Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

UPOL Search – webový vyhledávač pro doménu upol.cz

2018

Bc. Tomáš Mikula

Vedoucí práce:
Mgr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Bc. Tomáš Mikula
Název práce:	UPOL Search – webový vyhledávač pro doménu upol.cz
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2018
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	Mgr. Martin Trnečka, Ph.D.
Počet stran:	38
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Bc. Tomáš Mikula
Title:	UPOL Search – search engine for upol.cz domain
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2018
Study field:	Applied Computer Science, full-time form
Supervisor:	Mgr. Martin Trnečka, Ph.D.
Page count:	38
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Výsledkem této práce je vyhledávací engine UPOL search, který realizuje vyhledávání nad doménou upol.cz. Součástí práce je webový crawler, který prochází stránky v zadané doméně, indexační nástroj a samotný webový vyhledávač zprostředkovávající výsledky vyhledávání. Všechny zmíněné části jsou speciálně optimalizovány pro doménu upol.cz. Součástí práce je rovněž microformát (metadata), který je přímo určen pro UPOL search a zohledňuje běžné vyhledávané záležitosti jako jsou číslo kanceláře zaměstnance, informace o předmětech a podobně.

Synopsis

The result of this thesis is a search engine UPOL Search which implements searching on domain upol.cz. Search engine consists of web crawler which goes through the pages in the given domain, indexation tool and the web search engine realizing the results of the search. All mentioned parts are especially optimized for the domain upol.cz. Part of this thesis is also a microformat (metadata) which is directly determined for UPOL search and takes into account commonly searched terms like number of the office doors, information about subjects etc.

Klíčová slova: vyhledávač; crawler; index; metadata

Keywords: search engine; crawler; index; metadata

Děkuji vedoucímu diplomové práce Mgr. Martinu Trnečkovi, PhD., dále Ivu Fridrichovi, za technické zázemí a pomoc s nastavením serverů. Rovněž děkuji za konzultace Martinu Kirschnerovi (Seznam.cz) a Davidovi Kopeckému (Apple).

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Motivace

V první kapitole se budu věnovat motivaci pro vytvoření vlastního vyhledávače pro účely vyhledávání nad cílovou doménou. Pod takovou doménou si můžeme představit veřejnou, nebo privátní síť ve které vznikla potřeba vyhledávat informace. Může se jednat například o firemní síť, nebo v mém případě univerzitní síť.

Jako první očividnou motivací může být zjištění stavu sítě. Kolik se na dané doméně vyskytuje serverů, kolik obsahuje PDF dokumentů, jak rychlá je odezva serverů a mnoho dalšího. Mapování sítě je u použití vlastního vyhledávače spíše vedlejší produkt, protože existují specializované nástroje, které jsou pro monitorování sítě přímo navrženy.

Omezení se na určitou doménu přináší mnoho výhod. Celý vyhledávač můžeme značně přizpůsobit obsahu a struktuře domény a celkový návrh vyhledávače je jednoduší než v případě celého Internetu, kde můžeme narazit na mnoho nečekaných problémů. V menší doméně, kterou máme pod částečnou kontrolou, nemusíme řešit například snahy o manipulaci s výsledky ve vyhledávači. S podobnými problémy se musí společnosti jako DuckDuckGo [1], Google, nebo Seznam.cz, které provozují internetové vyhledávače, zabývat ve velké míře.

1.1 Přizpůsobení vyhledávače na doménu

Jako první zřejmá výhoda vlastního vyhledávače nad doménou je možnost plánování času a intenzity sběru potřebných dat ze sítě. Vyhledávače operující na celém Internetu musí být šetrné, protože neví kdy mohou server zatížit a kdy naopak nesmí. Pro řešení tohoto problému existuje několik technik. Žádná z těchto technik se ale nemůže vyrovnat přímé znalosti topologie sítě. Tato znalost vede k možnosti rychlého a šetrného sběru dat při použití relativně slabého hardwaru.

Další oblast přizpůsobení se týká obsahu samotné sítě. Pokud se v síti vyskytuji exotické dokumenty, které komerčně používané vyhledávače neumí rozpoznat, jedná se o ideální příležitost pro nasazení vlastního vyhledávače.

V neposlední řade můžeme navrhnout a implementovat vlastní metadata (microformat) pro potřeby dané domény. Běžné vyhledávače nám neumožňují pohodlně hledat informace o zaměstnancích ve firmě, nebo informace o předmětech vyučovaných na univerzitě. Tento problém lze elegantně vyřešit implementací vlastního microformátu a jeho integrací do sítě. V případě implementaci vlastního microformátu je třeba zvážit složitost nasazení tohoto formátu do již existujících dokumentů/webů.

2 Teoretická struktura webového vyhledávače

V druhé kapitole nastíním základní strukturu obecného fulltextového webového vyhledávače. Vyhledávač se skládá ze tří základních částí: *crawler*, *indexer* a *výdej*. Tuto strukturu respektuje i mnou navržený vyhledávač UPOLSearch. Odlišnostem a podrobnějším informacím ohledně praktické implementace se budu věnovat později v sekci 3.

2.1 Crawler

Crawler je součást vyhledávače starající se o sběr veškerých potřebných dat ze sítě. Jedná se o jednoduchý nástroj (viz pseudokód 1), který prochází síť skrze dostupné HTML (HyperText Markup Language) odkazy. Crawler se skládá z následujících částí: fronta, požádání o webovou stránku, stažení webové stránky, zpracování staženého dokumentu a přidání nových odkazů do fronty. V praktické implementaci crawleru se však narazí na poměrně komplexní problémy, které nastíním v jednotlivých částech.

Algorithm 1 Crawler

```
1: procedure CRAWL
2:     queue ← load_seed()
3:     while queue is empty do
4:         url ← dequeue_url()
5:         document ← download_document(url)
6:         links ← parse_document_for_links(document)
7:         queue.add_links(links)
```

2.1.1 Fronta

Fronta obsahuje nenavštívené odkazy, které musí crawler navštívit a zpracovat. Největším realizačním problémem, je velikost fronty a zamezení duplicit. V praxi fronta narůstá rychleji než je konzumována. Na nové webové stránce (z fronty byl odebrán jeden odkaz) nalezne vyhledávač většinou desítky nových odkazů (do fronty budou přidány desítky nových odkazů). Tento problém je řešen dostatečně velikou pamětí, případně odkládání částí fronty na pevný disk. Jeden z požadavků na frontu je rovněž její rychlost. Cílem je rychlé přidání a odebrání adres z fronty, rovněž je nutné efektivně detekovat duplicity. Fronta je proto často uložena v RAM paměti serveru, případně na poli SSD disků.

2.1.2 Žádost o webovou stránku

V následující částí crawler odebral jednu adresu z fronty a zažádá server o její stažení. Obecně platí, že zodpovědnost za ohleduplný přístup k serveru je na straně crawleru. Crawler by měl být schopný sám posoudit, zda může v danou

chvíli požadavek na daný server zaslat, nebo zda je vůbec oprávněn tuto adresu navštívit. Řešením těchto problémů se budu věnovat v praktické implementaci (sekce 3), zda však pouze nastíním, že první problém může řešit vnitřní plánovač crawleru a druhému problému může pomoci samotný administrátor webové stránky skrze použití souboru `robots.txt` a crawler jeho respektováním.

2.1.3 Stažení webové stránky

Na první pohled jednoduchý krok. Crawler však může uváznou na *timeoutu* (doba, která nesmí být překročena při čekání na událost). Rozlišujeme mezi dvěma typy timeoutů. *Connection timeout*, který udává kolik času (typicky něco okolo 3.05s, založeno na TCP packet retransmission window – doba za kterou bude spojení opakováno) bude crawler čekat na navázání spojení a *Read timeout* (typicky něco okolo 10s), neboli čas, který bude crawler čekat na přenos dat. V případě, že nastane timeout se musíme rozhodnout jak budeme postupovat. Tedy zda chceme URL (Uniform Resource Locator) adresu úplně zahodit, nebo se ji pokusit navštívit později případně kolikrát se o to budeme pokoušet, než URL adresu zahodíme?

Dalším problémem na který může crawler narazit je přesměrování. V případě přesměrování se nebavíme pouze o HTTP (Hypertext Transfer Protocol) odpovědi 301, ale i o URL adresách se stejným obsahem, například `http://inf.upol.cz` a `http://inf.upol.cz/` (přidané koncové lomítko). Tyto adresy spojujeme do takzvaných *kanonických množin*. Každá kanonická množina je potom zastoupena jednou URL adresou.

Na úrovni stažení webové stránky rovněž probíhá detekce typu obsahu. První naivní způsob detekce je predikce typu souboru z URL adresy. Tento způsob je poměrně efektivní například při predikci obrázkových souborů. V praxi se však vyskytují URL adresy bez očekávané přípony, které však obsahují obrázky. Druhá možnost je stahovat veškerý obsah a lokálně provádět detekci. Tento způsob je velice spolehlivý, ale časově náročný. Rozumným kompromisem je důvěřovat `Content-Type` z hlavičky HTTP odpovědi serveru. Stahování dokumentu tedy probíhá prvotním stažením HTTP hlavičky, detekcí typu obsahu, a případného stažení celého obsahu.

2.1.4 Zpracování staženého souboru

Pokud je stažený soubor HTML dokument, je jeho obsah zpracován za pomoci standardního HTML parseru (například lxml [2]). Následně jsou z dokumentu extrahovány veškeré odkazy. Před samotnou extrakcí odkazů je nutné kontrolovat zda nám stránka povoluje extrakci odkazu. Je vhodné respektovat HTML meta tagy, konkrétně meta tag `<meta name="robots"content="nofollow">`, který zakazuje následování jakýchkoli odkazů na aktuální stránce. Dále je nutné respektovat atribut `rel`, který stejné omezeni stanovuje pro jednotlivé odkazy na stránce, například `<a href="signin.php"rel="nofollow">Sign In</a>`. Odkazy je poté nutné validovat, odstranit nežádoucí kandidáty (například URL adresy na jiné

domény než je doména cílová) a eliminovat duplicity. Duplicity jsou odstraněny na základě výpočtu kontrolního součtu hash z URL adresy a následně porovnány s již navštívenými adresami a adresami ve frontě. Rovněž je potřeba vyřešit obsahové duplicity. Toho docílíme výpočtem (v ideálním případě) podobnosti obsahu s obsahem již navštívených odkazů a v případě nalezené duplicity spojit adresy do kanonické množiny.

2.1.5 Vložení získaných adres do fronty

Posledním krokem crawleru je vložení získaných a validovaných adres do fronty. Pokud je fronta reprezentována databází je vhodné jejich vložení provádět po větších kvantech, databáze taková kvanta umí zpracovat efektivněji.

2.2 Ohodnocení dokumentů

Dokumenty, které byly shromážděny crawlerem je nutné nějakým způsobem ohodnotit. Toto ohodnocení je později použito výdejem k relevantnímu seřazení výsledků. Dokumenty lze ohodnotit na základě několika možných kriterií. Mezi základní a nerozšířenejší způsob ohodnocení dokumentů v grafu patří *PageRank* [3]. U každého dokumentu jsou uloženy odchozí odkazy na ostatní dokumenty. Vzniká tedy orientovaný graf všech dokumentů, nad kterým lze PageRank počítat. Ve zjednodušené verzi lze vypočítat Pagerank dokumentu funkcí 1 (pro více informací čtenáře odkazuji na článek pojednávající o PageRanku [3]).

$$R(u) = c \sum_{v \epsilon B_u} \frac{R(v)}{N_v}, \tag{1}$$

Kde:

u = aktuální dokument,
B_u = dokumenty ukazující na dokument u,
v = dokument odkazující se na u,
N_v = počet odchozích odkazů z dokumentu v,
u = faktor použitý pro normalizaci,
$R(v)$ = rank dokumentu v.

Samotný PageRank však není všemocný. Pro řazení dokumentů jsou používány i další metriky. Mezi nejznámější patří stáří dokumentu (doba od poslední změny obsahu dokumentu), délka obsahu dokumentu, informace zda dokument obsahuje metadata, struktura dokumentu (v případě HTML jeho validita – dle W3C [4]), pravopisné chyby a další. Informace o tom, jaká všechna kriteria jsou brána v potaz nejsou veřejně dostupné. Webové vyhledávače se snaží tyto metriky neprozrazovat, aby nedocházelo k jednoduché manipulaci s pořadím výsledků ve vyhledávání.

2.3 Indexer

Pokud máme všechny dokumenty staženy, rozřazeny do kanonických množin a vypočítán PageRank, je na čase je vložit do indexu. Do indexu jsou vkládány pouze relevantní dokumenty, například adresy s odpovědí 404 (obsah nenalezen) do indexu nevkládáme. Proces indexace a potřebu indexu zde pouze okrajově nastíním. V praktické implementaci, jak bude uvedeno později, využívám full-textový systém databáze PostgreSQL [5]. Při popisu indexace se budu držet vnitřní struktury indexeru v této databázi PostgreSQL.

Prvním krokem je tokenizace vstupního textu dokumentu. Token je řetězec s přiřazeným a identifikovaným významem. K tomuto kroku PostgreSQL využívá vlastní implementaci parseru. Pro index je důležité rozeznávat mezi různými druhy tokenů, například emailové adresy, slova, číslice, adresy a další.

Dalším krokem je konvertování tokenů na *lexémy*. Lexém je řetězec stejně jako token, ale již proběhla jeho normalizace, neboli stejné slovo v různém tvaru bude zastoupeno jedním lexémem. Jako příklad normalizace si můžeme představit změnu velkých písmen na malá a odstranění předložek a přípon. Tento krok indexeru zjednoduší samotné vyhledávání, není totiž nutné uchovávat veškeré možné tvary slova, které uživatel může požadovat, ale ukládat pouze základní tvar slova. Uživatelův požadavek je následně na tento základní tvar vždy převeden. V tomto kroku jsou rovněž odstraněna *stop slova*, neboli slova, která nenesou žádný význam, například spojky. Například ve větě „Auta a letadla.“ bude odstraněna spojka „a“. Pro tento krok se většinou používá seznam stop slov pro daný jazyk dokumentu.

Posledním krokem je uložení zkrácené reprezentace daného dokumentu. Dokument je jednoznačně reprezentován polem lexémů. Rovněž je užitečné ukládat poziční informaci o daném lexému v dokumentu, neboli pozici na které se původní slovo vyskytovalo. Některé přístupy k vyhledávání umí využít informaci o nahuštěnosti určitých pojmů v dokumentu k jejich relevantnějšímu seřazení.

Jakmile jsou veškeré dokumenty uložené je možné provádět vyhledávání. Vyhledávání nad takovou reprezentací dokumentů je rychlejší než naivní vyhledávání v originálních textech dokumentů, jeho výkon však můžeme rapidně zvýšit vybudování indexu nad reprezentací prohledávaných dokumentů. Takový index si můžeme představit podobně jako rejstřík na konci knihy (příklad uváděný v oficiální dokumentaci [6]). Důležité pojmy jsou zde seřazeny podle abecedy a u každého pojmu máme uvedenou stránku na které se nachází (neboli pozici v databázi). Tento rejstřík čtenáři umožňuje rychle najít daný pojem v knize, vyžaduje však nějaké místo (stránky) v knize samotné. V databázových systémech je tomu podobně. Index umí zrychlit samotný požadavek na vyhledáváni, je však nutné ho někde uchovat. Při velkém počtu dokumentů to může pro databázi znamenat velkou zátěž. Index je vytvořen tím způsobem, že databázový systém analyzuje veškeré dokumenty, které chceme indexovat a vytvoří si interní index (rejstřík). Jakmile je dotázán na určitý pojem, pokusí se termín vyhledat v indexu, zjistí kde se v databázi tyto informace vyskytují a následně vrátí odpověď. Z tohoto postupu vidíme, že je celý proces rychlejší, než kdyby měl databázový

systém vždy prohledat veškeré (ač už zpracované) dokumenty.

2.4 Výdej

Výdej se stará o několik podstatných funkcí vyhledávače. Jeho první částí je webový server, který skrze formulář získá požadavek od uživatele. Následuje analýza požadavku od uživatele. V této části musí nastat minimálně převedení dotazu na lexémy. V pokročilých případech dochází k pochopení uživatelova dotazu. Pochopení dotazu může být následně použito ke zvýšení relevance poskytnutých výsledků. Technologii, která umožňuje pochopit uživatelův dotaz se nebudu v textu více věnovat, jelikož mnou vytvořený vyhledávač neobsahuje část, která by se starala o inteligentní pochopení dotazu od uživatele.

Dalším krokem je najít odpovídající výsledky v indexu, o tento krok se většinou stará samotná databáze. Nalezené výsledky je nutné rovněž seřadit a to hned dle několika kriterií. Hlavním z nich je již zmíněný PageRank, dále jsou do řazení promítnuta kritéria již zmíněná v kapitole 2.2. V případě použití metadat, jsou k výsledkům rovněž připojena relevantní metadata.

S výdejem jsou rovněž spojeny technické problémy jako například distribuce zátěže serveru a další.

3 Realizace fulltextového vyhledávače UPOLSearch

V následující hlavní sekci se budu věnovat popisu samotné realizace fulltextového vyhledávače UPOLSearch. V první části budou zmíněny veškeré důležité použité technologie, knihovny a volba programovacího jazyka. Dále se budu věnovat popisu realizace jednotlivých částí vyhledávače, které byly dosud popsány pouze teoreticky. V poslední části se budu věnovat nasazení vyhledávače do produkce, hardwar nárokům, monitoringu a dalším aspektům.

3.1 Volba technologií

3.1.1 Programovací jazyk

S původním plánem vytvoření prototypu crawleru byl zvolen programovací jazyk Python 3 (Python 3.6), následným otestováním jeho výkonu a přepsání do kompilovaného jazyka. Python je poměrně jasnou volbou s ohledem na parsování webu a jakékoli zpracování dat. Disponuje širokou uživatelskou základnou a nesčetným množstvím knihoven. Jeho nevýhodou je však nižší výkon a neefektivní paralelizace. Jelikož se jedná o jazyk interpretovaný, veškerá paralelizace je natolik efektivní jako rychlost jednoho interpretru. Během vývoje se však ukázalo, že Python je jazykem dostatečně výkonným. Především pak díky volbě podpůrných knihoven pro jeho efektivní paralelizaci.

3.1.2 Databázové technologie

Správný výběr databázové technologie je důležitý pro stabilní a správný chod vyhledávače jako celku. Pro vyhledávač UPOLSearch jsem použil následující dvě databáze.

PostgreSQL 10. Jedná se o objektově relační databázový systém vydaný pod open-source licencí MIT. PostgreSQL byl vybrán ze dvou důvodu. V první řadě se jedná o svobodně dostupný software, který je navíc hojně využíván v produkčním prostředí. Druhým důvodem je již popsaný fulltextový vyhledávací systém, kterým PostgreSQL disponuje. Na poli fulltextových nástrojů se vyskytují modernější a vhodnější řešení jako například Elasticsearch [7]. PostgreSQL byl však vybrán na základě jeho dobrého poměru jednoduchosti a stabilního nasazení do produkce vůči kvalitě fulltextového vyhledávání ve dvou jazycích (český a anglický jazyk).

MongoDB 3.6. Je flexibilní a škálovatelná open-source dokumentová NoSQL databáze se zaměřením na velká data a rychlost [8]. Data (dokumenty) jsou uloženy ve formátu podobném JSON (JavaScript Object Notation), což přináší možnost aby se struktura dokumentů lišila a průběžně byla měněna. MongoDB

rovněž podporuje komplexní dotazy, indexaci a realtime agregaci. Škálovatelnost databáze se projevuje v možnosti distribuce datasetu mezi několika instancemi databáze.

3.1.3 Ostatní technologie

RabbitMQ 3.7. Je open-source message broker, který slouží k realizaci mnoha asynchronních systémů [9]. RabbitMQ podporuje několik protokolů zpráv, spravování až několik front zpráv, zpětné potvrzení příjmu zprávy a mnoho dalšího. RabbitMQ bylo použito jako hlavní distributor zpráv mezi jednotlivými částmi UPOLSearch vyhledávače. Jedná se o jednu z podporovaných technologií knihovnou Celery, která bude zmíněna níže.

3.1.4 Použité Python knihovny

Celery 4.1.0. Je asynchronní distribuovaná fronta úloh pro jazyk Python [10]. Její hlavní zaměření jsou realtime operace, podporuje však i plánování úloh. Jednotlivé úlohy jsou zpracovány takzvanými workery. Workery podporují paralelní výpočty úloh a to na jednom nebo více serverů. Celery podporuje mnoho message brokerů (sloužící k předávání zpráv mezi producentem a workery), UPOLSearch používá již zmíněný RabbitMQ message broker.

V sekci o výběru jazyka 3.1.1 jsem zmiňoval problém paralelních výpočtu v jazyce Python. Celery poskytuje plnohodnotnou paralelizaci pro jazyk Python. Každý worker je samostatně běžící instancí Python programu. To zajistí neblokovanou plnohodnotnou paralelizaci. V praxi se ukázalo, že rychlost získaná za pomoci Celery paralelizace je více než potřebná a ve výsledku se musela limitovat (více v sekci pojednávající o výkonu 4.1).

Flask. Jedná se o microframework pro vývoj webových aplikací a API. V porovnaní s Django Frameworkem, který je druhou populární alternativou pro jazyk Python, je jeho hlavním rysem jednoduchost a přímočarost. Flask využívá šablonovací systém Jinja2 [11], který nabízí standardní funkce šablonovacího systému. Pro nasazení do produkce lze použít například Apache2 s `mod_wsgi` (modul pro Apache2 [12]). V rámci UPOLSearch byl použit na celou část výdeje a API pro monitorování chodu crawleru.

Requests. Je hlavní HTTP knihovnou pro jazyk Python. Knihovna je navržena s ohledem na jednoduchost a bezpečnost použití. HTTP požadavky jsou základem celého crawleru, proto bylo nutné vybrat jednoduchou a bezpečnou knihovnu pro jejich bezchybnou realizaci.

Beautifulsoup4 + lxml. Tyto dvě knihovny realizují parsování HTML dokumentů v celém projektu. Použití našly jak v crawleru tak v částí zpracování dokumentů před indexací. BeautifulSoup je knihovna podporující několik parseru

včetně lxml, poskytuje tak uživatelsky jednoduší nástroje při zachování rychlosti nízkourovňových parserů.

PDFminer. Je knihovna pro parsování PDF dokumentů. V UPOLSearch je použita pro extrakci textu z PDF dokumentů. Jedná se o poměrně pomalou implementaci PDF parseru, především pak u rozsáhlých dokumentů.

Langdetect. Jedná se o port Java knihovny `language-detection` od firmy Google do jazyka Python. V rámci projektu je použita pro veškerou detekci jazyka dokumentů. Knihovna rovněž trpí nízkým výkonem, především pak na dlouhých textech, proto jsem u dlouhých dokumentů testoval pouze část dokumentu.

Reppy. Jednoduchá knihovna pro práci s `robots.txt`. Podporuje použití cache, tím je značně snížena zátěž na cílové servery.

NetworkX. Knihovna pro vytváření, manipulaci a studování struktury, dynamiky a funkcí komplexních grafů. Jedná se o rozsáhlou knihovnu pro práci s grafy, která podporuje standardní algoritmy. V rámci UPOLSearch byla použita pro sestavení grafu sítě a následného výpočtu PageRank.

JsonSchema. Je implementace JSON schema pro jazyk Python. Slouží k jednoduché validaci a anotaci JSON dat. V rámci UPOLSearch je použita pro validaci metadat.

3.2 Adresářová struktura projektu

Struktura projektu dodržuje standarty Python balíčku, včetně souboru pro jeho instalaci `setup.py`. Podrobnější struktura aplikace je popsána v tabulce 1. Během vývoje bylo experimentováno s několika rozděleními projektu. Od separátních projektů pro jednotlivé částí vyhledávače (jednoduchá distribuce mezi více serverů) až po jeden monolitický projekt, který usnadnil některé implementační záležitosti a redukoval duplicity v kódu.

Tabulka 1: Adresářová struktura projektu

Adresář	Popis obsahu
`upol_search_engine/`	Hlavní podsložka projektu, kořenová složka aplikace.
`./db`	Nástroje a funkce pro práci s databázemi PostgreSQL a MongoDB.
`./upol_crawler`	Veškeré funkce crawleru.
`./upol_indexer`	Veškere funkce indexeru, včetně definice microformatu.
`./upol_search_engine`	Flask aplikace výdeje, včetně veškerých šablon a stylů.
`./utils`	Složka pomocných nástrojů, které jsou využiti napříč aplikací. Obsahuje například pomocé funkce pro práci s URL adresami a dokumenty.

3.3 Crawler

V rámci UPOLSearch jsem implementoval vlastní crawler, tento přístup se ukázal jako vhodný, především s ohledem na optimalizaci vyhledávače na doménu `upol.cz`. V komunitě Python jazyka se vyskytují hotová řešení, mezi které patří například nejznámější Scrapy framework [13] určený pro extrakci dat z webu. Problém Scrapy frameworku spočívá v jeho komplexnosti, integruje totiž samotný crawler a zpracování (parsování) dat do jednoho frameworku, proto jsem z důvodu větší přehlednosti a oddělenosti volil možnot vlastní impementace. Scrapy však posloužil jako vhodná inspirace při vlastní impementaci.

Crawler přijíma na vstup množinu URL adres zvanou *seed* a jeho výstupem je množina dokumentů v databázi MongoDB.

Crawler se na Internetu identifikuje za pomoci speciálního `user-agent` označení `Mozilla/5.0 (compatible; UPOL-Search-Engine VERZE; URL)`. Toto označení se uvádí pro jednoduchou identifikaci crawleru v Internetu. URL adresa projektu je důležitá v případě nutnosti kontaktovat autora crawleru. Tento přístup není nutný, jedná se spíše o slušnost a transparentnost. Crawler rovněž může vystupovat jako běžný uživatel s prohlížečem, zvyšujeme však šanci, že bude bezdůvodně zablokován.

Feeder

Feeder (krmič) se stará o plnění fronty crawleru novými odkazy. Nalezené odkazy jsou vložené do MongoDB databáze odkud je vyzvedává feeder. Feeder rovněž nastavuje startovní stav crawleru za pomoci importu počátečního seedu. Kr-

mení fronty probíhá v dávkách. Velikost dávky a frekvenci krmení lze nastavit v konfiguračním souboru UPOLSearch. S velikostí fronty uložené v RabbitMQ vzniká problém s náročností na paměť RAM. Z tohoto důvodu jsou vložené úlohy komprimovány algoritmem *zlib*.

Limiter

Limiter, neboli plánovač slouží k limitování zátěže crawlované sítě. Crawler v původní verzi limiter neobsahoval, což vedlo k nežádoucí zátěži serverů. V praxi se může stát, že pokud crawler neobsahuje plánovač bude stále dokola navštěvovat stejnou doménu. Crawlování je standardně prováděno na síti, ve které se vyskytuje několik serverů/domén. V případě správného rozvrhnutí zátěže, je možné procházet obsah sítě stejnou rychlostí s menší zátěži. Jelikož crawler běží paralelně v několika vláknech je důležité hlídat aby crawler v jeden okamžik stahoval dokumenty z několika různých serverů. Limiter kontroluje jak často může crawler navštívit konkrétní IP adresu serveru (například 1× za sekundu). Limit je důležité dodržovat vůči IP adrese serveru, nikoli doméně.

Pokud se crawler chystá zahájit stahování obsahu z určité URL, zeptá se limiteru, kdy naposledy tuto IP adresu navštívil. Pokud limiter zjistí, že od poslední návštevy uběhl požadovaný čas, povolí crawleru navštívení URL adresy, v opačném případě je URL adresa zařazena na konec fronty. Z důvodu nezávislosti jednotlivých Celery workerů je limiter realizován v MongoDB databázi a nikoli sdílen mezi procesy.

V průběhu crawlováni limiter nijak významně proces nezpomaluje. Problém nastává ke konci crawlování sítě. Ve frontě adres se již nevyskytuje dostatečné množství unikátních serverů, což vede ke zpomalení crawleru (crawler musí více čekat než stahovat). Jak se ale dozvíme později, tento fakt nemá na celkový výkon velký vliv. Vedlejší produkt limiteru je statistika poměru domén ku serverům v rámci skenované sítě (uvedené v sekci 4.2).

V kontextu limiteru rovněž můžeme mluvit o chování crawleru v případě timeoutu. UPOLSearch se k timeoutům zachovává podobně jak bylo řečeno v sekci 2.1.3. Adresy, které nebyly navštíveny z důvodu timeoutu nejsou znovu zařazeny do fronty a je nutné jejich opětovné stažení při dalším naplánovaném průchodu sítě.

Kanonické množiny

Kanonické množiny slouží k sjednocení URL adres se stejným obsahem. Kanonická množina je reprezentována jedním zástupcem, který je indexován a vracen výdejem uživateli. Duplicity jsou detekovány na základě vypočítané hash funkce z obsahu stránky. V první verzi byl hash počítán z kompletního HTML kódu stránky. Tento přístup projevil několik nedostatků, proto byl implementován postup, který z webové stránky extrahuje veškerý smysluplný viditelný text a hash je vypočítán nad tímto textem. Z HTML kódu jsou odstraněny tagy script, form, style, tagy obsahující styl hidden a další. Tento přístup již poskytoval kvalitnější

detekci duplicitních URL adres. V konečné verzi musel být však aplikován speciální filtr na detekci duplicit u specifických stránek, kde se v průběhu času mění textový obsah. Takovým příkladem může být například phpBB [14] diskuzní fórum, na kterém se v dolní části webu zobrazuje počet online uživatelů.

Kanonické množiny jsou rovněž důležité pro přesný výpočet PageRanku. Pokud crawler nesprávně detekuje duplicity, nachází se v grafu sítě více uzlů, což vede ke špatné distribuci PageRanku v grafu.

File-type detection

Jak už bylo v sekci 2.1.3 naznačeno, detekci typu souboru provádím na základě hodnoty `Content-Type` v hlavičce odpovědi HTTP serveru. Během fáze stažení URL adresy nejprve proběhne žádost o hlavičku, a poté o celkový obsah URL adresy.

Link extractor a Validator

Část parseru sloužící k nalezení veškerých odkazů v daném dokumentu. V prvním kroku probíhá detekce typu stránky. Pro urychlení celého procesu crawleru aplikuji ruční filtry extrakce odkazů na dva typy webových stánek. Jedná se o phpBB diskuzní forá a wikipedia stránky. Tyto systémy obsahují mnohé odkazy, které jsou pro vyhledávač nezajímavé a tudiž je neextrahuji.

Při extrakci je rovněž nutné dodržovat *kanonické adresy*. V dokumentu se může vyskytovat kanonická adresa ve tvaru `http://test.com/item`. Tato kanonická adresa nám říká, že veškeré adresy ve tvaru `http://test.com/item?a=b` budou obsahovat stejný obsah. Jedná se například o adresy, které mění pouze pořadí produktů v eshopu. Tato informace je pro crawler nesmírně důležitá a zvyšuje jeho efektivitu, protože pouze na základě hash hodnoty textu stránky bychom nebyli schopni poznat, že se jedná o identický obsah.

Při filtraci redundantních adres v dokumentu nám rovněž pomůže HTML atribut `rel`. Crawler ignoruje veškeré odkazy obsahující atribut rel s některou z následujících hodnot: `nofollow, bookmark, alternate, license, search`. V sekci 2.1.4 bylo rovněž naznačeno respektování meta tagu `robots`. V případě, že dokument obsahuje tag `<meta name="robots"content="nofollow">` nejsou extrahovány žádné odkazy. Jestliže dokument obsahuje `<meta name="robots"content="noindex">` u dokumentu je zaznamenáno, že nemá být indexován a jsou extrahovány veškeré odkazy.

Pro korektní fungování crawleru je nutné respektovat `base` URL adresu. Tato adresa je použita jako základ pro všechny relativní adresy vyskytující se v daném HTML dokumentu.

Při extrakci odkazů se provádí kontrola vůči validitě domény (zda spadají do cílové domény) a seznamu zakázaných adres (blacklist). Dále je kontrolováno, do jaké hloubky od prvního vstupu na doménu se odkazy extrahují. Ve výchozím nastavení jsou odkazy extrahovány do hloubky 10.

3.4 Výpočet PageRanku

Pro výpočet PageRanku pomoci knihovny NetworkX je nejprve nutné postavit orientovaný graf sítě. Z MongoDB jsou postupně načítáni reprezentanti kanonických množin jako uzly grafu. Dále jsou načteny všechny hrany grafu jakožto odkazy mezi kanonickými množinami. Tento proces je nejdelší částí výpočtu PageRanku, samotný výpočet nad sestaveným grafem v paměti RAM je mnohonásobně rychlejší než jeho vytvoření. Po výpočtu PageRanku jsou hodnoty uloženy zpět do MongoDB databáze.

3.5 Indexer

V sekci 3.1.2 jsem se věnoval důvodům výběru PostgreSQL databáze jako hlavního indexačního nástroje. V aktuální sekci se budu věnovat nastavení fulltextového vyhledávání v databázi PostgreSQL a implementaci vlastních filtrů, které aplikuji na text dokumentů před jeho indexováním.

Indexace je v PostgreSQL nastavena následovně. Tabulka 2 obsahuje názvy a popis sloupců v databázi. SQL tabulka má nastavený takzvaný trigger, který spouští interní funkce na indexaci řádku po jeho vložení/změně. SQL funkce je popsána ve zdrojovém kódu 1. Jak můžeme vidět, jednotlivým sloupcům jsou zde přiřazeny priority A až C. Tyto priority jsou později zohledněny při řazení výsledků, funkce rovněž počítá se dvěma jazyky dokumentu.

3.5.1 Filtrace vstupu do indexeru

Před vložením dokumentu do PostgreSQL databáze prochází obsah dokumentu několika filtry. Veškeré dokumenty kratší než 500 znaků jsou ignorovány. Z HTML kódu dokumentu jsou extrahovány meta tagy `description` a `keywords`. Z URL adresy jsou extrahována relevantní slova. Slova jako „cz“, „www“, „upol“ a jiné jsou ignorována. Dále jsou v dokumentu nalezeny důležité nadpisy (HTML tagy `h1` a `h2`).

U PDF dokumentů je v případě potřeby text zkrácen z důvodu limitace PostgreSQL databáze. Dále probíhá detekce jazyka dokumentu. Detekce jazyka probíhá na vzorku textu z dokumentu o délce 10 000 znaků. U dlouhých dokumentu detekce jazyka trvá déle, proto je detekce prováděna na kratším vzorku. Jako název dokumentu je pro jednoduchost použit název souboru bez přípony. PDF parsery dokáží extrahovat i metadata z PDF souboru. PDF soubory však nejsou konzistentní a ne všechny soubory metadata obsahují. Na parsování PDF dokumentu je rovněž stanoven časový limit, aby nedocházelo k velkému zpomalení indexace.

3.5.2 Metadata

Pro UPOLSearch byl navržen vlastní formát metadat. V následující sekci se budu věnovat jejich popisu a způsobu zpracování. Metadata musí být na stránce

Tabulka 2: Struktura hlavní tabulky v PostgreSQL

Název sloupce	**Funkce**
`hash`	Primární klíč tabulky, vypočítan jako hash URL adresy
`url`	URL adresa na které se dokument nachází
`url_decoded`	URL adresa po dekodování, použito ve výdeji na zobrazení "hezné"URL adresy, z důvodu rychlosti uloženo v databázi
`url_words`	Specialně extrahovaná slova z URL adresy, použito pro indexaci
`url_length`	Délka URL adresy
`title`	Název dokumentu, použito pro indexaci
`language`	Jazyk dokumentu, použito při výdeji
`keywords`	Obsah HTML tagu keywords, použito při indexaci
`description`	Obsah HTML tagu description, použito při indexaci
`important_headlines`	Specialně extrahované důležité nadpisy v dokumentu, použito při indexaci
`content`	Samotný text dokumentu
`content_hash`	Hash textu dokumentu, využit pro detekci změny v dokumentu
`depth`	Hloubka ve které byl dokument nalezen, použito při řazení výsledků ve výdeji
`is_file`	Přiznak zda je dokument souborem (například PDF), použito při výdeji
`file_type`	Typ souboru, použito při výdeji
`pagerank`	Hodnota PageRank, použito při řazení výsledků ve výdeji
`search_index`	Specialní sloupec ve kterém je uložen zaindexovaný obsah dokumentu, nad tímto sloupcem je postaven index

```
1    begin
2     new.search_index :=
3       setweight(to_tsvector('czech',
4          coalesce(new.description, '')), 'B') ||
5       setweight(to_tsvector('czech',
6          coalesce(new.keywords, '')), 'A') ||
7       setweight(to_tsvector('czech',
8          coalesce(new.important_headlines, '')), 'B') ||
9       setweight(to_tsvector('czech',
10         coalesce(new.content, '')), 'C') ||
11      setweight(to_tsvector('czech',
12         coalesce(new.title, '')), 'A') ||
13      setweight(to_tsvector('czech',
14         coalesce(new.url_words, '')), 'A') ||
15      setweight(to_tsvector('english',
16         coalesce(new.description, '')), 'B') ||
17      setweight(to_tsvector('english',
18         coalesce(new.keywords, '')), 'A') ||
19      setweight(to_tsvector('english',
20         coalesce(new.important_headlines, '')), 'B') ||
21      setweight(to_tsvector('english',
22         coalesce(new.content, '')), 'C') ||
23      setweight(to_tsvector('english',
24         coalesce(new.title, '')), 'A') ||
25      setweight(to_tsvector('english',
26         coalesce(new.url_words, '')), 'A')
27      ;
28     return new;
29   end
```

Zdrojový kód 1: Funkce indexující řádky v tabulce.

umístěny v HTML tagu `<script type="application/ld+json"id="upsearch">JSON metadata</script>`. Samotná metadata jsou ve formátu JSON, jejich ukázka je připravena ve zdrojovém kódu 2.

Všechny podporované typy dat v metadatech jsou zobrazeny ve zdrojovém kódu 3 ve formátu JSONSchema (tato šablona je použita pro jejich validaci). Z důvodu malého nasazení v UPOL síti microfomát obsahuje pouze následující možnosti: Katedra, Zamestnanec, Předmět. Microformát lze kdykoli jednoduše rozšířit o nové pole či typy.

Metadata jsou při indexaci extrahována z dokumentů a vkládána do speciální tabulky v PostgreSQL databázi. Tato tabulka má vystavěný index nad JSON formátem podobně jako v případě hlavního indexu. Mimo samotná metadata jsou do databáze uloženy hashe metadat pro zamezení duplicit.

```
//Katedra
{
"@type": "Department",
"url": "http://inf.upol.cz",
"name": "Katedra informatiky",
}

//Zaměstnanec
{
"@type": "Employee",
"url": "http://trnecka.inf.upol.cz",
"name": "Martin Trnečka",
"email": "martin.trnecka@upol.cz"
"phone": "+420 585 634 731"
"office": "5.042"
}

//Předmět
{
"@type": "Class",
"url": "http://trnecka.inf.upol.cz/teaching/alm3/",
"name": "Algoritmická matematika 3",
"abbreviation": "ALS"
}
```

Zdrojový kód 2: Ukázka vlastního microformátu.

3.5.3 Optimalizace indexace

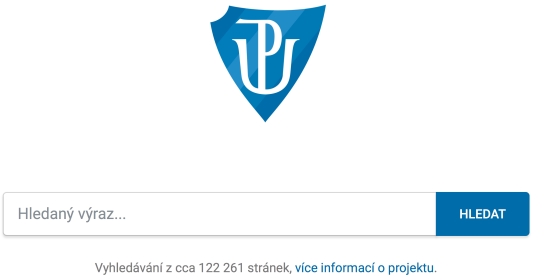
Při novém skenování sítě a následné indexaci je mnoho dokumentů indexováno znovu a bezezměny. Díky tomuto dochází ke zpomalení především u parsování PDF souborů. Proces parsování je pomalý a náročný.

Z tohoto důvodu je při indexaci porovnán hash obsahu dokumentu. Pokud je tento hash shodný s hashem dokumentu v existujícím indexu, je pouze zkopírován řádek z předchozí verze indexu. Tento proces výrazně optimalizuje celkový běh indexace.

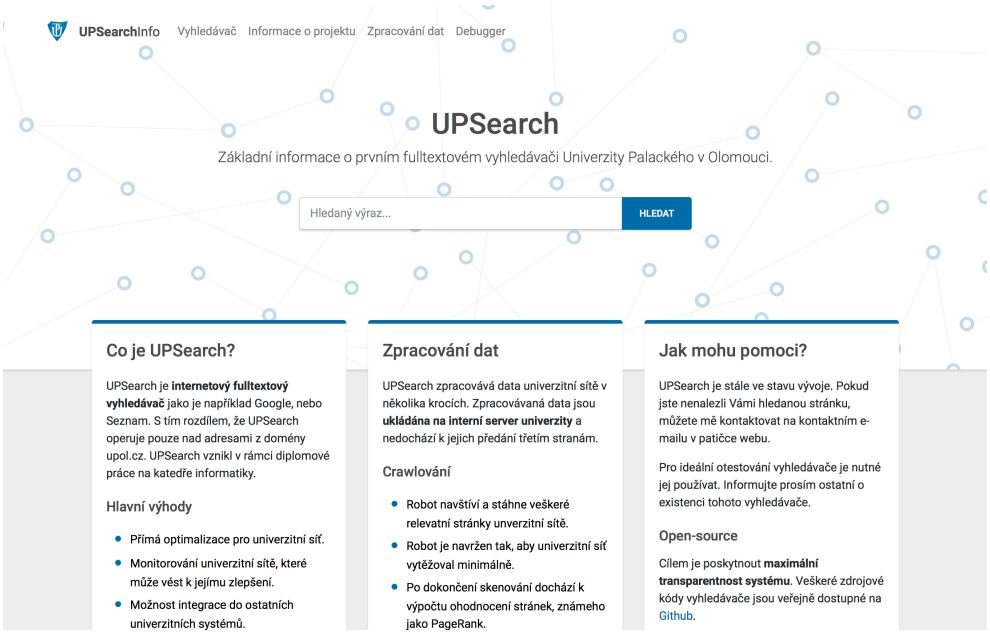
3.6 Výdej

Jak již bylo řečeno výdej slouží pro obsloužení dotazu uživatele. V případě UPOL-Search se jedná o jednoduchou webovou aplikaci implementovanou ve Flask frameworku. Veškerý front-end výdeje je navržen s ohledem na responzivitu, vyhledávač je tedy možné používat na všech zařízeních bez omezení.

Domovská stránka vyhledávače obsahuje jednoduchý formulář podobně jako ostatní webové vyhledávače (obrázek 1), rovněž obsahuje odkaz na informační stránku se základními informacemi o vyhledávači, jeho účelu na univerzitní síti a odkazem na zdrojový kód projektu (obrázek 2).



Obrázek 1: Hlavní formulář vyhledávače



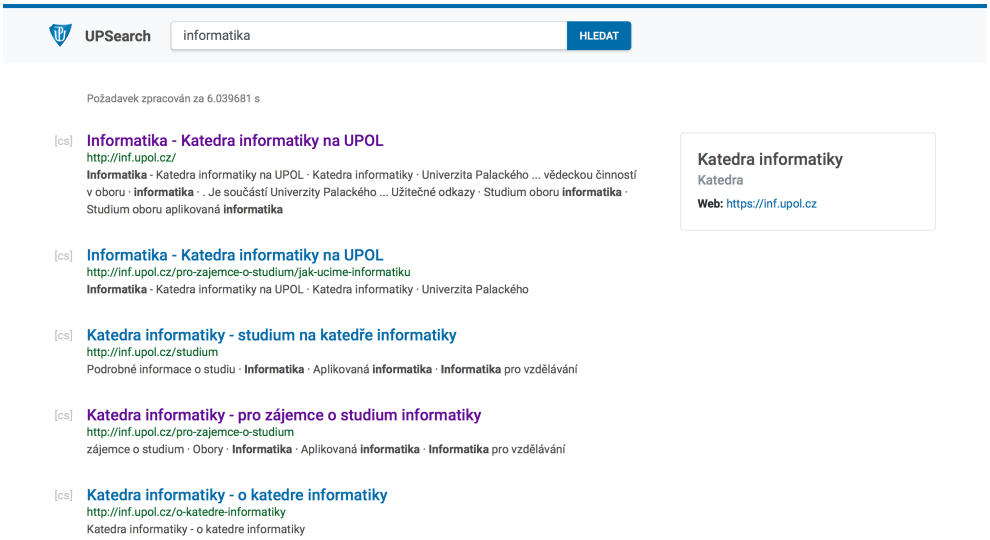
Obrázek 2: Stránka s informacemi o vyhledávači

Po zadání požadavku uživatelem nejprve dochází k detekci jazyka vyhledávané fráze. Tento krok je nutný pro správný výběr jazyka pro fulltextový vyhledávač v PostgreSQL databázi. Poté je sestaven SQL dotaz, který vyzvedne požadované dokumenty z databáze. Výsledky jsou řazený dle následujícího vzorce.

$$\log(ts\_rank\_cd * 0.6) + \log(pagerank) * 0.4$$

Kde ts_rank_cd je interní PostgreSQL funkce na řazení výsledku ve fulltextovém vyhledávači a $pagerank$ je PageRank dokumentu. Jak je ze vzorce patrné, větší ohled je brán na řazení dle fulltextového vyhledávání, než na PageRank dokumentu. Nastavení vah bylo zjištěno experimentálně. Funkce ts_rank_cd je použita v konfiguraci $1 \mid 4$ což znamená, že rank dokumentu je vydělen $1 + \log(délka\_dokumentu)$ v kombinaci s dělením ranku harmonickou vzdálenosti mezi extenty (pro více podrobností [15]). Tato konfigurace byla rovněž vybrána experimentálně.

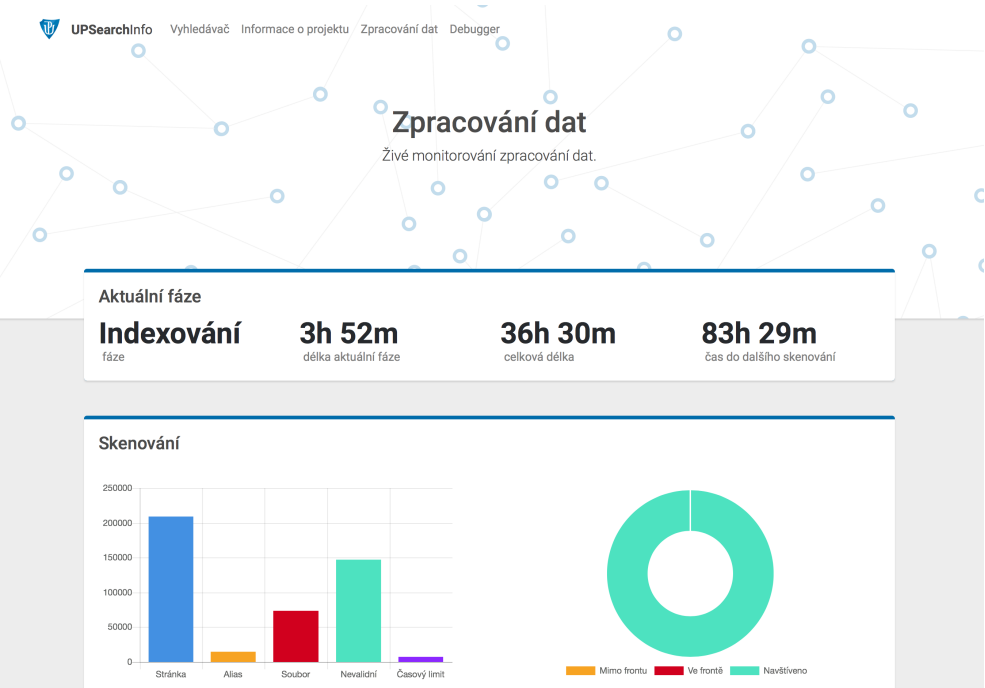
Následně je uživatel přesměrován na seznam nalezených výsledků (obrázek 3). Stránka s výsledky obsahuje vždy deset nalezených výsledků seřazených dle relevance. V dolní části stránky je dostupné stránkování, které uživateli poskytuje jednoduché listování výsledky. Každý relevantní výsledek obsahuje název, URL adresu dokumentu, jazyk dokumentu, náhled obsahu webové stránky se zvýrazněným nalezeným klíčovým slovem a případný formát souboru. Na pravé stráně od nalezených výsledků se zobrazují karty s rychlými informacemi založených na metadatech.



Obrázek 3: Stránka s výsledky pro hledaný termín informatika

Vyhledávač disponuje jednoduchým API na zjištění průběhu skenování (obrázek 4). S tímto API komunikuje stránka informující o průběhu skenování sítě. Dostupné informace jsou vypsány v tabulce 3.

Výdej rovněž obsahuje stránku na ladění (debugger, obrázek 5), na které uživatel může otestovat zda vyhledávač akceptuje danou URL adresu, zda se

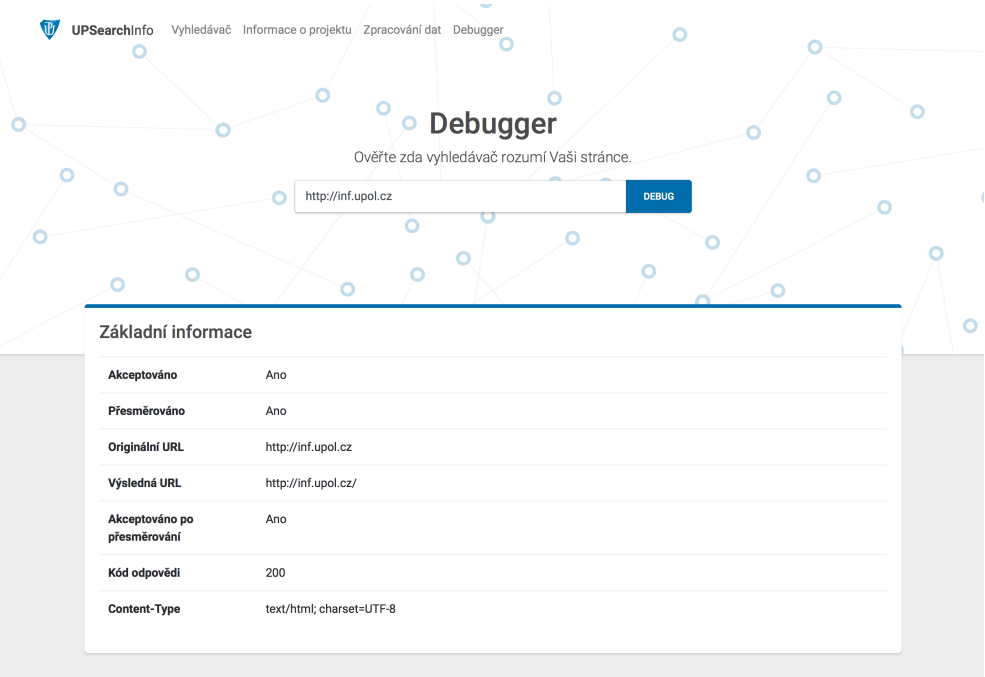


Obrázek 4: Stránka s monitorováním průběhu sběru dat

Tabulka 3: Dostupné informace o průběhu sběru dat.

Informace	**Popis**
Aktuální fáze	Aktuální fáze sběru dat, skenování, výpočet PageRanku, nebo indexace
Délka aktuální fáze	Čas, který uběhl od začátku aktuální fáze
Celková délka	Čas, který uběhl od začátku sběru dat
Čas do dalšího skenování	Čas, který zbívá do dalšího sběru dat
Graf informující o skenování	Graf informující o druhu obsahu na cílových URL adresách
Cílová doména	Doména na které probíhá sběr dokumentů
Počet domén	Počet nalezených domén v dané síti
Počet serverů	Počet nalezených serverů v dané síti
Počet URL adres	Počet nalezených URL adres v dané síti
PageRank	Informace o delce jednotlivých částí výpočtu PageRanku
Indexování	Počet zaindexovaných dokumentů

na stránce vyskytují validní metadata (obrázek 6), zda je soubor akceptován a indexován a mnoho dalšího. Na stránce je rovněž možné zjistit jaké odkazy jsou extrahovány a jaké zahazovány. Tyto údáje jsou především užitečné při lazení crawleru a realné produkci vyhledávače.

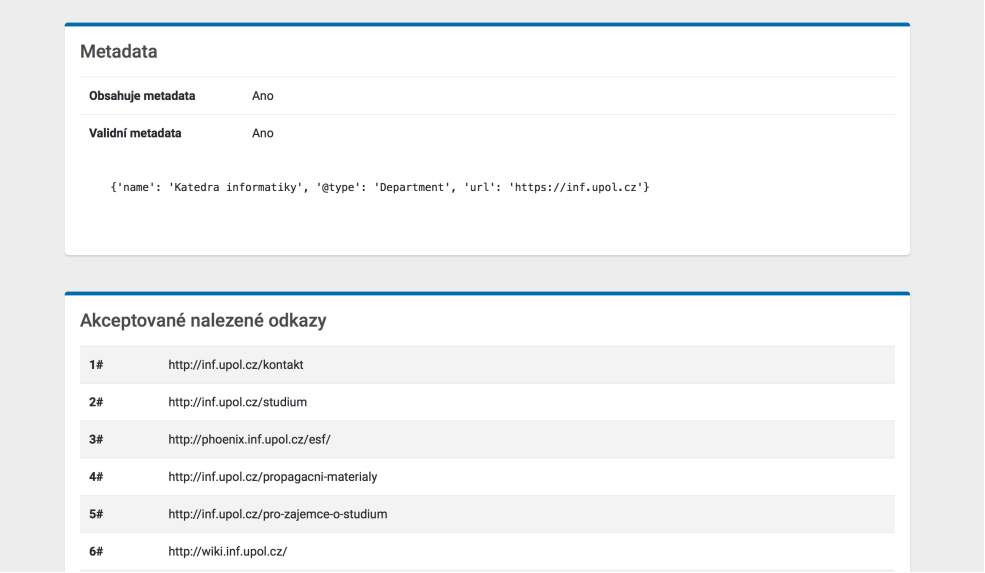


Obrázek 5: Stránka pro lazení URL adres v rámci vyledávače

3.7 Nasazení do produkce

Součástí práce bylo rovněž otestovat nasazení do produkce. Při těchto experimentech jsem vystřídal několik hardware konfigurací serverů. Aktuálně používaná konfigurace není ideální, ale umožňuje stabilní chod vyhledávače. Nejprve tedy představím aktuální konfiguraci a dále se budu věnovat ideální konfiguraci. Veškeré hardware specifikace jsou počítány s rezervou a neodpovídají čistým nárokům na výkon.

Aktuálně běží vyhledávač na dvou serverech. Tyto servery mohou být virtualizované. První server obsahuje obě databáze (PostgreSQL a MongoDB), druhý server obsahuje samotný crawler, indexer a výdej. Rozdělení serverů na dva je nutné z důvodu platformy databáze MongoDB. MongoDB databáze je navržena tak, že se snaží využít systémové prostředky na maximum. Další důvod rozdělení je ten, že v průběhu indexace jsou procesory vytíženy jak samotným procesem extrakce dat z dokumentů, tak databázemi. Vedle samotného crawleru a indexeru běží na druhém serveru rovněž HTTP server Apache2. Hardware specifikace databázového serveru je uvedena v tabulce 4 a hardware specifikace serveru vyhledávače je uvedena v tabulce 5. Nejzásadnějším hardware prvkem s ohledem na výkon je SSD disk. Vyhledávač nebyl ani testován na serveru s klasickým HDD



Obrázek 6: Stránka pro lazení URL adres v rámci vyledávače

Tabulka 4: Hardware specifikace DB serveru

Parametr	**Hodnota**
CPU	6x 3.07GHz
RAM	8GB
Disk	SSD 160GB

polem. SSD disky jsou potřebné pro zaručení rychlé odezvy na dotaz a na rychlé crawlování sítě. Druhým nejpodstatnějším prvkem je pamět RAM, celkový vyhledávač vykazuje velkou potřebu paměti RAM. CPU je posledním prvkem v ohledu na náročnost vyhledávače, s tím faktem, že díky paralelizaci je důležitější mít více jader, než menší počet výkonnějších jader.

Ideální konfigurace by obsahovala oddělený MongoDB/PostgreSQL virtuální server, dále virtuální server pro crawler a indexer (z důvodu CPU nároků) a poslední výdejový server pro Apache2 a Flask aplikaci výdeje. V případě této konfigurace by bylo nutné projekt rozdělit do dvou menších projektů a výdejovou

Tabulka 5: HW specifikace hlavního serveru

Parametr	**Hodnota**
CPU	8x 2.83GHz
RAM	28GB
Disk	Libovolné

část tímto způsobem separovat.

Aktuální produkce běží s validním certifikátem a povoleným HTTPS. Šifrování není v případě této aplikace důležité, protože mezi klientem a serverem nejsou předávány žádné výrazně citlivé informace, jedná se však o standart, který by měl být dle mého názoru podporován bez ohledu na potřeby projektu.

Frontend projektu rovněž obsahuje metadata ve formátu Open Graph [16] protokolu, která jsou čitelná sociální sítí Facebook a umožní ovlivnit vzhled a obsah sdíleného odkazu uživateli.

Produkce obsahuje plánovač v rámci knihovny Celery. Takzvaný Celery Beat umožňuje po vzoru CRON spouštět úlohy v určitém časovém intervalu případně ve specifickou denní dobu. Tento plánovač se ukázal být dostatečný a není potřeba větší režie. Tento fakt rovněž vyplývá z konstrukce vyhledávače. Crawlování je vždy spuštěno od začátku. Tím je eliminována jakákoli chyba, kterou by si crawler mohl nést postupem času. Již zmíněný monitoring umožňuje jednoduchý a pohodlný způsob dohledu nad funkčností vyhledávače.

Produkční nasazení obsahuje monitoring vyhledávaných termínu skrze Google Analytics. V prvotní verzi jsem se snažil implementovat vlastní systém monitoringu vyhledávaných frází s původní myšlenkou zveřejňovat nejpopulárnější fráze. Později se však ukázalo, že nasazení Google Analytics se projekt vyhnout nemůže, proto jsem využil tuto schopnost Google Analytics.

Instalace

Okrajově bych se rád věnoval instalaci samotného vyhledávače. Celá instalace je závislá na platformě, tudiž nebudu zacházet do detailů.

Prvním krokem je instalace programovacího jazyka Python 3.6, databáze MongoDB a databáze PostgreSQL 10 (zde je verze podstatná). Dále následuje instalace samotného vyhledávače, do virtuálního prostředí Pythonu, tento krok lze provést přímo z Github repozitáře (aktuálnější), případně s přiloženého CD. Následně je nutné nainstalovat potřebné slovníky do PostgreSQL databáze (z důvodu jejich licence, je nemohu přiložit na CD), vytvořit databázi a odpovídajícího uživatele. Následovně je možné spustit konfiguraci SQL tabulky a jazyků, pro tento úkon slouží příkazy `upol_search_engine setup languages` a `upol_search_engine setup functions`. V rámci konfigurace databází je rovněž potřeba vytvořit uživatele v MongoDB. Následně je nutné nainstalovat `rabbitmq-server`, který předává zprávy mezi Celery a vyhledávačem. V rámci konfigurace Celery je nutné vytvořit daemon pro workery a plánovač beat (odpovídající soubory, které musí být umístěny do adresáře `/etc/init.d/` jsou přiložené na CD, případně dostupné na Github repozitáři). Posledním krokem je nastavení `mod_wsgi` v rámci webového serveru Apache2. Následuje dobrovolná konfigurace https šifrování a samotné spuštění vyhledávače. Scanování sítě lze spustit manualně příkazem `upol_search_engine`, případně se scanování pustí automaticky v nastavený čas po spuštění daemonu celery beat.

Tabulka 6: Orientační časové rozložení běhu sběru dat

Fáze	Délka
Crawler	cca 28 hodin
Výpočet PageRank	cca 7 hodin
Indexace	cca 7 hodin

4 Zhodnocení

V poslední části se budu věnovat zhodnocení celé práce. Rád bych diskutoval celkový výkon vyhledávače, zjištěné poznatky o univerzitní síti a možné vylepšení vyhledávače, která z časového důvodu nemohla být implementována.

4.1 Výkon

Jak už bylo v sekci 3.1.1 naznačeno, v počátcích vývoje prototypu byly pochybnosti o výkonu programovacího jazyka Python 3.6. Během vývoje nastal problém, kdy crawler přetížil část univerzitní sítě a musel být zablokován. Tato událost vedla k implementaci omezovače a jednoznačně ukázala, že je výkon pro potřeby vyhledávače dostatečný.

Tabulka 6 obsahuje časové rozložení běhu jednotlivých částí vyhledávače. Celková délka běhu sběru dat pro univerzitní síť UPOL se pohybuje okolo 40 hodin. Během sběru dat je uloženo orientačně 470 000 URL adres. Tabulka 7 ukazuje rozložení typu nalezených URL adres.

Nejkritičtější částí pro sběr dat je indexace. Během indexace dochází k úplnému vytížení obou serverů. Délka odezvy na požadavek se z orientační délky jedna sekunda změní v nejhorším případě na několik sekund. Tento problém by byl řešitelný lepší distribucí mezi servery a omezení přiřazených prostředků.

4.2 Poznatky o univerzitní síťi

Během skenování univerzitní sítě jsem došel k několika zajímavým poznatkům, které bych zde rád prezentoval. Univerzitní síť obsahuje takzvané „ostrovy“ na které nevedou žádné odkazy. Tyto ostrovy crawler nedokáže nalézt. Jediným způsobem je přidat ostrovy do počátečního seedu. V tom případě se ale budeme potýkat s problémy při výpočtu PageRanku, který vyžaduje aby byly vrcholy grafu propojeny. Jediným spolehlivých řešením je začít na tyto ostrovy odkazovat z ostatních univerzitních webů. Tabulka 7 obsahuje orientační obsah univerzitní sítě. Tabulka 8 obsahuje orientační poměr zastoupených jazyků (měřeno pouze u webových stránek).

Dalším problémem, který se projevil při testování crawleru byl poměr domén vůči serverům. Z tohoto důvodu crawler vytvářel nečekanou zátěž na univerzitní

Tabulka 7: Rozložení typu obsahu URL adres

Typ	Počet
Celkem	469 029
Soubory	223 488
Nepodporované soubory	150 956
PDF	64 279
Alias	14 909
Timeout	6 100

Tabulka 8: Zastoupení jazyků na webových stránkých univerzity

Jazyk	Počet
Čeština	154 028
Angličtina	54 623
Slovenština	2046

síť. Podsítě obsahující dostatečný počet serverů na daný počet domén nevykazovaly žádný problém i při neomezené rychlosti crawleru. Orientační poměr domén ku IP adresám na celé univerzitní síti je cca 7:1. V tomto ohledu je síť katedry informatiky na které byl vyhledávač vyvíjen mnohem robustnější.

4.3 Možnosti vylepšení

V následující sekci se budu věnovat vylepšením, které jsem nestihl realizovat, nebo jsem realizovat nemohl vůbec. Sekci jsem rozdělil do částí odpovídajícím struktuře vyhledávače.

Crawler

Při konzultaci ve firmě Seznam.cz jsem zjistil, že návrh crawleru jako algoritmu s fixním začátkem a koncem (vyprázdnění fronty) je v praxi dosti neohrabaný. U takového crawleru pro správnou detekci konce algoritmu nesmí dojít k sebemenší chybě během jeho běhu. V praxi je navíc potřeba některé weby stahovat častěji než jiné. Tyto poznatky vedou k návrhu crawleru jako algoritmu, který běží kontinuálně. Takový algoritmus operuje rovněž s frontou a dalšími částmi, jeho hlavní rozdíl je však již zmíněná kontinualita. Proces crawlování nikdy nekončí a díky tomu, je možné určité stránky stahovat častěji než jiné. V mém návrhu algoritmu vždy dojde k aktualizaci celkové univerzitní sítě. S kontinuálním algoritmem ale přichází poměrně větší režie indexace a výpočtu PageRanku.

Stávající řešení není špatné vůči velikosti cílové domény. V měřítku se kterým se potýká Seznam.cz by však toto řešení bylo nepoužitelné.

Indexer

První očividnou úsporou času by mohlo být indexování dokumentů již během samotného crawlování. Toto vylepšení by bylo realizovatelné bez změny celkové struktury projektu. Ke konci crawlování je server nevyužit a jeho výkon by šel použit na indexování dokumentů.

Dalším možným rozšířením, které by nevyžadovalo změnu struktury je indexace více druhů souborů. Aktuálně indexer indexuje pouze PDF soubory. Otvírá se možnost indexovat Microsoft Word dokumenty (`.doc, .docx`), nebo prosté textové soubory (`.txt`). V pokročilé fázi by indexer mohl provádět detekci obrázků a vyhledávat rovněž nad obrázky z univerzitní sítě.

PageRank

Aktuální výpočet PageRanku splňuje svůj účel, je ale poněkud kostrbatý a zdlouhavý. První možnou optimalizací by mohl být výpočet v rámci samotné MongoDB databáze. Tím bychom mohli vynechat krok, ve kterém se z databáze staví graf v paměti a celý proces by se zrychlil. Vedlejším efektem by byla možnost výpočtu PageRank již během crawlování.

Další zvažovanou možností bylo použití grafové databáze. Taková databáze by umožňovala uchovávat navštívené adresy v takové struktuře v jaké doopravdy na síti existují. V tomto případě by byl výpočet PageRanku ještě rychlejší, na diskuzi však zůstává jakou databázi zvolit a zda by výkonnostně byla srovnatelná s MongoDB databází.

Výdej

Prvním potenciálním vylepšením je našeptávání často vyhledávaných frázi. Pro tuto funkci by bylo nutné ukládat veškeré požadavky od uživatelů a ty následně zpracovat. Důležitou částí by byla filtrace nevhodných slov, která se ukázala býti velice reálným problémem.

Dalším a poměrně složitým krokem by bylo již zmiňované porozumění dotazu od uživatele. Tato technika by umožnila dotaz lépe analyzovat a volit vhodnější požadavek v PostgreSQL databázi. Ne v každé situaci je vhodné mezi jednotlivé slova požadavku vkládat logický AND, někdy by bylo vhodné použít logický OR.

Implementace filtrování výsledků by mohlo být poměrně jednoduché rozšíření. Aktuální výdej nepodporuje vyhledávání pouze mezi soubory, nebo mezi vybranou jazykovou mutací dokumentů.

V neposlední řadě se nabízí vytvoření vlastního API pro poskytování výsledků pro ostatní produkty univerzity. Mezi reálně nasaditelné místa patří například mobilní aplikace UPlikace [17].

Formální konceptuální analýza

Vyhledávač UPOLSearch spadá do kategorie klasických fulltextových internetových vyhledávačů. Tyto vyhledávače vrací na základě vyhledáváne fráze lineární seznam navrhovaných výsledků seřazených dle určitých kriterií. Zajímavým rozšířením (nádstavbou) nad vyhledávačem by bylo použití formální konceptuální analýzy (FCA). Nechci zde zacházet do přílišných detailů, ale vyhledávač s nádstavbou FCA poskytuje jako výsledek určitou strukturu (část konceptuálního svazu). Tento přístup umožňuje návštěvníkovi vidět, že se pod hledaným pojmem vyskytuje více skupin dle významů. Příkladem může být slovo „apple“, které může reprezentovat jablko, nebo firmu Apple. Uživatel si následně vybere, jaký pojem měl na mysli a prohlédne si nabízené výsledky. Rovněž bych se rád experimentálně pokusil o implementaci poznatků z matematické psychologie, konkrétně pak problematiky basic-level konceptu popsaném v článku [18].

Univerzitní vyhledávač

Během testování vyhledávače nad univerzitní sítí Univerzity Palackého jsem došel k myšlence univerzitního vyhledávače napříč českými univerzitami. V aktuální fázi vývoje si netroufnu odhadovat, zda by byl vyhledávač dostatečně robustní pro zpracování takového množství dat. Výsledný vyhledávač by však mohl být velice dobře využitelný. Rovněž by mohl obsahovat funkce typu „vyhledávání pouze na určitých univerzitách“ a jiné.

```
//Společné pro všechny typy
{
   "type": "object",
   "required": ["@type"],
   "properties": {
      "@type": {"type": "string"},
   }
}

//Pro konkrétní typy metadat
{
   'employee': {
      "type": "object",
      "required": ["url", "name"],
      "properties": {
         "name": {"type": "string"},
         "url": {"type": "string", "format": "uri"},
         "phone": {"type": "string", "format": "phone"},
         "email": {"type": "string", "format": "email"},
         "office": {"type": "string"},
      },
   },
   'department': {
      "type": "object",
      "required": ["url", "name"],
      "properties": {
         "name": {"type": "string"},
         "url": {"type": "string", "format": "uri"},
      },
   },
   'class': {
      "type": "object",
      "required": ["url", "name", "abbreviation"],
      "properties": {
         "name": {"type": "string"},
         "url": {"type": "string", "format": "uri"},
         "abbreviation": {"type": "string"},
      },
   }
}
```

Zdrojový kód 3: JSONSchema validační šablona metadat.

Závěr

Výsledný fulltextový webový vyhledávač obsahuje veškeré klíčové funkce webových vyhledávačů. Celkový vývoj byl směřován k prakticky nasaditelnému řešení, které by bylo dále otevřeno budoucímu vývoji. Uživatelské rozhraní je navrženo s ohledem na jednoduché a responzivní používání napříč různými zařízeními. Vyhledávač má prostor pro výkonostní optimalizaci, avšak dosahuje výkonu, který nijak zásadně nelimituje ostré nasazení v praxi.

Conclusions

Final full-text web search engine contains all key features of web search engines. Overall development was directed towards practically deployable solution which could be furthermore open to future development. The user interface is designed with respect to easy and responsive use across numerous devices. The search engine has room for performance optimization although it reaches a performance level that doesn't limit practical use.

A Obsah přiloženého CD/DVD

src/
: Kompletní zdrojový kód vyhledávače UPOLSearch, rovněž dostupný na https://github.com/UPOLSearch/UPOL-Search-Engine

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii.

Literatura

[1] DUCKDUCKGO. *DuckDuckGo*. Dostupný z: ⟨https://duckduckgo.com⟩.

[2] RICHTER, Stephan. *XML and HTML with Python*. Dostupný z: ⟨http://lxml.de⟩.

[3] PAGE, Lawrence; BRIN, Sergey; MOTWANI, Rajeev; WINOGRAD, Terry. *The PageRank citation ranking: Bringing order to the web*. 1999.

[4] W3C. *W3C - Markup Validation Service*. Dostupný z: ⟨https://validator.w3.org⟩.

[5] GROUP, The PostgreSQL Global Development. *Chapter 12. Full Text Search*. Dostupný z: ⟨https://www.postgresql.org/docs/9.5/static/textsearch-intro.html⟩.

[6] GROUP, The PostgreSQL Global Development. *Chapter 11. Indexes*. Dostupný z: ⟨https://www.postgresql.org/docs/9.1/static/indexes-intro.html⟩.

[7] BV, Elasticsearch. *Elasticsearch*. Dostupný z: ⟨https://www.elastic.com⟩.

[8] MONGODB, Inc. *What is MongoDB?* Dostupný z: ⟨https://www.mongodb.com/what-is-mongodb⟩.

[9] PIVOTAL SOFTWARE, Inc. *RabbitMQ*. Dostupný z: ⟨https://www.rabbitmq.com⟩.

[10] SOLEM, Ask. *Celery: Distributed Task Queue*. Dostupný z: ⟨http://www.celeryproject.org⟩.

[11] RONACHER, Armin. *Jinja2: Template engine*. Dostupný z: ⟨http://jinja.pocoo.org⟩.

[12] DUMPLETON, Graham. *mod_wsgi*. Dostupný z: ⟨https://modwsgi.readthedocs.io/en/develop/⟩.

[13] SCRAPINGHUB. *Scrapy Frawework*. Dostupný z: ⟨https://scrapy.org⟩.

[14] LIMITED, phpBB. *phpBB*. Dostupný z: ⟨https://www.phpbb.com⟩.

[15] GROUP, The PostgreSQL Global Development. *12.3. Controlling Text Search*. Dostupný z: ⟨https://www.postgresql.org/docs/9.0/static/textsearch-controls.html⟩.

[16] FACEBOOK. *Open Graph protocol*. Dostupný z: ⟨http://ogp.me⟩.

[17] OLOMOUCI, Univerzita Palackého v. *Uplikace*. Dostupný z: ⟨http://uplikace.upol.cz⟩.

[18] BELOHLAVEK, Radim; TRNECKA, Martin. Basic Level in Formal Concept Analysis: Interesting Concepts and Psychological Ramifications. In. *IJCAI*. 2013, s. 1233–1239.