vlastnosti, čo je možné vidieť aj na obrázku 2.4, kde zelenou časťou je označený element ktorý bol vybraný používateľom a žltou farbou sú označené odporúčané prvky, ktoré automaticky navrhol systém. Ak používateľ klikne na dva a viac objektov, tak sa označia všetky objekty spadajúce do danej skupiny. ParseHub ponúka okrem jednoduchých funkcií aj detailnejšie nastavenie. V týchto nastaveniach má používateľ možnosť nastavovať kliknutia na určité elementy alebo dopĺňanie informácií do textových polí,

[6] https://www.parsehub.com/




**Obr. 2.4** Používanie ParseHub na webovej stránke "https://cnn.iprima.cz/", zvolený prvok je označený zelenou farbou a odporúčané prvky sú označené žltou farbou.

čo umožňuje aj extrahovanie dát z webových stránok na ktorých je potrebná autorizácia. ParseHub umožňuje používateľom využívať API, cez ktoré majú možnosť scraper nastavovať. Po extrahovaní si používateľ môže dáta uložiť vo formáte JSON alebo CSV.

Avšak verzia ktorá je dostupná zdarma je obmedzená vo funkciách ktoré ParseHub ponúka. V platenej verzii tohto programu má používateľ možnosť pristupovať ku viacerým funkciám ako napríklad pre vytvorený projekt je možné nastaviť plánovač, pripojiť DropBox úložisko alebo využiť rotáciu IP adries pre zlepšenie bezpečnosti[19].

ParseHub má priehľadné používateľské rozhranie vďaka ktorému sa jednoducho vyberajú prvky a taktiež tvoria dynamické interakcie so stránkou. Pri testovaní tejto aplikácie som podobne ako pri službe Web Scraper narazil na nedokonalosť v podobe nespracovania RSS dát. Avšak v kontexte tejto diplomovej práce by sa tento fakt dal čiastočne nahradiť práve schopnosťou tejto služby tvoriť interakcie s webovou stránkou, nakoľko niektoré webové portály obsahujú URL, ktorá zobrazuje aktuálne pridané články. Napríklad Prima CNN využíva URL „https://cnn.iprima.cz/prave-se-deje"pre zobrazenie najaktuálnejších článkov. Na túto URL je možné nasadiť službu ParseHub ktorá bude postupne extrahovať zvolené dáta. Takýmto spôsobom síce používateľ nevie získať dáta obsiahnuté v RSS ale má možnosť sledovať najnovšie články. Extrahované dáta avšak nie je možné ukladať priamo do databázy, je možné si ich stiahnuť alebo v platenej verzii ukladať do Amazon S3 alebo DropBoxu. Zaujímavým prvkom ktorý nepoužíva iná z analyzovaných služieb je zobrazenie zvolených dát na extrakciu v rôznych formátoch, presnejšie vo formáte CSV a JSON.

ParseHub je vhodnou aplikáciou pre používateľov, ktorý nemajú znalostí v programovaní. Jeho grafické prostredie je intuitívne a nie je zložité sa v ňom orientovať. V platenej verzii je možné plánovať spúšťanie scraperov či ukladanie dát do Dropboxu alebo Amazon S3. Nevýhodou je potreba nainštalovania aplikácie priamo do zariadenia, čo túto aplikáciu obmedzuje len na desktopové využívanie.

### 2.3.5 Apify

Spoločnosť Apify[7] vznikla v roku 2015 a v dnešnej dobe sídli v Prahe v Českej republike. Apify je cloudová platforma na ktorej si používatelia vedia zostaviť, vyskúšať a publikovať nástroje na web scraping, extrakciu dát a iné[7]. Práca na platforme je sprostredkovaná cez takzvaných Actors. Actors sú cloudové programy ktoré pracujú na Apify platforme. Tieto programy môžu vykonávať od jednoduchých vyplňovaní formulárov po web scraping veľkých dát. Každý program je upravený na špecifickú úlohu a používateľ k nim môže pristupovať cez samotnú platformu. Niektoré Actors programy sú pre užívateľov zdarma, za niektoré je potrebné platiť. Používatelia majú možnosť vytvoriť si aj vlastných Actors, ktorých môžu následne využívať na platforme alebo lokálne[20].

Apify ponúka dve možnosti ako vytvoriť Actor program na mieru. Prvou možnosťou je tvorba programu lokálne. Pre túto metódu je potrebné si nainštalovať Actor CLI čo je možné vykonať v príkazovom riadku pomocou príkazu *npm -g install apify-cli*. Následne má používateľ možnosť si vytvoriť Actor cez príkaz *apify create*. Ten spustí inštalačný proces v rámci ktorého musí používateľ definovať názov pre Actor, vybrať si programovací jazyk ktorý môže byť JavaScript, TypeScript alebo Python a následne si vyberá šablónu v ktorej chce daného Actor vyvíjať. Po naprogramovaní Actora je možné ho lokálne otestovať a používateľ má taktiež možnosť ho nahrať na Apify Cloud. Druhou možnosťou je tvorba Actors na samotnej platforme. Používateľ má možnosť vytvoriť si vlastných Actos cez webové IDE ktoré ponúka Apify. Ako prvé si používateľ musí vybrať z predpripravených šablón ktoré určujú v akom programovacom jazyku bude daný program vyvíjaný[20]. Používateľ programuje v samotnom webové IDE a taktiež vie tento program aj priamo otestovať. Platforma Apify poskytuje možnosť naplánovania automatického spúšťania web scraperov.

Služby od Apify na rozdiel od ostatných analyzovaných neposkytujú výber elementov na extrakciu pomocou kliknutia na daný element. Záleží na tom akého Actora si používateľ vyberie, v niektorých stačí definovať URL, v iných je potrebné definovať časť extrakcie pomocou kódu. Toto nastavovanie avšak neobsahuje náhľad na nastavenie extrahovaných dát v reálnom čase, takže používateľ nevidí aké dáta budú extrahované.

Spoločnosť Apify taktiež vyvinula JavaScript knižnicu Crawlee. Táto knižnica je určená pre tvorbu webových scraperov a crawlerov a je určená pre prácu s JavaScriptom alebo TypeScriptom. Knižnica Crawlee využíva funkcie od iných knižníc ako Cheerio Puppeteer. Jednou z výhod knižnice Crawlee je jej bezplatné používanie[21].

Apify je vhodným nástrojom pre veľké firmy ako aj jednotlivcov, nakoľko ponúka široké množstvo už pripravených programov pre web scraping a taktiež aj možnosti tvorby vlastného web scrapera podľa individuálnych preferencií. Pre používanie Apify a najmä knižnice Crawlee či Puppeteer musí mať používateľ aspoň základné znalosti programovacieho jazyka JavaScript alebo TypeScript.

### 2.3.6 Zhrnutie

Analyzovali sme niekoľko SaaS služieb pre vykonávanie web scrapingu. Všetky analyzované služby okrem Apify pristupujú ku extrakcii dát pomocou jednoduchého klikania na požadované elementy, čo je užívateľsky jednoduchšie ako písanie kódu ale obmedzuje to rozsah extrahovaných dát iba na tie dáta, ktoré je možné na stránke vidieť a nie možné získať sémantické dáta ani pracovať s RSS záznamom. V službe Apify sa pristupovalo ku extrakcii dát rôzne, podľa toho akého Actora si používateľ zvolil. V niektorých stačí definovať iba URL, v iných je potrené aby používateľ došpecifikoval určité parametre vo forme kódu. Actors od Apify ako jediný z analyzovaných služieb neposkytujú náhľad na nastavenia extrakcie v zmysle zobrazenia dát ktoré ešte len budú extrahované, ostatné služby obsahujú jasné a priehľadné zobrazenie dát ktoré budú ex-

[7] https://apify.com/

trahované a služba ParseHub ich zobrazuje vo viacerých formátoch ako JSON a CSV. Ani jedna z analyzovaných služieb neponúka priame ukladanie dát do databázy, dáta je možné stiahnuť ako JSON alebo CSV súbor, v platených verziách služby ako ParseHub a Web Scraper Cloud umožňujú odosielanie dát do Dropboxu alebo Amazon S3.

## 2.4 Analýza dostupných technológií pre tvorbu vlastného web scrapera

V tejto časti budeme analyzovať najpoužívanejšie programovacie jazyky a ich knižnice, rozšírenia ktoré sa využívajú pri tvorbe webových scraperov. Analyzovať budeme ich vlastnosti a funkcie ktoré sa pre web scraping využívajú a taktiež vyhodnotíme ktoré z nich sú najvhodnejšie pre vypracovanie tohto zadania.

### 2.4.1 Python

Python[8] je objektovo orientovaný programovací jazyk s dynamickou sémantikou. Python sa vyznačuje vysokoúrovňovými dátovými štruktúrami, ktoré v kombinácií s dynamickým zapisovaním vytvárajú prostredie, ktoré je často využívané pre rýchly vývoj aplikácií. Jeho jednoduchý syntax sa vyznačuje prehľadnosťou, čo znižuje možné náklady na údržbu kódu. Python podporuje širokú škálu knižníc, modulov a je vytvorených mnoho frameworkov ktoré boli postavené na tomto jazyku. Python[8] je v dnešnej dobe veľmi obľúbený a používa sa v rôznych odvetviach ako napríklad tvorba webu, strojové učenie, dátová analýza a iné . Python sa rovnako využíva pre tvorbu web scraperov a crawlerov, nakoľko existuje mnoho knižníc a frameworkov ktoré sú špecificky navrhnuté pre tieto účely. Najčastejšie využívané Python nástroje sú[22]:

- Beautiful Soap,
- Scrapy,
- Playwright a iné.

#### 2.4.1.1 BeautifulSoup

BeautifulSoup je Python knižnica ktorá slúži na extrahovanie dát z HTML a XML súborov. Automaticky analyzuje dáta a vytvorí pre dané dáta stromovú štruktúru pre rýchlejšiu a jednoduchšiu prácu s dátami[23] [22]. Taktiež jej syntax a samotné používanie je celkom jednoduché a intuitívne. Pre extrahovanie a analyzovanie dát je potrebné jednoducho zadať požadovaný súbor alebo využiť Python knižnicu *requests* pre získanie HTML dát z webovej stránky. Knižnica nepodporuje spracovanie JavaScriptu na webe, čo jej znemožňuje prácu s dynamicky sa tvoreným obsahom. V nasledujúcom kóde je najprv načítaný obsah z webovej stránky *example.com* pomocou knižnice *requests* a následne sú tieto dáta odovzdané v textovej podobe funkciám v knižnici BeautifulSoup ktoré tieto dáta spracujú, a následne sú spracované dáta vypísané v konzole.

**Výpis kódu 2.1** Využitie knižnice BeautifulSoup pre analýzu HTML dát

```
from bs4 import BeautifulSoup
import requests

url = "https://www.example.com"
req = requests.get(url)
page = BeautifulSoup(req.text, "html.parser")
print(page)
```

[8] https://www.python.org/doc/essays/blurb/

BeautifulSoup umožňuje jednoduchý prístup ku spracovaným dátam. Pre získanie napríklad obsahu v HTML tagu <*title*> je potrebné zadať iba *print(page.title)*. Knižnica BeautifulSoup je často využívaná kvôli jej jednoduchej syntaxe a taktiež rozsiahlej dokumentácií.

#### 2.4.1.2 Scrapy

Scrapy je framework napísaný v Pythone, ktorý je navrhnutý na web scraping a web crawling. Na extrahovanie HTML alebo XML dát Scrapy využíva XPath alebo CSS selektory. Scrapy taktiež obsahuje interaktívnu konzolu ktorá slúži na testovanie CSS a XPath výrazov alebo telnet konzolu pre debugovanie a spravovanie scrapera. Prvotne je potrebné si framework Scrapy nainštalovať. Následne si používateľ vie vytvoriť projekt pomocou príkazu „scrapy startproject newproject“. Scrapy automaticky vytvorí súborovú štruktúru[24]:

```
├ scrapy.cfg
└newproject
  ├ __init__.py
  ├ items.py
  ├ middlewares.py
  ├ pipelines.py
  ├ settings.py
  └spiders
    └__init__.py
```

Jednotlivé súbory sú určené pre špecifické funkcie[25].

- **settings.py**: Je miesto kde sú definované nastavenia scraper,
- **items.py**: Je model pre extrahované údaje,
- **pipelines.py**: Miesto kde sú definované Item Pipline, teda triedy ktoré spracujú extrahované dáta,
- **middlewares.py**: Slúži na úpravu toho ako bude požiadavka vytvorená,
- **scrapy.cfg**: Je konfiguračný súbor.

Nasledujúci kód je možné spustiť príkazom *scrapy crawl quotes*.

**Výpis kódu 2.2** Práca s frameworkom Scrapy na extrahovanie dát z definovanej url adresy.

```
import scrapy
class NewSpider(scrapy.Spider):
    name = 'newspider'
    start_urls = ['http://example.com/']

    def parse(self, response):
        pass
```

Popis kódu:

- **name**: Je to atribút ktorý identifikuje scraper. Je potrebné aby používateľ definoval jedinečný *name* atribút v rámci projektu,
- **start_urls**: Zoznam URL adries z ktorých scraper začne extrahovať dáta,
- **parse**: Je funkcia ktorá sa využíva na spracovanie odpovedí zo zadanej URL.

Po spustení je môže používateľ využiť konzolu ktorú Scrapy ponúka na testovanie selektorov. Napríklad ak používateľ použije príkaz *response.css("title::text").get()* v konzole tak definoval:

- **responce.css**: Definuje že sa jedná o CSS selektor a *responce* sa odkazuje na objekt ktorý obsahuje dáta z poslednej načítanej stránky,
- **title::text**: Určuje že používateľ chce získať iba textovú hodnotu ktorá je vrámci HTML tagu *title*,
- **get()**: Definuje návratovú hodnotu na prvý záznam ktorý vyhovuje zadanému selektoru.

Scrapy je framework pre web scraping, vďaka čomu obsahuje viac funkcionalít a je robustnejší ako knižnica BeautifSoup, čo mu umožňuje pracovať aj s dynamicky sa tvoriacimi dátami, avšak Scrapy je možné využiť v spolupráci s BeautifulSoup, kedy je obsah webovej stránky získaný pomocou Scrapy a vyhľadávanie v jej štruktúre je pomocou BeautifulSoup[24].

#### 2.4.1.3 Playwright

Playwright[9] je framework, ktorý slúži na testovanie a automatizáciu. Bol vyvinutý firmou Microsoft. Používateľ má možnosť definovať interakcie s webovým prehliadačom, čo umožňuje prístup aj ku dynamicky sa generovaným webovým stránkam[22]. Playwright sa vyznačuje tým, že je možné ho nastaviť pre používanie na rôznych prehliadačoch ako napríklad Google Chrome, Mozilla Fitefox, Safari. Ďalšou výhodou tohto frameworku je možnosť pracovať s ním v rôznych programovacích jazykoch. Playwright podporuje programovanie jazyky ako Python, JavaScript, TypeScript, .NET, Java. V kontexte web scrapingu Playwright využíva extrakciu prvkov na základe CSS alebo XPath selektorov.

**Výpis kódu 2.3** definuje základné využitie frameworku Playwright v jazyku Python pre načítanie a extrahovanie dát zo stránky.

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    page = browser.new_page()
    page.goto("https://playwright.dev")
    page.screenshot(path="example.png")
    element = page.query_selector(".hero__title")
    text_content = element.text_content()
    print("Text prvku:", text_content)
    browser.close()
```

Popis kódu:

- **browser = p.chromium.launch()**: Definuje spustenie novej inštancie prehliadača chronium, možné je však definovať aj iné prehliadače,
- **page = browser.new_page()**: Vytvára novú stránku a priradzuje ju do premennej page,
- **page.goto(ürl")**: Navigácia na požadovanú url,
- **page.screenshot(path=ëxample.png")**: Vytvorenie snímku obrazovky z danej url a uloženie do súboru *example.png*,
- **element = page.query_selector("class/id/tag")**: Vyhľadávanie na základe CSS selektora. Playwright využíva funkciu „query_selector“ v ktorej sa definuje XPath ale CSS atribút prvku ktorý chce používateľ extrahovať,
- **text_content = element.text_content()**: Vráti textový obsah z extrahovaného prvku,

[9] https://playwright.dev/

- **browser.close()**: Zatvorenie prehliadača.

Medzi výhody využívania Playwright jednoznačne patrí možnosť konfigurácie v rôznych programovacích jazykoch, schopnosť interakcie s webovou stránkou a možnosť extrahovať dáta aj zo stránok s potrebnou autentifikáciou, tvorba snímky či už samotného elementu alebo celej stránky. Avšak jednou z jeho nevýhod je nepodporovanie analýzy(parsovania) údajov.

### 2.4.2 Node.js

Node.js je open-source, multiplatformové JavaScriptové prostredie pre beh JavaScriptu mimo prehliadača, a patrí medzi najpoužívanejšie technológie pri tvorbe webových stránok[26]. Toto prostredie je napísané v JavaScriptovom engine Chrome V8 od spoločnosti Google ktorý je tiež používaný v prehliadači Google Chrome. Node.js slúži najmä na tvorbu serverovej časti webovej stránky v jazyku JavaScript[27]. Aplikácia napísaná v Node.js pracuje v rámci jedného procesu, bez potreby vytvárať pri každej požiadavke nové vlákno. Základnou vlastnosťou Node.js je že sa vstupno-výstupné požiadavky vykonávajú asynchrónne, teda jednotlivé Node.js procesy sa neblokujú medzi sebou čo zrýchľuje vykonávanie požiadaviek. Vďaka tomu dokáže Node.js pracovať s veľkým množstvom požiadaviek. Web scraping v Node.js vie používateľ vykonávať vďaka mnohým knižniciam ktoré Node.js ponúka[27].

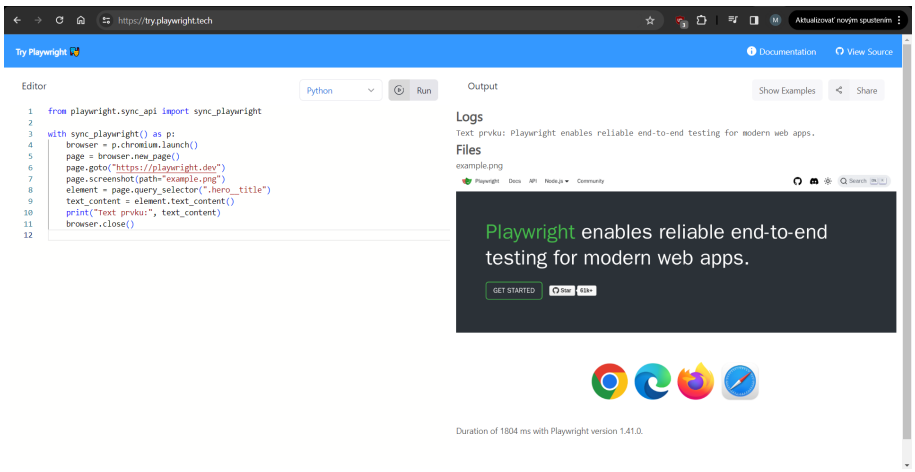
Medzi najznámejšie patrí[28]:

- Cheerio,
- Puppeteer,
- Playwright,
- Crawlee a iné.

#### 2.4.2.1 Cheerio

Cheerio[10] je JavaScriptová knižnica na analýzu HTML a XML dát. Cheerio sa vyznačuje tým, že dokáže pracovať v prostredí prehliadača ako aj serveru, jednoducho pracuje s DOM a jed-

[10] https://cheerio.js.org/




**Obr. 2.5** Výsledok kódu 1.3 prostredníctvom online Playwright prostredia na url adrese "https://try.playwright.tech/"

noduchým API. Cheerio samé o sebe nevykonáva HTTP požiadavky, preto je potrebné využiť pre získavanie obsahu z HTML stránok inú Node.js knižnicu ktorá tieto požiadavky vykonávať dokáže, ako napríklad knižnica axios, ktorá vykoná požiadavky na zadané HTML stránky. Cheerio používa podobný syntax ako jQuery na manipulovanie s HTML dátami. Knižnica Cheerio používa CSS selektory na definovanie prvkov, ktoré sa majú extrahovať. V knižnici je možné používať rôzne funkcie, ktoré pomáhajú s prechádzaním v DOM štruktúre, ako napríklad .find(), .parrent() a pod. Pri CSS selektoroch je možné definovať jednotlivé prvky HTML stránky, ako napríklad <h1> alebo , definovať triedy, identifikátory, vzťahy v štruktúre, atribúty, počet, poradie a mnoho iného. Cheerio[10] sa vyznačuje aj rýchlym spracovaním požiadaviek a vyhľadávaním v dokumente.

**Výpis kódu 2.4** Program získa obsah HTML stránky a následne spracuje a získa špecifikované údaje.

```
const axios = require('axios');
const cheerio = require('cheerio');

const url = 'http://example.com';

axios.get(url)
  .then(response => {
    const $ = cheerio.load(response.data);
    const title = $('h1').text();
    console.log('Nadpis:', title);
    const findContent = $('div').find('p');
    findContent.each((index, element) => {
      console.log($(element).text());
    });
  })
  .catch(error => {
    console.log('Error responce', error);
  });
```

Popis kódu:

- **cheerio.load(response.data)**: Načítanie HTML dát ktoré program získal ako odpoveď na axios požiadavku,
- **$('h1').text()**: Definovaný selektor určuje že program sa snaží získať textovú hodnotu ktorá je obsiahnutá v HTML pod tagom <h1>,
- **$('div').find('p')**: Definovaný selektor sa snaží pristúpiť ku dátam v tagoch <p> ktoré sú súčasťou prvého  elementu,
- **findContent.each((index, element) =>** : Funkcia vypíše jednotlivo všetky výsledky ktoré selektor findContent získal,

Obmedzením použitia Cheerio je absencia načítania externých zdrojov, ktoré sa na stránkach môžu vyskytovať ani nespúšťa JavaScript na stránkach. Avšak jeho syntax je jednoduchý na používanie a výsledky vráti rýchlo.

#### 2.4.2.2 Puppeteer

Puppeteer[11] je Node.js knižnica, ktorá ponúka API pre kontrolovanie prehliadača Google Chrome. Bol vyvinutý firmou Google. Možnosti[11] využitia tohto API sú podobné interakcii používateľa so samotným prehliadačom ako napríklad vytvoriť záznam obrazovky, vytvárať automatické

[11] https://pptr.dev/

testy, pracovať s dynamicky sa tvorenými web stránkami ale aj mnohé iné. Puppeteer využíva querySelektory v ktorý používateľ definuje ktoré dáta chce extrahovať.

**Výpis kódu 2.5** Využitie knižnice Puppeteer na získanie a extrahovanie dát z webovej stránky

```
const puppeteer = require('puppeteer');
(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('http://example.com');
  const title = await page.evaluate(() => {
    const textdata = document.querySelector('h1').innerText;
    return textdata
  });
  console.log('Nadpis h1:', title);
  await browser.close();
})();
```

- **await puppeteer.launch()**: Asynchrónna funkcia ktorá spúšťa novú inštanciu prehliadača Chrome,
- **await browser.newPage()**: Asynchrónna funkcia ktorá vytvorí novú stránku v prehliadači, teda ako keby si používateľ manuálne otvoril novú kartu v prehliadači,
- **await page.gote(url)**: Funkcia naviguje na zadanú stránku,
- **await page.evaluate(() => ... )**: Táto funkcia vykonáva hodnotenie JavaScript kódu v kontexte stránky,
- **document.querySelector('h1').innerText**: Táto funkcia vyhľadáva prvý element h1 na stránke a získava jeho textový obsah pomocou vlastnosti innerText,
- **await browser.close()**: Zatvorenie prehliadača po skončení operácií.

Puppeteer je vhodný pri práci s dynamicky sa tvoriacimi web stránkami, avšak oproti knižnici Cheerio je pomalší na spracovanie dát a syntax je komplikovanejší.

#### 2.4.2.3 Crawlee

Crawlee je JavaScriptová knižnica ktorá bola vyvinutá spoločnosťou Apify, ktorá slúži na vytváranie a správu web scraperov a crawlerov na extrakciu dát z webových stránok. Crawlee využíva vlastnosti iných web scrapingových knižníc v node.js. Crawlee podporuje takzvané „bezhlavé“ vyhľadávanie, teda vyhľadávanie cez webový prehliadač ktorý neobsahuje používateľské rozhranie. Crawlee využíva funkcie z knižníc na „bezhlavé“ vyhľadávanie ako sú Puppeteer a Playwright v kombinácií s vlastnými antiblokovacími funkciami. Pre extrakciu dát knižnica Crawlee využíva funkcie z iných knižníc ako sú Cheerio alebo JSDOM. Výhodou knižnice Crawlee je, že vytvorené scrapery má používateľ možnosť nahrať na Apify Cloud a sprístupniť ich širšej verejnosti[21].

**Výpis kódu 2.6** Jednoduchý program ktorý využíva knižnicu Crawlee pre získanie dát z webovej adresy

```
import { CheerioCrawler, Dataset } from 'crawlee';
const crawler = new CheerioCrawler({
    async requestHandler({ $, log }) {
        const title = $('h1').text();
        log.info(title);
```

```
        await Dataset.pushData({title});
    },
});
await crawler.run(['https://www.example.com/']);
```

Popis kódu:

- **new CheerioCrawler**: Vytvorenie novej inštancie CheerioCrawler,
- **async requestHandler( $, log )** : V tele funkcie requestHandler sa vykonáva logika scrapovania. Parameter '$' definuje vlastnosti knižnice cheerio a teda spracovanie HTML dát a parameter „log" umožňuje logovanie správ počas crawlingu. Funkcia requestHandler sa zavolá zakaždým keď crawler prejde na novú url,
- **Const title = $('h1').text()**: Využitie vlastností cheeria pre definovanie extrahovaného prvku,
- **Log.info(title)**: Vypíše do konzoly získaný element „title",
- **await Dataset.pushData(title)**: Uloženie extrahovaných dát do Datasetu.

Crawlee je knižnica ktorá ponúka mnoho prostriedkov na web scraping, nakoľko využíva viaceré funkcionality iných web scrapingových knižníc ako sú cheerio alebo puppeteer, ktoré dopĺňa o vlastné funkcie.

### 2.4.3 Ruby

Ruby[12] je open source, objektovo orientovaný programovací jazyk, ktorý bol vyvinutý v 90rokoch v Japonsku. Ruby má podobné vlastnosti ako iné programovacie jazyky, ako napríklad Python, Perl, Smalltalk. Kľúčovou vlastnosťou jazyka Ruby je, že všetko je považované za objekt, teda každej časti kódu je možné priradiť vlastné akcie a vlastnosti. Ruby[12] je využívaný najmä pri tvorbe webových stránok vdaka jeho frameworku Ruby on Rails. Avšak Ruby je využívaný aj na tvorbu webových scraperov a existuje mnoho knižníc ktoré boli pre tieto účely vytvorené ako napríklad Nokogiri alebo Mechnize.

#### 2.4.3.1 Nokogiri

Nokogiri je knižnica navrhnutá na spracovávanie HTML a XML dokumentov. Ponúka jednoduché API na prácu s týmito dátami. Vyhľadávanie v dátach je zabezpečené cez CSS alebo XPath selektory. Avšak samotná knižnica Nokogiri nedokáže vytvárať http dotazy, preto je často kombinovaná s inými knižnicami ktoré tieto dotazy vykonajú, ako napríklad knižnica open-uri [29].

**Výpis kódu 2.7** Jednoduchý Ruby program ktorý využíva knižnice nokogiri, httparty, csv na získanie a zápis dát z webovej stránky do csv súboru

```
require 'nokogiri'
require 'httparty'
require 'csv'
html = HTTParty.get("https://www.example.com/")
doc = Nokogiri::HTML(html.body)
title = doc.css('h1').text
data = {
  "title": title
```

[12] https://www.ruby-lang.org/en/about/

```
}
CSV.open('nadpisy.csv', 'w') do |csv|
  data.each do |key, value|
    csv << [key, value]
  end
end
```

Popis ku kódu:

- **HTTParty.get()**: Dotaz na zadanú url pre získanie obsahu HTML stránky,
- **Nokogiri::HTML(html.body)**: Vytvorenie nokogiri objektu z tela dát z dotazu pre ďalšie spracovanie,
- **doc.css('h1')**: vyhľadávanie v nokogiri objekte pomocou CSS selektora s definovaným elementom,
- **data = "title": title**: Vytvorenie štruktúry s názvom data ktorá obsahuje kľúč na názvom „title“ a k nemu prislúchajúcu hodnotu získanú z HTML,
- **CSV.open('nadpisy.csv', 'w') do |csv|**: Otvorenie súboru „nadpisy.csv“ na zapisovanie a vytvára blok kódu s premennou „csv“ ktorá reprezentuje otvorený súbor CSV,
- **data.each do |key, value|**: Iterácia cez štruktúru „data“ pri ktorej získava kľúč a jeho hodnotu,
- **csv << [key, value]** - Zápis do CSV súboru.

Nokogiri je prehľadná knižnica ktorá ponúka jednoduchú prácu s HTML dátami. Aj keď Nokogiri poskytuje robustné a priehľadné rozhranie na prácu s HTML dátami, jeho hlavným obmedzením je obmedzená podpora dynamického upravovania štruktúr dokumentu v porovnaní s niektorými inými nástrojmi.

#### 2.4.3.2 Mechanize

Mechanize je Ruby knižnica ktorá slúži na konfigurovanie automatickej interakcie s webovým prehliadačom. Medzi jej základné vlastnosti spadá získavanie, ukladanie, odosielanie cookies súborov, vypĺňovať formuláre, nasledovať linky a iné[30].

**Výpis kódu 2.8** Jednoduchý Ruby program ktorý využíva knižnicu mechanize na získanie dát z webovej stránky

```
require 'mechanize'
agent = Mechanize.new
example = agent.get('https://www.example.com')
titles = example.search('h1')
titles.each do |title|
  puts "Title h1: #{title.text}"
end
odkazy = example.links
odkazy.each do |odkaz|
  puts odkaz.href
end
```

Popis príslušného kódu:

- **Mechanize.new**: Vytvorenie novej inštancie objektu Mechanize, ktorý reprezentuje agenta(mechanizmus) ktorý slúži na interakciu so stránkou,

- **agent.get(url)**: Načítanie HTML dát zo stránky ktoré sú uložené do príslušnej premennej,
- **example.search('h1')**: Funkcia search získa všetky záznamy z uvedeného selektora, teda v tomto prípade všetky prvky <h1> a uloží ich do príslušnej premennej,
- **titles.each do |title|**: Funkcia prejde cez všetky získané dáta v premennej titles a následne ich vypíše pomocou funkcie: *puts "Title h1: #{title.text}"*, kde *.text* určuje že sa získa textová hodnota,
- **example.links** - Funkcia *links* získa všetky linky(<a>) ktoré sú na zadanej webovej stránke,
- **puts odkaz.href** - Funkcia vypíše všetky url adresy zo získaných linkov.

Medzi výhody knižnice mechanize určite patrí jej jednoduché použitie a taktiež aj podpora pre prácu s cookies. Knižnica však nemá až takú podporu JavaScriptu, takže pri niektorých stránkach, ktoré sa opierajú o veľkú časť ich funkcionality o JavaScript, táto knižnica nemusí byť vhodná.

### 2.4.4 Zhrnutie analyzovaných technológií

V tejto časti sme sa pozreli na 3 najpoužívanejšie programovacie jazyky, ktoré sú využívané pri tvorbe web scraperov. Pre každý z týchto programovacích jazykov sme analyzovali niektoré z najpoužívanejších knižníc, ktoré sa pri tvorbe web scraperov používajú. Každá z týchto knižníc pristupuje ku riešeniu automatizovaných web scraperov v niektorých oblastiach rozlične, no v jadre sú si veľmi podobné.

Pre programovací jazyk Python bola knižnica BeautifulSoup prispôsobená na rýchle extrahovanie dát s používateľky jednoduchým syntaxom, avšak medzi jej slabé stránky patrí fakt, že knižnica nie je prispôsobená na dynamicky sa JavaScriptom tvorené dáta. Scrapy je samotný framework pre prácu s web scrapermi čím vytvára celkovú súborovú štruktúru a je robustnejším nástrojom pre konfiguráciu web scraperov. Playwright je knižnica ktorá je prispôsobená nie len na Python ale aj na JavaScript alebo TypeScript. Táto knižnica je viac prispôsobená na testovanie interakcií s webovými stránkami a ich analýzu, čo sa však môže podpísať na jej efektivite pri spracovávaní veľkého množstva dát.

V Node.js sme analyzovali 3 knižnice. Prvou bola knižnica Cheerio. Táto knižnica poskytuje jednoduchý syntax a je jednoduchá na používanie a taktiež je efektívna a rýchla pri vyhodnocovaní dát. Nevýhodou tejto knižnice je to, že sama knižnica neponúka možnosti vytvárania dotazov a teda pre získanie HTML obsahu stránky je potrebné použiť inú knižnicu ktorá tieto dáta získa ako napríklad knižnica axios. Puppeteer je komplexnejšou knižnicou ktorá taktiež ponúka podporu pre viaceré programovanie jazyky ako je Python, JavaScript, TypeScript. Je vhodnejšia pre komplexnejšie testovanie webových stránok a interakcie na nich. Knižnica Crawlee je podľa môjho názoru zaujímavou knižnicou, pretože využíva funkcie zo spomenutých knižníc Cheerio a Puppeteer nad ktoré aplikuje svoje funkcie pre jednoduchšiu správu.

Pre programovací jazyk Ruby sme spomenuli dve knižnice. Knižnica Nokogiri sa vyznačuje rýchlym spracovaním a jednoduchým syntaxom, avšak podobne ako knižnica Cheerio neposkytuje priamo vytváranie dotazov na webové stránky a je potrebné využiť iné knižnice pre získanie HTML dát. To tiež spôsobuje, že nie je prispôsobená na JavaScriptom dynamicky sa vyvíjajúce stránky. Druhou knižnicou bola knižnica Mechanize, ktorá je viac prispôsobená na vytváranie interakcií s prehliadačom, avšak ani táto knižnica nie je prispôsobená na vykonávanie JavaScriptu na webových stránkach.

Po analýze dostupných jazykov a ich knižníc som sa rozhodol pre využívanie jazyka Node.js s knižnicami Cheerio a Puppeteer. Node.js je široko používaný pri tvorbe backend časti webových aplikácií v jazyku JavaScript. Využívanie Node.js zohľadňuje potreby a požiadavky na tvorbu tejto diplomovej práce ktorá má byť vo forme webovej aplikácie. Kombináciu knižníc Cheerio a Puppeteer som si zvolil najmä kvôli jednoduchosti syntaxu knižnice Cheerio, nakoľko nastavenia

extrakcie dát bude vykonávať používateľ, a tiež pre jej rýchlosť, za akú táto knižnica dokáže dáta spracovať. Knižnica Puppeteer doplňuje knižnicu Cheerio v možnosti extrahovania dát aj z dynamicky sa tvoriacich web stránok, pri zachovaní jednoduchosti syntaxu, ktorým bude používateľ scraper nastavovať.

# Kapitola 3
# Návrh webovej aplikácie

V tejto kapitole sa v prvom rade zameriame na určenie si funkčných a nefunkčných požiadaviek, zameriame sa na technológie tvorby webových aplikácií, ktoré som si vybral pre riešenie tejto diplomovej práce, konkrétne technológií na tvorbu frontendu, backendu a taktiež databáz. Následne predstavím návrh používateľského prostredia vo forme wireframu.

## 3.1 Požiadavky

Z analýzy existujúcich riešení web scrapingu a zadaných požiadaviek diplomovej práce som si definoval základné funkčné a nefunkčné požiadavky. Hlavnou časťou aplikácie bude nastavovanie a spúšťanie web scraperov. Vedľajšou časťou bude klientské vyhľadávanie, ktoré bude slúžiť na filtrovanie extrahovaných výsledkov.

### 3.1.1 Funkčné požiadavky

Funkčné požiadavky sú definované ako vlastnosti a funkcie produktu/aplikácie, ktoré umožňujú používateľovi dosiahnuť svoje ciele.

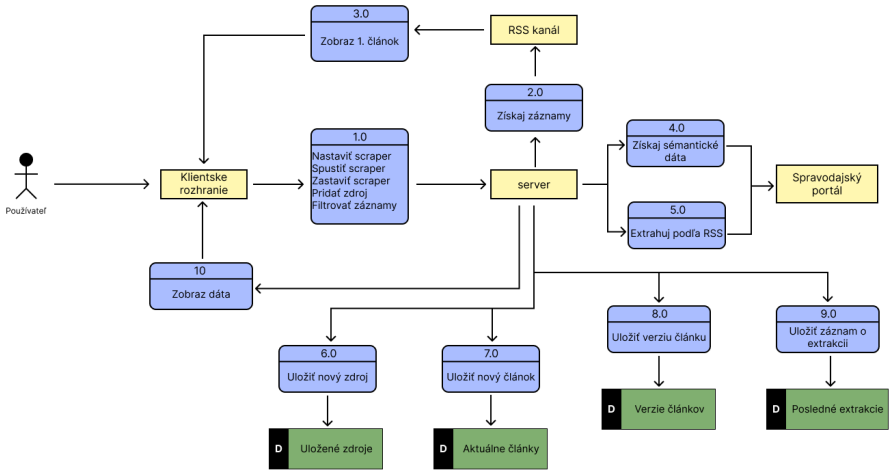
- **F1: Pridanie a odobratie RSS zdroja**: Používateľ bude mať možnosť pridať alebo odobrať RSS zdroj pre ktorý chce daný scraper spúšťať,
- **F2: Zobrazenie uložených zdrojov**: Používateľ bude mať možnosť jednoducho sa pohybovať medzi nastaveniami jednotlivých zdrojov, ktoré ho budú informovať o ich stave,
- **F3: Nastavenie extrakcie údajov**: Používateľ bude mať možnosť presne definovať dáta ktoré chce z jednotlivých článkov extrahovať,
- **F4: Možnosť výberu z preddefinovaných nastavení**: Používateľ bude mať možnosť si vybrať z preddefinovaných možností extrakcie,
- **F5: Zobrazenie aktuálnych nastavení**: Používateľovi sa po načítaní stránky na nastavenie scrapera zobrazia posledne uložené nastavenia,
- **F6: Náhľad na extrahované dáta v reálnom čase**: Používateľ bude v reálnom čase vidieť aké dáta sa budú extrahovať podľa jeho nastavení,
- **F7: Spustenie a zastavenie extrakcie dát**
- **F8: Plánovanie opakovanej extrakcie**: Možnosť nastavenia časového opakovania extrakcie dát,

- **F9: Zobrazenie extrahovaných dát**: Používateľ bude mať možnosť si zobraziť všetky extrahované dáta ktoré budú ukladané v databáze,
- **F10: Zobrazenie výsledkov**: Používateľ bude mať možnosť si zobraziť informácie o jednotlivých behoch scrapera.
- **F11: Filtrovanie extrahovaných dát**: Možnosť filtrovania výsledkov na základe používateľa,
- **F12: Zobrazenie verzií pre jednotlivé extrahované články**: Používateľ bude mať možnosť si zobraziť rozličné verzie pre extrahovaný článok,
- **F13: Zobrazenie originálneho článku na príslušnom portáli**: Používateľ bude mať možnosť si zobraziť článok na príslušnom portáli alebo priamo v aplikácií.

### 3.1.2 Nefunkčné požiadavky

Nefunkčné požiadavky definujú ako by sa systém mal chovať, nedefinujú priamo funkcie.

- **N1: Webová aplikácia**: Implementácia aplikácie bude vo forme webovej aplikácie,
- **N2: Funkčnosť aplikácie bez potreby autentifikácie**: Aplikácia na jej fungovanie nebude potrebovať autentifikáciu používateľa,
- **N3: Tutoriál ako aplikáciu používať**: Priehľadný používateľský tutoriál ktorý používateľovi vysvetlí jednotlivé kroky a požiadavky systému pre konfiguráciu scraperov, pridávania zdrojov a možnosti filtrovania,
- **N4: Dokumentácia**: Aplikácia bude poskytovať dokumentáciu v ktorej sa používateľ dozvie možnosti nastavovania extrakcie údajov a pravidlá ktoré pre extrakciu musí dodržiavať.




**Obr. 3.1** Jednoduchý diagram zobrazujúci základné procesy a časti systému

## 3.2 Výber technológií

Výber technológií je podstatnou súčasťou tvorby akejkoľvek webovej aplikácie. Výber technológií bol do značnej miery ovplyvnený analytickou časťou a požiadavkami zadania diplomovej práce.

### 3.2.1 Frontend

Frontend je časť webovej aplikácie ktorá je v priamom kontakte s používateľom. Sú to jednotlivé časti stránky ktoré používateľ vidí, s ktorými vie pracovať, ktoré mu poskytujú primárne informácie obsiahnuté na stránke[31]. Základom vývoju frontendu je UI(User Interface) a UX (User Experience). To ako stránka na prvý pohľad vyzerá je dôležitým faktorom toho, ako dlho a či vôbec používateľ chce s danou stránkou interagovať, a teda aké pocity v ňom daný systém zanechá. Dôležité sú akcie ktoré používateľ na stránke dokáže vykonať, a to, ako ho systém informuje o tom čo práve vykonal, a čo ešte vykonať môže[32]. Základom pri vývoji frontendu sú technológie HTML, CSS a JavaScript. Jazyk HTML ( Hyper Text Markup Language) slúži na definovanie štruktúry a elementov z ktorých sa samotná stránka skladá. CSS (Cascading Style Sheets) je kaskádový jazykový štýl ktorým sa definuje dizajn jednotlivých HTML elementov. JavaScript je programovací jazyk ktorý doplňuje HTML a CSS a vďaka ktorému je možné tvoriť dynamické stránky, pracovať s jednotlivými elementami, vytvárať animácie a mnohé iné. V dnešnej dobe sa na tvorbu webových stránok používajú JavaScriptové frameworky, ktoré zjednodušujú prácu programátorom pri tvorbe webových stránok. Medzi tieto frameworky patrí Angular, React, Vue.js jQuery a iné. Pre vytvorenie frontendovej časti tejto diplomovej práce som sa rozhodol pre využitie React. Tento framework som si vybral z hľadiska jeho flexibility, rozsiahlej komunite, množstve rozšírení a knižníc.

#### 3.2.1.1 React

React je frontendová open-source knižnica JavaScriptu. React patrí medzi najvyužívanejšie frameworky na tvorbu webových stránok[26]. React umožňuje tvoriť používateľské prostredie z komponentov[33], čo je vhodné pre opakujúce sa prvky na stránke ako tlačidlá, karty, menu a pod. React pracuje s takzvaným virtuálnym DOM (Document Object Model). DOM je reprezentácia obsahu dokumentu v stromovej štruktúre, kde každý uzol stromu je reprezentovaný ako objekt[34]. Virtuálny DOM prechádza a porovnáva stavy komponentov v štruktúre a aktualizuje iba tie komponenty v reálnom DOM ktorých stav sa zmenil, namiesto toho aby aktualizoval všetky komponenty pri každej úprave. React využíva JavaScriptové rozšírenie JSX (JavaScript XML) ktoré umožňuje písať HTML kód vo vnútri JavaScriptovej funkcie. React umožňuje jednoduchšie písanie dynamického kódu, pretože je potrebných menej riadkov kódu a obsahuje mnohé funkcie, a aj tým, že tiež využíva deklaratívny prístup. Deklaratívny prístup v React znamená, že programátor definuje to, ako by malo používateľské prostredie vyzerať na základe stavu komponentu a React aktualizuje DOM[35]. Pre React je vytvorených mnoho knižníc a rozšírení, ktoré zjednodušujú prácu s komponentami, vytvárajú interaktívne prvky a podobne.

### 3.2.2 Backend

Backend je časť aplikácie ktorú používatelia nevidia. Backend spravuje a spracováva funkcionalitu webovej stránky, ako aj spracováva požiadavky od používateľa. Ak používateľ interaguje s frontend častou aplikácie, tieto dáta sú odoslané z frontendu na backend, ktorý ich spracuje, vyhodnotí, vykoná príslušnú akciu, vráti požadované dáta. Backend bude poskytovať API pre získavanie informácií z frontendu a bude poskytovať funkcie na chod a správu scrapera, rovnako ako aj pristupovať do databázy ku uloženým dátam. Backendová časť sa zaoberá aj pripojovaním, zapisovaním, čítaním dát z databázových serverov. Pre vypracovanie tejto diplomovej práce som

sa na základe jej požiadaviek a analytickej časti 2.4 rozhodol využiť technológiu Node.js, ktorá je priblížená v analytickej časti 2.4.2.

### 3.2.3 Databáza

Databáza je organizovaná kolekcia štruktúrovaných dát ktoré sú zväčša uložené v elektronickom v nejakom počítačovom systéme a je kontrolovaná DBMS (Database Management System). Hlavnou úlohou databázy je uchovávať veľké množstvo dát. Existujú rôzne druhy databáz ako napríklad relačné databázy, NoSQL databázy, Key-Value databázy a iné[36]. Relačné databázy organizujú dáta do riadkov a stĺpcov, ktoré spoločne vytvárajú tabuľky v ktorých sú definované vzťahy. Dáta môžu distribuované medzi viacero tabuliek, ktoré je možné spájať alebo využívať primárne a sekundárne kľúče. Tieto kľúče slúžia ako identifikátory a vyjadrujú určité vzťahy medzi dátami[37]. Relačné databázy ukladajú dáta podľa predefinovaných pravidiel tabuliek, na rozdiel od NoSQL databáz. Tieto databázy nepracujú s pravidlami relačných databáz, čo im umožňuje spracovávať neštruktúrované alebo semištrukturované dáta. NoSQL databázy majú vlastnosti ako flexibilné schémy, možnosti horizontálneho škálovania, rýchle dotazovanie[38]. Medzi NoSQL patria MongoDB, Neo4j, Redis, Apache Casandra.

Na vypracovanie tejto diplomovej práce som si zvolil MongoDB databázu. MongoDB som si zvolil najmä z hľadiska flexibilnosti ukladaných záznamov, nakoľko schéma konfigurácie scrapera je určená len do určitej miery a nastavenia závisia od používateľa samotného, čo ovplyvňuje aj samotnú štruktúru záznamu ktorý bude ukladaný.

#### 3.2.3.1 MongoDB

MongoDB je open-source, NoSQL, dokumentovo založená databáza. Dokumentové databázy sú navrhnuté tak, aby jednotlivé záznamy ukladali do formátov podobných JSON. Dokumenty v MongoDB sú serializované ako JSON dokumenty, interne sú však tieto záznamy ukladané vo formáte BSON (Binárny JSON). Každému dokumentu ktorý je v MongoDB vytvorený je automaticky priradený jedinečný identifikátor „ObjectId“. MongoDB je rovnako aj multiplatformová databáza a teda je kompatibilná s rôznymi operačnými systémami ako je Windows, Linux, Mac OS. MongoDB databázu je možné jednoducho horizontálne škálovať a taktiež podporuje „sharding“, teda techniku, ktorá rozdeľuje dáta medzi viacero databázových serverov. MongoDB má flexibilný dizajn, teda nie je potrebné definovať vopred pevnú štruktúru do ktorej sa dáta budú ukladať ale je možné jednoducho upravovať štruktúru podľa potreby. Ďalšou z vlastností je Ad-hoc dotazovanie, čo zabezpečuje dotazovanie sa na dáta bez potreby definovania pevnej štruktúry. Záznamy v MongoDB sú štruktúrované tak, že je možné vkladať jednotlivé záznamy dokopy, avšak je potrebné dať pozor, pretože limit jedného dokumentu je obmedzený na 16MB[39].

**Výpis kódu 3.1** Štruktúra JSON záznamu s akými MongoDB pracuje

```
{
  "_id": {
    "$oid": "65f95aacd83bc4a15"
  },
  "name": "Example Website",
  "url": "https://www.example.com",
  "informations": {
    "names": ["Martin", "Michal", "Peter"],
    "places": {
      "Slovakia": {
        "capital": "Bratislava",
        "size": "49 035 km^2",
        "language": "slovak"
      }
```

```
    }
  }
}
```

## 3.3 Návrh databázy

Návrh štruktúry databázy je zobrazený na obrázku 3.2. Databáza sa bude skladať zo štyroch kolekcií.

V prvej kolekcii označenej ako *Data Source* budú ukladané všeobecné nastavenia pre jednotlivé scrapery. Každý záznam bude obsahovať jedinečný identifikátor (name), adresu pre RSS zdroj, nastavenú frekvenciu, aktívnu konfiguráciu a aktuálne nastavenia extrahovania dát, ktoré sa budú meniť dynamicky podľa nastavení používateľa.

Druhá kolekcia označená ako *Current Articles* bude ukladať jednotlivé extrahované záznamy. Každý záznam bude vytvorený podľa nastavenia extrakcie, ktoré je uložené v prvej kolekcii a taktiež aj identifikátor scrapera označený ako *sourceID*, aby sa dal jednoducho rozoznať zdroj odkiaľ boli dáta extrahované. Tento identifikátor je hodnota *name* z prvej kolekcie. V tejto kolekcii sa budú nachádzať jedinečné záznamy článkov.

V tretej kolekcii označenej *Versions* budú ukladané verzie jednotlivých článkov, a teda štruktúra ukladaných dát bude závisieť na nastaveniach scrapera. Záznamy v tejto kolekcii budú filtrované na základe „guid“ článku, ktorý je jedinečným identifikátorom pre konkrétny článok.

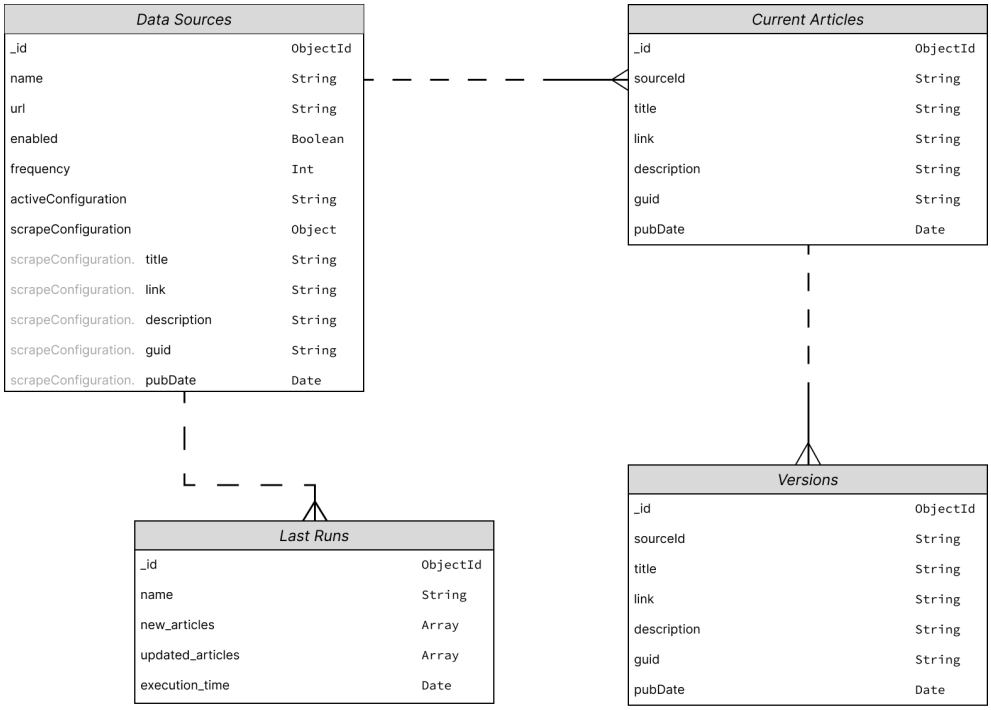
Štvrtá kolekcia označená ako *Last Runs* bude obsahovať záznamy o spustení jednotlivých extrakcií, ktoré scraper vykonal. V tejto kolekcii sa budú nachádzať údaje ako *execution_time*, ktorý bude určovať dátum a čas kedy bol scraper spustený, *new_articles* bude obsahovať *guid* nových článkov ktoré boli extrahované počas tohto behu, a tiež *updated_articles* bude obsahovať zoznam *guid* článkov, ktoré boli počas tohto behu upravené.

## 3.4 Návrh používateľského rozhrania

Používateľské rozhranie je časť systému ktorú používateľ vidí, priamo k nej pristupuje, interaguje s ňou. Návrh používateľského prostredia je dôležitým krokom pri tvorbe systému. To ako systém vyzerá, ako s ním vie používateľ interagovať, je dôležitým krokom pri dosiahnutí používateľského cieľa. Používateľské prostredie sa skladá z mnohých elementov ako napríklad tlačidiel, obrázkov, videí, animácií a ďalších iných. Cez používateľské prostredie môže systém prijímať vstupy od používateľa, na základe ktorých môže vykonávať rozličné akcie, napríklad filtrovanie a zobrazovanie záznamov z databáz, pridávanie záznamov, hlasovanie, vyplňovanie formuláru a mnohé iné. Pri návrhu používateľského rozhrania som sa opieral o funkčné požiadavky a tiež o analýzu existujúcich riešení. Súčasne som sa zameral na jednoduchý dizajn, aby používateľ nemal problém v navigácií a používaní aplikácie.

### 3.4.1 Domovská obrazovka

Základné menu sa bude nachádzať v hornej časti obrazovky, ktoré je rovnaké pre všetky jednotlivé podstránky. Aplikácia obsahuje možnosť pridania a odobrania zdroja a zoznam uložených zdrojov. Pre vytvorenie a pridanie nového zdroja je potrebné určiť jeho názov a vložiť URL adresu pre RSS kanál. Pre spracovanie týchto údajov stačí kliknúť na tlačidlo *Použiť*. Aby sa v aplikácií nenachádzali rovnaké zdroje viackrát, zobrazí sa po kliknutí aj správa, či sa zdroj podarilo uložiť, alebo boli použité neplatné informácie. Na tejto stránke sa tiež zobrazujú uložené zdroje. Pre prehľadnejšie vyobrazenie zdroja v zozname sú použité informácie o zdroji ako názov, nastavená frekvencia ktorá je určovaná v minútach a status, ktorý zobrazuje či scraper pre daný




**Obr. 3.2** Návrh databázovej štruktúry

zdroj beží alebo nie. Každý zdroj je možné jednoducho zmazať a nastaviť. Štruktúru domovskej stránky je možné vidieť na obrázku 3.3

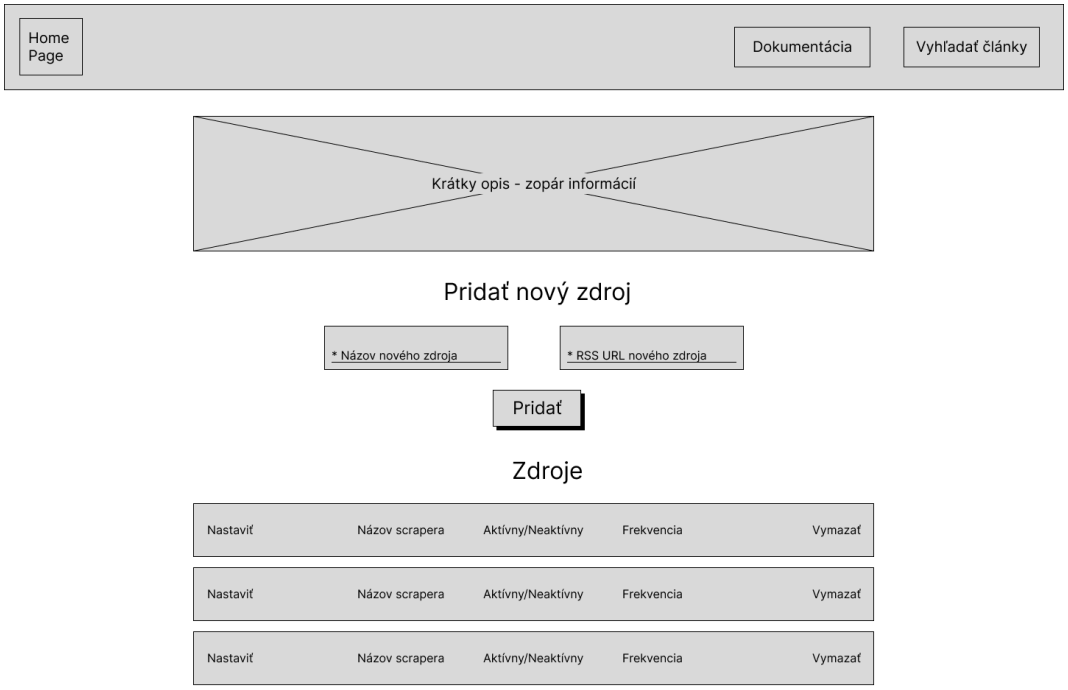
### 3.4.2 Nastavenia zdroja

Nastavenie scrapera pre určitý zdroj bude možné zobraziť po kliknutí na tlačidlo na domovskej stránke v zobrazení zdroja. Stránka pre nastavenie scrapera je najrozsiahlejšia čo sa týka poskytnutých funkcií a nastavení. V hornej časti stránky pod základnou navigáciou (navbar) sa nachádza názov a tiež doplňujúce informácie pre funkcionalitu nastavení scrapera pre daný zdroj. Nasledujúce prvky by sa dali rozdeliť do dvoch skupín.

Prvou skupinou sú prvky, ktoré slúžia na samotné nastavenie scrapera a na obrázku 3.4 sú znázornené na ľavej strane.

Nastaviť scraper pre zdroj bude možné z vopred pripravených nastavení, ktoré sú zobrazené na 3.4 na ľavej strane a sú označené tlačidlá šablón. Predpripravené nastavenia obsahujú dáta získané z RSS záznamu alebo zo sémantických dát uverejnených v článku. Pre komplexnejšie nastavenie scrapera je určená časť ktorá sa nachádza pod uvedenými tlačidlami šablón. Prvé 3 prvky sú určené na nastavenie pravidiel extrahovania dát pre povinné záznamy. Povinné záznamy sú určené kvôli konzistentnosti extrahovaných dát a ich možnosti ich následne vyhľadávať a porovnávať. Definovať v nastaveniach pre extrakciu je možné priamo HTML elementy, ktoré sú využívané v štruktúre článkov, aj s príslušnými triedami alebo identifikátormi, alebo dáta ktoré boli získané a sú obsiahnuté v predpripravených nastaveniach. Doplňujúci prvok ktorý sa nachádza pod týmito 3 povinnými prvkami, slúži na nastavenie ďalších údajov. Po nastavení pravidiel na extrahovanie nasleduje možnosť spustenia alebo zastavenia scrapera. Je možné nastaviť frekvenciu na základe ktorej sa scraper bude automaticky spúšťať. Nastavovanie frekvencie je určené v minútach.

V druhej skupine, ktorá je zobrazená na obrázku 3.4 na pravej strane sú prvky, ktoré slúžia



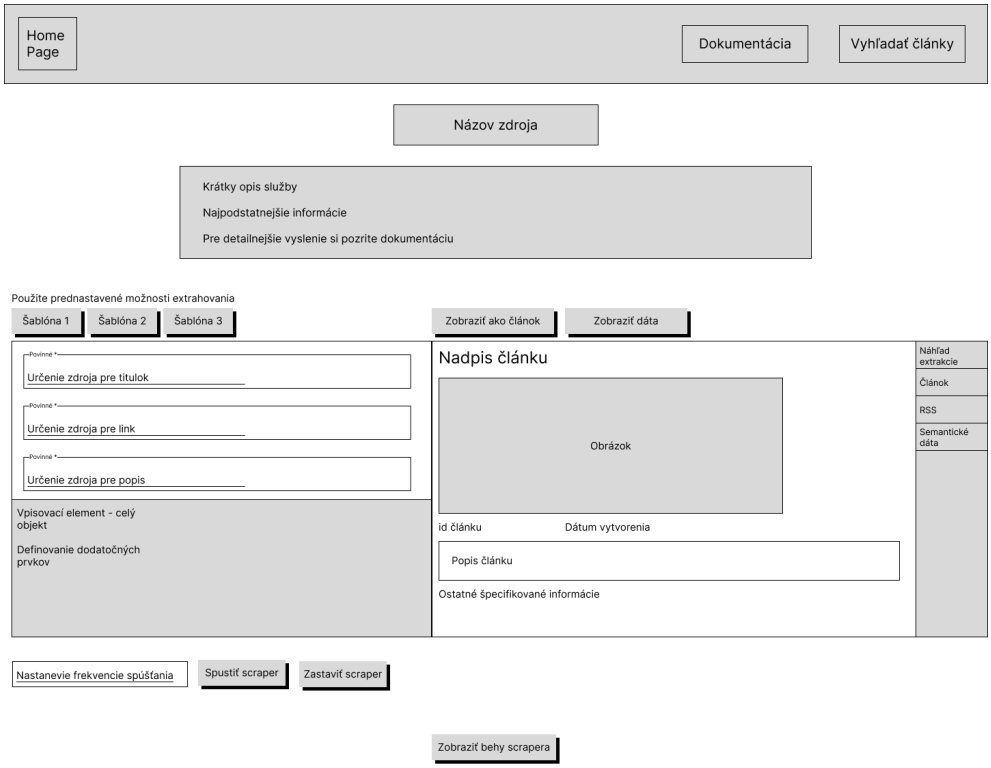

**Obr. 3.3** Návrh domovskej obrazovky

na zobrazovanie informácií o extrahovaní. Prvou možnosťou zobrazenia je náhľad na extrahované dáta. Tento náhľad je spracovávaný v reálnom čase a odvíja sa od nastavení, ktoré boli zadané a zobrazuje informácie, ktoré budú pri spustení scrapera extrahované. Tieto dáta sa dajú zobraziť v dvoch formách. Prvá varianta zobrazuje dáta vo forme podobnej JSON dokumentu. Druhý variant je zobrazený na obrázku 3.4, a táto forma zobrazuje dáta v štruktúre podobnej článkom. Medzi týmito variantami je možné sa preklikávať za pomoci tlačidiel.

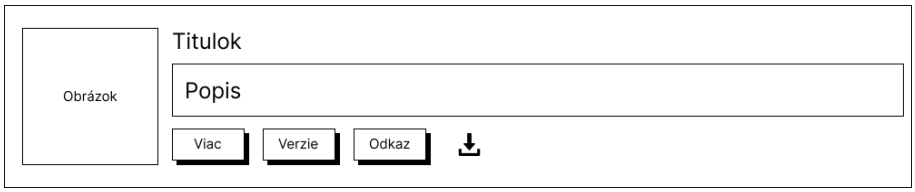
Aplikácia by mala obsahovať aj náhľad originálneho článku z internetového spravodajstva v takej podobe, ako ho publikovalo samotné spravodajstvo. Aplikácia získa z RSS kanálu záznam o prvom, teda najnovšom článku a na základe zverejnenej URL adresy v RSS zázname tento článok vyhľadá na spravodajskom portály, a zobrazí ho do náhľadu. Tento náhľad je možné zobraziť po stlačení tlačidla „Článok“, ktorý sa nachádza v bočnom menu. Samotný článok sa potom zobrazí na mieste, kde bol predtým predstavený náhľad nastavení. Tento prístup som zvolil z hľadiska toho, aby nebolo nutné opúšťať stránku s nastavením scrapera pre získanie HTML dát požadovaných prvkov na extrakciu. Bočné menu obsahuje 4 možnosti zobrazenia, okrem náhľadu na extrahované dáta a samotný článok aj možnosť zobrazenie dát, ktoré boli extrahované z RSS záznamu, alebo sémantických dát ktoré boli extrahované z daného článku.

### 3.4.3 Vyhľadávanie

V tejto časti aplikácie budú zobrazované záznamy ktoré boli vytvorené extrahovaním dát. Tieto záznamy bude možné filtrovať na základe určenia dátumu a času, definovania identifikátora pre scraper, alebo vyhľadania konkrétneho záznamu podľa prislúchajúceho identifikátora. Po použití filtra sa zobrazia záznamy článkov. Tieto záznamy sa dajú zoradiť od najnovších záznamov po najstaršie a naopak, cez pripravené tlačidlá 3.6. Zobrazenie článkov taktiež obsahuje tlačidlo na stiahnutie záznamov filtrovania. Z informácií získaných pri extrakcii obsahuje prvotný náhľad




**Obr. 3.4** Návrh obrazovky pre nastavenie scrapera



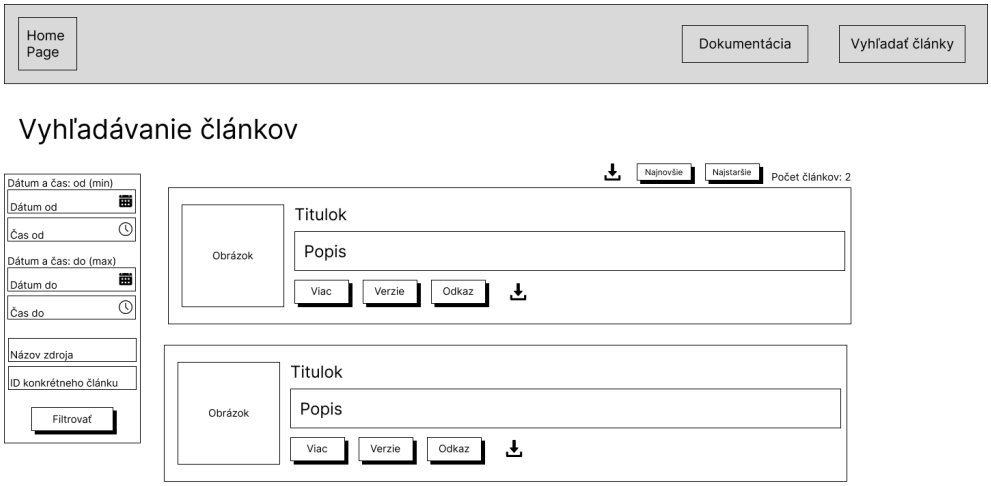

**Obr. 3.5** Návrh zobrazenia filtrovaného článku

článkov ich názov, obrázok, dátum vytvorenia článku, identifikátor článku a popis. Každý náhľad článku obsahuje 4 tlačidlá 3.5.

Prvé tlačidlo je odkaz na originálny článok zverejnený na spravodajskom portáli, druhé tlačidlo zobrazí verzie článku, tretie tlačidlo slúži na zobrazenia dodatočných informácií. Štvrté tlačidlo slúži na stiahnutie údajov. Po jeho kliknutí sa zobrazia možnosti formátov v akom je možné záznam stiahnuť.

### 3.4.4 Verzie článkov

Aplikácia by mala obsahovať možnosť nahliadnuť do verzií článku. Náhľad článku po vyfiltrovaní výsledkov obsahuje tlačidlo ktoré slúži na zobrazenie verzií. Po jeho stlačení sa zobrazí samostatná podstránka, ktorá obsahuje všetky uložené verzie tohto článku. Štruktúra zobrazenia verzií článkov je rovnaká ako pri filtrácií. Každá zobrazená verzia obsahuje titulok, obrázok, dátum vytvorenia článku, identifikátor článku. Každý náhľad verzie tiež obsahuje tlačidlo pre stiahnutie údajov a tlačidlo pre zobrazenie originálneho článku na spravodajskom portáli.

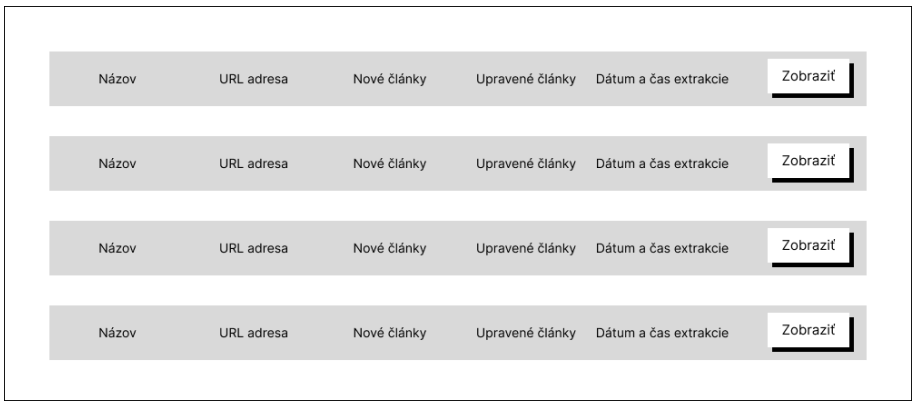



**Obr. 3.6** Návrh zobrazovania a filtrovania výsledkov

### 3.4.5 Ukončené extrakcie

V aplikácií by sa mala nachádzať možnosť zobrazenia informácií o jednotlivých behoch scrapera. Na stránke s nastavením scrapera sa nachádza tlačidlo ktoré zobrazí zoznam posledných behov scrapera. V tomto zozname sa nachádzajú informácie o čase a dátume kedy bola extrakcia zahájená, počet novo pridaných článkov, počet upravených článkov a tlačidlo pre zobrazenie náhľadu na tieto články 3.7.

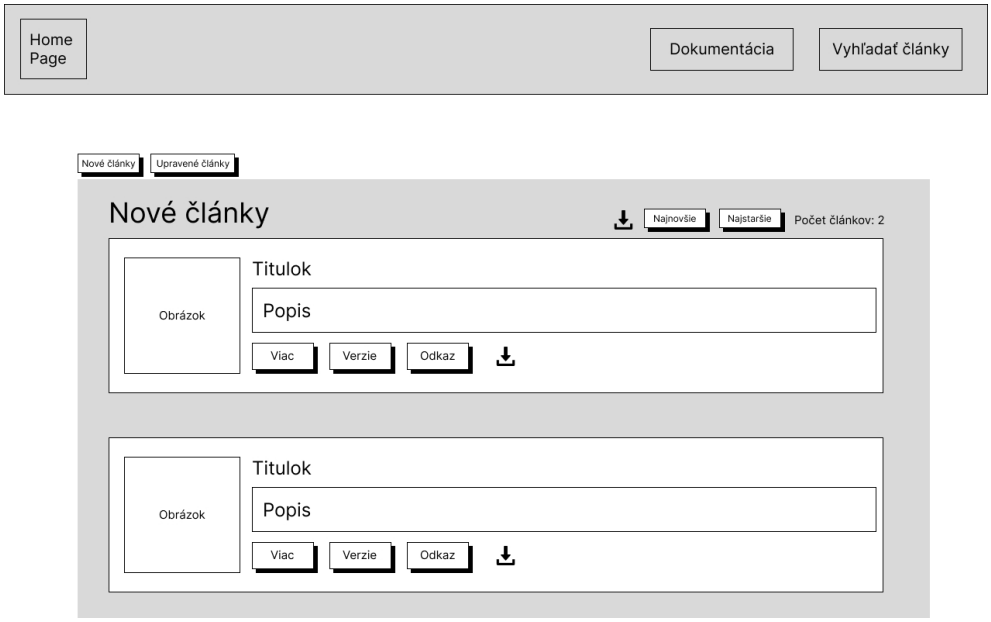
Štruktúra náhľadu je podobná ako pri filtrácií článkov, avšak články sú rozdelené na dve kategórie, novo pridané a upravené články, a medzi týmito kategóriami je možné sa preklikávať, ako je vidieť na obrázku 3.8. Každý náhľad na článok je rovnaký ako náhľad článku pri filtrovaní.




**Obr. 3.7** Zoznam behov scrapera

### 3.4.6 Dokumentácia

Dokumentácia dostupná zo základnej navigácie v hornej časti, bude obsahovať bližšie informácie o fungovaní aplikácie, ako aj postup a vysvetlenie nastavení scrapera. V dokumentácií je uvedená




**Obr. 3.8** Zobrazenie článkov ktoré boli v danom behu spracované

aj časť, ktorá približuje postup nastavovania scrapera, nastavenia databázy, približuje syntax zapisovania jednotlivých príkazov.

# Kapitola 4
# Implementácia

V tejto kapitole popíšem implementáciu funkčných požiadaviek ako pre frontend tak aj pre backend, taktiež priblížim použité knižnice a balíčky. Pri implementácií požiadaviek som vychádzal z návrhu používateľského rozhrania, ktoré som opísal v kapitole 3.4.

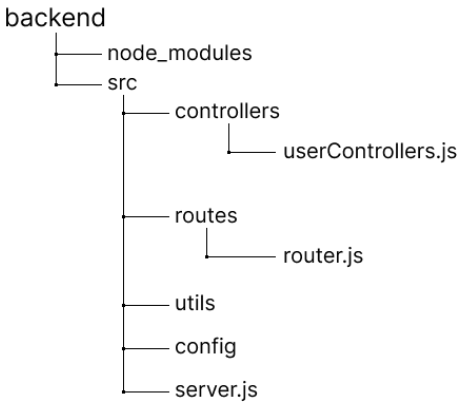
## 4.1 Backend

Backend-ovú časť aplikácie som implementoval v prostredí Node.js. Pre inicializovanie projektu som použil príkaz *npm init*. Pri použití tohto príkazu je potrebné odpovedať na pár otázok ohľadom projektu ako je napríklad meno, popis, kľúčové slová a pod. Výstupom tohto príkazu je *packet.json* súbor. Pre Node.js neexistuje vstavaný príkaz ktorý by vytvoril štruktúru projektu automaticky.

### 4.1.1 Štruktúra backendu aplikácie

Samotný Node.js neobsahuje vstavaný príkaz ktorý by umožňoval vytvoriť základnú štruktúru projektu automaticky, preto som sa rozhodol vytvoriť štruktúru manuálne. Node.js samotný nevyžaduje dodržiavanie takejto štruktúry kódu, avšak akákoľvek logická štruktúra zdrojového kódu je prehľadnejšia. Touto štruktúrou som zabezpečil priehľadnejšie orientovanie sa v zdrojovom kóde. Štruktúra backendovej časti aplikácie je zobrazená 4.1, avšak na obrázku nie sú vyobrazené všetky súbory, iba tie z môjho pohľadu najpodstatnejšie, a rovnako aj organizácia jednotlivých adresárov.

- **node_modules**: Implementované balíky a knižnice ktoré aplikácia používa sú uložené v tomto adresári. Tieto balíky nie je potrebné editovať, sú nainštalované automaticky keď sa balíky inštalujú cez *npm* alebo *yarn*,
- **controllers**: V tomto priečinku sa nachádzajú súbory, ktoré obsahujú hlavnú logiku aplikácie, a teda aké procesy sa budú vykonávať pri akých vstupoch,
- **routes**:V tomto priečinku sa nachádza súbor *router.js* ktorý slúži na definovanie všetkých API, ktoré aplikácia používa,
- **utils**: Tento priečinok obsahuje pomocné funkcie, ktoré sú využívané v rôznych častiach aplikácie,
- **config**: Tento priečinok obsahuje konfiguračné súbory,




**Obr. 4.1** Štruktúra backend aplikácii

- **server.js**: Hlavný súbor aplikácie v ktorom sa nachádza inicializovanie servera, middlewaru, CORS a pod. Spustením tohto súboru sa backendová časť aplikácie uvedie do prevádzky.

### 4.1.2 Využívané knižnice a rozšírenia

Pri riešení backendovej časti aplikácie som využil nasledujúce knižnice a rozšírenia. Vďaka týmto knižniciam a rozšíreniam som bol schopný implementovať predpripravené funkcie, ktoré mi uľahčili prácu s niektorými časťami vývoja. Skrz knižnice ako Cheerio a Puppeteer som bol schopný implementovať logiku nastavovania a samotného extrahovania dát.

- **Express**[1]: je flexibilný framework pre Node.js, ktorý slúži na tvorbu webových API koncových bodov pre spracovávanie HTTP požiadavkov,
- **Cheerio**[2]: bližšie popísané v analytickej časti 2.4.2.1,
- **Puppeteer**[3]:bližšie popísané v analytickej časti 2.4.2.2,
- **Axios**[4]: je HTTP klient pre Node.js a prehliadače, ktorý poskytuje jednoduché rozhranie na vytváranie asynchrónnych HTTP požiadavkov,
- **MongoDB**[5]: je ovládač pre Node.js, ktorý umožňuje komunikovanie s databázou z prostredia Node.js a umožňuje z databázy čítať, zapisovať, upravovať alebo mazať,
- **Dotenv**[6]: je knižnica pre Node.js, ktorá umožňuje načítanie konfiguračných premenných zo súboru *.env* do prostredia aplikácie,
- **Rss-url-parse**[7]: je to jednoduchý modul pre Node.js, ktorý slúži na analýzu a spracovanie URL adresy RSS zdroja,
- **Node-cron**[8]: Node-cron je knižnica pre Node.js, ktorá umožňuje plánovať pravidelné úlohy,
- **cron-parser**[9]: Je Node.js knižnica pre analýzu a manipuláciu s cron inštrukciami.

[1] https://www.npmjs.com/package/express
[2] https://www.npmjs.com/package/cheerio
[3] https://www.npmjs.com/package/puppeteer
[4] https://www.npmjs.com/package/axios
[5] https://www.npmjs.com/package/mongodb
[6] https://www.npmjs.com/package/dotenv
[7] https://www.npmjs.com/package/rss-url-parser
[8] https://www.npmjs.com/package/node-cron
[9] https://www.npmjs.com/package/cron-parser

### 4.1.3 Funkčné požiadavky

Pre backendovú časť webovej aplikácie som implementoval všetky funkčné požiadavky, ktoré som si definoval v kapitole 3.1, a v tejto kapitole popisujem ich implementáciu z hľadiska serverového spracovania.

#### 4.1.3.1 Pridávanie, odoberanie a zobrazovanie zdrojov

Aplikácia je navrhnutá tak, že na jej fungovanie nie je potrené sa autentifikovať, a dáta sú teda zobrazené priamo po jej načítaní. Pridávanie nového zdroja je umožnené definovaním jeho názvu a RSS adresy, kedy je názov následne využívaný ako identifikátor daného zdroja v rámci extrahovaných článkov. Preto som sa rozhodol implementovať funkcie pre overenie existujúceho zdroja a obmedzil som možnosť vytvárania duplicitných zdrojov tak, že pre zadanú RSS adresu je možné mať vytvorený iba jeden záznam. Na vytváranie API koncových bodov a spracovanie HTTP požiadaviek využívam Node.js framework Express.

Express je flexibilný framework ktorý umožňuje jednoducho definovať a vytvárať serverové aplikácie a API, čo zjednodušuje tvorbu samotných aplikácií. Po inštalácií frameworku Express ho stačí už len v aplikácií importovať a vytvoriť požadované API endpointy.

Použitím frameworku Express som vytvoril API-endpoint pre pridávanie nového zdroja, ktorý backend očakáva na *POST /api/newSource HTTP/1.1*. Aplikácia očakáva názov a RSS adresu zdroja. Po získaní informácií systém vykoná ich kontrolu. Aplikácia overí, či zadané informácie sa nezhodujú s názvami a RSS adresami už uložených zdrojov v databáze. Pokiaľ sa zhoduje názov alebo RSS adresa, aplikácia nový zdroj nepridá. Toto obmedzenie som navrhol najmä preto, aby nebolo možné mať viac záznamov s rovnakou RSS adresou, čím som obmedzil extrahovanie duplicitných informácií z rovnakého spravodajského portálu.

Pri overovaní RSS adresy sa okrem jej výskytu v databáze uložených zdrojov kontroluje aj jej platnosť. Platnosť RSS záznamu sa overuje na základe odoslania požiadavky na poskytnutú adresu. Ak je požiadavka platná, systém získa prvý záznam z RSS kanálu, čím sa overí dostupnosť a relevantnosť zadanej adresy. Pokiaľ však aplikácia nedokáže získať prvý RSS záznam, je zadaná adresa označená ako neplatná. Pri neplatných informáciách sa odošle chybové hlásenie ako odpoveď na požiadavku pridania nového zdroja na frontend pre informovanie užívateľa. Pokiaľ sú informácie overené, aplikácia vytvorí požadovanú štruktúru záznamu nového zdroja, doplní priložený názov a RSS adresu, nastaví frekvenciu na hodnotu 0 a záznam o novom zdroji uloží do príslušnej databázy.

**Výpis kódu 4.1** Kontrola zadanej RSS adresy pri pridávaní nového zdroja

```
function existingUrl(url){
        const result = await collection.findOne({ url: url });
        if(!result){
            let rssFeed = await parser.parseURL(url);
            const firstItem = rssFeed.items[0];
            if(firstItem === null){
                return false;
            }else{
                return true;
                }
            }
        else{
            return false;
            }
    }
```

Na ukladanie údajov využívam databázu MongoDB, s ktorou je komunikácia zabezpečená cez Node.js ovládač *„mongodb“*, ktorý umožňuje prístup ku uloženým záznamom na čítanie,

zapisovanie, úpravu a mazanie. Pri pristupovaní ku kolekciám v databáze využívam modul Dotenv, ktorý umožňuje pracovať so systémovými premennými, ktoré sú uložené v súbore *.env*. Využívaním systémových premenných som zabezpečil jednoduchšie definovanie často využívaných zdrojov v rámci aplikácie.

Pre odstránenie požadovaného zdroja som implementoval samostatné API pre prehľadnosť používaných funkcií. V tomto prípade backendový server spracuje požiadavku ktorá obsahuje iba názov požadovaného záznamu, ktorý má byť zmazaný. Nakoľko každý názov zdroja slúži ako jeho jedinečný identifikátor, funkcia na zmazanie skontroluje, či daný záznam vôbec v databáze existuje, a ak ho získa, tak ho pomocou ovládača pre MongoDB zmaže z príslušnej databázy.

Jednou z definovaných funkčných požiadaviek je zobrazenie uložených zdrojov pre extrakciu. Túto požiadavku som implementoval priamo pri zobrazovaní domovskej obrazovky. Vzhľadom k tomu, že aplikácia nevyužíva autentifikáciu používateľa a je priamo pripojená k databáze, nebolo potrebné implementovať mechanizmy identifikácie a autorizácie. Systém spracuje HTTP požiadavku pri zobrazení domovskej stránky, a následne odošle dotaz na databázu pre získanie záznamov o všetkých uložených zdrojoch a vráti tieto informácie ako odpoveď na túto požiadavku.

#### 4.1.3.2 Výber z preddefinovaných nastavení

Jednou z požiadaviek tejto diplomovej práce je pripraviť preddefinované nastavenie extrakcie, aby nebolo potrebné zakaždým scraper nastavovať. Pri implementácií týchto preddefinovaných nastavení som sa zameral na dáta pochádzajúce z dvoch zdrojov. Tieto zdroje sú sémantické dáta používané v publikovanom článku a RSS kanál. Nastavenia sú definované v štruktúre key-value.

Pre extrakciu dát z RSS kanálu som použil Node.js modul *Rss-url-parser*. Tento modul síce nepatrí medzi často sťahované, avšak pre účely potrebné na tvorbu tejto webovej aplikácie bol postačujúci. Tento modul spracuje adresu RSS záznamu a vráti jeho obsah ako JSON objekt. Získané dáta však nijako nespracováva, čo je jej výhodou ale zároveň aj nevýhodou. JSON objekt ktorý sa vráti obsahuje všetky záznamy, ktoré daný spravodajský portál zverejňuje v RSS kanály a môže sa tento výstup líšiť od dát, ktoré sú viditeľné pri zobrazení RSS kanála cez webový prehliadač. Pre túto skutočnosť som manuálne obmedzil štruktúru a ukladané informácií tak, aby mali podobnú štruktúru ako tá, ktorá je viditeľná pri zobrazení RSS záznamu cez webový prehliadač. Prvé preddefinované nastavenie scrapera teda využíva jedine dáta obsiahnuté v RSS zázname.

Druhé preddefinované nastavenie extrakcie dát je zabezpečené pomocou sémantických dát. Sémantické dáta získavam pomocou knižnice Cheerio a Axios. Knižnica Axios vykoná dotaz na URL adresu článku, ktorá bola sprostredkovaná z RSS záznamu o článku. Odpoveďou na túto požiadavku je HTML záznam dát, ktorý je následne odovzdaný knižnici *Cheerio* na spracovanie. Sémantické dáta môžu byť na stránkach definované rôzne, záleží na tom, aká technológia je používaná príslušným spravodajským portálom. Používané technológie môžu byť *microformats*, *microdata*, *openGraph*, *RDFa*, *JSON-LD* a ďalšie. Pre získavanie sémantických dát som sa obmedzil na technológiu *openGraph*, nakoľko pri skúmaní článkov, ktoré boli zverejnené na rôznych spravodajských portáloch, ako domácich tak aj zahraničných, bola táto technológia využívaná najčastejšie. Pre získanie sémantických *openGraph* dát zo získaného HTML som napísal *Cheerio* selektor. Pri implementovaní tejto šablóny som narazil na problém vo forme HTTP chybového hlásenia 403. Táto chyba informuje o tom, že prístup na požadovanú stránku bol zamietnutý. V takomto prípade je možné využívať iba dáta obsiahnuté v RSS zázname.

Tretie preddefinované nastavenie na extrakciu je spojenie dvoch predošlých nastavení, podľa RSS a sémantických dát. Ako sémantické tak aj RSS záznamy používajú rovnaké formáty na popisovanie jednotlivých prvkov, avšak tieto formáty sú zapísané odlišne. Napríklad, v RSS zázname sa názov často reprezentuje pomocou elementu *<title>*, zatiaľ čo v sémantických dátach openGraph sa používa vlastnosť *og:title*. Tieto dva formáty teda poskytujú informácie o rovnakých elementoch, ale sú zapísané rozlične. V tejto štruktúre som sa rozhodol dať vyššiu prioritu sémantickým dátam pre RSS dátami. V použitom nastavení sa najprv definujú sémantické dáta,

ktoré sa zapíšu v podobe key-value. Následne sa s týmito hodnotami porovnávajú RSS dáta. Porovnávajú sa jednotlivé formáty v ktorých sú dáta zapisované. Ak sa nájde zhoda v týchto formátoch, napríklad RSS *<title>* a openGraph *og:title*, tak sa táto hodnota preskočí a nezapisuje sa, pretože už je definovaná z openGraph záznamu. Ak sa zhoda nenájde, zapíšu sa tieto RSS dáta podobne vo formáte key-value. Tým sa obmedzila tvorba duplicitných záznamov pri následnom extrahovaní dát.

**Výpis kódu 4.2** Cheerio selektor ktorý vyhľadáva OpenGraph v článkoch a uloží ich vo formáte key-value

```
    const $ = cheerio.load(response.data);
    const metadata = {};
    $('meta[property^="og:"]').each(function () {
        const key = $(this).attr('property').replace('og:', '');
        const value = $(this).attr('content');
        metadata[key] = value;
    });
}
```

#### 4.1.3.3 Nastavenie scrapera

Pre nastavenie scrapera využívam viaceré knižnice ako Cheerio, Puppeteer, Axios a Node-cache. Node-cache je jednoduchá knižnica pre Node.js, ktorá slúži na ukladanie dát do pamäte cache, čo k nim zabezpečuje rýchly prístup. Dáta ktoré sa do cache pamäte ukladajú sú vo formáte *key-value*, a teda pre zapísanie tejto hodnoty je potrebné definovať jej kľúč pod ktorým je možné ju neskôr identifikovať a určiť hodnotu, ktorá sa pod kľúčom uloží. Pri zapisovaní do cache pamäte je tiež potrebné definovať záznam aj s TTL (Time to Live), teda s časovým intervalom, po ktorom sa daný záznam z pamäte vymaže. Túto knižnicu používam pre ukladanie troch druhov záznamov. Týmito záznamami sú RSS dáta najnovšieho článku v RSS kanály, sémantické dáta pre tento článok a súčasne HTML dáta daného článku, ktoré boli získané pomocou knižnice Puppeteer. Jednotlivé dáta ukladám do pamäte s časovým obmedzením 5 minút. Dáta o najnovšom článku som sa rozhodol uložiť do cache pamäte kvôli tvorbe náhľadu na nastavenia extrakcie v reálnom čase. Pri každej zmene ktorú používateľ vykoná pri nastavovaní scrapera, či už sa jedná o prepínanie medzi preddefinovanými šablónami, alebo určenie si vlastných nastavení, sa na backend odošle nová požiadavka, ktorá vygeneruje nový náhľad podľa aktuálne zadaných informácií. Dáta ukladám do cache pamäte z toho dôvodu, aby sa tieto náhľady tvorili práve z nich, čím som obmedzil posielanie väčšieho množstva požiadaviek priamo na servery spravodajských portálov. Táto cache sa využíva iba pri tvorbe náhľadov a spúšťanie scrapera to nijak neovplyvňuje.

Knižnice Puppeteer a Axios využívam pre získanie HTML dát zverejneného článku. Knižnicu Axios využívam kvôli jej rýchlosti spracovania dotazov. Axios bola navrhnutá tak, aby získavala HTML kód zo statických stránok, a teda pri poslaní dotazu sa v odpovedi od servera vráti počiatočný kód HTML stránky. Požiadavka však už nevie zachytiť dáta ktoré sú generované neskôr JavaScriptom. Túto knižnicu využívam ako predvolenú na získavanie dát zo zverejnených článkov, nakoľko väčšinu informácií je schopná získať už požiadavka Axios, čím sa výrazne skráti čakanie na odpoveď od serverov. Knižnicu Puppeteer využívam ako druhú variantu získavania HTML dát, pretože táto knižnica dokáže spracovať JavaScriptom generované dáta, avšak časový úsek medzi odoslaním požiadavky na server spravodajského portálu a jeho odpoveďou je dlhší, ako pri posielaní požiadavky skrz Axios. Informácia o tom, ktorá knižnica má byť pre extrahovanie použitá, sa nachádzajú v tele požiadavky pre nastavenie scrapera pre daný zdroj, a závisí na definovaní užívateľa.

Pre nastavovanie scrapera som na backende vytvoril samostatné API, ktoré očakáva informácie o konfigurácií. Získané dáta sa skontrolujú a spracujú. Na základe poskytnutých informácií sa vytvorí šablóna pre extrahovanie údajov. Šablóna môže byť vytvorená buď to z

```
{
 sourceID: 'primacnn',
 title: { source: 'h1' },
 link: { source: 'RSS' },
 description: { source: 'Semantics ' },
 guid: { source: 'RSS' },
 pubdate: { source: 'RSS' },
 image: {
  source: 'div:nth-child(1) > div > a:nth-child(1) > picture > img[src]'
 },
 tagy: { source: '#article-content > div > section > div onlyValue()' },
 author: { source: 'article > header > ul > li:nth-child(2)' },
 type: { source: 'Semantics' },
 category: { source: 'RSS' },
 portal: {
  source: 'div.atom-breadcrumbs.py-5.my-0.flex.flex-wrap > a:nth-child(1) > span'
 }
}
```

**Obr. 4.2** Databázová reprezentácia používateľom definovanej šablóny pre extrakciu

parametrov ktoré priamo definoval používateľ, alebo sa využije jedno z preddefinovanych nastavení extrakcie. Parametre extrahovania ktoré definoval používateľ prechádzajú kontrolou repetitívnosti kľúčov, aby som zabezpečil jedinečnosť kľúčov v rámci nastavenia extrakcie. Pokiaľ parametre boli zadané korektne, vytvorí sa požadovaná šablóna. Ak parametre neboli korektne definované systém vráti ako odpoveď upozornenie na ich dodatočnú kontrolu. Na základe vytvorenej šablóny sa scraper pokúsi doplniť relevantné informácie získané z RSS kanálu, sémantických dát, alebo vyhľadať definované elementy v článku. Definovať parametre extrakcie je možné v dvoch formátoch.

Prvým formátom je využitie extrahovaných RSS a sémantických openGraph dát. V tomto prípade sa využijú konkrétne hodnoty, ktoré je možné vidieť pri použití šablón. Syntax pre získanie daného prvku som definoval tak, že je potrebné zadať presný názov kľúča a následne definovať hodnotu zdroja, *RSS* pre získanie konkrétnej hodnoty z RSS záznamu, alebo *Semantics*, pre získanie hodnoty zo sémantických openGraph dát.

Druhým formátom je definovanie selektora pre daný prvok. Pre toto používateľské nastavenie som si vybral knižnicu Cheerio. Túto knižnicu som sa rozhodol použiť kvôli jej jednoduchému syntaxu a jej rýchlosti spracovania a vyhľadania dát. Vstup od používateľa je teda tiež vo forme CSS selektora ktorý napíše, alebo využije možnosť skopírovania selektora, ktorá je zabudovaná v DevTools. V šablóne som definoval aj pevné prvky ako identifikátor článku a dátum zverejnenia. Tieto údaje sa dopĺňajú automaticky a nie je možné ich mazať alebo meniť. Oba prvky sú extrahované z RSS záznamu, nakoľko ani identifikátor článku a ani dátum zverejnenia článku nie sú zahrnuté v sémantických dátach. Výsledná šablóna teda môže obsahovať rôzne kombinácie hodnôt ako na obrázku 4.2.

Súčasťou aplikácie má byť aj náhľad na extrahované dáta v reálnom čase. To som implementoval spoločne s tvorbou šablóny. Zakaždým keď sú nastavenia extrakcie pozmenené, server spracuje novú požiadavku a upraví šablónu do požadovanej podoby. Pre získanie skutočných informácií som vytvoril funkciu, ktorá na základe vytvorenej šablóny vyhľadá potrebné informácie. RSS a sémantické dáta spracuje z uložených záznamov a pre definované CSS selektory vykoná extrakciu dát pomocou knižnice Cheerio. Pri tejto extrakcii najprv funkcia overuje, či zadaný selektor je napísaný syntakticky správne. Následne prechádza ku vyhľadávaniu zadaných selektorov. Implementoval som možnosť pre používateľa nastaviť scraper len za pomoci CSS selektorov, avšak bolo potrebné aby som upravil syntax pri zapisovaní niektorých funkcií knižnice *Cheerio*, aby som neobmedzil na jej funkcionalite. Príklad rozdielností syntaxu je zobrazený na obrázku 4.3. Vyhľadávanie je možné nad HTML dátami ktoré boli získané s pomocou knižnice *Axios* alebo *Puppeteer*. Extrahované dáta sú zapísané v forme JSON objektu a sú odoslané na frontend ako odpoveď na zadanú požiadavku.

```
Cheerio zápis: $('div').children('p');
Zápis v aplikácií: div children(p);

//Funkcia na transormáciu príkazu
if(selector.includes('children(')){
        const source = selector.split('children(');
        const regex = /children\(([^)]+)\)/;
        const matches = selector.match(regex);
        const parentSelectors = source[0];
        const childSelectors = matches[1]
        const elements = $(parentSelectors)
                .children(childSelectors)
                .map((index, element) => $(element).text().trim())
                .get()
                .filter(Boolean);
```

**Obr. 4.3** Funkcia využívajúca parameter .children() z Cheerio knižnice spoločne s rozdielnymi zápismi selektora

#### 4.1.3.4 Plánovanie a extrakcia dát

Webový scraper je možné nastaviť tak, aby sťahoval dáta z jedného, alebo viacerých RSS zdrojov zároveň. Extrahovanie dát z konkrétneho RSS zdroja (ďalej iba ako „zdroj“) je možné spustiť jednorázovo, alebo nastaviť plánované spúšťanie. Pre spúšťanie a zastavovanie extrahovania dát z jednotlivých zdrojov som vytvoril samostatné API. V tele požiadavky, ktorú scraper očakáva na tomto API je definovaný názov zdroja, informácie na vytvorenie šablóny, frekvencia a informácia, či sa jedná o spustenie alebo zastavenie sťahovania. Pri spustení extrakcie dát pre vybraný zdroj sa ako prvé skontroluje, či sú všetky povinné prvky definované v šablóne. Medzi povinné prvky, ktoré musia byť pre extrakciu definované som zaradil názov článku, odkaz na článok, popis článku, identifikátor článku a dátum vytvorenia článku. Identifikátor a dátum vytvorenia sú dopĺňané automaticky a používateľ ich nenastavuje manuálne. Tieto informácie som určil ako povinné, aby sa zachovala aspoň určitá štruktúra extrahovaných článkov, a aby bolo možné následne články vyhľadávať, či porovnávať. Ak používateľ spustí extrahovanie dát iba jednorázovo, frekvencia zdroja ostáva nastavená na hodnote 0 a scraper stiahne dáta z daného zdroja iba raz. Ak je hodnota frekvencie vyššia ako 0, spustí sa automatický plánovač. Hodnota frekvencie označuje čas v ktorom má web scraper extrahovať dáta z definovaného zdroja. Frekvencia spoločne s vytvorenou šablónou sa aktualizujú v databáze pre konkrétny zdroj. Nastavenie a aktualizovanie údajov o extrahovaní sa vykonáva len pre ten konkrétny zdroj a nemenia sa tým parametre ostatných zdrojov. Scraper je nastavený tak, že každý uložený zdroj má definované vlastné pravidlá extrakcie, či už sa jedná o definovanú šablónu sťahovania, alebo frekvenciu. Scraper si udržuje zoznam aktívnych zdrojov, teda zdrojov pre ktoré bola spustená plánovaná extrakcia dát. Z každého zdroja sú následne extrahované dáta podľa jeho konkrétnych špecifikácií. Scraper vykonáva kontrolu zadaných informácií pre nastavenie zdroja iba pri registrácii novej požiadavky. Pri opakovanom spúšťaní sa kontrola nevykonáva až do momentu, kedy príde nová požiadavka na aktualizovanie údajov pre daný zdroj. Používateľ taktiež definuje pre každý zdroj individuálne, či chce použiť Puppeteer alebo Axios na získanie HTML dát z článkov. Na plánovanie akcií využívam modul *Node Cron* a knižnicu *cron-parser*.

Modul *Node Cron* umožňuje jednoduché plánovanie automatických akcií. *Cron* umožňuje nastavovať rozličné časové hodnoty, počnúc od sekúnd až po nastavenie plánovača v mesiacoch. Pre implementáciu v tejto práci som sa rozhodol obmedziť rozsah zadávaných hodnôt. Najnižšiu povolenú hodnotu som nastavil na 3 minúty a najvyššiu časovú hodnotu som nastavil na 24 hodín. Spodnú hranicu som nastavil z toho dôvodu, aby scraper bol schopný dokončiť extrahovanie dát z daného zdroja predtým, ako ich začne extrahovať znova, teda aby čas samotnej extrakcie nebol dlhší ako časová medzera medzi opakovanými spusteniami. Pri implementácií som sa odklonil od prvotného návrhu nastavovania frekvencie iba v minútach a prispôsobil som fungovaniu a

vlastnostiam modulu *Node Cron*. Plánovanie automatického spúšťania extrakcie pre konkrétny zdroj je možné definovať vo viacerých variantoch. Prvá možnosť je opakovanie extrakcie každú *x-tú* minútu. Pri tejto možnosti sa extrakcia spustí každú *x-tú*. minútu v hodine, teda napríklad pri nastavení frekvencie na hodnotu 5, scraper spustí extrahovanie dát pre daný zdroj každú 5. minútu. Druhým variantom je naplánovanie opakovania v hodinách. Pri tomto variante sa podobne ako pri prvej možnosti, nastavuje frekvencia radovou číslovkou, v tomto prípade pre hodiny. V tomto variante sa okrem hodín nastavujú aj minúty, tie však definujú presný čas v ktorý sa extrakcia dát spustí. Napríklad pri použití hodnoty 2 pre hodiny a hodnoty 15 pre minúty, automatické extrahovania sa bude vykonávať každú 2. hodinu a 15. minútu. Posledným variantom je nastavenie presného času extrakcie dát. Pri tejto metóde sa extrakcia vykoná raz, za 24 hodín v presne definovaný čas, ktorý si volí používateľ. Na obrázku 4.4 sú zobrazené všetky 3 varianty nastavovania frekvencie spoločne aj s príklad nasledujúcich extrakcií.


**Obr. 4.4** Zobrazenie jednotlivých variantov zadávania frekvencie spoločne s ukážkou spustenia nasledujúcich extrakcií

Pri definovaní frekvencie využívam knižnicu *cron-parser*. Táto knižnica dokáže spracovať a manipulovať s inštrukciami *Cron* plánovača. Túto knižnicu využívam pri náhľade nastavenej frekvencie, teda ešte predtým ako sa odosiela požiadavka na spustenie automatickej extrakcie. Pre realizáciu tohto náhľadu som vytvoril samostatné API. Pri každej zmene, ktorú používateľ vykoná pri nastavovaní frekvencie, sa odošle nová požiadavka na toto API s aktuálnou časovou hodnotou v forme *Cron* záznamu. Táto časová hodnota sa spracuje a vytvorí sa 8 po sebe nasledujúcich reálnych časov, ktoré zobrazujú ako a kedy by sa extrakcia spúšťala, ak by používateľ spustil extrahovanie s touto frekvenciou.

Web scraper po spustení na konkrétny zdroj vyhľadáva a sťahuje údaje len z tých článkov, ktoré sa nachádzajú v príslušnom RSS kanáli daného zdroja. Web scraper získa všetky prvky z RSS kanálu, nie len ten prvý. Na informovanie používateľa som použil vstavaný Node.js mudul „Events“, ktorý slúži na implementáciu vlastných eventov. Po začatí iterácie cez jednotlivé záznamy z RSS kanála, scraper odošle event na frontend s informáciou o začatí extrahovania dát. Pre každý záznam z RSS kanálu sa ako prvé získajú práve informácie, ktoré sú obsiahnuté v tomto kanáli o danom článku. V RSS kanáli sa vyskytuje pre každý záznam o článku jeho originálna URL adresa, teda adresa pod ktorou je publikovaný na spravodajskom portále v plnej podobe. Po získaní tejto URL adresy sú z originálneho článku získané openGraph sémantické dáta. Scraper postupne prechádza vytvorenú šablónu pre extrahovanie a dopĺňa získané dáta. Ak sa v šablóne vyskytuje CSS selektor, vyhľadá podľa neho príslušné dáta z článku. Novo vytvorený

záznam je prekontrolovaný. Na základe jedinečného identifikátora článku sa systém pokúsi získať záznam z databázy aktuálne uložených článkov. Ak sa záznam s takýmto identifikačným kľúčom v databáze nenachádza, znamená to, že sa jedná o nový článok o ktorom ešte nemáme uložené dáta. Ak sa ale takýto článok v databáze už nachádza, je porovnaný tento uložený záznam s novo vytvoreným. Porovnávajú sa všetky hodnoty ktoré jednotlivé záznamy obsahujú. Ak sú tieto hodnoty v oboch článkoch rovnaké, nový článok nenahrádza už uložený záznam, nakoľko sa jedná o tie isté dáta, ktoré už sú uložené. Keď sa dáta nezhodujú, znamená to, že sa v článku zmenili publikované informácie, alebo sa zmenila štruktúra extrahovania dát. Nový záznam teda nahradzuje existujúci záznam v databáze aktuálnych článkov. Predošlý záznam sa z tejto kolekcie odstráni a uloží sa do kolekcie obsahujúcej verzie jednotlivých článkov. Pri uložení do kolekcie verzií sa do tohto záznamu pridá aj časová stopa. Časová stopa reprezentuje čas extrakcie, teda kedy scraper spustil extrakciu pre zadaný zdroj. Táto časová stopa reprezentuje moment, kedy sa z aktuálneho článku stala verzia článku. Po prejdení všetkých prvkov v RSS kanáli sa fáza extrakcie ukončí a odošle sa na nový event. Informácie ktoré event odošle sa odvíjajú od toho ako bol scraper spustený, či sa jedná o naplánovanú akciu alebo to bolo jednorázové spustenie. Pri naplánovanom automatickom spúšťaní odošle event informáciu o tom, že scraper dokončil extrakciu dát a je v stave čakania. Ak bol scraper spustený jednorázovo, event odošle informáciu o ukončení extrakcie a zastavení scrapera.

Posledným krokom je uloženie informácií o aktuálne ukončenej extrakcii. Tieto informácie zahŕňajú dátum a čas kedy bola extrakcia spustená pre definovaný zdroj, zoznam jedinečných identifikátorov článkov, ktoré boli počas tejto extrakcie pridané do databázy aktuálnych článkov ako novo vytvorené, teda ešte o nich neboli uložené žiadne údaje. Súčasne sa pridá aj záznam jedinečných identifikátorov článkov, ktoré boli počas tejto extrakcie upravené. Tieto údaje sa uložia do kolekcie *Last Runs*.

Pri extrahovaní dát som narazil na *HTTP Error 429* ktorý sa vyskytol napríklad pri extrakcii dát zo spravodajského portálu Deník.cz. Error 429 znamená, že na daný server bolo odoslaných príliš veľa požiadaviek v krátkej dobe a požadovaný server už neprijíma ďalšie požiadavky z danej IP adresy. Pre vyriešenie tohto problému som implementoval možnosť definovania oneskorenia odosielania požiadaviek. Toto oneskorenie nastavuje používateľ na takú hodnotu, akú uzná za vhodnú. Oneskorenie je implementované v prechádzaní cez záznamy v RSS kanáli. Pre každý záznam sa počká s odoslaním akýchkoľvek požiadaviek na servery spravodajských portálov. Toto oneskorenie sa aplikuje len na konkrétny zdroj, pri ktorom bolo nastavené.

Pre zastavenie a zrušenie automatického spúšťania extrakcie dát využívam rovnaké API ako pri jeho spúšťaní. Pri požiadavke na zastavenie extrakcie sa odstráni záznam zo zoznamu aktuálne spustených extrakcií a vymaže sa aj plánovač. Ak požiadavka prišla práve v momente extrakcie dát, je najprv táto extrakcia dokončená, teda spracované sú všetky záznamy z RSS kanála. Po dokončení extrakcie sa už v zozname aktívnych extrakcií nenachádza záznam o danom zdroji a extrahovanie sa opätovne nespustí, pokiaľ nepríde nová požiadavka o jeho zaktívnenie.

#### 4.1.3.5 Filtrovanie, zobrazenie verzií, zobrazenie vykonaných extrakcií

Pre filtrovanie medzi uloženými článkami som vytvoril samostatné API na ktorom očakávam požiadavku s informáciami o nastavení filtrácie. Tieto informácie využívam na vyhľadanie vyhovujúcich záznamov z kolekcie aktuálnych článkov. Používateľ si môže zvoliť dátum a k tomu došpecifikovať presný čas v inej premennej, či už pre spodnú hranicu alebo pre hornú hranicu. Nastavil som transformáciu týchto údajov do jedného prvku pre vrchnú a jedného prvku pre spodnú hranicu, na základe ktorých je možné v databáze vyhľadávať záznamy. Ak je zvolený iba dátum a nie je zvolený presný čas, dotaz sa vytvorí tak, ako keby bol čas definovaný na 00:00 pre spodnú hranicu a 23:59 pre vrchnú hranicu. Oproti používateľskému návrhu som do filtra pridal aj možnosť vyhľadávania na základe slova. MongoDB umožňuje vyhľadávanie na základe slova alebo slovného spojenia či viacerých slov. Výhodou tohto vyhľadávania je, že MongoDB automaticky vyhľadá zadané slovo aj v jeho iných podobách, napríklad pri skloňovaní alebo

časovaní a nie len v tej podobe ako bolo definované v dotaze. Pri vyhľadávaní záznamov podľa slova je potrebné na backende indexovať kľúče, v ktorých má MongoDB toto slovo vyhľadať. Pre indexovanie som určil záznamy týkajúce sa titulku (title) a popisu (description), pretože tie som určil ako povinné. Na základe vytvoreného dotazu z poskytnutých údajov vykonám vyhľadávanie v databáze aktuálnych článkov. Výsledný zoznam vyfiltrovaných článkov je následne odoslaný ako odpoveď na požiadavku vo formáte JSON objektu.

Každý článok ktorý je vyfiltrovaný by mal obsahovať možnosť nahliadnutia jeho predošlých verzií. Na implementovanie tohto vyhľadávania som vytvoril samostatné API. Na základe poskytnutého identifikátoru článku ktoré je súčasťou požiadavky viem vyhľadať v databázovej kolekcii obsahujúcej verzie článkov tie správne záznamy. Aby som zabezpečil možnosť porovnania rozdielov medzi aktuálnym článkom a jeho verziami, získam obsah článku aj z kolekcie aktuálnych článkov. Záznam aktuálneho článku a zoznam jeho získaných verzií odosielam vo forme JSON objektu ako odpovedi na požiadavku.

V aplikácií som implementoval aj možnosť nahliadnutia nad jednotlivé extrakcie ktoré boli vykonané pre daný zdroj. Tieto nahliadnutia nad vykonané extrakcie v sebe obsahujú informácie o tom kedy sa extrakcia spustila, koľko nových záznamov bolo pridaných a koľko záznamov bolo počas tejto extrakcie upravených. Na vykonanie tejto funkcionality som opäť vytvoril samostatné API, ktoré očakáva tieto presné informácie extrakcie pre špecifický zdroj. Prostredníctvom týchto informácií sa vytvoria dotazy na získanie záznamov ktoré boli počas tejto extrakcie pridané ako nové alebo ako upravené.

Prvý dotaz využije zoznam identifikátorov pre novo pridané záznamy. Prostredníctvom tohto zoznamu sa vyhľadajú záznamy o článkoch, ktoré boli počas tejto extrakcie pridané. Druhý dotaz smeruje na databázovú kolekciu ktorá obsahuje verzie článkov. Na vyhľadanie záznamov sa opäť využije zoznam identifikátorov článkov, však nie tých ktoré boli pridané ako nové, ale tých, ktoré boli počas tejto extrakcie upravené. Pri druhom dotaze sa využíva aj čas extrakcie, aby som zabezpečil, že sa získajú len tie verzie článkov, ktoré boli počas tejto extrakcie upravené. Zobrazením tohto náhľadu informujem užívateľa o tom, aké články boli počas konkrétnej extrakcie získané a upravené.

## 4.2 Frontend

Frontend aplikácie som implementoval v JavaScriptovom frameworku React. Pre inicializovanie projektu som využil natívny príkaz *npx create-react-app <názov_projektu>* ktorý React využíva pre tvorbu základnej štruktúry projektu. Tento príkaz vytvorí okrem základnej štruktúry už aj samotné JavaScriptové a CSS súbory vzorovej aplikácie. Po inicializovaní projektu je možné vzorovú aplikáciu spustiť ihneď, bez potreby dodatočného konfigurovanie a to príkazom *npm start*.

### 4.2.1 Štruktúra frontendu aplikácie a použité knižnice

Základnú štruktúru projektu ktorú vytvoril React automaticky som rozšíril o dodatočné adresáre. React podobne ako Node.js nevyžaduje aby bola takáto štruktúra dodržiavaná, avšak sprehľadní to orientáciu v súborovom systéme. Na obrázku 4.5 je znázornená použitá štruktúra, avšak sú z nej vynechané jednotlivé koncové súbory, zachovaný je len adresárový systém a hlavný súbor aplikácie.

- **components**: V tomto priečinku sa nachádzajú komponenty, ktoré sú využívané pri tvorbe webovej aplikácie,
- **css**: Tento priečinok obsahuje všetky CSS súbory ktoré definujú vzhľad webovej aplikácie,
- **images**: Tento priečinok obsahuje obrázky použité vo webovej aplikácií,


**Obr. 4.5** Štruktúra frontendovej časti aplikácie

- **pages**: V tomto priečinku sa nachádzajú súbory ktoré definujú jednotlivé stránky webovej aplikácie,
- **index.js**: Je hlavný súbor, na základe ktorého sa aplikácia spúšťa.

#### 4.2.1.1 Pridávanie, odoberanie a zobrazovanie zdrojov

Pre pridávanie nového zdroja som použil textové komponenty <TextField>, ktoré sú z knižnice MUI, spoločne s tlačidlo na odoslanie požiadavky. Pre odoslanie vyplnených dát na backend využívam knižnicu Axios. Odpoveď od backendového servera sa vyhodnotí a vypíše sa správa ktorá buď informuje o tom, že zdroj bol pridaný, alebo sa zobrazí upozornenie, ktoré poukazuje na neplatné zadané údaje.

Požiadavka na získanie dát o aktuálnych zdrojoch je odosielaná na server pri každom načítaní domovskej obrazovky. Získané dáta sú zapísané vo formáte JSON. Štruktúra komponentu v akej sú tieto dáta zobrazené používateľovi sa od používateľského návrhu nezmenila, avšak pre prehľadnejšie zobrazenie som implementoval rozličné varianty tohto komponentu na základe stavu jednotlivých zdrojov, ktoré môže nadobúdať.
Zdroje môžu byť v troch stavoch:

- práve aktívny,
- neaktívny-zastavený,
- naplánovaný/aktuálne neaktívny.

Stav *práve aktívny* informuje používateľa o tom, že scraper aktuálne vykonáva extrakciu dát pre ten konkrétny zdroj. Stav *neaktívny-zastavený* informuje používateľa o tom, že extrahovanie dát pre daný zdroj je zastavené a nie je naplánované automatické spúšťanie. Stav *naplánovaný/aktuálne neaktívny* informuje používateľa o tom, že extrakcia dát bola spustená spoločne s naplánovaním opakovaných extrakcií, avšak aktuálne čaká na automatické spustenie, ktoré je podmienené nastavenou frekvenciou.

Pre vymazanie daného zdroja som do komponentu vložil tlačidlo, ktoré odošle požiadavku pre odstránenie na backendový server. Ak od servera príde odpoveď o úspešnom vymazaní záznamu, opäť sa na server pošle nová požiadavka na získanie dát o aktuálnych zdrojoch, čím sa aktualizuje zobrazený zoznam.

#### 4.2.1.2 Nastavenie scrapera a náhľad extrakcie

Pre získanie aktuálnych nastavení scrapera pre daný zdroj som nastavil automatické odoslanie požiadavky na backend pri každom znovu načítaní podstránky. V tele požiadavky je zahrnutý iba názov zdroja ktorý budeme chcieť nastavovať. Odpoveď od backendu obsahuje viacero informácií, ktoré rozdeľujem a ukladám do rôznych stavových premenných. V odpovedi sa nachádzajú RSS

a sémantické dáta pre daný článok, ktoré následne ukladám do samostatných stavových premenných, aby bolo možné zobraziť ich náhľad. Taktiež sa v odpovedi od backendu nachádzajú informácie o tom v akom stave sa zdroj práve nachádza, a aké nastavenia extrakcie boli uložené naposledy. Tieto získané nastavenia extrakcie sa automaticky aplikujú na jednotlivé komponenty. Tým zabezpečujem informovanie používateľa o jeho predošlých nastaveniach pre ten daný zdroj. Ak boli nastavenia zvolené z preddefinovaných šablón, táto šablóna sa automaticky aktivuje a zobrazí príslušné dáta. Ak boli použité vlastné nastavenia extrakcie, jednotlivé prvky sa automaticky vyplnia a zobrazia v príslušných komponentoch a vytvorí sa nový náhľad.

Výber z preddefinovaných nastavení som zabezpečil pomocou tlačidiel. Pri zvolení si šablóny sa odošle nová požiadavka na server aj s príslušnou konfiguráciou šablóny, čím sa šablóna zaktívni a zobrazia sa požadované dáta. Šablóny nie je možné medzi sebou kombinovať, a teda aktívna môže byť vždy len jedna.

```
const handleChange = (key) => {
  setChecked(prevState => ({
  onlyRSS: key === "onlyRSS" ? !prevState.onlyRSS : false,
  onlySemantics: key === "onlySemantics" ? !prevState.onlySemantics : false,
  rssAndSemantics: key === "rssAndSemantics" ? !prevState.rssAndSemantics : false
  }));
  setActiveNow(prevState => {
    switch (key) {
     case "onlyRSS":
      return prevState === "onlyRSS" ? "noData" : "onlyRSS";
     case "onlySemantics":
      return prevState === "onlySemantics" ? "noData" : "onlySemantics";
     case "rssAndSemantics":
      return prevState === "rssAndSemantics" ? "noData" : "rssAndSemantics";
     default:
        return "";
    }
  });
};
```

**Obr. 4.6** Funkcia ktorá aktivuje vybranú šablónu

Pre definovanie vlastných nastavení extrakcie 3 povinných prvkov využívam komponent <*Textfield*> z knižnice MUI. V tomto komponente som nastavil automatické vypĺňanie údajov, ktoré sa vykoná na základe dát získaných od backend servera po načítaní stránky. Tieto údaje som implementoval cez parameter *value* komponenty <*Textfield*>, čo mi umožnilo manipulovať s týmito údajmi v rámci komponentu. Pre zabezpečenie plynulosti vyplňovania údajov sú dáta ukladané do stavovej premennej z ktorej sú následne načítané do parametra *value*. Tento prístup som zvolil z toho dôvodu, aby bolo možné automaticky vložiť uložené nastavenia, ale zároveň aby sa s týmito údajmi dalo ďalej pracovať a upravovať ich. Odoslanie požiadavky o zmene nastavení extrakcie na server sa nevykoná hneď po akejkoľvek zmene hodnôt, ale nastavil som zdržanie na 1 sekundu. Toto zdržanie zabezpečuje, že pri vyplňovaní hodnôt v komponente sa požiadavka odošle s kompletnejšími údajmi selektora. Bez toho zdržania by sa požiadavky odosielali pri každej zmene, teda už pri pridaní alebo zmazaní jedného písmena, čo je neefektívne pri písaní vlastných selektorov, pretože napríklad pri určovaní selektora *h1*, by sa požiadavky poslali zbytočne dve, prvá s hodnotou selektora iba *h* a druhá s hodnotou *h1*.

Pri zadaní syntakticky nesprávneho selektora sa do komponentu *TextField* pridá jeho vlastnosť *error*, ktorá graficky upraví komponent, pre vyobrazenie chybného zápisu. Pre zamedzenie nesprávnych výsledkov extrakcie som znemožnil spustenie scrapera pri nesprávne zadanom selektore.

Pre určovanie nepovinných parametrov využívam <textarea>. V tomto textovom poli je potrebné definovať nie len selektor samotný ale aj jeho kľuč, pod ktorým sa tieto hodnoty budú ukladať. V tomto textovom poli odchytávam vyplnené hodnoty na základe regulárneho výrazu. Pokiaľ zadané informácie nebudú správne zapísané, nebudú považované za validný vstup. Vo funkcii ktorá tieto dáta zapisuje som rovnako použil 1 sekundové zdržanie pri odosielaní požiadavky

```
const handleTextArea = (event) ⇒ {
  const value = event.target.value;
  const regex = /(?:"([^"]+)"|([\w:]+)):\s*([^\s;]+(?:\s+[^\s;]+)*)/g;
  let match;
  let jsonData = {};
  if (value !== '') {
    while ((match = regex.exec(value)) !== null) {
      if (match[2] && match[3].trim() !== '') {
        const name = match[1] || match[2];
        const val = match[3].trim();
        jsonData[name] = { source: val };  }  }
  }
  clearTimeout(timeout);
  timeout = setTimeout(() ⇒ {
    const additionalDataValue = Object.keys(jsonData).length === 0 ? '' : JSON.stringify(jsonData, null, 2);
    setAdditionalData(additionalDataValue);
    setChecked({
      onlyRSS: false,
      onlySemantics: false,
      rssAndSemantics: false
    });
    setActiveNow('clientConfig');
  }, 1000);
};
```

**Obr. 4.7** Funkcia ktorá kontroluje či sú správne zapísané selektory pre nepovinné parametre

na server.

Náhľad nad nastaveniami extrakcie dát zobrazuje reálne informácie ktoré sú obsiahnuté či už v RSS kanáli alebo priamo v článku na základe definovaných selektorov alebo použitých šablón. Náhľad som vypracoval v dvoch formátoch. Prvý formát zobrazí získané dáta v jednoduchšej key-value podobe. Druhý formát je skonštruovaný, tak aby pripomínal zverejnený článok na spravodajskom portáli. Zobrazované údaje sú aktualizované za každým, keď sa vykoná zmena v nastaveniach extrakcie. Ak sa pri definovaní selektora v náhľade nezobrazia požadované dáta, znamená to, že scraper nebol schopný získať dáta z takto definovaného selektora a pri jeho spustení sa táto prázdna hodnota zapíše v každom spracovanom prvku z RSS záznamu.

Náhľad RSS dát, sémantických dát alebo originálneho článku je možné zobraziť si z bočného menu. V tomto menu je zahrnutý aj náhľad nad nastavenia extrakcie, takže je možné jednoducho sa preklikávať medzi jednotlivými náhľadmi. Forma zobrazenia RSS a sémantických dát je podobná JSON objektu. Na zobrazenie náhľadu na originálny článok využívam HTML tag <iframe>. Vďaka tomu je možné získavať informácie o štruktúre HTML článku priamo z mojej aplikácie, pretože pri zobrazení si DevTools je možné získať jednotlivé HTML prvky článku priamo z <iframe>.

#### 4.2.1.3 Plánovanie extrakcie dát

Spustiť extrahovanie pre daný zdroj je možné dvoma spôsobmi. Prvý spôsob je jednorázové spustenie, kedy sa odošle požiadavka s nulovou frekvenciou a scraper spustí extrahovanie pre daný RSS záznam iba raz. Druhým spôsobom je naplánovanie automatického spúšťania. Od prvotného návrhu som upustil a vytvoril som komponent pre prehľadnejšie určovanie hodnôt.

Prvým variantom je voľba opakovania v minútach. V tejto časti používateľ zadáva len minútové hodnoty cez jeden prvok. Zadávané hodnoty som obmedzil na rozsah od 3 do 59. Druhý variant obsahuje možnosť nastavenia presného času, teda opakované spúšťanie raz za 24 hodín, alebo možnosť nastaviť opakovanie v hodinách. Pri opakovaní v hodinách používateľ nastavuje hodnoty hodiny a minúty skrz samostatné parametre. V komponente som implementoval aj náhľad nad časy extrakcie. Pri každej zmene ktorá je vykonaná, sa spracuje vstup od používateľa, tento vstup sa pretransformuje do záznamu ktorý využíva *Node Cron*, a odošle sa požiadavka na server. Odpoveď obsahuje ukážkové časy, ktoré sú následne zobrazené v komponente. Ukážkové

časy informujú používateľa o tom, že ak by spustil extrakciu s takto nastavenou frekvenciou, v akých časoch by sa extrakcia automaticky spúšťala. Na monitorovanie aktuálneho stavu scrapera používam GET požiadavku na backendové API pomocou EventSource. EventSource umožňuje prácu s SSE (Server-Sent Events), ktoré umožňujú serveru posielať eventy klientovi. Pri vytvorení objektu EventSource sa nadviaže HTTP spojenie so serverom. EventSource poskytuje API *open*, *message* a *error* pre spracovanie udalostí. S využitím API *message* potom rozlišujem jednotlivé eventy, na základe ktorých zobrazujem správy v UI o stave scraperu.

#### 4.2.1.4 Zobrazenie vykonaných extrakcií

Pre zobrazenie vykonaných extrakcií scrapera pre daný zdroj som implementoval jednoduchý <*input*> pre číselné hodnoty, ktoré reprezentujú počet záznamov, ktoré sa majú vyhľadať v databáze po odoslaní požiadavky. Dáta získané zo servera potom zobrazujem v rovnakej štruktúre ako som definoval v návrhu používateľského rozhrania. Bližší náhľad ukončenej extrakcie sa dá zobraziť prislúchajúcim tlačidlom. V bližšom náhľade sú zobrazené jednotlivé články, či už tie ktoré boli počas extrahovania získané ako nové články, alebo tie ktoré boli počas extrakcie upravené. Obsah akéhokoľvek článku je možné stiahnuť vo formáte JSON, CSV alebo XML, rovnako tak aj celkový zoznam novo pridaných článkov, alebo upravených článkov. Na sťahovanie záznamov v definovaných formátoch využívam knižnicu Export From JSON. Táto knižnica umožňuje jednoduchý spôsob transformácie a sťahovania dát z JSON objektov. Okrem možnosti sťahovania dát v spomenutých formátoch podporuje aj formáty ako CSS, HTML, TXT alebo XLS.

#### 4.2.1.5 Filtrovanie článkov a ich verzie

Pri tvorbe filtra som využil komponenty *DatePicker* a *TimePicker* z knižnice MUI. Tieto komponenty som vybral, pretože obsahujú prehľadné grafické spracovanie vyberania času a dátumu. Určovanie času som podmienil tým, že je potrebné si najprv zvoliť presný dátum, až potom sa sprístupní možnosť výberu času. Toto obmedzenie som použil, pretože ak je filter prázdny a odošle sa požiadavka, server vráti všetky záznamy z databázy, a ak by bol definovaný iba čas, tak by to bolo považované ako neplatný vstup. Ďalšími prvkami ktoré sa môžu definovať je samotný identifikátor článku, identifikátor zdroja a kľúčové slovo.

Zobrazenie získaných dát som obmedzil na 10 záznamov s možnosťou stránkovania, aby bolo prehľadné sa v tom pohybovať. Záznamy je možné zoradiť podľa ich dátumu vytvorenia (pubdate) od najnovších, alebo najstarších. Taktiež som pridal možnosť si stiahnuť výsledky filtrovania vo formátoch JSON, CSV a XML, čo platí aj pre články samotné.

V komponente zobrazujúcom náhľad na stiahnutý článok som podľa návrhu používateľského rozhrania implementoval tlačidlá ako odkaz na originálny článok, zobrazenie verzií a zobrazenie kompletných údajov o danom článku.

Pri zobrazení verzií využívam rozšírenie Diff, ktoré umožňuje rozlišovanie medzi slovami v JavaScripte. Diff ponúka niekoľko API na prácu s textom a v tejto implementácií som využil API diffWords. DiffWords má dva vstupné parametre vo forme textových blokov. DiffWords porovnáva tieto dva bloky textu tak, že každé slovo alebo každé interpunkčné znamienko sa považuje za symbol. S využitím tohto API zobrazujem rozdiely medzi aktuálnym záznamom a jeho verziami. Pri zobrazení verzie graficky znázorním rozdiely tak, že prvky podfarbené červenou farbou sú prvky ktoré sú vo verzií navyše od aktuálneho záznamu, a prvky ktoré sú podfarbené na zelenou farbou sú prvky, ktoré vo verzii chýbajú, teda nenachádzajú sa vo verzii oproti aktuálnemu záznamu.

# Kapitola 5
# Testovanie

Webová aplikácia bola otestovaná pomocou používateľského testovania. V tejto časti popisujem vytvorený zoznam úloh, položené otázky a taktiež aj informácie ktoré som počas testovania získal. Aplikáciu som následne publikoval na GitHub ako verejný projekt.

## 5.1 Používateľské testovanie

Používateľské testovanie je proces počas ktorého overujeme používateľnosť, jednoduchosť a intuitívnosť aplikácie z hľadiska koncového používateľa. Používateľské testovanie nám ukazuje to, ako s aplikáciou pracujú používatelia, s čím majú problémy, čo je pre nich nezrozumiteľné alebo naopak, čo sa im páči a čo pokladajú za vhodné. Počas tohto testovania sa môžu ukázať chyby a nedostatky aplikácie, ktoré by inak mohli byť prehliadnuté a prišlo by sa nich až po nasadení aplikácie do prevádzky. Medzi takéto nedostatky môže patriť neprehľadné prostredie, ťažká orientácia v rámci aplikácie alebo systémové chyby.

Používatelia netolerujú zlý používateľský dizajn. To, či o aplikáciu alebo webovú stránku majú záujem, sa ukáže krátko po tom, ako si ju prvýkrát otvoria. Ak sa webová stránka z nejakého dôvodu už na prvý pohľad používateľom nepozdáva, nemajú záujem s ňou pracovať a hľadajú inú alternatívu bez toho, aby vôbec zistili čo všetko stránka alebo aplikácia ponúka.

Používateľke testovanie je možné vykonať rôznymi spôsobmi, napríklad ako priame testovanie alebo vzdialené testovanie. Priame testovanie spočíva v tom, že sa aj testovaný ako aj zadávateľ testu nachádzajú na rovnakom mieste. Či už ide o kancelárske priestory alebo špecificky pripravenú miestnosť na testovanie. Výhodou priameho testovania je možnosť zadávateľa testu pozorovať priamo reakcie testovaného na zadané úlohy, pýtať sa otázky počas testu alebo odpovedať na jeho otázky. Avšak tento druh testovania je časovo a častokrát aj finančne náročnejší. Taktiež sa testovaní môžu cítiť pod tlakom, nakoľko sú pozorovaní pri vykonávaní jednotlivých úloh. Vzdialené testovanie je forma testovania kedy sa zadávateľ testu a testovaný nenachádzajú pri sebe. Toto testovanie je možné vykonať aj pomocou video hovorov, je tiež nazývané ako moderované, teda testovanie kde zadávateľ testu môže sledovať testovaného prostredníctvom video hovoru. Druhou metódou je nemoderované online testovanie, čo je druh testovania, počas ktorého nevzniká žiadna interakcia testovaného a zadávateľa testu, a výsledky testovania sú zaznamenávané pomocou videa alebo podľa odpovedania na dotazník.

Pre používateľské testovanie je dobré pripraviť scenár, ktorým sa testovaný uvedie do deja. Taktiež je potrebné si pripraviť úlohy, ktoré má testovaný splniť. Úlohy by nemali byť až príliš konkrétne aby používateľa priamo nenavádzali na vykonanie úlohy, ale štruktúrované, tak aby používateľ získal len okrajovú znalosť toho čo sa od neho očakáva, a interagoval s aplikáciou spontánne[40].

Pre testovanie vytvorenej webovej aplikácie som zvolil online formu cez videohovor s doplnkovým dotazníkom. Dotazník bol vytvorený cez Google Forms. Testovania sa zúčastnilo 7 respondentov. Nakoľko aplikácia bude dostupná vo forme zdrojového kódu prostredníctvom GitHubu, pre testovanie som poskytol všetkým respondentom prístup ku Git repozitáru na stiahnutie si kódu. Každý respondent bol vopred oboznámený s požiadavkami na beh aplikácie. Všetkých 7 respondentov súhlasilo s nainštalovaním potrebných technológií pre beh aplikácie. Do testovania nebolo zahrnuté nastavenie a inštalovanie potrebných technológií. Ani jeden z respondentov s aplikáciu pred začatím testovania nepracoval, ani ju nevidel.

### 5.1.1 Úlohy

Pre testovanie som vytvoril niekoľko úloh ktoré testovaní museli splniť. Jednotlivé úlohy na seba nadväzovali, takže bolo potrebné aby ich používateľ vykonával postupne. O nadväznosti úloh boli oboznámení aj testovaní. Úlohy boli skonštruované tak, aby testovali všetky funkčné požiadavky webovej aplikácia a intuitívnosť používateľského prostredia pri ich vykonávaní.

#### 5.1.1.1 Nové zdroje

Web scraper používate prvýkrát a nemáte v ňom žiadne zdroje. Zaujímajú Vás články z Prima CNN a Blesk.cz. Pridajte si zdroje pre tieto spravodajské portály.

#### 5.1.1.2 Nastavte scraper pre prvý zdroj

Nastavte scraper pre zdroj Blesk.cz. Zaujímajú Vás RSS a sémantické dáta, zobrazte a prezrite si ich. Nastavte scraper tak, aby ste používali iba RSS dáta, následne spustite extrakciu pre Blesk.cz jednorázovo. Zistili ste, že Vám RSS dáta nestačia a chcete to zmeniť a stiahnuť aj sémantické. Nastavte a jednorázovo spustite extrahovania iba sémantických dát pre Blesk.cz.

#### 5.1.1.3 Nastavte scraper pre druhý zdroj

Nastavte extrahovanie pre zdroj Prima CNN. Pre spravodajský portál Prima CNN zadajte konkrétne hodnoty, ktoré chcete získať z daných článkov. Tieto hodnoty sú:

- Nadpis chcete extrahovať priamo z článku, takže potrebuje nájsť jeho HTML označenie.
- Pre link ste sa rozhodli využiť ten, ktorý ste získali z RSS dát.
- Popis a obrázok článku ste sa rozhodli použiť zo sémantických dát.
- Z článku si vyberte akúkoľvek časť, ktorú by ste chceli sťahovať, a nastavte scraper tak, aby ju sťahoval a ukladal pod názvom „values“.

Scraper si chcete naplánovať a spustiť tak, aby sa opakovane vykonával každú 5. minútu.

#### 5.1.1.4 Zobrazte si dáta z poslednej extrakcie

Zistite, z akých článkov sa Vám dáta extrahovali pri poslednej extrakcii pre zdroj Prima CNN.

#### 5.1.1.5 Zobrazte si uložené záznamy

Prezrite si, čo všetko sa Vám stiahlo. Zobrazte iba tie záznamy článkov, ktoré sa extrahovali z portálu Blesk.cz. Chcete zistiť ktorý z nich je najnovší a zaujíma Vás, či máte aj jeho verzie.

### 5.1.2 Priebeh testovanie

Testovanie prebiehalo vo forme videohovoru, kedy testovaní dostali zoznam úloh ktoré mali vykonať. Pre začatím testovania som všetkým testovaným priblížil základné informácie o aplikácií tak, aby som tým neovplyvnil priebeh testovania. Počas testovania som pozoroval jednotlivých testovaných, zaznamenával si poznámky o práci a priebehu testovania. Po ukončení testovania som poprosil každého testovaného, aby vyplnil pripravený dotazník a poprípade diskutovali svoje postupy, nápady a pripomienky v širšom rozsahu. Testovaní mali rôzne znalosti v informatike a nepochádzali z rovnakého odvetvia, čo sa prejavilo aj počas testovania pri vykonávaní jednotlivých úloh. Testovaným som pred začatím testu naznačil, že aplikácia obsahuje aj dokumentáciu s ktorou si môžu pomôcť.

#### 5.1.2.1 Poznatky z testovania

V tejto časti priblížim poznatky a poznámky, ktoré som si zapisoval počas priebehu testovania pri jednotlivých úlohách.

V prvej úlohe, teda pri pridávaní zdrojov nemali testovaní vážne problémy. Všetci túto úlohu dokázali urobiť v pomerne krátkom čase, aj keď až 4 testovaní zadali prvotne zlú URL adresu, teda nedefinovali správne RSS URL. Dvaja z nich sa rýchlo opravili, avšak ostatní dvaja testovaní mi povedali, že si nie sú istý, či vedia čo je to RSS adresa. Obom som pripomenul, že aplikácia obsahuje aj dokumentáciu. Obaja potom vyhľadali a zadali správne adresy. Už pri tejto časti som si začal všímať drobné nedostatky v aplikácií. Testovaní mali tendenciu pracovať rýchlo, a teda začali vykonávať úlohu ešte predtým, ako si ju kompletne dočítali, alebo priamo prechádzali na interaktívne prvky, ako vyplňovanie údajov. Tu som si všimol, že informácie ktoré boli zobrazené pri pridávaní nového zdroja, si používatelia prečítali až potom, ako zadali zlú RSS adresu.

V druhej úlohe pri ktorej mali testovaní použiť predpripravené šablóny a spustiť extrahovanie jednorázovo bolo vidieť, že testovaným chvíľku trvalo, kým sa na podstránke nastavenia zdroja zorientovali. Akonáhle prišli na fungovanie jednotlivých častí, nastavenie šablóny už nebol ani pre jedného z nich problém. Avšak problém sa objavil pri spustení extrakcie. Tento problém spočíval v tom, že po spustení extrakcie nebolo dostatočné jasné, že sa niečo stalo, teda v podstránke s nastavením zdroja chýbalo jasné zobrazenie toho, že extrahovanie bolo zahájené. Taktiež si testovaní mýlili používanie šablón a zobrazovanie náhľadov.

Tretia úloha sa ukázala byť pre niektorých testovaných náročnejšia ako očakávali. Tí z nich, ktorí už mali skúsenosti s iným web scraperom, túto úlohu zvládli relatívne jednoducho, ale už aj pri nich sa začali ukazovať informačné nedostatky aplikácie. Testovaní nevedeli, v akom formáte majú dáta vpisovať, alebo odkiaľ majú získať jednotlivé RSS a sémantické záznamy. Moje očakávanie pri riešení tejto úlohy bolo, že aj testovaný, ktorý nemá veľké znalosti ohľadom web scrapingu a nevie v akom tvare má zadávať jednotlivé selektory, tak sa pozrie do dokumentácie a pokúsi sa nájsť informácie, ktoré ho navedú ku riešeniu. Moje očakávanie však nebolo úplne správne. Keď si testovaní nevedeli dať rady, alebo nemali tušenie ako nastavovať scraper samostatne, väčšina začala skúšať všetky tlačidlá, ktoré pri nastavovaní extrakcie videli, skúšali rôzne formy zápisov selektorov a až po chvíľke neúspešných pokusov si 4 testovaní uvedomili, že aplikácia obsahuje dokumentáciu a nahliadli do nej. Ostatní sa ma začali pýtať otázky a požiadali ma o pomoc a vysvetlenie, na čo som im odpovedal iba tým, že v aplikácii sa nachádza aj dokumentácia. Nakoniec sa všetkým podarilo úlohu dokončiť bez toho, aby som im odpovedal na akékoľvek otázky alebo im pomáhal. Pri tejto úlohe sa jasne ukázalo, že pri konkrétnom nastavení extrakcie, ktoré určuje používateľ nebolo vôbec jasné, ako sa majú jednotlivé hodnoty zapisovať a chýbali prvky, ktoré by naviedli testovaných do dokumentácie, kde už je všetko vysvetlené.

Pri štvrtej úlohe, zobrazenie dokončených extrakcií nebolo až tak prehľadné ako som očakával a testovaným chvíľku trvalo, kým našli tlačidlo s ktorým si tieto dokončené extrakcie zobrazili.

Poslednú úlohu, a teda zobrazenie si všetkých stiahnutých záznamov a prezretie si verzií

článku zvládli všetci testovaní pomerne rýchlo a nemali pri riešení tejto úlohy problémy. Všetci prekontrolovali, či sa údaje ktoré scraper extrahoval zhodovali s tými, ktoré nastavovali pri jednotlivých zdrojoch. Testovaní overili a potvrdili, že údaje boli extrahované podľa zadaných špecifikácií.

### 5.1.3 Dotazník

Po ukončení testovania som všetkých testovaných poprosil o vyplnenie dotazníka. Kompletné hodnoty dotazníka sú obsiahnuté v prílohe.

Z otázky A.3 vyplýva, že iba dvaja zo 7 mali skúsenosti s iným web scraperom pred začatím testovania, čo sa aj ukázalo pri samotnom testovaní.

Na otázku A.4, 5 testovaných odpovedalo, že podľa nich bolo používanie aplikácie dostatočne intuitívne a 2 zvolili, že aplikácia bola iba mierne intuitívna.

V otázke A.7 mali testovaní možnosť napísať, čo sa im na aplikácií páčilo. V tejto časti spomenuli predpripravené nastavenia extrakcie, možnosť filtrovania stiahnutých záznamov, rýchlosť odozvy pri zmene v nastaveniach extrakcie.

V otázke A.8 mali testovaní možnosť vyjadriť, čo sa im na aplikácií nepáčilo. Pri tejto časti upozornili na nedostačujúce informovanie používateľa o tom, že extrakcia začala, prvky nastavenia neboli úplne prehľadné a nebolo jasné, ktoré časti sú spoločné, teda popisujú rovnakú časť, a ktoré patria do inej. Spomenutý bol aj responzívny dizajn, ktorý pri zmenšovaní obrazovky začal byť neprehľadný. Taktiež jeden testovaný poznamenal, že aplikácia nie je úplne intuitívna pre tých používateľov, ktorý toho o web scrapingu veľa nevedia.

V poslednej otázke A.10 mohli testovaný vyjadriť svoje nápady, vylepšenia, dotazy, čo by v aplikácií zmenili, alebo čo by do nej pridali, ak by na nej pracovali oni. Dvaja poznamenali, že zaujímavým rozšírením by bolo, ak by sa do aplikácie pridalo vyhľadávanie a sťahovanie článkov aj podľa kľúčového slova. Používateľ by si vytvoril šablónu pre extrahovanie dát v takej podobe ako je to dostupné teraz, ale mal by možnosť okrem sťahovania dát z RSS kanálu, určiť si kľúčové slovo. Na základe kľúčového slova by sa spustil web crawler, ktorý by našiel články obsahujúce toto slovo a extrahoval by z nich dáta podľa definovanej šablóny. Ďalším rozšírením bolo spomenuté automatické kategorizovanie stiahnutých článkov. Tým by sa rozšírila možnosť pri filtrovaní uložených článkov a používateľ by vedel bližšie špecifikovať o aké záznamy má záujem. Okrem rozšírení testovaní uviedli aj úpravy, ktoré by v aplikácií urobili. Jednou z úprav je výraznejšie odlíšenie toho, či je scraper pre daný zdroj aktívny alebo nie. Jeden testovaný navrhol pridanie funkcie, ktorá by spúšťala scraper pre všetky zdroje naraz s rovnakými nastaveniami extrakcie dát, čím by sa podľa neho urýchlilo spúšťanie a zastavovanie extrakcie. Ďalším návrhom na úpravu bolo aj zvýraznenie popisu jednotlivých tlačidiel kvôli prehľadnosti a popis častí, ktoré menia nastavenia scrapera pre daný zdroj.

### 5.1.4 Nasadenie

V testovaní sa ukázali niektoré nedostatky v podobne nedostatočnej informovanosti používateľa pri spúšťaní extrakcie, nedostatočných popisov jednotlivých častí, nedostatočného grafického rozdelenia komponentov. Pre informovanie užívateľa o spustení extrakcie som implementoval *alert*, ktorý jasne informuje o tom, že extrahovanie začalo a súčasne som do podstránky s nastavením extrakcie implementoval viacero prvkov, ktoré informujú používateľa o aktuálnom stave extrakcie. Rovnako som pridal do vrchného menu aj možnosť prepnutia si zdroja, pretože doteraz to bolo možné len z domovskej obrazovky. Ku každému tlačidlu som pridal *<Tooltip>* z MUI knižnice. Tým som zabezpečil, že každé tlačidlo obsahuje krátky popis jeho funkcionality, poprípade odkaz na časť v dokumentácií v ktorej je táto funkcionalita bližšie popísaná. Rovnako som pridal aj informačné prvky s *<Tooltip>*, ktoré bližšie informujú o danej časti a zväčša obsahujú aj odkaz na presnú časť v dokumentácií, kde sa nachádzajú bližšie informácie tejto danej časti. Rovnako som vylepšil aj responzivitu stránky, aby bola čitateľná aj na menších obrazovkách.

Aplikáciu som nahral do verejného repozitára na GitHub. Aplikácia je sprístupnená ako celok, teda je potrebné mať backednovú aj frontendovú časť na spustenie aplikácie. Ku aplikácií som v README došpecifikoval požiadavky, ktoré aplikácia potrebuje na korektné spustenie. Pripravil som aj návod ako si aplikáciu prvý krát nainštalovať a spustiť. Pre prvotnú inštaláciu som vytvoril jednoduchý script, ktorý vytvorí a nainštaluje všetky potrebné rozšírenia. Súčasne som vytvoril spúšťací script, ktorý umožňuje spustiť aplikáciu ako celok.

# Kapitola 6
# Záver

Hlavným cieľom tejto diplomovej práce bolo vytvorenie konfigurovateľného webového scrapera pre extrahovanie dát zo spravodajských portálov. Hlavný cieľ a postup práce sa rozdelil na niekoľko častí.

V prvej časti sme priblížili funkciu a vlastnosti spravodajských portálov spoločne s ich možnosťami zverejňovania dát. Následne sme sa venovali obecnému vysvetleniu web scrapingu, jeho vlastnostiam, druhom a použitiu.

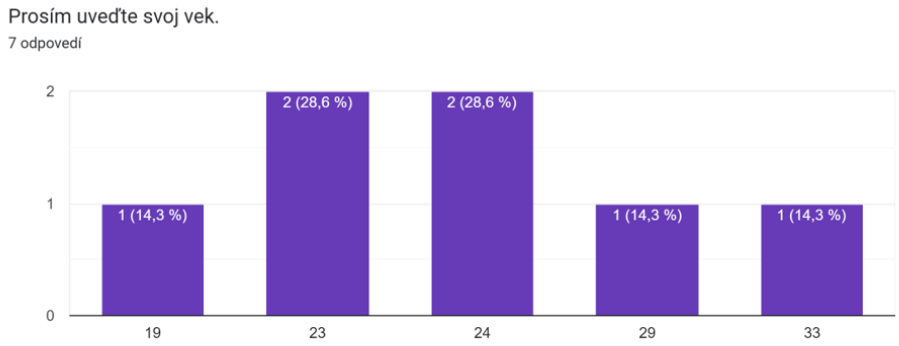
V druhej časti sme sa venovali analýze existujúcich riešení, ktoré ponúkajú web scraping ako SaaS službu. Analyzovali sme niekoľko služieb, ich používanie a funkcionalitu, silné a slabé stránky, možnosti využitia pre riešenie tejto práce. Rovnako sme sa venovali analýze často používaných programovacích jazykov, ich rozšíreniam a knižniciam, ktoré sa používajú pri tvorbe vlastných webových scraperov. Medzi analyzované jazyky patrili Python, Node.js a Ruby. Na základe tejto analýzy sme určili Node.js a jeho knižnice a rozšírenia ako najvhodnejšie pre pokračovanie v tejto diplomovej práci.

Z výsledkov analýzy sme ako prvé v tretej časti definovali funkčné a nefunkčné požiadavky práce. Podľa týchto požiadaviek sme následne vytvorili návrh používateľského prostredia, rovnako tak aj návrh databázovej štruktúry pre ukladanie extrahovaných článkov. Súčasne sme si určili aj technológie s ktorými sa pracovalo pri tvorbe. Pre frontendovú časť sme si zvolili framework React a ako databázu sme využili MongoDB.

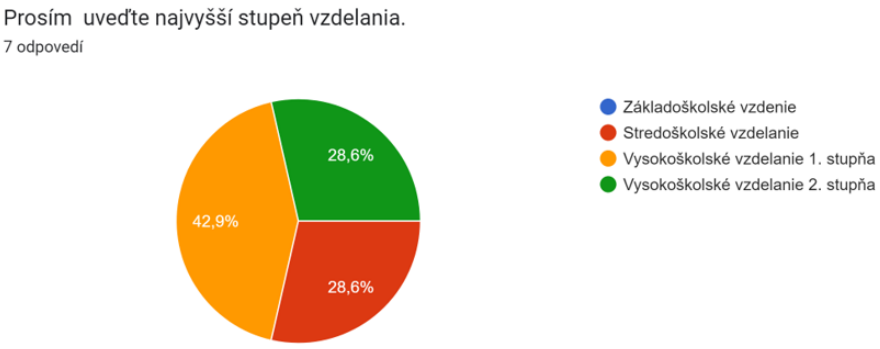
V poslednej časti sme úspešne implementovali všetky funkčné požiadavky. Požiadavky zo zadania diplomovej práce boli tiež splnené. Pre prácu so scraperom bolo vytvorené používateľské prostredie, cez ktoré je možné scraper ovládať, rovnako tak aj zobrazovať extrahované dáta a ich príslušné verzie. Systém bol navrhnutý tak, aby umožňoval samostatné nastavovanie uložených zdrojov spravodajských portálov. Pre každý zdroj je možné vytvoriť samostatnú šablónu, podľa ktorej sa budú dáta extrahovať, súčasne s možnosťou plánovania opakovanej extrakcie. Nastavenie extrahovania pre zdroj je zobrazené aj na obrázku B.4, podobne ako implementovaná časť pre filtrovanie je zobrazená na obrázku B.3. Webová aplikácia bola otestovaná používateľmi, ktorý počas testovania poukázali na pár nedostatkov. Tieto nedostatky boli po testovaní opravené. Aplikácia po implementácií požiadavky pre ukladanie verzií článkov bola schopná zaznamenať zmeny v článkoch pri opakovanom extrahovaní dát. Tieto zmeny je možné vidieť na obrázkoch B.1, B.2. Pri testovaní boli používateľmi spomenuté možné vylepšenia aplikácie o možnosť kategorizovania článkov alebo možnosti extrahovania dát aj pomocou web crawlera.

# Dodatok A

# Dotazník

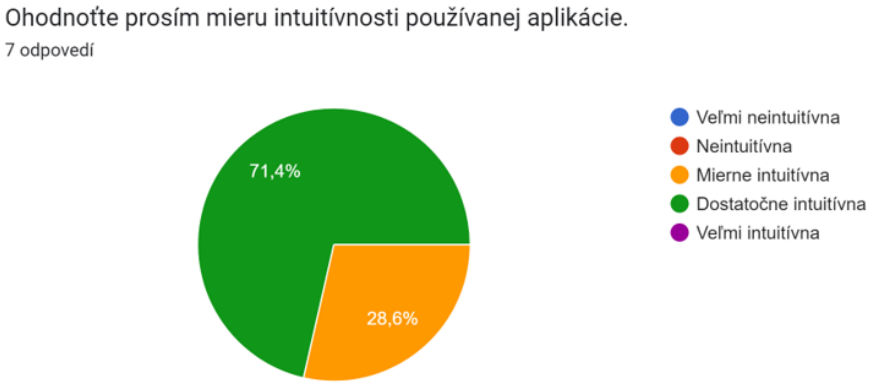



**Obr. A.1** Dotazník - 1. otázka

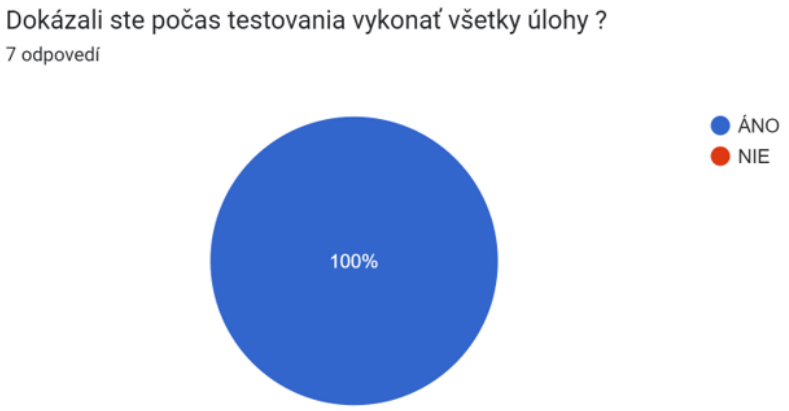



**Obr. A.2** Dotazník - 2. otázka


**Obr. A.3** Dotazník - 3. otázka

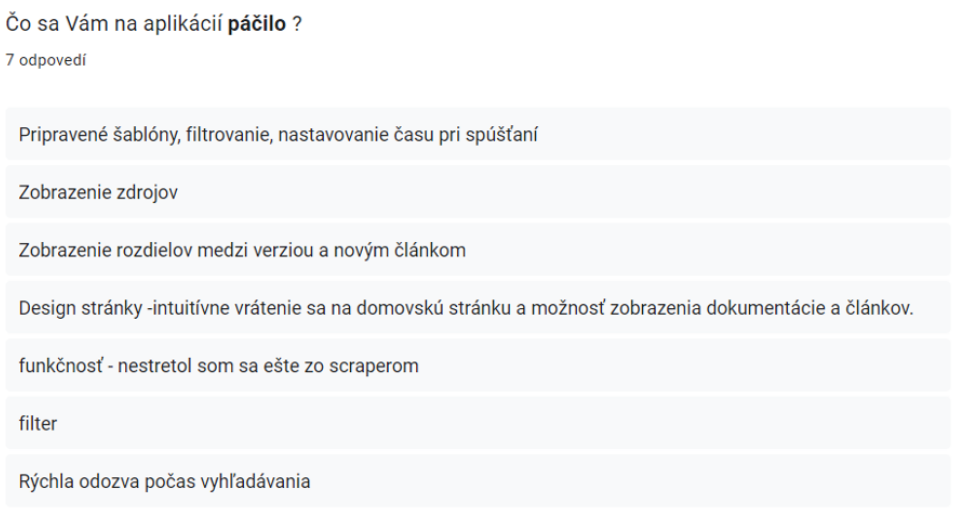



**Obr. A.4** Dotazník - 4. otázka

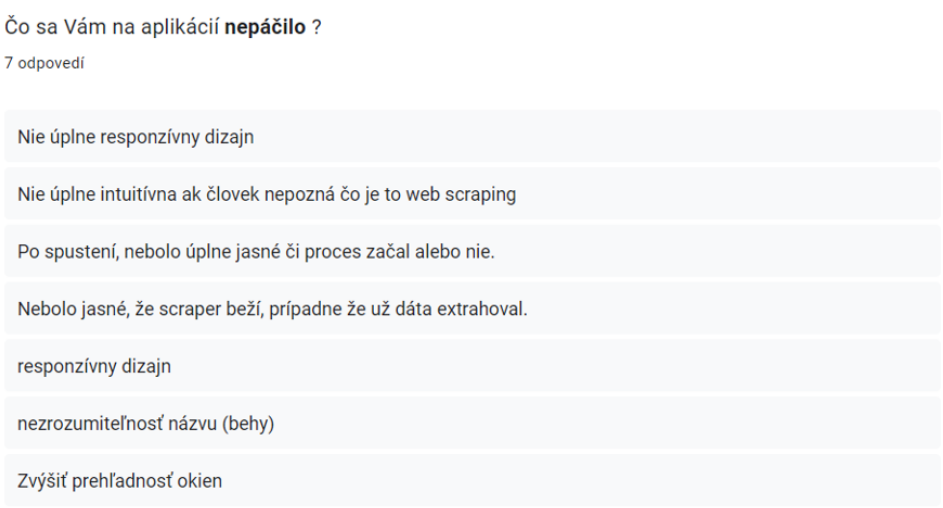



**Obr. A.5** Dotazník - 5. otázka

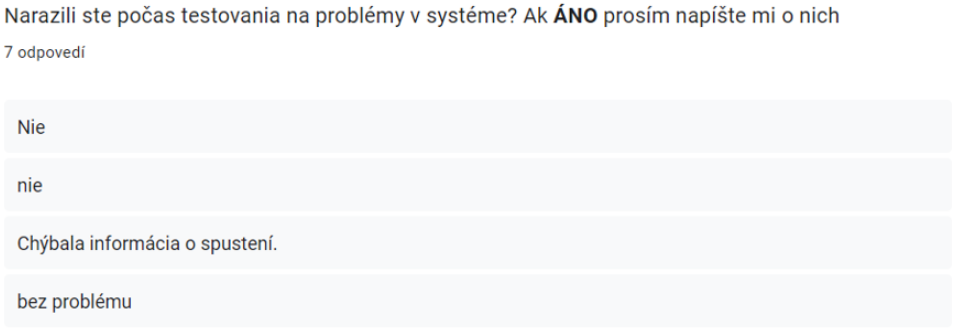

**Obr. A.6** Dotazník - 6. otázka



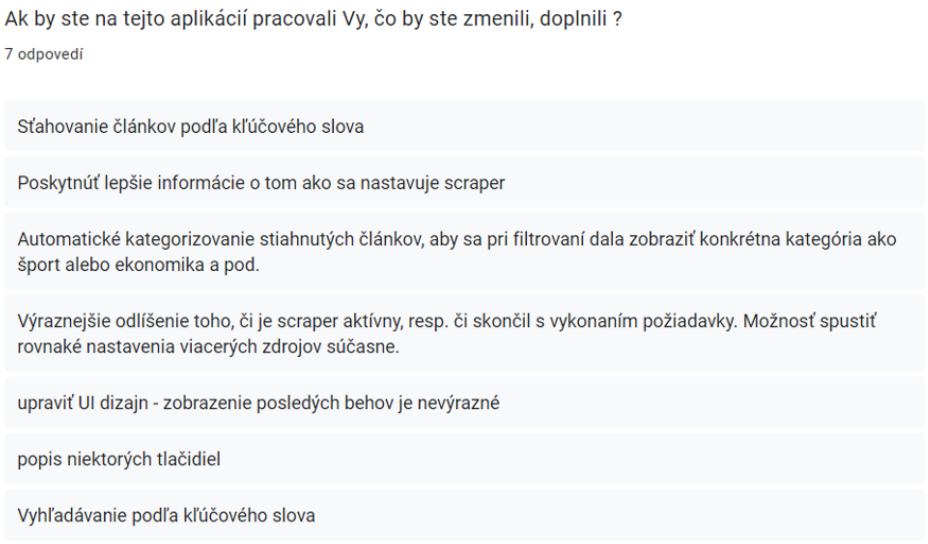

**Obr. A.7** Dotazník - 7. otázka




**Obr. A.8** Dotazník - 8. otázka




**Obr. A.9** Dotazník - 9. otázka



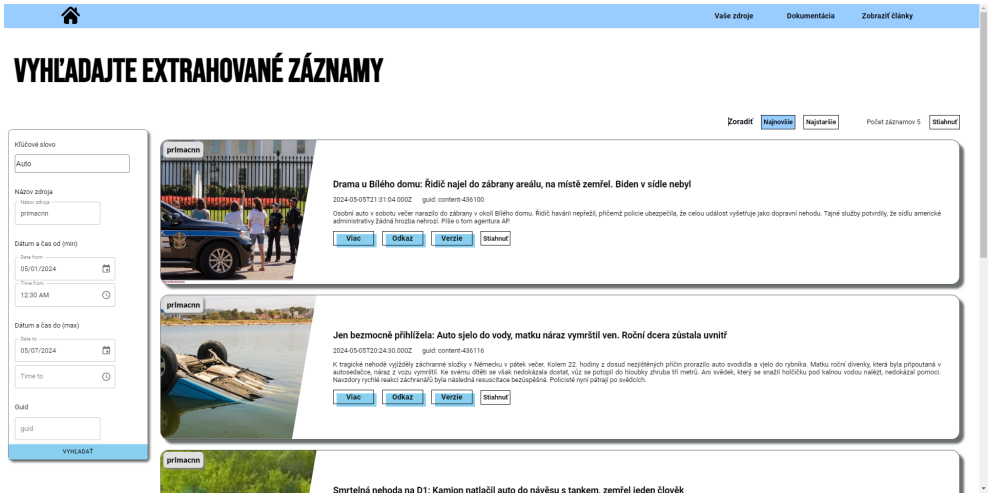

**Obr. A.10** Dotazník - 10. otázka

# Dodatok B

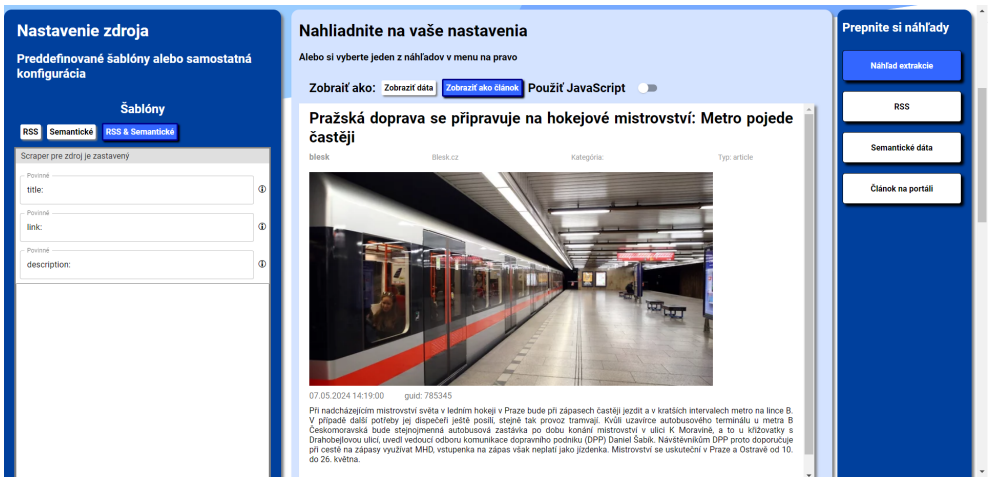
# Používateľské prostredie


**Obr. B.1** Zobrazenie aktuálne uloženého článku


**Obr. B.2** Zobrazenie verzie článku. Viditeľné zmeny v obsahu článku




**Obr. B.3** Filtrovanie článkov. Nastavená filtrácia na kľúčové slovo **auto**, záznamy iba zo zdroja **primacnn**, od **1.5.2024** do **7.5.2024**




**Obr. B.4** Nastavenie extrakcie pre kontrétny zdroj

# Bibliografia


1. TAMBE, Nikita; JAIN, Aashika. *Top Website Statistics And Trends*. 2024-02. Dostupné tiež z: https://www.forbes.com/advisor/in/business/software/website-statistics/.
2. *In-depth guide to how Google Search works*. Google, 2024. Dostupné tiež z: https://developers.google.com/search/docs/fundamentals/how-search-works.
3. *News Sites Information*. GlobalSpec. Dostupné tiež z: https://www.globalspec.com/learnmore/business_services/news_sites.
4. ÇELIKBAŞ, Zeki. *What is RSS and how can it serve libraries?* Istanbul Technical University, 2024. Dostupné tiež z: https://core.ac.uk/download/pdf/11877833.pdf.
5. CALVELO, Manuel. *All About The Semantic Web*. Simple A LLC, 2023-04. Dostupné tiež z: https://simplea.com/Articles/semantic-web.
6. *Learn about sitemaps*. Google, 2024. Dostupné tiež z: https://developers.google.com/search/docs/crawling-indexing/sitemaps/overview.
7. ROUSE, Margaret. *Web Scraping*. 2023-08. Dostupné tiež z: https://www.techopedia.com/definition/5212/web-scraping.
8. BASQUES, Kayce; EMELIANOVA, Sofia. 2016-03. Dostupné tiež z: https://developer.chrome.com/docs/devtools/overview.
9. BASQUES, Kayce; EMELIANOVA, Sofia. *Lighthouse: Optimize website speed*. 2018-11. Dostupné tiež z: https://developer.chrome.com/docs/devtools/lighthouse#compression.
10. ROUSE, Margaret. *Web Crawler*. 2017-01. Dostupné tiež z: https://www.techopedia.com/definition/10008/web-crawler.
11. *What is robots.txt? — How a robots.txt file works*. [B.r.]. Dostupné tiež z: https://www.cloudflare.com/learning/bots/what-is-robots-txt/.
12. BEAL, Vangie. *Proxy Server*. 2024. Dostupné tiež z: https://www.techopedia.com/definition/4200/proxy-server.
13. *Introduction: Scraping API & Smartproxy Public API management documentation for your Smartproxy user dashboard control*. Smartproxy, [b.r.]. Dostupné tiež z: https://help.smartproxy.com/reference/introduction-1.
14. *Documentation: Scraping a site*. Web Graph SIA. Dostupné tiež z: https://webscraper.io/documentation/scraping-a-site.
15. *Documentation: Web Scraper Cloud*. Web Graph SIA. Dostupné tiež z: https://webscraper.io/documentation/web-scraper-cloud.

16. CEYLAN, Burak. *New: In-Depth Diffbot Review: Key Capabilities & Pricing in 2024*. AIMultiple, 2024-01. Dostupné tiež z: https://research.aimultiple.com/diffbot/.

17. *Products Overview*. Diffbot. Dostupné tiež z: https://docs.diffbot.com/docs/what-diffbot-product-do-i-need.

18. *A free web scraper that is easy to use*. ParseHub. Dostupné tiež z: https://www.parsehub.com/.

19. *Extract any data from the web. Easily.* ParseHub. Dostupné tiež z: https://www.parsehub.com/intro.

20. *Development: quick start: Local development*. Apify. Dostupné tiež z: https://docs.apify.com/platform/actors/development/quick-start.

21. *Crawlee is a web scraping and browser automation library*. Apify. Dostupné tiež z: https://crawlee.dev/.

22. KARATAS, Gulbahar. *Top 7 Python Web Scraping Libraries & Tools in 2024*. AIMultiple, 2024-01. Dostupné tiež z: https://research.aimultiple.com/python-web-scraping-libraries/#easy-footnote-bottom-2-63879.

23. RICHARDSON, Leonard. *Beautiful Soup Documentation*. Dostupné tiež z: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

24. KOUZIS-LOUKAS, Dimitrios. *Learning Scrapy: Learn the art of efficient web scraping and crawling with Python*. Packt Pub Ltd, 2016.

25. *Scrapy Beginners Series Part 1: How To Build Your First Production Scraper*. ScrapeOps. Dostupné tiež z: https://scrapeops.io/python-scrapy-playbook/scrapy-beginners-guide/.

26. *Most popular technologies: Web frameworks and technologies*. Stack Overflow, 2023. Dostupné tiež z: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies.

27. *Introduction to Node.js*. Node.js. Dostupné tiež z: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs.

28. RIVERA, Isabel. *The Best Node.js Libraries for Web Scraping in 2024*. 2024-01. Dostupné tiež z: https://proxyway.com/guides/the-best-node-js-libraries-for-web-scraping.

29. DALESSIO, Mike. *Parsing an HTML / XML Document*. 2024. Dostupné tiež z: https://nokogiri.org/tutorials/parsing_an_html_xml_document.html#basic-searching.

30. *Mechanize: Description*. 2024. Dostupné tiež z: https://github.com/sparklemotion/mechanize.

31. LINDLEY, Cody. *What Does a Front-End Developer Do? Complete Guide to the Front-End Developer Profession*. 2018. Dostupné tiež z: https://frontendmasters.com/guides/front-end-handbook/2018/what-is-a-FD.html.

32. YAKYMENKO, Valentyn. *The Role of UX in Front-End Development*. Medium, 2022-12. Dostupné tiež z: https://medium.com/@vyakymenko/the-role-of-ux-in-front-end-development-8caac7171e7f.

33. *React*. 2024. Dostupné tiež z: https://react.dev/learn. Oficiálna stránka.

34. *Document Object Model (DOM)*. MDN Web Docs, 2023. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.

35. MALIK, Faizan. *What is Declarative programming in React?* Educative. Dostupné tiež z: https://www.educative.io/answers/what-is-declarative-programming-in-react.

36. SOOD, Deepak. *Categories of Databases — A Primer: Categories of Databases*. 2020. Dostupné tiež z: `https://medium.com/analytics-vidhya/categories-of-databases-a-primer-9781a3b24285`.

37. *What is a relational database?* Google Cloud. Dostupné tiež z: `https://cloud.google.com/learn/what-is-a-relational-database`.

38. *What is NoSQL?* MongoDB, 2024. Dostupné tiež z: `https://www.mongodb.com/nosql-explained`.

39. SHARMA, Manu. *MongoDB Complete Guide: Develop Strong Understanding of Administering MongoDB, CRUD Operations, MongoDB Commands, MongoDB Compass, MongoDB Server, MongoDB ... and MongoDB Sharding (English Edition)*. BPB Publications, 2021.

40. BARNUM, Carol M. *Usability Testing Essentials: Ready, Set ...Test!* Morgan Kaufmann, 2020.

# Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `git.txt` ..... odkaz na projekt na GitHub
  - `code` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF