Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Bavcon

Spletni robot in iskalnik po evropskih projektih

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Izdelajte spletni iskalnik sprejetih evropskih projektov, ki bo s pomočjo prilagodljivega spletnega robota agregiral podatke z več spletnih mest. Zbrani podatki naj bodo predstavljeni na poenoten način. Spletni iskalnik naj bo realiziran kot enostranska spletna aplikacija, ki preko REST storitev komunicira z iskalnim strežnikom. Ta naj ponuja različne, fleksibilne načine iskanja po podatkih o projektih, ter fasetno iskanje v kombinaciji z gručenjem. Pristop ovrednotite v primerjavi s sorodnimi, ter poudarite njegove prednosti in morebitne slabosti.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani David Bavcon, z vpisno številko **63090058**, sem avtor diplomskega dela z naslovom:

Spletni robot in iskalnik po evropskih projektih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. septembra 2014 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Matjažu Kukarja za pomoč in vse koristene nasvete pri izdelavi diplomskega dela.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Sorodni pristopi 2

2 Viri podatkov o evropskih projektih **5**

2.1 Cordis 5

2.2 Adam 6

2.3 Adriatic ionian macroregion area 6

2.4 Alpine Space 7

2.5 Central 2013 7

2.6 Med programme 7

2.7 South east Europe 7

2.8 Urbact 8

2.9 Environment Life programme 8

2.10 Nekaj drugih virov podatkov, ki niso vključeni 8

2.11 Ostali viri podatkov 9

3 Namenski spletni robot za pridobivanje podatkov iz različnih virov **11**

3.1 Podroben opis delovanja posameznih korakov 13

3.2 Delovanje robota 16

3.3 Primer vključitve nove strani 23
3.4 Primerjava našega namenskega robota z drugimi roboti 24
3.5 Slabosti in pomankljivosti 27

4 Arhitektura celotnega sistema **31**
4.1 Zgradba podatkovnega modela 31
4.2 Arhitektura našega sistema 33

5 Iskalnik **37**
5.1 Tehnologije 37
5.2 Implementacija 42

6 Uporabniški vmesnik **51**
6.1 Enostranska spletna aplikacija 51
6.2 AngularJS 52
6.3 Uporabniški vmesnik 55

7 Rezultati **63**

8 Sklepne ugotovitve **65**
8.1 Možnosti za nadaljnje delo 67

Povzetek

Cilj našega dela je izdelava iskalnika evropskih projektov, s katerim bomo na enem mestu lahko poiskali vse projekte iz različnih virov in s tem tudi nadomestili obstoječe pomanjkljive iskalnike. Zaradi tega je naše delo sestavljeno iz dveh delov. V prvem delu izdelamo namenskega spletnega robota ki, če je to potrebno, izvede tudi JavaScript in vse spletne strani shrani v lokalni datotečni sistem. Potem razčleni vse podatke iz shranjenih strani in jih zapiše v podatkovno bazo. V drugem delu se posvetimo izdelavi samega iskalnika s pomočjo JavaScripta, ki dobiva podatke preko storitev REST. Za iskanje uporabljamo iskalno platformo Apache Solr, ki nam omogoča številne napredne funkcije iskanja in tako poskušamo dobiti čim boljše rezultate iskanja ter navigiranje po njih.

Ključne besede: spletni robot, iskalnik, JavaScript, iskalnik evropskih projektov, enostranska spletna aplikacija

Abstract

The goal of our work is to develop a search engine, which will have in one place all projects from various sources and replace the lack of existing search engines. For this reason, our work consists of two parts. In the first part we create dedicated web robot that also, if necessary, execute JavaScript, and saves all pages in local file system. Then we parse all the data from saved pages and insert data into database. The second part of our work is developing a single page application using JavaScript, which receives data from REST services. For searching we use search platform Apache Solr that enables a number of advanced search features and so we try to get as better search results and navigate through them.

Keywords: web robot, search engine, JavaScript, european project search engine, single page application

Poglavje 1

Uvod

Viri podatkov o evropskih projektih so zelo razpršeni in nimajo na voljo enotnega iskalnika. Zaradi tega smo se odločili, da naredimo iskalnik, ki bo to omogočal. Seveda ima trenutno vsak vir projektov svoj iskalnik včasih pa zgolj nekakšen filter projektov. Tako sedaj pri določenih virih podatkov sploh ne moremo iskati s ključnimi besedami, ker tega nimajo na voljo ali pa lahko iščemo z besedami samo po nekaterih atributih. Zaradi tega smo se odločili, da naredimo iskalnik, ki bo iskal po različnih virih podatkov na enem samem mestu. V samem iskalniku smo omogočili napredne funkcije iskanja, tudi iskanje po celotnem tekstu, ki ga marsikateri vir podatkov ne omogoča.

V prvem delu našega dela smo morali za uresničitev našega cilja najprej pripraviti spletnega robota, ki bo zbral ustrezne podatke. Zaradi vse modernejših spletnih strani in vse večje uporabe JavaScripta naš robot izvede tudi JavaScript. Z izvedbo samega JavaScripta tako pridobimo tudi vsebino, ki je podana samo preko JavaScripta in drugače ni prisotna. V letošnjem letu je tudi Google objavil, da je začel z izvajanem JavaScripta na spletnih straneh, da bo tako pridobil vso vsebino, če bo le mogoče [30]. Med samim delom smo se pri robotu srečali s problemi zaradi določenih strani, ki niso delovale ali so se med pisanjem diplomskega dela spremenile. Vse zbrane podatke robot ustrezno razčleni in shrani v podatkovno bazo, ki se jo bo lahko uporabljalo

tudi za druge namene. Našega namenskega robota smo primerjali z drugimi tehnikami in opisali razlike med možnimi rešitvami.

V drugem delu našega dela smo se ukvarjali s samim iskalnikom. Za iskanje smo uporabili platformo za iskanje Apache Solr [3], s katerim smo realizirali napredne funkcije iskanja, da bi še lažje in hitreje prišli do želenega rezultata. Uporabljali smo funkcije samodokončaj, fasetno iskanje, mehko iskanje, gručenje rezultatov za uporabo pri fasetnem iskanju in označevanje zadetkov. Za iskanje poskrbi odzivna aplikacija v eni sami strani in z uporabo JavaScripta, ki se izvaja na strani odjemalca. Aplikacija podatke pridobiva preko storitev REST, ki jih nudi Solr.

V okviru našega dela smo pripravili tudi dokumentacijo in robota pripravili tako, da ga bo mogoče nadgraditi še za druge strani s projekti, ki jih v naše delo nismo vključili. V dokumentaciji smo označili, katere atribute smo pridobivali iz posameznih strani. Seveda naša rešitev ne bo dolgotrajna, saj bodo verjetno kmalu spet potrebni kakšni popravki zaradi sprememb strani in podatkov. V diplomskem delu smo tudi primerjali različne možne rešitve za naše probleme in opisali njihove prednosti ter pomanjkljivosti.

1.1 Sorodni pristopi

Robota za pridobivanje podatkov iz Cordisa so naredili v več člankih. V članku [31] so to naredili za to, da so potem s temi podatki delali vizualno gručenje podatkov. V primerjavi z našim delom je z omenjenim člankom skupno to, da smo naredili namenskega robota in smo shranjevali strani v datoteke. Uporabili so tudi gručenje, ki ga pa neposredno ne moramo primerjati z našim delom, ker je bil končni cilj gručenja drugačen.

V članku [25] je predstavljen iskalnik, ki išče po znanstvenih in izobraževalnih ustanovah v Argentini. Namesto izdelave centralne baze, v katero bi moral vsak vnašati podatke so se raje odločili, da uporabijo robota, ki bo zbral vse podatke iz spletnih strani. Iz zbranih podatkov potem pridobijo entitete, osebe, institucije in kraje. Te podatke pa pridobivajo s procesira-

njem naravnega jezika. V iskalniku so uporabili fasetno iskanje, ker so želeli, da bo iskalnik čimbolj prijazen navadnim uporabnikom.

Primerjava našega dela s prej opisanim delom [25], se razlikuje bistveno v tem, da smo podrobno razčljenjevali posamezne atribute iz same izvorne kode. V primerjanem delu so se raje odločili za pridobivanje želenih podatkov iz samega teksta, kar je seveda zelo zahtevno. Pri tem moramo upoštevati, da smo lahko delali na takšen način, ker smo imelo malo virov podatkov, če bi imeli več kot 25 virov podatkov, bi bilo vprašljivo, če je naše delo smiselno in ali ni boljše uporabiti takšen način, kot so ga uporabili v primerjanjem članku. Z uporabo iskalne platforme smo v primerjavi s člankom na istem in podobno s samo spletno aplikacijo, kjer so raje uporabili GWT (Google web toolkit).

Našli smo tudi bazo s podatki iz Cordisa predstavljeno v RDF obliki [7], za katero pa ne vemo, če ni bila namenjena le manjši skupini projektov v katerih je sodelovala njihova univerza. V primerjavi s to bazo [7], smo vsekakor boljši, saj oni nimajo ustreznega uporabniškega vmesnika za iskanje in ravno tako ne nudijo naprednih funkcij iskanja, potrebno je uporabiti poizvedovalni jezik.

Omeniti moramo še tri članke, ki so se ukvarjali tudi s Cordisovo bazo. Z omenjenimi članki je skupno to, da uporabljamo Cordisovo bazo za razne analize. V enem članku je tako predstavljeno iskanje podobnosti [36], v drugem pa se ukvarjajo s hibridnim sistemom in sistemom za pridobivanje informacij [32]. Tretji članek [29] opisuje učinke in organizacijo financiranja z javnimi sredstvi s pomočjo Cordisove baze.

Poglavje 2

Viri podatkov o evropskih projektih

Iskalnika ne moremo narediti, če nimamo podatkov. Tako smo se morali najprej odločiti, katere vire podatkov evropskih projektov bomo uporabili. Odločili smo se tudi, da bomo uporabljali pojem vir podatkov, ker imajo same spletne strani razne druge podatke o projektih, kot so na primer novice, ki pa nas ne zanimajo. Viri podatkov vključujejo projekte, ki so se že končali ali pa se trenutno izvajajo. Seveda to gotovo niso vsi viri podatkov o evropskih projektih, na voljo so še drugi, ki pa jih zaenkrat nismo vključili. Sledijo kratki opisi virov podatkov in področje, ki ga pokrivajo.

O projektih nas na tem mestu zanimajo: naslov, datum začetka, datum konca, opis, referenčna koda, akronim, status, predmet, vodja projekta, partnerji, vrednost projekta, vrednost sofinanciranja s strani Evropske unije in spletna stran projekta. Vsi viri podatkov ne ponujajo vseh želenih atributov.

2.1 Cordis

Cordis je naš največji vir podatkov, ki vsebuje podatke o raziskovalnih projektih, ki so bili sofinancirani s strani Evropske unije. Osredotočili smo se zgolj na projekte, drugih informacij, kot so rezultati in novice nismo pridobi-

vali. Poleg tega smo vzeli samo projekte iz okvirnih programov FP5, FP6 in FP7, kar predstavlja približno polovico vse projektov v tem viru podatkov. Ta vir podatkov je tudi služil za izdelavo podatkovnega modela, ker vsebuje največ atributov za posamezen projekt. Med našim delom so spletno stran prenovili zato smo morali našega robota ustrezno popravljati. Poudariti je potrebno, da je stran preobremenjena zato velikokrat ne dela. Zelo pogosto se zgodi, da preteče časovna omejitev ob obisku posameznega projekta. Podatke iz te strani lahko pridobimo tudi v dokumentu Excel ali pa formatu CSV za vsak okvirni program v svoji datoteki.

2.2 Adam

Adam je portal, namenjen predstavitvi različnih projektov iz evropskega programa izobraževanja in usposabljanja. Na voljo ima tudi rezultate projektov, ki jih nismo uporabili. Uporablja se tudi za iskanje partnerjev pri projektih. Pri tem viru podatkov smo se srečali z zelo neprijazno strukturo DOM, zato smo morali iz posameznih vrstic pridobiti ključ in vrednost, da smo izvedeli, za kakšen atribut sploh gre. Vsi projekti niso imeli vseh atributov, sicer bi lahko upoštevali številke vrstic.

2.3 Adriatic ionian macroregion area

Ta vir podatkov smo poimenovali s krajšim imenom Ai macroregion in pokriva projekte iz držav, ki so okoli Jadranskega morja. Vključuje projekte, ki so iz naslednjih virov podatkov: Adriatic IPA, Med Programme, South East Programme, Interreg IVC in Urbact. Kljub temu, da so določeni viri podatkov že vključeni smo mi vseeno še ločeno zajeli te vire podatkov (Med Programme, South East Programme in Urbact). Opazili smo namreč, da določeni projekti manjkajo in imajo določene atribute drugače predstavljeno. Razlog je mogoče to, da tega vira podatkov redno ne posodabljajo in zaradi tega pride do neujemanja. Drugi možni razlog pa je, da projekt ne ustreza

določenim kriterijem in ga za to niso vključili. Slaba stran tega pa je, da imamo tako določene projekte podvojene v bazi. Na tej spletni strani sploh nimamo možnosti iskanja po projektih s ključnimi besedami, ampak imamo na voljo zgolj filter po več atributih. Filter po posameznem atributu ima zgolj izbirnik, iz katerega lahko izberemo že definirane možnosti.

2.4 Alpine Space

Alpine Space je vir podatkov je geografsko omejen na območje Alp. Tudi tukaj nimamo iskalnika, s katerim bi lahko iskali s ključnimi besedami, ampak ponovno le filter po že definiranih vrednostih.

2.5 Central 2013

Tudi ta vir podatkov je geografsko omejen na države Srednje Evrope.

2.6 Med programme

V naši bazi smo ga poimenovanovali Programmemed. Tudi tukaj se srečamo z iskalnikom, ki sicer po nekaj atributih omogoča iskanje s ključnimi besedami. Drugače pa ima izbirnik z ogromno možnostimi. Ravno tako so projekti v tej bazi geografsko omejeni na države ob Sredozemskem morju. Ta vir podatkov je vključen tudi v viru podatkov Ai macroregion.

2.7 South east Europe

Na tej spletni strani imamo, tako kot v predhodno opisanih primerih, pomankljiv iskalnik. Izpostaviti je potrebno tudi geografsko omejenost. Ta vir podatkov je vključen tudi v viru podatkov Ai macroregion.

2.8 Urbact

Vir podatkov, ki pokriva projekte iz urbanih izzivov v mestih in povezuje mesta, da skupaj razvijajo rešitve za urbane spremembe. Iz tega vira podatkov smo dobili najmanj atributov, čeprav je vsak projekt opisan z veliko teksta. Ta vir podatkov je vključen tudi v viru podatkov Ai macroregion.

2.9 Environment Life programme

Vir podatkov smo poimenovali Life. Vsebuje odobrene projekte s področja okolja, ohranjanja narave in podnebnih sprememb. Na tej spletni strani se uporablja za prehajanje med samimi stranmi z rezultati iskanja JavaScript. Spletni naslov se zaradi uporabe metode POST ne spreminja. Da smo prelistali vse strani, smo morali klikati z gumbom "naprej".

2.10 Nekaj drugih virov podatkov, ki niso vključeni

2.10.1 Cost

Cost ima za razliko od ostalih virov podatkov namesto projektov akcije. To je bil tudi razlog, da ga nismo vključili v našo bazo. Atributi, ki so na voljo v tem podatkovnem viru so podobni, kot jih lahko dobimo iz ostalih. Podatkovni vir tako vsebuje čez 1000 akcij, ki pokrivajo 10 področij.

2.10.2 NWE Programme

Program INTERREG North-West Europe (NWE) geografsko pokriva severozahodno Evropo. Tega vira podatkov nismo vključili, ker projetki v tej bazi geografsko ne pokrivajo naše države. Program pokriva: ekonomske, okoljske in socialne projekte.

2.10.3 European Social Fund

V tem viru podatkov je trenutno malo manj kot 35000 projektov. Projekti omogočajo ljudem boljše službe in podpirajo delovna mesta za evropske državljane. Ta baza geografsko pokriva tudi Slovenijo iz katere je malo več kot 700 projektov. Tega vira nismo vključili zaradi treh razlogov: opisi pri projektih niso prevedeni v angleščino, malo atributov in vir podatkov se že več let ne posodablja.

2.10.4 Daphne Projects

Daphne Projects pokriva projekte, ki preprečujejo nasilje nad otroki, mladimi in ženskami. Tega vira nismo vključili zaradi pomanjkanja atributov in zaradi tega, ker so projekti starejši od 10 let.

2.10.5 CEI Central European Initative

CEI pokriva projekte regijskega sodelovanja. Tega vira podatkov nismo vključili, ker ima ta vir podatkov malo projektov. Ta vir podatkov tudi bistveno bolj odstopa od delovanja našega robota v primerjavi z ostalimi viri podatkov. To je bil tudi drugi razlog, da tega vira nismo vključili.

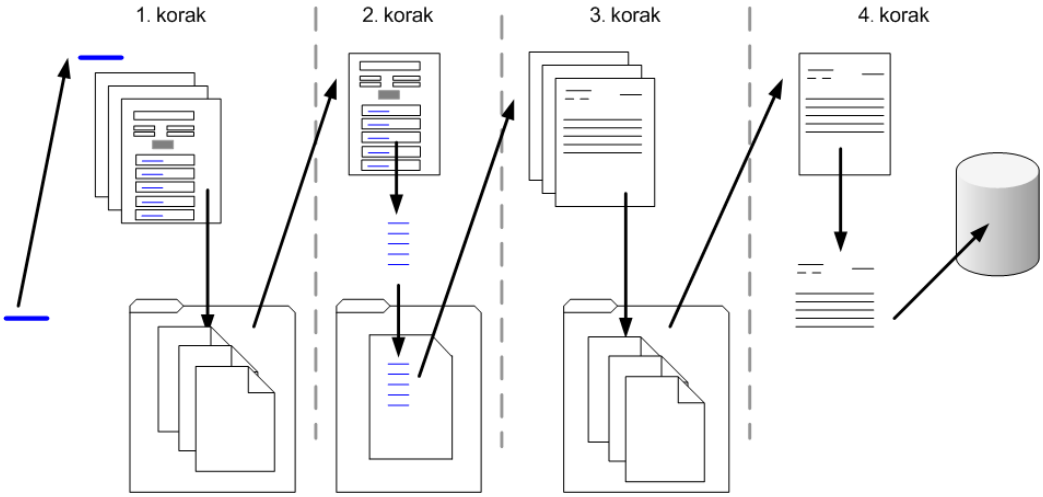
2.11 Ostali viri podatkov

Vsi programi, ki financirajo evropske projekte nimajo svojega vira podatkov, kjer bi bili predstavljeni vsi sprejeti projekti. Večinoma imajo vsi programi financiranja na svoji straneh objavljene klice po projektih, ampak samega vira podatkov o projektih pa ne. Druga možnost je tudi ta, da vir podatkov ni dosegljiv na internetu in je namenjen zgolj interni uporabi. Iskalnik projektov zaradi navedenih razlogov ne vsebuje vseh sprejetnih projektov, ki so se izvedli ali se še izvajajo. Verjetno tudi v prihodnosti ne bomo mogli razširiti našega iskalnika na dejansko vse projekte zaradi prej navedenih razlogov.

Poglavje 3

Namenski spletni robot za pridobivanje podatkov iz različnih virov

Za potrebe iskalnika projektov smo morali najprej pridobiti podatke, zato smo najprej razvili robota v programskem jeziku Python, ki dobi potrebne podatke. Odločili smo se za razvoj lastnega namenskega robota, ki bo deloval zgolj za določene specifične spletne strani in ga je mogoče razširiti še z dodatnimi spletnimi stranmi, ki seveda glede na delovanje robota ne odstopajo preveč od že izbranih. Zaradi vse večje uporabe JavaScripta in bolj dinamičnih strani smo se odločili, da mora robot znati pridobiti vsebino, ki je na strani podana z JavaScriptom in je vidna šele, ko se JavaScript izvede. Potrebo po izvedbi JavaScripta lahko razložimo tudi na konkretnem primeru, ki smo ga uporabili pri enem viru podatkov. Pri tem viru podatkov se med posameznimi stranmi rezultati iskanja spletni naslov ni spreminjal, saj je vse potekalo preko metode POST. Gumb "naprej"je tako ob kliku klical funkcijo JavaScript, ki je ustrezno spremenila parametre. Za rešitev problema smo realizirali klik na gumbu naprej, ki je potem ob kliku izvedel funkcijo JavaScript in tako smo lahko prišli na naslednjo stran. Naveden primer bi sicer lahko rešili tudi z ustreznimi parametri, ki bi jih podali metodi POST.



Slika 3.1: Shema delovanja našega robota.

Za delovanje robota smo uporabili več modulov za Python, in sicer: Selenium za potrebe spletnega gonilnika, BeautifulSoup4 za razčlenjevanje izvorne kode in PyMySQL za potrebe povezave ter operacije s podatkovno bazo MySQL. Poleg navedenih modulov smo uporabili še nekaj standardnih uporabljenih modulov (random, time, datetime, sys, re, os in codecs).

Robotovo delovanje lahko strnemo v štiri korake, ki se med različnimi spletnimi stranmi malo razlikujejo, saj ima vsaka stran kakšno posebnost. S štirimi strnjenimi koraki lahko kljub določenim posebnostim predstavimo bistvo delovanja robota. 4 osnovni koraki robota:

1. robot shrani stran-i, kjer so rezultati iskanja,
2. iz prej shranjenih rezultatov iskanje razčlenimo in pridobimo spletne povezave do posameznih strani,
3. shranjevanje posameznih strani,
4. razčlenjevanje shranjenih strani in pridobivanje podatkov iz njih, ki jih nato shranimo v podatkovno bazo.

3.1 Podroben opis delovanja posameznih korakov

3.1.1 Shranjevanje rezultatov iskanja (prvi korak)

Robotu moramo najprej podati spletno povezavo, kjer se nahajajo rezultati iskanja, s katerimi nato dobimo spletne povezave do posameznih projektov. Tukaj moramo zelo paziti kako s samim iskalnikom na določeni strani dosežemo, da nam iskanje vrne vse rezultate. Na nekaterih straneh lahko dobimo vse rezultate izpisane na eni strani, na drugih straneh pa smo omejeni s prikazom določenega števila rezultatov na posamezno stran. V prej omenjenem primeru se mora robot tako sprehoditi čez vse strani rezultatov iskanja, da je pa teh korakov čim manj, izberemo največje možno število prikazov rezultatov na stran, če nam iskalnik na določeni strani to omogoča.

Pri prehajanju med stranmi rezultatov iskanja moramo določiti tudi zadnjo stran, kjer se končajo rezultati iskanja, če je potrebno, to naredimo v svoji funkciji. Našli smo tudi stran, ki je imela še dodaten problem, in sicer iskalnik ni vrnil več kot določeno število rezultatov. Zaradi tega smo morali za to spletno stran v iskalniku uporabiti filtre, da smo prišli do vseh možnih spletnih povezav filtrov, ki se ne spreminjajo. Šele znotraj pridobljenih povezav smo se nato sprehajali med posameznimi stranmi rezultatov iskanja in tako določili zadnjo stran. Vse pridobljene strani rezultatov iskanja smo si shranili v obliki datotek HTML. Med samim prehajanjem med stranmi imamo nekaj sekund zamika, da ne obremenjujemo preveč spletnih strežnikov.

3.1.2 Razčlenjevanje rezultatov iskanj in pridobivanje povezav (drugi korak)

Drugi korak je namenjen razčlenjevanju spletnih povezav iz shranjenih datotek HTML prvega koraka. Z zanko se sprehodimo čez vse datoteke HTML določene spletene strani in iz vsake datoteke pridobimo vse povezave, ki

kažejo na posamezne strani projekta. Ostale povezave in vsebine nas ne zanimajo. Vse tako pridobljene povezave nato shranimo v tekstovno datoteko (vsaka spletna stran ima svojo), kjer vsaka vrstica predstavlja eno povezavo.

3.1.3 Shranjevanje posameznih strani (tretji korak)

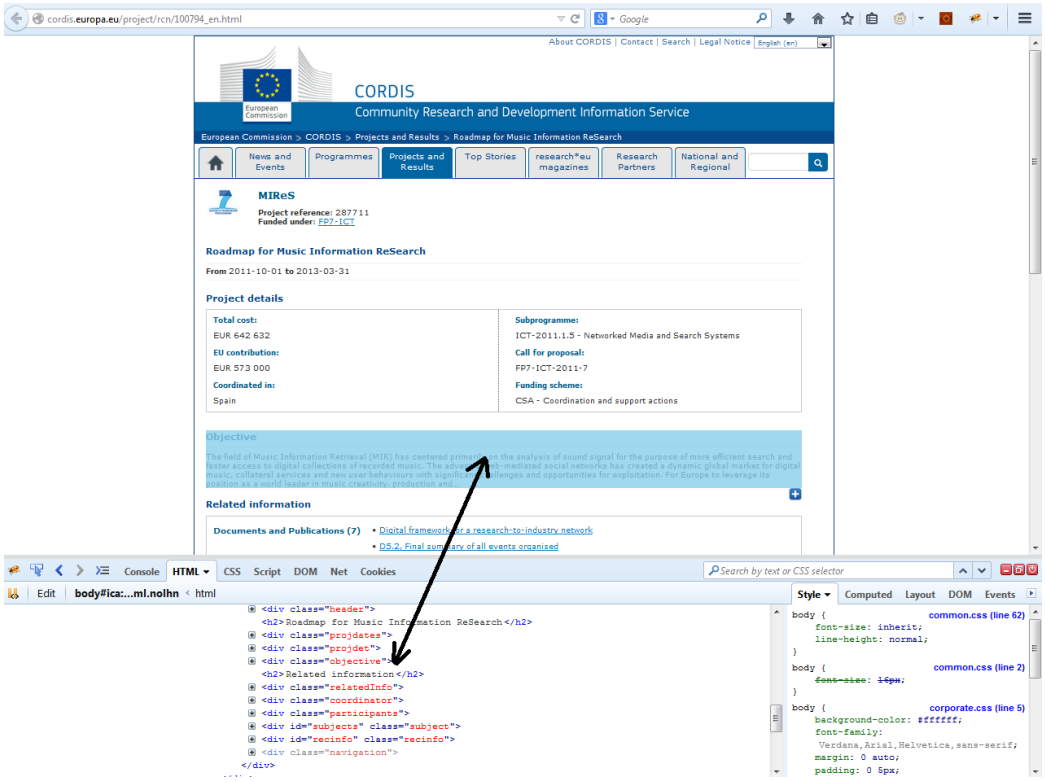
Robot odpre tekstovno datoteko s spletnimi povezavami do posameznih strani projektov. Nato vsako povezavo obišče in shrani izvorno kodo strani v datoteko HTML. Po vsaki obiskani in shranjeni strani robot počaka nekaj sekund pred naslednjim obiskom in shranjevanjem strani. Med obiskovanjem strani pride tudi do napak. Ko smo delali, smo se srečali z naslednjimi napakami:

- strežnik trenutno ni dosegljiv,
- pretok časovne omejitve,
- na strani pride do napake pri izpisu iz baze
- notranja napaka strežnika (napaka 500)

Seveda pa naštete napake niso vse, ki se lahko zgodijo. V prvih dveh primerih napak si robot to povezavo shrani v seznam in na koncu, ko obišče vse strani, naredi še en obhod po shranjenem seznamu napak in poskuša obiskati ter shrani te stran.

3.1.4 Razčlenjevanje posameznih strani in shranjevanje v podatkovno bazo (četrti korak)

Vse shranjene posamezne strani projektov v datotekah HTML sedaj razčlenimo glede na to, katere podatke potrebujemo in kateri podatki so sploh na voljo. Razčlenjevane izvorne kode je čisto specifično za vsako stran posebej, ker ima vsaka stran čisto drugačno strukturo DOM. Strani se zelo razlikujejo po številu podatkov, ki jih dobimo o posameznemu projektu. Kakšna spletna stran ima določene podatke predstavljene, tako da jih ne moremo nedvoumno razčleniti. Za razčlenjevanje izvorne kode smo uporabili BeautifulSoup



Slika 3.2: Prikaz uporabe dodatka Firebug v brskalniku Firefox.

[37], kjer je možno uporabiti različne razčlenjevalnike. Za vizualno pomoč pri izboru ustreznega elementa smo si pomagali z brskalnikom Firefox in dodatkom Firebug [10], ki nam omogoča, če gremo z miško na določen del strani nam v izvorni kodi vizualno označi izvorno kodo pripadajočega elementa (glej sliko 3.2). Lahko pa naredimo tudi obratno, da gremo z miško na določen del izvorne kode in se nam pripadajoči element vizualno označi na strani. Vsak podatek je bilo potrebno posebej razčleniti. Da bi več podatkov na posamezni strani razčlenili na čisto identičen način, ni možno. Pridobljene podatke o projektu nato shranimo v podatkovno bazo MySQL oz. jih posodobimo, če že obstajajo [24].

Na določenih spletnih straneh imamo rubrike, kjer je tako nič kot več podatkov enakih (npr. sodelujoči na projektu). Te podatke shranimo v

ločeno tabelo, isto naredimo s predmeti projekta, ki jih je lahko tudi več. Vsi ostali podatki so shranjeni v eno tabelo. To ponavljamo toliko časa, dokler ne razčlenimo vseh datotek. Seveda se vmes pojavi tudi kakšna datoteka, ki je ne moremo razčleniti. Zgodi se, da dobimo stran z obvestilom o napaki, če na primer določenemu viru podatkov ne deluje v ozadju podatkovna baza ni pa to edina možnost. Na mestu, kjer bi morala biti vsebina, se prikaže samo stavek, kjer piše, da so težave s podatkovno bazo. Ker iz takšnih strani razčlenjevalnik ne more dobiti ustreznih podatkov, se ničesar ne shrani v bazo. Posebnih obravnav takšnih strani nismo naredili, ker bi enostavno morali zbrati vse takšne primere napak. V primeru, da stran deluje normalno, je take primere nemogoče dobiti. V takšnih primerih dobimo vsebino šele ob naslednjem obhodu robota, vendar le v primeru, da se ne pojavijo strani s tekstom o napaki.

3.2 Delovanje robota

Naš cilj je bil, da robot zna dobiti vsebino, ki je tako ali drugače vidna šele po izvedbi JavaScripta. Te vsebine pa ne vidimo, če na primer shranimo posamezno stran z ukazom wget v konzoli. Prvi možni način, kako rešiti ta problem, je, da bi imeli odprt brskalnik in bi robot krmaril po njem. Za to možnost se nismo odločili, ker nismo želeli imeti stalno odprtega brskalnika in gledati, kako robot krmari po njem. Odločili smo se za uporabo Seleniuma [38, 17] s spletnim gonilnikom PhantomJS [15]. Lahko bi seveda uporabili tudi spletni gonilnik za brskalnik Firefox. Selenium [38, 17] je namenjen za avtomatizaciji spletnih aplikacij predvsem za testne namene vendar vsekakor ni omejen samo na to. Naše delovanje robota lahko povežemo z web scraping [21] zaradi uporabe brskalnika in tudi zaradi izvedba JavaScripta, ker tako robota ne ločimo od navadnega uporabnika. Da gre za robota, je vidno šele s tem, da odpira stran za stranjo in gre čez vse strani.

3.2.1 PhantomJS

Namenjen je testiranju spletnih strani, zajemanju posnetkov strani, avtomatizaciji strani in nadzorovanju delovanja nalaganja hitrosti spletnih strani. Vse to dela brez grafičnega načina, uporablja pa pogon WebKit [15]. Seveda upodobitev strani, ki jo naredi, ni popolnoma identična upodobitvi strani, ki jo naredi npr. brskalnik Firefox zaradi uporabe drugega pogona. Najpomembnejše od vsega pa je, da izvede JavaScript. Seveda PhantomJS ni edini, ki izvede JavaScript, obstajajo še druge možnosti. Uporabili smo ga kot spletni gonilnik za Selenium. Z njim smo pridobili dopolnjeno izvorno kodo HTML, ki smo jo nato shranili v datoteko.

3.2.2 BeautifulSoup

BeautifulSoup je knjižnica za Python, namenjena pridobivanju in manipuliranju s podatki iz strukture HTML-ja in XML-ja, pri čemer lahko uporablja različne razčlenjevalnike. Zanj imamo na voljo tri različne razčlenjevalnike: Python html.parser, lxml in html5lib. Razčlenjevalniki se med sabo razlikujejo po hitrosti in po tem, kako obravnavajo napake, če dokument HTML vsebuje napake, kot so nezaključene ter manjkajoče značke. V primeru, da dokument ne vsebuje napak, pri vseh dobimo identičen rezultat [37].

Pri samem razčlenjevanju smo uporabljali predvsem metode: find_all, s katero smo poiskali vse elemente, find, s katerim smo poiskali samo en element, next za naslednji element, nextSibling za sosednji element in samo navigiranje (body.div.p) po strukturi DOM. S tem smo prišli do vseh želenih podatkov. Poleg teh metod smo uporabljali še standardne metode za delo z nizi. Te metode so bile: replace, split, strip in seveda regularni izrazi. Z replace smo se predvsem znebili teksta, ki je bil zraven atributa. Potem smo uporabljali split predvsem za datume, da smo jih ustrezno ločili in spravili v ustrezni format. V nekaterih primerih smo uporabili tudi regularne izraze, da smo dobili ustrezne podatke. Metodo strip smo uporabljali predvsem v primerih, kjer smo se želeli znebili raznih presledkov.

Zaradi veliko izvorne kode HTML, ki jo imajo vsi viri podatkov, se nismo odločili za prikaz razčlenjevanja konkretnega vira podatkov, ker bi nam vzelo preveč prostora. Tako smo se odločili za izdelavo primera razčlenjevanja dokumenta, ki smo ga naredili za vzorčni namišljeni primer projekta. V primeru želimo pokazati in povzeti, na kakšne načine smo razčlenjevali različne vire podatkov in na kakšen način je bila zgrajena struktura DOM. Zelo neugodno je bilo predvsem takrat, kadar je bil projekt predstavljen v tabeli, medtem ko posamezne vrstice in celice niso imele nobenega atributa. Podatkov ni bilo možno dobiti drugače, kot da smo morali podati številko vrstice. To lahko na primeru razčlenjevanja vidimo za primer posameznih programov projekta (programme) (glej v kodi HTML 3.2.2).

```
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title>EU SEARCH</title>
<style>table, td {border: 1px solid black;}</style>
</head>
<body>
<div id="ptitle">Developing search application for searching eu
   projects</div>
<div>
<div class="acronym">EU SEARCH</div>
<table>
       <tr>
              <td>Dates<div>1.1.2014 <br /> 1.1.2015</div></td>
              <td><ul><li><h3>Total cost:</h3>5 000 000,00
                  EUR</li><li
                  class="contribution">Contribution</li><li>1 000
                  000,00 EUR</li></ul></td>
       </tr>
       <tr class="description">
              <td>Description</td>
              <td>Main goal of the project is develop very modern
                  web application ...</td>
       </tr>
       <tr>
              <td> </td>
```

```
                <td><span>Programme:</span> <span>1.2.3</span>
                    <span>Computer science</span></td>
        </tr>
        <tr>
                <td>Project coordinator FRI</td>
                <td><div>Participants <div><ul><li>Univerza v
                    Ljubljani UL</li><li>Fakulteta za racunalnistvo
                    in informatiko</li></ul></div></div></td>
        </tr>
        <tr>
                <td>Project website:</td>
                <td><a href="www.fri.uni-lj.si">FRI</a></td>
        </tr>
</table>
</div>
</body>
</html>
```

Primer izvorne kode HTML za namišljeni projekt.

Developing search application for searching eu projects
EU SEARCH

Dates 1.1.2014 1.1.2015	• **Total cost:** 5 000 000,00 EUR • Contribution • 1 000 000,00 EUR
Description	Main goal of the project is develop very modern web application ...
	Programme: 1.2.3 Computer science
Project coordinator FRI	Participants • Univerza v Ljubljani UL • Fakulteta za računalništvo in informatiko
Project website :	FRI

Slika 3.3: Izgled prejšnjega dokumenta HTML v brskalniku

Primer, kako lahko razčlenimo in pridobimo podatke iz prejšnjega do-

kumenta HTML. Do določenih podatkov lahko pridemo na različne načine. Razčlenjevalnik povzema večino načinov, ki smo jih potrebovali za ustrezno razčlenitev in pridobitev podatkov.

```
from bs4 import BeautifulSoup
import codecs

page = codecs.open("euproject.html", 'r', "utf-8").read()
soup = BeautifulSoup(page)
project_title = soup.body.find('div', attrs={'id' : 'ptitle'}).text
print(project_title)
project_acronym = soup.body.find('div', attrs={'class' :
    'acronym'}).text
print(project_acronym)
project_start = soup.body.table.tr.div.next
print(project_start)
project_end = soup.body.table.tr.div.br.next
print(project_end)
project_total = soup.body.find('li', attrs={'class' :
    'contribution'}).parent.li.h3.nextSibling
print(project_total)
project_contribution = soup.body.find('li', attrs={'class' :
    'contribution'}).parent.find_all('li')[2].text
print(project_contribution)
project_description = soup.body.find('tr', attrs={'class' :
    'description'}).find_all('td')[1].text
print(project_description)
project_programme =
    soup.body.find_all('tr')[2].find_all('span')[1].next.next.next.text
print(project_programme)
project_participants =
    soup.body.find_all('tr')[3].find('ul').find_all('li')
print(len(project_participants))
for participant in project_participants:
        print(participant.text)
project_website =
    soup.body.find_all('tr')[4].find('a').attrs['href']
print(project_website)
```

Primer razčlenjevalnika za prej podani namišljeni primer projekta.

razčlenjevalnik	striktnost	čas (v sekundah)
html5lib	popravi nezaključeno značko	1550
lxml	ignorira nepravilno zaključene značke	972
html.parser	ignorira nepravilno zaključene značke	1108

Tabela 3.1: Primerjava hitrosti različnih razčlenjevalnikov pri razčlenjevanju 24908 datotek HTML iz vira podatkov Cordis in zapisovanju razčlenjenih podatkov v prazno podatkovno bazo. Striktnost je povzeta po [37].

Razčlenjevanje z različnimi razčlenjevalniki

Pri uporabi različnih razčlenjevalnikov smo naredili test, v katerem smo preverjali hitrost. Zanimalo nas je ali prihaja do izgube morebitnih podatkov zaradi nepravilno zaključenih značk v nekaterih datotekah in s tem do neveljavnega dokumenta. Rezultati so pokazali, da se precej razlikujejo v hitrosti (glej tabelo 3.1). Najhitreje smo vsebino razčlenili z razčlenjevalnikom lxml, sledil mu je html.parser. Najpočasnejši je bil html5lib. Glede primerjave vsebine razčlenitve smo podatke zapisovali na standardni izhod, ki smo ga preusmerili v datoteko. Po tem smo datoteke primerjali med sabo in nismo opazili kakšnih razlik v podatkih zaradi uporabe različnih razčlenjevalnikov. Zadeva bi se zelo spremenila, če bi bile kakšne druge značke nepravilno zaključene. Tako bi nastalo več napak. V tem primeru bi bilo potrebno preveriti, s čim bi dobili največ podatkov [37].

3.2.3 Razlogi za shranjevanje v datoteke

Datotek HTML projektov nam seveda ne bi bilo treba shranjevati na disk. Za to potezo smo se odločili, ker ponoven obhod robota traja več dni pri nekajsekundnem zamiku. V primeru, da bi želeli pridobiti še kakšen podatek ali pa samo malo popraviti trenutno razčlenjevanje, bi nam to čakanje vzelo veliko časa. Zaradi zamika med obiskom posamezne strani nam predstavlja branje in shranjevanje datotek bolj zanemarljiv čas v primerjavi z nekajse-

kundnim zamikom. Način shranjevanje v datoteke so tudi že uporabili za eno spletno stran evropskih projektov v članku [31] in so podatke šele nato razčlenjevali.

3.2.4 Vzporedno izvajanje

Robot zaporedoma obiskuje različne vire podatkov, s tem celoten obhod traja zelo dolgo. Zaradi tega smo predvideli možnost, da bi robota vzporedno izvajali za vsak vir podatkov posebej. To lahko naredimo tako, da klice funkcij ločimo v več datotek in vsako potem zaženemo. S tem si lahko malo skrajšamo čas, ne pa veliko, saj ima ena stran zelo veliko projektov druge pa precej manj.

3.2.5 Ponoven obhod robota

Ponoven obhod robota opazuje spremembe na strani in v primeru, da se vsebina projekta spremeni, popravi zapis v bazi. To smo realizirali na način, da glede na naslov projekta in spletne povezave, kjer se nahajajo informacije o projektu, poiščemo identifikator zapisa v podatkovni bazi, seveda če že obstaja. V primeru, da zapis že obstaja, naredimo update stavek SQL. Če naredimo v podatkovni bazi MySQL update in so vrednosti atributov enake, se stavek SQL update ne izvede [24]. V podatkovno bazo smo dodali še en atribut, ki se proži samo na spremembe in takrat vnese časovni žig. Na ta način sledimo spremembam v opisih projekta in lahko nato dobimo tudi zadnje spremenjene projekte. S tem načinom sledenja sprememb imamo težave, če bi želeli uporabiti katero drugo podatkovno bazo, saj bi morali najprej preveriti, ali to deluje enako ali ne bi morali narediti ustrezne spremembe. Seveda bi lahko izbrali možnost, da izračunamo določeno zgoščevalno funkcijo in glede na to gledamo spremembe. Druga rešitev bi bila bolj neodvisna od podatkovne baze.

3.3 Primer vključitve nove strani

Za primer vključitve nove strani sledimo prej opisanim štirim glavnim korakom robota. Najprej stran poimenujemo z imenom in jo dodamo v seznam že obstoječih strani. Nato z istim imenom naredimo še dve mapi na disku, kamor se bodo shranjevali rezultati iskanja in v drugo mapo strani posameznega vira podatkov. Sedaj naredimo novo datoteko PY, ki jo poimenujemo z imenom strani. V datoteki najprej uvozimo vse metode iz datoteke robot. Nato se lotimo prvega koraka robota.

Na viru podatkov, ki ga želimo dodati, poiščemo iskalnik projektov. Sedaj moramo ugotoviti, kako nam bo ta iskalnik (ali pa zgolj filter) vrnil kot rezultate vse projekte. Možnosti je več: imamo gumb, ki izpiše vse projekte, če kliknemo na iskanje z vsemi praznimi vrednostmi dobimo vse rezultate, možno je, da so rezultati že izpisani, iskalnik pa nam služi samo kot filter, naslednja možnost je, da imamo v izbirniku za izbor prikaza števila rezultatov možnost, da prikažemo vse. V primeru, da nismo dobili povezave do izpisa vseh projektov na eni strani, moramo dobiti prvo stran in kateri parameter se spremeni ob obisku naslednje strani rezultatov iskanja ter moramo napisati metodo, ki bo našla številko zadnje strani. Za prvi primer, ko dobimo vse rezultate ne eni strani zadostuje, da zgolj pokličemo metodo iz razreda robot, ki stran shrani. V drugem primeru pa moramo narediti še zanko, ki se sprehodi od prve strani do zadnje strani (številko zadnje strani moramo dobiti preko druge metode) tako, da povečuje številko strani za 1. V primeru, da stran ne uporablja metode GET, moramo realizirati klikanje na gumb "naprej", da pridemo čez vse strani. Na nekaterih straneh, kjer se v spletnem naslovu spreminja zgolj identifikator projekta, bi se temu lahko izognili, da pridobivamo povezave do projektov preko rezultatov iskanja. Lahko bi samo generirali vse povezave od prvega identifikatorja do zadnjega (ki bi ga pa spet vseeno morali nekako dobiti, če se stran še dopolnjuje).

Nadaljujemo z drugim korakom, kjer se z zanko sprehodimo čez vse shranjene datoteke HTML in iz njih razčlenimo povezavo do strani posameznega projekta. Sedaj si moramo ogledati izvorno kodo datoteke HTML, kjer naj-

prej poiščemo odsek, znotraj katerega so samo rezultati iskanja (to naredimo da ne bi pridobili odvečnih povezav na strani, ki ne kažejo na posamezne projekte). Sedaj z BeautifulSoup enostavno izberemo vse značke **a** in njihov atribut **href** ali pa gremo čez vse vrstice rezultatov in vzamemo značko **a** in njen atribut **href**. Sedaj moramo povezavo sestaviti v absolutno povezavo, ker po navadi so podane relativne povezave. Pridobljene povezave dodajamo v seznam, ki ga na koncu zapišemo v tekstovno datoteko. Naslednji korak robota je, da v zanki za vsako povezavo shranimo stran in nekaj sekund čakamo, odvisno od strani (pri prehitrih zahtevah nas lahko avtomatsko blokirajo).

Sledi še četrti končni korak, v katerem se v zanki sprehodimo čez vse datoteke HTML in sedaj moramo za vsak podatek, ki ga želimo dobiti, ustrezno poiskati v strukturi DOM in ga razčleniti. Pri tem nam je v veliko pomoč lahko brskalnik Firefox in dodatek Firebug [10]. S pomočjo tega dodatka se lahko po izvorni kodi z miško premikamo in pri tem se nam vizualno na strani pokaže pripadajoči del, s katerim vidimo, kaj bomo dobili, če izberemo določeni element. Določene pridobljene podatke moramo ustrezno konvertirati v ustrezno obliko. To velja predvsem za datume. Na koncu vse podatke, ki smo jih prej shranili v spremenljivke, zapišemo v podatkovno bazo. Podatke o predmetih in sodelujočih (če so ti podatki na strani) moramo zapisati v ločeni tabeli, za kar moramo najprej pridobiti identifikator vstavljenega zapisa, da ga nato dodamo v podrejeno tabelo. Zraven v tabelo dodamo še atribute, ki nam lahko koristijo, prav tako nam koristi časovni žig vstavljanja vrstice v bazo. To v zanki ponavljamo, dokler ne razčlenimo vseh datotek.

3.4 Primerjava našega namenskega robota z Apache Nutch in OpenSearchServer

Na voljo imamo številne že izdelane splošne robote, ki bi jih lahko uporabili v našem delu. Omenili bomo dva, ki sta najbolj povezana z našim delom nista pa edina, ki bi jih lahko uporabili. Seveda ima vsak robot določene prednosti

in slabosti, ki jih moramo upoštevati glede na to, za kakšen problem jih bomo rabili. Za naš problem so tako za izbiro robota pomembne naslednje funkcije: povezava z Apache Solr (opisan je v 5.1.1), povezava s podatkovno bazo MySQL, izvedba JavaScripta, omejitev samo na vir podatkov projektov in kako je s pridobivanjem posameznih projektnih atributov.

3.4.1 Apache Nutch

Apache Nutch [2] ima glede JavaScripta omejeno podporo samo za iskanje povezav, če je vsebina podana preko JavaScripta, robot te vsebine ne bo pridobil. Uporablja se samo za zbiranje in shranjevanje strani. Za samo indeksiranje imamo na voljo več možnosti. Za indeksiranje se tako lahko uporablja Solr. Ena od možnosti je tudi shranjevanje v podatkovno bazo MySQL. Narejen je za uporabo s Hadoopom v primeru, da bi imeli ogromno podatkov [2, 14]. Nam bi ustrezalo indeksiranje z Apache Solr, ampak ne bi imeli podatkov ustrezno razčlenjenih. Za ustrezno razčlenitev bi morali narediti dodatek, ki bi ga morali spisati po ustrezni strukturi v Javi tako, da bi ustrezno dedovali in razširili obstoječe razrede. V sami kodi dodatka bisprogramirali razčlenjevanje posamezih atributov, ki bi se potem indeksirali. Tukaj pa se pojavi težava, da Nutch ne shranjuje izvorne kode, ampak uporablja binarni format. Če bi želeli na isti način razčlenjevati podatke iz izvorne kode, bi morali najprej poskrbeti za to, da bi robot shranjeval izvorno kodo HTML. To je seveda možno realizirati, ampak bi porabili več časa.

3.4.2 OpenSearchServer

OpenSearchServer [22] ima možnost preko Seleniuma uporabiti Firefox ali PhantomJS, in s tem tudi izvede kodo JavaScript na spletni strani. Če strani uporabljajo JavaScript, je OpenSearchServer vsekakor boljša izbira kot prej opisani Apache Nutch. Poleg tega pa ni samo spletni robot, ampak lahko indeksira tudi datoteke na datotečnem sistemu, vire REST in podatkovne baze preko JDBC.

3.4.3 Uporabnost različnih obstoječih robotov za naš problem

Za naš primer nobeden od opisanih robotov ali kateri drugi robot ni najbolj uporaben. Čeprav bi seveda lahko prej opisana robota (Apache Nutch, OpenSearchServer) uporabili pri našem problemu. V primeru, da bi jih vseeno uporabili, bi morali najprej poskrbeti, da bi sledila samo določenim povezavam na teh virih podatkov in tako ne bi obiskovala nepotrebnih strani na strani. Druga, še večja težava, je problem je razčlenjevanje podatkov, ki jih o projektu potrebujemo za napredno iskanje po posameznih atributih. Za ta problem bi seveda lahko napisali dodatek, ki bi razčlenil izvorno kodo in bi pridobil posamezne atribute. Če tega ne bi naredili ne bi mogli realizirati naprednega iskanja po posameznih atributih. Pisanje dodatkov je seveda možno in sistemi to že omogočajo, ampak bi nam to vzelo veliko več časa, ker je treba podrobneje spoznati celoten sistem npr. Apache Nutch ali pa katerikoli drugega. Če bi imeli samo splošno iskanje in ne iskanja po posameznih atributih, bi bila bolj smiselna uporaba katerega od omenjenih robotov ali kakšnega drugega, ker se z razčlenjevanjem posameznih podatkov sploh ne bi ukvarjali. Oba robota se razlikujeta tudi v tem, da ne hranita neposredno celotne izvorne kode posamezne strani, kot to dela naš robot. V binarnem formatu (Apache Nutch) in lastnem indeksu (OpenSearchServer) hranita tako zgolj določene meta podatke in tekst, ki ga pridobita ostale podatke pa zavržeta.

Ima pa naš robot pomankljivost, ki se je pokazala tudi med izdelavo diplomskega dela. V primeru spremembe strukture DOM na posamezni strani robot ne bo mogel razčleniti vseh podatkov, ker določenega elementa ne bo več ali pa se bo spremenilo ime atributa elementa. Lahko se seveda tudi zgodi, da ne bo dobil nobenega podatka iz te spletne strani. Kot že omenjeno se je to zgodilo tudi nam, ker so eno spletno stran malo spremenili in preimenovali določene atribute elementov. Če bi spletne strani zelo pogosto spreminjali, naš robot zaradi prej opisanega ni učinkovit. Ravno tako bi se v primeru, da bi naš robot shranjeval veliko večje število različnih spletnih strani, to še

precej pogosteje dogajalo zaradi morebitnih prenov spletnih strani. V tem primeru bi morali uporabiti splošnega robota in se lotiti problema na način, ki smo ga prej opisali in so ga uporabili v članku [25].

3.5 Slabosti in pomankljivosti

Naš robot je zelo ozko namensko usmerjen in zbira podatke, ki se nahajajo na točno določenih mestih v strukturi DOM. Za takšno rešitev smo se odločili, ker smo imeli malo spletnih strani, potrebnih za razčlenitev. V primeru, da bi bilo teh strani veliko več, in vseh sploh ne bi poznali, ker bi jih robot sam iskal pa naš pristop ne bi bil uporaben. V primerjavi s problemom, ki so ga imeli v članku [25], ko so iskali po vseh, ki so pripadali tisti državi, naše rešitev ni uporabna, saj imamo v tem primeru preveč strani, da bi sploh lahko za vsako posebej napisali razčlenjevalnik. Če bi se vseeno lotili takšnega načina, kot je opisan, bi morali narediti nekakšen splošen razčlenjevalnik, ki bi iz množice strani dovolj dobro dobil čim več podatkov, ki jih potrebujemo. S tem pristopom bi naleteli na težave, da bi pri pridobljenih podatkih imeli še kakšne dodatne informacije, potem iz določenih spletnih strani recimo ne bi uspeli nobenega podatka ustrezno razčleniti. Tudi rezultati ne bi bili dovolj natančni. Tukaj bi seveda morali uporabiti čisto drugačne metode, kako pridobiti ustrezne podatke. Vse to pa je seveda možno do neke meje. Lahko bi se pa odločili za vmesno rešitev, kjer ne bi imeli naprednega iskanja po posameznih atributih in bi iskali zgolj po celotnem tekstu, kar bi bila čisto zadovoljiva rešitev. Na tak način po celotnem tekstu deluje tudi naše osnovno iskanje in mehko iskanje z uporabo atributa all_text, ki ima shranjen celoten odsek projekta z vsemi podatki.

3.5.1 Enotna strukturiranost podatkov

Na spletnih straneh, ki smo jih mi razčlenjevali za naš iskalnik, smo pogrešali predvsem to, da ni nekakšne skupne strukture teh podatkov in da so, zelo različni ter (glej primer kode HTML 3.2.2, ki ne pokaže najboljše

strukture). Mi bi si predvsem želeli, da bi bile informacije na vseh straneh na primer v formatu XML. Poleg tega bi bilo zelo zaželeno, da bi imele vse strani uporabljale enako shemo podatkov, le da bi se razlikovale po številu teh podatkov. V tem primeru nam ne bi bilo potrebno za vsako stran delati svojega, razčlenjevalnika, ampak bi imeli samo enega. Še boljše kot sam format XML bi bilo, da bi spletne strani imele predstavljene podatke v tehnologijah semantičnega spleta in na vseh straneh z uporabo enake tehnologije. Na internetu najdemo bazo evropskih projektov, predstavljeno s tehnologijami semantičnega spleta, ki vsebuje več kot 500 projektov [7]. Baza je zgrajena s formatom RDF (Resource Description Framework), ki je nadgradnja XML-ja za semantični splet. Po bazi lahko poizvedujemo z jezikom SPARQL, ki je podoben SQLu, le da je namenjen poizvedovanju po podatki predstavljenih v obliki RDF. Za samo bazo nimamo ustreznega grafičnega vmesnika s katerim bi lahko iskali po podatkih, ampak moramo napisati poizvedbo v jeziku SPARQL ali uporabiti kakšnega odjemalca. Seveda bi tudi mi lahko imeli podatke, predstavljene v obliki RDF in bi naš iskalnik zgolj naredil ustrezno poizvedbo. Uporabnik tako niti ne bi videl, da uporabljamo RDF. Pri tej rešitvi je nastane težava, da uporabljena iskalna platforma tega nima na izbiro, čeprav zna indeksirati tudi podatke, ki so podani v obliki RDF [4].

3.5.2 Težave z atributi

Kot smo že omenili, velikokrat se zgodi, da ima posamezna spletna stran določene atribute, zato pa je od posameznega projekta odvisno, ali pri določenem projektu podatek je ali ga ni. Če pogledamo za vse spletne strani, se za atribut project_url izkaže, da 80 odstotkov projektov nima svoje spletne strani ali pa podatek ni naveden. Seveda je to razumljivo, da se ravno za vsak projekt ne naredi svojo spletno stran. Bolj problematično je, da projekti imajo svojo spletno stran, ampak na spletnih straneh ni navedene povezave. Izpostaviti moramo, da tudi med 20 odstotki projektov, ki imajo navedeno spletno povezavo velikokrat ni veljavna. Kar nekaj povezav do spletnih strani je zatipkanih ali imajo dvakrat HTTP na začetku ipd. Prisotnost podatkov

drugih atributov se seveda tudi zelo razlikuje in je zelo odvisna od atributa.

Baze evropskih projektov so bile že večkrat analizirane po podatkih in narejeneso bile razne raziskave. Na to temo obstajajo številni članki, ki predstavljajo pri koliko projektih se pojavi določeni partner ali razne druge stvari [29, 36, 31]. To je bil tudi razlog, da smo se odločili za shranjevanje v podatkovno bazo MySQL in ne direktno v Solr (opisan je v 5.1.1), v primeru, da se bo delalo razne analize iz teh podatkov. Seveda za določene informacije bi bilo potrebno podatke še malo obdelati, kot smo že omenili težavo z denarjem, ki je shranjen kot tekst. Iz teh podatkov bi na primer lahko raziskovali, koliko je skupen znesek vseh projektov v določenem obdobju, v katerem sodeluje vsaj en iz določene države.

Za Solr imamo na voljo tudi različne odjemalce (tudi za Python). V primeru, da ne bi uporabljali podatkovne baze MySQL, bi podatke vstavljali direktno v Solr. Razlog, da smo shranili podatke v podatkovno bazo MySQL je tudi ta, da če bi Solr v prihodnosti zamenjali s kakšno alternativno možnostjo. Tako bi potem samo ustrezno uvozili podatke ali pa zgolj uporabljali podatkovno bazo MySQL. V vsakem primeru, bi morali ustrezno predelati uporabniški vmesnik.

Poglavje 4

Arhitektura celotnega sistema

Celotni sistem je sestavljen iz dveh glavnih delov: spletnega robota in spletnega iskalnika. Prvi del sistema skrbi za pridobivanje in shranjevanje podatkov v podatkovno bazo MySQL. Drugi del, to je sistema spletni iskalnik pa je sestavljen iz spletne aplikacije, ki preko storitev REST komunicira z iskalno platformo Apache Solr. Med podatkovno bazo MySQL in platformo za iskanje se naredi uvoz podatkov v platformo za iskanje.

4.1 Zgradba podatkovnega modela

Podatkovni model (slika 4.1) smo zasnovali po viru podatkov z največ atributi, ker so ostale strani so imele zelo podobne atribute le v primerjavi z največjim virom podatkov precej manj. Drugi viri podatkov so imeli kakšne dodatne atribute na primer več tekstovnih opisov, kot so razni rezultati, z opisanim statusom, aktivnostmi, to smo pa zajeli zgolj v okviru celotnega teksta. Tako smo od vsakega projekta vzeli naslov (atribut poimenovan title), ki je bil prisoten v vseh virih podatkov. Potem smo nadaljevali z začetnim (atribut start) in končnim (atribut end) datumom projekta. Z datumi smo imeli kar nekaj težav, ker datum na dveh straneh ni bil napisan v enakem formatu, tako da smo morali datume ustrezno konvertirati, da so ustrezali podatkovnemu tipu date. Pojavile so se tudi težave, da so bili datumi zapisani

samo kot mesec in leto, v tem primeru smo kot dan vzeli prvi dan meseca. Imeli smo tudi primere, kjer so bile zapisane le letnice. V tem primeru smo vzeli kot dan in mesec 1. 1., da smo dobili ustrezen format.

Nadaljevali smo s kratkim (atribut short_desc) in dolgim opisom (atribut long_desc) projekta, ki je prinesel tudi nekaj dilem, saj je bila vsaka stran malo drugačna. Nekatere strani so imele oba opisa, ena stran je imela še nekakšen vmesni opis. Še večja težava je nastala, ker je bil opis samo en in kratek, v tem primeru smo ga vnesli pod atribut short_desc, če pa je imela spletna stran daljše opise, smo jih vnesli pod atribut (long_desc). Na ta način smo dobili za vse spletne strani v bazi približno enake podatke. Naprej sledita atributa referenčna koda (atribut reference_code) in status. Naslednja dva atributa, skupni znesek (atribut total) in eu prispevek (atribut contribution), sta bila dana pod podatkovni tip tekst, ker so bili na eni strani zneski določeni v obliki od vrednosti do vrednosti. Če bi želeli kot podatkovni tip število, potem bi morali iz ene strani vzeti ali od vrednosti ali pa do vrednosti. Res pa je, da če bi želeli vsote seštevati ali pa delati poizvedbe po številkah, potem bi morali izbrati število, ker pa tega ne potrebujemo, smo se odločili za tekst. Sledita atributa akronim projekta in contract, ki sta bila odvisna od posamezne strani.

Potem imamo tri atribute za glavnega koordinatorja: to so ime (atribut coordinator), država (atribut coordinator_country) in celoten naslov (atribut coordinator_address). Podatki o sodelujočih na projektu so v ločeni tabeli. Sledita atributa številka zapisa (atribut record_number) in datum spremembe (atribut record_updated), če je objavljen na sami strani. Atribut url predstavlja stran, s katere smo shranili sam projekt in služi, da lahko kasneje obiščemo stran z vsemi podrobnimi informacijami, ki jih v iskalniku ne predstavimo medtem ko atribut project_url hrani povezavo na spletno stran projekta. Naslednji atribut (database_url) je povezava do same baze projektov. Atribut, ki predstavlja ime baze, je namenjen, da vemo, v kateri bazi je projekt in ga potem tudi lahko filtriramo.

Določenih podatkov nismo mogli ustrezno pridobiti ali so bili manj po-

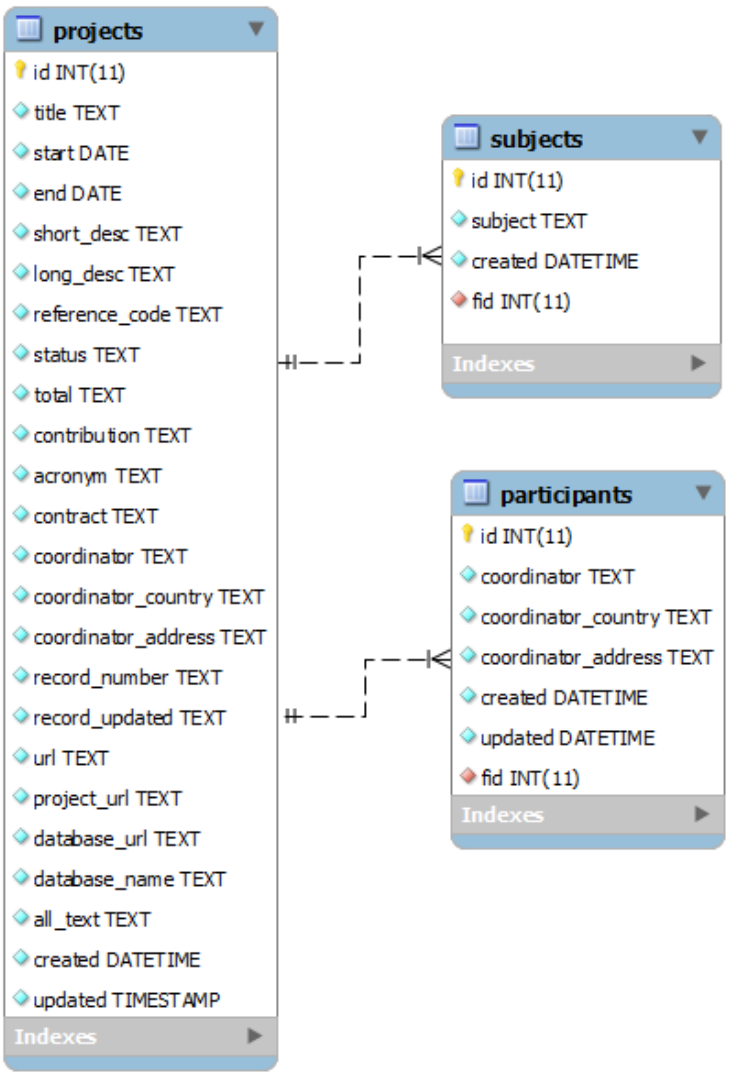
membni. Ker smo vseeno želeli shraniti še vse te podatke, smo naredili atribut all_text, ki ima shranjen tekst celotnega odseka, kjer se nahajajo podatki o posameznem projektu, vključno z že zajetimi podatki. Zadnje dva atributa created in updated, nam povesta kdaj je bil zapis vstavljen v tabelo in updated, ki se proži ob spremembah in vstavi časovni žig, kot smo že prej opisali njegovo delovanje. Vse dosedaj opisane atribute imamo v eni tabeli projects. Poleg te tabele imamo še dve podrejeni tabeli za podatke, ki jih ima lahko vsak projekt več ali pa tudi nobenega. Gre za sodelujoče na projektu in predmete projekta. Pri predmetih projekta imamo tako atribut subject. V tabeli participants pa imamo tri enake atribute za sodelujoče kot v glavni tabeli, kjer imamo glavnega koordinatorja.

Na samem začetku še nismo točno vedeli, kakšno bo število znakov posameznega atributa, ker še nismo imeli vseh virov podatkov. To je bil tudi razlog, da smo izbrali podatkovni tip text, ki seveda ni najbolj optimalen. Potem, ko smo že imeli vse podatke iz vseh virov v podatkovni bazi, bi lahko pogledali, kolikšna je maksimalna dolžina znakov za posamezen atribut. Na ta način bi zoptimizirali podatkovne tipe za vse tekstovne atribute. Zaradi tega, ker pa ne uporabljamo podatkovne baze MySQL za neposredno iskanje, se s tem nismo ukvarjali.

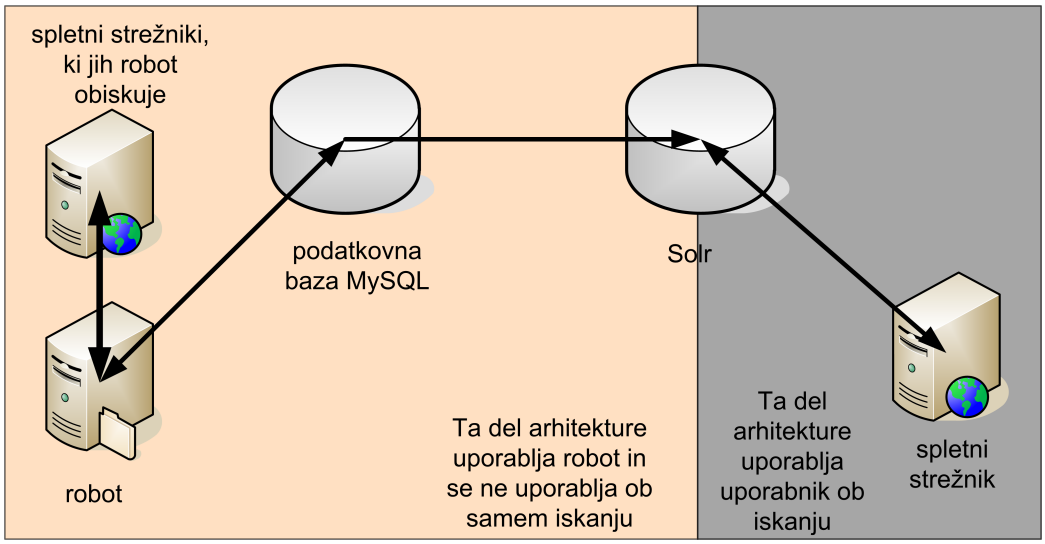
4.2 Arhitektura našega sistema

Celoten naš sistem je sestavljen iz robota in podatkovne baze MySQL, ki sestavljata prvi del našega sistema. Drugi del sistema je sestavljen iz aplikacijskega strežnika Jetty za Apache Solr, spletnega strežnika Apache in same spletne strani, ki je sestavljena iz ene datoteke html in nekaj ostalih datotek z JavaScriptom ter stilom CSS. Mejo med deloma predstavlja Solr, ki je v obeh delih, saj uvozi podatke iz podatkovne baze.

Vse skupaj lahko teče na enem računalniku ali pa sistem razdelimo na več računalnikov, odvisno od naših potreb. V sistemu ne rabimo nujno spletnega strežnika, če delamo samo na enem računalniku, ker v tem primeru lahko



Slika 4.1: Shema podatkovnega modela baze evropskih projektov



Slika 4.2: Arhitektura sistema, ki lahko teče v celoti na enem računalniku ali pa je razdeljen na več računalnikov

samo odpremo datoteko HTML v brskalniku. Vse poizvedbe se delajo preko spletne povezave na storitve REST od Solr, ki tudi vrača rezultate v formatu JSON. Podatkovna baza MySQL se rabi samo za shranjevanje razčlenjenih podatkov in potem uvaža le-te v Solr. Že prej smo omenili, da podatkovne baze ne bi nujno potrebovali, ker bi podatke lahko vstavljali direktno v Solr. Celoten sistem lahko razporedimo tudi tako, da imamo na enem računalniku robota, na drugem podatkovni strežnik, na tretjem aplikacijski strežnik (za Solr) in na četrtem spletni strežnik. Seveda lahko tudi kakorkoli drugače stvari skombiniramo po različnih računalnikih. Celoten sistem je pripravljen, tako da bi lahko sprejemal velike količine prometa, saj spletni strežnik samo enkrat vse servira, potem se vse dogaja na strani odjemalca. Solr tudi nima težav z velikimi količinami prometa, le drugače mora biti postavljen (5.1.1). Naš sistem ne bo uporabljen za take namene, zato se ni potrebno bati preobremenitve.

Poglavje 5

Iskalnik

Drugi del našega dela je razvoj iskalnika po zgrajeni bazi evropskih projektov, ki smo jo naredili v prvem delu. Naš iskalnik je ozko usmerjen po specifični domeni (vertical search), za razliko od splošnih iskalnikov, ki iščejo po vsem [20, 25]. Naš cilj je bil, da izdelamo aplikacijo, ki bo v celoti samo ena stran (single page application) in bo uporabljala standard HTML5. Za vse ostalo pa bo poskrbel JavaScript na strani odjemalca, podatke pa se bo pridobivalo preko storitev REST. Na strani strežnika se tako ne izvaja nič drugega, razen Solr, ki odgovarja na zahteve.

5.1 Tehnologije

5.1.1 Lucene

Lucene je knjižnica, ki indeksira dokumente kateri so sestavljeni iz več polj. Za vsako polje vzdržuje obratni indeks [11], ki je najbolj popularna podatkovna struktura v sistemih za pridobivanje dokumentov. Deluje tako, da hrani pojme in nato za vsak pojem hrani identifikatorje dokumentov, v katerih se pojavi pojem. Obratni indeks pri Lucene tako vzdržuje za vsak pojem posting list, ki je sestavljen iz unikatnih številk dokumentov, ki vsebujejo ta pojem. Zraven vzdržuje tudi, kje v tekstu se pojavi ta pojem, lahko pa shranjuje tudi še kakšne dodatne informacije zraven (payload). Posting element

tako sestavljajo trojke, ki so sestavljene iz teh treh prej opisanih parametrov. Ob samem iskanju se tako uporabi posting list preko katerega se hitro iterira in dobi ujemajoče dokumente in se za vsak zadetek izračuna še oceno relevantnosti rezultata [11]. Na koncu dobimo kot rezultat najbolj relevantne dokumente. Z uporabo fasetnega iskanja se pa pri ujemajočih dokumentih naredi še dodatno procesiranje [26, 28]. Sama knjižnica nudi iskanje in še dodatne stvari, kot so popravljanje teksta, označevanje zadetkov, napredne analize teksta in razbijanje teksta. Sposobna je pridobiti tekst iz različnih formatov, (kot so PDF, HTML, Microsoft Word ...) [13]. Na tej knjižnici temeljita tako Apache Solr kot tudi Elastic search [23], ki jo uporabljata kot jedro sistema. V primeru, da ne uporabimo nobenega od njiju, moramo še veliko stvari sprogramirat, da lahko naredimo iskalnik. Lahko pa seveda knjižnico uporabimo za del kakšnega projekta, če ne potrebujemo možnosti, ki nam jih ponuja Solr in imamo kakšen lasten iskalni sistem.

5.1.2 Apache Solr

Apache Solr uporablja knjižnico Lucene in je platforma za iskanje, napisana v Javi. Poleg navadnega iskanja omogoča številne funkcije. To so samodokončaj, označevanje zadetkov v tekstu, fasetno iskanje, gručenje in grupiranje rezultatov. Razne načine iskanja, kot so mehko iskanje, bližnje iskanje, lokacijsko iskanje. Poleg naštetih funkcij pa omogoča tudi integracijo s podatkovno bazo. Ravno tako lahko z njim indeksiramo dokumente v različnih formatih (PDF, Word, HTML ...) [3]. Teče na aplikacijskem strežniku Jetty, lahko pa uporabimo kateri drug aplikacijski strežnik (Tomcat, GlassFish ...). Narejen je tako, da lahko indeksira ogromne količine podatkov in odgovarja na zahteve. V takem primeru se ga uporablja skupaj s Hadoopom. Vse operacije potekajo preko spletnih povezav, v smislu storitev REST, čeprav niso čisto to, ker gre bolj za spletni naslov z različnimi parametri. Operacije lahko opravljamo v štirih formatih (XML, JSON, CSV, binary data) in rezultate dobimo tudi v enem od teh formatov. Ima tudi administracijsko spletno konzolo. Za Solr so napisani številni odjemalci za različne programske jezike.

Uporablja tudi transakcije, s katerimi lahko naredimo razveljavitev oziroma potrditev transakcije. Privzeto se nastavi, da se transakcije same potrjujejo. Uporabljamo jih ob samem uvozu, vstavljanju in brisanju dokumentov ter jih avtomatično potrjujemo ali pa to storimo šele na koncu. S tem lahko dosežemo, da iskalnik ne vrača rezultatov, predno niso uvoženi vsi podatki, ki jih na koncu potrdimo.

Solr vse podatke hrani v eni sami veliki tabeli, posamezen atribut (namesto izraza atribut Solr uporablja izraz polje) pa ima lahko tudi več vrednosti, če tako definiramo v shemi. Če podatke uvažamo iz relacijske podatkovne baze, potem tu že naletimo na oviro, ker moramo podatke spravit v eno tabelo, čeprav vseeno ponuja možnost relacije straš otrok med dokumenti, ampak so potem tudi poizvedbe drugačne. Z eno tabelo se izognemo kompleksnim stičnim operacijam, ki po navadi tudi dolgo trajajo in jih tipično uporabljamo v klasičnih podatkovnih bazah za iskanje. To naredimo glede na to kako bomo pri iskanju posamezne podatke potrebovali za potrebe fasetnega iskanja. Kljub temu da Solr shranjuje podatke v tabeli, ga ne moremo uporabljati namesto kakšne druge podatkovne baze.

Za same atribute lahko uporabljamo različne filtre, s katerimi izboljšamo iskanje. Uporablja se predvsem stop words filter, s katerim izločimo najpogostejše besede v jeziku ali podatkih, ki nas potem ovirajo. S filtrom za sinonime lahko omogočimo, da za določeno besedo dodamo več sinonimov in potem pri samem iskanju dobimo rezultate, tudi če se iskana beseda ne pojavi nikjer v tekstu, v vsakem primeru pa se mora pojaviti njen sinonim. Slabost sinonimov je ta, da jih je treba ročno vnesti, da dobimo želene rezultate. Izogibanja težav z velikimi in malimi črkami se lotimo s filtrom, ki vse spremeni v male črke. Potem moramo omeniti še tokenizacijo besed, ki v osnovi deluje glede na presledke in razbije tekst po besedah. Na izbiro imamo več načinov. Omeniti moramo še stemming, ki iz posameznih besed dobi nek osnoven koren besede. Vse opisane načine lahko uporabimo tudi za vsak jezik posebaj, ker ima vsak jezik svoje specifičnosti. Lahko pa tudi opisane načine spustimo in ne delamo nobenih operacij nad tekstom [4].

Shema baze evropskih projektov v Apache Solr

V Solr smo morali pripraviti malo drugačno shemo, saj imamo tukaj vse podatke v eni sami tabeli. Za atribute smo nastavili, da imajo lahko več vrednosti, zato smo tako v eno polje spravili vse pripadajoče vrednosti iz podrejenih tabel. To smo naredili za predmete in sodelujoče na projektu, pri čemer smo sodelujoče združili z glavnim koordinatorjem v tabeli. Uporabili smo tudi funkcijo copyField s katero smo naredili še en atribut, ki vsebuje večino vsebine atributov brez celotnega teksta, če bi ga potrebovali za samo iskanje, da nimamo preveč odvečnih besed, ki se pojavijo v atributu celotnega teksta.

Za atributa predmeti in ime baze, ki jih uporabljamo za fasetno iskanje, smo morali nastaviti podatkovni tip, na katerem ne razbijamo besed in v ničesar ne spreminjamo same vrednosti atributa, saj v nasprotnem primeru ne bi ohranili celotnega niza. Atribute z datumi smo definirali kot podatkovni tip datum. Pri vseh ostalih tekstovnih atributih smo uporabili standardne filtre, kot so: sprememba vseh znakov v male črke in odstranjevanje najpogostejših besed, kot so razni vezniki za angleški jezik. S spremembo vseh znakov v male črke dosežemo, da iskalnik ni občutljiv na velikost črk. V primeru, da potrebujemo razlikovanje malih in velikih črk, bi morali ta filter odstraniti. Seveda bi za vsak posamezen atribut lahko določili različne filtre, ki še ustrezno obdelajo tekst. Za primer lahko vzamemo odstranjevanje določenih besed, ki jih prej moramo ročno vnesti. Za posamezne atribute bi lahko uporabili tudi še sinonime za posamezne besede, ki bi jih morali ročno definirati. Ravno tako bi lahko uporabili še različne filtre za luščenje korena beseda. K sreči imamo vse podatke v angleškem jeziku tako, da z uporabo privzetih filtrov ni težav. V primeru, da bi imeli podatke v slovenskem jeziku, bi to prineslo kar nekaj težav. Vse filtre bi bilo potrebno ustrezno modificirati za potrebe slovenskega jezika. Z uporabo katerih drugih svetovnih jezikov teh težav ne bi bilo, ker so že vključeni v Solr.

```
<field name="title" type="text_general" indexed="true"
    stored="true" multiValued="false"/>
```

```
<field name="subject" type="string" indexed="true" stored="true"
    multiValued="true"/>
<field name="url" type="text_general" indexed="true"
    stored="true"/>
<field name="start" type="date" indexed="true" stored="true"
    multiValued="false"/>
<field name="end" type="date" indexed="true" stored="true"
    multiValued="true"/>
<field name="shortdesc" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="longdesc" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="referencecode" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="status" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="total" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="contribution" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="acronym" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="contract" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="coordinator" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="coordinatorcountry" type="text_general"
    indexed="true" stored="true" multiValued="true"/>
<field name="coordinatoraddress" type="text_general"
    indexed="true" stored="true" multiValued="true"/>
<field name="recordnumber" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="recordupdated" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="projecturl" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="databaseurl" type="string" indexed="true"
    stored="true" multiValued="false"/>
<field name="databasename" type="string" indexed="true"
    stored="true" multiValued="false"/>
<field name="alltext" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="created" type="date" indexed="true" stored="true"
```

```
    multiValued="false"/>
<field name="updated" type="date" indexed="true" stored="true"
    multiValued="false"/>
<field name="longdescgoogle" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
<field name="description" type="text_general" indexed="true"
    stored="true" multiValued="true"/>
```

Definirana shema posameznih atributov v Apache Solr.

5.1.3 Primerjava Apache Solr z Elastic search

Oba temeljita na knjižnici Lucene in tako imata v splošnem na voljo zelo podobne funkcionalnosti. Za potrebe našega iskalnika bi z obema lahko realizirali isto, ne bi bilo nobenih težav. Razlikujeta se predvsem v podrobnostih. Glede na to, kaj bomo potrebovali se odločimo, katerega bomo izbrali. Med sami verzijami se zgodi da kar ima en v eni verziji, ima potem v naslednji verziji še drugi in obratno glede na to kar smo prebrali na raznih straneh [5].

Omeniti moramo še Sphinx [19], ki je ravno tako kot Solr in Elastic search platforma za iskanje. Ravno tako lahko podatke indeksira tudi iz različnih podatkovnih baz tudi NoSQL. Ponuja vse možnosti iskanja, ki jih poznamo tudi iz drugih baz. Mogoče omeniti to, da lahko na samih rezultatih delamo agregatne funkcije, ki jih poznamo iz SQLa. Vendar sta si Solr in Elastic search bolj podobna, čeprav s Sphinx verjetno ravno tako lahko realiziramo vse stvari kot s tema dvema, edino kakšno stvar je treba še sprogramirati [19].

5.2 Implementacija

5.2.1 Preprosto iskanje (simple search)

Preprosto iskanje išče po celotnem tekstu projekta in ne po posameznih atributih, kot to omogočamo v naprednem iskanju. Tekst celotnega projekta zajema tudi takšne podatke, ki jih nismo mogli ustrezno razčleniti. Ome-

njeni podatki tako niso prisotni v nobenem drugem atributu. V preprostem iskanju lahko uporabljamo različne operatorje (glej 6.3.2) in ravno tako lahko z ustreznim iskalnim nizom iščemo samo po določenem atributu.

5.2.2 Mehko iskanje (fuzzy search)

Mehko iskanje nam išče podobne nize, ki so zelo podobni našemu iskanemu nizu, vendar ima lahko v samem nizu tudi znake, ki jih iskani niz nima. S tem lahko dobimo rezultate, tudi če se v iskanem nizu malo zatipkamo, ker v splošnem iskanju v takem primeru ne bi dobili nobenega rezultata. Solr uporablja za samo mehko iskanje razdaljo imenovano Damerau–Levenshtein distance [8]. Ta razdalja nam vrne oddaljenost dveh nizov podanega s štetjem števila operacij, ki jih potrebujemo za preoblikovanje enega niza v drugega. Te operacijo so vstavljanje posameznega znaka, brisanje določenega znaka in zamenjava znaka v nizu [4, 8, 12]. Če vzamemo za primer besedo projekt:

vstavljanje: projektA

brisanje: projek

zamenjava: prIjekt

Četrta operacija pa je transpozicija dveh sosednjih znakov. Po četrti operaciji se razdalja Damerau–Levenshtein razlikuje od osnovne razdalje Levenshtein, ki uporablja zgolj prve tri omenjene operacije. S tem operacijami naj bi po trditvah odpravili okoli 80 % tipkarskih napak [8], ki jih naredi človek. Seveda ni nujno, da te operacije delamo na samo enem znaku, ampak lahko to delamo na več znakih na primer vstavljanje dveh znakov v niz. Prvotno so bile te razdalje mišljenje predvsem za odpravljanje tipkarskih napak, da bi izboljšale popravljalnike teksta [8].

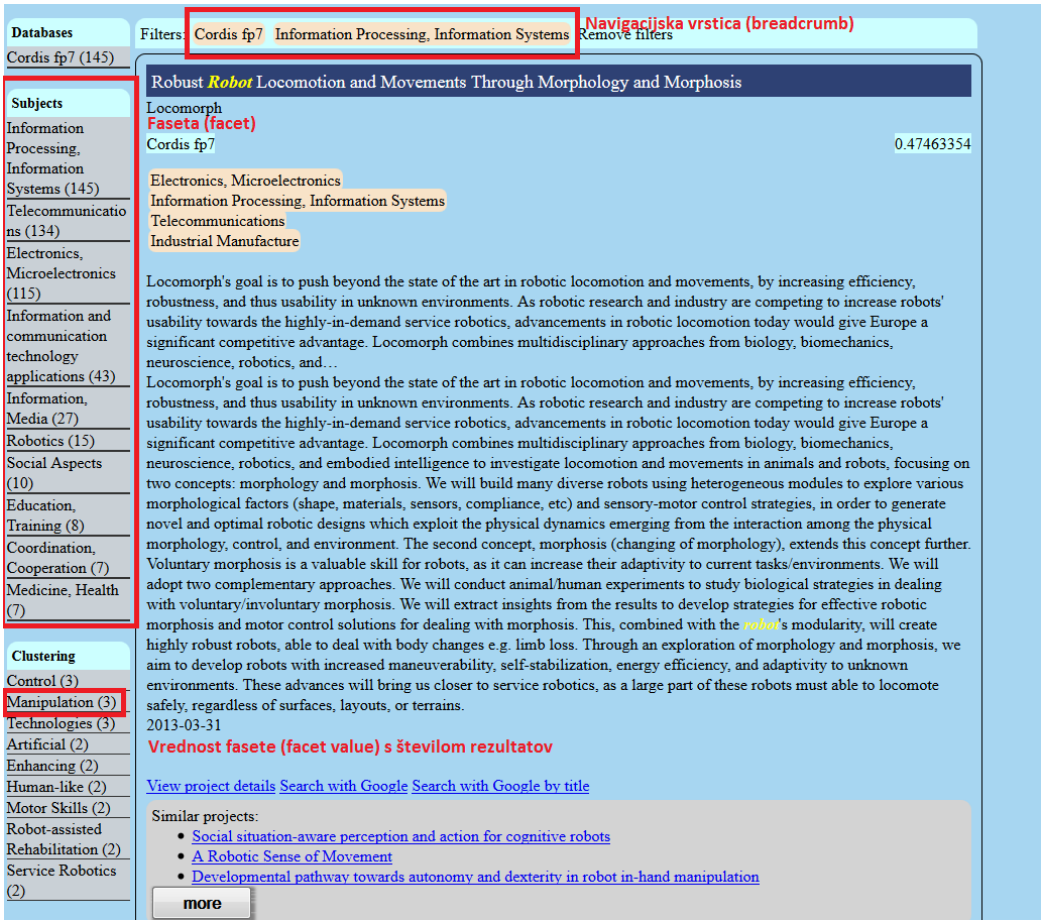
V Solr mehko iskanje uporabimo tako, da na koncu iskane besede dodamo operator ~, ki privzeto uporablja do 2 največ operaciji, da pridemo do drugega niza. Lahko pa podamo zraven operatorja še število operacij na način beseda~3. V starejših verzijah Solr se je uporabljala vrednost med 0.0 in 1.0 [4].

5.2.3 Fasetno iskanje (faceted search)

V slovarju informatike za fasetno iskanje najdemo več izrazov: usmerjeno iskanje, navigacijsko iskanje in brušeno iskanje, ki se v angleščini imenuje faceted search (imenovan tudi faceted navigation, faceted browsing) [9]. Odločili smo se da bomo uporabljali izraz fasetno iskanje, ker je najbolj podoben angleški besedi. Fasetno iskanje smo zagotovo že srečali predvsem v spletnih trgovinah, kjer filtriramo izdelke po znamki ali od do določene cene. Tipično imamo zraven prikazano tudi število rezultatov v vsaki posamezni kategoriji. Za samo filtriranje lahko uporabimo več filtrov. Vsak posamezni sklop filtrov (npr. filtriranje po ceni, kjer imamo od 0 do 100, od 100 do 250 in od 250 do 500) se imenuje faseta, medtem ko posamezne cene iz primera predstavljajo vrednost fasete ali omejitve (glej sliko 5.1). Same fasete naredimo glede na ustrezne atribute, ki jih imamo, kar pa je odvisno od vsakega posameznega primera. Zraven sodi še navigacijska vrstica, kjer imamo prikazane uporabljene filtre in jih lahko odstranjujemo. S samim fasetnim iskanjem ponudimo uporabnikom tudi nove možnosti, kako lahko pregledujejo rezultate in delno tudi primerjajo stvari.

Implementacija fasetnega iskanja

Da lahko naredimo fasetno iskanje, moramo imeti ustrezno pripravljene podatke in dovolj atributov, po katerih bodo narejene fasete (kar smo opisali že pripravi shema za Apache Solr 5.1.2). Priprava atributov nam lahko vzame kar nekaj časa in dela, ki mora tudi vplivati na odločitev, ali bomo fasetno iskanje sploh uporabili. Odločiti se moramo, ali bomo imeli hierarhično ali ploščato strukturo fasetnega iskanja (Solr podpira obe možnosti, čeprav je s hierarhičnim načinom več dela). Pri ploščatem načinu posamezna vrednost fasete še nima podrejenih vrednosti. Hierarhični način pa lahko ima za posamezno fasetno vrednost še celo drevo fasetnih vrednosti, ki so v več nivojih in se po navadi prikažejo šele ob kliku. Naslednja težava je omejevanje filtrov, če bi se jih pokazalo preveč, moramo za vsako faseto imeti tudi omejitev števila filtrov in jih potem ob kliku na več prikazati. Za dobro delovanje samega



Slika 5.1: Slika prikazuje, kaj je faseta, fasetna vrednost in navigacijsko vrstico

fasetnega iskanja je zelo pomembno, da imam vsaka vrednost v bazi podane vse vrednosti, ker v nasprotnem primeru hitro pridemo do tega, da ob samem iskanju ne bo dovolj filtrov ali pa bo samo en prazen, če ga prikazujemo. S to težavo smo se srečali tudi mi, ker določene podatkovne baze niso imele navedenih predmetov projekta, po takih projektih je bila naša faseta prazna. Druga težava manjkajočih vrednosti pa se lahko pojavi pri preseku več faset, kjer potem tisti rezultati z manjkajočimi atributi izpadejo.

Poznamo tako dinamične kot statične fasete. Statične so že prej definirani v bazi, medtem ko se dinamične stalno spreminjajo. Uporabili smo tako statične kot dinamične fasete. Dinamične fasete uporabljamo v kombinaciji z gručenjem, ki se naredijo glede na vrnjene rezultate, in tako sploh niso shranjeni nikjer v bazi. Pri uporabi več faset se soočimo tudi s problemi številčenja in preseka, ker lahko en rezultat pade v več faset in je tako večkrat štet pri različnih fasetah [9, 4].

Razlika med filtri in fasetnim iskanjem

Samim fasetnim vrednostim lahko rečemo tudi filtri, ker se ob kliku na vrednost podatki sfiltrirajo. Tu pa se pojavi vprašanje, kakšna je razlika med filtrom in fasetnim iskanjem, če se v končni situaciji zgodi isto. Tako moramo razlikovati med fasetnim iskanjem in filtri, čeprav meja med njima včasih ni najbolj razvidna. Skupno jima je, da filtrirata ogromno količino podatkov in prikažeta le želeni del glede na izbrani filter. Razlikujeta se v tem, da ima fasetno iskanje več filtrov (več fasetnih vrednosti), medtem ko ima filter samo enega [40].

Za samo uporabo fasetnega iskanja moramo v shemi za Solr atribute polja ustrezno nastaviti. Preprečiti moramo, da nam atribut razbija, odstranjuje besede in da ne spreminja vsega v male črke, ker po navadi je niz takšnega atributa že ustrezno pripravljen in mora biti tako tudi prikazan.

5.2.4 Gručenje (clustering)

Gručenje nam omogoča razvrščanje podatkov v skupine. Delamo lahko clustering na podlagi samih dokumentov ali na rezultatih iskanja, predvsem za uporabo dinamičnega fasetnega iskanja. Predvsem je velikokrat dilema, ali je samo fasetno iskanje s predefiniranimi fasetnimi vrednosti boljše od gručenja in tako pridobiti ustrezne fasetme vrednosti [33]. To je zelo odvisno od posamezne težave. Odločili smo se za uporabo gručenja v okviru faset, poleg samega faseta po predmetih, ker veliko virov podatkov ni imelo uvrstitve po predmetih. Tako potem tudi za vire podatkov, ki nimajo nobenih uvrstitev ob samih rezultatih, naredimo gručenje in jih tako uvrstimo pod določene teme. Solr za gručenje uporablja Carrot2 ogrodje [4], v katerem so na voljo trije odprtokodni različni algoritmi: Lingo [35], STC [35], K-means [4]. Na voljo je pa tudi en komercialen algoritem Lingo3G [6], ki omogoča, za razliko od ostalih, še hierarhično gručenje in naj bi bil najhitrejši med vsemi. Ni pravila, kateri algoritem je najboljši, zelo je odvisno od samega problema [4]. Razlikujejo pa se tudi v hitrosti, kar moramo pri sami uporabi upoštevati. Pri sami uporabi gručenja rezultatov iskanja moramo upoštevati tudi dolžino teksta, če je tekst predolg, pride do težav, ker procesiranje traja predolgo in ne bi dobivali dovolj hitro rezultatov. Za dobivanje dovolj dobrih rezultatov naj bi Carrot2 rabil vsaj 20 rezultatov. Za vsako gručo dobimo tudi oceno, ki je primerljiva le z ostalimi gručami in nam pove, koliko je gruča dobra glede na rezultate. Dobimo tudi neuvrščene rezultate v svojo gručo. Carrot2 pričakuje, da je vsak rezultat sestavljen iz naslova (obvezno), teksta (neobvezno) in spletne povezave (neobvezno). Lahko nastavljamo večjo prioriteto naslovu in manjšo tekstu, da tako dobimo boljše rezultate, ker naj bi naslovi po navadi bili krajši in vsebovali manj odvečnih besed. Kot tekst lahko podamo le fragment teksta (fragment lahko dobimo z označevanje zadetkov), ker s tem lahko še izboljšamo rezultate [4].

Algoritmi za gručenje:

- STC (Suffix Tree Clustering)

Uporablja predpostavko, da je tema po navadi podana v identičnih frazah, ki se morata pojaviti vsaj v dveh dokumentih. Njegova prednost je v tem, da so fraze veliko bolj informativne, kot če uporabljamo določene frekvence besed. Deluje v dveh fazah: s prvo fazo poišče množico dokumentov, ki imajo isto frazo in jih nato samo tiste od določene meje uporabi v drugi fazi. Druga faza potem entitete skombinira v končno gručo. Ta algoritem je hitrejši, a vrača malo slabše rezultate kot algoritem Lingo [35, 4, 39].

- Lingo

  Glavna ideja algoritma je, da proces gručenja ravno obrne. Tako najprej poiščemo oznake z ustreznim pomenom za gruče in jim potem dodamo kratke odlomke teksta, s katerimi ustvarimo skupine [35].

- K-means

  Ta algoritem ima na začetku podano število točk k, ki so naključno razporejene. Okoli teh točk skuša ustvariti gruče, to pa stalno iterativno ponavlja in točke prestavlja, dokler ne dobimo centralnega mesta vsake skupine oziroma nas prej lahko omeji maksimalno število iteracij [35].

Gručenje se uporablja tudi za razne vizualizacije pridobljenih pojmov, kar omogoča tudi program Carrot Workbench. Gre za tortne diagrame in Voronoieve diagrame, ki so lahko tudi interaktivni. Ravno tako so določene vizualizacije baze z evropskimi projekti z gručenjem že delali [39].

5.2.5 Iskanje podobnih rezultatov (MoreLikeThis)

Lucene nudi tudi funkcionalnost MoreLikeThis, ki jo seveda tudi Solr. Podobne dokumente lahko dobimo glede na posamezen dokument. To lahko naredimo glede na celoten dokument ali na posamezne atribute dokumenta. Na razpolago imamo več parametrov, s katerimi lahko vplivamo na izbor podobnih dokumentov: podamo lahko minimalno in maksimalno število znakov v besedi, minimalno število ponovitev pojma, maksimalno število vrnjenih

pojmov in še druge parametre. Lahko pa že prej izločimo določene besede, da se tako ne pojavljajo v tekstu in ne vplivajo na podobnosti med dokumenti [34, 4].

Ni nujno, da za vsak dokument dobimo podobne dokumente, ker v primeru, da so v obeh dokumentih vse besede različne, ne bo nobenega ujemanja na pojme. Posameznemu polju lahko že v shemi definiramo atribut termVectors. To funkcionalnost smo uporabili za izpisovanje podobnih projektov glede na posamezen projekt. S pomočjo te funkcionalnosti lahko naredimo še klasifikacijo dokumentov v posamezne kategorije zgolj po imenih kategorij, kot so to uporabili v [27, 4]. Mi smo raje uporabili gručenje, kot da bi delali klasifikacijo v kategorije po opisanem načinu.

5.2.6 Označevanje zadetkov v tekstu (hit highlighting)

Na voljo imamo tudi možnost označevanja zadetkov in vračanja fragmentov teksta, kjer se nahaja zadetek. V vrnjenem fragmentu ali celotnem tekstu je zadetek označen z značko HTML, da ga lahko takoj prikažemo. Pri samem označevanju imamo na razpolago številne parametre, s katerimi ustrezno priredimo vrnjeni tekst [4].

Poglavje 6

Uporabniški vmesnik

6.1 Enostranska spletna aplikacija

Enostranska spletna aplikacija, angleško imenovana Single page aplication, (srečamo se tudi z imenom one page application) je aplikacija, ki ima samo eno stran, lahko pa ima več pogledov. Aplikacija na začetku prenese vse morebitne datoteke z JavaScriptom in datoteke s stilom CSS. Aplikacija se nikoli med samim izvajanjem ne prenese ponovno. Med samim izvajanjem aplikacija dobiva podatke iz virov REST, menja predloge (templating), poleg tega imamo lahko v aplikaciji različne spletne naslove za kar skrbi modul za poti (routing). Po navadi imamo tudi kontrolerje, če delamo večjo aplikacijo. Ker se večina stvari dogaja na strani odjemalca, manj obremenjujemo spletni strežnik.

Iskalniki imajo lahko kar nekaj težav s takšnimi aplikacijami zaradi uporabe JavaSripta, saj vsa vsebina ni vidna. Drugi problem so spletni naslovi, če se med različnimi pogledi spreminjajo in kakšni takrat so. Google indeksira aplikacije SPA, ki se začnejo v spletnem naslovu z #!. Takšne aplikacije imajo s stališča optimizacije za iskalnik kar nekaj težav. V primeru, da nam je pomembno, da nas iskalniki najdejo in dobijo vso vsebino, moramo dobro premisliti, če se bomo odločili za aplikacijo SPA oziroma moramo ustrezno poskrbeti, da bodo iskalniki videli ustrezno vsebino. Ker se vsa JavaScript

koda izvaja na odjemalcu, se moramo zavedati, da ima tako vsak odjemalec celotno izvorno kodo aplikacije in lahko aplikacijo v celoti popravlja. Takšne aplikacije imajo težave tudi z brskalniki, predvsem z gumboma "nazaj", "naprej" in "zgodovino", ker če se spletni naslov ne spreminja, potem gumb "nazaj" ne deluje. Tako moramo v sami aplikaciji poskrbeti, tudi za ustrezno delovanje gumba "nazaj" in "zgodovine". Aplikacije SPA lahko tečejo tudi brez spletnega strežnika na samem računalniku, tako da odpremo datoteko v samem brskalniku, ni pa to nujno. Razvoj aplikacij SPA omogočajo številna JavaScript ogrodja, kot sta na primer AngularJS in Backbone.js [18].

6.2 AngularJS

AngularJS je JavaScript ogrodje za razliko od popularnejšega jQuerya, ki pa je knjižnica. Vzdržuje ga skupnost in Google, uporablja pa se pretežno za aplikacije SPA [1]. Ker gre za ogrodje v JavaScriptu, vse teče na strani odjemalca. Z njim lahko naredimo aplikacije po vzorcu MVC (model view controller), saj naj bi omogočal lažji razvoj in testiranje aplikacij. V kodo HTML dodajamo ustrezne atribute, ki se potem izvedejo na strani odjemalca, in pa pripadajočo kodo JavaScript. Podpira sinhronizacijo med pogledom in modelom v obe strani, kar je njegova velika prednost. V primeru, da želimo imeti veljaven dokument HTML5, moramo v kodi HTML atribute pisati z dodanim data. Če vzamemo za primer ng-controller, moramo v atributu elementa pisati tako data-ng-controller. Sam kontroler je namenjen ustrezni pridobitvi podatkov, ki jih obdela in potem posreduje naprej, kar je tudi tipična naloga kontrolerja v vzorcu MVC. Delovalo bo v vsakem primeru, ne glede na to, ali data pišemo na začetku ali pa ne.

Poznati moramo $scope, ki je storitev, s katero pridobivamo podatke med modelom in pogledom. Omogoča tudi predloge (templating). Z njim naredimo več datotek HTML, v katerih je zgolj košček kode HTML z dodanimi AngularJS atributi. Potem moramo omeniti poti (routing), ki omogoča prehajanje med stranmi, ki imajo različen spletni naslov. Če vzamemo za

primer, kliknemo na gumb in dobimo izpis na strani, ki ima drugačen spletni naslov. Seveda moramo pri tem biti pozorni, če želimo, da bodo iskalniki indeksirali celotno stran in bodo do posamezne strani lahko prišli z ustreznim spletnim naslovom. Za delovanje določenih AngularJS modulov moramo naložiti ločene datoteke, ker celotno ogrodje AngularJS ni zgolj v eni JavaScript datoteki.

Same izraze v kodi HTML pišemo v obliki {{ izraz }}, če pa gre za AngularJS atribut elementa na primer ng-show="izraz == 1", pa brez oklepajev. Deluje v treh fazah (app bootstrap) po tem, ko se celotni DOM naloži. Najprej kreira injektor, potem prevede celotno storitev in v DOM-u poišče ng-* atribute, ki jih v zadnji fazi ustrezno poveže z $scope. V samem dokumentu HTML moramo tako definirati ng-app z imenom naše aplikacije in potem lahko naredimo ločeno datoteko JS s pripadajočo kodo, kjer imamo kontroler, ki dela določeno stvar. Njegova velika uporabnost so filtri in razvrščanja, ki na primer filtrirajo podatke vrstice v tabeli glede na pogoj. Dodamo lahko več filtrov in drugih stvari, ki jih ločujemo z |. Ne potrebujemo klicati nobene funkcije, ki bi nam razvrstila polje ali pa bi morali napisati svojo funkcijo. Napišemo lahko tudi svoje funkcije, ko imamo bolj kompleksno strukturo. Pojavijo se seveda tudi dogodki ob kliku na gumb, potem ustrezen odziv sprogramiramo. Ti prikažejo kaj želimo, medtem ko sama aplikacija ustrezno menja te predloge glede na izvajanje in interakcijo z uporabnikom. Upoštevati moramo isto, kot velja za SPA, da odjemalec vidi celotno izvorno kodo JavaScript [1, 16].

```
<!-- Uporaba ng-app -->
<html data-ng-app="euProjects">
<head></head>
<body>

<!-- Uporaba ng-controller -->
<div data-ng-controller="SearchCtrl">
       <!-- Uporaba ng-model in ng-click -->
       <form>
       <label class="shtext">Simple search:</label><br />
```

```
        <input type="text" data-ng-model="keywords"
            placeholder="Keywords..." data-auto-complete>
        <button type="submit" class="sbutton"
            data-ng-click="searchSimple(true,
            'simple')">Search</button>
        </form>

        <!-- Uporaba ng-show in oklepajev -->
        <div data-ng-show="result.length > 0">Number of results:
            {{resultLength}}</div>

        <!-- Uporaba ng-repeat -->
        <div class="results" data-ng-repeat="data in result"
            style="float:left;width:100%;">
                <div class="resultitem">

                <!-- Uporaba ng-bind-html, da uporabi kodo HTML,
                    zaradi em -->
                <div class="projecttitle"
                    data-ng-bind-html="data.title"></div>
                        <div class="databasescore">

                                <!-- Izpis posameznega atributa -->
                                <p>{{ data.databasename }}</p>
                        </div>
                </div>
        </div>
</div></body></html>
```

Prikaz dodajanje atributov v sam dokument HTML.

```
// Definiranje modula
myApp.controller('SearchCtrl', function($scope, $sce, $http,
    $filter) {
        ...
        // Opazovanje sprememb
        $scope.$watch('keywords', function() {
                ...
                var url = 'http://localhost:8983/solr/euprojects/' +
                    suggestName + '?q=' + $scope.keywords +
                    '&rows=50&wt=json&json.wrf=JSON_CALLBACK';
                $http.jsonp(url).success(function(data, status) {
                        $scope.data = data.suggest.suggestions;
```

```
                ...
            })
            .
            error(function(data, status) {
                alert("Error");
            });
        });
        // Funkcija
        $scope.searchSimple = function(newSearch, mode) {
            ...
            var url =
                'http://localhost:8983/solr/euprojects/clustering?q='
                + $scope.keywords + ' ... ';
            $http.jsonp(url).success(function(data, status) {
                ...
                $scope.result = data.response.docs;
                ...
            })
            .
            error(function(data, status) {
                alert("Error");
            });
        }
        ...
});
```

Prikaz AngularJS JavaScript kode.

6.3 Uporabniški vmesnik

6.3.1 Izdelava uporabniškega vmesnika

Sam uporabniški vmesnik je izdelan, kot smo že omenili, v eni sami strani (SPA). Za njegovo izdelavo smo uporabili ogrodje AngularJS. Stran ima tako eno datoteko HTML, kjer je koda HTML (seveda smo uporabili HTML5) in JavaScript koda. Poleg tega imamo še datoteko s stilom CSS same strani, kjer smo upoštevali tudi to, da naj bo stran odzivna za morebitno delovanje na drugih napravah. Uporabili smo tudi jQuery, za potrebe funkcije samodokončaj v osnovnem iskanju in koledarja za potrebe izbire datuma v naprednem

iskanju. Pri tem moramo zelo paziti, v kakšnem vrstnem redu vključujemo posamezne JavaScript datoteke, ker v nasprotnem primeru stvar ne deluje. AngularJS lahko uporablja jQuery samo, če se jQuery naloži pred njim. V nasprotnem primeru namreč AngularJS uporabi jqLite, ki je pa majhen del jQuerya in tako pride do konflikta. Pazljivi moramo biti tudi, če uporabljamo datoteke JavaScript preko omrežja CDN, ker se potem zgodi, da datoteke ne pridejo v pravem vrstnem redu. Zaradi navedenih razlogov nismo uporabljali datotek JavaScript preko omrežja CDN, ker bi morali ustrezno poskrbeti, da ne bi prišlo do konfliktov. Ob samem iskanju se tako sestavi ustrezni spletni naslov z iskanim nizom in se tako sproži zahteva na Solr, ki potem vrne rezultat v formatu JSON in ga ustrezno prikažemo.

Pri sami izdelavi in iskanju napak smo si pomagali predvsem z dodatkom Firebug za brskalnik Firefox (glej sliko 3.2), kjer smo lahko tudi spremljali, kaj nam Solr vrne v odgovoru na zahtevo. Ob sami uporabi filtrov pri iskanju v spletnem naslovu naredimo filter poizvedbe na Solr, da ne spremenimo ocene relevantnosti pri rezultatih. Zahteve na Solr se tako prožijo ob kliku na filter, odstranjevanju filtra in sortiranju. Pri podobnih dokumentih ob klikanju na več takih dokumentov ne delamo novih zahtev, samo več jih prikažemo. Podobne strani smo realizirali na drugačen način kot ima to na primer iskalnik Google oziroma so to naredili v članku [25]. Tipično iskanje podobnih strani deluje tako, da se ob kliku na povezavo podobne strani (zraven posameznega rezultata iskanja) naredi novo iskanje z istimi besedami, ampak dodano možnostjo, da se išče podobne strani. Mi pa smo to naredili na način, da že prikažemo do tri naslove s povezavami na posamezne projekte, s klikanjem pa jih je možno prikazati še več.

6.3.2 Opis uporabniškega vmesnika

Naš uporabniški vmesnik ima štiri načine iskanja (glej sliko 6.1).

Osnovno iskanje

V uporabniškem vmesniku imamo najprej osnovno iskanje (glej sliko 6.2) s katerim iščemo po celotnem tekstu projektov, poleg tega imamo na razpolago še funkcijo samodokončaj (pri drugih načinih iskanja te funkcije nimamo). V osnovnem iskanju lahko uporabljamo različne operatorje, kot so točno ta fraza, izločanje besede, manjkajoč znak, začetek niza, mehko iskanje in bližnje iskanje.

Mehko iskanje

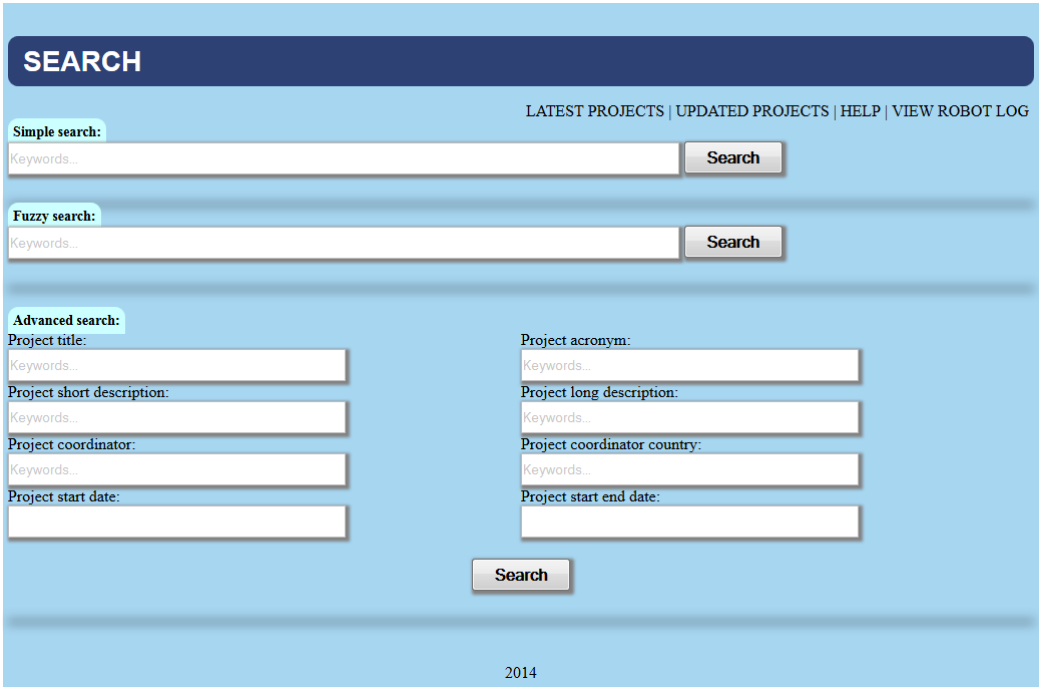
Mehko iskanje (glej sliko 6.2) imamo ločeno, čeprav lahko že v osnovnem iskanju uporabimo mehko iskanje z ustreznim operatorjem. Omogoča ravno tako enake operatorje kot osnovno iskanje.

Napredno iskanje

Napredno iskanje (glej sliko 6.2), ki omogoča iskanje po več atributih: naslovu, akronimu, opisu, imenu koordinatorja, državi koordinatorja in časovnemu obdobju začetka projekta. Med vsemi navedenimi atributi se uporabi operator AND. Napredno iskanje ima pomanjkljivosti, ker vsi projekti nimajo podanih vseh atributov in tako je pomembno, po katerem atributu iščemo. Če izberemo določeno obdobje v naprednem iskanju, v katerem se je projekt začel, nikoli ne bomo dobili vseh, ki so lahko bili v tem obdobju, ker na spletni strani ni bilo podatka o začetku projekta. V naprednem iskanju je precej bolje, če iščemo z manj atributi kot večimi. Uporabljamo lahko pri vseh atributih enake operatorje kot pri osnovnem iskanju.

Iskanje spremenjenih in novih projektov

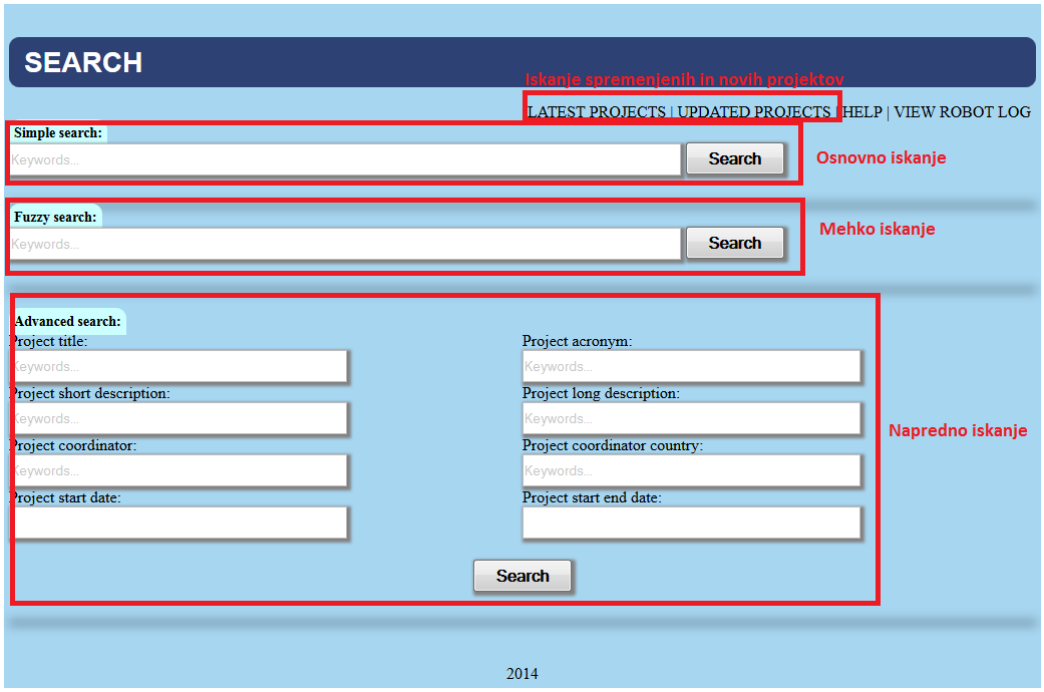
Kot zadnjo možnost imamo iskanje s klikom na gumb zadnji projekti ali pa posodobljeni projekti (glej sliko 6.2), s čimer dobimo s prvim načinom vse nove projekte glede na datum, ko je bil projekt dodan prvič v podatkovno bazo. S tem lahko sledimo novim projektom, ki se pojavijo v bazah. Z



Slika 6.1: Zaslonski posnetek našega uporabniškega vmesnika z različnimi možnimi načini iskanja.

drugim načinom pa dobimo projekte, ki so bili posodobljeni glede na to, če je robot zaznal spremembe v podatkih.

Če naše iskanje vrne rezultate, se pojavijo sami rezultati iskanja in še dodatne funkcionalnosti. Najprej se izpiše število rezultatov, potem lahko rezultate sortiramo v naraščajočem vrstnem redu po naslovu, relevantnosti, datumu začetka projekta, medtem ko je privzeto sortiranje po relevantnosti. Na levi strani imamo faset filtre po bazi, predmetih in gručenju projektov. Pri vsakem od teh filtrov lahko izberemo največ eno izbiro, ki jo potem lahko tudi odstranimo. V desnem delu so prikazani rezultati iskanja po 10 na stran. Pri posameznem rezultatu imamo naveden najprej naslov projekta, sledi akronim, če obstaja. V nadaljevanju imamo napisano, iz katere baze je projekt in samo relevantnost zadetka. Nato se izpišejo predmeti projekta, če obstajajo. Sledi dolgi opis projekta, če seveda obstaja, v nasprotnem



Slika 6.2: Zaslonski posnetek našega uporabniškega vmesnika z označenimi različnimi možnimi načini iskanja.

Slika 6.3: Zaslonski posnetek, ki prikazuje rezultate iskanja. Levo imamo na razpolago fasetno iskanje v okviru treh faset (po imenih baze, predmetih in gručenju). Na vrhu imamo možnost sortiranja po treh atributih. V sredini vidimo rezultate iskanja.

primeru se prikaže kratek opis, če tudi tega ni, se tekst ne prikaže. Potem sledijo povezave na spletno stran, kjer je opis projekta na samo spletno stran projekta, če podatek obstaja. Nato sta na razpolago še dve povezavi za iskanje z Googlom. Z eno povezavo lahko iščemo na Googlu glede na ime projekta medtem ko z drugo pa po kratkem tekstu opisa projekta, če obstaja seveda. Za samimi povezavami imamo na voljo funkcijo s podobnimi projekti, ki nam izpiše naslove do 3 podobnih projektov s samim klikanjem na gumb pa se nam izpisuje še več podobnih projektov. Ob kliku na posamezen naslov se nam odpre spletna stran, kjer se nahaja sam opis projekta. Na koncu rezultatov spodaj sledita še gumba za prehod med stranmi.

Uporabniški vmesnik je narejen odzivno, tako da ni večjih težav z uporabo na kakšnih drugih napravah in ne samo na računalniku.

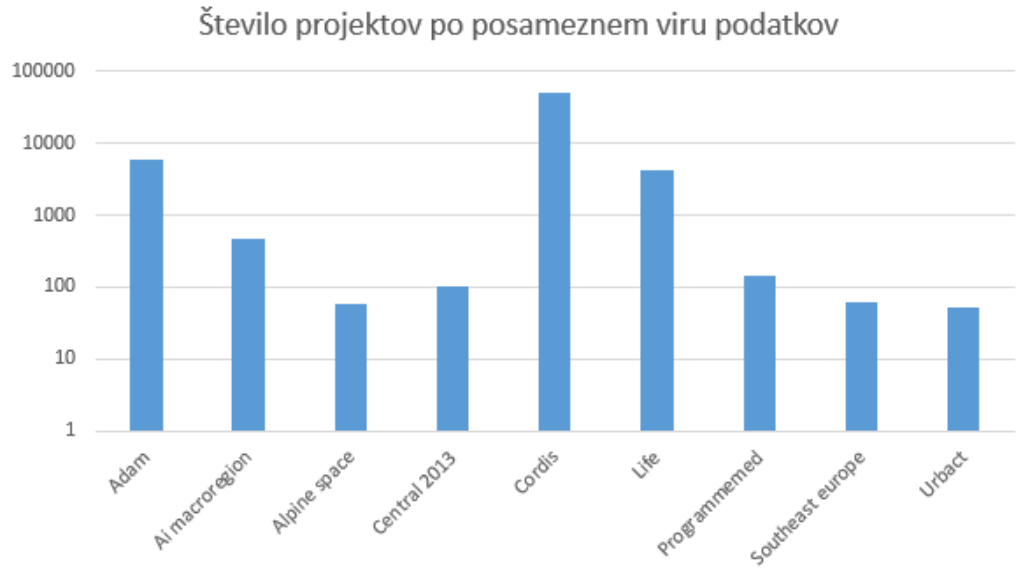
Poglavje 7

Rezultati

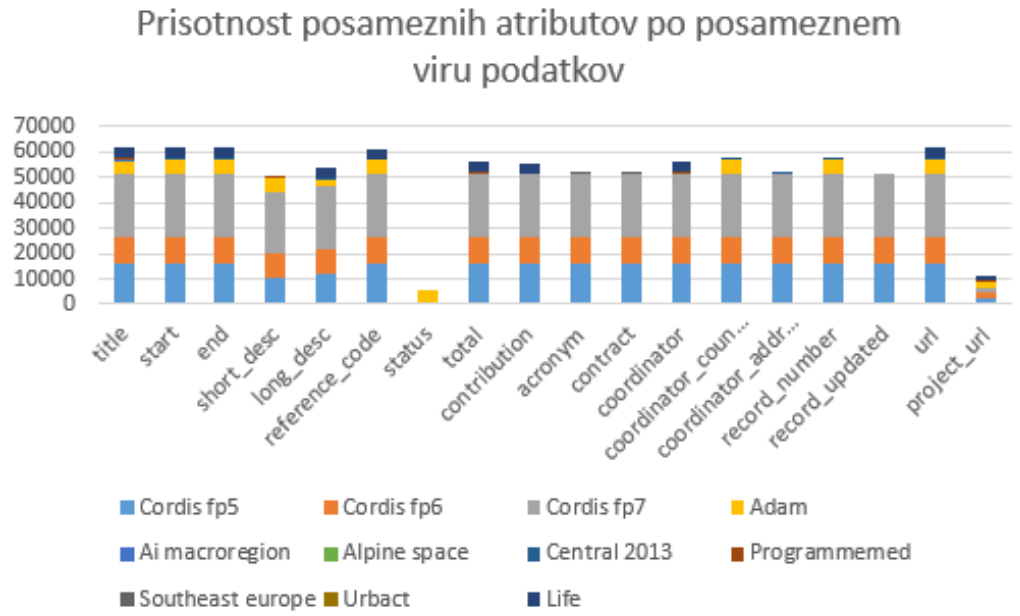
Našega dela ne moremo kvantitativno ovrednoti, ampak lahko zgolj predstavimo nekaj dejstev in številk o našem delu.

V okviru našega dela smo naredili bazo z evropskimi projekti, ki vsebuje več kot 60000 projektov. Zbrani podatki so iz devetih virov podatkov. Pri tem moramo upoštevati, da je več kot 50000 (kar predstavlja več kot 80 % vseh projektov v bazi) iz podatkovnega vira Cordis. Sledita mu še dva večja vira podatkov, Adam (pod 6000 projektov) in Life (pod 5000 projektov). Preostalih šest virov podatkov ima večinoma okoli 100 ali pa še manj projektov in tako skupaj ne presežejo števila 1000 (glej graf 7.1). Med sabo se posamezni projekti zelo razlikujejo po številu atributov (glej graf 7.2), ki so na voljo. Predvsem je to odvisno od samega vira podatkov.

Pri samem razčlenjevanju smo ugotovili, da se hitrosti in striktnost samega razčlenjevana med posameznimi razčlenjevalniki zelo razlikujejo. Iz rezultatov, ki so predstavljeni v tabeli 3.1 ugotovimo, da je najhitrejši razčljenjevalnik lxml.



Slika 7.1: Graf, ki prikazuje število projektov za posamezni vir podatkov (skala je logaritmična).



Slika 7.2: Graf, ki prikazuje prisotnost posameznih atributov v bazi.

Poglavje 8

Sklepne ugotovitve

Naredili smo namenskega spletnega robota, ki zna izvesti tudi JavaScript in s tem pridobiti celotno vsebino spletnih strani. Pri izvajanju robota smo naleteli na težave z določenimi spletnimi strežniki, ki večkrat niso vrnili rezultata, ampak zgolj napako, ali pa so bili začasno nedosegljivi. Poleg tega smo med našim delom srečali s tem, da so spletno stran prenovili ravno v tem času zato smo morali našega robota ustrezno popraviti, kar je tudi ena od pomanjkljivosti našega robota. Predstavili smo razlike med našim namenskim robotom in drugimi možnimi roboti. V primeru uporabe drugih robotov imeli veliko težav, ker ne bi znali ustrezno razčleniti določene informacije. Pri razčlenjevanju smo ugotovili, da se razčlenjevalniki razlikujejo po sami striktnosti, s katero pa nismo imeli težav. Druga stvar, po kateri se razlikujejo, je hitrost. Pri hitrosti so se pokazale občutne razlike med razčlenjevalniki. Delo je najhitreje odpravil lxml. V delu smo z narejenim vzorčnim primerom pokazali, da imajo strani včasih tudi zelo slabo strukturirane podatke. Še bolj zadovoljni bi bili, če bi imele vse strani enako strukturirane podatke ali pa bi imele na razpolago XML oziroma bi uporabljale kakšno od tehnologij semantičnega spleta.

V drugem delu našega dela smo se ukvarjali s samim iskalnikom. Podatke, ki jih je robot shranil v podatkovno bazo MySQL, smo uvozili v Solr, ki je bil namenjen, da smo ponudili napredne funkcije iskanja, kot sta fase-

tno in mehko iskanje. Poleg tega smo uporabili še funkcijo samodokončaj in gručenje samih rezultatov, ki smo jih potem uporabili v eni faseti. Kot alternativo Solr smo poiskalidruge možnosti in jih primerjali, kakšne so omejitve. Celotna aplikacija je izdelana v eni sami strani in z JavaScriptom, ki se izvaja na strani odjemalca. Za pridobivanje podatkov smo uporabili storitve REST. Takšna aplikacija ima določene pomanjkljivosti, predvsem za iskalnike, ker morajo izvesti JavaScript, da dobijo vso vsebino. Ima pa tudi svoje prednosti, in sicer se v celoti prenese samo na začetku, medtem ko se med samim izvajanjem generirajo zgolj zahteve na Solr. Naše delo je tako pripravljeno tudi, če bi se sistem uporabljal z ogromnimi količinami prometa, saj lahko Solr uporabimo skupaj s Hadoopom. Za naše potrebe tega nismo naredili saj je zadostovalo delovanje na samo enem računalniku. Celotna arhitektura sistema lahko tako teče vse na enem samem računalniku ali pa na večih, odvisno od potreb in želja.

Naše delo se v primerjavi z ostalimi članki, ki smo jih omenili v uvodu razlikuje v tem, da obravnavajo zgolj en vir podatkov o evropskih projektih. Naše delo pa vključeje devet virov podatkov o sprejetih evropskih projektih. V primerjavi s člankom [25], kjer so podatke pridobivali iz celotnega teksta, smo mi uporabili natančno razčlenjevanje atributov iz izvorne kode HTML. Ravno tako se v primerjavi s tem člankom razlikujemo po številu virov podatkov, ki je pri nas zelo majhno v primerjavi z omenjenim člankom.

Pri delu smo tako pridobili nove izkušnje predvsem v povezavi z iskanjem podatkov, kjer smo spoznali fasetno iskanje in številne druge možnosti iskanja. Ravno tako smo spoznali iskalno platformo Apache Solr za iskanje in številne alternativne možnosti, za katere prej še nismo slišali. Pri sami spletni aplikaciji smo se naučili še AngularJS, ki ga prej nismo poznali in je vsekakor uporaben. Nova spoznanja bomo lahko kasneje še velikokrat uporabili.

Vsak del naše realizirane rešitve uporablja eno možnost, ki smo jo po lastni presoji izbrali. Lahko bi izbirali med številnimi alternativnimi možnostimi (npr.: namesto Apache Solr bi uporabili Elastic search). Težko ocenimo, ali je naš izbor možnosti dejansko najboljša kombinacija. Vsekakor smo spoznali,

kaj imamo na voljo v vsakem delu. Morda nam bo v drugih primerih koristila alternativna možnost.

8.1 Možnosti za nadaljnje delo

V našem delu smo izdelali podatkovno bazo evropskih projektov, ki pa lahko služi za številne razširitve, različna druga dela in raziskave. V povezavi z bazami evropskih projektov je bilo napisanih že kar nekaj člankov, ki so delali določene analize iz teh podatkov. Za razširitev našega dela obstaja ogromno možnosti. Našteli bomo le nekaj naših predlogov za razširitev:

- Prva zelo dobra možnost bi bila, da bi dejansko vključili še več ali celo vse strani z evropskimi projekti (vključili smo vse nam znane).
- Iz naših podatkov je možno potencialno narediti številne statistične analize
- Lahko bi opazovali razne trende, npr. kaj je po posameznih področjih trenutno najbolj aktualno in s čim se ukvarjajo na projektih.
- Pri rezultatih iskanja v našem iskalniku bi lahko še bolj povezali podatke, da bi na primer ob kliku na partnerja dobili še vse projekte, v katerih je sodeloval.
- Ob taki količini podatkov lahko naredimo tudi številne vizualizacije, kot na primer število partnerjev in število projektov vizualiziramo na zemljevid.
- Takšna baza in toliko teksta je uporabno tudi za podatkovno rudarjenje in strojno učenje, ter učenje iz teksta (text mining).

Vse te predloge bi lahko kot razširitve vključili v naš iskalnik ali pa bi bili namenjeni ločenim namenom.

Literatura

[1] AngularJS. Dosegljivo na http://en.wikipedia.org/wiki/AngularJS [dostopano 18.8.2014].

[2] Apache Nutch. Dosegljivo na http://nutch.apache.org/ [dostopano 18.8.2014].

[3] Apache Solr. Dosegljivo na http://lucene.apache.org/solr/ [dostopano 18.8.2014].

[4] Apache solr reference guide. Dosegljivo na https://cwiki.apache.org/confluence/display/solr/Apache+Solr+Reference+Guide [dostopano 18.8.2014].

[5] Apache Solr vs ElasticSearch. Dosegljivo na http://solr-vs-elasticsearch.com/ [dostopano 18.8.2014].

[6] Carrot2 Algorithms. Dosegljivo na http://project.carrot2.org/algorithms.html [dostopano 18.8.2014].

[7] Cordis database in RDF format. Dosegljivo na http://wifo5-03.informatik.uni-mannheim.de/cordis/ [dostopano 18.8.2014].

[8] Damerau–Levenshtein distance. Dosegljivo na http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance [dostopano 18.8.2014].

[9] Faceted search. Dosegljivo na http://en.wikipedia.org/wiki/Faceted_search [dostopano 18.8.2014].

[10] Firebug. Dosegljivo na http://getfirebug.com/ [dostopano 18.8.2014].

[11] Inverted index. Dosegljivo na http://en.wikipedia.org/wiki/Inverted_index [dostopano 18.8.2014].

[12] Levenshtein distance. Dosegljivo na http://en.wikipedia.org/wiki/Levenshtein_distance [dostopano 18.8.2014].

[13] Lucene. Dosegljivo na http://lucene.apache.org/ [dostopano 18.8.2014].

[14] Nutch Features. Dosegljivo na https://wiki.apache.org/nutch/Features [dostopano 18.8.2014].

[15] PhantomJS. Dosegljivo na http://phantomjs.org/ [dostopano 18.8.2014].

[16] PhoneCat Tutorial App. Dosegljivo na https://docs.angularjs.org/tutorial [dostopano 18.8.2014].

[17] Selenium (software). Dosegljivo na http://en.wikipedia.org/wiki/Selenium_(software) [dostopano 18.8.2014].

[18] Single-page application. Dosegljivo na http://en.wikipedia.org/wiki/Single-page_application [dostopano 18.8.2014].

[19] Sphinx overview. Dosegljivo na http://sphinxsearch.com/about/sphinx/ [dostopano 18.8.2014].

[20] Vertical search. Dosegljivo na http://en.wikipedia.org/wiki/Vertical_search [dostopano 18.8.2014].

[21] Web scraping. Dosegljivo na http://en.wikipedia.org/wiki/Web_scraping [dostopano 18.8.2014].

[22] Welcome to OpenSearchServer! Dosegljivo na http://www.opensearchserver.com/documentation/ [dostopano 18.8.2014].

[23] What is elasticsearch? Dosegljivo na `http://www.elasticsearch.com/products/elasticsearch/` [dostopano 18.8.2014].

[24] UPDATE Syntax, 2014. Dosegljivo na `http://dev.mysql.com/doc/refman/5.7/en/update.html` [dostopno 18.8.2014].

[25] Marcelo G Armentano, Daniela Godoy, Marcelo Campo, and Analia Amandi. Nlp-based faceted search: Experience in the development of a science and technology search engine. *Expert Systems with Applications*, 41(6):2886–2896, 2014.

[26] Ori Ben-Yitzhak, Nadav Golbandi, Nadav Har'El, Ronny Lempel, Andreas Neumann, Shila Ofek-Koifman, Dafna Sheinwald, Eugene Shekita, Benjamin Sznajder, and Sivan Yogev. Beyond basic faceted search. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 33–44. ACM, 2008.

[27] John Berryman. Easy k-nn document classification with solr and python, 2013. Dosegljivo na `http://opensourceconnections.com/blog/2013/09/30/easy-k-nn-document-classification-with-solr-and-python/` [dostopano 18.8.2014].

[28] Andrzej Białecki, Robert Muir, and Grant Ingersoll. Apache lucene 4. In *SIGIR 2012 Workshop on Open Source Information Retrieval*, pages 17–24, 2012.

[29] Maryann P Feldman and Frank R Lichtenberg. The impact and organization of publicly-funded research and development in the european community. Technical report, National Bureau of Economic Research, 1997.

[30] Google. Understanding web pages better, 2014. Dosegljivo na `http://googlewebmastercentral.blogspot.com/2014/05/understanding-web-pages-better.html` [dostopano 25.5.2014].

[31] Marko Grobelnik and Dunja Mladenic. Approaching analysis of eu ist projects database. In *Proceedings of the International Conference on Information and Intelligent Systems (IIS-2002)*, 2002.

[32] Junzhong Gu, Ulrich Thiel, and Jian Zhao. *Efficient retrieval of complex objects: Query processing in a hybrid DB and IR system.* GMD, 1993.

[33] Marti A Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4):59–61, 2006.

[34] A. Johnson. How MoreLikeThis Works in Lucene, 2008. Dosegljivo na `http://cephas.net/blog/2008/03/30/how-morelikethis-works-in-lucene/` [dostopano 18.8.2014].

[35] Stanisław Osiński. Dimensionality reduction techniques for search results clustering. *Master's thesis, The University of Sheffield*, 2004.

[36] Euripides GM Petrakis and Kostas Tzeras. Similarity searching in the cordis text database. *Software Practice and Experience*, 30(13):1447–1464, 2000.

[37] Leonard Richardson. Beautiful soup documentation.

[38] SeleniumHQ. What is Selenium? Dosegljivo na `http://www.seleniumhq.org/` [dostopano 18.8.2014].

[39] Dawid Weiss Stanisław Osiński. Carrot2 User and Developer Manual for version 3.9.3. Dosegljivo na `http://doc.carrot2.org/` [dostopano 18.8.2014].

[40] Kathryn Whitenton. Filters vs. facets: Definitions, 2014. Dosegljivo na `http://www.nngroup.com/articles/filters-vs-facets/` [dostopano 18.8.2014].