VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZHRANÍ PRO ASPEKTOVÉ VYHLEDÁVÁNÍ V INDEXU WIKIPEDIE

INTERFACES FOR FACETED SEARCH IN INDEXED WIKIPEDIA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PETER CILIP**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20838/2017/xcilip00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Cilip Peter, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Rozhraní pro aspektové vyhledávání v indexu Wikipedie**
Interfaces for Faceted Search in Indexed Wikipedia
Kategorie: Algoritmy a datové struktury

Pokyny:

1. Seznamte se s metodami vývoje facetových grafických uživatelských rozhraní a technikami efektivního počítání četnosti aspektů v semi-strukturovaných datech.
2. Navrhněte a implementujte systém pro vyhledávání v indexu Wikipedie a extrahovaných informací na základě aspektů (facetů) odvozených z počátečního příkladu.
3. Srovnejte snadnost a efektivitu dotazování s tradičními způsoby, dostupnými v konkurenčních nástrojích.
4. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Funkční prototyp řešení

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cieľom tejto práce je študovať vyhľadávacie systémy s využitím aspektového filtra, následne implementovať vlastný systém, ktorý bude predstavovať aspektové vyhľadávanie nad indexom Wikipédie. Práca sa teda venuje existujúcim riešeniam aspektového vyhľadávania. Z existujúcich systémov a ich chýb bol navrhnutý systém, ktorý je výstupom tejto práce. Navrhnutý systém je popísaný z pohľadu návrhu aj samotnej implementácie. Výstupom je teda aplikačné a grafické rozhranie. Aplikačné rozhranie sa dá integrovať do ľubovoľného informačného systému, kde môže slúžiť aj ako pomocný viacdimenzionálny filter. Grafické rozhranie poskytuje možnosť ako sa dá aplikačné rozhranie využiť v reálnom systéme. Dôraz bol kladený hlavne na využitelnosť a jednoduchosť takéhoto systému tak, aby sa dal využiť v existujúcich informačných systémoch.

Abstract

Main aim of this thesis is to study existing systems of faceted search and to design own system based on faceted search in the index of Wikipedia. In this thesis we can meet with existing solutions of faceted search. From mistakes and failures of existing solutions was designed our own system, that is output of this thesis. Designed system is described in way of design and implementation. Product of thesis is application and graphical interface. Application interface can be integrated into existing informational system, where it can be used as multidimensional filter. Graphical interface provides option how can application interface be used in real system. System was created focusing on usefullness and simplicity, for using in existing information systems.

Kľúčové slová

vyhľadávanie, aspekty, aspektové vyhľadávanie, Wikipédia, Wiki, vedomosť, systémy založené na vedomostiach, RDF, turtle, databázy, relačné databázy, grafové databázy

Keywords

searching, facets, faceted searching, Wikipedia, Wiki, knowledge, knowledge based systems, RDF, turtle, database, relational database, graph database

Citácia

CILIP, Peter. *Rozhraní pro aspektové vyhledávání v indexu Wikipedie*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Rozhraní pro aspektové vyhledávání v indexu Wikipedie

Prehlásenie

Prehlasujem, že som diplomovú prácu vypracoval samostatne pod vedením pána Doc. RNDr. Pavla Smrža, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Peter Cilip
22. mája 2018

Poďakovanie

Ďakujem vedúcemu práce Doc. RNDr. Pavlovi Smržovi, Ph.D. za jeho cenné poznatky, rady a vecné pripomienky, ktoré mi boli nápomocné pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Táto práca sa bude zaoberať vyhľadávaním. Pretože vyhľadávanie inforomácií je úzko späté s ľudskou zvedavosťou. Práve ľudská zvedavosť stojí za všetkými vedomosťami, ktorými ľudia disponujú. A to je podstatou tejto práce, umožniť vyhľadávanie informácií v systémoch, ktoré boli založené na informáciach a vedomostiach o vzťahoch medzi informáciami.

Za svoje ciele si táto práca kladie štúdium vyhľadávania ako takého. Techník, ktoré nám vyhľadávanie dokážu zefektívniť, uľahčiť a nakoniec aj verifikovať správnosť vyhľadaných informácií.

S vyhľadávaním nám veľmi často pomáhajú informačné systémy, ktoré by nám to hľadanie mali uľahčiť, avšak častejšie nám ho pomôžu skôr spomaliť, pretože tieto systémy sú prehltené informáciami a to podstatné ostáva skryté v neprehľadnom systéme. Tým pádom sa pozrieme na najčastejšie chyby týchto systémov a ako sa ich programátori snažili vyriešiť, či schovať. Okrem jednoduchých systémov si priblížime aspektové vyhľadávanie a metódy vývoja aspektových grafických užívateľských rozhraní.

V práci sa taktiež dozvieme o verifikácií vyhľadaných dát. Či systémy vracajú dáta, ktoré majú byť zobrazované a taktiež prečo sa niektoré dáta nezobrazujú, aj keď by mohli.

Nakoniec v práci navrhneme a implementujeme celý systém, ktorý bude schopný vyhľadávať a filtrovať v dátach Wikipédie. Systém, ktorý bude implementovaný, by sa mal poučiť z existujúcich riešení a mal by byť jednoduchý na použitie. Výkonnosť systému bude zaznamenaná a porovnaná so systémami, ktoré implementujú podobný typ vyhľadávania. Samotné rozhranie bude implementované s ohľadom na najnovšie trendy v užívateľských rozhraniach, aby sa užívateľ nemusel učiť nové veci alebo niečo zložito hľadať v užívateľskom rozhraní.

V kapitole 2 sa pozrieme čo vyhľadávanie znamená a priblížíme si ako sa dá vyhľadávanie klasifikovať. Ako názov práce napovedá budeme sa zaoberať vyhľadávaním podľa aspektov, čo znamená že vyhľadávanie obohatíme o ďalšie dodatočné informácie.

Ďalej sa pozrieme ako je možné informácie a vedomosti uchovávať v počítačových systémoch. Ako a prečo sa takto informácie ukladajú sa dozvieme v kapitole 3. Táto kapitola je celkom rozsiahla a vysvetľuje a ukazuje, ako také dáta môžu vyzerať.

Ak už budeme vedieť, ako dáta budú vyzerať, kapitola 4 nám ukáže, ktoré systémy sú založené na spomínaných informáciach a ako z nich odvodzuje vzťahy medzi jednotlivými informáciami. Hlavným cieľom práce je študovať ako vyzerajú vzťahy medzi informáciami a ako sú popísané v počítačových systémoch a taktiež ako v týchto informáciach efektívne vyhľadávať pomocou aspektov (2). Aby sa dalo jednoducho odpovedať na komplexnejšie otázky ako napríklad „Ktoré osoby, mali povolanie umelec, a narodili sa pred rokom 1500 n.l. v meste Rím?“ a rôzne ďalšie...

Okrem samotných dát práca študuje existujúce riešenia, ktoré využívajú aspektové vyhľadávanie. Tieto riešenia budú popísané v kapitole 6, kde budú jednotlivé riešenia detailne znanalyzované. V kapitole návrhu sa z problémov a chýb existujúcich riešení inšpirujeme, aby nenastali rovnaké chyby pri navrhovaní vlastného systému.

Pri návrhu vlastného riešenia budeme klásť dôraz na rozšíriteľnosť a nezávislosť jednotlivých celkov systému, aby sa systém mohol použiť v rozličných druhoch vyhľadávacích aplikácií. Kapitoly v ktorých sa budeme venovať vlastému riešeniu, jeho návrhu (7) a jeho samotnej implementácií (8) budú najobšírnejšie a budú popisovať ako systém bude fungovať ako celok, ale aj ako budú fungovať samostatné časti systému.

Kapitola 2

Aspektové vyhľadávanie

Je doba, v ktorej sa lpie na nutnosti vyhľadať si správne informácie v krátkom čase. Lenže občas nestačí vyhľadávať len pomocou kľúčových slov. Vyhľadávanie pomocou kľúčových slov hľadá poskytnutý zoznam slov v danej dátovej sade.

Výstupom takéhoto hľadania môže byť rovnaká entita, ktorá sa vo výstupe môže vyskytovať viackrát, pretože toto vyhľadávanie je jednoduché a neberie do úvahy vzťahy medzi vyhľadanými entitami [24]. Takéto vyhľadávanie je neefektívne, ak by sme chceli vyhľadávať entitu, ktorá je definovaná vzťahmi. Teda z tohto hľadiska môžeme vyhľadávanie rozdeliť na:

- Vyhľadávanie pomocou kľúčových slov.
- Vyhľadávanie pomocou vzťahov medzi entitami.

Poďme si však povedať, čo to vlastne vyhľadávanie je. Vyhľadávanie v počítačovej vede je algoritmus, ktorý rieši vyhľadávací problém. Ide o extrahovanie určitého okruhu dát zo všetkých poskytnutých informácií, či už môže ísť o rozsiahle dátove štruktúry a kolekcie, alebo jednoduchý vyhľadávací priestor.

Samotné vyhľadávanie je možné rozdeliť do niekoľkých skupín. Vzhľadom na použitý algoritmus je možné uvažovať nad lineárnym a binárnym vyhľadávaním, vyhľadávaním s delením intervalov a v neposlednom rade aj s vyhľadávaním s využitím hašovacej techniky. Niektoré vyhľadávacie algoritmy využívajú zložitejšie štruktúry, aby sa zlepšila časová a výpočtová náročnosť. Medzi najpopulárnejšie algoritmy vyhľadávania patrí vyhľadávanie v stromoch. Najpoužívanejší z nich je binárny vyhľadávací strom, b-strom a ternárny vyhľadávací strom. Všetky spomenuté algoritmy sa radia medzi vyhľadávanie pomocou kľúčových slov. Avšak existujú aj systémy či vyhľadávacie algoritmy, ktoré študujú vzťahy medzi entitami vo vyhľadávacom priestore [12]. Samozrejme každý prístup ma svoje pre a proti. Zásadnú rolu pri vyhľadávaní hrajú samotné dáta, v ktorých sa bude vyhľadávať.

Aspektové vyhľadávanie, známe pod pojmami ako aspektová navigácia, či aspektové prehliadanie, je technika prístupu k informáciam, ktoré sú organizované do systému. Systém dovoľuje užívateľom prehliadať kolekcie informácií prostredníctvom filtra, ktorý si užívateľ nadefinuje. Tento systém informácie klasifikuje na základe aspektov, teda vzťahov, či popisom danej informácie.

Aspekty sú väčšinou odvodené analýzou textu, pomocou techniky extrakcie entity[1]. V akademickej komunite sa aspektové vyhľadávanie stalo populárnym, kvôli knižniciam

[1] https://en.wikipedia.org/wiki/Named-entity_recognition

a výskumníkom zaoberajucími sa informačnými technológiami, ktorý sa špecializujú na extrahovanie informacií [26].

Hlavný rozdiel medzi aspektovým a klasickým vyhľadávaním je, že klasické vyhľadávanie má obmedzenú sadu hlavných atribútov. Atribúty aspektového vyhľadávania sú odvodzované zo semištrukturovaných dát, ktoré nie sú pevne dané. Dátová sada sa však môže neustále meniť. Preto algoritmus, ktorý odvodzuje hlavné atribúty musí byť inteligentný, aby vyberal správne. Rýchlosť algoritmu záleží od uchovania dát. Štruktúra uchovaných dát musí byť nadefinovaná tak, aby mohli byť nad týmito dátami vykonávané základné operácie [1, 27]. Ako sa dáta pre vyhľadávanie ukladajú v informačných systémoch si povieme v kapitole 3.

Kapitola 3

Spôsob uchovávania štruktúrovaných informácií

Táto práca sa taktiež zaoberá podobou štruktúrovaných dát, ale aj ako tieto dáta spracovať do využiteľnej podoby pre vyhľadávanie. Pre veľký objem dát sa štruktúrované dáta ukladajú a uchovávajú pomocou serializačných formátov, ktoré z väčšej časti zdieľajú rovnaké princípy. V následujúcej časti si priblížime, ako môžu štruktúrované dáta vyzerať. Je nutné si však povedať, ako jediný kúsok informácie vyzerá, či ako je definovaný. Nato nám slúži jazyk *Resource Description Framework*, skrátene *RDF*, o ktorom si napíšeme v následujúcej časti. Ide o jazyk, ktorý reprezentuje informácie o zdrojoch vo svete internetu. Tento jazyk predstavuje základné koncepty a popisuje ich XML syntax [28]. Taktiež popisuje ako si nadefinovať RDF slovníky, použitím RDF slovníkového popisného jazyka (z angl. *RDF Vocabulary Description Language*) [29, 7].

3.1 Základný koncept

Jazyk RDF sa používa na vytvorenie tvrdenia (z angl. *statement*) o nejakom zdroji, ktorý sa na internete vyskytuje. Napríklad, ak by sme chceli vytvoriť tvrdenie o niekom, kto sa volá *John Smith*, že vytvoril webovú stránku. V prirodzenom jazyku by sa to priamočiaro dalo napísať asi takto:

http://www.example.org/index.html has a *creator* whose value is *John Smith*.

Podstatné časti tohto tvrdenia sú:

- Vec, ktorú tvrdenie popisuje (webová stránka).
- Špecifickú vlastnosť (*creator*, tvorca) veci, ktorú tvrdenie popisuje.
- Hodnota vlastnosti (*John Smith*), ktorú tvrdenie popisuje.

V tomto tvrdení je použité *Uniform Resource Locator*, skrátene *URL*, na identifikáciu webovej stránky. Na poskytnutie identifikácie vzťahu je použité slovo *creator* a na identifikáciu hodnoty je použité slovné spojenie *John Smith*. Rovnako ako spomenutý príklad je možné popísať aj rôzne ďalšie vlastnosti zdrojov. Napríklad si môžeme uviesť ďalšie:

- *http://www.example.org/index.html* has a *creation-date* whose value is *August 16, 1999*.
- *http://www.example.org/index.html* has a *language* whose value is *English*.

RDF jazyk je založený na myšlienke, že ľubovoľná vec má vlastnosti, ktoré nadobúdajú určité hodnoty. Jednotlivé zdroje môžu byť popísané vytvorením tvrdení, ktoré sú podobné tvrdeniam spomenutým vyššie a špecifikujú vlastnosti a hodnoty [32]. Tento jazyk používa konkrétnu terminológiu, pre popis konkrétnych častí tvrdenia. Teda tvrdenie má presne 3 časti a sú nimi:

- *subject* – časť, ktorá identifikuje o čom dané tvrdenie je.
- *predicate* – časť charakteristická pre *subject*, ktorú tvrdenie špecifikuje napr. tvorca, dátum vytvorenia, apod.
- *object* – časť, ktorá definuje hodnotu vlastnosti.

Pre posledné spomenuté tvrdenie by to vyzeralo následovne:

- *subject* je URL http://www.example.org/index.html.
- *predicate* je slovo „language“.
- *object* je slovo „English“.

3.2 RDF jazyk

Avšak, angličtina je dobrá pri komunikácií medzi anglicky hovoriacimi ľudmi, RDF je o vytváraní strojovo spracovaných tvrdeniach. Aby tvrdenia bolo možné spracovať strojovo je nutné splniť dve podmienky:

- Stroj musí vedieť rozoznať *subject*, *predicate*, *object* bez možnosti zmýliť sa.
- Musí existovať jazyk, ktorý reprezentuje tvrdenia a je možné ho využiť na výmenu tvrdení medzi strojmi.

Existuje však webová architektúra, ktorá poskytuje obidve podmienky. Ako bolo ilustrované v podkapitole 3.1, internet ako taký poskytuje jednu z foriem identifikátora – URL. Je to vlastne reťazec identifikujúci internetový zdrojový dokument, tým že poskytuje jeho prístupový mechanizmus, teda jeho sieťovú adresu. Internet poskytuje viac všeobecnú formu identifikátora nazvaného *Uniform Resource Identifier*, skrátene *URI* [32].

3.3 Uniform Resource Identifier – URI

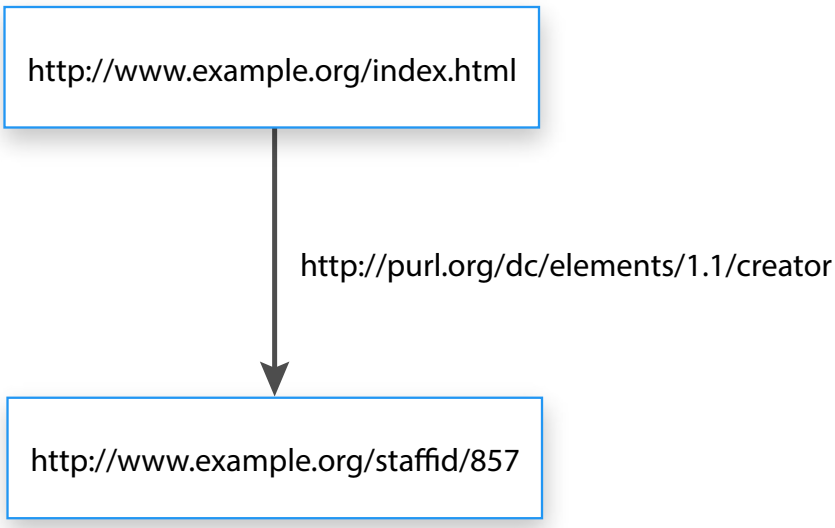
URL sú iba určitým typom URI. Všetky URI zdieľajú vlastnosti, ktoré môžu byť nezávislé, vytvorené rôznymi osobami alebo organizáciami, ktoré používame na identifikáciu vecí. Avšak URI nie sú limitované identifikáciou vecí, ktoré majú sieťovú adresu, alebo používajú nejaký druh prístupového mechanizmu počítačov. V skutočnosti URI môže byť vytvorené na identifikáciu ľubovoľnej veci, ktorá potrebuje byť odkazovaná v RDF tvrdeniach. Kvôli všeobecnosti, RDF jazyk používa URI ako základ pre mechanizmus identifikácie subjektu, predikátu a objektu v tvrdeniach. Respektíve RDF používa URI odkazy (z angl. *URI reference*).

3.4 RDF model

Ako bolo spomínané v predošlej sekcií, URI odkazy nám slúžia na identifikáciu vecí v tvrdieniach RDF jazyka. Taktiež vieme, že každé RDF tvrdenie sa skladá z subjektu, predikátu a objektu.

Definícia 3.4.1 *RDF graf.* RDF tvrdenie pozostáva z kolekcie trojíc, kde každá trojica obsahuje subjekt, predikát, objekt. Kolekcia takýchto trojíc definuje RDF graf.

Teda RDF graf je reprezentovaný trojicou subjekt, predikát, objekt, kde subjekt a objekt je uzlom grafu a predikát definuje vzťah. Podstatný je smer šípky, ktorý začína v uzle subjektu a smeruje do uzlu objektu.



Obr. 3.1: Jednoduchý RDF graf.

Kapitola 4

Znalostné systémy

Znalostné systémy (z angl. *Knowledge based systems*) – tento pojem popisuje program, ktorý pozná a používa istý druh vedomosti na riešenie zložitejších problémov, teda v istom zmysle zahŕňajú veľké množstvo systémov. Ale podstatou, či princípom takéhoto systému je reprezentovať vedomosť pomocou určitého druhu nástroja, kde môže ísť o ontológiu alebo pravidlá. Spomínané systémy väčšinou zahŕňajú tri podsystémy:

- Vedomostný základ (z angl. *Knowledge base*).
- Uživateľské rozhranie.
- Záverový systém – ktorý využíva pravidlá na odvodenie novej informácie.

Vedomostný základ reprezentuje potvrdené fakty o svete, najčastejšie vo forme ontológie. Záverový systém reprezentuje logické tvrdenia a podmienky o svete, väčšinou vo formáte *IF-THEN* pravidiel [22].

4.1 Wikipédia

Wikipédia začala ako doplnkový projekt k projektu Nupédia. Projekt Nupédia, ktorá sa vo svojej dobe snažila byť voľnou online anglickou encyklopédiou. Jej články a príspevky boli napísané expertmi, ktoré boli následne preskúmané prostredníctvom formálneho procesu. Teda nové informácie sa neobjavovali tak často, ale zase ich obsah bol potvrdený. Nupédia bola založená 9. marca 2000 a jej obsah bol licencovaný.

4.1.1 História

Projekt Wikipédie bol odštartovaný začiatkom roka 2001, presnejšie 15. januára. Z počiatku encyklopédia bola podporovaná iba v jednom jazyku a to angličtine. Ďalšie jazyky bolí pridané neskôr. Nupédia a Wikipédia koexistovali spolu, až pokial sa zakladatelia nerozhodli permanentne vypnúť servery v roku 2003, následne sa obsah Nupédie začlenil do Wikipédie. Anglická verzia prekročila métu 2 milióny článkov 9. septembra 2007, čo z nej urobilo najväčšiu encyklopédiu, ktorú kto kedy zostrojil. Týmto Wikipédia vzala rekord *Yongle encyklopédií*, ktorá tento rekord držala skoro 600 rokov.

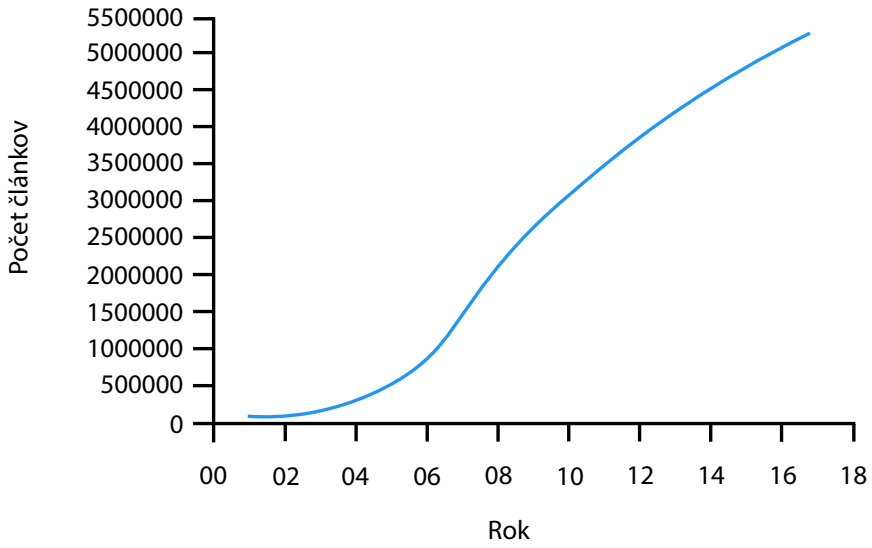
Z obavy komerčných reklám a nedostatočnej kontroly vo Wikipédií sa používatelia španielskej Wikipédie rozhodli ako protest vytvoriť *Enciclopedia Libre*. Stalo sa tak vo februári 2002. To podnietilo zakladateľov pôvodnej Wikipédie, aby vyhlásili, že Wikipédia nebude zobrazovať reklamy a taktiež zmenili doménu Wikipédie z *wikipedia.com* na *wikipedia.org*.

4.1.2 Veľkosť

Prvými prispievateľmi, ktorý začali publikovať na Wikipédií boli z projektu Nupédia. Ku dňu 8. augusta 2001 sa na Wikipédií vyskytovalo cez 8 tisíc článkov. Na konci septembra toho istého roku to bolo už cez 13 tisíc článkov. Pred koncom roka 2001 encyklopédia narástla na zhruba 20 tisíc článkov v osemnástich jazykových verziách. Dnes je to niečo vyše 5,5 milóna článkov. Na obrázku 4.1 je možné vidieť priebeh ako rástol počet článkov na encyklopédií v priebehu rokov. Pred rokom 2012 sa rast Wikipédie približoval Gompertzovej funkcií

$$y^{(t)} = ae^{be^{ct}} \tag{4.1}$$

s parametrami:
$a = 4378449$
$b = -15.42677$
$c = -0.384124$
$e = 2.71828$(Eulerovo číslo)
a t je čas v rokoch od 1.1.2000 (teda napríklad 1.1.2010 je t = 10.00).



Obr. 4.1: Počet článkov na Wikipédií za jednotlivé roky.
Zdroj: https://wikipedia.com/.

V júni 2015 výpis všetkých stránok Wikipédie s kompletnou históriou v XML formáte má veľkosť asi 10 TB, ak je obsah nekomprimovaný a približne 100 GB s využitím kompresie. Do tejto veľkosti sa nerátajú mediálne súbory ako obrázky a pod.

4.1.3 Wikidáta

Wikidáta je projekt kolektívne upravovanej podpornej databázy pre Wikipédiu. S návrhom projektu prišla nemecká pobočka nadácie *Wikimedia*, *Wikimedia Deutschland*. Wikidáta majú vytvoriť spoločné úložisko databázových údajov (napríklad dátumov narodenia), podobne ako je Wikimedia Commons spoločné úložisko pre fotografie alebo zvukové súbory, ktoré potom môžu byť používané inými projektmi.

Projekt bol spustený 30. októbra 2012. V tejto dobe bola dostupná iba fáza 1. Čo znamená, že artikly mohli byť vytvorené a vyplnené základnými údajmi, názvom (meno,

titulok), aliasmi (alternatívne názvy pre názov), popis a odkazy na články o danej téme v rozdielnych jazykových mutáciach Wikipédie (prvými dátami, ktoré sú centralizované na Wikidátach, sú medzijazykové odkazy prepájajúce články na Wikipédii v rôznych jazykoch) [4].

4.2 DBpédia

DBpédia je projekt zaoberajúci sa štrukturovaným obsahom z informácií vytvorených projektom Wikipédie (kapitola 4.1). Tieto informácie sú poskytované na internete. DBpédia tiež dovoľuje vyhľadávať dáta, ich vzťahy a vlastnosti, ktoré sú odvodené zo zdrojov Wikipédie, čo tiež zahŕňa odkazy do podobných datových sád.

DBpedia teda slúži na extrakciu štrukturovaných dát z Wikipédie a kladie si za úlohu tieto informácie publikovať širokej verejnosti na internete. Ako bolo spomenuté DBpédia je zložená z viacerých služieb, jedna z nich dovoľuje pýtať sa zložité, sofistikované otázky voči Wikipédií a prepájať dáta z internetových stránok na dáta z Wikipédie. Ľudia, ktorý stoja za DBpédiou dúfajú vo využitie dát novými spôsobmi. A to hlavne v navigácií, prepájaní a zlepšovaní encyklopédie samotnej.

4.2.1 Veľkosť extrahovaných dát

Anglická verzia znalostného systému DBpédie popisuje 4,58 milióna vecí, z čoho je 4,22 milióna klasifikovaných v konzistentnej ontológií. DBpédia používa RDF jazyk (3) na reprezentáciu extrahovaných informácií.

Ontológia systému DBpédie zahŕňa 1 445 000 osôb, 735 tisíc lokalít (čo zahŕňa 478 tisíc miest), 411 tisíc kreatívnych prác (čo zahŕňa 123 tisíc hudobných albumov, 87 tisíc filmov a 19 tisíc video hier), 241 tisíc organizácií (čo zahŕňa 58 tisíc spoločností a 49 tisíc vzedavacích inštitúcií), 251 tisíc druhov a 6 tisíc chorôb.

Projekt DBpédie samozrejme poskytuje dáta v ďalších 125 jazykoch. Všetky tieto verzie spolu popisujú 38,3 milióna vecí, z ktorých je 23,8 milióna lokalizovaných popisov vecí, ktoré taktiež existujú v anglickej verzií DBpédie. Celá dátová sada DBpédie poskytuje 38 miliónov popiskov a abstraktov v pätnástich rozdielnych jazykoch, 25,2 milióna odkazov na obrázky a 29,8 milióna odkazov na externé internetové stránky, 80,9 milióna odkazov na kategórie Wikipédie a nakoniec 41,2 milióna odkazov na kategórie YAGO. DBpédia je ako taká prepojená s ostatnými dátovými sadami zhruba pädesiatimi miliónmi RDF odkazov. Dokopy DBpédia 2014 pozostáva z 3 miliárd kúskov informácií (RDF trojice), z ktorých bolo 580 miliónov extrahovaných z anglikej verzie Wikipédie, 2,46 miliardy bolo extrahovaných z ostatných jazykových mutácií. Detailné štatistiky o dátových sadách DBpédie v 24 populárnych jazykoch, ktoré sú poskytované na internetovej stránke DBpédie.

4.2.2 Aspektové vyhľadávanie v dátach DBpédie

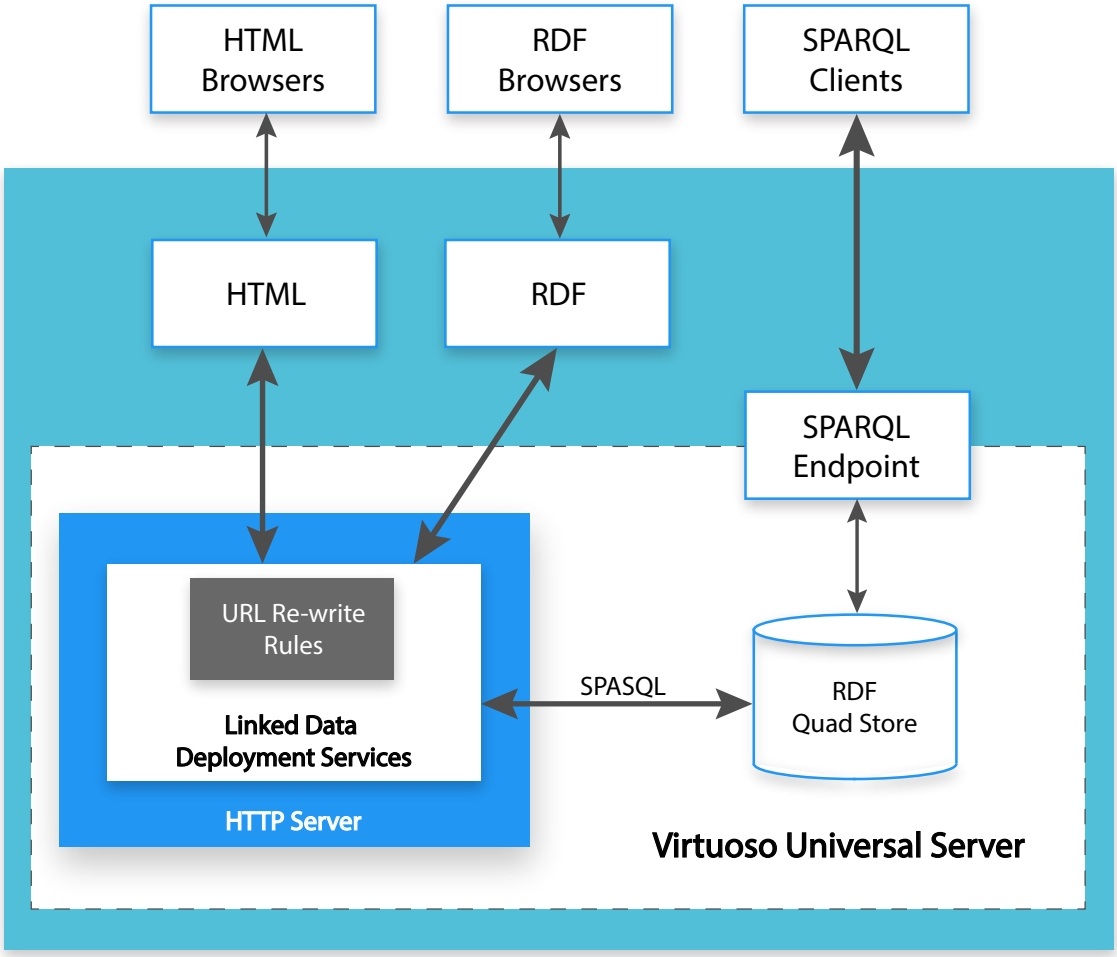
DBpédia extrahuje aspektové informácie z jednotlivých stránok Wikipédie. Užívateľom je dovolené pýtať sa otázky nad týmito dátami. Dáta sú prístupné pomocou jazyka podobného jazyku SQL zvaného *SPARQL*. Napríklad, ak by sme chceli vyhľadať vizuálneho umelca, ktorý je holandského pôvodu a jeho maľbu, SPARQL požiadavok by mohol vyzerať následovne:

```
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX db: <http://dbpedia.org/resource/>
```

```
SELECT ?artist ?painting
WHERE {
        ?artist dbprop:nationality db:Dutch .
        ?painting dbprop:createdBy ?artist .
}
```

Výpis 4.1: SPARQL dotaz

Systém DBpédie je hostovaný a publikovaný pomocou *OpenLink Virtuoso*. Infraštruktúra systému Virtuoso poskytuje prístup k RDF dátam pomocou SPARQL prístupového bodu. Taktiež disponuje podporou internetových požiadavkov na získanie internetovej stránky alebo RDF reprezentáciu zdrojov DBpédie.



Obr. 4.2: Infraštruktúra virtuoso systému.
Zdroj: https://virtuoso.openlinksw.com/.

4.2.3 Štruktúra extrahovaných dát

Všetky dáta, ktoré DBpédia poskytuje boli extrahované z výpisov samotnej Wikipédie. S každým novším výpisom je vydaná nová dátová sada ale aj ontológia. Extrahovaná dátová sada je poskytovaná oddelene v rôznych súboroch. Taktiež celá sada je lokalizovaná. To

znamená, že existujú rovnaké sady súborov pre rôzne jazyky. Takže je možné si vybrať z niekoľkých jazykov. Okrem spomínaných rozdelení je možné si vybrať z rôznych serializačných formátov:

- Turtle(ttl) – poskytuje dáta vo formáte n-trojíc (<predmet> <predikát> <objekt> .) ako podmnožinu Turtle serializácie [31].
- Quad-turtle(tql) – dáta sú serializované pomocou quad turtle serializácie, čo vyzerá následovne (<predmet> <predikát> <objekt> <graph/context> .). Pridá sa teda kontext, informácia pre každú trojicu obsahujúca meno grafu a pôvodnú informáciu [30].

Turtle

Ako bolo spomenuté vyššie, tento formát obsahuje trojice popísané formátom:

<predmet> <predikát> <objekt> .

Teda obsah súboru by mohol vyzerať následovne:

```
<http://dbpedia.org/resource/Anarchism>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Thing> .

<http://dbpedia.org/resource/Autism>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Disease> .

<http://dbpedia.org/resource/Achilles>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Thing> .
```

Výpis 4.2: Turtle formát.

Quad turtle

Quad turtle formát vyzerá následovne:

<predmet> <predikát> <objekt> <graph/context> .

```
# started 2015-11-06T08:48:44Z
<http://dbpedia.org/resource/Anarchism>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Thing>
<http://en.wikipedia.org/wiki/Anarchism?oldid=683845221#section=
   Etymology_and_terminolog&relative-line=2&absolute-line=19> .

<http://dbpedia.org/resource/Autism>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Disease>
<http://en.wikipedia.org/wiki/Autism?oldid=682116152#absolute-line=10> .
```

```
<http://dbpedia.org/resource/Achilles>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Thing>
<http://en.wikipedia.org/wiki/Achilles?oldid=682099597#section=
   External_links&relative-line=9&absolute-line=279> .
```

Výpis 4.3: Quad turtle formát.

Okrem rozdielnej serializácie niektoré dátové sady sú dostupné v dvoch verziách:

- Lokalizované – táto dátová sada obsahuje trojice extrahované z korešpondujúcej Wikipédie, zahŕňajúc tie, ktorých URI *nemajú* ekvivalent v anglickej Wikipédií.
- Kanonizované – tieto dátové sady obsahujú trojice taktiež extrahované z korešpondujúcej Wikipédie, ktorých predmet a objekt *majú* ekvivalent v anglickej Wikipédií.

Celá dátová sada je logicky rozdelená do súborov podľa toho aké vzťahy sú dátami popisované. Jednotlivé súbory môžu popisovať vzťahy ako typy zdrojov, odkazy na druhé dátové sady, zaradenie do kategórií, citácie, externé odkazy, geografické koordináty, údaje o osobách, domovské stránky, odkazy na stránky Wikipédie a mnoho ďalších.

Kapitola 5

Systémy ukladania štruktúrovaných dát

Už vieme odkiaľ a aké dáta sú dostupné z teoretického hľadiska. Na uloženie takýchto dát sa používa najčastejšie nejaký databázový systém. Tento systém musí byť schopný dáta uchovávať a rýchlo v nich vyhľadávať.

Aby systémy reagovali rýchlo, pre veľké množstvo dát a s nízkou odozvou používajú tieto systémy určité výkonnostné vylepšenia. Najčastejšie je reč o využívaní určitého druhu cacheovania. Ide zväčša o optimalizáciu, zrýchlenie z časového hľadiska na úkor zväčšenia pamäťových nárokov. Ide teda o druh pamäte, ktorý má nízku dobu prístupu avšak musí spĺňať určité princípy tak, aby bolo toto úložisko využívané efektívne. Spomínaný princíp je princípom lokality, ktorý hovorí, že aplikácia pracuje v jednu chvíľu len s malou časťou svojho pamäťového priestoru. Avšak existujú dva typy lokálneho prístupu. Dočasná lokalita (z angl. *temporal locality*) znamená, že použitý pamäťový blok bude znovu použitý v blízkej budúcnosti. Druhý typ je priestorová lokalita (z angl. *spatial locality*), teda pokiaľ je použitý pamäťový blok, je pravdepodobné, že čoskoro bude použitý ďalší v jeho tesnej blízkosti [25].

Cacheovanie v systémoch Na úrovni celých systémov hovoríme o cacheovaní celej odpovede servru. Systémy na takúto cache používajú veľa mechanizmov od predpočítavania výsledkov, predvytvorenie metadát, šablón a pod. Na úrovni databázových systémov, čiže systémov, ktoré sú schopné ukladať štrukturované dáta, hovoríme o cacheovaní požiadavkov. Niektoré databázove systémy majú zabudovanú pamäť, kde sa ukladajú požiadavky v samotnom jadre systému ako je to v prípade *MySQL* a v podobných databázových systémoch. Avšak je možné ako cache považovať aj ukladanie vypočítaných výsledkov pre znovu použitie [23]. Pri využití na úrovni systému, ak sa preskočí všetko počítanie a vrátí naposledy vypočítaná odpoveď, takto sa vieme dostať k takmer konštantnému časovému prístupu, resp. k niekoľkonásobnému zrýchleniu systému.

5.1 Databázový systém

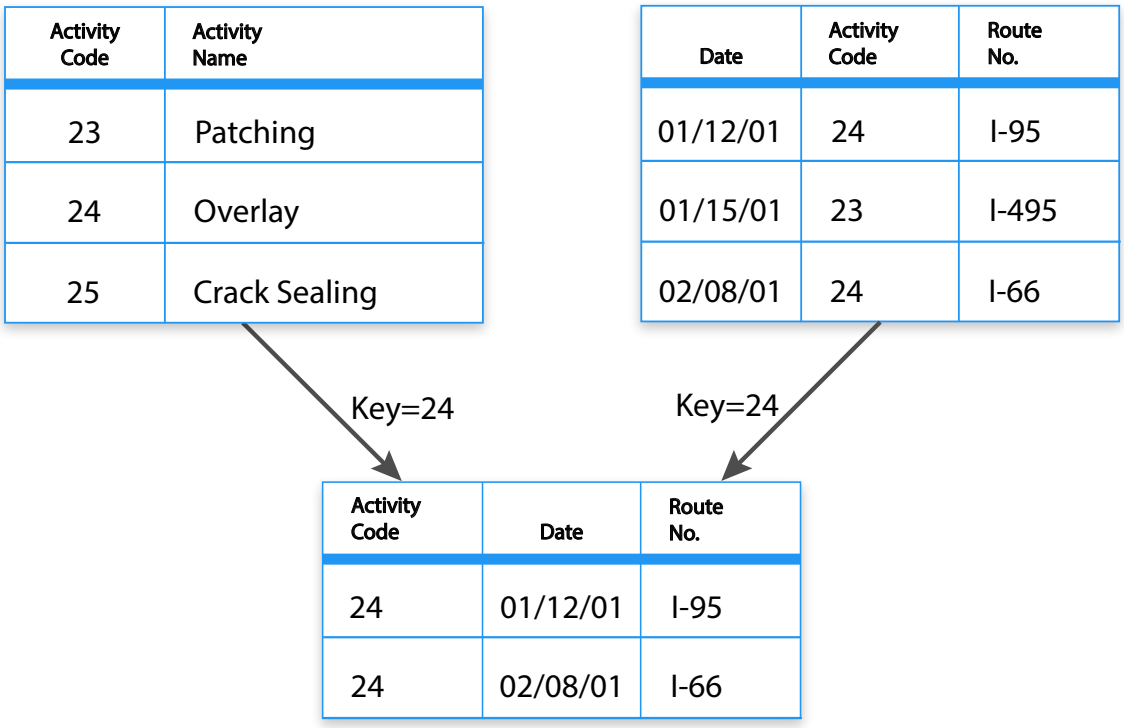
Definícia 5.1.1 *Databáza.* Databáza je zvyčajne veľká kolekcia dát, ktoré sú organizované sa účelom rýchleho vyhľadávania a nájdenia výsledkov.

Nad databázou fungujú systémy manažmentu databázy (z angl. *Database management system – DBMS*). Ich úlohou je interagovať s koncovými používateľmi, druhými aplikáciami a aj so samotnou databázou za účelom zachytiť a analyzovať dáta [6].

Samotná databáza nie je všeobecne prenositeľná naprieč rôznymi systémami manažmentu databázy. Ale systémy manažmentu môžu interagovať s rôznymi databázami. Systémy manažmentu databázy sa klasifikujú na základe databázových modelov [13].

5.2 Relačné databázové systémy

Medzi najpopulárnejšie patrí *relačný model*. Tento model podporujú najznámejšie systémy ako Oracle, MS-SQL Server, MySQL a ďalšie. Ostatné tradičné modely ako hierarchické dáta a sieťové dátové modely sa stále používajú v priemyslnej sfére, kvôli ich zložitosti. Tieto druhy databázových systémov využívajú na reprezentáciu dát tabuľky, kde riadky tabuľky predstavujú jednotlivé záznamy a stĺpce popis jednotlivých políčok. Väčšinou sú tabuľky prepojené vzťahmi, či väzbami [21]. Ukážka prepojenia tabuliek vzťahmi je uvedená na obrázku 5.1.



Obr. 5.1: Relačný model.

Relačné dátabázové systémy na vyhľadanie a nájdenie dát v databáze používajú SQL jazyk – štruktúrovaný dopytovací jazyk (z angl. *Structured Query Language*). Tento jazyk slúži na manipuláciu (výber, vkladanie, úpravu a mazanie) a definíciu údajov v databáze. V súčasnosti je to najpoužívanejší jazyk tohto druhu.

Avšak posledných rokoch boli predstavené nové objektovo-orientované dátové modely. Tento model je v systéme databázového manažmentu reprezentovaný vo forme objektov ako je tomu v objektovo orientovanom programovaní. Objektovo orientované systémy databázového manažmentu kombinujú výhody klasických (relačných) databázových systémov s výhodami objektovo orientovaných jazykov.

5.3 NoSQL databázové systémy

Tieto typy databázových systémov poskytujú mechanizmy pre úložisko a vyhľadávanie dát, ktoré nie sú modelované pomocou tabuľkových vzťahov ako je tomu u relačných databázových systémov. NoSQL systémy sa vo väčšine používajú v big-data a internetových aplikáciach bežiacich v reálnom čase. Niektoré systémy aj napriek názvu môžu podporovať nejaký dopytovací jazyk podobný jazyku SQL.

NoSQL databázové systémy sa dostávajú do popredia a veľa spoločností ich výhody začína používať. Avšak poznáme niekoľko druhov NoSQL systémov vzhľadom na zameranie systému a typu používaných dát. Rozdelenie systémov podľa dátového modelu vyzerá následovne:

- Stĺpcové.
- Dokumentové.
- Kľúč-hodnota.
- Grafové.
- Multimodelové.

5.3.1 Kľúč-hodnota

Fundamentálnym základom tohto modelu je úložisko, ktoré používa asociatívne pole (tiež známe ako mapa alebo slovník). V tomto modeli sú dáta reprezentované ako kolekcia párov kľúč-hodnota s tým, že kľúč sa môže v kolekcii vyskytovať iba raz. Medzi tieto patria softvérové databázy napr. ArangoDB, InfinityDb, Oracle NoSQL Database, Redis.

5.3.2 Dokumentové úložisko

Základnym konceptom dokumentového úložiska je predstava „dokumentu“. Každý dokumentovo orientovaný systém predpokladá, že dokument zabaľuje a kóduje dáta (alebo informáciu) v jednom zo štandardných formátoch a kódovaniach. Medzi používané kódovania patrí XML [28], YAML [33], JSON [5], ale aj binárne formy ako BSON. Dokumenty sú adresované v databáze pomocou unikatného kľúča, ktorý reprezentuje celý dokument. Ďalšou charakteristikou dokumentovo orientovaného systému je to, že systém poskytuje aplikačné rozhranie alebo dopytovací jazyk, ktorý nájde dokumenty vzhľadom na ich obsah. Rozdielne implementácie ponúkajú rôzne smery organizovania a zhlukovania dokumentov. Medzi základné typy patrí:

- Kolekcie.
- Tagy.
- Neviditeľné metadáta.
- Priečinková hierachia.

V porovnaní s relačným databázovým systémom, kolekcie môžu byť analogicky považované za tabuľky a dokumenty ako jednotlivé záznamy v tabuľke. Avšak sú rozdielne, pretože každý záznam v tabuľke má rovnakú sekvenciu políčok, ktoré obsahuje, ale dokument v kolekcií môže obsahovať políčka, ktoré sú úplne odlišné.

5.3.3 Grafové databázy

Tento druh databázy je navrhnutý pre dáta, ktorých vzťahy sú veľmi dobre reprezentovateľné ako graf. Tento graf pozostáva z elementov, ktoré sú navzájom poprepájané konečným množstvom vzťahov. Typy dát môžu byť napríklad sociálne vzťahy, verejná hromadná doprava, cestné mapy alebo sieťová topológia [20].

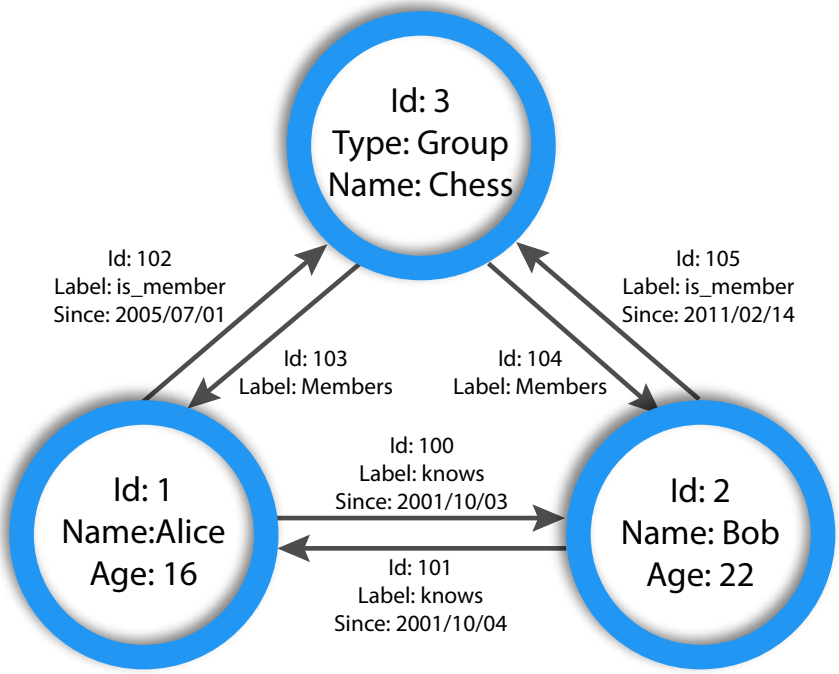
Názov	Jazyk
AllegroGraph	AQL, JavaScript, GraphQL
DEX/Sparksee	C++, Java, .NET, Python
FlockDB	Scala
IBM DB2	SPARQL
InfiniteGraph	Java
MarkLogic	Java, JavaScript, SPARQL, XQuery
Neo4j	Cypher
OpenLink Virtuoso	C++, C#, Java, SPARQL
Oracle	SPARQL 1.1
OrientDB	Java, SQL
OWLIM	Java, SPARQL 1.1
Sqrrl Enterprise	Java

Tabuľka 5.1: Grafové databázové systémy a ich (dopytovacie) jazyky

Grafové databázové systémy sú založené na grafovej teórií. To znamená, že dáta sú reprezentované grafom. Graf sa skladá z následujúcich častí:

- Uzly – reprezentujú entity ako napríklad ľudí, obchody, účty alebo ľubovoľnú vec, ktorá má byť dohľadateľná.
- Hrany – tiež nazývané vzťahy, sú čiary medzi jednotlivými uzlami, ktoré ich spájajú a reprezentujú vzťah medzi uzlami.
- Vlastnosti – informácie, ktoré patria uzlu.

V porovnaní s relačnými databázami sú grafové často rýchlejšie pre asociatívne dátové sady. A taktiež sa skôr približujú štruktúre objektovo-orientovaných aplikácií. Škálujú sa viac prirodzene pre veľké dátove sady, pretože nepotrebujú typické *JOIN* operácie, ktoré sú výpočtovo náročné. Ukážka takéhoto systému ilustruje obrázok 5.2.



Obr. 5.2: Grafový model.

5.3.4 Porovnanie výkonu

Výkonnosť a škálovateľnosť databázových systémov sa vykonáva pomocou benchmarku známeho pod skratkou YCSB, čo znamená „Yahoo! Cloud Serving Benchmark“. Táto otvorená špecifikácia sa využíva k porovnaniu relatívnej výkonnosti NoSQL systémov databázového manažmentu.

Dátový model	Výkon	Škálovateľnosť	Flexibilita	Zložitosť	Funkčnosť
Kľúč-hodnota	vysoký	vysoká	vysoká	žiadna	premenná (žiadna)
Stĺpcový	vysoký	vysoká	stredná	nízka	minimálna
Dokumentový	vysoký	premenná(vysoká)	vysoká	nízka	premenná(nízka)
Grafový	premenný	premenná	vysoká	vysoká	grafová teória
Relačný	premenný	premenná	nízka	stredná	relačná algebra

Tabuľka 5.2: Výkonnosť databázových systémov

Kapitola 6

Existujúce riešenia

Prakticky väčšina eshopových a im podobných riešení podporuje určitý druh aspektového vyhľadávania. Napríklad pri vstupe do určitej sekcie obchodu sa zobrazí tabuľka, kde je možné si vybrať z nejakých parametrov – facetov.

Každopádne táto práca sa zaoberá aspektovým vyhľadavaním nad indexom Wikipédie. Na prvý pohľad by sa mohlo zdať, že Wikipédia poskytuje aspektový vyhľadávač no v skutočnosti tomu tak nie je. Existujú, či existovali vyhľadávače tretích strán, ktoré poskytovali aspektové vyhľadávanie alebo nejakú jeho podobu.

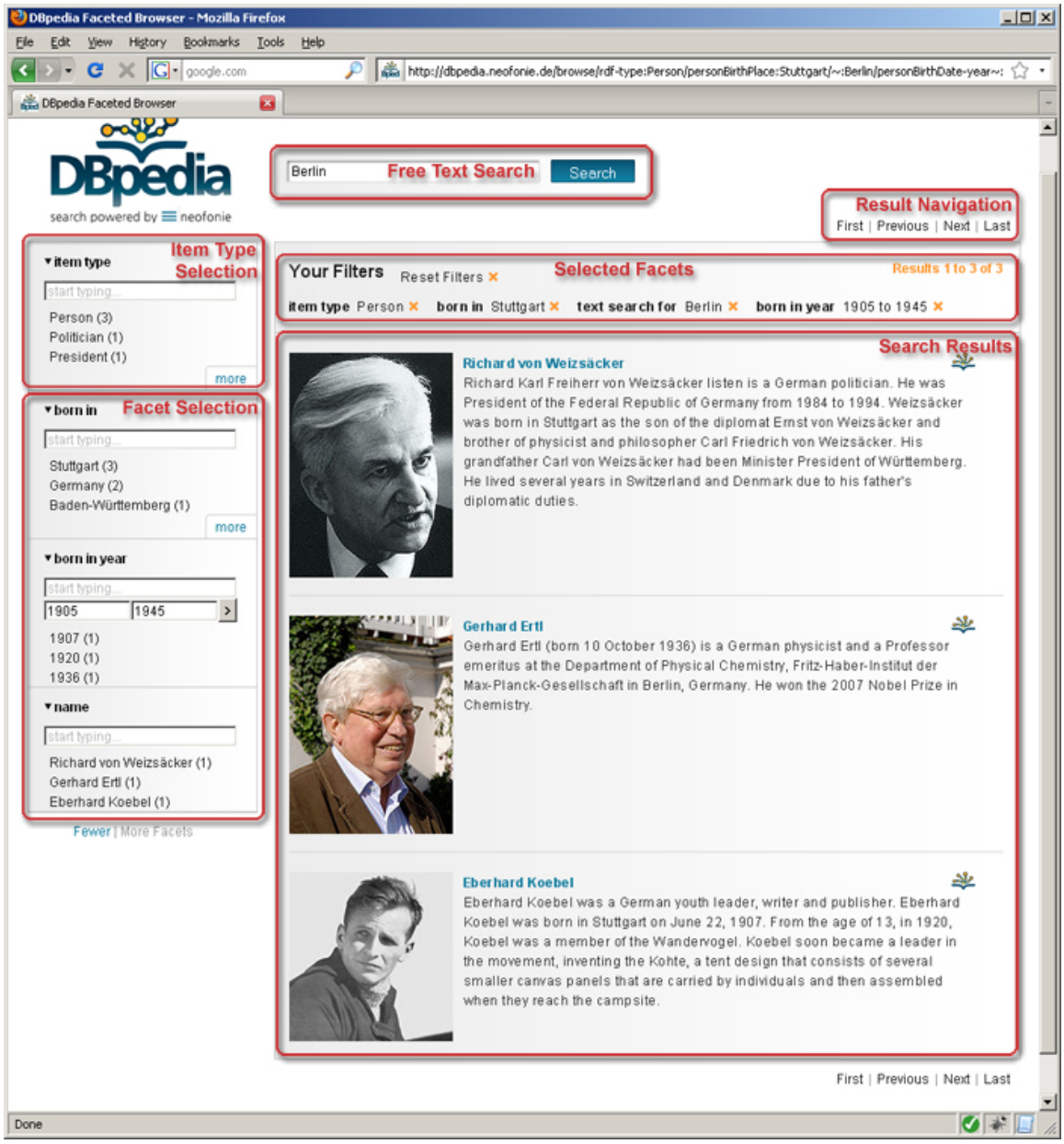
6.1 Faceted Wikipedia Search

Systém umožňuje používateľom klásť komplexné otázky ako „Ktorá rieka sa zlieva do Rýnu a je dlhšia ako 50 kilometrov?“ alebo „Ktorý mrakodrap v Číne má viac ako 50 poschodí a bol postavený pred rokom 2000?“. Odpovede na tieto otázky neboli generované pomocou vyhľadávania kľúčovými slovami ako to robia vyhľadávače ako Google alebo Yahoo ale boli generované na základe štruktúrovaných informácií, ktoré boli extrahované z rôznych článkov Wikipédie. Všetky obrázky v tejto sekcií boli prevzaté z [9].

6.1.1 Užívateľské rozhranie

Rozhranie tohto systému pozostávalo z niekoľkých interagujúcich komponent, ktoré môžeme vidieť na obrázku 6.1. V [9] je spomínaných niekoľko hlavných komponent:

- Vstupné pole pre kľúčové slová.
- Komponenta zobrazujúca vybrané aspekty pre vyhľadávanie.
- Výber typu vyhľadávanej entity.
- Výber jednotlivých ďalších aspektov.
- Komponenta vykresľujúca nájdené výsledky.
- Komponenta pre stránkovanie výsledkov.



Obr. 6.1: DBpédia – aspektový vyhľadávač.

Jednotlivé komponenty sú dobre rozložené na stránke, ich význam je jednoznačný a jasný. Avšak zobrazovanie aktuálneho nastavenia aspektové filtra zaostáva za prepracovanosťou ostatných komponent. Veľkou výhodou tohto systému je, že výsledky aktuálneho požiadavku sú zobrazované vo forme kariet. Karty sú jasne grafický rozlíšené a poskytujú základne informácie spolu s náhľadovým obrázkom. Aj napriek roku, v ktorom systém vznikol pôsobí pokročilejšie, ako systém spomenutý v podkapitole 6.2. Je vidieť, že jeho tvorcovia si dali záležať pri jeho návrhu a tvorbe.

6.1.2 Pozadie systému

Systém bol vyvinutý niekoľkými spoločnosťami. Konkrétne išlo o spoločnosti *neofonie GmbH* a *Web-based Systems Group at Freie Universität Berlin*. Technicky je tento systém založený na dátach poskytovaných projektom DBpédia a na vyhľadávacom systéme spoločnosti *neofonie*.

Vyhľadávací systém *neofonie* poskytoval vykonanie komplexných požiadavkov nad extrahovanými dátami. Vyhľadávač taktiež agregoval RDF dáta z DBpédie s fulltextovými dátami z Wikipédie. Agregované dáta boli rozdelené do hierarchických aspektov.

Naneštastie tento systém, ani žiadna jeho časť, nemohla byť ďalej poskytována a teda tento systém bol z verejnej stránky stiahnutý v roku 2012 [9].

6.2 Faceted Search & Find service

Tento systém je aktuálne poskytovaný samotnou DBpédiou a je dostupný na `http://dbpedia.org/fct/`. Je možné si vytvárať vlastnú požiadavku pridávaním podmienok do aspektového filtra, avšak nie je možné tento filter ručne upravovať. Pridávať môžeme iba prednastavené aspekty z už vyhľadaných entít alebo vymazať aspektovú podmienku [3]. Užívateľské rozhranie si podrobnejšie zanalyzujeme v následujúcich sekciách a následne zhodnotíme akým dojmom toto užívateľské rozhranie pôsobí. Všetky obrázky v tejto sekcií boli prevzaté z [3].

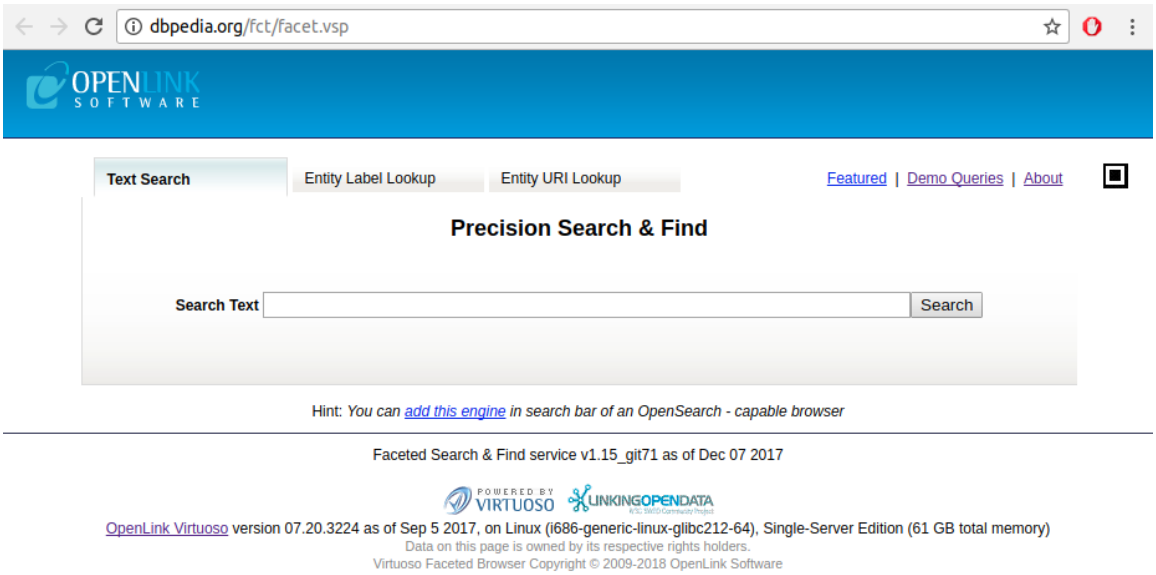
6.2.1 Užívateľské rozhranie

Samotné rozhranie pozostáva z niekoľkých obrazoviek. Najpodstatnejšie tri sú:

- Úvodná stránka – obrázok 6.2.
- Stránka s vyhľadanými výsledkami – obrázok 6.3.
- Stránka zobrazujúca detail entity – obrázok 6.4.

Úvodná stránka

Táto stránka slúži ako hlavný rozbočovač pri textovom, štítkovom a URI vyhľadávaní. Úvodná stránka by mala pôsobiť jednoduchým dojmom a mala by implementovať niektoré zo základnych návrhových vzorov, ktoré sú nutné na úspech vyhľadávacieho rozhrania [15]. Tieto vyhľadávacie vzory si prejdeme neskôr v kapitole 7.

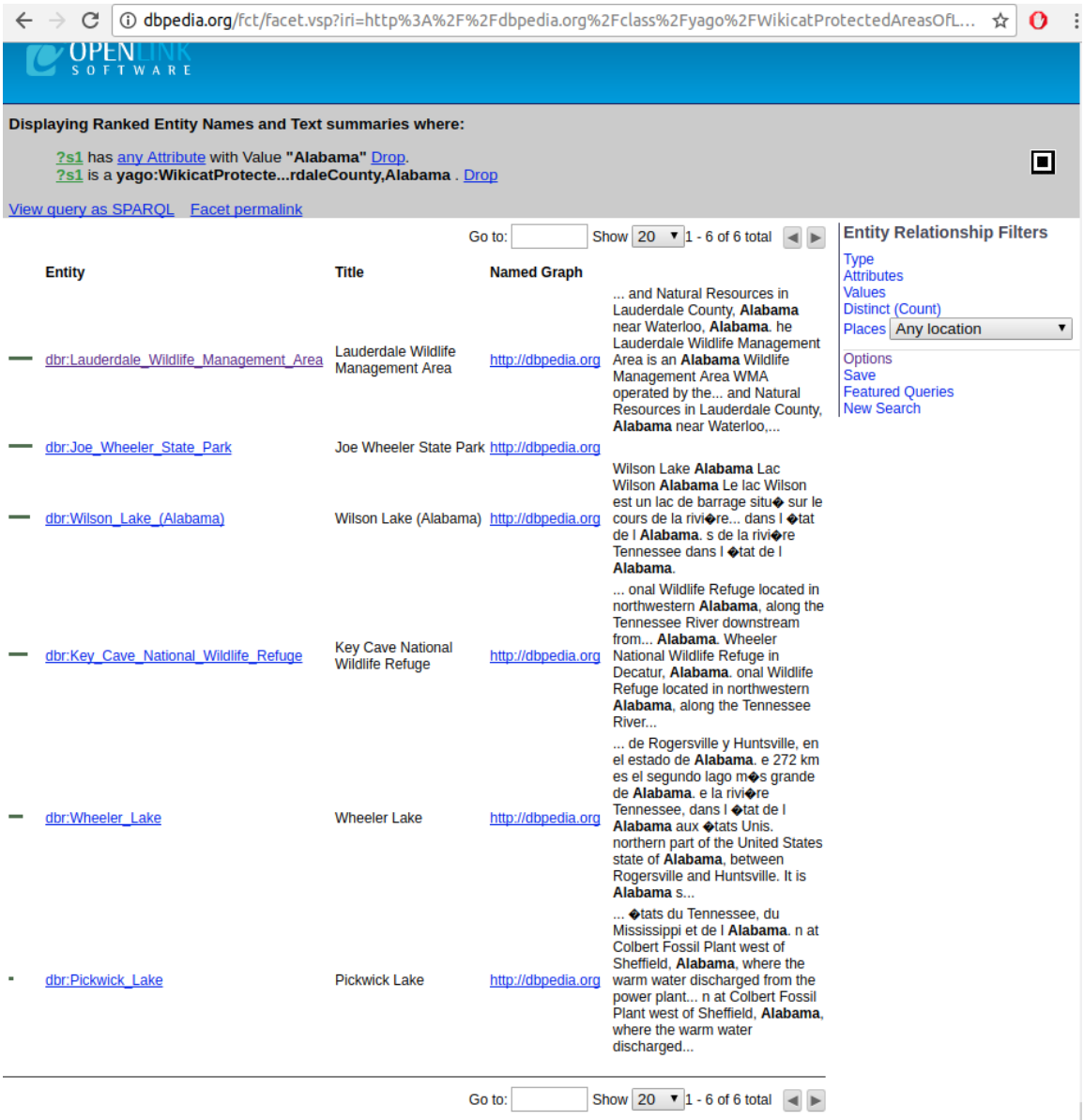


Obr. 6.2: DBpédia – úvodná stránka aspektového vyhľadávača.

Ako najväčší nedostatok môžeme vidieť, že úvodná stránka je pomerne komplexná a umožňuje hneď niekoľko vecí, čo môže zmiasť užívateľa pokiaľ chce vyhľadávať iba na základe vstupného textu. Ďalším nedostatkom je, že hlavné vstupné políčko nemá implementované automatické dopĺňanie textu, teda užívateľ nevie, či pod daným kľúčovým slovom existuje nejaký výsledok. Taktiež sa na hlavnej stránke nachádza veľké množstvo odkazov kam môže užívateľ kliknúť. Niektoré odkazy vedú na prázdne stránky, ktoré nemajú skoro žiadnu vypovednú hodnotu.

Vyhľadané výsledky

Stránka tohto typu by mala byť jednoduchá na použitie. Užívateľ by mal zvládnuť základné používanie bez nutnosti študovania manuálu. Každopádne na tejto stránke je možno nájsť veľké množstvo informácií. Niektoré by mohli byť skryté pred užívateľovým zrakom a neplýtvať tak miestom na obrazovke.



Obr. 6.3: DBpédia – tabuľka vyhľadaných výsledkov.

Stránka je delená na niekoľko logických celkov, ktore môžme vidieť na obrázku 6.3. Jednotlivé časti obrazovky si priblížime a vysvetlíme. Medzi najpodstatnejšie celky, ktoré môžeme vidieť patria:

- Komponenta zobrazujúca vybraté aspekty pre vyhľadávanie.
- Komponenta vykresľujúca nájdené výsledky, v ktorej je zahrnuté aj stránkovanie.
- Komponenta pre filtrovanie vzťahov.

Aktuálny aspektový filter predstavuje výpis v hornej časti stránky so šedým pozadím, kde môžme vidieť použité aspekty. Grafická podoba aspektového filtra je zastaralá a niekedy užívateľ nevidí načo sa pozerá. Pre dlhšie názvy predikátov sa zobrazujú skrátené verzie, teda nevieme aké aspekty sa v skutočnosti nachádzajú vo filtri.

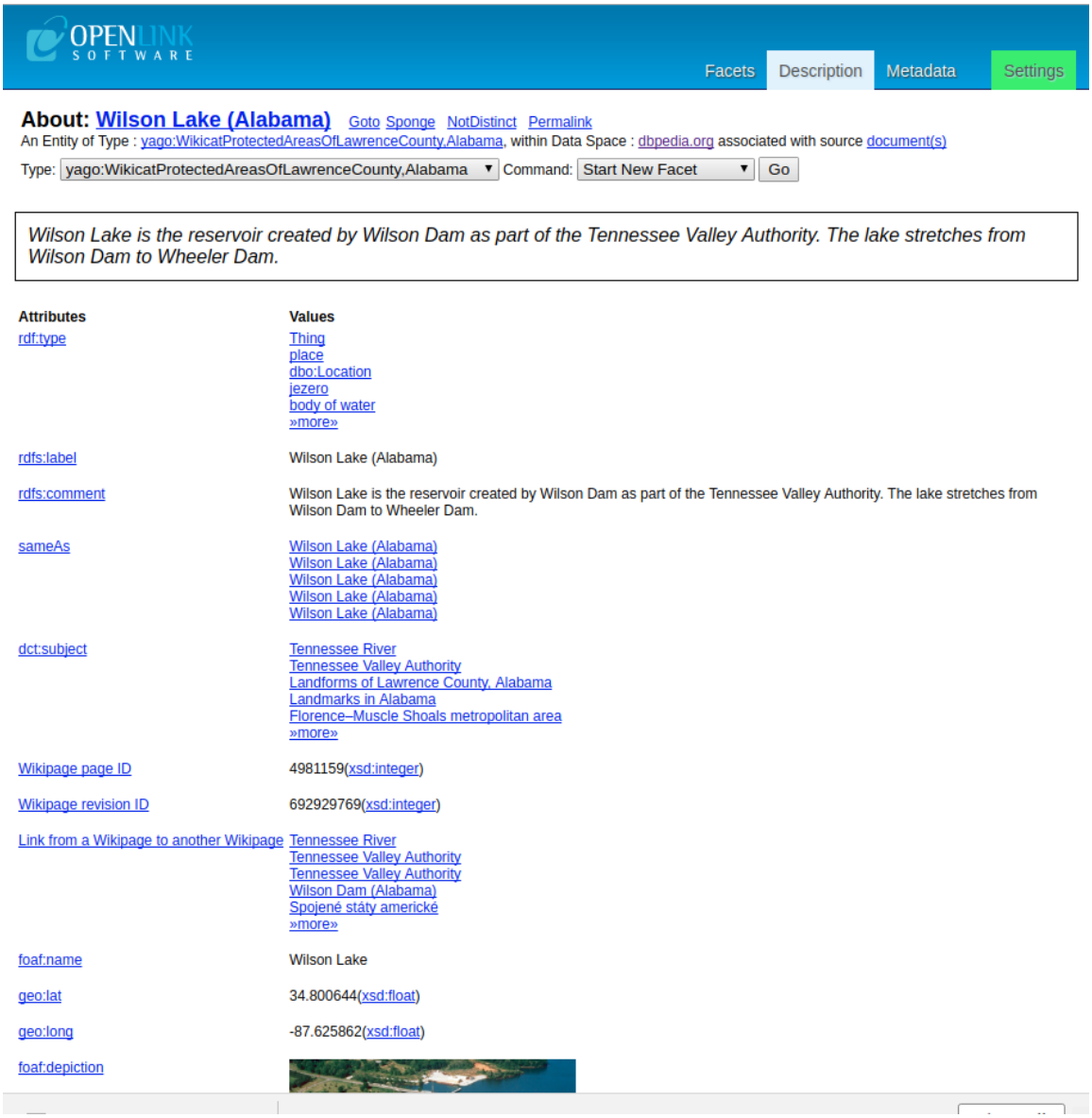
Tabuľka nájdených výsledkov sa trochu vymyká štandardom. Výsledky tabuľky nepôsobia jednodnačným dojmom. Riadky tabuľky zobrazujú rozsiahly text, ktorý nie je podstatný pri vyhľadavaní ale až v detaile entity. Avšak v tabuľke je možno vidieť aj vyextrahovaný názov entity. V tabuľke sa však nemusí vyskytovať celý popis entity a ten môže byť teda zobrazený až v detaile.

Nakoniec v pravej časti obrazovky môžme vidieť filter vzťahov, ktorý sa používa na definovanie pravidiel do aspektového filtra. Filter obsahuje základne prekliky do dialógov, ktoré nám pomôžu vybrať si správny aspekt. Preklikávanie medzi dialógmi je trochu zmätočné, niekedy nevieme pre ktorý predikát vyberáme objekt. Taktiež v tomto filtri sú ďalšie nastavenia a odkazy, dokonca je možné si nastavenie filtra uložiť.

Detail entity

Detail je stránkou, ktorá nám má priblížiť jednu z entít, ktoré sme si vyhľadali. Jej popis môže byť rozsiahly s rôznymi ďalšími informáciami a odkazmi, ktoré by užívateľ mohol zúžitkovať. Avšak stránka by mala byť prehľadná, aby užívateľ vedel na aký typ dát sa pozerá.

Na stránke nájdeme všetky potrebné informácie. Je možné vidieť typ entity, všetky aspekty ako aj ich hodnoty. Na tejto stránke je možné upraviť aspektový filter a pridať doň nový riadok popisujúci ďalší aspekt.



Obr. 6.4: DBpédia – detail vyhľadanej entity.

Stránka obsahuje istý druh pozmenej tabuľky kde prvý stĺpec tvorí názov atribútu a ďalší jeho hodnoty. Klikateľné odkazy sú jasne odlíšené. Horná časť však obsahuje veľa informácií na jednom mieste, takže užívateľ, ktorý by so systémom pracoval, by si musel dávať pozor na čo kliká.

6.2.2 Budúcnosť

Systém má byť v budúcnosti rozšírený o ďalšiu funkcionalitu. Má ňou byť vizualizačný nástroj, v ktorom bude možné upravovať vygenerovaný SPARQL požiadavok. Systém bude môcť využívať poznatky z predchádzajúcich užívateľských vyhľadávaní. Tie môžu byť následne použité na prispôsobovanie vyhľadavania.

Posledná revízia systému bola 7. decembra 2017, teda vyzerá, že systém je vylepšovaný a poskytovatelia služby dbajú na jeho aktuálnosť.

6.2.3 Zhodnotenie rozhrania

Vstupná stránka systému vyzerá síce trochu zastarane, obsahuje veľa zbytočností, ktoré môžu užívateľa odradiť od jeho používania, ale účel splní. Tým účelom je dostať sa k vyhľadávaniu. Avšak v úvodnej stránke nie sú implementované základne vyhľadávacie vzory ako je automatické dokončovanie a doplňovanie, čo je kritickým prvkom pri vyhľadávacom systéme. Ak nie je tento vzor implementovaný, nemôžeme užívateľovi poskytnúť ani normálne zoraďovanie podľa najvyhľadávanejších výsledkov a ani im jednoduchým spôsobom povedať, že používajú zlé kľúčové slovo. Zobrazovanie výsledkov je taktiež zastarané, a zobrazujú sa aj dáta, ktoré užívateľov nemusia zaujímať. Tabuľka má totiž stĺpce, ktoré užívateľ nevyužije. Samotný detail entity je zložený z obrovského množstva informácií, ale užívateľ nevidí skutočný popis dát, pretože sú prefixované skratkami. To znemožňuje prenos dát z jedneho systému do druhého, ktorý pracuje s približne podobnou dátovou sadou.

Každopádne úspech systému by sa dal vylepšiť implementovaním rôznych vyhľadávacích vzorov, doplnením funkcionality, ktorá je sľubovaná a taktiež, ak sa v systéme implementuje lepšie zobrazovanie dát, ktoré nebude zastarané.

Kapitola 7

Návrh

Pri návrhu užívateľského rozhrania sme vychádzali z poznatkov a výskumov vo vyhľadávaní, čo nám pomáhalo lepšie pochopiť správanie užívateľov, ich chovanie pri vyhľadávaní a pri prezeraní výsledkov. Po pochopení potrieb a chovania užívateľov pri vyhľadávaní by sme mali byť schopní navrhnúť lepší prístup vyhľadávania s využitím patričných vzorov [15]. Medzi spomínané vzory, ktoré sa oplatí v našom systéme použiť patria:

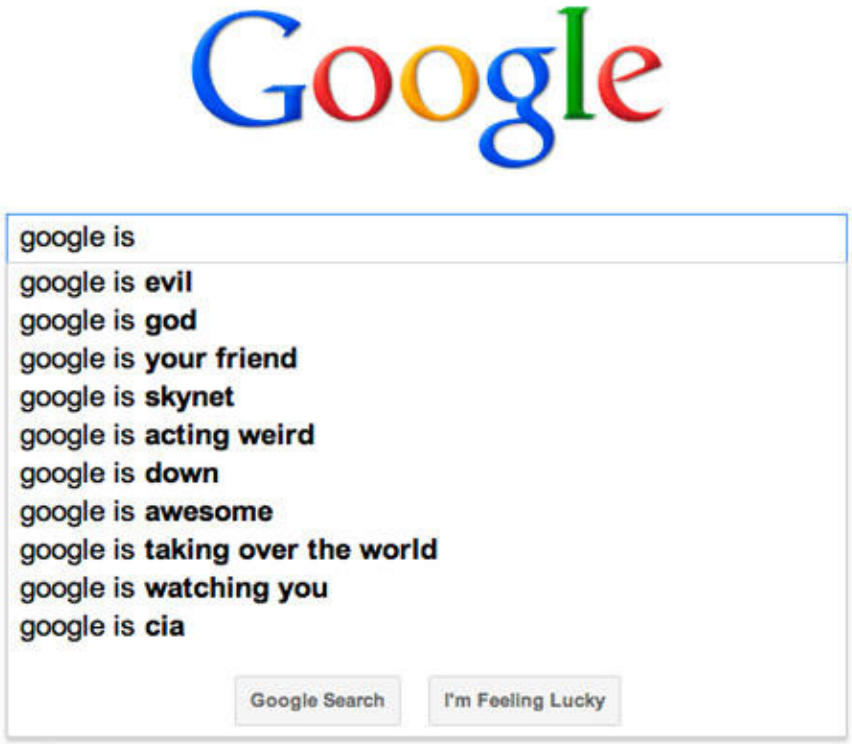
- Automatické dopĺňanie a predpoveď výsledkov (z angl. *autocomplete and autosuggest*).
- Najlepšie výsledky prvé.
- Aspektová navigácia/vyhľadávanie.
- Personalizácia.
- Stránkovanie.
- Štruktúrované výsledky.
- Výsledky nabádajúce k akcii.
- Jednotný objav.

Pri využití niektorého zo spomínaných vzorov je treba prehodnotiť očakávania používateľov od systému. Či nejaký z vyhľadávacích vzorov skôr neodradí uživateľa od používania systému. Preto je vhodné si položiť otázku, ktoré zo vzorov je vhodné použiť a implementovať. Následujúce podkapitoly sa budú venovať jednotlivým vyhľadávacím vzorom, ktoré boli presnejšie definované v knihe [15].

7.1 Automatické dopĺňanie a predpoveď výsledkov

Automatické dopĺňanie je vlastnosť, kedy je aplikácia schopná predikovať koniec slova, ktoré užívateľ vpisuje do príslušného vstupného políčka. Spravidla môže užívateľ potvrdiť výber kliknutím na predikovanú položku. Avšak je možné využiť aj klávesnicu, kde pomocou šipiek alebo použitím tlačidla *TAB* je možné navigovať na konkrétny návrh, ktorý následne potvrdíme stlačením tlačidla *ENTER*. Dopĺňanie má dopomôcť užívateľovi vybrať si relevantnejší výsledok, taktiež však slúži na zrýchlenie interakcie užívateľa so systémom.

Každopádne pri využití tohto vzoru sa počíta s istou stratou informácie, pretože užívateľovi sa nezobrazujú všetky možné výsledky ale len podstatne menšia podmnožina dát, ktorá je však upravená podľa určitých pravidiel. S čím súvisí ďalší vzor 7.2 .



Obr. 7.1: Automatické dopĺňanie na stránkach spoločnosti Google.
Zdroj: https://google.com/.

7.2 Najlepšie výsledky prvé

Tento vzor súvisí s vyhľadávaním a usporiadaním výsledkov na výstupe. Uvedený problém je možné sformulovať ako problém relevancie a radenia. V prípade, že hľadáme všeobecnú informáciu, výsledkom býva obrovské množstvo záznamov, ktoré väčšinou nebývajú zoradené podľa abecedy, ale podľa miery relevancie. V našom prípade bude miera relevancie mierou popularity stránky na Wikipédií alebo počet zobrazení stránky a podobne.

7.3 Paginácia

Tento vzor je celkom dôležitý keďže pracujeme s veľkým množstvom dát. Základnou myšlienkou tohto vzoru je rozdelnie výsledkov do stránok, ktoré je možné si zobrazovať, ako keby sme listovali stránky v knihe. Je to prakticky o zmenšení zobrazovaného množstva dát za účelom prehľadnejšieho rozhrania, aby nedochádzalo k jeho presýteniu a zneprehliadneniu. Pri správnej relevancii zobrazovaných výsledkov, by sa však všetky údaje mali zobraziť na prvej stránke. V prípade, že najrelevantnejšie výsledky nebudú na prvej stránke, užívateľ môže prehliadať stránky a dôjsť k požadovaným výsledkom. Z tohto problému sa treba poučiť a prepracovať algoritmus relevantnosti dát tak, ako to je spomenuté v podkapitole 7.2.

Obr. 7.2: Paginácia na stránkach spoločnosti Google.
Zdroj: https://google.com/.

Podľa výskumov publikovaných v [11] až 68% užívateľov vyhľadávacích nástrojov kliká na výsledky vyhľadávania na prvej strane a až 92% užívateľov klikne na výsledky v rámci prvých troch strán.

	2008	2006	2004	2002
Prvá strana nájdených záznamov	68%	62%	60%	48%
Prvé dve strany nájdených záznamov	17%	19%	20%	23%
Prvé tri strany nájdených záznamov	7%	9%	8%	10%
Viac ako tri strany nájdených záznamov	8%	10%	12%	19%

Tabuľka 7.1: Správanie užívateľov pri klikaní na rôzne stránky vyhľadávania.

Z tabuľky 7.1 je jasné, že používatelia sú zvyknutí, že najrelevantnejšie záznamy sú umiestňované ako prvé a podľa toho sa aj správajú [11].

7.4 Štruktúra, akcia, objavovanie...

Skupina vyhľadávacích vzorov, ktorá núti užívateľa pracovať so systémom je založená na vzoroch, ktoré sa viažu k zobrazovaniu dát. Štruktúra hovorí o jednoduchom a hlavne jednotnom zobrazovaní, kde každý výsledok alebo typ výsledku je zobrazený rovnakým spôsobom. Pojmy akcia a objavovanie hovoria o tom, aby boli výsledky zobrazované atraktívnym spôsobom pre používateľa. Taktiež vedú k tomu, aby bol užívateľ nabádaný ku kliknutiu na výsledok a bolo jasné kam má klikať.

7.5 Aspektové vyhľadávanie

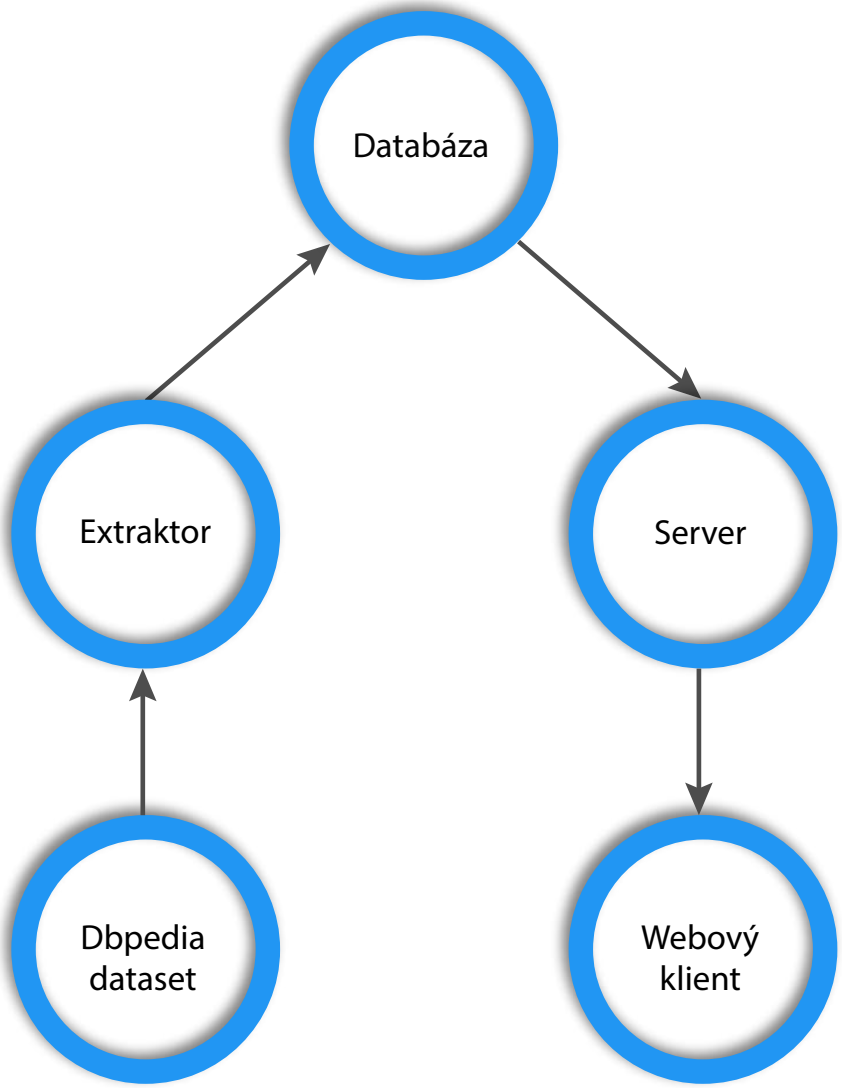
Aspektová navigácia alebo aspektové vyhľadávanie využíva určitý druh aspektovej klasifikácie, ktorá predstavuje proces klasifikácie prostredníctvom aspektového klasifikačného systému. Základný princíp takejto klasifikácie je založený na jednoduchých kategóriach (aspektoch), ktoré vyjadrujú všeobecné pojmy organizácie. Na rozdiel od jednoduchej hierarchickej schémy aspektová klasifikácia umožňuje nájsť položky podľa viac, ako len jednej dimenzie. Podľa odborníkov aspektové vyhľadávanie predstavuje významný pokrok v oblasti vyhľadávania. Taktiež predstavuje techniku prístupu k množstvu informácií, ktoré využíva aspektovú klasifikáciu. Užívatelia môžu hľadať informácie tak, že si nastavujú rôzne filtre. Aspektová navigácia, predstavuje vynikajúci prístup na zlepšenie používateľskej skúsenosti pri vyhľadávaní informácií. Taktiež tento prístup nachádza široké uplatnenie predovšetkým v elektronických obchodoch, ale aj pri vyhľadávaní v digitálnych knižniciach [26].

7.6 Navrhovaný systém

Uživateľské rozhranie sme sa snažili navrhnúť s ohľadom na vyhľadávacie postupy a prístupy spomenuté v predošlých podkapitolách. Taktiež sa treba poučiť z chýb existujúcich riešení, ktoré sú spomenuté v kapitole 6. Avšak netreba zabúdať na rozšíriteľnosť celého systému. A to z pohľadu funkčnosti samotného vyhľadávania ako aj možnosti rýchlej zmeny použitého databázového systému. Kvôli týmto požiadavkám sme sa rozhodli celý systém rozdeliť do niekoľkých častí:

- Extrakciou a uloženie dát do databázového systému.
- Vyhľadávací systém.
- Grafické uživateľské rozhranie.

Takého rozdelenie je nutnosť, ak by sme chceli systém udržiavať, verzovať a po častiach implementovať novú funkcionalitu. Taktiež je možné nezávisle vylepšovať jednotlivé časti a napríklad urýchliť chod systému, vyhľadávanie alebo doplniť nové informácie a pod.



Obr. 7.3: Návrh systému.

Každá časť systému bude teda samostatná a bude možné ju vymeniť, či vylepšovať nezávisle na ostatných. Samotné rozhranie vykonávajúce vyhľadávanie bude možné vyvíjať samostatne. Čo umožní dopĺňať funkcionalitu, meniť dizajn ale aj meniť spôsob zadávania vstupných parametrov.

Vyhľadávací systém bude postavený na presne definovaných vstupných dátach a podľa toho bude aj zohľadňovať vyhľadávanie. Pre správnu funkcionalitu je nutné dodržať vstupný formát dát. Vyhľadávanie je adaptérom medzi vstupnými dátami, databázovým systémom a výstupnými dátami. Extrakciu dát bude sprostredkúvať skript, ktorý spracuje súbor a jeho obsah uloží do databázy. Pre implemenovanie nového databázového systému do vyhľadávania bude samozrejme nutné poskytnúť extraktor, ktorý nahrá dáta do požadovaného databázového systému s podporovanou schémou dát.

7.7 Dátová štruktúra

Navrhnuté systémy budú musieť dodržiavať presnú štruktúru dát, aby bola zaručená spätná kompatibilita s ďalšími časťami systému a aby bolo možné systémy vymieňať a nedošlo ku komplikáciam za behu celého systému. Štruktúra dát bude presne definovaná v nasledujúcej podkapitole.

7.7.1 Vyhľadávacie dáta

Vyhľadávací server bude pracovať s dátami, ktoré budú zakódované do reťazca pomocou serializačného formátu *JSON*, pretože je najrozšírenejším serializačným formátom, ktorý má syntax, ktorou sa zapisujú javascriptové objekty. Taktiež jeho kompaktnosť umožňuje zasielať veľké množstvo dát. Výstupné dáta budú zakódované pomocou formátu *JSON*. Nato, aby sme mohli vyhľadávať pomocou aspektov a tiež aj pomocou kľúčového slova sme navrhli štruktúru vstupných dát. Táto štruktúra bude zahŕňať políčko pre kľúčové slovo a pole pomocou ktorého sa bude filtrovať na základe zvolených aspektov. Samozrejmosťou bude aj políčko, v ktorom sa nadefinuje paginátor. Pre vyhľadávanie bude teda štruktúra dát bude vyzerať nasledovne:

```
{
  "query": <SEARCH_KEYWORD>,
  "paginator": {
    "page": <WANTED_PAGE>,
    "items_per_page": <WANTED_ITEMS_PER_PAGE>
  },
  "aspects": [
    {
      "predicate": {
        "uri": <UNIQUE_PREDICATE_URI>,
        "op": <COMPARISON_OPERATOR>
      },
      "object": <UNIQUE_OBJECT_URI>
    }
  ]
}
```

Výpis 7.1: Štruktúra vstupných dát.

7.7.2 Dáta pre automatické dopĺňanie

Pre implementovanie dopĺňania návrhov pri vyhľadávaní budeme odosielať jediný parameter a to vyhľadávané slovo, na základe ktorého sa budú dopĺňať výsledky. Počet výsledkov bude obmedzený, pretože v návrhoch nebudeme potrebovať zobrazovať veľké množstvo záznamov.

Vstup

Vstupom pre dopĺňanie bude štruktúra popísaná nasledovne. Tá vyžaduje iba zaslanie kľúčového slova, ktoré sa bude dopĺňať.

```
{
```

```
  "query": <SEARCH_KEYWORD>
}
```

Výpis 7.2: Štruktúra vstupných dát dopĺňania.

Výstup

Výstupom dopĺňania bude niekoľko výsledkov, ktoré budú zoradené zostupne podľa počtu výskytov v dátovej sade. Najviac spomínaná entita bude zobrazená ako prvá, tak ako to býva pri vyhľadavání s dopĺňaním.

```
{
  "values": {
    "<UNIQUE_SUBJECT_URI>": {
      "name": "<DECODED_SUBJECT_NAME>",
      "predicates_count": <PREDICATES_COUNT>,
      "occurencies_count": <OCCURENCIES_IN_DATASET>
    },
    .
    .
    .
  }
}
```

Výpis 7.3: Štruktúra výstupných dát dopĺňania.

7.8 Extrakcia dát

Je zrejme, že databázových systemov je hneď niekolko druhov, tak ako bolo spomenuté v kapitole 5. Náplňou práce je taktiež porovnávať rôzne systémy a teda bude dobré, ak systém bude podporovať vyhľadávanie v relačnej databáze a taktiež aj vo vybranej grafovej databáze.

7.8.1 MySQL

Pre tento typ databázového systému, bude vhodné rozdelenie dát na viacero typov, pričom každý typ bude uložený vo vlastnej tabuľke s tým, že bude mať presne definované vzťahy.

7.9 Návrh vyhľadávacieho systému

Táto časť, by sa mala starať ako vstupné dáta v presne definovanej podobe prekonvertuje na vybraný požiadavok, podľa výberu databázového systému. Následne sa tento požiadavok použije na vyhľadanie dát v databáze. Tieto dáta sa následne spracujú a odošlú ako odpoveď.

Systém bude implementovaný vo forme REST[1]-ového aplikačného rozhrania, ku ktorému bude môcť pristupovať viacero aplikácií zároveň. Toto rozhranie bude pomocou URL rozdelené na niekoľko častí, každá časť bude spolupracovať s priradeným databázovým systémom. Napríklad metódy, ktoré budú vyhľadávať v relačnej resp. grafovej databáze by mohli mať nasledujúce URL:

[1] https://en.wikipedia.org/wiki/Representational_state_transfer

- *http://example.com/mysql/api/.*
- *http://example.com/graph/api/.*

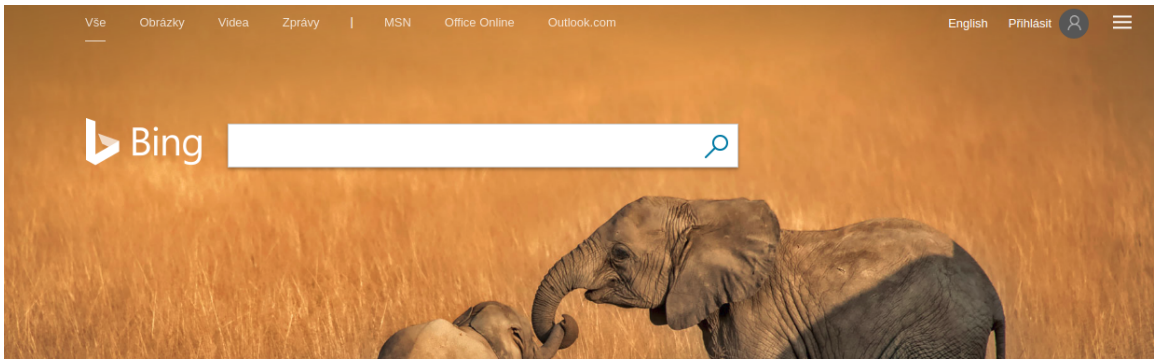
Pod týmito URL sa rozhranie bude ďalej deliť na niekoľko metód, kde niektoré budú slúžiť na rýchle automatické dopĺňanie do užívateľského rozhrania, ale aj komplexné vyhľadávanie pomocou aspektového filtra.

Server by mal hlavne rýchlo reagovať aj na zložitejšie požiadavky od mnohých aplikácií. S týmto problémom by malo pomocť implementovať nejaký druh cachingového systému, ktorý môže byť na úrovni cachovania celej odpovede servru alebo aj menších celkov, napríklad na úrovni databázových požiadavkov.

7.10 Návrh grafického uživateľského rozhrania

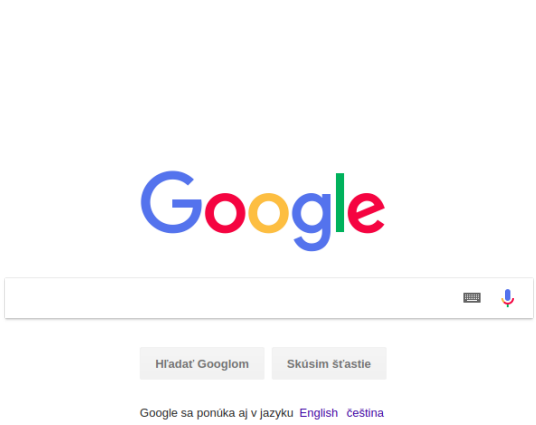
Grafické užívateľské rozhranie, bude interagovať s koncovým užívateľom. V tomto rozhraní bude môcť užívateľ zadať kľúčové slovo, podľa ktorého chce vyhľadávať. Taktiež si užívatelia budú môcť upravovať a vytvárať aspektový filter. Aspektový filter by mal podporovať viac operátorov, ktoré môže užívateľ využívať, ako zadanie vzťahu, ktorý entita musí mať, alebo naopak, kde entita taký vzťah nesmie obsahovať. Toto rozhranie bude poskytovať aj preddefinované nastavenia aspektového filtra, čo znamená, že si užívateľ bude môcť vybrať už z nejakých preddefinovaných typov entít alebo požiadavkov.

V tomto rozhraní by mali byť využité najnovšie trendy ako *Material Design* a vyhľadávacie vzory, ktoré boli spomenuté na začiatku kapitoly 7. O samotnom *Material Design-e* si povieme v podkapitole 7.11.



Obr. 7.4: Vyhľadávač Bing od spoločnosti Microsoft.
Zdroj: https://www.bing.com/.

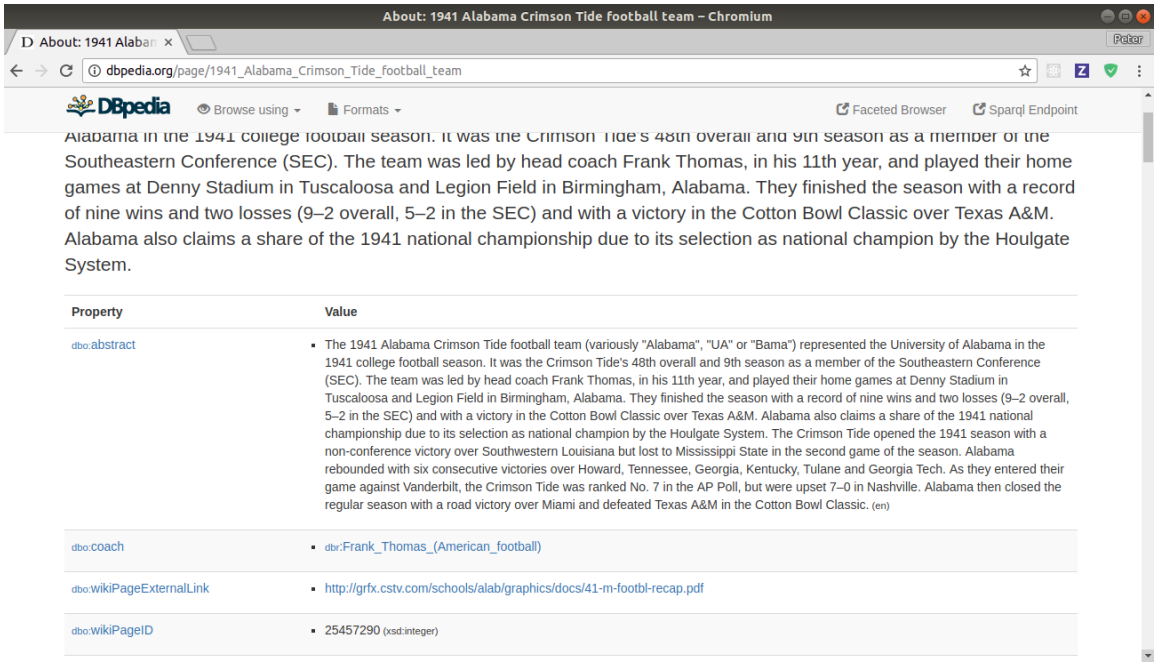
Väčšina vyhľadávacích systémov je rozdelených na niekoľko častí. Najhlavnejšiu časť tvorí vstupná stránka. Táto stránka by mala byť graficky príťažlivá a mala by zaujať a urobiť dobrý prvý dojem na užívateľa. Príklad stránky tohto typu môžeme vidieť na obrázkoch najznámejších vyhľadávačov a to na obrázku 7.4 a obrázku 7.5.



Obr. 7.5: Vyhľadávač od spoločnosti Google.
Zdroj: https://www.google.com/.

Hlavná stránka by mala implementovať automatické dopĺňanie, aby naviedla užívateľa na správne vyhľadávanie. Táto stránka je vlastne prostredníkom k skutočnému vyhľadávaniu.

Ďalšou stránkou by mala byť stránka s vyhľadanými výsledkami, ktorá by mala byť jednoduchá, štruktúrovaná s jednoznačným miestom, kde treba kliknúť, tak aby sme sa dostali ku konkrétnemu výsledku. Nakoniec v našom prípade to bude aj stránka, ktorá bude zobrazovať údaje o konkrétnom výsledku, teda ide o stránku s detailom entity. V systéme DBpédie sú dostupné stránky jednotlivých entít. Ako taká stránka môže vyzerať je možné vidieť na obrázku 7.6.



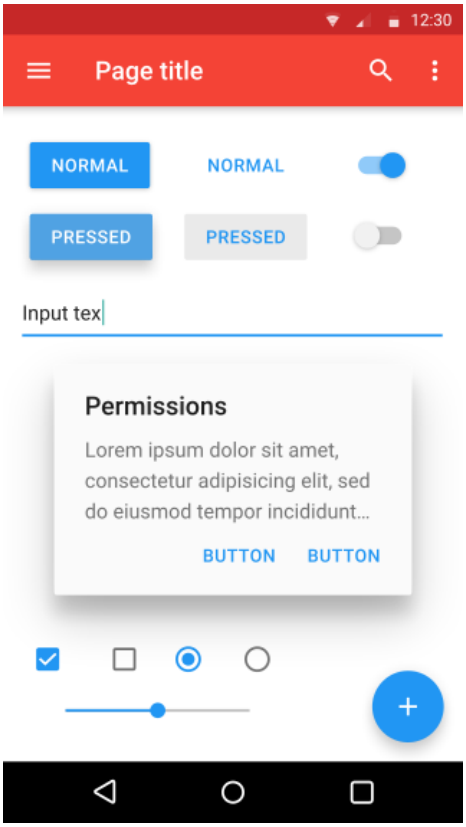
Obr. 7.6: Detail entity na stránkach DBpédie.
Zdroj: http://dbpedia.org/page/1941_Alabama_Crimson_Tide_football_team.

7.10.1 Aspektový filter

Najtažšou časťou bude vymyslieť logiku zadávania jednotlivých aspektov do aspektového filtra tak, aby s ním bolo možné jednoducho pracovať pre koncového užívateľa. Nie je ťažké zadávať už presne nadefinované názvy predikátov alebo ich hodnoty. Problémom je nadefinovať podmienku, ktorá je závislá na nejakom už zadanom filtrovacom pravidle. Taktiež z pohľadu užívateľského rozhrania je nutné ponúknuť rozumný mechanizmus zadávania logických operátorov medzi filtrovacie pravidla.

7.11 Material design

Material Design je unifikovaný systém, ktorý kombinuje teóriu, zdroje a nástroje na vytváranie digitálneho užívateľského rozhrania. Jedná sa o dizajnový jazyk, ktorý bol vytvorený spoločnosťou *Google*.



Obr. 7.7: Možnosti Material design-u.
Zdroj: https://material.io/.

Obr. 7.8

Systém otvára možnosti využitia mriežkovo založených rozloženiach, responzívnych animácií a prechodov, odsadení a hĺbkových efektov ako sú svetlo a tiene. Tento systém sa najskôr usídlil v mobilných zariadeniach s operačným systémom Android. No dnes ho využívajú skoro všetky webové aplikácie spoločnosti *Google* ako napríklad Gmail, YouTube, Google Drive, Google Docs, Sheets, Slides, Google Maps a mnoho ďalších. Väčšina užívate-

ľov sa s týmto dizajnom zžila, pretože ho najskôr bolo možné nájsť v mobilných telefónoch, no v dnešnej dobe nie je problém ho využiť aj vo webovej aplikácií [8].

Na obrázku 7.8 je možné vidieť rôzne možnosti ako využívať črty tohto systému. Tlačidlá, ktoré sú vyfarbené, nevyfarbené, s tieňom, bez neho a podobne. Taktiež je možné si všimnúť tiene na dialógu, ktoré nám dodávajú priestorový pocit.

Kapitola 8

Implementácia

Táto kapitola vychádza z poznatkov a chýb existujúcich riešení a taktiež z kapitoly 7, ktorá popisuje ako by riešenie mohlo vyzerať. Teda podľa návrhu implementovaný systém pozostáva z niekoľkých samostatných častí:

- Extraktor.
- Webový vyhľadávací server – REST-ové vyhľadávacie rozhranie implementujúce logiku vyhľadávania.
- Webový klient – implementujúci samotné používateľské vyhľadávanie, zadávanie aspektov, kľúčového slova a pod.

8.1 Extrakcia dát

Prvou časťou systému je program, ktorý sa stará o ukladanie dát zo súborov DBpédie do databázy. Tento program obsahuje niekoľko podporovaných databáz, z ktorých si používateľ systému bude môcť vybrať pri nasadzovaní systému. Teda tak, ako bolo spomenuté v návrhu, extraktor implementuje relačnú aj grafovú databázu a to z dôvodu porovnania ich využiteľnosti a výkonnosti.

Extraktor je napísaný v jazyku *Python* a je teda optimalizovaným skriptom, ktorý spracúvava priečinok súborov, kde sú extrahované dáta poskytnuté DBpédiou v špecifickom formáte. Na prácu s databázou je použitý mapovač objektov na relačné dáta *SQLAlchemy*, ktorý rieši načítavanie a ukladanie objektov do databázy.

Pre extrakciu je nutné dodať súbory, ktoré sa majú spracovať. Podľa požadovaného typu vyhľadávaných údajov je možné nechať do databázy spracovať viacero súborov s odlišnými informáciami. Popis súborov aké informácie obsahujú je možné nájsť v kapitole 4.2, kde je napísané čo taký súbor môže obsahovať. Výber súborov záleží od významu a použitia vyhľadávania.

Samotná extrakcia prebieha spracovaním súborov, kde je jeden súbor spracovávaný po riadkoch. Každý riadok je skontrolovaný, či spĺňa požadovaný serializačný formát pomocou regulárnych výrazov. Konkrétne:

```
(?P<subject><URI regex>) (?P<predicate><URI regex>) (?P<object><URI regex>) .
```

Výpis 8.1: Regulárny výraz pre spracovanie jedného riadku súboru.

Kde časť *<URI regex>* je nahradená skutočným regulárnym výrazom pre URI, ktorý vyzerá následovne:

```
^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
```

Výpis 8.2: Regulárny výraz pre URI.

8.1.1 Relačná databáza

S využitím ORM[1] nad relačnou databázou sme navrhli základné typy entít, ktoré sú vhodné, a to takým spôsobom, že je možné ich rýchlo spracuvávať a nekladú veľké nároky na databázový systém pri vkladaní. Indexy a ďalšie optimalizačné faktory boli na databázu aplikované až po úspešnom vložení dát, kvôli urýchleniu vkladania. Implementované typy entít:

- SimpleMapping.
- SubjectMetadata.
- QueryCache.

Tieto objekty stačia na správne fungovanie systému. Objekty, ich vlastnosti a úloha v systéme budú následne popísané.

Jednotka vyhľadávania

Tento objekt prakticky predstavuje trojicu. Pri vyhľadávaní sa tento objekt používa v aspektovom filtri. Schéma objektu zapísaná v jazyku *Python* a frameworku *SQLAlchemy* vyzerá nasledovne:

```
class SimpleMapping(Base):
  __tablename__ = "simple_mapping"
  id = Column(Integer, primary_key=True)
  subject = Column(Unicode(500))
  predicate = Column(Unicode(500))
  object = Column(Unicode(500))
```

Výpis 8.3: Objekt SimpleMapping.

Ako je možné vidieť objekt je veľmi jednoduchý. Uchováva informácie o trojici zo vstupného súboru. Ak budeme vyhľadávať, tak sa budeme pýtať na unikátnu entitu, ktorú predstavuje políčko *subject*.

Metadáta

Metadáta sú uložené vo vlastnej tabuľke a tieto záznamy sa v databázovom systéme vytvoria až po uložení skutočných dát. Teda vypočítavajú sa z dát typu *SimpleMapping*.

```
class SubjectMetadata(Base):
  __tablename__ = "subject_metadata"
  id = Column(Integer, ForeignKey(SimpleMapping.id))
  name = Column(Unicode(100), default='Undefined')
  unique_uri = Column(Unicode(500), primary_key=True)
  predicates_count = Column(Integer())
  occurencies_count = Column(Integer())
```

Výpis 8.4: Objekt SubjectMetadata.

[1] Object relational mapper

Tieto dáta slúžia na zoraďovanie entít podľa počtu výskytov v dátovej sade. Avšak majú tiež informačný charakter. Informujú o tom, či sa výsledky približujú tomu čo mal užívateľ v pláne vyhľadávať. Pokiaľ je počet výskytov nízky môže to indikovať, že sa táto entita v dátovej sade nevyskytuje v takom množstve a vlastne nemusí predstavovať to, čo chcel užívateľ vyhľadať, ale niečo, čo sa svojím názvom môže danej entite približovať.

Dopytová cache

Relačná databáza využíva cacheovací systém pre urýchlenie vyhľadávania. Kedže databáza je statická a jej obsah sa po prvom vložení dát nebude meniť, je možné tento fakt zúžitkovať. Teda ak nejaký užívateľ systému niečo vyhľadal, výsledok vyhľadávania sa uloží a nie je nutné ho znovu počítať ale iba znovu použiť. To znamená, že čím viac sa bude systém používať, tým sa bude systém zrýchľovať.

```
class QueryCache(Base):
  __tablename__ = "subject_metadata"
  id_query_cache = Column(Integer, primary_key=True)
  hash = Column(Unicode(512))
  data = Column(JSON())
```

Výpis 8.5: Objekt QueryCache.

Princíp využitia cache je celkom jednoduchý. Vstupné parametre vyhľadávaia sa serializujú a zahašujú pomocou algoritmu *SHA512*, ak je takýto hash uložený v databáze, tento záznam sa načíta a použíje sa výsledok z databázy. Ak tento hash v databáze nie je, tak sa štandardne vyhľadá na základe parametrov a výsledok sa uloží do databázy zoserializovaný pomocou formátu *JSON* a následne sa čerstvo vypočítaný výsledok použije.

Vkladanie dát

Vkladanie dát do relačnej databázy prebiehal načítávaním trojice predmet, predikát a objekt a vložením záznamu do databázy bez dotazovania, či daný záznam existuje. Po načítaní všetkých dát sa spustil ešte *SQL* skript, ktorý upravil tabuľky a doplnil indexy a cudzie kľúče, čo viedlo k zrýchleniu pri samotnom vyhľadávaní. Následne po týchto úkonoch sa spustil skript, ktorý vygeneroval z dostupných uložených dát metadáta ako počet výskytov v dátovej sade, počet predikátov, vyextrahovaný názov entity a nakoniec aj abstrakt.

8.1.2 Grafová databáza

V systéme bola taktiež využitá grafová databáza. Z grafových databázových systémov bol vybraný systém *Neo4j*, ktorý je najvyužívanejšou grafovou databázou.

V tomto systéme sme navrhli dve jednoduché entity:

- Neo4jResource.
- Neo4jResourceHas.

Neo4jResource

Predstavuje v databáze jeden typ uzlu, ktorý sa dá vyhľadávať. Jeho definícia je triviálna a vyzerá nasledovne:

```
class Neo4jResource(StructuredNode):
  uri = StringProperty(unique_index=True)
  predicates = RelationshipTo('Neo4jResource', 'HAS', model=
      Neo4jResourceHas)
```

Výpis 8.6: Objekt Neo4jResource.

Do tohto typu uzlu sa ukladá informácia o zdroji, ktorý sa dá vyhľadať a je určený unikátnym odkazom. Tento uzol má dynamický vzťah na ďalšie uzly a tento vzťah má názov, ktorý je daný taktiež unikátnym odkazom. Teda uzol môže mať neobmedzené množstvo rozdielnych vzťahov, ktoré ukazujú na ďalšie uzly.

Neo4jResourceHas

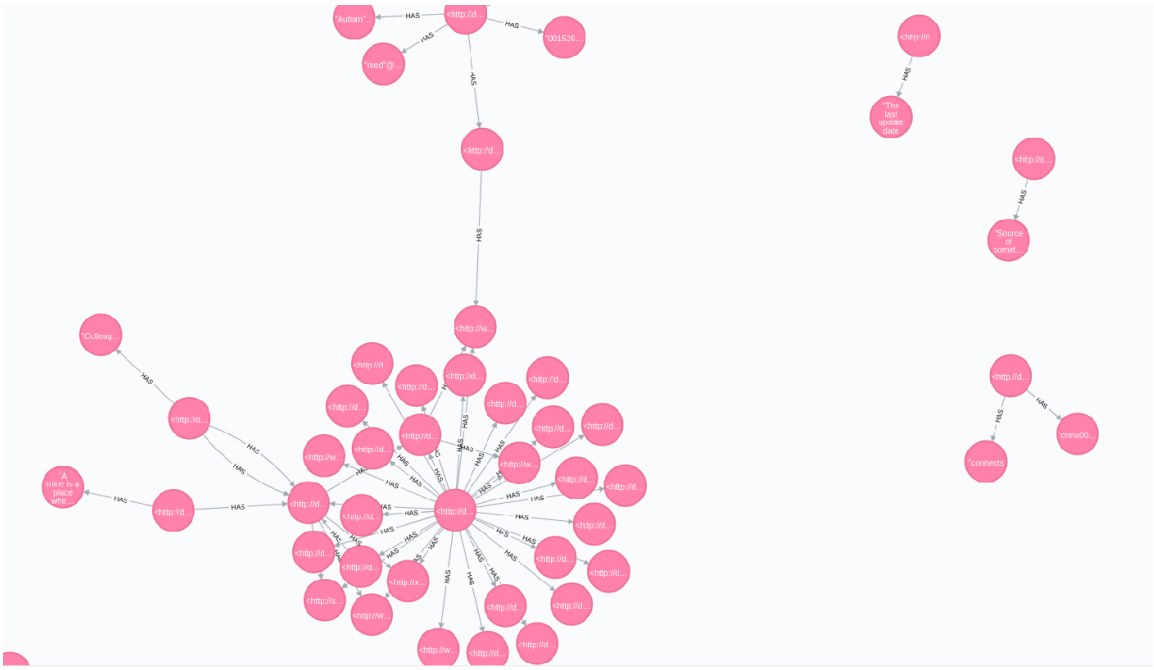
Tento objekt je vlastne popis vzťahu medzi dvoma uzlami typu *Neo4jResource*.

```
class Neo4jResourceHas(StructuredRel):
  uri = StringProperty()
```

Výpis 8.7: Objekt Neo4jResourceHas.

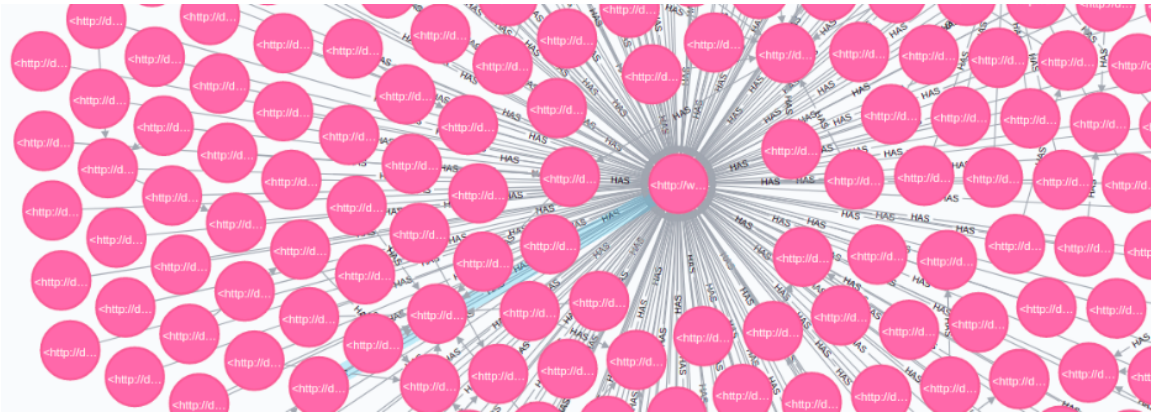
Vkladanie dát

Vkladanie dát do tohto typu databázy je zložitá činnosť, pretože databáza neumožňuje nízkoúrovňové veci. Vkladanie dát prebiehalo tak, že bolo nutné si vyhľadať uzol, ktorému sme chceli priradiť ďalšiu väzbu, teda prvý prístup do databázy. Pri vkladaní vzťahu, bolo nutné abý koncový uzol, ktorý chceme prepojiť bol už vytvorený, teda bolo nutné ho vytvoriť, alebo získať z databázy. Čo vlastne viedlo k mnohým prístupom do databázy a neúnosne zaťažovalo databázu. Čas, ktorý bol nutný na vloženie určitého množstva dát bol niekoľkonásobne väčší ako pri relačnej databáze.



Obr. 8.1: Vizualizácia jednoduchého výstupu grafovej databázy.

Pri veľkosti dát aké sú vidieť na obr 8.1 nie je problém, pretože to je malé množstvo, a vkladanie prebehne rýchlo. Avšak pri veľkom množstve dát čas vkladania stúpa pretože databázový systém si interne spracúvava údaje, vytvára indexy a podobne. Pri veľkom množstve dát to vyzerá následovne 8.2. Týmto chcem poukázať na veľkú prepojenosť dát, s čím sa spája veľa logiky a výpočtového výkonu. To sa prejaví na dobe potrebnej pre vloženie dát do takéhoto databázového systému.



Obr. 8.2: Vizualizácia zložitého výstupu grafovej databázy.

8.2 Vyhľadavací systém

Vyhľadávací webový server je najpodstatnejšou časťou celého systému. Jeho úlohou bude spracovať vstupné dáta v podobe vyhľadávaného kľúčového slova a štruktúry aspektov, ktorá predstavuje spôsob vytvorenia požadovaného požiadavku do príslušnej databázy. Výstupom systému bude výsledok vyhľadávania v podobe poľa s vyhľadanými výsledkami a ďalšími dodatkovými informáciami, ktoré majú informačný charakter.

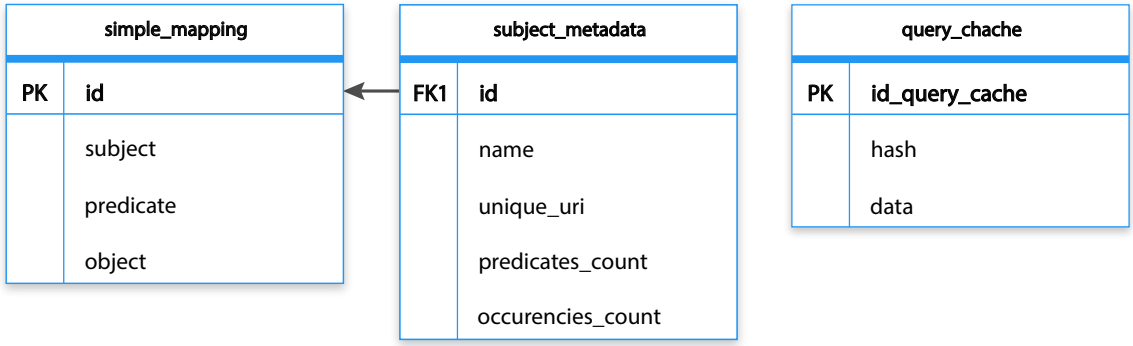
Použitým jazykom pre túto časť je *PHP*. Jazyk *Python* s ORM frameworkom *SQLAlchemy* ako bolo uvažované v návrhu nebol na prácu s veľkými dátami vhodný, pretože spôsoboval niekoľkonásobné spomalenie.

Server je teda napísaný v *PHP* s využitím frameworku *Nette* a knižnice *Dibi*. Na server sa bude možné pripojiť pomocou poskytovaných aplikačných rozhraní. Všetky implementované aplikačné rozhrania budú vyžadovať vstupné dáta v podobe JSON objektu, ktorý bude mať enkódované požadované informácie. Výstupom budú taktiež dáta vo formáte JSON. Každa odpoveď zo servru bude patrične cacheovaná tak, aby bolo možné so systémom pracovať svižne. Serverom podporované aplikačné rozhrania:

- Vyhľadávanie na adrese /search.
- Návrhové API na adrese /suggest/subjects.

8.2.1 Metodika vyhľadávania v relačnej databáze

Pri navrhovaní vyhľadávania sme vychádzali zo spôsobu poskytovania dát uložených v súboroch poskytovaných DBpédiou a ich následným uložením do relačnej databázy. Spôsob uloženia dát v databáze je možné vidieť na obrázku 8.3, ktorý je vlastne obrazom definovaných tried v podsekcií 8.1.1.



Obr. 8.3: ER diagram entít v databázovom systéme.

Zo spomínanej schémy databázových entít vyhľadávanie prebieha nasledovne. Najprv sa podľa kľúčového slova vyhľadáva v tabuľke *subject_metadata*, kde sa nájdu výsledky podľa mena aplikovaním porovnávania *LIKE* na stĺpec *name* so zostupným zoraďovaním podľa stĺpca *occurencies_count*, ktorý označuje koľkokrát sa daná entita nachádza v dátovej sade. Avšak SQL požiadavok sa pripraví, ale nevykonáva. Tento pripravený SQL požiadavok sa použije ako podpožiadavok konečného komplexného požiadavku.

Následne sa odoslané aspekty pretransformujú do podoby, kde jeden predikát má viac objektov. Z týchto kombinácií sa vytvoria ďalšie podpožiadavky pre každý predikát. Teda pre každý predikát sa vytvorí SQL požiadavok z tabuľky *simple_mapping*, kde sa však už nepýtame na názov resp. meno subjektu, ale hľadáme takú entitu, ktorá bude mať ako predikát náš hľadaný predikát a ako jeho hodnoty môžu byť nami vyhľadávané objekty. Teda podmienkou je, aby daná entita mala predikát zhodný s našim a zároveň tento predikát môže obsahovať hodnoty objektu, ktoré môžu byť zhodné s hodnotami našich objektov (OR). Nakoniec sa tieto podpožiadavky použijú pri vyhľadaní entít a ich predikátov a objektov, ktoré budú zoradené zostupne podľa počtu výskytov entít v dátovej sade. SQL požiadavok bude teda tvorený dynamicky a bude mať viac podmienkových častí. V prvej časti sa vyhľadávajú entity, ktoré spĺňajú zadané kľúčové slovo. Ďalšie podmienky sú tvorené iteratívne podľa vstupných aspektov. V každej iterácii sa pripraví požiadavok na vyhľadanie entít, ktoré spĺňajú podmienku, že majú daný predikát a nejakú hodnotu objektu zo zadaných objektov predikátu ako vlastnú hodnotu objektu [2].

Výsledný SQL požiadavok

Výsledný požiadavok je teda tvorený všetkými spomínanými časťami. Tieto časti sa nakoniec skombinujú do výsledneho požiadavku, ktorý využíva operátor *IN*, ktorému sa poskytnú výsledky zo všetkých podpožiadavkov.

```
SELECT ‘subject‘
FROM ‘simple_mapping‘
WHERE ‘subject‘ IN (
       SELECT ‘unique_uri‘ FROM ‘subject_metadata‘ WHERE ‘name‘ LIKE ’%
           Alabama%’ ORDER BY ‘occurencies_count‘ DESC
       )
AND ‘subject‘ IN (
```

```
        SELECT ‘subject‘ FROM ‘simple_mapping‘ WHERE ‘predicate‘ = ’<http://
            xmlns.com/foaf/0.1/name>’ AND (‘object‘ = ’"State of Alabama"@en
            ’)
        )
GROUP BY ‘subject‘
LIMIT 100
OFFSET 0
```

Výpis 8.8: Príklad požiadavku na vyhľadávanie.

Požiadavok 8.8 vyhľadá entity, ktorých meno môže obsahovať podreťazec *Alabama* a zároveň vyhľadávané entity musia spĺňať podmienky, že jeden z ich predikátov sa musí nazývať *<http://xmlns.com/foaf/0.1/name>* a jeho hodnota musí byť *"State of Alabama"@en*. Takýmto spôsobom je možné vyhľadávať entity na základe filtrovania ich predikátov a objektov, ktoré môžu predstavovať ľubovoľnú vec od vlastnosti objektu, vzťah na inú entitu alebo hierarchické zaradenie.

8.2.2 Prístupové body relačnej databázy

Aplikačné rozhranie pre MySQL databázu pozostáva iba z niekoľkých prístupových bodov, ktoré obsahujú funkcionalitu pre vyhľadávanie aj pre automatické dopĺňanie. Jeho chovanie bude popísané nasledovne. Všetky endpointy implementované pre túto databázu spracúvavajú iba *JSON* formát vstupu, ako to bolo spomenuté v kapitole 7 pri návrhu. Iné formáty vstupných dát nie sú podporované.

8.2.3 Vyhľadávací endpoint

Vyhľadávať je možné na adrese */search* pomocou metódy *POST*. Ako vstupné parametre v tele požiadavku je možné poslať objekt, ktorý následne popíšeme. Parametre objektu budú tiež popísane pretože je možné vybrať si z niekoľkých voliteľných parametrov.

Vstupné dáta vyhľadávania

```
{
  "query": "Alabama",
  "paginator": {
    "page": 1,
    "items_per_page": 10
  },
  "aspects": [
    {
      "predicate":
        {
          "uri": "<http://dbpedia.org/ontology/type>",
          "op": "EXACT_MATCH"
        },
      "object": "<http://dbpedia.org/resource/City>"
    },
    .
    .
```

```
    .
  ]
}
```

Výpis 8.9: Vstupné dáta.

Príklad objektu ukazuje všetky možné parametre ktoré je možné zaslať, políčko s názvom *query* je určené pre kľúčové slovo podľa ktorého sa bude vyhľadávať. Využíva sa na porovnávanie názvu subjektu uloženého v databáze.

Políčko *paginator* popisuje objekt v ktorom sa dá nastaviť to, akú ktorú stránku si uživateľ chce zobraziť, poprípade políčko *items_per_page*, ktoré určuje počet výsledkov na stránku ako je zvyklosťou pri systémoch so stránkovaním. Políčko *paginator* je voliteľné. Ak nie je poskytnuté, tak endpoint vracia všetky výsledky spĺňajúce vyhľadávanie. Avšak pri neposkytnutí paginátora je nutné počítať s dlhším spracovaním vyhľadávania ako aj spracovaním výstupu.

Objekt *aspects* je pole objektov podľa ktorých sa vyhľadávanie zohľadňuje. Políčko *predicate* predstavuje predikát a políčko *object* objekt vyhľadávania. Táto dvojica môže vyzerať napríklad aj takto *<http://xmlns.com/foaf/0.1/name>* pre predikát a *"Gautama Buddha"@en* pre objekt. Políčko *op* predstavuje operátor, ako sa má predikát porovnávať.

Výstupné dáta vyhľadávania

Výstupný objekt má presne definovanú štruktúru, ktorá bola navrhnutá v kapitole 7. Táto štruktúra je nemenná a vyzerá nasledovne:

```
{
  "paginator": {
    "page": 1,
    "items_per_page": 10,
    "total_pages": 1
  },
  "data": [
    {
      "id": 27600,
      "name": "<http://dbpedia.org/resource/Gautama_Buddha>",
      "type": "RESOURCE",
      "predicates": [
        {
          "predicate": {
            "id": 19773,
            "name": "<http://xmlns.com/foaf/0.1/name>",
            "type": "PREDICATE"
          },
          "object": {
            "id": 27601,
            "name": "\"Gautama Buddha\"@en",
            "type": "VALUE"
          }
        },
        {
          "predicate": {
```

```
          "id": 19773,
          "name": "<http://dbpedia.org/ontology/birthYear>",
          "type": "ONTOLOGY"
        },
        "object": {
          "id": 27601,
          "name": "\"-0563\"^^<http://www.w3.org/2001/XMLSchema#gYear>",
          "type": "VALUE"
        }
      }
    ]
   }
  ]
}
```


Výpis 8.10: Výstupné dáta.

Paginátor objekt je nastavený len v prípade, ak bol na vyhľadávanie použitý paginátor, podstatná časť z tohto objektu je vlastne počet stránok, ktorý sa zobrazuje v tabuľke na klientskej aplikácií. Políčko *data*, je pole výsledkov, ktoré boli vyhľadané. Jednotlivé výsledky potom obsahujú políčka popisujúce samotný objekt. Nakoniec políčko *predicates* predstavuje pole aspektov danej entity. Jedna položka tohto poľa sa skladá z predikátu a objektu.

8.2.4 Endpoint pre automatické dopĺňanie

Grafické rozhranie, ktoré si popíšeme v podkapitole 8.3, implementuje niektoré z vyhľadávacích vzorov uvedených v kapitole 7. Niektoré z týchto vzorov využíva na svoju funkcionalitu iný druh dát ako sa používa pri vyhľadávaní a je nutné implementovať prístupový bod, ktorý bude spĺňať požiadavky pre daný vyhľadávací vzor. Vzorom o ktorom sa bavíme je automatické dopĺňanie do vstupného poľa. Pre toto dopĺňanie sme určili presnú štruktúru dát. Táto štruktúra bola popísaná v sekcii 7.7.2.

Vstupné dáta automatického dopĺňania

Ako bolo spomenuté v kapitole 7, pre dopĺňanie je nutné zaslať iba kľúčové slovo na server, kde sa slovo spracuje a následne sa vrátia dáta, ktoré by mohli odpovedať niektorému z možných vyhľadávaných slov.

```
{
  "query": "Aachen"
}
```

Výpis 8.11: Štruktúra vstupných dát automatického dopĺňania.

Výstupné dáta automatického dopĺňania

Výstupné dáta sa generujú na základe vstupného kľúčového slova. Ako výstup pre tento prístupový bod sa použije prvých desať výsledkov, kde sa bude vyhľadávať pomocou operátora *LIKE* na stĺpec *name* v tabuľke *subject_metadata*. Zoraďovanie pri tomto požiadavku

bude na základe počtu výskytov v dátovej sade. Jedná sa teda o zostupné zoraďovanie podľa stĺpca *occurencies_count*.

```
{
  "values": {
    "<http://dbpedia.org/resource/Aachen>": {
      "name": "Aachen",
      "short_abstract": "Aachen or Bad Aachen, traditionally known in
         France and the United Kingdom as Aix-la-Chapelle, is a spa and
         border town in North Rhine-Westphalia, Germany.",
      "predicates_count": 15,
      "occurencies_count": 66
    },
    "<http://dbpedia.org/resource/FH_Aachen>": {
      "name": "FH_Aachen",
      "short_abstract": null,
      "predicates_count": 17,
      "occurencies_count": 1
    },
    .
    .
    .
  }
}
```

Výpis 8.12: Štruktúra výstupných dát automatického doplňania.

Od návrhu systému sa do výstupu pridalo políčko *short_abstract*, ktoré popisuje krátky abstrakt k danej entite, ak ho entita má. Taktiež tento prístupový bod je cacheovaný a jeho reakcia je rádovo v desiatkach milisekúnd.

8.3 Vlastné grafické užívateľské rozhranie

Poslednou časťou systému je webový klient, ktorý rieši uživateľské vyhľadávanie, zadávanie požadovaných aspektov ale aj kľúčového slova. Ako bolo spomenuté v kapitole 7 rozhranie by sa malo skladať z niekoľkých najdôležitejších stránok. Podľa vzoru najväčších vyhľadávačov rozhranie obsahuje hlavnú vstupnú stránku, stránku kde sa zobrazujú vyhľadané výsledky, detail entity a v našom prípade aj stránku, kde sa dá modifikovať aspektový filter.

Klient je napísaný v jazyku *TypeScript* za použitia frameworku *Node.js* [17], kde sa taktiež využívajú ďalšie technológie, ktoré slúžia na implementovanie reaktívnej webovej aplikácie [16]. Najdôležitejšími knižnicami sú:

- ReactJs – JavaScriptová knižnica na budovanie užívateľských rozhraní.
- Redux – predvídateľný kontajner stavov pre JavaScriptové aplikácie.
- domain-task – knižnica na riešenie medzi servrových požiadavkov.
- React-material-ui – knižnica, ktorá zapúzdruje prvky a je CSS nadstavbou nad komponentami v Reacte. Je založená na princípoch Material dizajnu.

S využitím Reactu je aplikácia rozdelená do komponent, kde každá komponenta reaguje na zmenu stavu aplikácie prekreslením a ďalšou vnútornou funkcionalitou. Reduxový kontajner udržiava stav aplikácie konzistentný a jeho zmenu môžu realizovať iba špecializované funkcie. Priama zmena stavu z grafickej komponenty teda nie je možná.

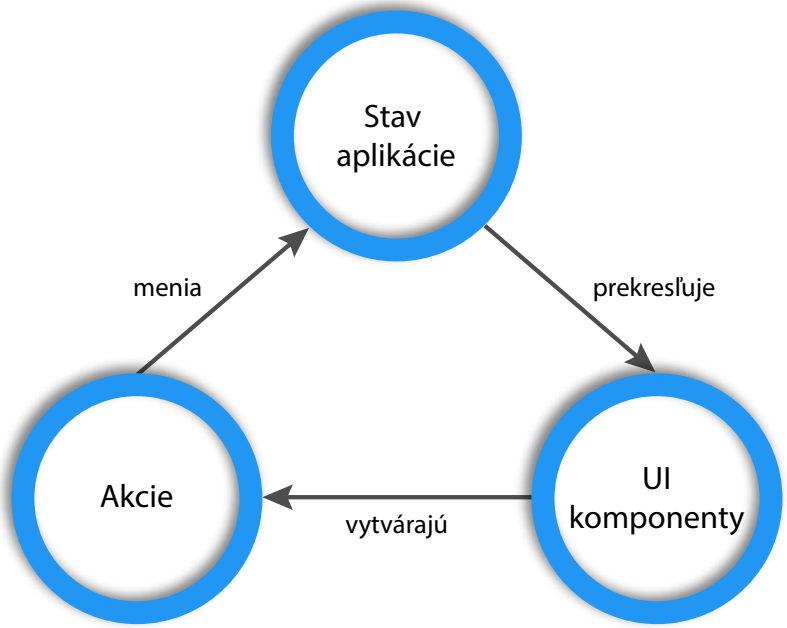
Ako bolo spomenuté samotná aplikácia je zložená z viacerých komponent, kde každá komponenta sa stará o istú časť stavu aplikácie a premieta tento stav do grafického rozhrania. Medzi hlavné komponenty patria:

- Blokovacie modálne okno.
- Komponenta obsluhujúca vstupné dáta, ktoré vstupujú do vyhľadávania.
- Komponenta vykresľujúca výstupné dáta, teda vyhľadané výsledky.
- Komponenta starajúca sa o aspektový filter.

Každá komponenta je nezávislá, jediným spájajúcim elementom je stav aplikácie. Čo obsahuje a načo sa používajú jednotlivé časti stavu aplikácie si popíšeme nasledovne.

8.3.1 Aplikačný stav

Stav aplikácie obsahuje niekoľko objektov nutných na správne vykresľovanie. Kedže React funguje na princípe návrhového vzoru zvaného *Observer*, kde každá komponenta pozoruje zmeny na konkrétnej časti stavu aplikácie. Ak nejaké zmeny nastanú komponenta sa prekreslí s novými dátami. Ako funguje tento princíp je možné vidieť na obrázku 8.4.



Obr. 8.4: Princíp fungovania React aplikácie.

Tieto skutočnosti umožňujú vytvoriť reaktívnu aplikáciu, ktorá reaguje na zmeny v reálnom čase. Samotný stav aplikácie sa delí na dva objekty, ktoré sú popísané nasledovne na výpise 8.13.

```
export interface AppState {
  globalLock: boolean
  search: SearchState
}
```

Výpis 8.13: Aplikačný stav a jeho súčasti.

Hlavné položky v tomto stave sú:

- Prepínač zobrazovania modálneho okna,
- Objekt, ktorý uchováva stav samotného vyhľadávania,

Prepínač *globalLock* slúži na prepínanie stavu, kedy aplikácia môže interagovať s užívateľom. Teda prakticky slúži na blokovanie rozhrania, kým sa nevykoná operácia na pozadí. Na základe tohto prepínača sa zobrazuje, či nezobrazuje blokujúce modálne okno na obrazovke. Modálne okno sa zobrazuje v strede obrazovky a neumožňuje užívateľovi ďalej pracovať s rozhraním. Spomínané modálne okno je možné vidieť na obrázku 8.6.

Vyhľadavací stav

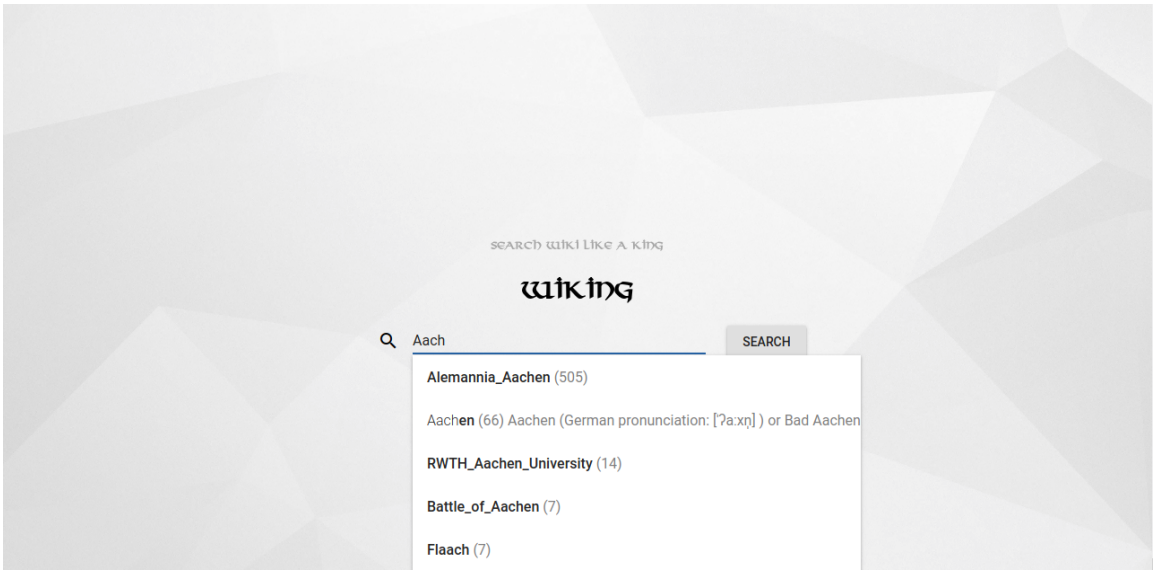
Tento stav slúži na uchovanie informácií o vykonávanom vyhľadávaní. V tomto stave je možné nájsť informácie o vyhľadaných výsledkoch, použitej paginácií a vyhľadávané kľúčové slovo. Avšak stav obsahuje aj aspektový filter, ktorý si udržiava informácie o použitých aspektoch a taktiež informáciu o tom, ako majú byť použité pri vyhľadávaní. Túto časť celkového stavu využívajú komponenty, ktoré vykresľujú vyhľadané výsledky. Konkrétne ide o komponentu vykresľujúcu detail entity a komponentu starajúcu sa o aspektový filter.

```
export interface SearchState{
  actualResults: SearchResult[] | null,
  paginator: Paginator | null,
  query: string | null,
  facetFilter: PredicateTuple[]
}
```

Výpis 8.14: Stav vyhľadávania.

8.3.2 Vstupná stránka

Po vzore veľkých vyhľadaváčov sme sa rozhodli implementovať vstupnú stránku vyhľadávacieho systému. Ako bolo spomenuté pri návrhu v kapitole 7, dobrá vstupná stránka by mala zaujať, zanechať dobrý prvý dojem, tak aby užívateľ vyskúšal systém. Preto úvodná stránka musí obsahovať funkcionalitu automatického dopĺňania ako bolo spomenuté v podkapitole 7.1. Nami implementovanú stránku je možno vidieť na obrázku 8.5.



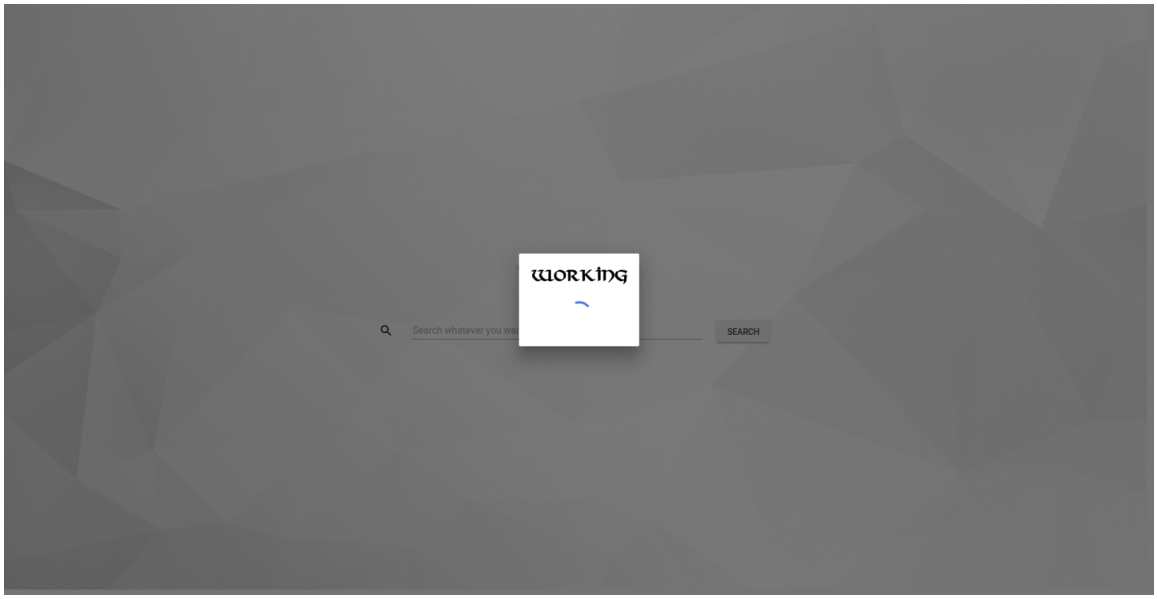
Obr. 8.5: Úvodná stránka.

Ako je možné vidieť na stránke sa nachádza vstupné pole do ktorého sa zadáva kľúčové slovo. Na tomto vyhľadávacom poli je implementované automatické dopĺňanie na základe vstupného slova. V zátvorke je možno vidieť číslo, podľa ktorého sa zoraďujú entity. Číslo predstavuje počet výskytov entity v dátovej sade. Takže na začiatok sa dostanú entity, ktoré boli v dátach spomenuté najčastejšie. Ak je vo vyhľadávanej entite zahrnutý abstrakt pri napovedaní sa zobrazuje jeho prvá veta. Táto krátka ukážka by mala bližšie napovedať o tom, čo daná entita predstavuje.

Ak si užívateľ vybral z napovedaného výberu hodnota výberu je použitá na vyhľadanie. Použitý výber sa nastaví do vyhľadávacieho stavu (výpis 8.14) ako kľúčové slovo. Následne sa vyhľadávací stav použije na vyhľadanie výsledkov a presmeruje na stránku s výsledkami. Stránka s výsledkami je popísaná v podsekcii 8.3.4.

8.3.3 Blokovacie okno

Blokovacie okno je dobrou praktikou ak chceme užívateľa informovať, že systém pracuje a na pozadí prebieha nejaká operácia. Nie je teda možné s ním v danej chvíli pracovať. Práve na túto činnosť je modálne okno navrhnuté. Na stránke sa zobrazuje v prípade, že sa klientská aplikácia pýta aplikačného rozhrania na výsledky.

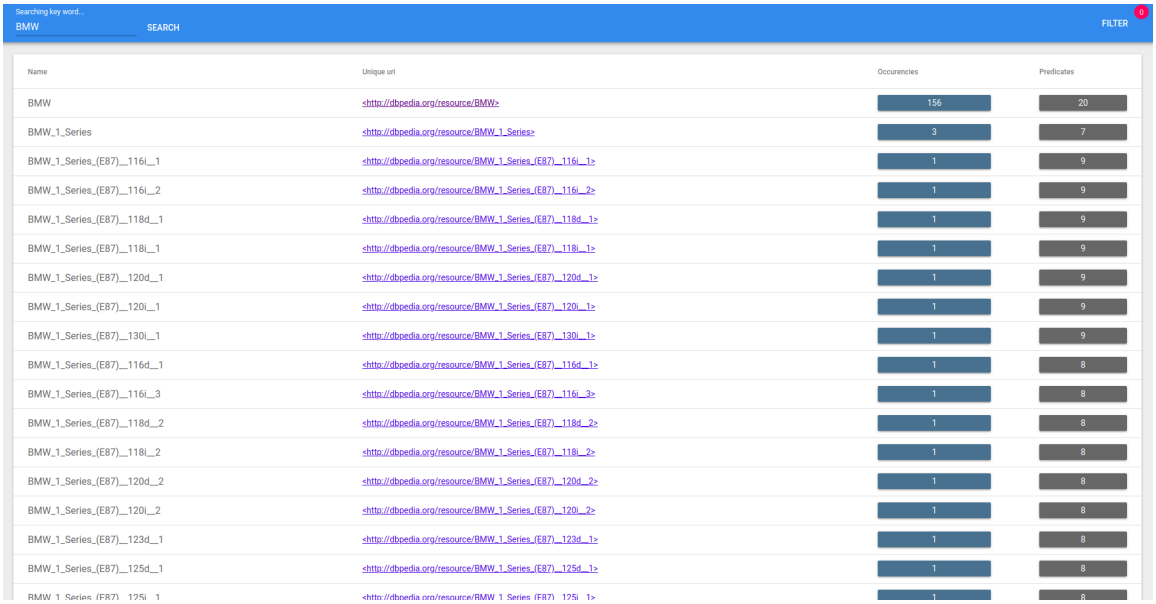


Obr. 8.6: Blokovacie okno aplikácie na úvodnej stránke.

Na modálnom okne sa nachádza text s výpovednou hodnotou, čo systém v pozadí robí a taktiež aj loader na spríjemnenie užívateľského zážitku.

8.3.4 Vyhľadané výsledky

Táto stránka zastrešuje skoro celú funkcionalitu systému. Na tejto stránke je možné meniť kľúčové slovo či aspektový filter alebo prejsť na detail entity. Dostať sa na vyhľadané výsledky je možné z úvodnej stránky po zadaní kľúčového slova a výberu z napovedača. Stránka zobrazuje základné informácie o vyhľadaných entitách v jednoduchej tabuľke. Okrem konkrétnych záznamov entít sú v záhlaví stránky informácie o kľúčovom slove a informácia o veľkosti aspektového filtra. Ako stránka vyzerá je možné vidieť na obrázku 8.7.



Obr. 8.7: Vyhľadané výsledky v tabuľke.

Záznam v tabuľke obsahuje niekoľko základných informácií podľa, ktorých užívateľ vie prečo tento výsledok bol zaradený do vyhľadávania. Prvý stĺpec tabuľky popisuje vyextrahovaný názov entity z odkazu entity. Ďalší stĺpec obsahuje samotný unikátny odkaz na entitu v systéme. Taktiež tento odkaz slúži ako odkaz na stránku v DBpédii. Po kliknutí na unikátny odkaz sa dostaneme na samotné stránky DBpédie a to pre prípad, že by náš systém neobsahoval všetky informácie. Posledné dva stĺpce obsahujú agregované hodnoty, ktoré majú informačný charakter. Prvou hodnotou je počet výskytov entity v dátach a druhý koľko daná entita obsahuje vlastností. Tieto informácie môžu slúžiť užívateľovi pri rozhodovaní či sa oplatí v danej entite vyhľadávať alebo či daná entita je relevantná pri danom vyhľadávaní.

Okrem spomínanej tabuľky v hornej časti je taktiež možné nájsť tlačidlo, ktorým sa dostaneme do stránky aspektového filtra. Pri tlačidle sa zobrazuje v krúžku číslo, ktoré predstavuje počet položiek vo filtri. V ľavej časti záhlavia je možné nájsť kľúčové slovo, ktoré je pri vyhľadaní nepovinné ak aspektový filter obsahuje nejaké položky.

Stránku sme sa snažili nezahlcovať nadbytočnými informáciami, ako tomu bolo pri existujúcich riešeniách v kapitole 6. Stránka neobsahuje informácie ako aktuálne nastavenie filtra, pretože filter nikdy nebude nastavený na veľké množstvo položiek. Nemalo by dôjsť k tomu, aby si užívateľ nepamätal čo vyhľadáva. Tabuľka taktiež neobsahuje rozsiahle texty v ktorých by sa užívatelia mohli strácať. Týmto všetkým stránka prispieva k jednoduchosti.

Využitie stavu aplikácie

To čo stránka vykresľuje zo stavu vyhľadávania je políčko *actualResults*. Typ políčka je rozhranie, ktoré obsahuje informácie o jednotlivých entitách, ktoré je popísané na výpise 8.15.

```
export interface SearchResult {
  name: string;
  id: number;
  type: string;
```

```
  occurencies_count: number;
  unique_uri: string;
  predicates: { [id: string]: string[] } | {};
}
```

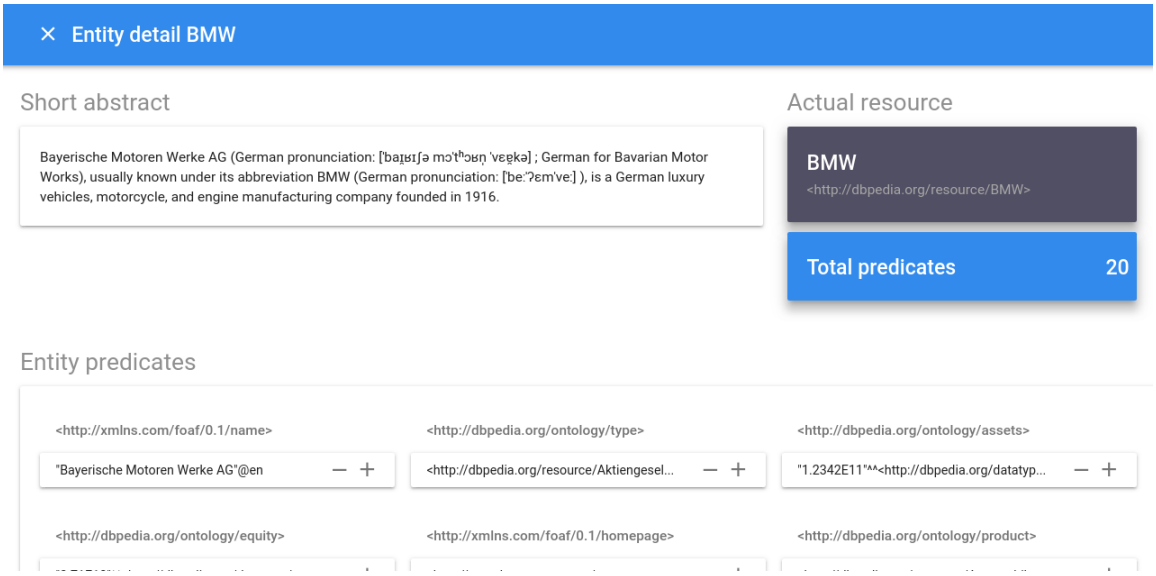
Výpis 8.15: Vyhľadaný výsledok.

Toto rozhranie obsahuje všetky informácie, ktoré sú nutné na zobrazovanie v tabuľke. Väčšina údajov už bola popísaná v predchádzajucej sekcii, okrem políčka *predicates*, ktoré obsahuje informácie o vlastnostiach entity, čo sa používa až pri samotnom detaile entity.

8.3.5 Detail entity

Detail entity je stránka, ktorá obsahuje a zobrazuje všetky dostupné informácie o danej entite. Základom je zobrazenie všetkých predikátov danej entity. Tieto predikáty majú informačnú hodnotu. Zároveň slúžia na pridávanie podmienok do aspektového filtra. Aktuálne sú podporované dve možnosti chovania podmienok v aspektovom filtri. Predikát sa musí vyskytovať vo výsledku a naopak nesmie sa vyskytovať vo výsledkoch.

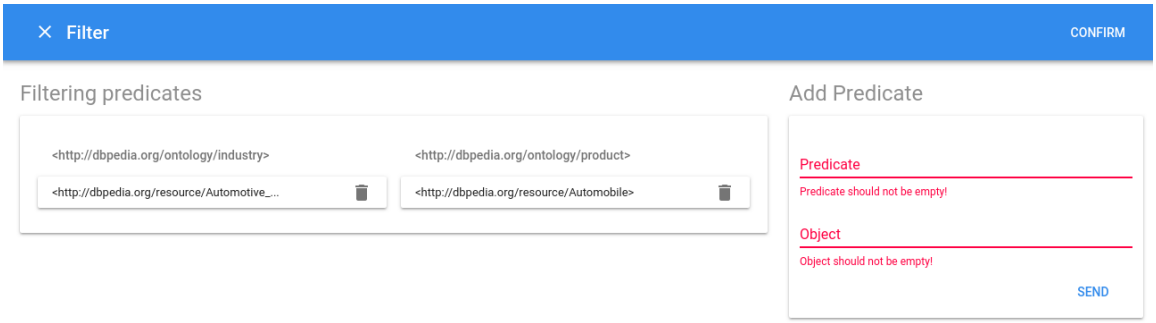
Okrem samotných skutočných prepojených hodnôt stránka obsahuje aj dodatkové informácie. Ak entita obsahuje abstrakt, tak je zobrazený pod záhlavím stránky. Pod záhlavím napravo je možné vidieť počet predikátov, ktoré obsahuje daná entita. Taktiež tam môžeme nájsť extrahovaný názov ale aj unikátny odkaz entity.



Obr. 8.8: Detail entity.

8.3.6 Stránka pre aspektový filter

Táto stránka je podobná tomu čo sme mohli vidieť v predošlej sekcii na obrázku 8.8. Na pravej strane je možné vidieť formulár pomocou ktorého je možné pridávať nové záznamy do aspektového filtra. Vľavo a v strede je možné vidieť nastavenie filtra, teda jeho hodnoty a použité operácie. Z filtra je možné jednoducho odstraňovať a pridávať jednotlivé podmienky, ktoré nie sú použíté pokiaľ zmeny vo filtri nepotvrdíme tlačidlom. Po tom čo sa zmeny potvrdia prebieha znova vyhľadávanie a po načítaní nových hodnôt prebehne prekreslenie tabuľky, kde sa vyhľadané výsledky zobrazujú.



Obr. 8.9: Aspektový filter.

Spomínaný formulár, ktorý sa nachádza na pravej strane umožňuje zadávať záznamy do filtra úplne bez žiadných kontrol tak, aby užívateľ nemal zviazané ruky pri zadávaní jednotlivých záznamov. Jedinou podmienkou je, aby tieto políčka neboli prázdne. Ako by mohol byť tento formulár rozšírený sa dozvieme v kapitole 10.

Kapitola 9

Testovanie

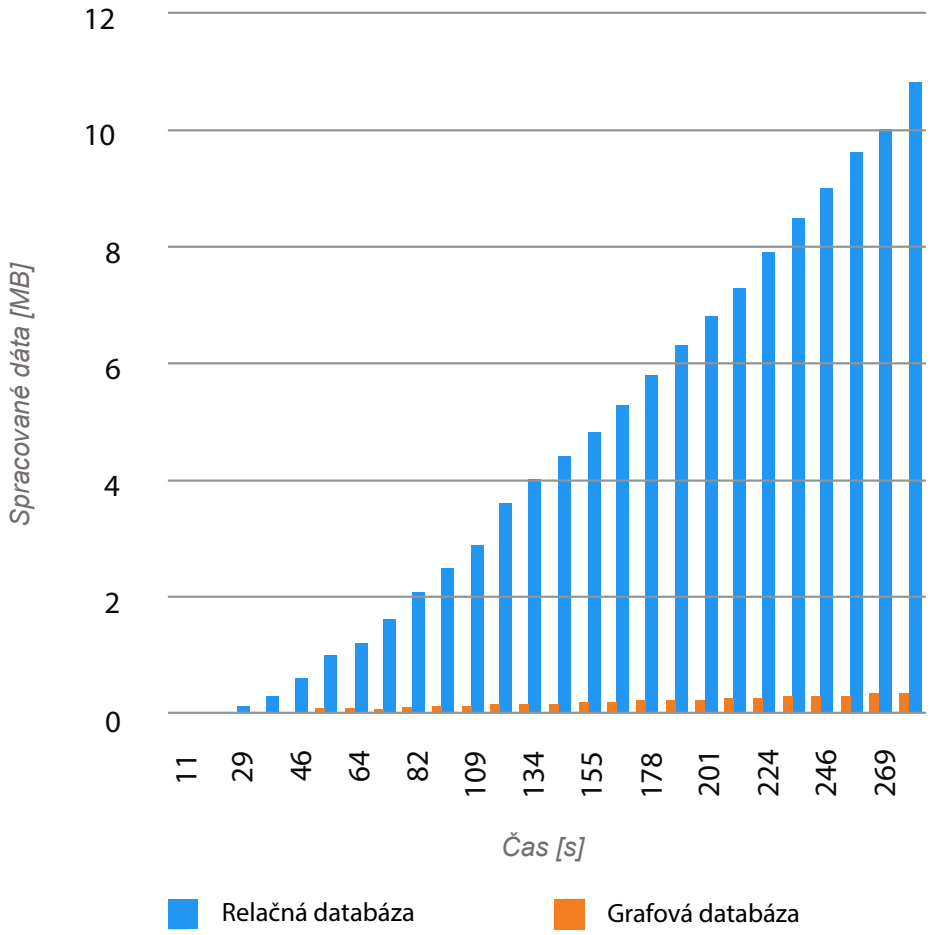
V tejto kapitole sa budeme venovať testovaniu implementovaného systému, jeho parametrom a implementovaným funkciám. Kedže sa systém skladá z viacerých častí, jednotlivé časti je možné testovať oddelene.

9.1 Špecifiká testovacieho stroja

Všetky testy ako aj vývoj celého projektu prebiehal na notebooku s operačným systémom Ubuntu 17.10. Procesor notebooku nesie označenie Intel® Core™ i7-2670QM, ktorý je taktovaný na 2,2 Ghz. A dostupná pamäť notebooku predstavuje 8 GB. Celý chod notebooku zabezpečuje SSD disk o veľkosti 256GB s čítacou a zapisovacou rýchlosťou 550MB/s.

9.2 Extrakcia dát

Pri testovaní extrakcie sme sa zamerali na rýchlosť spracovania súboru a ukladania dát do príslušnej databázy. Na príslušnom grafe 9.1 môžeme vidieť závislosť veľkosti spracovaného množstva dát za časovú jednotku. Tohto testovania sa zúčastnili dva databázové systémy. Prvým je štrandardný *MySQL* server a jednotka v grafových databázach – databázový systém *neo4j*.



Obr. 9.1: Porovnanie výkonu relačného a grafového databázového systému pri vkladaní dát.

Ako je vidieť na grafe uvedenom vyššie relačná databáza má navrch pri vkladaní dát do databázového systému. Skript bol optimalizovaný na úrovni vkladania samotných dát, tak ako to bolo možné z hľadiska možností databázového systému. Distribúcia vkladania medzi viacero vlákien nebola riešená z dôvodu komplikácií pri vkladaní dát do grafovej databázy. Toto komplikované vkladanie dát do grafovej databázy je dané tým, že pokiaľ chceme definovať vzťah medzi dvomi uzlami, tieto uzly musia byť už v databáze definované. Túto skutočnosť sme v relačnej databáze obišli vkladaním záznamov s unikátnymi hodnotami, vďaka unikátnym URL samotných záznamov z DBpédie. Táto skutočnosť sa odráža v grafe a to nízkym množstvom spracovaných dát grafovou databázou. Relačná databáza je teda oproti grafovej v značnej výhode. Množstvo spracovaných dát presahuje grafovú databázu približne tridsaťkrát. Vďaka tomu grafová databáza nebola v systéme reálne implementovaná.

9.3 Rýchlosť dotazovania a verifikácia dát vyhľadávacieho systému

Pri testovaní vyhľadávacieho systému sme sa zamerali hlavne na jeho rýchlosť vygenerovať odpoveď. Testovali sme odozvu systému na rôzne požiadavky od jednoduchých až po zložitejšie.

Princípom testu bolo vyhľadať určitú množinu entít tak, že sme vygenerovali požiadavok do API vyhľadávacieho systému. Toto testovanie prebiehalo v niekoľkých scenároch a testovala sa hlavne rýchlosť odozvy na naš požiadavok. Testovanie prebiehalo pomocou programu *Postman*, ktorý slúži na odosielanie požiadavkov na server.

9.3.1 Testovanie vyhľadávaním

V scenári pre vyhľadávanie sme sa snažili nadefinovať množinu entít, ktorá by popisovala všetky automobilky. Začali sme tým, že sme vyhľadali entity, ktoré sú spojené s automobilovým priemyslom. Vytvorili sme v programe *Postman*[1] požiadavok na server s uvedenými parametrami:

Parameter	Hodnota
Vyhľadávacie slovo	prázdne – chceme všetky entity na základe aspektov
1. Aspekt typ	<http://dbpedia.org/ontology/industry>
1. Aspekt hodnota	<http://dbpedia.org/resource/Automotive_industry>

Tabuľka 9.1: Parametre prvého požiadavku.

Požiadavok bol odoslaný na server niekoľkokrát. Časy jeho odozvy boli zaznamenané v tabuľke 9.2. Z tabuľky jasne vidieť, kedy je odpoveď serveru uložená v cache pamäti. Okrem prvého požiadavku, kedy odpoveď nie je uložená v cache pamäti sú časy odpovedí do 200ms, čo môžeme považovať za priateľné.

Odosielanie	Čas[ms]
1	10187
2	152
3	149
4	125
5	164

Tabuľka 9.2: Rýchlosť odpovede prvého požiadavku.

Odpoveď obsahovala štuktúru popísanú v kapitole 8. Vo výsledku sme dostali tieto entity:

Entita
<http://dbpedia.org/resource/BAW>
<http://dbpedia.org/resource/BMW>
<http://dbpedia.org/resource/Fiat>
<http://dbpedia.org/resource/Acura>
<http://dbpedia.org/resource/Aixam>
<http://dbpedia.org/resource/Abarth>
<http://dbpedia.org/resource/Brembo>
<http://dbpedia.org/resource/FAW_Group>
⋮

Tabuľka 9.3: Čiastočná odpoveď prvého požiadavku.

[1] Dostupné z https://www.getpostman.com/

V tabuľke 9.3 je možno vidieť čiastočný výpis entít, ktoré systém vyhľadal. Čiastočný preto, lebo entít ktoré spĺňali naše podmienky bolo veľa. Avšak všetky entity po krátkej kontrole obsahovali náš vyhľadavací aspekt, takže o žiadny výsledok nemáme naviac ani menej. Ako je vidieť v tabuľke 9.3 v entitách je možné nájsť spoločnosti, ktoré nie sú automobilky a preto je nutné naše vyhľadávanie spresniť ďalším aspektom. Podmienka, ktorou sme definovali našu skupinu musí byť doplnená o ďalšiu.

Parameter	Hodnota
Vyhľadávacie slovo	prázdne – chceme všetky entity na základe aspektov
1. Aspekt typ	<http://dbpedia.org/ontology/industry>
1. Aspekt hodnota	<http://dbpedia.org/resource/Automotive_industry>
2. Aspekt typ	<http://dbpedia.org/ontology/product>
2. Aspekt hodnota	<http://dbpedia.org/resource/Automobile>

Tabuľka 9.4: Parametre druhého požiadavku.

Rýchlosť odpovede rozšíreného požiadavku možno vidieť v tabuľke 9.5. Ako tomu bolo v predošlom požiadavku je možné vidieť skok v rýchlosti odpovede kvôli ukladaniu dát do cache pamäte. Taktiež sa vo výsledku nenachádzajú entity, ktoré by nespĺňali naše požiadavky.

Odosielanie	Čas[ms]
1	102687
2	147
3	146
4	126
5	239

Tabuľka 9.5: Rýchlosť odpovede druhého požiadavku.

Ako vyplýva z predošlých krokov definovanie množiny entít prechádza niekoľkými etapami. Ak by sme chceli upresniť vyhľadávanie, jednoducho by sme pridali ďalší aspekt a dali vyhľadávať. V prípade, žeby sa nám to vrátili neočakávané výsledky, tak by sme jednoducho z aspektové filtra odobrali posledne pridanú položku.

9.3.2 Záver testu

Z tabuliek v predchádzajúcej sekcii je jasné, že čím viac užívateľov bude systém používať, tým rýchlejšie bude systém odpovedať na rôznorodé požiadavky. Veľký skok pri prvom a druhom rovnakom požiadavku je daný výkonom počítača na ktorom nebežal iba webový server ale veľké množstvo služieb. Ak by sa výkon počítača zameriaval iba na tento vyhľadávací systém tak, by bolo možné prvotné časy minimalizovať aj pod jednu sekundu. Rýchlosť odozvy systému je teda pri prvom požiadavku nevyhovujúca, čo sa však optimalizovalo pomocou cachovacieho mechanizmu, kde sa čas odozvy výrazne znížil. Test ukázal ako je možné nadefinovať skupinu entít a ako si túto definíciu spresniť. Takýmto spôsobom je možné si definovať lubovoľnú množinu entít.

9.4 Testovanie užívateľského rozhrania

V kapitole 8 sme sa dozvedeli všetky informácie o implementovanom systéme. Na základe týchto informácií sme sa rozhodli vytvoriť úlohy a dotazník pre budúcich používateľov systému. Niekoľkým ľudom bol systém sprístupnený a dostali úlohy, ktoré so systémom majú vypracovať. Na konci testovania mali vypracovať dotazník, ktorý by nám pomohol zlepšiť veci pri prípadnom budúcom vývoji projektu.

Úlohy o ktoré mali respondenti vypracovať boli nasledovné:

- Vyhľadať množinu entít, ktoré predstavujú automobilové spoločnosti.
- Vyhodnotiť chovanie automatického dopĺňania.
- Vyhodnotiť spôsob zadávania nového aspektu do aspektového filtra.

Tieto úlohy by mali stačiť nato, aby preverili základnú funkcionalitu systému a taktiež to, či užívatelia sú schopný ho používať. Ďalšou skúmanou vlastnosťou bol čas strávený danou úlohou.

Hlavnou časťou bolo vyhľadanie množiny automobilových spoločností. Pri tejto úlohe si užívatelia museli vyhľadať nejakú automobilku a dostať sa na aspektové vyhľadávanie. Užívatelia teda museli prejsť aj cez vstupnú stránku a teda otestovať automatický dopĺňač.

Z vyplneného dotazníka vyplynulo, že automatický dopĺňač funguje tak, ako by mal. Užívatelia s ním nemali problém. Akurát fakt, že nie všetky entity mali abstrakt malo rozhodujúcu rolu pri výbere položky z automatického dopĺňania.

Pri úlohe s vyhľadaním spomínanej mnnožiny entít bol užívateľovi navrhnutý postup ako má toho docieliť. Do postupu bolo zahrnuté vyhľadanie nejakej automobilky a následne výber správnych aspektov filtra. Následne potvrdenie vyhľadávania a nakoniec validácia výsledkov.

Tabuľka 9.6: Úloha vyhľadávania.

Odpoveď	Počet
Menej ako minútu.	4
Menej ako dve minúty	6
Viac ako dve minúty	1

Z dotazníka vyplynulo, že respondenti nemajú problém orientovať sa v systéme. Avšak dostali sme pripomienky na priamu úpravu aspektového filtra, kde by sa hodilo prísnejšie kontrolovať vstupy. Pretože pri vykonávaní úlohy respondenti nasledovali poskytnuté kroky. Ak by sa však rozhodli upravovať filter priamo, vyhľadávanie by sa im výrazne znepríjemnilo. To vlastne súvisí s poslednou úlohou, ktorou bolo vyhodnotiť pridávanie aspektu do aspektového filtru. O možných zlepšeniach sa dozvieme v kapitole 10.

Z testovania vyplynulo, že užívatelia by nemali problém systém využívať. Medzi hlavné výhody systému patrí jeho jednoduchosť. Užívatelia však navrhli určité vylepšenia systému, ktoré môžu byť súčasťou budúceho vývoja. V konečnom dôsledku bol systém ohodnotený ako fungujúci s menšími nedostatkami.

9.5 Porovnanie spôsobov dotazovania nášho systému oproti kokurenčným nástrojom

Náš vyhľadávací systém využíva relačnú databázu a požiadavok do databázy je generovaný na základe vstupov, ktoré sa dostanú na jeho aplikačné rozhranie. Na samotné zrýchlenie je využitý istý druh cacheovania odpovede. Podobné systémy tohto typu využívajú zväčša databázový systém, ktorý používa dotazovací jazyk *SPARQL*. Niektoré systémy môžu používať grafovú databázu, na ktorú sa dajú aplikovať grafové algoritmy hľadania cesty v grafe. Konkurenčné systémy a ich databázové systémy potrebujú viac výpočtového výkonu na ukladanie dát. Vyhľadávanie je však iné a pri vyhľadávaní majú určitú výhodu a to tú, že pokiaľ by sme si definovali zložitejšiu podmienku, ktorá by bola závislá na ďalšej podmienke, tak tieto systémy majú natívnu podporu takéhoto vyhľadávania. V našom systéme by sme takúto podporu museli implementovať ručne.

Náš systém má vlastnú databázu, vlastné predpočítané hodnoty a beží samostatne. Kedže nie je na ničom závislý, jeho integrácia do stávajúceho vyhľadávacieho systému by bola jednoduchá. Nevyužilo by sa užívateľské rozhranie, ale iba vyhľadavací systém, ktorého aplikačné rozhranie by bolo využívané aplikáciami tretích strán.

Konkurenčné systémy zabúdajú na používateľa a ignorujú základne vyhľadávacie vzory. Napríklad nevyužívajú automatické dopĺňanie, či sú príliš komplikované, nepredradzujú často vyhľadávané výsledky a neimplementujú ďalšie vyhľadávacie vzory. Nie sú určené pre bežných používateľov a vyžadujú vedomosti, ktoré by už užívatelia mali mať z používania podobných systémov. Preto sa tento typ vyhľadávania veľmi nerozšíril medzi bežné vyhľadávacie systémy.

Kapitola 10

Budúci vývoj projektu

Náš systém nie je dokonalý a má svoje nedostatky, ktoré môžu byť jednoducho odstránené. V tejto kapitole sa dozvieme čo by sa na systéme dalo zlepšiť, aby náš systém, bol kvalitnejší a užívateľský prívetivejší. Kedže sme sa pri návrhu rozhodli systém robiť modulárnym spôsobom, a teda rozdeliť jednotlivé logické celky, je možné túto skutočnosť využiť.

10.1 Kontainerizácia

Kontainerizácia znamená rozdeliť celý systém na samostatné aplikácie, čo v našom prípade znamená jednotlivé celky do kontainerov. Kontainer je samostatná jednotka v ktorej beží samotná aplikácia a nie je ovplyvňovaná rozličnými faktormi, ktoré môžu nastať. Nad kontainermi môže byť nastavená a implementovaná logika na reštart kontainerov pri páde, jednoduchšia údržba a podobné rutiny, ktoré si teraz musíme riešiť sami. Ďalšou výhodou kontainerov je, že si bezstavové aplikácie, v našom prípade vyhľadavací systém, môžeme replikovať. Replikácia kontainerov nám umožní vytvoriť rozdeľovač zátaže (z angl. *load balancer*), pomocou ktorého môžeme rozdeľovať zátaž na jednotlivé kontainery. Princíp celej kontainerizácie spočíva v možnosti nastavenia celého systému v cloudovom clustri, čím získame výkon, ale hlavne vysokú dostupnosť (z angl. *high availability*) celého systému.

Na vytváranie kontainerov by sme mohli využiť aplikáciu *docker* [14] alebo *rkt* [18], ktorá by umožňovala vytváranie a správu kontainerov. Samozrejme, ak máme kontainery, ich manažment v cloude môžeme riešiť ručne alebo je možné využiť ďalší systém, ktorý by sa staral o manažment v cloude. Potenciálny systém, ktorý by sme mohli využiť je *Kubernetes* [10, 19]. Všetky systémy sú zadarmo a ich využitie pre náš systém by bolo veľmi užitočné.

10.2 Aspektový filter

Aspektový filter je v rozhraní možné upravovať priamo bez akýchkoľvek nápoved, či doporučení, čo iritovalo niekoľkých ľudí, ktorí so systémom pracovali. Aspektový filter by okrem aktuálneho formulára, ktorý umožňuje zadávať ľubovoľné hodnoty, mal obsahovať aj ďalší s prepracovanejšu logikou. Pri tomto formulári vidíme možnosť zlepšenia a to takú, že sa dodá nový typ formulára, ktorý bude dávať na výber v prvom políčku z dostupných predikátov. V ďalšom políčku by sa dali vyberať iba objekty, ktoré vybratý predikát môže nadobúdať. V tomto formulári by si samozrejme užívateľ mohol vybrať typ porovnávania. Takýto formulár by určite vylepšil užívateľský zážitok zo systému. A taktiež by umožnil so systémom pracovať aj úplnym laikom.

10.3 Personalizácia

Z pohľadu grafického rozhrania môžeme za nedostatok považovať nemožnosť ukladať si obľúbené hľadania, záložky, dôležité veci a podobne. Väčšina vyhľadávacích systémov si zapamätáva užívateľove vyhľadávania a na ich základe mu umoží napovedať podobné vyhľadávania alebo spresniť to čo vyhľadáva. V našom prípade by išlo o ukladanie definície množiny entít. Napríklad definujeme si množinu automobilových spoločností alebo množinu maliarov a podobné množiny entít, ktoré by sme mohli využiť. Na základe personalizovaných dát, by systém mohol byť schopný upravovať automatické napovedanie a predradzovať personalizované výsledky pred ostatnými. Predradzovanie a ohodnocovanie relevancie týchto dát vo vyhľadávaniach by moholo byť riešené pomocou neurónových sietí, teda s využitím *machine learningu*.

10.4 Hierarchické dáta

Zatiaľ nám náš systém umožňoval vyhľadávať na základe kľúčového slova a aspektového filtra, kde sa dajú definovať jednotlivé filtrovacie záznamy. Pridávanie do filtra je umožnené pomocou formuláru alebo z detailu entity. To, čo však systému chýba je dopĺňanie alebo návrhy záznamov do aspektového filtra. Ide hlavne o hierarchické dáta a tým sa myslí napríklad typ entity, kde typ môže byť hierarchický usporiadaný. Tieto dáta by mohli byť zobrazované napríklad v ľavej časti obrazovky, kde by boli na výber podobné typy entít. Okrem typu entity sa hierarchické dáta dajú využívať ku kategorizácií entít. Tento princíp by nam mohol pomôcť vyberať si entity z podobných kategórií. Tieto dáta by mali byť zobrazené stromovou štruktúrou tak, aby boli zreteľné vzťahy medzi jednotlivými záznamami. Kažopádne takáto funkcionalita si vyžaduje dobre navrhnutú logiku možnosti výberu typu, či kategórie a to tak, aby zbytočne nezaťažovala systém. Bolo nutné si značnú časť dát predpočítať a následne cacheovať. Poprípade pri výbere správnych záznamov využívať natrénovanú umelú inteligenciu.

Kapitola 11

Záver

Na začiatku práce sme sa v úvodných kapitolách dozvedeli aké veľké sú dátové sady ukladané v informačných systémoch, aké jazyky a serializačné formáty sú v takýchto prípadoch používané. Ďalej sme sa v práci dozvedeli ako sú udržiavané vzťahy medzi entitami. Následne sme sa pozreli na riešenie podobných systémov, ktoré boli implementované väčšími spoločnosťami pre vlastné potreby.

Štandardné prehľadávanie Wikipédie neposkytuje aspektové vyhľadávanie vo svojom rozsiahlom obsahu článkov. Táto nutnosť jednoduchého aspektového vyhľadávania vyústila v niekoľko projektov. Tie boli popísané v kapitole 6. Niektoré z týchto projektov sa už nepoužívajú. Existujúce používajú neintuitívny prístup vyhľadávania, funkcionalita je obmedzená alebo obsahujú nedokonalosti, čo odrádza ľudí ich používať. Preto vznikla táto práca, ktorá študuje existujúce riešenia a učí sa z ich chýb pri návrhu vlastného systému.

Nami implementované riešenie vychádza z niekoľkých návrhových vzorov a poznatkov, ktoré boli zozbierané viacerými spoločnosťami pri navrhovaní užívateľských rozhraní. Samotné návrhové vzory sme si popisovali v kapitole 7, kde sme sledovali ich využitie a prínos v informačných systémoch. Pri rozoberaní užívateľských rozhraní sme sa snažili osvojiť si dobré praktiky pri navrhovaní systémov poučením sa už implementovaných rozhraní. Preto sme náš systém pri návrhu rozdelili na viac častí.

Najťažšou časťou bol návrh databázovej schémy tak, aby vyhovoval svojou jednoduchosťou implementovanému systému vyhľadávania a taktiež, aby rýchlosť vyhľadávania bola dostatočná a so systémom sa dalo pracovať v reálnom čase. Bolo vyskúšaných niekoľko databázových schém na ktorých boli vykonané testy, ktoré sledovali čas za ktorý sa jednotlivé požiadavky vykonajú. Vybraná a použitá bola databázová schéma s najnižším časom spracovania dát (kapitola 9). Pri tejto časti sme strávili veľa času, pretože experimenty pri vkladaní dát do databázy vyžadovali neustálu zmenu databázovej schémy a teda aj neustále odstraňovanie dát. To spôsobovalo spomalenie pri vývoji.

Ďalšou časťou, ktorá sa výrazne podieľa na využiteľnosti systému bol aspektový filter, ktorý bol navrhnutý s ohľadom na jednoduchosť použitia. Do aspektového filtra je možné pristupovať viacerými spôsobmi, čo ocenia užívatelia, ktorý si budú chcieť nadefinovať vlastné filtrovacie pravidlá a nielen využiť pravidlá nadefinované dátovou sadou. Navrhnúť túto časť tak, aby umožňovala užívateľom čo najviac možností bolo veľmi zložité a podarilo sa nám to čiastočne. Pri tejto časti by bolo dobré využiť dáta od užívateľov, ktoré by v kombinácií s využítím machine learningu mohli slúžiť ako skutočný aspektový napovedač. To je už však na budúcom vývoji projektu.

Samotné grafické rozhranie systému je tvorené najnovšími knižnicami pre tvorbu reaktívnych webových aplikácií. Možnosti týchto knižníc sú veľmi rozsiahle a je možné nimi

vytvoriť veľmi komplexné grafické rozhrania a taktiež aj komplexnú logiku v pozadí. Naše grafické rozhranie taktiež využíva Material design, čo je dnes trendom a na jeho vlastnosti a mechanizmy si užívatelia navykli z rozhraní, ktoré sú implementované v mobilných zariadeniach. Práve prepracovanosť Material designu posúva hranice užívateľských rozhraní na ďalšiu úroveň. Kombinácia týchto knižníc nám umožnila vytvoriť štandardizované a ľahko udržovateľné grafické rozhranie. Bližšie informácie ohľadom nášho rozhrania sme poskytli v sekcii 8.3.

Nami navrhnuté riešenie je vlastne oddelený systém od systému Wikipédie. Beží nezávisle a je rozdelený na niekoľko samostatných celkov. To zabezpečuje jednoduchý vývoj celkov, novej funkcionality, zobrazovania podľa najnovších trendov a pod. Pritom nie je nutné riešiť ostatné moduly. Systém umožňuje vyhľadávanie v relačnom databázovom systéme s vyžitím aspektového filtra, ktorý je možné upravovať priamo alebo je možné zadávať nové aspekty prostredníctvom formulára. Tento systém bol taktiež navrhnutý tak, aby sa dal použiť v už implementovaných systémoch ako ich rozšírenie pri filtrovaní výsledkov. Úplnú špecifikáciu aplikačného rozhrania našeho systému je možné nájsť v sekcii 8.2. Náš systém nie je náhradou skutočného vyhľadávania vo Wikipédií, avšak môže slúžiť ako jeho doplnok tým, že bude filtrovať množinu výsledkov pomocou aspektov.

V kapitole 7 boli priblížené návrhové princípy, ktoré sme sa snažili využit. Taktiež sa dočítame o dôvodoch prečo sa dané metódy využili. Z týchto princípov sa implementoval spomínaný systém, ktorého konkrétnu implementáciu môžeme nájsť v kapitole 8, kde je jeho presnejšia špecifikácia z ktorej sme vychádzali pri navrhovaní testovania, ktoré je možne nájsť v kapitole 9. V tejto kapitole sme systém testovali oddelene, po častiach, až na poslednú časť, ktorá používala vyhľadávací systém. Časť testovania prebiehala automaticky, no niektoré časti sme museli vyhodnotiť ručne. Pri testovaní sme došli k záveru, že systém sa dá používať na aspektové vyhľadávanie, avšak pre zlepšenie kvality by bolo nutné systém rozšíriť minimálne o aspektový napovedač, ktorý by bol postavený na machine learningu. Nakoniec v kapitole 10 sme si povedali o možných zlepšeniach systému. Tieto zlepšenia sa týkali samotného grafického rozhrania. Išlo hlavne o jeho zlepšení kvôli užívateľskému zážitku. V tejto kapitole sú uvedené vylepšenia, ktoré by nám pomohli systém dostať do cloudového systému, s jeho jednoduchou replikáciou, výkonnostným škálovaním a pod.

Literatúra

[1] Auer, S.; Bizer, C.; Kobilarov, G.; aj.: DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC+ASWC 2007)*, November 2008, ISBN 978-3-540-76298-0.

[2] Date, C. J.: *SQL and Relational Theory: How to Write Accurate SQL Code*. Beijing Cambridge Mass: O'Reilly Media, Inc., prvé vydanie, 2009, ISBN 0596523068.

[3] DBpedia.org: *Entity Search, Find, and Explore - Documentation*. [Online; navštívené 10.03.2018].
URL http://dbpedia.org/fct/facet_doc.html

[4] Deutschland Wikimedia: *Wikidata*. [Online; navštívené 14.01.2018].
URL https://www.wikidata.org/wiki/Wikidata:Main_Page

[5] Ecma International: *The JSON Data Interchange Format*. Standard ECMA-404, Oct 2013, [Online; navštívené 01.05.2018].
URL http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[6] Elmasri, R.; Navathe, S.: *Fundamentals of Database Systems*. Boston, MA: Addison-Wesley Publishing Company, 6 vydanie, 2010, ISBN 0136086209.

[7] Erling O., M. I.: *Networked Knowledge - Networked Media Integrating Knowledge Management, New Media Technologies and Semantic Systems*. 2009, ISBN 3642021832.

[8] Google Inc.: *Material Design*. 2014, [Online; navštívené 10.04.2018].
URL https://material.io/

[9] Hahn, R.; Bizer, C.; Sahnwaldt, C.; aj.: *Faceted Wikipedia Search*. In *Business Information Systems*, Springer Berlin Heidelberg, 2010, s. 1–11, doi:10.1007/978-3-642-12814-1_1, [Online; navštívené 10.03.2018].
URL https://doi.org/10.1007/978-3-642-12814-1_1

[10] Hightower, K.; Burns, B.; Beda, J.: *Kubernetes: Up and Running Dive into the Future of Infrastructure*. O'Reilly Media, Inc., prvé vydanie, 2017, ISBN 1491935677.

[11] iProspect: Blended Search Results Study. Apr 2008.

[12] Knuth, D. E.: *Art of computer programming. Volume 3, Sorting and searching*. Mathematical Sciences Publisher, 1998, ISBN 978-0-201-89685-5.

[13] Kroenke, D. M.; Auer, D. J.; Vandenberg, S. L.; aj.: *Database Concepts*. Pearson, 8 vydanie, 2017, ISBN 978-0134601533.

[14] Merkel, D.: *Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux J.*, ročník 2014, č. 239, Marec 2014, ISSN 1075-3583, [Online; navštívené 20.04.2018].
URL http://dl.acm.org/citation.cfm?id=2600239.2600241

[15] Morville, P.: *Search patterns*. Sebastopol, CA: O'Reilly Media, Inc., 2010, ISBN 9781449380205.

[16] Osmani, A.: *Learning JavaScript design patterns*. Sebastopol, CA: O'Reilly Media, Inc., 2012, ISBN 978-1-449-33181-8, I-XII, 1-235 s.

[17] Pasquali, S.: *Mastering Node.js: expert techniques for building fast servers and scalable, real-time network applications with minimal effort*. Birmingham, UK: Packt Publishing, 2013, ISBN 1782166327.

[18] RedHat, Inc.: *rkt, A security-minded, standards-based container engine*. 2018, [Online; navštívené 29.04.2018].
URL https://coreos.com/rkt/

[19] Rensin, D. K.: *Kubernetes - Scheduling the Future at Cloud Scale*. 1005 Gravenstein Highway North Sebastopol, CA 95472, 2015, ISBN 9781491935996, [Online; navštívené 13.03.2018].
URL http://www.oreilly.com/webops-perf/free/kubernetes.csp

[20] Robinson, I.; Webber, J.; Eifrem, E.: *Graph Databases*. Sebastopol, Calif. Sebastopol, CA: O'Reilly Media, Inc., 2013, ISBN 1449356265.

[21] Schwartz, B.: *High performance MySQL*. Beijing Sebastopol: O'Reilly Media, Inc., Dec 2008, ISBN 0596101716.

[22] Smith, R. G.: *Knowledge-Based Systems. Concepts, Techniques, Examples*, May 1985.

[23] Stallings, W.: *Computer Organization and Architecture: Designing for Performance*. Upper Saddle River, NJ, USA: Prentice Hall Press, 8 vydanie, 2009, ISBN 9780136073734.

[24] Suchanek, F. M.; Kasneci, G.; Weikum, G.: Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-654-7.

[25] Tanenbaum, A. S.; Woodhull, A. S.: *Operating Systems Design and Implementation*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., tretie vydanie, 2005, ISBN 0131429388.

[26] Tunkelang, D.: *Faceted search*. San Rafael, Calif: Morgan & Claypool Publishers, 2009, ISBN 9781598299991.

[27] Völkel, M.; Krötzsch, M.; Vrandecic, D.; aj.: Semantic Wikipedia. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-323-9.

[28] W3C: *Extensible Markup Language (XML)*. [Online; navštívené 02.03.2018].
URL https://www.w3.org/XML/

[29] W3C: *RDF 1.1 Concepts and Abstract Syntax*. [Online; navštívené 13.01.2018].
URL https://www.w3.org/TR/rdf11-concepts/

[30] W3C: *RDF 1.1 N-Quads*. [Online; navštívené 13.01.2018].
URL https://www.w3.org/TR/n-quads/

[31] W3C: *RDF 1.1 Turtle*. [Online; navštívené 13.01.2018].
URL https://www.w3.org/TR/turtle/

[32] W3C: *RDF Primer*. [Online; navštívené 13.01.2018].
URL https://www.w3.org/TR/rdf-primer/

[33] yaml.org: *YAML Ain't Markup Language*. [Online; navštívené 02.03.2018].
URL http://yaml.org//

Zoznam obrázkov

Zoznam tabuliek

Príloha A

Inštalácia

Pre správne spustenie systému je nutné spraviť niekoľko krokov. Súborová štruktúra je rozdelená a každý priečinok obsahuje iba príslušnú časť. Pamäťové médium ma tedá nasledovnú štruktúru:

- extractor – priečinok obsahuje časť zaoberajúcu sa extrakciou dát a ukladaním do databázy.
- web_php – priečinok obsahuje webový server, ktorý je vyhľadavacím systémom.
- web_client – priečinok obsahuje grafické užívateľské rozhranie v ReactJs.
- tex – priečinok obsahuje zdrojové kódy dokumentu diplomovej práce.

Pre úspešné nainštalovanie systému je nutné vykonať niekoľko krokov. Keďže je aplikácia rozdelená na viacero častí je nutné nastaviť prístupy na databázu alebo aplikačné rozhranie. Teda po rozbalení by sa mali vykonať nasledujúce kroky:

1. Nastaviť prístup do MySQL databázy a to upravením konfigurácie v súbore

   `extractor/app/database/Database.py`

2. Nainštalovať všetky závislosti pre *Python*.
3. Stiahnúť súbory[1] s dátami a uložiť do zložky `extractor/data`.
4. Následne spustiť extraktor zavolaním hlavného skriptu, ktorý naimportuje dáta zo stiahnutých súborov. Ak bude v dátach súbor `short_abstracts_en.tql`, tak sa naimportujú aj abstrakty entít.

   ```
   python3 extractor/main.py
   ```

5. Upravíme konfiguráciu MySQL databázy pre vyhľadavací server v súbore

   `web_php/app/config/config.local.neon`.

6. Nainštalujeme všetky závislosti pomocou príkazu

   `composer update`.

7. Vytvoríme priečinky `log` a `temp` a nastavíme im práva 777.

[1] http://wiki.dbpedia.org/Datasets

8. Spustíme vyhľadávací server pomocou príkazu

   `php -S localhost:80 -t www/`.

9. Nainštalujeme závislosti pre grafické rozhranie – v zložke `web_client/` zavolaním príkazu

   `npm install`.

10. Upravíme konfiguračný súbor pre grafické rozhranie tak, aby sa pripájalo na aplikačné rozhranie, kde bude bežať vyhľadavací systém. Konkrétne ide o premennú `baseRestUrl` v súbore `web_client/src/api/urls.tsx`.

11. Spustíme grafické rozhranie pomocou príkazu

    `npm run start`.

Popísaná inštalácia je vhodná na vývoj systému nie však na produkčný režim. Je potrebné si pri konfigurácií dávať pozor na porty a nastavenie aplikačného rozhrania a databázového prístupu.

Príloha B

Dotazník

1. Funguje a dopĺňa správne automatické dopĺňanie na vstupnej stránke ?
   - Áno.
   - Nie.
2. Abstrakt v automatickom dopĺňaní nie je dostupný pri všetkych entitách. Hralo to rozhodovaciu rolu pri Vašom výbere ?
   - Áno.
   - Nie.
3. Ako dlho Vám trvalo vyhľadanie všetkých automobilových spoločností ?
   - Menej ako minútu.
   - Menej ako dve minúty.
   - Viac ako dve minúty.
4. Ako hodnotíte rozhranie z pohľadu užívateľskej prívetivosti ?
   - Rozhranie je prehľadné, jednoduché, všetky funkcie mi boli hneď jasné.
   - Niektoré funkcionality som nemohol nájsť.
   - Neprehľadné a chaotické.
5. Čo by ste na systéme zlepšili ?
   - Nič.
   - Doplňte.

Príloha C

Plagát

Obr. C.1: Plagát práce.