Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Webový vyhledávač s web crawlerem

Bakalářská práce

2024 Artem Puzik

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Artem Puzik**
Osobní číslo:	**I21218**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Webový vyhledávač s web crawlerem**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je návrh a implementace nástroje realizující vyhledávání textů v rámci webových stránek. Nástroj bude disponovat vlastním web crawlerem pro sběr textů a jejich indexování.
V textové části práce budou popsány základní principy a koncepce návrhu webového vyhledáče založeného na indexování obsahu s užitím web crawleru. V textu práce budou popsány možnosti procházení webu, problematika procházení stránek generovaných v javascriptu na klientovi a obecné možnosti prohledávání zaznamenaného textu a jeho řazení dle relevance.
V praktické části bude zrealizován vlastní webový vyhledávač s web crawlerem. Backend webového vyhledávače bude zrealizován pomocí ASP.NET Core nabízející REST API. Web crawler bude zrealizován v C# a výsledky prohledávání budou indexovány do vybrané databáze či jiného úložiště.

Rozsah pracovní zprávy: **(min. 30 stran)**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

LEVENE, Mark. *An Introduction to Search Engines and Web Navigation*. Wiley, 2011, 504 s. ISBN 9781118060346.
LOCK, Andrew. *ASP.NET Core in action*. Second edition. Shelter Island, NY: Manning, [2021]. ISBN 9781617298301.

Vedoucí bakalářské práce: **Ing. Roman Diviš, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlašuji:

Práci s názvem webový vyhledávač s web crawlerem jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 5. 2024

Artem Puzik v. r.

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Romanu Divišovi, Ph.D. za vstřícný přístup a cenné rady při zpracovaní bakalářské práce. Dále rád bych poděkoval Ing. Viktoriie Savinove za pomoc s formální úpravou práce a účinnou motivaci během psání této bakalářské práce.

ANOTACE

Cílem této bakalářské práce je návrh a implementace webového vyhledávače s web crawlerem, zaměřeného na efektivní vyhledávání, sběr a indexaci dat z webových stránek. V teoretické části se analyzuje struktura webových vyhledávačů, včetně principů web scraping technologií a indexovacích strategií. Praktická část popisuje realizaci systému pomocí platformy .NET, doplněnou o databáze Elasticsearch, Neo4j a Redis. Součástí praktické části je realizace rozhraní REST API pro snadnou integraci s dalšími aplikacemi. Testování aplikace potvrzuje schopnost systému rychle a přesně zpracovávat dotazy a poskytovat relevantní výsledky, čímž se prokazuje vysoká účinnost a robustnost navrženého řešení.

KLÍČOVÁ SLOVA

Webový vyhledávač, Web crawler, Indexace webových stránek, Extrakce dat z webu

TITLE

Web search engine with web crawler

ANOTATION

The aim of this bachelor thesis is to design and implement a web search engine with an integrated web crawler, focusing on efficient searching, collection, and indexing of data from web pages. The theoretical part analyzes the structure of web search engines, including the principles of web scraping technologies and indexing strategies. The practical section describes the system implementation using the .NET platform, supplemented with Elasticsearch, Neo4j, and Redis databases. The practical part also includes the implementation of a REST API for easy integration with other applications. Testing confirms the system's ability to process queries quickly and accurately and provide relevant results, thereby demonstrating the high efficiency and robustness of the proposed solution.

KEYWORDS

Web search engine, Web crawler, Web pages indexing, Web scraping

Obsah

Seznam obrázků

Seznam výpisů

Seznam zkratek

AJAX – Asynchronous JavaScript and XML

API – Application Programming Interface

DOM – Document Object Model

HTML – Hyper Text Markup Language

HTTP – Hypertext Transfer Protocol

JSON – JavaScript Object Notation

MVC – Model-View-Controller

OWL – Web Ontology Language

REST – Representational State Transfer

RPC – Remote Procedure Call

SMTP – Simple Mail Transfer Protocol

SOAP – Simple Object Access Protocol

SPA – Single-page application

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

WWW – World Wide Web

XML – Extensible Markup Language

XPath – XML Path Language

Úvod

Cílem této bakalářské práce je návrh a implementace nástroje pro vyhledávání webových stránek na základě vyhledávacího dotazu. Tento nástroj bude disponovat vlastním web crawlerem, který bude sloužit k sběru a indexaci textů z různých zdrojů. Základním účelem práce je poskytnout komplexní řešení, které umožní efektivní vyhledávání relevantního obsahu ve velkém objemu webových dat.

První část práce se zaměří na teoretické základy a koncepční přístupy k návrhu webových vyhledávačů. Budou zde podrobně rozebrány principy indexování obsahu a využití technologií pro procházení webu. Speciální pozornost bude věnována problematice efektivního zpracování webových stránek generovaných pomocí JavaScriptu, což představuje významnou výzvu v kontextu moderního webového vývoje. Dále bude analyzována možnost prohledávání indexovaného textu a jeho řazení podle relevance, což je klíčové pro užitečnost vyhledávače.

V praktické části bude popsán návrh a implementace webového vyhledávače. Backend vyhledávače bude realizován pomocí aplikačního rámce ASP.NET Core, který poskytuje robustní prostředí pro vývoj serverových aplikací a webových rozhraní. Vlastní web crawler bude implementován v jazyce C#, což umožní efektivní sběr a zpracování dat. Výsledky vyhledávání budou indexovány a ukládány do vybraného typu databáze či jiného typu úložiště, což zaručí rychlý přístup k datům a jejich snadnou aktualizaci.

1 Principy vyhledávačů

Vyhledávače představují klíčový prvek moderního digitálního světa, který umožňuje uživatelům snadno navigovat ve velkém množství informací. Základem fungování vyhledávačů jsou sofistikované algoritmy a principy, které řídí proces vyhledávání, aby uživatelům byly poskytnuty co nejrelevantnější výsledky. Tyto principy zahrnují komplexní metody a techniky, které společně tvoří základní kameny, na kterých stojí efektivita a užitečnost vyhledávačů.

1.1 Definice vyhledávačů

Pod pojmem vyhledávač se obecně rozumí softwarová aplikace, jejíž hlavním cílem je prohledávání a výběr relevantních záznamů z předem vytvořené databáze na základě vyhledávacího dotazu. Vyhledávače umožňují uživatelům spustit proces vyhledávání a okamžitě získat výsledky hledání v strukturovaném tvaru [1]. Přitom zadaný vyhledávací dotaz má zásadní význam pro určení obsahu výsledných informací vracených vyhledávačem uživateli [2].

Ačkoliv vyhledávače jsou běžně asociované se systémy, které umožňují hledání informací ve World Wide Web (WWW), čili webovými vyhledávači, např. Google, existují také další implementace. Vyhledávače, které prozkoumávají jiné typy zdrojů, zahrnují širokou škálu specializovaných systémů určených pro specifické účely nebo typy dat. Příkladem takových systémů jsou akademické vyhledávače, vnitrofiremní vyhledávače nebo vyhledávače v digitálních knihovnách a archívech. Tyto a obdobné alternativní druhy vyhledávačů jsou primárně jednoúčelové, pro něž je specifické, že vyhledávací dotaz často souvisí s omezeným rozsahem témat nebo typu zdrojů k prohledávání. v takových případech není potřeba řešit řádu dílčích úkolů, které jsou typické pro systémy fulltextového vyhledávání informací [2]. Táto práce se zaměřuje na webové vyhledávače a proto dále v textu práce se pojmem vyhledávač rozumí webový vyhledávač.

Vzhledem k postupnému vzniku a vývoji vyhledávacích systémů, existuje několik druhů vyhledávačů, v nichž se různým způsobem přistupuje k principu fungování vyhledávače. Historicky první vyhledávače fungovali na manuálním principu - do jejichž správy byly zapojeni lidé. Později vznikly poloautomatické a automatické vyhledávací systémy. Nyní lze vyčlenit čtyři základní typy webových vyhledávačů: [1]

Katalogy vytvořené lidmi – jsou výčty webových stránek vytvořených člověkem. Katalog se doplňuje předložením příslušného požadavku od majitele webové stránky, který obsahuje adresu stránky, název a krátký popis obsahu. Požadavek je pak prozkoumán redaktory, kteří posoudí zda stránka bude zařazena do katalogu. Hledání v katalogu se pak uskutečňuje jen podle popisu předložených stránek. Výhodou katalogů je lepší kvalita obsahu a vyšší přesnost vracených výsledků hledání ve srovnání s vyhledávači založenými na crawleru jelikož jsou udržované ručně. Z jiné strany, nevýhodou je, že změna obsahu již předložené stránky se nezaktualizuje až do momentu dalšího odevzdání a kontroly redaktorem. v současné době se tento typ webových vyhledávačů téměř nepoužívá pro účely všeobecného vyhledávání informací na WWW především kvůli enormnímu růstu a rozsahu WWW. Moderní katalogy jsou primárně využívané v specializovaných nebo tematicky zaměřených oblastech, kde přesnost a kvalita informací je důležitější než pokrytí širokého rozsahu internetu a pravidelná aktualizace obsahu.

Vyhledávače založené na crawleru – zpravidla obsahuje tři hlavní části. První část je crawler resp. robot, který prochází internet a vytváří index webových stránek. Druhá část je index, což je databáze všech navštívených stránek, v němž jsou uložená indexovaná slova z každé stránky spolu s metadaty. Třetí část je software vyhledávače, který umožňuje zpřístupnit relevantní indexované stránky pomocí vyhledávacího dotazu. Vzhledem k tomu, že crawler prohledává web v pravidelných intervalech, tyto vyhledávače jsou schopné reflektovat změnu webu s relativně krátkým zpožděním, nicméně nelze zaručit okamžitou aktualizaci dat.

Meta vyhledávače – načítají výsledky z jiných vyhledávačů a přiřazují vlastní rankování výsledkům. Tenhle typ vyhledávačů byl obzvlášť užitečný v době, kdy vyhledávače měly značně různé indexy a také byly méně inteligentní. Jelikož kvalita vyhledávačů se významně zlepšila, potřeba v meta vyhledávačích se hodně zmenšila.

Hybridní vyhledávače – kombinují v sobe vlastnosti vyhledávačů založených na crawleru

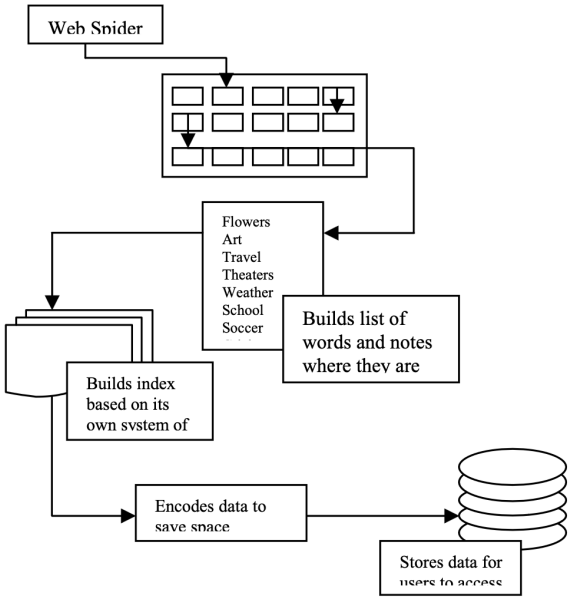
a katalogů vytvořených lidmi. Na tomto principu fungují populárni vyhledávače jako je například Google, čímž využívají výhody obojích přístupu aby poskytovaly co nejlepší výsledky.

I přesto, že nejpopulárnější vyhledávače jsou hybridní[1], jejích hlavní částí je crawler, proto táto práce se zaměřuje na webové vyhledávače založené na crawleru.

1.2 Princip fungování vyhledávačů

Princip fungování typických moderních vyhledávačů zahrnuje několik klíčových komponent. První z nich umožňuje automatický sběr dat z webových zdrojů, což lze provést pomocí "crawlerů" neboli "robotů". Druhá část, která se jmenuje "index", slouží jako úložiště pro načtené soubory dat a zajišťuje efektivní fulltextové vyhledávání relevantních zdrojů podle vyhledávacího dotazu. Třetí části vyhledávače je algoritmus čili software na sestavení odpovědí, který obsahuje informace získané z indexu. [2]

Přehled dílčích částí vyhledávače, založeného na crawleru je zobrazen na obrázku 1



Obrázek 1: Vnitřní struktura vyhledávače. Zdroj: [1]

Dále v dílčích oddílech této podkapitoly budou popsány jednotlivé složky vyhledávače založeného na crawleru.

1.2.1 Crawler

Crawler je počítačový program, který prochází síť propojených stránek ve WWW bez lidské interakce, což je běžně nazýváno pojmem "robot". Robot je typ programu, navržený k automatizaci opakovaných úloh s efektivitou a rychlostí, které překonávají lidské schopnosti [3]. Pojem robot v kontextu WWW také znamená počítačový program, který imituje lidské chování, jako je posílání textových zpráv, nebo automaticky sbírá data [1].

Vyhledávače používají crawlery pro procházení webových stránek, které se následně indexují a ukazují se ve výsledcích hledání. Úkolem crawleru je projít velké množství přístupných webových stránek a sestavit seznam nalezených slov. Proces sestavování seznamu slov se označuje pojmem "Web Crawling". [1]

Při svém prvním spuštění crawler zahajuje proces procházení WWW z předem stanovených webových stránek, které slouží jako jeho výchozí bod. Zpravidla pro výchozí bod jsou zvolené intenzivně používané servery a obecně rozšířené webové stránky, které obsahují dostatečný počet odkazů resp. hyperlinků směřujících na jiné různorodé webové zdroje. Crawler následně indexuje přítomná slova a prostřednictvím hyperlinků objevených v Hyper Text Markup Language (HTML) těle stránky, se rozšiřuje na další weby. [1]

V průběhu tohoto postupu crawler musí pečlivě volit následující stránku, kterou navštíví. Proto, aby došlo k regulaci zmíněného procesu, byly zavedeny politiky pro řízení crawleru, některé z nichž jsou: [1]

1. Výběrová politika – deklaruje které stránky stahovat.
2. Politika opakovaného navštívení – deklaruje frekvence prověření změn na stránce.
3. Zdvořilostní politika – deklaruje jak se vyhnout přetížení webových stránek.
4. Politika souběžností – deklaruje jak koordinovat různé crawlery pracující souběžně.

Crawler je velmi důležitá část systému vyhledávače, která prochází a stahuje velké množství webových stránek pro další zpracování.

1.2.2 Index

Index je databáze která udržuje nalezená slova a místa, kde byla nalezená. Proces ukládání záznamů do indexu se často nazývá "indexování". v rámci ukládání dat vyhledávače

používají kompresní techniky pro minimalizaci potřebného úložiště, což je nezbytné kvůli obrovskému množství informací, které musí být zpracováno a uchováno. k dosažení tohoto cíle se často využívají sofistikované datové struktury, které umožňují efektivní ukládání a vyhledávání dat. [1]

Zvolená metoda indexování informace bezprostředně ovlivňuje rychlost se kterou vyhledávač může získat uloženou informaci. Jeden z efektivních způsobů je využití hašovací tabulky pro indexaci. Fungování hašovací tabulky je založeno na principu vypočítání hašovací funkce pro každé slovo a použití výsledku tohoto výpočtu jako klíče do tabulky hodnot. Hašovací tabulka zajišťuje také přibližně stejnou rychlost hledání pro libovolný záznam, což by nebylo možné při použití struktury, která implikuje sekvenční prohledávání záznamů. Jelikož objem slov, které začínají vybraným písmenem se řádově liší, hledání bez vyrovnání může být zpomalené pro specifické případy. Hašování také zmenšuje průměrný čas hledání záznamu, protože obsahuje přímé odkazy na skutečná data. Tudíž využití efektivního indexování a metody ukládání dat umožňuje poskytovat výsledky rychleji a to platí i pro komplikovanější dotazy. [1]

Data získaná během crawlingu musí být uložená způsobem, který je pro koncového uživatele užitečný, což zahrnuje integraci metadat pro zlepšení kvality a relevance výsledků hledání. Metadata webové stránky typicky obsahují doplňující informace o výskytu každého slova na stránce, což zahrnuje údaje jako jsou pozice od začátku textu, velikost písma a okolní text. Tato zásadní metadata umožňují nabídnout relevantnější a uživatelsky přívětivé výsledky tím, že přidávají kontext k nalezeným slovům a přiřazují jím váhu. [3]

1.2.3 Software vyhledávače

Třetí úkol v systému vyhledávače je vykonáván jeho softwarem. Software vyhledávače zkoumá veškeré indexované stránky a provádí hodnocení na základě jejích relevance pomocí faktorů specifických pro každý systém. Některé zásadní faktory jsou ale společné pro všechny vyhledávače. Níže jsou uvedeny vybrané faktory používané pro hodnocení ve většině vyhledávačů: [1]

- Umístění klíčových slov. Vyhledávače přiřazují každému slovu váhu, založenou na jeho pozici a umístění na stránce, tzn. slova umístěná v titulcích, podtitulcích nebo klíčových odstavcích mohou být považována za relevantnější než slova skrytá v hlubších částech obsahu.

- Četnost klíčových slov. Stránka, na které se vyskytuje více klíčových slov je relevantnější.

Zmíněné faktory, používané pro hodnocení relevancí ve vyhledávačích, jsou cílem zneužití na některých webových stránkách, které prostřednictvím opakovaného vkládání klíčových slov, použitím neviditelného textu a dalšími technikami byly zejména v minulosti [2] schopné zmást crawlery. Tato praxe spočívá v umělém zvyšování pozice stránky ve výsledcích vyhledávání tím, že se klíčová slova umísťují do titulků, podtitulků nebo klíčových odstavců, kde mají vyšší váhu, nebo se klíčová slova opakují častěji než by odpovídalo přirozenému výskytu v textu, čímž se stránka jeví jako více relevantní. Tato metoda však vede k negativnímu uživatelskému zážitku a je penalizována vyhledávači, které se snaží prosazovat kvalitní a relevantní obsah [3].

1.3 Shrnutí

Vyhledávač je softwarová aplikace, která na základě vyhledávacího dotazu prohledává a vybírá relevantní záznamy z předem vytvořené databáze. Historicky existovalo několik typů vyhledávacích systémů, nicméně moderní vyhledávače jsou často založené na crawleru. Veškeré vyhledávače založené na crawleru se skládají z třech hlavních částí, a to jsou: crawler, index a software vyhledávače. I v rámci těchto základních složek, každá realizace vyhledávacího systému může mít svá specifika, což je důvodem proč různé vyhledávače poskytují odlišné výsledky v reakci na identický vyhledávací dotaz [4]. Například, volba jiné metody přidělování vah může způsobit odlišné řazení výsledku hledání [1].

2 Extrakce dat z webu (web scraping)

Extrakce dat z webu, také známá jako web scraping, je technika určená k extrakcí informací z webových stránek WWW. Extrakci dat lze provádět buď manuálně nebo automaticky pomocí webového robota resp. webového crawlera. Nicméně kvůli enormnímu počtu dat generovanému ve WWW, manuální způsob je příliš neefektivní, proto se běžně používají webové crawlery [5]. Webové crawlery provádí extrakci dat buď pomocí pouhé implementace nízkoúrovňových dotazu v Hypertext Transfer Protocol (HTTP) anebo pomocí automatizace neboli zabudování plnohodnotného webového prohlížeče jako Mozilla Firefox nebo Google Chrome do procesu extrakce dat [6]. Každopádně web scraping je velice náročný proces, a to i s využitím webových crawlerů, tudíž zásadní úlohou je promyšlení robustní architektury této části systému [3].

2.1 Webové crawlery

Webový crawler je program, který na základě jedné nebo více startovních Uniform Resource Locator (URL) adres stahuje webové stránky spojené s těmito adresami, extrahuje soubor všech hypertextových odkazů a rekurzivně pokračuje ve stahování webových stránek příslušných těmto odkazům. Webové crawlery představují klíčovou součást vyhledávačů, kde jsou využívány ke shromažďování obsahu webových stránek, které se následně ukládají v indexu. [7]

Navzdory své konceptuální jednoduchosti, implementace výkonných webových crawlerů představuje významné inženýrské výzvy z důvodu rozsahu současného webu. Aby bylo možné procházet podstatnou část aspoň "povrchového" čili již indexovaného webu v rozumném čase, webové crawlery musí stahovat tisíce stránek za sekundu. Proto jsou webové crawlery typicky distribuovány přes desítky nebo stovky počítačů. Dvě hlavní datové struktury, které používají crawlery jsou "frontier" neboli soubor URL adres čekajících na procházení a soubor objevených URL adres. Crawlery v provozu shromažďují velké množství dat, které obvykle převyšuje kapacitu operační paměti a proto je potřeba také zajistit efektivní reprezentaci v sekundární paměti (např. pevný nebo polovodičový disk). [7]

Jelikož počet nově objevených odkazů resp. "frontier" nekontrolovaně roste, je důležité stanovit priority jejich navštěvování. Pro stanovení priority nelze použit historická

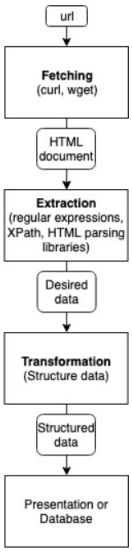
data o samotné stránce, je však možné provádět informované odhady o kvalitě stránky na základě statistik pro daný web. Kvalitu stránky je obtížné kvantifikovat, a proto se často využívají nepřímé ukazatele, jako jsou metriky založené na četnosti odkazování typu PageRank nebo behaviorální metriky, jako je počet návštěv stránky nebo webu, získané například z webových majáků (Web beacon). [7]

2.2 Proces extrakce dat

Proces interakce crawleru s webovým serverem s cílem extrahovat data je možné rozdělit do třech [5] sekvenčních kroků:

1. Pořízení dat ze zdrojů resp. webových stránek – crawler sestavuje HTTP požadavky na webové servery pro získání webové stránky. Požadavek může obsahovat jednu z více HTTP metod, např. GET nebo POST. v momentě když zvolený server obdrží HTTP požadavek a úspěšné ho zpracuje, požadováný zdroj bude vrácen crawleru v odpovědi serveru. Obsah odpovědi může být vrácen buď v textových formátech, nejvyužívanějšími z nichž jsou HTML, Extensible Markup Language (XML) a JavaScript Object Notation (JSON), anebo zahrnovat multimediální data, např. video, obrázky, atd [5]. Tato práce se zabývá pouze textovým obsahem a neřeší otázku sběru multimediálních dat.
2. Extrakce – následně po úspěšném získání dat z webového serveru crawler extrahuje data, která jsou pro účely crawleru klíčová. Tento krok obvykle využívá techniky, které zahrnují regulární výrazy, knihovny pro parsování HTML a dotazy XML Path Language (XPath) [8].
3. Transformace získaných nestrukturovaných dat – web scraping je úzce spojen s indexováním v kontextu vyhledávačů, poněvadž dalším účelem crawleru je transformace dat do tvaru, vhodného pro ukládání a analýzu v indexu. v okamžik když crawler obdrží odpověď webového serveru její obsah je následně rozebrán čili parsován, přeformátován a reorganizován do strukturovaného tvaru [5].

Uvedené kroky procesu extrakce dat jsou znázorněné na obrázku 2.



Obrázek 2: Proces interakce s webovým serverem. Zdroj: [8]

2.3 Přístupová politika na Webu

Přístupová politika na webu je určená k regulaci chování crawlerů při procházení webových stránek, aby se předešlo přetížení serverů a zbytečnému indexování. Protokol pro vyloučení robotů (angl. Robots Exclusion Protocol) byl poprvé navržen Marjitinem Kosterem v roce 1994, nicméně doposud je de-facto standardem a nepatří žádné organizaci. Tento protokol je implementován umístěním souboru s názvem "robots.txt" v kořenové složce webového serveru anebo vložením meta tagů na samotných webových stránkách. [9]

Obsah souboru "robots.txt" popisuje která část webového zdroje není určená k procházení daným crawlerem. Existují tři typy záznamů přítomných v souboru a na samotných stránkách: [9]

- Vyloučení v rámci celého serveru – nařizuje které složky webového serveru nejsou určené pro procházení a indexování. Přidáním instrukcí do souboru "robots.txt", které obsahují název cílového crawleru, resp. user-agent, lze vyloučit vybrané složky, jejichž seznam musí být pod uvedeným user-agentem.
- Vyloučení po stránkách – je provedeno pomocí meta tagů v hlavičce HTML odpovědi serveru. Meta tagy jsou zahrnuté ve standardu HTML a umožňují přiřadit záznamy typu klíč-hodnota webové stránce. v případě tagů relevantních pro crawlery, klíčem

je “robots” a hodnotou může být “noindex” která nařizuje neindexovat stávající stránku, “nofollow” pro zákaz sledování hyperlinků přítomných na stránce, nebo kombinace obojích.

- Vyloučení z mezipaměti – je využíváno vydavateli, kteří prodávají přístup do jejích informací. Pro účely zveřejnění vydavatele dovolují crawlerům indexovat celou stránku a zároveň nařizují vyhledávačům neukazovat stánku uloženou v mezipaměti, ale místo toho poskytnout pouze odkaz na originál. Toto je prováděno pomocí stejného meta tagu s klíčem “robots” a hodnotou “nocache”.

2.4 Automatizace webového prohlížeče

WWW prošel několika etapy rozvoje a to vznik Web 1.0 a následný přechod na Web 2.0. Přechod na Web 2.0 přinesl do webu tzv. Single-page application (SPA), které využívají nejmodernější technologie webu jako HTML5. Zároveň většina funkcionality SPA je implementována pomocí velkého množství JavaScript resp. Asynchronous JavaScript and XML (AJAX) kódu pro asynchronní získávání dat ze serveru a dynamickou změnu obsahu webové stránky. Weby, které mají pouze jednu stránku a které využívají AJAX kód pro dynamickou změnu obsahu, napodobují chování spíše desktopové *aplikace* než klasického webu, což stanovuje výzvy pro crawlery, jelikož aplikace využívají interakce s elementy stránky pro navigaci místo klasických hyperlinků [10]. Existuje řada specifik při řešení tohoto problému, nicméně tato práce se nezabývá crawlingem SPA pomocí interakce s elementy stránky.

Pro účely zpracování webových aplikací lze využit techniku automatizace webového prohlížeče, která umožní interpretaci JavaScript kódu na klientské stráně, zajistí přesnější přehled dynamické webové stránky a získání většího objemu dat z webového serveru. Pro vzdálené ovládání webového prohlížeče existuje protokol WebDriver. Rozhraní WebDriver poskytuje abstrakci příkazů od konkretního prohlížeče, což zahrnuje akce jako přechod na daný odkaz nebo provedení uživatelské akce na webové stránce. [10]

2.5 Shrnutí

Web scraping představuje důležitou část systému vyhledávače, která zajišťuje získávání dat z webových zdrojů. Web scraping je téměř vždy prováděn pomocí webových

crawlerů čili robotu z důvodu zvýšené efektivity procesů pořízení, extrakce a transformace dat. Webové crawlery interagují s webovým serverem buď pomocí pouhých HTTP dotazů anebo se zapojením plnohodnotného prohlížeče, což umožňuje interpretaci JavaScript kódu na klientské stráně která je především užitečná pro zpracování SPA. Nicméně pro zajištění kvalitního sběru dat z webových serverů a také dodržení "zdvořilostní" politiky byly nabízené standardy omezení indexace webových zdrojů pro crawlery. De-facto standardem je protokol pro vyloučení robotů (Robots Exclusion Protocol), který je implementován umístěním souboru "robots.txt" v kořenové složce webového serveru.

3 Indexování dat

V kontextu vyhledávačů index čili fulltextový index je úložiště dat, které optimalizuje rychlost přístupu k indexovaným webovým stránkám pomocí vyhledávacího dotazu, což umožňuje fulltextové hledání v obrovském množství dokumentů. Existuje více typu indexů, ale většina webových vyhledávačů používá pro vykonávání dotazů tzv. invertovaný textový index, který ukládá pro každé slovo seznam dokumentů, ve kterých se toto slovo vyskytuje. Jelikož aktualizace invertovaných indexů je náročný proces, obvykle vyhledávače pravidelně opakovaně vytvářejí svůj index od základu. Čím častěji je možné index rekonstruovat, tím rychleji se aktualizace odrazí na výsledcích vyhledávání, což zlepšuje celkovou kvalitu vyhledávače. [11]

3.1 Fulltextové hledání

Fulltextové hledání představuje pokročilou techniku vyhledávání v textových zdrojích, která překračuje rámec klasického textového hledání založeného na přesné shodě. Tato technika umožňuje uživatelům vyhledávat relevantní dokumenty na základě jejich celého obsahu pomocí vyhledávacího dotazu, který může zahrnovat jednotlivá slova nebo fráze, booleovské operátory, žolíkové znaky, atd [12]. Výkonné fulltextové hledání v WWW lze dosáhnout pomocí využití fulltextového indexu, který je často implementován jako invertovaný textový index, což také umožňuje provádění fulltextových vyhledávacích dotazů jako jsou [11]:

Standardní dotaz – dotaz, který obsahuje pouze slova nebo víceslovné fráze. Je obvykle implementován jako implicitní booleovský dotaz, kde se mezi jednotlivé vyhledávací slovy vkládají skryté operátory *AND*. Tudíž pro každé slovo ve vyhledávacím dotazu je vrácená množina relevantních dokumentu, na kterých se následně aplikuje operátor *AND*. Tento přístup však může vést k problémům v případě, když uživatel zadá dost velký počet slov, jelikož dokument obsahující většinu, ale ne všechny slova, nebude zařazen do výsledků vyhledávání, i když je velká pravděpodobnost, že je relevantní. Proto některé realizace volí jinou strategii, která umožňuje libovolný počet slov. Tato strategie spočívá ve využiti operátoru *OR* a seřazení výsledků podle počtů nalezených slov v dokumentu.

Booleovský dotaz – dotaz, ve kterém slova jsou kombinována pomocí operátorů Booleovy logiky jako jsou *AND*, *OR* a *NOT*, přičemž lze použít závorky pro seskupení slov. Implementace těchto dotazů využívá invertovaný index pro získání množin dokumentů relevantních pro daná slova a následným provedením logických operací nad těmito množinami. Operátor *OR* spojuje množiny dokumentů obsahující alespoň jedno z slov, zatímco *AND* vyžaduje přítomnost všech slov v dokumentu. Operátor *NOT* odstraňuje dokumenty obsahující dané slovo z předběžné množiny výsledků. Výkonnost provádění těchto operací lze zvýšit optimalizací pořadí zpracování podle počtu relevantních dokumentů pro každé slovo získané z indexu, přičemž slova s menším počtem dokumentů jsou zpracována přednostně. Další optimalizace zahrnuje paralelní získávání a vykonávání operací nad množiny dokumentů pro každé slovo.

Frázový dotaz s pevnou sekvenci slov – dotaz, který slouží k vyhledání dokumentů obsahujících zadaná slova ve specifikovaném pořadí. Typickým indikátorem těchto dotazů je text uvedený uvnitř uvozovek. Vyhledávání pevné sekvence slov je obzvláště užitečné pro lokalizaci dokumentů, kde jsou běžná slova použita ve velmi specifickém kontextu, například při hledání autora citátu. Implementace těchto dotazů je rozšířením booleovských dotazů *AND*, avšak je náročnější na zpracování, jelikož vyžaduje sledování pozic slov v dokumentu. Pro zrychlené hledání pevné sekvence slov je možné využití pomocných indexů, například "index následujícího slova", který indexuje vybrané dvě po sobe jdoucí slova a nabízí výrazné zrychlení s mírným zvetšením objemu indexu.

Proximitní dotaz – dotaz ve tvaru *slovo1 NEAR(n) slovo2*, který slouží k nalezení dokumentů, kde se *slovo1* nachází maximálně n pozic od *slovo2*. Implementace proximitních dotazů je podobná frázovým dotazům s tím rozdílem, že místo ověřování přesných relativních pozic slov může být mezi pozicemi rozdíl maximálně n. Zároveň platí pokud n je 1, proximitní dotaz lze realizovat jako kombinaci booleovských a frázových dotazů.

Dotaz s žolíkovými znaky (angl. wildcard) – dotaz, který představuje formu nepřesného vyhledávání čili "fuzzy". Běžná operace, která se provádí před zpracováním dotazu je "rozšiřování" žolíkového znaku čili získávaní množiny odvozených slov, které lze vytvořit nahrazením žolíkového znaku alfanumerickými písmeny. Pro každé

slovo produkované rozšířením žolíkového znaku je získána množina relevantních dokumentů a je aplikován operátor *OR*. Efektivní implementace na invertovaném indexu vyžaduje také tzv. lexikon čili slovní zásobu pro zrychlení operace rozšíření.

Je důležité zmínit, že fulltextový vyhledávací dotaz může být také kombinaci každého z těchto a dalších typů.

3.2 Invertovaný textový index

Invertovaný textový index lze představit jako datovou strukturu "slovník" neboli kolekci spojených klíčů a hodnot, kde klíčem je slovo a hodnotou je seznam prvků popisujících výskyt (angl. "hit") tohoto slova v dokumentu [11]. Základní algoritmus vytváření invertovaného indexu je popsán v Python pseudokódu na výpisu 1.

```
def memoryInvert(documents):
    index = {}
    for d in documents:
        for t in tokenize(d):
            if t.word not in index:
                index[t.word] = CompressedList() # komprimace
            index[t.word].add(t)
    return index
```

Výpis 1: Algoritmus vytváření invertovaného indexu. Zdroj: [11]

Existuje několik variant invertovaných indexů. v jeho základní verzi pro každé objevené crawlerem slovo je uchováván seznam dokumentů resp. jedinečných identifikátorů dokumentů (docID), ve kterých se toto slovo vyskytuje [13]. Pokud požadavkem je podpora frázových a proximitních dotazů, je pak nutné ukládat pozice slov v každém dokumentu, tedy konkrétní místa, kde se slovo nachází. Granularita pozice slov může být definována různě – od pozice písmena, přes pozici slova, až po odstavec nebo sekci dokumentu. Běžně se granularita ale definuje na úrovni pozice slova. Lze také ukládat pouze frekvenci slov v dokumentu místo jejich pozic, což může být užitečné pro optimalizaci vykonávání vyhledávacích dotazů [11].

3.3 Předzpracování textu

Jelikož webové stránky nejsou indexovány přímo v jejich původní podobě, pro implementaci vyhledávače na invertovaném indexu je potřeba se uchýlit k použití různých technik pro předzpracování textu. Každá webová stránka prochází lexikální analýzou, kde celý obsah je převeden na tokeny, následně každý token je transformován na jeho morfologický kořen a nakonec mohou být některé tokeny v rámci odfiltrování zcela vynechány. [11]

3.3.1 Lexikální analýza

Lexikální analýza je proces převodu dokumentu z sekvence znaků na sekvenci tokenů, kde každý token je posloupnost písmen a cifer resp. alfanumerické slovo. Typicky existuje také maximální délka pro samotný token, aby se zabránilo neomezenému růstu velikosti indexu ve zvláštních případech. Pro generování tokenů ze vstupního proudu znaků je nejprve provedena konverze na malá písmena. Poté je každá skupina alfanumerických znaků oddělená nealfanumerickými znaky jako jsou mezera, interpunkce atd., a přidána do sekvence tokenů. Tokeny obsahující velké množství číselných znaků jsou obvykle z sekvence odstraněny, protože zvětšují velikost indexu bez výraznějšího přínosu. Popsány přístup funguje pouze pro abecední jazyky, ideografické jazyky jako čínština, japonština a korejština, které nemají slova složená ze znaků, vyžadují jiný přístup [11] . Problematika ideografických jazyků je není součásti této práce protože vyžaduje znalosti v lexikologii těchto jazyků.

3.3.2 Transformace tokenů

Tokeny po lexikální analýze jsou transformovány na jejich morfologické kořeny, které jsou následně indexovány. Například slova “počítat”, “počítač”, “výpočet”, “počítače”, “spočítaný” a “počítání” by mohla být indexována jako “počítat”. Algoritmy transformace slov se vyznačují vysokou mírou komplexity a obsahují množství výjimek, což vede k četným chybám a také ke snížení přesnosti při zpracování dotazů, zejména v případě frázových dotazů. v současné době se mnoho vyhledávačů, jako je například Google, rozhoduje neaplikovat transformace na morfologický kořen slova. [11]

3.3.3 Odfiltrování tokenů

Slovní druhy, jako jsou spojky nebo předložky, obvykle jsou tak běžné, že je obsahuje téměř každý text, proto jsou tyto slova zcela ignorována při indexaci webové stránky. Pro běžné dotazy odfiltrování těchto slov obvykle nezhorší výsledky a navíc ušetří místo v indexu. v některých zvláštních případech ale může používání těchto slov úplně znemožnit nalezení požadovaných informací. Moderní vyhledávače, jako například Google, nepoužívají odfiltrování slov, jelikož z hlediska byznysu schopnost vyhledávat libovolnou kombinaci slov má větší výhody než mírné zmenšení velikosti indexu. [11]

3.4 Shrnutí

Indexování dat v efektivní datové struktuře umožňuje vyhledávačům dosáhnout vysoké rychlosti při zpracování uživatelských dotazů. Většina vyhledávačů využívá tzv. invertovaný textový index pro indexování dat. Invertovaný index ukládá data podobně slovníku, kde klíčem je slovo a hodnotou je seznam prvků, které obsahují informace o výskytů slova na webových stránkách čili dokumentech. Různé techniky předzpracování textu lze využit pro zmenšení objemu indexu a zahrnutí většího počtu relevantních dokumentů ve výsledcích hledání. Přesto moderní vyhledávače se neuchylují k těmto technikám kvůli objemu současného webu a poskytování lepší služby uživatelům.

4 Software vyhledávače

Existují aplikace, které využívají crawler a index pro zpracování vyhledávacích dotazů ale neposkytují žádné specifické seřazeni výsledků, což není vyžadované například pro jiné crawlery a boty na webu, které konzumují veškeré dokumenty identicky [11]. Cílem této práce je implementace vyhledávače pro lidské využití, což implikuje inteligentní rankování a zejména poskytování nejrelevantnějších dokumentů na prvních místech ve vraceném výsledku. Moderní vyhledávače nabízejí také řadu pokročilých funkcí a technologií, což zahrnuje fulltextové hledání, podpora sofistikovanějších typů dotazů, sémantické hledání, využití synonym, atd [12].

4.1 Sémantický Web

Sémantický web je web, kde informace jsou strukturované a uložené podle standardizovaných pravidel s cílem poskytování sémantického hledání, které usnadní nalezení relevantních dokumentů. Sémantické hledání je rozšířením klasického fulltextového vyhledávaní, které přináší zvýšení relevantnosti výsledků webového vyhledávání pomocí technik sémantické analýzy. Ačkoliv fulltextové vyhledávače nabízejí efektivní prostředek pro hledání na webu, nezohledňují ale sémantiku dotazů a slov v dokumentech. Sémantické hledání umožňuje nalezení relevantních výsledků ne jen na základě klíčových slov ale také na základě sémantické shody vyhledávacího dotazu a obsahu dokumentu. [14]

Sémantická analýza zohledňuje kontextové vyhledávání, synonyma, úmysl vyhledávacího dotazu, variace slov, generalizované a specializované dotazy, konceptuální shodu a dotazy v přirozeném jazyce. Obvykle metody implementace sémantického vyhledávání využívají ontologie pro vytváření sofistikovaných modelů. Ontologie je formalizovaný popis vztahů, které vyjadřují příslušnost do dané třídy a závislosti mezi entity. Jinými slovy ontologie zahrnuje hierarchický popis důležitých tříd neboli konceptů společně s popisem vlastností instancí každého konceptu ve vybrané doméně. Web Ontology Language (OWL) je skupina jazyků navržených pro definování a sdílení ontologií na webu. Deskriptivní logika je skupina formálních jazyků používaných pro reprezentaci znalostí v doméně a je základním stavebním kamenem sémantického webu. [14]

4.2 Algoritmy rankování stránek

Při implementaci vyhledávače pro současný web je důležité zohlednit pořadí vracení výsledků hledání, jelikož existuje obrovské množství relevantních stránek vzhledem k danému vyhledávacímu dotazu, které člověk není schopen pojmout. Výkon vyhledávačů, které jsou určené pro lidské využití, velice záleží na algoritmu řazení čili rankování stránek, který zajistí umístnění vysoce kvalitních stránek na prvních místech ve výsledcích hledání. [15]

Algoritmy rankování stránek zpravidla spoléhají na různé techniky vytěžování dat z webu (angl. Web Mining). Web mining představuje typ Data Miningu, ve kterém hlavním zdrojem dat je WWW. Techniky data miningu jsou aplikované na datech z webových stránek pro vyhodnocení jejich kvality. Obecně rozlišujeme tři kategorie web miningu:[16]

Vytěžování obsahu webu – proces získávání užitečných informací z obsahu webových dokumentů, což zahrnuje text, obrázky, audio, video, nebo strukturované záznamy, jako jsou tabulky a seznamy.

Vytěžování struktury webu – proces vytvoření strukturálního souhrnu webu. Tento typ web miningu spočívá v objevování struktury odkazů hyperlinků mezi webovými stránkami. Na základě topologie hyperlinků jsou webové stránky následně vyhodnocené.

Vytěžování použití webu – proces extrakce užitečných informací z sekundárních dat odvozených z interakcí uživatele při prohlížení webu. Tento typ vytěžování zpracovává data uložená v logách a cookies prohlížeče, uživatelských profilech, atd.

Je důležité zmínit, že pro implementaci algoritmu rankování stránek lze kombinovat uvedené kategorie web miningu pro poskytování lepších výsledků. [16]

4.2.1 Algoritmus PageRank

Jeden z nejvýznamnějších algoritmů je PageRank původně vyvinutý pro vyhledávač Google a který využívá princip analýzy citování. Cílem analýzy citování je objevení zásadních článků, autorů nebo žurnálu ve vybraném vědeckém studijním oboru, přičemž každá citace článku zvyšuje jeho dopad při citací jiných článků. Algoritmus PageRank aplikuje obdobnou logiku na weby ve WWW, kde citace jsou nahrazený hypertextovými odkazy, proto hlavním způsobem vyhodnocení stránky pro PageRank je vytěžování struktury webu. [16]

Nechť A je libovolná webová stránka a T_1 až T_n jsou webové stránky, které odkazují na A. Skóre PageRank pro A lze spočítat podle vzorce 1. [17]

$$PR(A) = (1 - d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)}) \tag{1}$$

Parametr d představuje činitel tlumení (angl. damping factor), který je obvykle nastaven na 0,85 a předchází tomu aby ostatní stránky měly příliš velký vliv na celkový součet. $C(A)$ je počet odkazů, které vedou ven z stránky A. PageRank lze spočítat pomocí iterativního algoritmu, kde na začátku přidělíme každé stránce skóre PageRank 1 a postupným vypočítáním vzorce 1 dosáhneme konvergování těchto skóre k limitní hodnotě. [16]

4.2.2 Term Frequency–Inverse Document Frequency

Term Frequency–Inverse Document Frequency (TF-IDF) je populární metodou vyhodnocení důležitosti slov ve webových dokumentech, kterou lze využit pro efektivní rankování stránek. TF-IDF lze zařadit do algoritmů rankování stránek, které využívají techniky vytěžování obsahu webu. Mechanismus fungování TF-IDF určuje důležitost slov na základě jejích frekvence vyskytování v obsahu dokumentů a vzácnosti napříč ostatní dokumenty. [18]

Frekvence vyskytování (TF) je ukazatel, který určuje četnost slov v daném dokumentu. TF lze spočítat podle vzorce 2. Větší hodnota znamená, že slovo v kontextu dokumentu je důležitější. [18]

$$TF_{w,d} = \frac{f_{w,d}}{max\{f_{w',d} : w' \in d\}} \tag{2}$$

Kde $f_{w,d}$ označuje opakující se výskyt slova w v dokumentu d. [18]

Inverzní frekvence dokumentu (IDF) posuzuje vzácnost slova napříč všemi dokumenty, kde vyšší hodnota IDF indikuje slovo jako vzácnější. IDF lze spočítat podle vzorce 3. [18]

$$IDF_{w,D} = log\frac{|D|}{|\{d \in D : w \in d\}|} \tag{3}$$

Kde $IDF_{w,D}$ je logaritmická škála, která dělí celkový počet dokumentů $|D|$ počtem dokumentů, které obsahují slovo w. [18]

Kombinací ukazatelů TF a IDF do TF-IDF je možné určit důležitost slov ve vztahu k uživatelskému dotazu a podle toho provést řazení výsledků. Vysoká hodnota TF-IDF pro slovo v dokumentu znamená, že slovo je zároveň často používané v daném dokumentu a vzácné v ostatních dokumentech, což naznačuje vysokou kvalitu dokumentu pro vyhledávací dotaz obsahující toto slovo. Hodnotu TF-IDF je vypočítaná podle vzorce 4. [18]

$$TF{-}IDF_{w,d} = TF_{w,d} \times IDF_{w,D} \tag{4}$$

4.3 Shrnutí

Software vyhledávače zpracovává data uložená v indexu a umožňuje vyhledávání pomocí pokročilých technik hledání. Vyhledávače využívají různé algoritmy rankování stránek, které spoléhají na metody web miningu, pro poskytování nejvíc kvalitních stránek na prvních místech ve výsledku vyhledávání.

5 Programové rozhraní vyhledávače

Aby vyhledávač byl užitečný pro koncového uživatele je potřeba zveřejnit jeho funkce, což lze udělat například pomocí implementace uživatelského nebo aplikačního rozhraní. Uživatelské rozhraní vyhledávače minimálně umožňuje zadání vyhledávacího dotazu a zobrazení přehledu výsledků vyhledávání. Aplikační rozhraní neboli Application Programming Interface (API) je určeno pro konzumaci jinými programy a typicky nabízí pouze efektivní způsob přenosu dat bez grafického obalu [19]. v rámci této práce bude implementováno pouze aplikační rozhraní, aby byla umožněna integrace vyhledávače s jinými systémy.

5.1 Application Programming Interface

API představuje soubor pravidel a protokolů, který je zásadní pro vývoj a integraci softwarových aplikací. Hlavním účelem API je zajištění a usnadněni vzájemné komunikace mezi různými softwarovými aplikacemi, což umožňuje sdílení dat a propojení funkcionalit. API definuje metody a datové struktury, které programátoři mohou využít k získání přístupu k funkcím nebo datům poskytovaným externím softwarovým systémem, a to bez nutnosti znát detailní implementaci tohoto systému. [20]

Existuje více typu API, nicméně v současné době převládají zejména webová API, která poskytují přístup k datům a funkcionalitě aplikace prostřednictvím internetu. Typický webová API poskytují přístup k webovým službám jako jsou platební brány, internet věcí, online mapy, atd. Webová API lze rozdělit do několika typů, a to jsou: [20]

- Veřejná API – otevřená API pro veřejné použití přes vystavené endpointy.
- Partnerská API – slouží pro integraci systémů byznys partnerů a typicky vyžaduje autentizaci pro přístup.
- Vnitrofiremní API – skrytá API pro interní použití a vylepšení firemních procesů.

Pro webová API je vyvinuto několik přístupu pro standardizovanou komunikaci mezi webovými aplikaci. Některé z nichž jsou: [20]

Remote Procedure Call (RPC) – jednoduchý mechanismus, který abstrahuje volání programové procedury na jiném adresovém prostoru.

Simple Object Access Protocol (SOAP) – protokol, který využívá XML pro reprezentaci dat, což usnadňuje komunikaci aplikací napsaných v různých programovacích jazycích. SOAP také podporuje přenos dat pomocí protokolů nižších úrovní jako jsou Simple Mail Transfer Protocol (SMTP) a HTTP.

Representational State Transfer (REST) – není protokol, ale soubor principů vývoje webového API, které definují REST architektonická omezení. REST API běžně využívají JSON pro reprezentaci dat a HTTP jako protokol přenosu dat. Typickou vlastnosti těchto API je tzv. statelessness, což znamená, že každý požadavek musí obsahovat všechny informace potřebné k jeho zpracování.

Jelikož moderní webové API zpravidla dodržují REST principy [20], programové rozhraní vyhledávače bude implementováno prostřednictvím REST API.

5.1.1 RESTful rozhraní

Webové API, které dodržuje principy REST architektonického stylu lze také popsat termínem RESTful API [20]. REST definuje 6 omezení pro API a to jsou [21]:

1. Jednotné rozhraní – definuje rozhraní pro spotřebitele a poskytovatele webové služby. Cílem je zjednodušení a oddělování architektury, což umožňuje nezávislost každé části systému. Rozhraní obsahuje jednotlivé zdroje, které lze zpřístupnit pomocí Uniform Resource Identifier (URI).
2. Bezstavová komunikace – naznačuje, že každý jednotlivý požadavek od klienta serveru obsahuje veškerou informaci potřebnou pro zjištění stavu komunikace. Pro reprezentaci stavu v požadavku jsou využité možnosti protokolu HTTP, což zahrnuje URI cestu a parametry, hlavičky a tělo požadavku.
3. Využití mezipaměti – umožňuje klientům ukládat odpovědi serveru do mezipaměti. Předpokladem pro splnění tohoto požadavku je označení odpovědi jako cachovatelné aby se zabránilo v použití zastaralých dat.
4. Architektura klient-server – architektura, která spočívá v oddělení klientů od serverů a rozdělení jejích úloh. Tento přístup nezatěžuje servery úlohami grafické prezentace dat a klienty metodami ukládání a manipulace těchto dat.
5. Vrstvený systém – naznačuje, že klient nemůže rozpoznat zda komunikuje s hlavním

serverem nebo s prostředníky. Tato vlastnost zlepšuje škálovatelnost systému tím, že prostřednici můžou poskytovat tzv. load balancing a sdílenou mezipaměť.

6. Kód na vyžádání – volitelná funkcionalita, která umožňuje serverům dočasně rozšířit nebo přizpůsobit klienta tím, že mu přenesou logiku, kterou můžou vykonávat.

REST API popisuje množinu zdrojů a operací, které lze provést na těmito zdroji. Veškeré zdroje také mají tzv. bázová cesta (angl. base path), vůči kterému jsou definované URI těchto zdrojů. Bázová cesta funguje jako izolovaný kontext, ve kterém používáme služby serveru, proto můžeme tuto cestu i měnit aby přepínat cílový REST API. [21]

5.2 Shrnutí

Pro umožnění komunikace vyhledávače s jinými aplikacemi lze implementovat aplikační rozhraní čili API, které vystaví funkčnost systému. API je souborem pravidel, které zajišťuje komunikaci mezi různými programy, v kontextu webového API komunikace je mezi webovými servery. Populární volbou pro moderní API je REST, který definuje řádu omezení a architektonických stylů API.

6 Realizace webového vyhledávače

V rámci této práce byl implementován webový vyhledávač s web cralwerem, který disponuje REST API pro umožnění integrace s dalšími systémy. Větší část logiky vyhledávače byla realizovaná pomocí vyššího programovacího jazyka, nicméně součásti implementace jsou také i externí nástroje, aplikační rámce a databáze, které usnadňují implementaci dílčích části vyhledávače. Implementovaný systém je horizontálně škálovatelný, tzn. existuje podpora zvětšení jeho kapacity prostřednictvím přidávání dalších počítačových strojů do systému. Veškeré složky systému byly zprovozněné v izolovaných na úrovni jádra operačního systému, samostatných instancích čili kontejnerech pro usnadnění testování a nasazení. Veškeré podklady a zdrojové kódy řešení lze najít v elektronické příloze práce.

6.1 Vývojové nástroje

Pro realizaci logiky aplikace byla použita vývojová platforma .NET 8 a vyšší programovací jazyk C# 12. Psání programového kódu probíhalo ve vývojovém prostředí Rider, který nabízí široké možnosti vývoje na platformě .NET včetně vývoje webových aplikací a aplikací pro příkazový řádek. Samotná platforma .NET také nabízí různé nástroje pro vytvoření řešení (angl. solution) a projektů, správu NuGet balíčků, sestaveni a spuštění programu.

Pro vyšší flexibilitu veškeré části systému vyhledávače byly sestavené a nasázené v kontejnerech. Docker byl zvolen jako platforma pro vytvoření tzv. image a zprovoznění kontejnerů. Externí softwarové programy, které jsou součásti systému byly stažené ve formě již připraveného image z registru Docker. Pro správu aplikace, která se skládá z více kontejnerů, byl použit nástroj Docker Compose.

6.2 Web Crawler

Součásti řešení je konzolová aplikace crawler, která prochází web a ukládá stránky do databáze. Celá logika programu byla psána v programovacím jazyce C# a zahrnuje komunikace s webovými servery, parsování získaného obsahu webových stránek a ukládání příslušných záznamů do databáze. Crawler udržuje dvě datové struktury a to jsou fronta nenavštívených odkazů a množina navštívených odkazů. Crawler začíná procházeni webu

z stránek uvedených v konfiguraci programu a ukládá nalezené odkazy do fronty. Po procházení webové stránky její URI se ukládá do množiny navštívených odkazů. Před vložením do fronty crawler kontroluje, že nalezený odkaz není součásti množiny navštívených odkazů, pro prevence opakovaného navštívení.

Crawler implementuje většinu požadavků, které definuje Robots Exclusion Protocol. Byl stažen externí NuGet balíček *TurnerSoftware.RobotsExclusionTools* pro parsování souboru `robots.txt`, ze kterého je vyčteno požadované pravidla pro daný webový zdroj. Crawler sdělí svoje jméno prostřednictvím HTTP hlavický `User-Agent` a umožňuje cílená pravidla pro daný crawler. Stránky webového zdroje, které nejsou určené pro indexováni, crawler buď zcela odmítá zařádit do fronty nenavštívených odkazů nebo neukládá do indexu, pokud objeví příslušný meta tag v obsahu stránky. Je také striktně dodržen požadavek webového zdroje o minimálním intervalu mezi HTTP dotazy. Pro každou doménu crawler sleduje čas posledního HTTP dotazu a vypočítává nejdříve možný čas následujícího dotazu na základe `crawl-delay` záznamu v souboru `robots.txt`, pokud je k dispozici.

Pro parsování obsahu stránky byl využit externí NuGet balíček *HtmlAgilityPack*, který poskytuje sadu metod pro čtení a manipulaci HTML Document Object Model (DOM). Po stažení webové stránky crawler extrahoval podstatnou informaci z HTML obsahu stránky, což zahrnuje obsah těla a název webové stránky, meta tagy v hlavičce určené pro crawlery a také hyperlinky přítomné v těle stránky. Před uložením záznamu do indexu bylo také provedeno odstranění HTML elementů z těla stránky čili normalizace obsahu.

Významnou výzvou také představovalo zpracování nalezených hyperlinků v těle webové stránky. Pro prevence opakovaného navštívení webových stránek a duplikaci záznamů v indexu je důležité správně určit, zda dva syntakticky odlišné URI mohou být ekvivalentní. Pro řešení tohoto problému bylo rozhodnuto před porovnáním odkazů provést normalizaci čili převedení daných URI na jejich standardizovaný tvar. Normalizace URI je z hlediska přesnosti identifikace odkazovaného zdroje poměrně obtížný úkol, který není možné provést dokonale, jelikož existují různé specifika webových serverů, které nejsou zaručené standardy. Pro zmenšení počtu duplikovaných odkazu byla provedená normalizace, kterou poskytuje platforma .NET prostřednictvím třídy `Uri`, a také byly použité funkce NuGet balíčku *Toimik.UrlNormalization*.

Pro zvětšení počtu možných případů použití vyhledávacího systému bylo realizováno

tři režimy a příslušná pravidla procházení webu crawlerem, a to jsou:

1. Procházení webu bez omezení. Veškeré nalezené odkazy jsou posílané do fronty nenavštívených odkazů bez specifického filtrování.
2. Omezení na vybrané domény. Konfigurace crawleru musí obsahovat seznam povolených domén. Crawler rozhoduje zda zařadit nalezený odkaz na základě doménové části URI.
3. Předdefinování vybraných webových zdrojů. Crawler pouze vybírá již vložené odkazy z fronty nenavštívených odkazů, veškeré nalezené odkazy jsou ignorované.

Byla realizovaná podpora dvou režimu provozu crawleru a to jsou spuštění jedinečné instance a distribuované spuštění více instancí souběžně. Odlišnosti těchto dvou režimů spočívají v způsobu spravovaní datových struktur crawleru. v případě jedinečné instance crawler ukládá veškeré URI do datových struktur, které se nacházejí v operační paměti. v případě spuštění víc instancí crawlerů vzniká problém synchronizace, zejména prevence opakovaného navštíveni webových stránek a dodržování pravidla `crawl-delay` vůči webu. Pro řešení těchto problému bylo zvoleno zavedení centrální databáze, oproti které veškeré instance crawlerů synchronizují své aktivity. Jako centrální databáze byla zvolena NoSQL databáze Redis. Pro distribuované crawlery Redis slouží úložištěm pro frontu nenavštívených odkazů a množinu navštívených odkazů. v režimu spuštění více crawlerů souběžně každá instance komunikuje s centrální Redis databází aby získat další URI pro procházení a pro kontrolu a přidání jíž navštívených stránek.

Z hlediska kvality vyhledávače je zásadní poskytovat relevantní výsledky hledání, což vyžaduje přesnější ponětí obsahu webové stránky. Jedním z přesných způsobu zjištěni obsahu webové stránky je přímé napodobení reálného uživatele, což lze dosáhnout pomocí automatizace webového prohlížeče pro interpretaci JavaScript kódu na klientské stráně. Účelem samotného crawleru je soustředění na logice programu, nikoliv dosažení výkonného či rychlého procházeni webu, proto jedna instance crawleru má k dispozici pouze jednu instanci automatizovaného prohlížeče. Předpokládá se, že pro produkční nasazení bude spuštěno více instancí crawleru pro efektivnější procházení webu. Selenium byl zvolen jako technologie, která umožňuje automatizaci webového prohlížeče.

6.2.1 Selenium

Selenium je sada nástrojů určená pro automatizaci webových prohlížečů. Především Selenium je určen pro automatizaci webových aplikací pro účely testování, nicméně se neomezuje pouze na to. Klíčovou složkou Selenium je Selenium WebDriver. Pod Selenium WebDriver se rozumí serverová část implementace protokolu WebDriver a také knihovny v různých programovacích jazycích, které poskytují rozhraní pro komunikací s tímto serverem podle protokolu WebDriver. [22]

Selenium WebDriver poskytuje implementace protokolu WebDriver pro veškeré významné moderní prohlížeče včetně Google Chrome, Mozilla Firefox, Safari a jiných. Ovládač (angl. driver) v kontextu Selenium je softwarová aplikace, která překládá příkazy jednotného API WebDriver na specifické příkazy pro daný prohlížeč. Toto oddělení je součástí vědomé snahy, aby dodavatelé prohlížečů převzali odpovědnost za implementaci ovládačů pro své prohlížeče. Selenium využívá tyto ovladače třetích stran, kde je to možné, ale poskytuje také vlastní ovladače pro případy, kdy to není možné. Aplikační rámec Selenium spojuje všechny tyto části dohromady prostřednictvím poskytování jednotného rozhraní, které umožňuje transparentní používání různých backendů prohlížečů, což umožňuje automatizaci napříč prohlížeči a platformami. [22]

Pro implementaci crawleru byl zvolen webový prohlížeč Mozilla Firefox a příslušný ovládač *geckodriver*. Selenium nabízí NuGet balíček pro platformu .NET, který umožňuje ovládání vybraného prohlížeče z kódu C#. Pro účely stažení obsahu webových stránek byly použité metody knihovny pro navigaci prohlížeče na vybranou URL adresu a také získání HTML obsahu stánky s aplikovanými změnami provedené JavaScript kódem.

6.3 Index

Index hraje významnou roli v systému vyhledávače, jelikož zajišťuje rychlé hledaní v rozsáhlých textových dokumentech stažených z webu. Pro index vyhledávače byl zvolen nástroj Elasticsearch, který se specializuje na fulltextovém hledání. v rámci této práce nebyla programovaná logika ohledně indexu, poněvadž Elasticsearch již implementuje veškeré požadavky pro index vyhledávače.

6.3.1 Elasticsearch

Elasticsearch je distribuovaný vyhledávací a analytický nástroj, který lze použit pro řešení velkého množství problémů. Vyhledávací a analytická funkcionalita Elasticsearch je založená na knihovně Apache Lucene, která nabízí funkce indexace, tokenizace, kontroly pravopisu a vyhledávání. Elasticsearch je zásadní součást Elastic Stacku (dříve známého jako ELK stack), který zahrnuje nástroje jako jsou Elasticsearch, Kibana, Logstash a další. Elasticsearch je navržen pro rychlé a horizontálně škálovatelné vyhledávání, analýzu a ukládání velkých objemu dat v reálném čase. [23]

Základní jednotkou pro ukládání dat v Elasticsearch je tzv. dokument, který je reprezentován ve formátu JSON. Dokumenty jsou organizovány do indexů, což jsou kolekce dokumentů s podobnými poli. Ve starších verzích Elasticsearch rozděloval indexy dál do typů, nicméně tento přístup byl zrušen pro zjednodušení architektury. Mapping definuje schéma pro dokumenty v indexu, a to včetně datových typů pro jednotlivá pole a způsobu indexace a ukládání dokumentů. [23]

Elasticsearch poskytuje rozsáhlé RESTful API, které umožňuje spravovat systém a provádět operace jako indexace, vyhledávání, aktualizace a mazání dokumentů pomocí HTTP dotazů [23]. Pro ukládání dat do indexu byl využit NuGet balíček *Elastic.Clients.Elasticsearch*, který poskytuje Elastic pro komunikaci s Elasticsearch databází. v databázi byl vytvořen Elasticsearch index, který sloužil úložištěm pro veškeré stažené webové stránky. Ačkoliv Elasticsearch je určený pro distribuované nasazení, byl zprovozněn pouze jeden node, který poskytoval postačující dostupnost indexu vyhledávače. Pro zobrazení grafické analýzy aktivit crawlerů a uložených dokumentů ke Elasticsearch byl připojen nástroj Kibana.

6.4 Software vyhledávače

Pro účely rankování webových stránek bylo realizováno ukládání dat do NoSQL grafové databáze Neo4j, která nabízí algoritmy pro výpočet hodnot PageRank jednotlivých uzlů v grafu. Graf se postupně doplňuje během crawlingu webu a to tak, že po parsování webové stránky uzly s výchozím a nalezenými URI a také hrany reprezentující spojení mezi uzly jsou vloženy do Neo4j. Pro účely spojení indexu vyhledávače a grafové databáze byl vytvořen program, který zahájí výpočet hodnot PageRank nad vybraném grafem a

následně aplikuje výsledky výpočtu na příslušné dokumenty v indexu. Pro komunikaci s Neo4j databázi prostřednictvím C# kódu byl stažen externí NuGet balíček *Neo4j.Driver*.

Realizovaný vyhledávač poskytuje REST API, které umožňuje posílat jednoduché textové vyhledávací dotazy. Rozhraní vrací odpověď ve formátu JSON, která obsahuje pole s maximálně deseti objekty, které představují nejrelevantnější výsledky hledání. Každý výsledek hledání obsahuje název a URI webové stránky, skóre relevantnosti vzhledem k vyhledávacímu dotazu a úryvky obsahu stránky, které obsahují klíčová slova z vyhledávacího dotazu. Tato část systému slouží jako jediný veřejný vstupní a výstupní bod a to s cílem zabránit zneužití systému a omezit přístup uživatelům do Elasticsearch indexu pouze pro čtení. Pro realizaci REST API byl zvolen aplikační rámec ASP.NET Core, který umožňuje vývoj webových aplikací na platformě .NET.

6.4.1 ASP.NET Core

ASP.NET Core je moderní aplikační rámec pro vývoj webových aplikací. ASP.NET Core je postaven na .NET Core, což je multiplatformní verze .NET, která umožňuje aplikacím běžet na Windows, Linux a macOS. To poskytuje vývojářům flexibilitu výběru operačního systému, na kterém chtějí svoje aplikace vyvíjet a provozovat. [24]

ASP.NET Core umožňuje vytvářet jak webové stránky pomocí Model-View-Controller (MVC) vzoru, tak webové API pro mobilní aplikace a webové služby. ASP.NET Core podporuje moderní webové technologie, jako jsou HTML, CSS, JavaScript a také umožňuje snadnou integraci s různými databázemi a externími službami. Aplikační rámec nabízí řadu výhod, jako je modulární architektura, rozšířená konfigurace prostředí, vylepšené možnosti testování a podpora pro kontejnerizaci. [24]

Historicky vzor MVC se používal i pro webové API, nicméně byl zaveden modernější přístup Minimal APIs, který je vhodný pro minimalistické API s menším počtem závislosti. Minimal APIs byl představen v ASP.NET Core 6.0 jako způsob, jak jednoduše a rychle definovat jednoduché HTTP endpointy bez nutnosti vytvářet celé kontrolery, což je běžný přístup v tradičních ASP.NET Core MVC aplikacích. [24]

API pro vyhledávač bylo implementováno v samostatném projektu podle vzoru Minimal APIs s jedním endpointem `/search`, který očekává parametr `query` pro vyhledávací dotaz. Pro komunikaci a dotazování Elasticsearch databáze byl použit NuGet balíček *Elastic.Clients.Elasticsearch*. Účelem API bylo překlad jednoduchého vyhledávacího dotazu

na tvar, který očekává Elasticsearch.

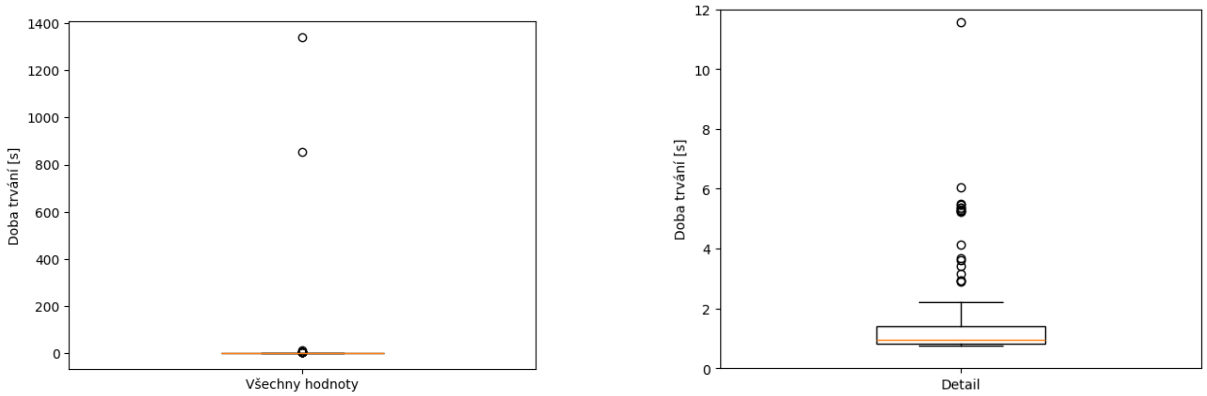
6.5 Testování aplikace

První fáze testování aplikace zahrnovala spuštění vícero instancí crawleru pro procházení a stažení webových stránek. Fáze sběru webových stránek byla provedena pomocí webu *crawler-test.com*, který disponuje velkým počtem odkazů pro testování crawlerů. Crawler byl spuštěn v režimu omezení na doménové jméno *crawler-test.com*, nicméně nebyla specifikována omezení na schéma URI, proto crawler akceptoval odkazy obsahující schéma *http://* a *https://*.

Bylo spuštěno 5 instancí crawlerů se začátečním URI *https://crawler-test.com/*. Crawlery běžely přibližně 40 minut a došlo k indexování 833 webových stránek v Elasticsearch. Celkově bylo navštíveno 1206 webových stránek, což bylo vidět podle velikosti množiny navštívených odkazů uložených v Redis. Velký rozdíl mezi počtem navštívených a skutečně indexovaných stránek je odůvodněn velkým počtem nestandardních stránek přítomných na webu pro testování crawlerů, které bylo odmítnuto crawlery uložit do indexu. Veškeré instance crawlerů skončily po 40 minutách, jelikož nebyly nalezené další odkazy na vybrané doméně, které jsou dosažitelné z počátečního bodu.

Během provozu byly také snímané různé metriky, které ukazují vlastnosti vybrané instance crawleru. Pro jednu zvolenou instanci bylo akumulováno 134 záznamů, které zahrnují časovou značku kdy metrika byla snímaná a číselnou hodnotu metriky.

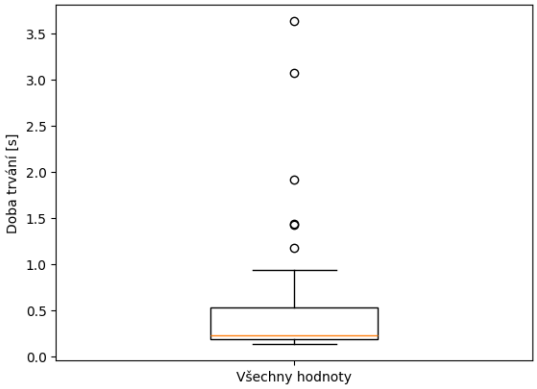
Výsledky měření metriky, která ukazuje rozdělení proměnné celkového času crawlingu webové stránky, je vidět na krabicovém grafu na obrázku 3. Pro dobu trvání crawlingu stránky platí, že minimální hodnota je 0,76 sekund, maximální 1340,84 sekund a medián je 0,94 sekund. Odhadovaný důvod dlouhé maximální doby trvání je specificky vyrobená webová stránka, která zpomaluje získání zdrojového kódu stránky.



(a) Včetně všech hodnot **(b)** Detail podstatné části

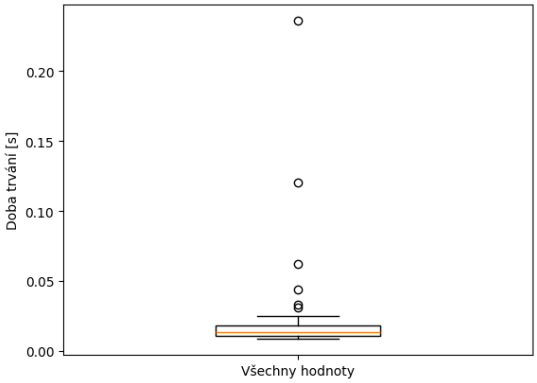
Obrázek 3: Rozdělení proměnné času crawlingu webové stránky. Zdroj: Vlastní

Výsledky měření metriky, která ukazuje rozdělení proměnné času provedení dotazu do webového serveru, je vidět na krabicovém grafu na obrázku 4. Pro dobu trvání dotazu platí, že minimální hodnota je 0,13 sekund, maximální 3,63 sekund a medián je 0,22 sekund. Podle dané metriky je vidět, že web pro testování crawlerů vykazuje poměrně krátkou dobu odezvy.



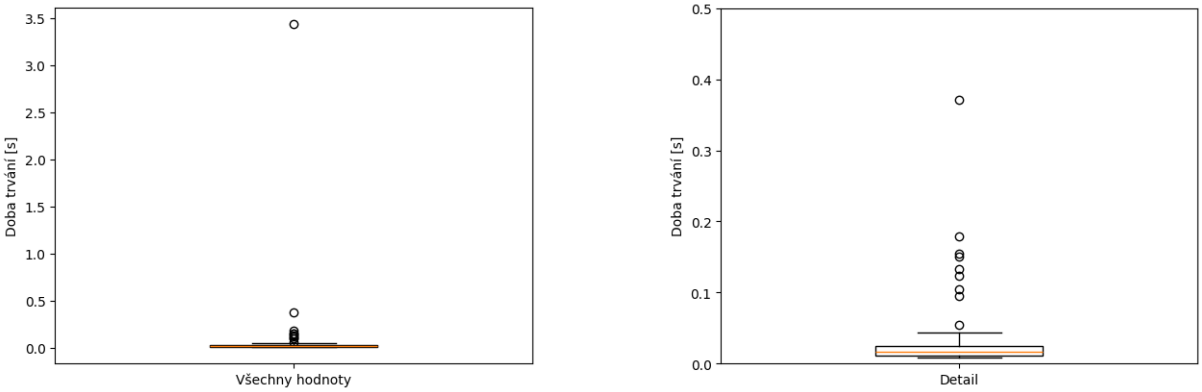
Obrázek 4: Rozdělení proměnné času dotazu do webového serveru. Zdroj: Vlastní

Výsledky měření metriky, která ukazuje rozdělení proměnné času uložení stránky do indexu Elasticsearch, je vidět na krabicovém grafu na obrázku 5. Pro dobu trvání indexování platí, že minimální hodnota je 0,009 sekund, maximální 0,24 sekund a medián je 0,013 sekund. Podle dané metriky je vidět, že Elasticsearch je dobře optimalizován pro ukládání dokumentů do indexu.



Obrázek 5: Rozdělení proměnné času indexování stránky. Zdroj: Vlastní

Výsledky měření metriky, která ukazuje rozdělení proměnné času aktualizaci webového grafu v Neo4j, je vidět na krabicovém grafu na obrázku 5. Aktualizace webového grafu zahrnuje přidání URI a odkazu mezi stránkami, které ještě nejsou v grafu. Pro dobu trvání aktualizace platí, že minimální hodnota je 0,008 sekund, maximální 3,44 sekund a medián je 0,016 sekund.



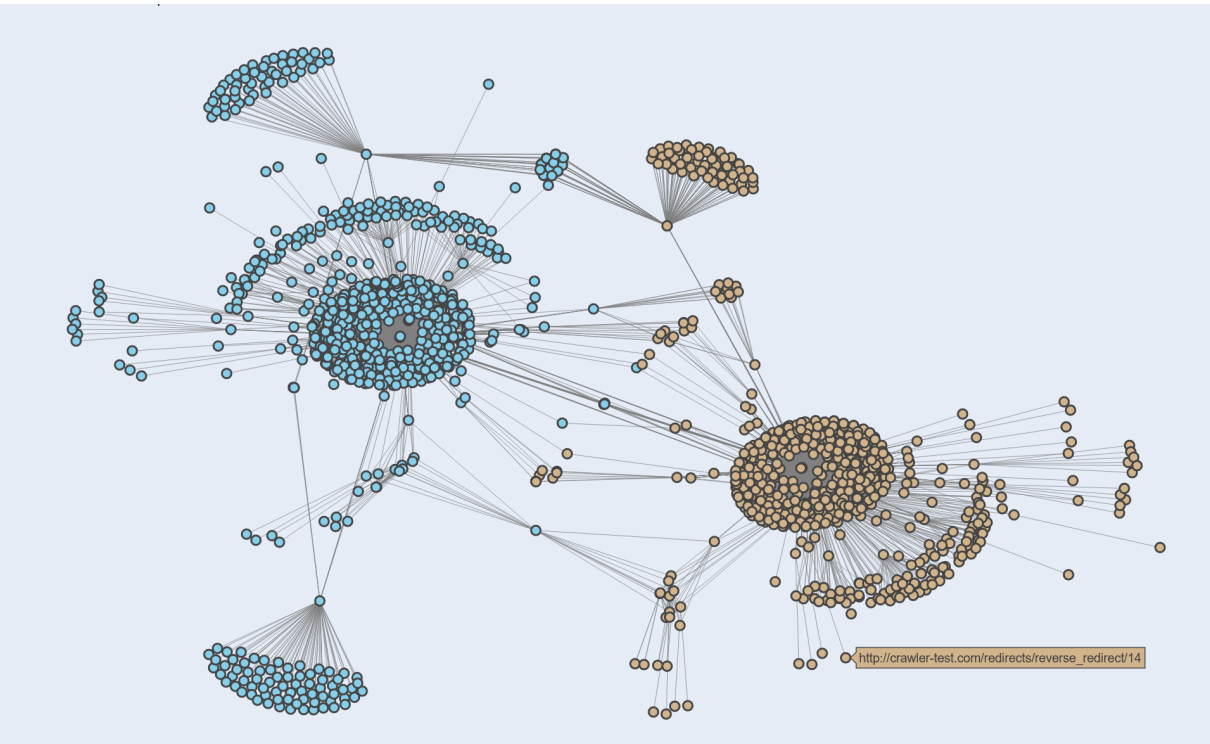
(a) Včetně všech hodnot **(b)** Detail podstatné části

Obrázek 6: Rozdělení proměnné času pro aktualizaci webového grafu. Zdroj: Vlastní

Podle sledovaných metrik je vidět, že čas dílčích operací crawleru pokrývá jen menší část celkového času crawlingu stránky, což je způsobeno rozhodnutím sledování důležitějších ukazatelů crawleru. Sledované metriky nezahrnují operace jako jsou interpretace JavaScript

kódu stránky, parsování obsahu stránky, zařazení odkazů do fronty, atd.

Po dokončení crawlingu v databázi Neo4j byla uložena grafová reprezentace struktury webu *https://crawler-test.com/*, která obsahovala veškeré URI dostupné z počátečního bodu a jejích propojení. Byl vytvořen Python skript pro vizualizaci uloženého grafu, kterou lze vidět na obrázku 7. Světle hnědá barva označuje URI se schématem *http://*, světle modrá barva označuje URI se schématem *https://*. Interaktivní verzi grafu pro otevření v prohlížeči lze najít v elektronické příloze práce.



Obrázek 7: Graf webu *crawler-test.com*. Zdroj: Vlastní

Následně byl spuštěn program pro aktualizaci hodnot PageRank v indexu Elasticsearch. Celkově aktualizace včetně výpočtu hodnot PageRank v databázi Neo4j a úpravy dokumentů v indexu Elasticsearch trvala přibližně 4 sekundy.

Po dokončení všech úprav indexu vyhledávače bylo spuštěno API pro testování vyhledávání. Na API byl poslán dotaz `/search?query=crawler` a byla obdržená odpověď s deseti výsledky, kterou lze vidět v tabulce v příloze A.

6.6 Shrnutí

V rámci práce byl realizován webový vyhledávač s web crawlerem, který nabízí REST API pro integraci s jinými systémy. Crawler vyhledávače byl napsán s využitím

vyššího programovacího jazyka C# a externích NuGet balíčku. Realizovaný systém vyžívá externí databáze jako jsou Elasticsearch, Redis a Neo4j pro ukládání webových stránek a synchronizaci distribuovaných crawlerů. Aplikační rámec ASP.NET Core byl požit pro zveřejnění API, které umožňuje získat výsledky hledání z indexu vyhledávače. Vyhledávač nabízí hledání dokumentů podle zadaného dotazu a zajišťuje relevantní řazení výsledků prostřednictvím využití PageRank skóre vypočteného v Neo4j.

Testování vyhledávače bylo provedeno pomocí webu *crawler-test.com*, přičemž crawler byl omezen pouze na doménu tohoto webu. Během 40 minut 5 instancí crawlerů navštívily veškeré nalezené stránky a uložily je do indexu. Následně byl spuštěn program pro výpočet a aktualizaci skóre PageRank pro každou stránku v indexu. Testovací vyhledávací dotaz na API úspěšně vrátil 10 výsledků hledání.

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat webový vyhledávač s web crawlerem. Realizovaný systém nabízí efektivní způsob sběru, indexace a vyhledávání webových stránek s využitím moderních technologií a programovacích nástrojů.

Práce zahrnovala podrobný přehled principů a mechanismů, které jsou klíčové pro fungování webových vyhledávačů. Byly prozkoumány a popsány hlavní komponenty typického vyhledávače – crawler, index a software pro vyhledávání. Detailní analýza web scraping technologií a strategií indexace poskytla základnu pro efektivní implementaci crawleru a indexovacího mechanismu.

Implementace vyhledávače byla provedena s využitím platformy .NET a aplikačního rámce ASP.NET Core. Vybrané databázové systémy, Elasticsearch a Neo4j, umožnily efektivní ukládání, zpracování velkých objemů dat. Databáze Redis umožnila synchronizaci distribuovaných crawlerů. Systém byl doplněn o REST API, což umožňuje jeho snadnou integraci s jinými aplikacemi a platformami.

Výsledky testování demonstrují schopnost systému efektivně procházet, indexovat a vyhledávat stránky v reálném čase. Testy ukázaly, že vyhledávač poskytuje rychlé a relevantní výsledky, což potvrzuje vysokou účinnost implementovaného řešení. Vyhledávač také úspěšně aplikuje algoritmy pro řazení stránek, zejména PageRank, což zajišťuje, že na vyšších pozicích se objevují kvalitnější a relevantnější webové stránky.

Tato práce tedy představuje kompletní řešení problematiky vyhledávání na webu s ohledem na současné požadavky a technologické možnosti. Navržený systém je sice vhodný pro nasazení a realizaci vyhledávání na zvoleném webu, ale neumožňuje opakované navštívení webových stránek a postrádá implementaci dalších funkcionalit.

Literatura

1. TARAKESWAR, Mr K; KAVITHA, D. Search engines: a study. *Journal of Computer Applications (JCA)*. 2011, roč. 4, č. 1, s. 29–33. ISSN 0974-1925.
2. YU, Wen-Jen; CHOU, Shrane Koung. A Bibliometric Study of Search Engine Literature in the SSCI Database. *J. Softw.* 2010, roč. 5, č. 12, s. 1317–1322. ISSN 1796-217X.
3. BRIN, Sergey; PAGE, Lawrence. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*. 1998, roč. 30, č. 1-7, s. 107–117. ISSN 0169-7552. Dostupné z DOI: 10.1016/S0169-7552(98)00110-X. Proceedings of the Seventh International World Wide Web Conference.
4. SULLIVAN, Danny. *How search engines work* [online]. 2002. [cit. 2024-03-04]. Dostupné z: <http://tv-prod.s3.amazonaws.com/documents/2515-How%20Search%20Engines%20Work.pdf>.
5. ZHAO, Bo. Web scraping. *Encyclopedia of big data*. 2017, roč. 1. ISBN 978-3-319-32001-4.
6. BANERJEE, Ritu. *Website Scraping* [online]. 2014. [cit. 2024-03-04]. Dostupné z: <https://www.happiestminds.com/whitepapers/website-scraping.pdf>.
7. NAJORK, Marc. *Web Crawler Architecture* [online]. 2009. [cit. 2024-03-04]. Dostupné z: <https://marc.najork.org/papers/eds2009a.pdf>.
8. PERSSON, Emil. *Evaluating tools and techniques for web scraping*. 2019. Dostupné také z: <https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-271206>. Dis. pr. School of Electrical Engineering a Computer Science (EECS).
9. JHA, Pooja; GOYAL, Soni; KUMARI, Tanya; GUPTA, Neha. Robots exclusion protocol. *International Journal of Emerging Science and Engineering*. 2014, roč. 2, č. 5. ISSN 2319–6378.
10. FEJFAR, Petr. *Interactive web crawling and data extraction*. 2018. Dostupné také z: <http://hdl.handle.net/20.500.11956/103482>. Dipl. pr. Univerzita Karlova, Matematicko-fyzikální fakulta.

11. MAHAPATRA, Ajit Kumar; BISWAS, Sitanath. Inverted indexes: Types and techniques. *International Journal of Computer Science Issues (IJCSI)*. 2011, roč. 8, č. 4, s. 384. ISSN 1694-0814.

12. MONGODB, INC. *Full-Text Search: What Is It And How It Works* [online]. [cit. 2024-03-04]. Dostupné z: <https://www.mongodb.com/basics/full-text-search>.

13. TRUICA, Ciprian-Octavian; BOICEA, Alexandru; RADULESCU, Florin. Building an inverted index at the dbms layer for fast full text search. *Journal of Control Engineering and Applied Informatics*. 2017, roč. 19, č. 1, s. 94–101. ISSN 1454-8658.

14. SHAH, Dharmish; SOMAIYA, Jheel; NAIR, Sindhu et al. Fuzzy semantic search engine. *International Journal of Computer Applications*. 2014, roč. 975, s. 8887. ISSN 0975-8887.

15. SHARMA, Prem Sagar; YADAV, Divakar; GARG, Pankaj. A systematic review on page ranking algorithms. *International Journal of Information Technology*. 2020, roč. 12, č. 2, s. 329–337. ISSN 2511-2112.

16. SINGH, Ashutosh Kumar; P, Ravi Kumar. A Comparative Study of Page Ranking Algorithms for Information Retrieval. *International Journal of Computer and Information Engineering*. 2009, roč. 3, č. 4, s. 1154–1165. ISSN 1307-6892. Dostupné také z: <https://publications.waset.org/vol/28>.

17. PAGE, Lawrence; BRIN, Sergey; MOTWANI, Rajeev; WINOGRAD, Terry. The PageRank Citation Ranking : Bringing Order to the Web. In: *The Web Conference*. 1999. Dostupné také z: <https://api.semanticscholar.org/CorpusID:1508503>.

18. AL-OBAYDY, WN Ibrahem; HASHIM, Hala A; NAJM, YA; JALAL, Ahmed Adeeb. Document classification using term frequency-inverse document frequency and K-means clustering. *Indonesian Journal of Electrical Engineering and Computer Science*. 2022, roč. 27, č. 3, s. 1517–1524. ISSN 2502-4752.

19. LÓPEZ, José Antonio Hernández; CUADRADO, Jesús Sánchez. An efficient and scalable search engine for models. *Software and Systems Modeling*. 2022, roč. 21, č. 5, s. 1715–1737. ISSN 1619-1374.

20. GOODWIN, Michael. *What is an application programming interface (API)?* [online]. 2024. [cit. 2024-03-12]. Dostupné z: <https://www.ibm.com/topics/api>.

21. SURWASE, Vijay. REST API modeling languages-a developer's perspective. *Int. J. Sci. Technol. Eng.* 2016, roč. 2, č. 10, s. 634–637. ISSN 2349-784X.

22. SELENIUM. *WebDriver | Selenium* [online]. [cit. 2024-04-13]. Dostupné z: <https://www.selenium.dev/documentation/webdriver/>.

23. ELASTIC NV. *Elasticsearch: The Official Distributed Search & Analytics Engine | Elastic* [online]. [cit. 2024-04-14]. Dostupné z: <https://www.elastic.co/elasticsearch>.

24. MICROSOFT. *ASP.NET Core | Open-source web framework for .NET* [online]. 2024. [cit. 2024-04-15]. Dostupné z: <https://dotnet.microsoft.com/en-us/apps/aspnet>.

Přílohy

Příloha A: Tabulka s výsledky vyhledávání

docId	uri	title	pageRank	totalScore	highlights
CqW9I...	https://crawler-test.com/	Crawler Test Site	80.843	4.511144	Crawler Test two ...
BKXDI...	http://crawler-test.com/	Crawler Test Site	80.019	4.511144	Crawler Test two ...
gqXMI...	https://crawler-test.com/?parameter-on-hostname-root=parameter-value	Crawler Test Site	0.32012	4.511144	Crawler Test two ...
o6XCI...	https://crawler-test.com/?amp;retained_parameter=1&removed_parameter=1	Crawler Test Site	0.32012	4.511144	Crawler Test two ...
TqXGI...	http://crawler-test.com/redirects/redirect_content	Crawler Test Site	0.31969	4.511144	Crawler Test two ...
q6XII...	http://crawler-test.com/other/crawler_request_headers	Crawler Request Headers	0.31969	4.511144	Crawler Test ...
SKXGI...	http://crawler-test.com/?parameter-on-hostname-root=parameter-value	Crawler Test Site	0.31839	4.511144	Crawler Test two ...
LaXLI...	http://crawler-test.com/?amp;retained_parameter=1&removed_parameter=1	Crawler Test Site	0.31839	4.511144	Crawler Test two ...
y6XCI...	https://crawler-test.com/redirects/redirect_content?parameter-on-hostname-root=parameter-value	Crawler Test Site	0.15130	4.511144	Crawler Test two ...
5qXKI...	https://crawler-test.com/redirects/redirect_content?amp;retained_parameter=1&removed_parameter=1	Crawler Test Site	0.15130	4.511144	Crawler Test two ...

Zdroj: Vlastní