Aplikacija na analizu akademske produktivnosti

Benić, Igor

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Igor Benić

APLIKACIJA ZA ANALIZU AKADEMSKE PRODUKTIVNOSTI

Diplomski rad

Pula, kolovoz, 2018. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

Igor Benić

APLIKACIJA ZA ANALIZU AKADEMSKE PRODUKTIVNOSTI

Diplomski rad

JMBAG: 0246043151, redoviti student

Studijski smjer: Informatika

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, kolovoz, 2018. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Igor Benić**, kandidat za magistra **informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA
o korištenju autorskog djela

Ja, **Igor Benić** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskoritavanja, da moj diplomski rad pod nazivom "**Aplikacija za analizu akademske produktivnosti**" koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

Sadržaj

Uvod

Ovim radom nastoji se istražiti mogućnosti trenutno dostupnih servisa i izvora podataka o akademskoj zajednici, akademskim jedinicama i rezultatima akademskih radova u svrhu analize učinkovitosti istih. Jednostavno pretraživanje i donošenje zaključaka nad raznim izvorima podataka jedan je od problema trenutno otvorenih servisa. Iako se uglavnom radi o otvorenim podacima dostupnima na određenim web-stranicama, do nekih podataka teško je doći korištenjem aplikacijskih programskih sučelja. Postoje ograničenja na količinu podataka koju je moguće preuzeti, vremenskih ograničenja poput dopuštenog broja radova koje je moguće preuzeti u određenom vremenu ili pak ograničenja na samu obradu podataka.

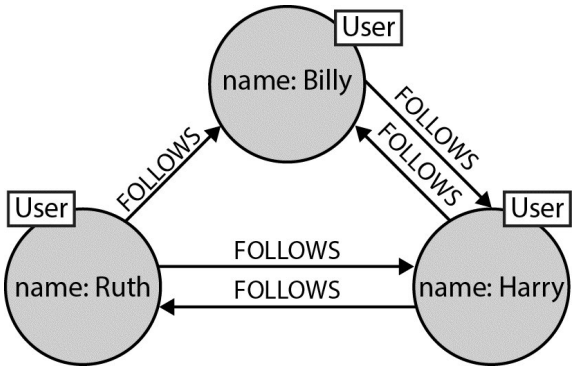
Podaci o znanstvenim člancima i ostalim radovima u akademskoj zajednici nestrukturirani su i podatke je potrebno prilagoditi nekom standardu kako bi se uklonila redundantnost i izbjegla netočnost u podacima. Strukturirani i cjeloviti podaci olakšavaju pretragu, pregled, analizu i pronalaženje zajedničkih čimbenika. Otvaranje podataka i pružanje zajedničke i predvidljive strukture podataka omogućilo bi jednostavnije pronalaženje znanja u podacima. Također, onima koji zahtijevaju vrlo specifične podatke kao što je to za razne analize ili statistike olakšalo bi se realiziranje takvih projekata. Određeni servisi nude pristupačna, ali obično drugim faktorima ograničena aplikacijska programska sučelja. U Hrvatskoj takvih servisa nema ili se oslanjaju na stare tehnologije i vrlo ograničen izbor podataka. Razni servisi dobili su reputaciju visoko kvalitetnih repozitorija znanstvenih radova. Poželjno je imati izvor podataka koji obuhvaća i ove repozitorije budući da su oni jedno od mjerila kvalitete i priznatosti akademskih radova. Sustav koji se opisuje u nastavku rada nastoji od raznih nepovezanih i često u različitom obliku pohranjenih podataka pronaći strukturu i smanjiti redundantnost u podacima.

1 Problem i prijedlog rješenja

Cilj ovog rada bio je obuhvatiti podatke o akademskim radovima hrvatskih autora i njihovim radovima i strukturirati ih na način koji omogućava jednostavniju analizu i prikaz tih podataka. Ograničenje je pronalaženje servisa koji nude takve podatke s obzirom na to da ti podaci često nisu otvoreni, nepristupačni su ili s nepotpunim podacima. Ako se koriste podaci zatvorenih poslužitelja potrebno je pomno proučiti i njihove uvjete korištenja preuzetih podataka s obzirom na to da postoji mogućnost da uvjetima ograničavaju mogućnosti prikaza preuzetih podataka. Ograničenja prikaza i korištenja podataka mogu biti razna. Neki servisi navode kako se ne smije prikazivati podatake preuzete iz njihovog repozitorija korisnicima koji već nemaju pristup istim tim podacima. Drugi servisi pak ograničavaju korištenje postavljajući ograničenja na upotrebe za koje se smije koristiti preuzete podatke. Neki servisi ograničavaju i agregaciju podataka što je neizbježno kada se ti podaci koriste u svrhu analize. *Web of Science* repozitorij ne dopušta korištenje dohvaćenih podataka za obradu u svrhe analize i ne smiju se prikazivati korisnicima koji i sami nemaju pristup aplikacijskom programskom sučelju [9].

Bez obzira na izvor podataka postoji razina greške koja može nastati kod unosa podataka. Podaci su često nepovezani te je potrebno naknadno stvoriti strukturu i dokučiti moguće veze i reprezentacije podataka. Iz tog razloga, potrebno je unaprijed analizirati podatke, odrediti njihovu strukturu, odrediti moguće probleme u strukturi. U slučaju naknadnog ažuriranja podataka i baze podataka nužno je osigurati prilagodbu nestrukturiranih podataka strukturi naše baze podataka ili sličnog načina pohrane. Podatke je potrebno strukturirati na način koji će obuhvatiti sve potrebne informacije za prikaz akademskih radova. Osim definicije strukture nužno je odrediti, ovisno o izvoru podataka, kako se ti podaci uklapaju u odabranu strukturu i kako će se oni prilagoditi da odgovaraju ako ne postoji jasna povezanost sa postojećim podacima. Ideja je otvoriti za korištenje sve prikupljene podatke sa svrhom poticanja daljnjeg istraživanja i razvoja. Mnogi servisi koji poslužuju informacije o akademskim radovima nisu otvoreni, a oni koji jesu imaju nepristupačnih aplikacijskih programskih sučelja koja često ne omogućavaju direktnu analizu podataka. Potrebno je odabrati tehnologije prilagođene

za performantno i pristupačno korištenje podataka koji su definirani svojim vezama i podliježu raznim analizama. Očekuje se i relativna ispravnost strukture i prihvatljiva točnost podataka.



Slika 1: A small social graph [31]

2 Pregled tehnologija

2.1 Graf baza

Za pohranu podataka odabrana je tehnologija graf baze podataka zbog toga što je orijentirana prema poveznicama između podataka. Ako se graf baza koristi s ovom namjerom moguće je postići mnogo bolje performanse u odnosu na tradicionalne relacijske baze podataka. Graf baze koriste vrhove i rubove, odnosno čvorišta i veze kao strukturu podataka koja potiče od matematičkog modela grafa koji se koristi u teoriji grafova. Pohranjeni podatci reprezentirani su kao skup čvorišta čiji je odnos definiran međusobnim vezama. Najveću povezanost i smanjenje redundantnosti postiže se korištenjem već unesenih podataka u graf bazu i eventualnim proširenjem podataka čvorišta i njegovih veza kada je to moguće.

Na slici 1 može se vidjeti mali primjer strukture podataka u graf bazi na primjeru korisnika Twitter socijalne mreže kao čvorova koji su međusobno povezani vezama koje pokazuju tko je s kime povezan čime veze predstavljaju prijateljstvo između korisnika [31]. Zbog velike povezanosti podataka graf baza pokazala se kao odličan izbor tehnologije u područjima gdje su veze između podataka jednako bitne koliko i sama informacija koju sadrže. Graf baze nude mogućnost definiranja kompleksnih veza njihovim nazivom pa tako i sadržajem. Podatke je moguće pohraniti u čvorove, kao što bi u relacijskoj bazi koristili tablice, uz izuzetak što ne postoji strogo definirana shema podataka pa je istu moguće mijenjati samim dodavanjem podataka drukčije strukture u čvoru postojećeg tipa. Na isti način podatke se može pohraniti i u samoj vezi između čvorova ako za to postoji potreba. Iz tog razloga, graf bazom je moguće vrlo detaljno opisati podatke i njihove veze. Sve informacije koje su potrebne za daljnji razvoj baze podataka sadržani su u vezama i čvorovima što također olakšava razvoj.

Za razliku od tradicionalnih relacijskih baza podataka, graf baze koriste veze kao glavni pokretač u pretrazi i pronalasku informacija. Relacijske baze omogućuju povezanost svojim vanjskim ključevima koji sami po sebi ne nose nikakve informacije o tome što ta veza zapravo predstavlja. Programsko rješenje poput poslužitelja potrebno je za definiranje konkretnog značenja vanjskih ključeva u relacijskoj bazi. Načinom na

koji se te veze koriste na poslužitelju moguće je otkriti namjenu veze. Kako bi otkrili značenje veze u relacijskoj bazi nužno je proučiti korištenje u takvom softverskom rješenju. Tablice povezane sa ”više na više” mogu sa sobom nositi i informacije budući da se na vezne tablice može postaviti određene podatke pored vanjskih ključeva dviju tablica koje povezuje. Pristup ”više prema više” vrlo je efikasan za dohvaćanje podataka prve razine, odnosno podatke dviju povezanih tablica preko vezne tablice. Graf baza svoju efikasnost pokazuje u dohvaćanju druge i svake sljedeće razine podataka. Prema Ian Robinsonu dohvat podataka u relacijskoj bazi realizira se dohvaćanjem svih rezultata pretraživanja i naknadnog filtriranja prema zatraženim uvjetima upita. Graf baza uzima tražena čvorišta kao početne točke za pretraživanje zbog čega je mnogo efikasnija u daljnjem pretraživanju direktno koristeći veze između čvorišta za pripremu rezultata upita.

Tablica 1: Pronalaženje prijatelja u relacijskoj bazi u usporedbi s učinkovitim nalazima u Neo4j graf bazi [31].

Dubina	**Relacijska baza (s)**	**Graf baza (s)**	**Broj rezultata**
2	0,016	0,01	~ 2500
3	30,267	0,168	~ 110000
4	1.543.505	1,359	~ 600000
5	∞	2,132	~ 800000

U tablici 1 može se vidjeti ispitivanje nad podacima koje je obavio Ian Robinson, koristeći kao primjer socijalnu mrežu s milijun čvorišta koja predstavljaju ljude od kojih svaki ima po pedeset prijatelja u vidu veza između čvorišta. U ovom istraživanju pokazalo se da relacijska baza ima slične performanse kod dohvaćanja prijatelja od prijatelja odnosno na dubini podataka od dva. Već na trećoj dubini prijatelj od prijatelja od prijatelja može se vidjeti velika razlika između graf baze i relacijske baze, gdje je trajanje dohvata do tristo puta duže za relacijsku bazu. Za visoko povezane podatke i kompleksniju analizu podataka graf baza se pokazala kao bolje rješenje. Jedna od graf baza, Neo4j, koristi upite pod nazivom Cypher koji nastoji korisnicima tradicionalnih relacijskih baza olakšati prijelaz sa SQL upita na Cypher upite uz dodatno korištenje veza. Čvor se može opisati koristeći labelu, odnosno tip čvora, i njegovim parametrima. Veze se opisuju svojim nazivom i njezinim parametrima. Najveću brzinu

```
{
 "name": "John Smith",
 "address": "121 John Street, NY, 10010",
 "age": 40
}
```

Slika 2: JSON struktura osobe s adresom

```
{
 "name": "John Doe",
 "age": 38,
 "email": "john.doe@company.org"
}
```

Slika 3: JSON struktura osobe s email adresom

upita graf baze moguće je dobiti samo u slučaju da su nam već poznati polazni čvorovi za razliku od pretraživanja po svim čvorovima određenog tipa. Sa svrhom ubrzavanja općenitog pretraživanja graf baze može se koristiti ElasticSearch pretraživač o kojem će se više govoriti u sljedećem poglavlju.

2.2 ElasticSearch pretraživač

ElasticSearch je dokument baza koja se ne definira strogom strukturom podataka. U dokument bazu podataka moguće je pohraniti nestrukturirane podatke. Svaki zapis u dokument bazi može se tretirati kao redak u relacijskoj bazi podataka. Jedan od formata za pohranu dinamičkih struktura podataka je JSON koji prirodno podržava različite tipove podataka. Na slikama 2 i 3 može se vidjeti primjer podataka o osobi strukturiran u JSON formatu [6]. Oba primjera mogu se pohraniti u dokument bazi grupirani pod istim tipom podataka, primjerice kupci. Ovakva prezentacija podataka daje nam fleksibilnost kod uređivanja i pohranjivanja podataka, te nismo ograničeni trenutnom strukturom podataka kao što je slučaj kod tradicionalnih relacijskih baza podataka. Nije nužno ažurirati strukturu baze podataka na novu strukturu kako bi obuhvatili i novu i staru strukturu baze podataka ako želimo koristiti novu ili prilagođenu strukturu. ElasticSearch također ima razne mogućnosti za horizontalno skaliranje pretraživanja na više instanci s distribuiranim podacima i računalnom snagom po raznim čvorovima sustava.

Glavna prednost ElasticSearch pretraživača je u obrađivanju tekstualnih podataka. Tradicionalne relacijske baze obično nemaju napredne mogučnosti pretraživanja po tekstualnim podacima te su uglavnom ograničene na pretraživanje dijelova teksta ili u nekim slučajevima pretraživanje regularnim izrazima. Naprednim indeksiranjem ElasticSearch daje nam mogućnost pretraživanja svih pohranjenih podataka. Indeksiranje podataka ElasticSearch radi nad svakim od tipova dokumenata koji pohranimo u ovu dokument bazu, te se stoga može reći da je indeks grupa podataka prema tipu podataka u ElasticSearch pretraživaču.

Analiza podataka također je jedna od mogućnosti ElasticSearch pretraživača s raznim funkcionalnostima za agregaciju podataka. Svaka ElasticSearch instanca može biti dio većeg sustava ako želimo horizontalno skalirati, odnosno koristiti više ElasticSearch instanci u vidu klastera. Svaka jedinica klastera može imati jedan ili više indeksa i ima sve mogućnosti pojedinog ElasticSearch čvora. U slučaju da koristimo klaster, ElasticSearch pohranjuje samo dio podataka na svakom čvoru te se svaki čvor brine o tome da pruža sve mogućnosti za svoj dio podataka. Za upravljanje ElasticSearch instancom koriste se aplikacijska programska sučelja odnosno API sučelja. Neka od ElasticSearch aplikacijskih programskih sučelja su:

- Dokument API
- API za pretraživanje
- API za agregaciju
- API za indekse
- Klaster API

Sva programska sučelja ElasticSearch baze primaju i vraćaju podatke u JSON formatu. Dokument API nudi nam sve metode za izradu, čitanje, ažuriranje i brisanje dokumenata baze [33].

2.3 Express i React okviri

Javascript je skriptni jezik koji je nastao 1995. godine i programerima pruža mogućnost kreiranja dinamičkih programa na web stranicama koje poslužuje pregled-

nik Netscape Navigator. Podršku za Javascript s vremenom počinju nuditi i ostali preglednici, te on postaje *de facto* standard za izradu dinamičkih i interaktivnih web stranica. Bilo da se radi o dijelom dinamičkim stranica s povremenim osvježavanjem stranice prilikom navigacije ili pak u potpunosti dinamičkim stranica bez osvježavanja pri navigaciji. Javascript s vremenom postaje jedino rješenje za izradu dinamičkog sadržaja na klijentskoj strani korisničkog preglednika [16]. Alternative poput ActiveScript i Java appleta dijelom zbog svojih loših performansi i sigurnosnih propusta padaju u drugi plan kao izbor za dinamički sadržaj na internetu [1][38].

Dinamički jezici poput Javascript-a programeru daju veliku slobodu s obzirom na to da nemaju tipove podataka te se interpretiraju kada je potrebno metodom JIT, odnosno samo kada je određeni dio koda potrebno izvoditi. Ovakav sistem pokretanja programa je u kontrastu s jezicima koje je potrebno kompajlirati prije pokretanja programa kao što je slučaj sa statičkim jezicima. Javascript ne podržava eksplicitno postavljanje tipova podataka na varijable programa te se implicitno dodijeljeni tip podatka provjerava kada je potrebno, za vrijeme interpretacije programa. Ova fleksibilnost često znači da je sustav otporniji na greške kod pretvorbe podataka. Implicitno rješavanje grešaka mnogo zahtjevnije je nego što je slučaj sa jezicima kod kojih je moguće definirati tipove podataka. Potrebno je poznavati sustav koji implicitno rješava razlike u tipovima podatka s obzirom na to da mnoge pretvorbe nisu intuitivne i mogu dovesti do grešaka u programu. Program će se u većini slučajeva program nastaviti izvoditi ako dođe do greške, ali protivno našem očekivanju.

Javascript se iz dinamičkog jezika koji se pokreće isključivo u preglednicima, s vremenom i određenim projektima prebacuje i na poslužiteljsku i aplikacijsku stranu [16]. Node.js jedan je od takvih projekata i omogućuje nam pokretanje Javascript aplikacija na gotovo svakoj platformi i bez okružja internetskog preglednika [26]. Projekt se zasniva na V8 Javascript interpreteru kreiranom od strane Google razvojnog tima primarno za interpretaciju Javascript-a u Google Chrome internetskom pregledniku.

Kako je V8 interpreter softver otvorenog koda iskorišten je za pokretanje Javascript programa bez okružja internetskog preglednika. Tako je uz pomoć Node.js skupa biblioteka moguća izrada web poslužitelja i aplikacija za stolna računala. Ovakve apli-

kacije imaju sve pristup svim funkcionalnostima koje bi imale u pregledniku uz dodatne mogućnosti za razvoj poslužiteljskih i desktop aplikacija [8]. Neke od mogućnosti Node.js platforme su [25]:

- Klastering
- Pokretanje potprocesa
- Kriptografski alati
- Prepoznavanje DNS adresa
- Alati za asinkrono izvođenje koda
- Upravljanje datotečnim sustavom
- HTTP, HTTP2 i HTTPS protokoli
- TCP i IPC alati
- Alati za rad s operativnim sustavom
- Podrška za tokove podataka
- Alati za TLS/SSL sigurnosne protokole
- UDP datagrami
- URL parsiranje i prepoznavanje
- Alati za kompresiju

Popularnost Javascript programskog jezika i njegova nepobitna pozicija kao jedinog jezika internet preglednika koji su zastupljeni na gotovo svim platformama rezultirala je velikom podrškom i kontinuiranim unaprjeđenjem ovog dinamičkog jezika. Javascript tako uspijeva stati uz rame s mnogim statičkim jezicima, svojom brzinom [37], jednostavnošću razvoja [10], velikom količinom biblioteka otvorenog koda [23] i velikom zajednicom razvojnih programera [34].

Za izradu aplikacija koriste se određeni skupovi biblioteka ili programski okviri ovisno o domeni problema. Programski okviri nam omogućuju lakše rješavanje problema ovisno o namjeni našeg programa kako ne bi već riješene probleme rješavali iznova. Izuzetno je važno koristiti već isprobane tehnologije ako se radi o implementaciji sigurnosti unutar našeg sustava. Arhitektura projekta i inicijalno postavljanje strukture projekta također su neke od stvari koje nam programski okviri obično nude. Prema Riehle i Gross programski okvir je dizajn i implementacija za višekratnu upotrebu. Dizajn predstavlja model aplikacije ovisno o namjeni okvira, dok implementacija definira kako se takav model može koristiti. Dobro poznavanje domene za koju se programski okvir priprema i programira presudno je za izradu programskog okvira. Programski okvir predstavlja kumulativno znanje i iskustvo o tome kako bi softverska arhitektura i njezina implementacija za aplikacije određene domene trebala izgledati ostavljajući dovoljno slobode za uređivanje i rješavanje problema domene aplikacije [30].

ReactJS jedan je od programskih okvira za izradu sučelja web, mobilnih i aplikacija za stolna računala. Napravljen od strane Facebook razvojnog tima. Ovaj okvir olakšava ažuriranje sučelja time što se programer brine za ažuriranje podataka aplikacije i obavještavanje okvira da je došlo do promijene podataka, dok se okvir bavi optimiziranim ažuriranjem sučelja. Ovisno o tome koliko su znatne promjene na sučelju s obzirom na promjenu u podacima okvir će prilagoditi ažuriranje sučelja kako bi se što efikasnije zatražene promijene pojavile na sučelju. Kako bi ReactJS mogao biti posrednik između našeg koda i onoga što se prikazuje na sučelju, preporučuje se korištenje JSX, odnosno Javascript ekstenzija. JSX jedan je od načina kako možemo pisati predloške u kombinaciji sa ReactJS kodom. JSX dopušta korištenje dobro poznatih HTML elemenata u vidu Javascript funkcija. Činjenica da se radi o Javascript funkcijama skrivena je u načinu na koji ReactJS predlošci programiraju. ReactJS predlošci obradom kroz okvir postaju HTML elementi koji se postavljaju na sučelje aplikacije koje će internet preglednik prepoznati [13].

ReactJS također potiče korištenje metoda funkcionalnog programiranja kao što su čiste funkcije i nepromjenjive strukture podataka. Čiste funkcije su funkcije koje za isti set ulaznih parametara, uvijek vraćaju istu vrijednost. Ovakve funkcije lako je testirati s obzirom na to da ne ovise o trenutnom vanjskom stanju aplikacije. Čiste funkcije

```
app.get("/", (_, res) => res.send("Hello world"));
app.post("/", (_, res) => res.send("Hello world"));
```

Slika 4: ExpressJS posrednik metode

ne smiju ažurirati stanje aplikacije osim vrijednošću koju vraćaju kao rezultat. Funkcionalno programiranje posebno je pogodno za korištenje s ReactJS okvirom budući da je svaku promjenu u podacima potrebno proslijediti ReactJS okviru. Ako se naša aplikacija sastoji od funkcija koje nisu čiste i tako do promjene stanja aplikacije dođe nepredvidivim putem, postoji mogućnost da ReactJS neće biti obaviješten o promjeni u našim podacima. Ako ReactJS nije obaviješten o promijeni on neće ažurirati sučelje s točnim podacima ili uopće [3][40].

Jedan od najpoznatijih okvira za razvoj web poslužitelja temeljen na Node.js platformi je ExpressJS okvir. Zbog svoje jednostavnosti i fleksibilnosti najčešće je i prvi odabir tehnologije za većinu specijaliziranih okvira za izradu programa web poslužitelja u Javascript jeziku. ExpressJS nam nudi tanku apstrakciju nad mogućnostima Node.js platforme za izradu programa web poslužitelja. On ne definira i arhitekturu projekta zbog čega ga je pogodno korisitit za izradu novih okvira ili jednostavnih programa web poslužitelja. Zbog velike količine Node.js paketa otvorenog koda, ExpressJS ne nastoji riješiti sve probleme razvoja programa web poslužitelja. Razvojnom programeru tako je prepušten odabir tehnologija za veliki dio slučajeva. Mnogi slučajevi podržani su u nekom obliku pri razvoju programa web poslužitelja na Node.js platformi. Neke od mogućnosti ExpressJS okvira su:

- Posluživanje statičkih datoteka
- Postavljanje zaglavlja upita
- Primanje svih HTTP/HTTPS upita
- Iscrtavanje HTML i ostalih predložaka
- Alati za rad s podizanjem i spuštanjem datoteka
- Ekstenzibilnost kroz sustav posrednika u rješavanju upita

ExpressJS može poslužiti bilo koji oblik datoteke i programskog aplikacijskog sučelja. Omogućava uređivanje i čitanje zaglavlja upita. Podržava sve HTTP metode poput GET, POST, PATCH, PUT, DELETE, OPTIONS i HEAD. Sustavom proširivosti, koristeći metodu posrednika, moguće je kod svakog upita proširiti mogućnosti ExpressJS okvira. Metoda posrednika jedan je od glavnih načina na koji ExpressJS obrađuje upite koji dolaze na web poslužitelj. Posluživanje klijenata ExpressJS web poslužitelja zapravo je odgovor jednog od posrednika u lancu posrednika koji je prepoznao da može odgovoriti na proslijeđeni upit. Prepoznavanje može biti razno i jedno od najčešćih je prepoznavanje po nazivu putanje. Ako naziv putanje odgovara zadanom, izvodi se funkcija povezana s putanjom. Na slici 4 može se vidjeti jedan od primjera kako ExpressJS poslužuje upite. App predstavlja ExpressJS instancu poslužitelja. GET i POST metode na app instanci naznačuju da će putanja, definirana sa prvim parametrom metode GET i POST, biti odgovorena sa funkcijom koja je drugi parametar ako je upit tipa GET odnosno POST. Redom kojim su napisane izvoditi će se metode upita dok jedna ne završi upit jednom od funkcija koje prekidaju daljnje pozivanje. Jedna od metoda koje prekidaju daljnju provjeru je i send metoda na res, odnosno response objektu upita.

ExpressJS također ima veliki odabir programa za iscrtavanje predložaka kod posluživanja statičkih web stranica. Ima podršku za razne programe za obradu upita. Moguće je uključiti metode za dekodiranje prilikom ulaznih i enkodiranje prilikom odlaznih JSON i sličnih podatkovnih formata. U ovom radu, ExpressJS okvir koristit će se kao tanki web poslužitelj u vidu posrednika između klijenta i graf baze i klijenta i ElasticSearch pretraživača s ciljem postavljanja SSL sigurnosnih certifikata [18].

2.4 Typescript

Kako je Javascript vrlo fleksibilan te nema eksplicitno definiranje tipova podataka, kod zahtjevnijih aplikacija preporučuje se korištenje određenih paketa za dodavanje funkcionalnosti prepoznavanja tipova podataka. Na ovaj način možemo biti djelomično sigurni da barem unutar naše aplikacije ne može doći do neželjenih implicitnih promjena u tipovima podataka. Na slici 5 možemo vidjeti kako se kreira Typescript tip podataka s nazivom Point2D koji je definiran JSON strukturom s dva broja, x i y tipa

```
interface Point2D {
    x: number;
    y: number;
}
```

Slika 5: Kreiranje Typescript sučelja

```
let point2D: Point2D = {x: 0, y: 10};
```

Slika 6: Korištenje Typescript sučelja

broj. Na slici 6 prikazano je kako se koristi takav tip podataka kako bi osigurali da varijabla Point2D ima odgovarajuću strukturu koju definira Point2D sučelje odnosno x i y varijable tipa broj. U ovom slučaju x je postavljen na vrijednost 0 i y na vrijednost 10.

Kod podataka koje dohvaćamo s web poslužitelja i dalje postoji mogućnost greške. Ova greška može nastati kada nam web poslužitelj pošalje krivi tip podatka u odgovoru na upit. Typescript ne nudi provjeru za vrijeme izvođenja već samo prilikom razvoja. Zbog toga je bez obzira na korištenje Typescript jezika potrebno provjeriti ispravnost podataka iz vanjskih izvora. Nakon razvoja, Typescript je potrebno pretvoriti u Javascript kod za izvođenje. To je moguće na dva načina:

- Transpaliranje koda
- Učitavanje Typescript koda u memoriju

Transpajliranje je naziv za pretvaranje koda iz izvornog u izvorni kod. Ovime se ne dobiva strojni kod već samo druga reprezentacija polaznog koda. Kod razvoja sučelja, tipove podataka možemo koristit oslanjajući se na mogućnosti Typescript jezika. Typescript kod tada pretvaramo u Javascript koji se potom koristi prilikom posluživanja koda pregledniku. Korak transpajliranja rezultira duplim datotekama, jedna je izvorni kod, dok je druga rezultat pretvorbe, u ovom slučaju Javascript. Na isti način može se programirati i aplikacije web poslužitelja. U tom slučaju postupak pretvaranja koda možemo izbjeći. Typescript aplikaciju moguće je učitati u memoriju poslužitelja u Javascript obliku bez prijašnjeg pretvaranja izvornog koda.

Osim tipova podataka, Typescript nam nudi mogućnost odabira u koju verziju Javascript koda želimo koristiti prilikom pretvorbe. Ovime je moguće olakšati razvoj

koristeći najnovije mogućnosti Javascript jezika. Nakon koraka transpajliranja one će se pretvoriti u mogućnosti istih funkcionalnosti bilo koje od starijih verzija Javascript-a. Ovo je također korisno za kod na web poslužitelju ako želimo veću kompatibilnost sa starijim verzijama Node.js platforme. Sva konfiguracija povezana s Typescript transpajliranjem nalazi se u tsconfig.json datoteci u glavnoj mapi projekta. Neke od mogućnosti koje nam tsconfig.json datoteka nudi su [22]:

- Podešavanje postavki transpajlera
- Uključivanje i isključivanje mapa koje želimo uključiti u kompajliranje
- Dopusti kompajliranje Javascript datoteka
- Razne postavke za osiguravanje korektnosti koda:
    - Dopusti nedostupne dijelove koda
    - Dopusti nekorištene varijable
    - Implicitno korištenje bilo kojeg tipa podatka
    - Implicitno vraćanje vrijednosti funkcija
    - Implicitni ”this” izraz
    - Dopusti nekorištene lokalne varijable unutar funkcija
    - Dopusti korištenje nekorištenih parametara funkcije
    - Prikaži grešku ako funkcija nema definiran tip podatka koji vraća
    - Uključi obavezno inicijaliziranje varijabli
- Uključi provjeru Javascript datoteka
- Generiraj datoteke s definicijama Typescript sučelja
- Postavljanje nekih od izlaznih mapa
    - Izlazna mapa za Typescript definicije
    - Izlazna mapa za kompajlirani Typescript kod
- Prikaži dijagnostičke podatke
- Postavljanje ograničenja na veličinu Javascript projekta

- Određene postavke za napredne Javascript funkcije

  Omogući korištenje Javascript iteratora

  Omogući korištenje dekoratora

- Omogući generiranje datoteka s definicijama izvornog Typescript koda
- Uključi podršku za JSX predloške
- Odabir Javascript biblioteka koje se smiju koristiti
- Odabir rješavanja Javascript modula
- Obriši komentare kod kompajliranja

Typescript transpajler nudi nam mnoge mogućnosti za preciznu konfiguraciju izrade i provjere našeg koda. Moguće je naznačiti koje datoteke želimo uključiti u transpilaciju te eksplicitno isključiti određene datoteke i mape. Osim postavki transpilacije, možemo definirati određene postavke kojima možemo osigurati veću kvalitetu koda. Određeni parametri Typescript konfiguracije obavijesti će nas ako postoje varijable koje nismo inicijalizirali, funkcije na kojima nismo naznačili tip podatka koji ona vraća, ako imamo varijabli koje se ne koriste, ako imamo parametre funkcije koji se ne koriste unutar funkcije i ostalo. Obzirom da Typescript transpajler cijeli kod pretvara u neku od starijih verzija Javascript-a, Typescript također podržava i najnovije, tek predložene mogućnosti Javascript-a. Jedan od primjera su dekoratori. Oni su trenutno eksperimentalna mogućnost Javascript jezika. Dekoratori su primjer kompozitnih funkcija, te nam omogućuju definiranje funkcija koje primaju druge funkcije. U isto vrijeme one ih i mijenjaju uz pomoć metapodataka povezanih s originalnom funkcijom. Ovo zahtijeva uključivanje određenih postavki Typescript transpajlera budući da Typescript transpajler zahtijeva definiranje biblioteka koje želimo koristiti ovisno o verziji Javascript-a. Typescript je dodatak na Javascript bez ikakvih izmijena na Javascript mogućnostima. Zbog toga je moguće postaviti verziju Javascripta koja će se koristiti u kombinaciji s dodacima Typescript jezika [36]. Typescript je kreiran od strane Microsoft kompanije kako bi Javascript programski jezik približili većim razvojnim timovima i olakšali razvoj na većim softverskim projektima. Tako se postojeći Javascript projekti mogu bolje analizirati prebacimo li ih u Typescript jezik. Transpajler nam može ukazati

Slika 7: GraphiQL sučelje

na greške čak i u Javascript kodu. On nudi i modularno učitavanje koda, već spomenute opcionalne tipove podataka i statičko prepoznavanje tipova i simbola, te ako koristimo tipove podataka, nudi nam i preporuke u kodu prilikom razvoja i dovršavanje izraza [17].

2.5 GraphQL

GraphQL je skraćeno od ”Graph Query Language”, odnosno jezik upita graf baze. Moguće ga je koristiti u svakom slučaju kada poslužitelj ili baza podataka podržavaju GraphQL programsko sučelje. Većina izvora podataka, baze podataka i ostala aplikacijska sučelja, moguće je povezati sa njegovim programskim sučeljem pa čak iz više izvora podataka. GraphQL upiti šalju se na jednu putanju na serveru, te je tamo moguće izvoditi sve upite. Upiti su prvenstveno namijenjeni za konstruiranje s klijentskih aplikacija, što je pogotovo pogodno za korištenje u slučajevima kada se radi o otvorenim aplikacijskim sučeljima u kojem slučaju nije od velike važnosti sakriti kod upita na poslužiteljskoj strani aplikacije. GraphQL optimizira svaki od upita pa tako vraća klijentu samo i isključivo zatražene podatke kako ne bi dohvaćali višak podataka odnosno one podatke koji se ne koriste na klijentu. Također dopušta pronalaženje dostupnih parametara svakog od mogućih upita i popis svih mogućih upita aplikacijskog sučelja. Koristimo li ReactJS okvir za razvoj aplikacije, postoje određene biblioteke koje nam olakšavaju povezivanje sa njegovim aplikacijskim programskim sučeljem. Apollo je jedna od takvih biblioteka i nudi nam integraciju pomoću koje možemo definirati koja komponenta aplikacije zahtijeva koje podatke sa GraphQL apli-

kacijskog programskog sučelja. Komponenta koja traži podatke može biti u tri stanja, učitavanje, stanje kada je došlo do greške i stanje kada je upit uspješno završio. U slučaju da je upit uspješno izveden, komponenta dobiva rezultat upita te ga može prikazati na odgovarajući način na sučelju. Jedna od alternativa GraphQL upitima su i REST upiti. REST je skup preporuka po kojem se mogu napraviti RESTful sučelja između kojih je i HTTP koji u svom tijelu poruke između ostalih formata podržava JSON. Za omogućavanje JSON strukture potrebna je instalacija dodatnih biblioteka na poslužitelju kako bi prikazali dokumentaciju povezanu sa JSON aplikacijskim programskim sučeljem. Osim dokumentacije poželjno je pratiti zadanu strukturu kako bi dokumentacija mogla prikazati definiciju JSON aplikacijskog programskog sučelja.

Kako se u ovom radu koristi graf baza koja također podržava i GraphQL programsko sučelje on je odabran u kombinaciji s klijentskim kodom za izradu GraphQL upita radi što bolje integracije klijentske strane i graf baze podataka. On je prvenstveno orijentiran prema izvorima podataka koji prate strukturu grafa. Programsko sučelje tako ostaje otvorenim za korištenje svakom klijentu koji koristi odgovarajuće klijentske biblioteke. Također je moguće konstruirati upite u obliku JSON strukture koje baza podataka prepoznaje. GraphQL podržava i GraphiQL sučelje za vizualni prikaz strukture aplikacijskog programskog sučelja uz mogućnost izvođenja upita. Ovime se olakšava razvoj budući da se struktura nalazi na jednom mjestu. Ovakva otvorena dokumentacija posebno je pogodna kod korisnika koji se žele bolje upoznati s mogućnostima otvorenog aplikacijskog programskog sučelja.

Na slici 7 može se vidjeti primjer GraphQL upita koji se može izvoditi kroz GraphiQL sučelje. Ono prepoznaje tipove podataka svakog od omogućenih programskih sučelja ili baza podataka pa je tako moguće vidjeti i koji su nam ponuđeni parametri svakog od upita ovisno o traženom resursu [3].

Alternativa izradi upita u slučaju korištenja Neo4j graf baze jest kreiranje Cypher upita u obliku niza znakova koji se potom šalje na obradu na otvorenu putanju graf baze za izvođenje Cypher upita. Na ovaj način na klijentskoj strani nemamo nikakvu sigurnost pri razvoju kod izrade upita, ne znamo koji su nam parametri ponuđeni, s kojim resursima je moguće graditi upite, kako su nam resursi povezani, koje su veze među

njima niti koji tipovi veza postoje, ni kakvu strukturu s graf baze možemo očekivati. Ovo su samo neki od razloga zašto odabrati GraphQL kao veznu tehnologiju između bilo koje baze podataka sa strukturom koja može iskoristiti graf strukturu pri izradi upita nad podacima. Na ovaj način koristi se potpuna sintaksa graf baze na klijentu ostvarena korištenjem GraphQL klijentskih biblioteka. U slučaju da ne koristimo GraphQL potrebno je implementirati poslužitelja koji bi bio posrednik u razmijeni podataka između klijenta i baze podataka. U ovom slučaju nije bila potrebna implementacija posrednika isključivo za otvaranje sučelja prema bazi podatka s obzirom na to da su podaci i baza otvoreni te je na klijentu moguće konstruirati potpune upite koji bi bili potrebni kod izrade klijentske aplikacije.

3 Postavljanje okoline

3.1 Izvori podataka

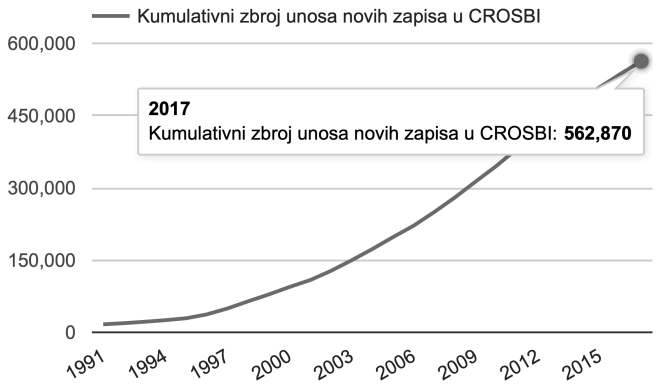
Određeni servisi nude prilagođena programska sučelja za ”harvesting” ili prikupljanje podataka o akademskim radovima. Jedan od protokola za prikupljanje podataka zove se OAI-PMH ili ”Open Archives Initiative Protocol for Metadata Harvesting”. Arhive koje nude mogućnost preuzimanja i pregledavanja podataka o akademskim radovima putem ovakvog protokola moguće je preuzeti i otvoreno koristiti u svoje svrhe. Ovakve protokole uglavnom koriste poslužitelji koji već na neki način omogućavaju pristup svojim podacima o akademskim radovima bilo kroz web sučelje ili druga aplikacijska programska sučelja. Ostali agregatori akademskih podataka uglavnom imaju ograničen pristup podacima koji su potrebni za stvaranje potpune slike o akademskoj jedinici; autoru, akademskom radu, instituciji i sličnom. Poslužitelje koji podržavaju OAI-PMH moguće je pronaći na web stranicama Open Archives inicijative [27]. Određeni podaci poslužitelja OAI-PMH protokola nisu potpuni te je u određenim slučajevima potrebno uključiti i više izvora podataka ili drugih poslužitelja OAI-PMH protokola kako bi upotpunili podatke o akademskim radovima. Jedan od servisa koji se koriste prikupljanjem podataka preko poslužitelja sukladnih OAI-PMH protokolu je i ARA(Agregator hrvatskih repozitorija i arhiva) [2].

Pristup podacima o akademskim radovima, radilo se o otvorenim ili zatvorenim podatcima, uglavnom je ograničen. Otvoreni podaci su često nepotpuni ili nestrukturirani i uglavnom nedostupni u elektroničkom obliku prikladnom za pristup preko aplikacijskog programskog sučelja. Nešto zatvoreniji repozitoriji ograničavaju pristup podacima, ponekad zbog nedostatka odgovarajućeg aplikacijskog programskog sučelja. Kod nekih servisa gdje su podaci javno vidljivi i dalje je programski teško do njih doći. Podaci su ponekad zatvoreni na razini akademske jedinice, pa tako član određene znanstvene zajednice ima pristup akademskim radovima ovisno o akademskoj ustanovi kojoj pripada. Ovime se ograničava mogućnost istraživanja nad podacima akademske zajednice, produktivnosti akademskih jedinica i donošenja određenih zaključaka podržanih istima. Ovakav pristup ne poklapa se s naočigled otvorenom akademskom zajednicom, te ne potiče otvorenu konkurentnost i stvaranje jasne slike

trenutne situacije u akademskoj zajednici na temelju cijelovitih podataka. Servisi poput Scopus i Web of Science, ograničavaju posluživanje svojih podataka članovima akademske zajednice i članovima zajednice koja plaća određene pretplate ovisno o radovima kojima pojedinac želi pristupiti. Postoje određena programska rješenja i programska sučelja za pristup, pod uvjetom da je korisnik kojem se podaci poslužuju član akademske zajednice koja ima pristup traženim podacima. Servisi za prikupljanje podataka ili ”harvesting” servisi, dobar su primjer jednostavnog i relativno pristupačnog načina za razmjenu podataka o akademskim radovima. Njihova mana jest što nisu cjeloviti, ovakve podatke potrebno je proširiti i povezati s podacima iz više izvora kako bi imali potpune podatke o akademskom radu.

CROSBI servis jedan je od vrlo cjelovitih izvora podataka o akademskim radovima. Moguće je saznati određene informacije koje servisi za ”harvesting” nemaju. Neki od tih podataka su:

- Indeksiranost od strane Scopus, Web of Science i sličnih servisa
- U sklopu koje ustanove je znanstveni rad napisan
- Lista autora znanstvenog rada
- Razni izvori akademskih radova
    - Knjige
    - Članak časopisa
    - Kao rezultat projekta
    - Izlaganja na konferencijama
- Datum izdavanja rada ili časopisa u kojem je izdan članak
- Sudionici u projektu
- Jezik
- Vlasnik patenta koji povezanim s dotičnim radom
- Status rada



Slika 8: Kumulativni broj CROSBI objava

- Mjesto izlaganja
- Područje istraživanja
- Ključne riječi

Znanstvenici na stranicama CROSBI servisa mogu uređivati i upotpunjavati podatke što je u interesu autorima radi vidljivosti i kvalitete informacija o njihovima radovima. Na ovaj način znanstvenici imaju mogućnost ispraviti format i informacije svojih radova u svrhu točnosti dostupnih informacija. Ovo je samo jedan od mogućih izvora kod agregacije podataka o akademskim radovima. U svrhu analize, radi preciznosti i točnosti podataka, potrebno je uzeti u obzir više izvora podataka. Na odgovarajući način potrebno je izvoditi analizu nad podacima ovisno o znanju koje želimo dobiti iz agregiranih podataka. CROSBI servis pokazao se kao dobar primjer za demonstraciju problema u podacima o akademskim radovima, te će se u nastavku rada više govoriti o analizi, prikupljanju i povezivanju podataka.

3.2 Prikupljanje i semantička analiza

Osim već spomenutih metoda prikupljanja podataka o akademskim radovima kao što su ”harvesting” i aplikacijska programska sučelja, do podataka određene web stranice moguće je doći ”scraping” metodom. Radi se o metodi preuzimanja specifičnih podataka web stranice koji su potrebni za daljnju obradu. Preuzimanje podataka ovim putem može se poistovjetiti s preuzimanjem putem aplikacijskog programskog sučelja koje podržava XML ili JSON format podataka. Obzirom da je HTML for-

mat podataka vrlo sličan XML formatu s određenim razlikama u načinu izrade elemenata, ograničenim izborom elemenata koji se mogu koristiti i činjenicom da preglednici osim HTML-a prepoznaju i vezane podatke poput CSS stilova i Javascript programa koji se potom izvode ili primjenjuju na stranicu koja ih sadrži. Unatoč ovoj sličnosti u formatu podataka koji se preuzimaju i načinu preuzimanja, preuzimanje HTML, CSS i Javascript podataka programskim putem ne smatra se izjednačeno s preuzimanjem XML i JSON podataka s isto tako otvorenog aplikacijskog programskog sučelja.

Na slici 8 može se vidjeti broj zapisa unesenih u CROSBI servis kroz vrijeme. Zadnja informacija o broju radova pokazuje 562.870 unesenih radova. Prema njihovoj stranici, radove unose autori te su njihovi podaci ovisni o točnosti unosa od strane autora. Također, korištenje podataka je dopušteno uz izričito navođenje izvora i poveznice na originalni sadržaj stranice odakle su podaci preuzeti [11]. Podaci u CROSBI servisu nisu povezani. Svaki od podataka potencijalno se već koristi u jednom od drugih akademskih radova. Iz zajedničkih podataka moguće je napraviti poveznice koje prikazuju radove ili više detalja ovisno o tome o kojoj se vrsti resursa radi. Primjerice, pregledavajući autora, odaberemo li pregled njegovih izlaganja na konferencijama, mogle bi postojati poveznice gdje mogli bi bilo moguće vidjeti koji su sve autori izlagali na istoj konferenciji. Osim autora može se prikazati koje su se teme spominjale. Ako postoje informacije o povezanim projektima može se prikazati projekte koji su nastali iz predstavljenih ideja. Ove informacije nije moguće dobiti ako je jedini mogući resurs generički format akademskog rada bez poveznice s ostalim akademskim tijelom. Ovo dovodi do redundancije u podacima jer je svaki resurs izoliran, ima svoje podatke te nije vezan poveznicama uz druge resurse, autore, radove, konferencije i slično. Također, podaci uneseni u CROSBI, Scopus, Web of Science i slične servise, ili ne prolaze provjeru osiguravanja strogog formata u podacima ili postoje određene greške u samim podacima, gramatičke greške i slično. Iz ovog razloga, vrlo je teško podatke razložiti i direktno povezati među sobom bez dodatne pripreme. Obradom je potrebno uspostaviti stanje podataka i pronaći moguće greške u podacima. Nakon analize podataka moguće je napraviti programsku obradu podataka za daljnju obradu i pohranu podataka.

```
<div class="row">
    <p class="item-label">
        <strong>Autori</strong>
        <br>
        Gabelić Tereza ; Krbot Magdalena ; Adamec Ivan ; Habek Mario
    </p>
</div>
```

```
{
 "Autori": ["Gabelić Tereza ; Krbot Magdalena ; Adamec Ivan ; Habek Mario"]
}
```

Slika 9: Preuzeti HTML i dobivena JSON struktura

Preuzimanje podataka moguće je već spomenutom ”scraping” metodom. Za preuzimanje mogu se koristiti Javascript ili drugi jezik za izradu dinamičkih skripti koje pojednostavljuju rad s podacima u kratkoročnom i nekritičnom sustavu za preuzimanje podataka. Sadržaj HTML datoteka može se pohraniti u datotečni sustav, dok se informacije o preuzimanju i njegovom statusu mogu pohraniti u bazu podataka poput PostgreSQL ili MySQL baze podataka. Podatke o indeksu rada povezanog s HTML datotekom, informacije o rezultatu preuzimanja i eventualnoj grešci nastaloj kod preuzimanja HTML-a također je poželjno pohraniti u bazu podataka.

Kako Javascript nema podršku za paraleliziranje poslova poput nekih statičkih jezika, za ubrzavanje obrade podataka moguće je pokrenuti više instanci istog programa te koristiti memoriju za sinkronizaciju stanja između procesa kako više instanci ne bi obrađivalo iste podatke. Jedna od mogućih metoda za pohranu trenutnog stanja kroz sve procese je korištenje Redis memorijske baze podataka. Redis podatke sprema u memoriju što je čini prigodnom bazom za brzo posluživanje podataka koji se ne mijenjaju često ili za razmjenu brzih poruka između svih strana koje su povezane s ovom bazom. Redis se iz tog razloga koristi kao predmemorija, baza podataka ili kao posrednik za razmjenu poruka [29]. U ovom sustavu za paralelizaciju može se koristiti Redis baza koja sadrži indeks podatka koji bi se sljedeći trebao početi obrađivati. Svaki od Javascript servisa kod početka rada ili završetka određenog zadatka zatražuje sljedeći indeks za obradu podataka čime ujedno i povećava indeks za sljedeći servis koji ga zatraži. Na ovaj način ne dolazi do kolizije koja se može pojaviti kod korištenja datoteke za pohranu sljedećeg indeksa, odnosno za smanjenje mogućnosti kolizije brine se Redis baza. Ovo ubrzavanje obrade podataka možemo iskoristiti u svakom od koraka ako se koristi Javascript za obradu i pripremu podataka. Radi jed-

nostavnije obrade u slučaju korištenja Javascript jezika, zbog vrlo bliske integracije s JSON formatom podataka, moguće je pretvoriti HTML datoteke u JSON format podataka analizom strukture podataka HTML datoteke i definiranjem prigodne JSON strukture za određene podatke. Na slici 9 može se vidjeti primjer HTML strukture koju možemo parsirati i pojednostaviti za korištenje translacijom podataka iz HTML strukture u JSON strukturu. Postoje razni alati za pretvaranje HTML strukture u Javascript model podataka, od dobivenih elemenata potrebno je izvući nama potrebne podatke.

3.3 Sinteza podataka

Podjela podataka preuzetih HTML datoteka postignuta je na temelju HTML ¡div¿ elementa s CSS klasom ”row” koja predstavlja redak u sučelju s jednom labelom, koju možemo prepoznati po HTML elementu ¡strong¿, dok su podaci vezani uz ovaj tip podatka, u ovom slučaju Autori, ispod HTML elementa ¡br/¿, odnosno HTML elementa za prelazak u novi red. Preuzeti podaci pokazali su da se radi o raznim objavama, člancima, knjigama, konferencijama, poglavljima knjiga i ostalim objavama koje nemaju puno zajedničkih čimbenika. Iz tog razloga, preporučuje se spremanje tipa podatka, u ovom slučaju Autori, i vrijednosti za taj redak, u ovom slučaju *Gabelić Tereza ; Krbot Magdalena ; Adamec Ivan ; Habek Mario* u strukturu poput JSON objekta s ključem koji je naziv podatka i vrijednošću povezanom s tim nazivom. Ovo radimo kako se ne bi izgubili određeni podaci, kao što je to moguće u slučaju da ne pohranjujemo sve ključeve i vrijednosti nego samo određene.

Kako svaki HTML zapravo predstavlja jedan od radova i svaki rad ima svoj identifikacijski broj, parsirane podatke može se pohraniti u Redis bazu za lakše manipuliranje u budućim koracima. Redis baza prima ključ i vrijednost, ključ je u tom slučaju identifikacijski broj rada i vrijednost je JSON svih ključeva i vrijednosti iz redaka koje smo parsirali. Tijekom iteracije kroz sve HTML datoteke za vrijeme parsiranja i izvlačenja podataka iz svakog od redova koji nas zanimaju, poželjno je dobiti sve ključeve koji se mogu pojaviti u nekom od redova HTML datoteke. To je moguće postići, bez ponovne iteracije kroz sve datoteke, tako da se za vrijeme parsiranja svaki od pronađenih ključeva također sprema u sporednu JSON strukturu kako bi u njoj na kraju imali sve moguće ključeve radova. JSON struktura je u ovom slučaju također

poslužila za spremanje vrijednosti u sporednu JSON strukturu kako bi dobili osjećaj za moguć izgled podatka na svakom od ključeva. JSON format također ne dopušta iste ključeve na istoj razini u svojoj strukturi čime su se zapravo ključevi pregazivali onoliko puta koliko puta su se i pojavljivali. Po potrebi, može se dobiti informacije o broju pojavljivanja određenih ključeva, broju puta kada su određeni ključevi pronađeni u istom HTML-u i tome slično.

Nakon spremanja podataka i analizom JSON strukture sa svim mogućim ključevima HTML datoteka, dobiveni su sljedeći ključevi: *naslov*, *autori*, *izvornik*, *vrsta*, *podvrsta i kategorija rada*, *ključne riječi*, *sažetak*, *projekt / tema*, *izvorni jezik*, *znanstvena područja*, *ustanova*, *knjiga*, *urednik/ci*, *izdavač*, *grad*, *godina*, *raspon stranica*, *ISBN*, *skup*, *mjesto i datum*, *vrsta sudjelovanja*, *vrsta recenzije*, *fakultet*, *mjesto*, *datum*, *stranica*, *mentor*, *vrsta, podvrsta i kategorija knjige*, *vrsta*, *podvrsta*, *neposredni voditelj*, *broj patenta*, *datum patenta*, *nositelj prava*, *status rada* i *kolaboracija*. Bez obzira na vrijednosti koje stoje iza ovih ključeva već možemo vidjeti kako se radi o prilično redundantnim podacima, poput ključeva *datum i mjesto* i zasebnih ključeva *datum* i *mjesto*. Nakon prikupljanja podataka potrebno je iz dobivenih vrijednosti izvući podatke. Kao što se može vidjeti na slici 9, u HTML dijelu koda i u JSON rezultatu, dobivena vrijednost je polje s jednim poljem karaktera. Vrijednost je polje budući da je moguće da se u HTML retku pojavljuje više HTML elemenata ¡br/¿, dok je tip podatka samo jedan pa se stoga nalazi u ključu JSON strukture. Ovo je samo jedan od primjera podataka koji su nestandardizirani, još neki od primjera za ključ ”Autori” su sljedeći:

- Deželjin, Jadranka ; Deželjin, Josip ; Dujanić, Marčelo ; Vujić, Vidoje
- Radan, D ; Lovric, J
- Petak, Antun ; Ivan Lajić ; Sonja Podgorelec i Dragutin Babić
- Alija Kulenović , Tina Balenović , Vesna Buško

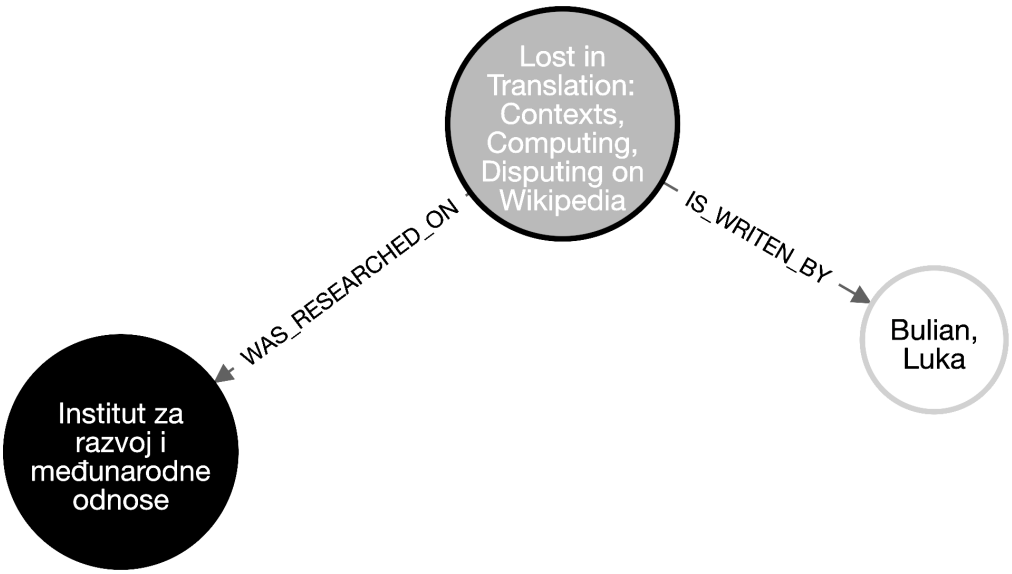
Iz ovoga je vidljivo da je na primjeru autora pojedinog autora moguće dobiti već tako što podijelimo svaku od vrijednosti po karakteru ”;” te iskoristimo *trim* naredbu za otklanjanje nevidljivih znakova s početka i kraja slijeda znakova. Iz toga razloga, napravljena je provjera kada postoji ”;” karakter u vrijednosti zaključimo da se radi o više autora,

dok je u protivnom, kada nema znaka ”;”, ali postoji znak ”,” potrebno provjeriti postoje li barem dva slijeda znakova prije nego što se pojavi znak ”,” da bi ga tretirali kao i znak ”;” pa podijelili vrijednost i zaključili da se radi o više autora. Problem nastaje kod prepoznavanja radi li se o istom autoru kada se radi o slučaju kada je autorovo ime na prvom mjestu, drugom ili pak ima samo inicijal. Ova razina prepoznavanja nije uključena u obradu podataka s obzirom na to da se za prepoznavanje autora i ostalih vrijednosti koje mogu imati vrlo slične vrijednosti u nekim slučajevima može desiti da spajanje istih vrijednosti rezultira greškom. Više će se o povezivanju govoriti u jednom od sljedećih poglavlja.

ElasticSearch pretraživač jedan je od alata kojim možemo metodom fuzzy pretraživanja odrediti koja je vjerojatnost da se radi o traženoj vrijednosti ovisno o unesenom terminu za pretraživanje. Jedna od mogućih metoda za prepoznavanje datuma u bilo kojem od podataka može biti korištenje jezika koji je često naveden u radovima pod ključem izvorni jezik. Često se format datuma poklapa s očekivanim formatom po prepoznatom jeziku. Vrlo česte greške su dakle autori i način na koji su njihova imena, prezimena, srednja imena i separatori poput zareza i točka-zarez znakova raspoređeni unutar vrijednosti.

3.4 Stvaranje strukture podataka

Radi jednostavnosti i s ciljem izbjegavanja redundantnosti podataka, svi tipovi podataka koji su dobiveni obradom podataka izdvojeni su u posebne čvorove graf baze. Podaci koji su specifični za određeni tip podatka i neće se ponavljati postavljeni su kao jedan od parametara na čvoru. Jedan od primjera je apstrakt objave koji je postavljen na čvoru tipa objava. Važno je na identificirajuće podatake postaviti indekse kako bi ubrzali pretragu. Zato što svi čvorovi imaju svoj ključni parametar. Za objavu to je naziv objave. Iz parsiranih podataka dobiveno je dvadeset i devet tipova podataka, isti su navedeni u prošlom poglavlju kao rezultat obrade podataka. Veze između čvorova definirane su na temelju tipova podataka između kojih je potrebno napraviti vezu. Cijela lista veza može se vidjeti na tablici 2.



Slika 10: Vizualni prikaz Neo4j razultata upita

Centralni čvor za većinu veza je naravno objava, ili ”ARTICLE”. Ovim pristupom odvajanja tipova podataka u svakom slučaju kada se podatak ponavlja, možemo jednostavnije doći do vezanih podataka. Pretraživanje je olakšano ako za jedan od početnih čvorova uzmemo čvor koji je određenog tipa i sadrži podatak koji nas zanima. Tako možemo vrlo jednostavno vidjeti sve radove objavljene u određenoj godini, članke povezane s određenom konferencijom, broj radova u instituciji. Najveća prednost graf baze može se vidjeti u slučaju kada nas zanimaju informacije koje su više razina udaljene po čvorovima grafa. Kao što smo pokazali u jednom od prijašnjih poglavlja, kod upita nakon druge razine veza može se vidjeti velika prednost u performansama graf baze. Primjer upita koji koristi više razina čvorova može biti pretraživanje svih autora određene institucije s radovima u indeksu ”Scopus”.

Prilikom lokalnog razvoja, za testiranje podataka korištena je Neo4j baza na lokalnom računalu inicijalizirana s aplikacijom ”Neo4j Desktop”. Neo4j Desktop aplikacija omogućava brzo i jednostavno postavljanje graf baze podataka kroz svoje sučelje. Osim postavljanja baze podataka, Neo4j Desktop nudi vizualni prikaz baze i mjesto za unošenje upita na graf bazi koji se predstavljaju kao čvorovi i veze u vizualnom obliku. Primjer rezultata upita na graf bazu koristeći Neo4j Dekstop može se vidjeti na

slici 10. Ovaj vizualni prikaz nastao je kao rezultat Cypher upita: MATCH (a:Author)-[:IS_WRITEN_BY]-(b:Article)-[:WAS_RESEARCHED_ON]-(i:Institution) RETURN a,b,i.

Rezultat obrade HTML datoteka bio je JSON pohranjen u Redis memorijskoj bazi. Kako bi se ubrzalo učitavanje velikog broja podataka u Neo4j graf bazu podaci su potom pretvoreni u CSV format podataka, odnosno u tabularni format sa separatorom točka-zarez. Određene baze podataka omogućavaju jednostavnije učitavanje pripremom velikog broja podataka u prilagođenom obliku što je često bolji odabir od izvođenja mnogo upita na bazi. Čvorovi graf baze su se tako postavili kroz CSV datoteke. Veze između čvorova definirane su u odvojenim datotekama, koristeći metodu MERGE graf baze koja provjerava postojanje zatraženih čvorova. Ako su čvorovi MERGE upita pronađeni, mijenja se njihova struktura na način definiran upitom.

Ako bilo koji od čvorova u MERGE upitu nije pronađen, cijeli slijed čvorova izrađuje se iznova, bez obzira postoji li već neki od čvorova upita. Kako se podaci ne bi ponavljali, potrebno je postaviti indekse na podacima koje ne želimo imati više puta u svom grafu kako ovaj upit ne bi prošao izvođenje ako se slučajno dogodi da jedan od čvorova ne postoji kod izvođenja MERGE upita. Budući da su svi čvorovi unaprijed učitani u bazu, moguće je pokrenuti povezivanje podataka tako što uključimo veze u pretraživanje što će rezultirati novim vezama između postojećih čvorova podataka. Nakon završetka testiranja s podacima na lokalnoj bazi podataka, budući da su svi podaci sada učitani u lokalnu bazu, koristila se DUMP metoda baze za izvoz podatka. DUMP metodom izvozimo podatke u također prilagođenom obliku koji je mnogo manje veličine od CSV datoteka i namijenjen je za uvoz u drugu bazu istog tipa. Za razliku od metode CSV datotekom, za korištenje DUMP metode potrebno je imati već ispunjenu instancu baze podataka [19].

3.5 Podizanje servisa baze podataka

Na DigitalOcean servisu za postavljanje virtualnih privatnih poslužitelja, napravljena je nova instanca s Ubuntu operativnim sustavom. Neo4j graf baza postavljena je koristeći instalacijsku datoteku sa službenih stranica Neo4j baze podataka. Nakon postavljanja poslužitelja iskorišten je DUMP lokalne baze za uvoz podataka u

novu instancu Neo4j baze podataka. Mnogo manja veličina DUMP datoteke, u odnosu na CSV, olakšala je podizanje podataka na privatni virtualni server na DigitalOcean servisu. DigitalOcean odabran je zbog svoje znatno manje cijene od ostalih servisa, specifično servisa koji nude Neo4j baze kao servis. Cijena ovakvih servisa kreće se u razini cijene jedne DigitalOcean instance ako je broj čvorova Neo4j baze oko 10000. U našem slučaju broj čvorova koji su nastali izvozom podataka bio je 2.288.490 pa je cijena za ovakav servis bila neprimjerena. Za produkcijsko korištenje trebalo bi razmotriti i specijalizirane servise za posluživanje Neo4j baze podataka. Korištene DigitalOcean instance bile su dostatne za primjer korištenja ovakvih podatka.

Neo4j instanca postavljena je na ”Neo4j” poddomenu. Ona je zaštićena korištenjem ExpressJS instance za prosljeđivanje podataka na Neo4j bazu podataka sigurnim putem potpisivanjem svih podataka. U komunikaciji sa Neo4j bazom podataka automatski generirani SSL certifikat koristi se za potpisivanje svake komunikacije. SSL certifikat dobiven je od Let’s Encrypt servisa koji nudi besplatne SSL certifikate s mogućnošću obnove svakih pola godine što je moguće programski obnavljati. Za generiranje SSL certifikata korištena je biblioteka pod nazivom ”greenlock”. Ovom bibliotekom olakšano je generiranje Let’s Encrypt certifikata te je za njegovo korištenje u ExpressJS okviru dovoljno navesti naziv poddomene za koju se certifikat generira.

Tablica 2: Pregled svih veza u napravljenoj graf bazi

Početni čvor	**Naziv veze**	**Krajnji čvor**
Index	BELONGS_TO	IndexGroup
Article	IS_INDEXED_BY	Index
Article	IS_WRITTEN_BY	Author
Article	COVERS	Field
Article	WAS_RESEARCHED_ON	Institution
Article	WAS_PUBLISHED_IN	Journal
Article	IS_DESCRIBED_BY	Keyword
Article	IS_FROM	Source
Article	OF_TYPE	Type
Article	OF_SUBTYPE	Subtype
Article	CATEGORIZED_AS	Category
Article	MADE_WITH	Project
Article	MADE_ON	Year
Article	PART_OF	Project
Article	PRESENTED_IN	Place
Article	MADE_ON	Date
Article	PARTICIPATED_AS	Participation
Article	REVIEWED_AS	Review
Article	EDITED_BY	Author
Article	PUBLISHED_BY	Publisher
Publisher	LOCATED_IN	City
Article	PRESENTED_ON	Faculty
Article	OF_STATUS	Status
Article	MENTORED_BY	Author
Article	PATENTED_AS	Patent
Patent	PATENDED_ON	Date
PatentHolder	THE_RIGHTS_TO	Patent
Article	PRESENTED_ON	Conference
Article	LEAD_BY	Author

```
type Author{
   author: String,
   wrote: [Article] @relation(name:"IS_WRITTEN_BY", direction:IN),
   edited: [Book] @relation(name:"EDITED_BY", direction:IN),
   mentored: [Article] @relation(name:"MENTORED_BY", direction:IN),
   lead: [Article] @relation(name:"LEAD_BY", direction:IN)
}
```

Slika 11: GraphQL tip podatka

4 Implementacija

4.1 Otvaranje aplikacijskog programskog sučelja GraphQL

Aplikacijsko programsko sučelje(API), je otvoreno sučelje biblioteke kroz koje korisnici mogu koristiti omogućene funkcionalnosti biblioteke. Pristup ovakvom sučelju može biti unutar aplikacije, u vidu više biblioteka koje međusobno razgovaraju preko sučelja, između procesa na istom računalu ili između različitih računala. U slučaju kada se radi o više računala uglavnom se radi o HTTP protokolu [5].

Kako bi se olakšao pristup bazi podataka s raznih klijenata, poput web aplikacija, mobilnih aplikacija i sličnih, koristi se GraphQL dodatak Neo4j baze podataka. U ovom slučaju, na Neo4j graf bazi moguće je postaviti tipove podataka koji se koriste u Neo4j instanci i njihove veze. Osim tipičnih GraphQL upita Neo4j GraphQL integracija omogućava nam korištenje određenih naprednih Neo4j funkcionalnosti, a da i dalje ne koristimo drugo programsko sučelje specifično Neo4j bazi.

U slučaju da želimo otvoriti bazu podataka gdje sama baza podataka ne nudi API pristup, bilo bi potrebno svaki od čvorova aplikacije i relacija aplikacije definirati na programskoj razini u aplikaciji koja bi posluživala API kao posrednik između klijenata i Neo4j instance baze podataka. Ako je moguće, u okviru koji koristimo za razgovor sa bazom podataka potrebno je otvoriti automatsko aplikacijsko programsko sučelje prema klijentima aplikacije. Ako nemamo tu mogućnost potrebno je osmisliti upite koje želimo otvoriti prema klijentima i implementirati ih posebno za svaki slučaj. Za proširenje mogućnosti Neo4j graf baze sa GraphQL aplikacijskim programskim sučeljem koristio se programski paket pod nazivom Neo4j-graphql koji je za nas odra-

```
CALL ga.es.queryNode('{
    "query":{
        "match":{
            "author":"Ime Autora"
        }
    }
}') YIELD node RETURN node
```

Slika 12: ElasticSearch upit kroz Neo4j

dio postavljanje GraphQL sučelja baze podataka [24]. Na temelju obrađenih HTML datoteka također je izvučena informacija o svim vezama i tipovima podataka. Na temelju ovih informacija generirani su tipovi podataka i veze koje opisuju podatke na način koji GraphQL razumije. Na slici 11 može se vidjeti GraphQL definicija autor tipa podatka. Tip podatka ”Author” ima parametre *author*,*wrote*, *edited*, *mentored* i *lead*. Author parametar predstavlja ime i prezime autora. *Wrote*, *edited*, *mentored* i *lead* su veze od ”Article” tipa podatka prema autoru sa nazivima veza iz Neo4j baze redom ”IS_WRITTEN_BY”, ”EDITED_BY”, ”MENTORED_BY” i ”LEAD_BY”. *Wrote* predstavlja vezu kada je autor napisao povezanu objavu. *Edited* predstavlja vezu na urednika objave. *Mentored* je mentor objave i *lead* voditelj objave.

4.2 Integracija sa ElasticSearch pretraživačem

Neo4j graf baza prilično je spora kod općenitog pretraživanja tekstualnih podataka po svim čvorovima i nema napredne funkcionalnosti za prepoznavanje sličnih tekstualnih unosa za pretraživanje. Već je spomenuto da se glavna prednost graf baze vidi kada znamo od koji čvorova želimo dalje dohvaćati podatke. To se postiže korištenjem indeksa i preciznih argumenata kod dohvaćanja čvorova u pretragama. Kako bi došli do čvorova koji nam trebaju može se koristiti ElasticSearch pretraživač za pretragu čvorova. ElasticSearch pretraživač indeksira tekstualne dokumente u svojoj bazi podataka. Za aktivaciju ove opcije koristila se Neo4j-to-elasticsearch biblioteka koju je razvio GraphAware [15]. Kod postavljanja Neo4j-to-elasticsearch biblioteke moguće je postaviti čvorove koje želimo prenijeti u ElasticSearch tražilicu i sinkronizirati za kasnije tekstualno pretraživanje. Ovakva integracija ne zahtijeva korištenje novog aplikacijskog programskog sučelja. ElasticSearch upite moguće je izvoditi di-

rektno u Neo4j sučelju odnosno bilo kojem sučelju koje poslužuje Neo4j, u našem slučaju to je GraphQL. Jednom kada dobijemo odgovor na određeni ElasticSearch upit, u rezultat mogu biti uključeni i čvorovi graf baze na kojoj se upit izvodi. Rezultat upita moguće je dalje koristiti u upitima graf baze direktnom integracijom ili naknadno s klijenta. Primjer poziva ElasticSearch upita kroz Neo4j bazu može se vidjeti na slici 12 gdje se može vidjeti da pozivamo ga.es.queryNode funkciju s jednim parametrom koji je tipa tekst i sadrži upit za ElasticSearch. Rezultat ovog upita je ”node”, te se po krajnjem parametru ”author” može zaključiti koji će tip podatka on i vratiti. Čvor node je zapravo čvor s tipom Author koji možemo uključiti u običan Cypher upit kao da smo ga direktno zatražili iz graf baze. Ovakvo je dohvačanje općenitih podataka mnogo brže od pretraživanja po svim čvorovima nekog tipa unutar graf baze.

Sve ostale mogućnosti ElasticSearch tražilice možemo iskoristiti na ovaj način bez direktne interakcije sa njom. ElasticSearch tražilica postavljena je na odvojenom poslužitelju zbog velike razlike u zahtjevima resursa. Neo4j baza podataka zahtijevala je veliku količinu memorije i procesorske snage, dok je ElasticSearch samo kod inicijalnog pokretanja nešto zahtjevniji na memoriji nakon čega ga je čak moguće smanjiti ako postoji ta mogućnost kod servisa virtualnog privatnog poslužitelja. Jednom kada je ElasticSearch postavljen, potrebno je dodati Neo4j-to-elasticsearch dodatak u Neo4j mapu s dodacima. Zatim je potrebno u postavkama naznačiti gdje se nalazi ElasticSearch instanca. U postavkama se također dodaje lista tipova čvorova koje želimo sinkronizirati između graf baze i ElasticSearch pretraživača. Potom je potrebno ponovno pokrenuti ElasticSearch pretraživač nakon čega će se čvorovi početi sinkronizirati.

4.3 Uvođenje protokola za sigurnost

Kako bi zaštitili podatke koji se prosljeđuju između instanci Neo4j baze, ElasticSearch pretraživača i ostalih dijelova sustava, predlaže se potpisivanje paketa u prometu između poslužitelja i klijenata poslužitelja SSL certifikatom. SSL certifikat moguće je dobiti sa servisa Let’s Encrypt koji certifikate izdaje besplatno te ih je moguće obnoviti svakih pola godine [20]. Dobiveni certifikati uključeni su u Neo4j i ElasticSearch instance koristeći tanki proxy implementiran koristeći ExpressJS okvir. Postavke instance Neo4j i ElasticSearch baza se ne mijenjaju već se sav promet

```
const GET_USERS = gql`
  query{
    user{
      name
    }
  }
`;

<Query query={GET_USERS}>
  {({loading, data, error}) => {
    if(loading) return <p>Učitavanje…</p>;
    else if(error) return <p>Došlo je do greške! {error.message}</p>;
    else return data.user.map(user => <p>{user.name}</p>)
  }}
</Query>
```

Slika 13: React Apollo GraphQL upit

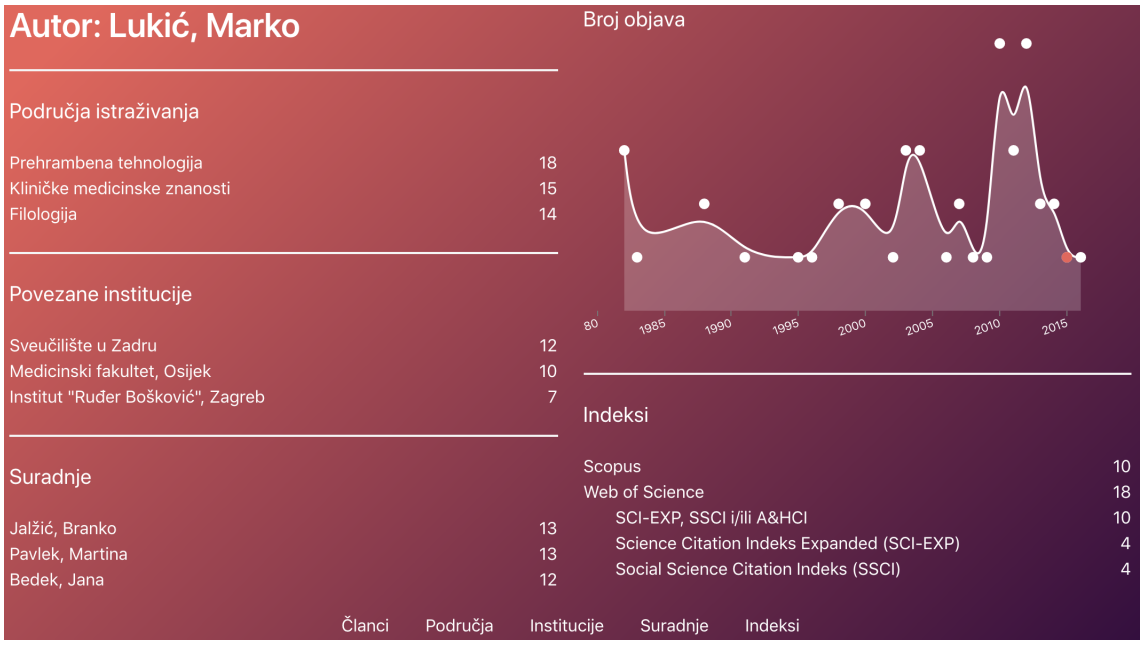
prosljeđuje kroz ExpressJS aplikaciju koja na sebi ima uključen odgovarajući SSL certifikat. Svaka poddomena ima svoj certifikat koji se učitava ovisno o ExpressJS instanci i da li poslužuje Neo4j ili ElasticSearch bazu podataka. Koristeći pristup posrednika možemo kontrolirati pristup bazi podataka, ograničiti napade i smanjiti broj mogućih upita u određenom vremenu ako je to potrebno. Baze podataka ovako nam nisu direktno izložene vanjskom prometu bez našeg nadzora [14].

4.4 Korisničko sučelje

Za izradu korisničkog sučelja koristio se ReackJS okvir. Osnovne postavke projekta moguće je generirati koristeći create-react-app biblioteku za izradu projekta iz jednog od predložaka koju održava Facebook razvojni tim [12]. Predložak koji se koristio je react-scripts-ts. Ovaj predložak nudi Miscrosoft i ima zadane postavke za korištenje Typescript jezika u kombinaciji s ReactJS okvirom [21]. Za nas postavlja tsconfig.json datoteku s postavkama potrebnim za pokretanje projekta o kojoj se više govorilo u potpoglavlju Typescript. Za kontrolu stila i ispravnosti koda, postavljena je biblioteka Tslint. Tslint je takozvani ”linter”, odnosno biblioteka za provjeru koda. Više o Tslint postavkama stila i ograničenja na kodu može se pročitati na strancima Tslint-a kojeg održava razvojni tim iz poduzeća Palantir [28].

Create-react-app biblioteka također postavlja i Webpack postavke projekta. Webpack se koristi za pripremu projekta za produkcijsko posluživanje ili lokalno pokretanje. Sastoji se od raznih postavki ovisno o okružju u kojem se projekt pokreće ili ako

Slika 14: Element za pretraživanje autora



Slika 15: Prikaz detalja za odabranog autora

se pak projekt priprema za produkciju. U slučaju lokalnog razvoja, moguće je postaviti Webpack tako da provjerava postoje li promjene na datotekama našeg projekta te se potom lokalna web stranica za razvoj ažurira. Ovo nam olakšava razvoj budući da nije potrebno pakirati aplikaciju kod svake promjene. Kada se projekt priprema za produkciju potrebno je iskoristiti Webpack postavke koje optimiziraju projekt. Optimizacije se kreću od brisanja viška koda, minimizacije biblioteka koje koristimo, u našem slučaju potrebna je i transpilacija projekta iz Typescript u Javascript kod, kompilacija stilova u slučaju kada se ne koristi CSS i ostale optimizacije ako su uključene u konfiguraciju [39]. Kada se napravi create-react-app projekt, on nam ne omogućuje uređivanje postavki Webpack-a, obzirom da su osnovne postavke dovoljne za početak razvoja. Njihovo mijenjanje se ne preporuča budući da Facebook uz ažuriranja ReactJS okvira ažurira i optimizira i Webpack konfiguraciju.

Za stilove koristio se SCSS, odnosno jezik koji dodaje nove mogućnosti na CSS datotekama. SCSS se potom prevodi u običan CSS za posluživanje u produkciju. SCSS se koristio zato što je njime moguće proširiti mogućnosti mnogih CSS biblioteka korištenjem SCSS varijabli za uređivanje postavki. U protivnom bilo bi potrebno nasljeđivati CSS klase i nadjačati identifikatore postavljene sa strane korištene biblioteke kako bi dobili određene izmijene u stilovima aplikacije [32].

Za osnovne komponente aplikacije korištena je biblioteka Bulma. Bulma ima osnovni izbor stilova koje je moguće postaviti na naše HTML komponente kako bi dobili osnovnu strukturu aplikacije sa zadanim stilovima. Za naprednije postavke može se izmijeniti SCSS varijable koje nam nudi Bulma okvir [7].

Za korištenje GraphQL aplikacijskog programskog sučelja koje smo omogućili na Neo4j bazi podataka koristila se biblioteka react-apollo. React Apollo biblioteku koristi se za pozivanje upita s klijentske aplikacije prema Neo4j instanci. Na ovaj način olakšava se korištenje GraphQL aplikacijskog programskog sučelja. React Apollo nudi nam i određene ReactJS komponente koje možemo koristiti kako bi izbjegli redundantnu logiku kod dohvaćanja podataka s interneta. Obično to podrazumijeva pokretanje upita, ažuriranje korisničkog sučelja sa početkom izvođenja upita, ispisivanje greške ako nam upit ne prođe uspješno i prikaz podataka kada je upit uspješno izve-

den. React Apollo ”Query” komponenta olakšava nam svaki od ovih koraka. Na slici 13 može se vidjeti korištenje ”Query” komponente. Definirana je GET_USERS konstanta sa vrijednošću rezultata pozivanja gql funkcije s primjerom GraphQL upita.

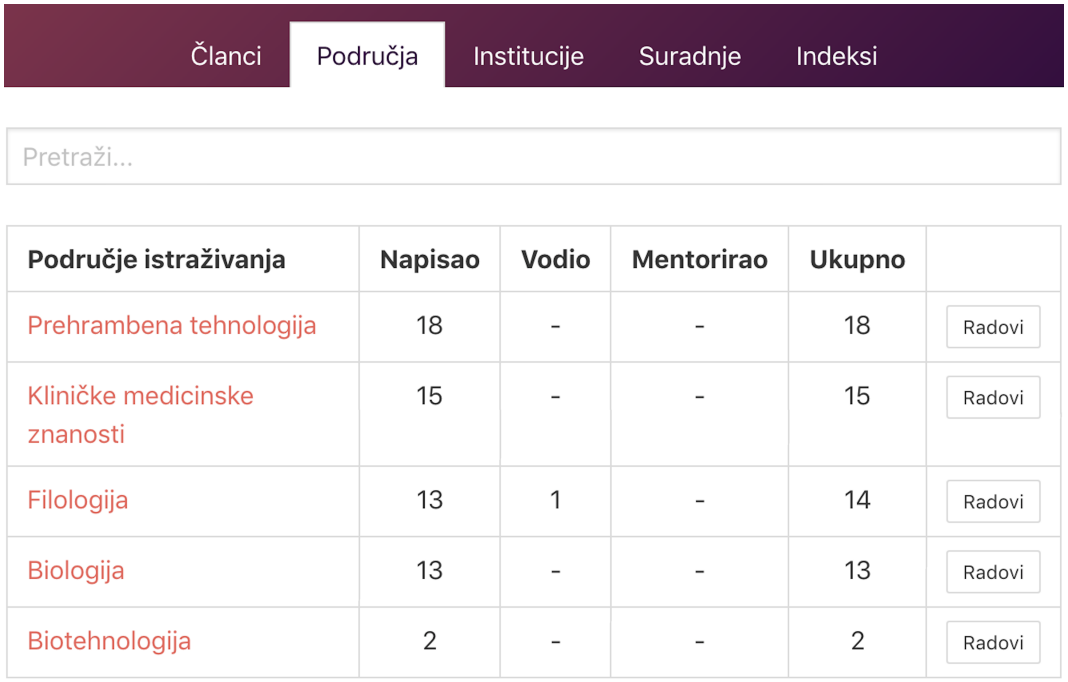
Gql je biblioteka za konstruiranje GraphQL upita. Upite je moguće izvoditi korištenjem Query komponente React Apollo paketa ili drugim paketom za pozivanje GraphQL aplikacijskog programskog sučelja. Upit u ovom slučaju dohvaća sve instance s tipom podatka user i za svakog korisnika i njegov parametar koji sadrži ime. Nakon postavljanja upita, možemo ga pozvati kao query parametar na Query komponenti pa je tako i postavljeno u djelu aplikacije gdje želimo prikazati rezultat izvođenja ovakvog upita. Query komponenta traži potkomponentu s tipom podatka koji prima tri argumenta. Prvi argument nam govori da li se upit pokrenuo. Drugi argument predstavlja rezultat upita, u ovom slučaju objekt s jednim ključem ”user” i poljem objekata od kojih svaki ima ključ ”name” s vrijednošću imena koje pripada svakom od korisnika. Treći argument predstavlja grešku ako do nje dođe, te ga možemo provjeriti i prikazati grešku ako on postoji. Kada se upit pokrene, ova unutarnja funkcija Query komponente izvodi se s postavljenim ”loading” argumentom i praznim ostalim parametrima. Dok je ”loading” argument postavljen možemo prikazati HTML element kojem bi naznačili da je upit započet, u ovom slučaju to je paragraf sa tekstom ”Učitavanje...”. Sljedeće izvođenje unutarnje funkcije sadrži ili postavljen ”error” ili ”data” argument. Ako postoji ”error” argument u prikazujemo HTML paragraf koji sadrži tekst ”Došlo je do greške!”. Kako se radi o objektu koji sadrži više informacija o nastaloj grešci, ispisujemo vrijednost pod ključem *message* kako bi prikazali odgovarajuću poruku iz ”error” objekta. Ako ne postoji ”loading” niti ”error” parametar, ”data” argument sadrži rezultat izvođenja upita te je sada moguće pripremiti HTML elemente kojima bi prikazali ”data” argument. Objekt ”data” ima jedan ključ ”user” koji sadrži sve dobivene ”user” objekte s ključem ”name”. Svi dohvaćeni korisnici se prikazuju u obliku liste HTML paragraf elemenata.

GraphQL također nudi mogućnost introspekcije. Na ovaj način moguće je u JSON obliku dobiti sve definicije podataka i vrste upita koje možemo izvoditi na GraphQL omogućenom aplikacijskom programskom sučelju. Introspekcija se osim informativno može koristiti u kombinaciji s alatima za komandnu liniju, koje također nudi Apollo

Slika 16: Prikaz radova odabranog autora

razvojni tim, za generiranje određenih datoteka koje nam mogu pomoći u razvoju. Kako Typescript podržava izradu sučelja, za razliku od Javascript-a, korisno je generirati Typescript sučelja koja definiraju izgled svih čvorova i veza GraphQL aplikacijskog programskog sučelja. Ovime dobivamo prednost da je u kodu moguće naznačiti koji tip podataka i koje parametre možemo očekivati od rezultata upita [4].

Kod prvog pokretanja aplikacije, prikazuje se polje za unos imena autora radi pretrage kao što se može vidjeti na slici 14. Iz rezultata pretrage vidi se i broj radova što je odvojeno u tri sekcije. Prva prikazuje radove za koje je iz podataka izvučeno da ih je autor i napisao. Druga sekcija pokazuje broj radova koje je autor vodio, uglavnom su to i radovi koje je mentorirao. Treća sekcija prikazuje mentorirane radove. Ispod informacija o broju radova može se vidjeti koliko je puta korišteno neko područje istraživanja u radovima ovog autora. Na sučelju napravljena je mogućnost označavanja više rezultata iz liste rezultata autora odabirom ”Uključi u pregled” ako smatramo da se radi o istom autoru. Stranica za prikaz autora potom uzima sve odabrane autore i prikazuje zbirne podatke.



Slika 17: Pregled tablice područja autorovih radova

Slika 18: Prikaz objava odabrane stavke

Na slici 15 prikazan je rezultat odabira jednog ili više autora. Na lijevoj strani vidi se tri najviše korištena područja istraživanja, povezane institucije i najčešće kolaboracije s drugim autorima. U rezultatima su uključeni napisani, vođeni i mentorirani radovi. S desne strane može se vidjeti graf koji prikazuje sve objave po godinama, ako je iz podataka o objavi bilo moguće prepoznati godinu. Na donjoj desnoj strani nalazi se popis koji pokazuje indeksiranost radova grupirano po repozitoriju kao što su Web of Science i Scopus.

Na slici 16 prikazan je donji dio stranice za prikaz detalja autora. Odabran je prikaz ”Članci” koji autorove članke prikazuje u tri kategorije, napisane, vođene i mentorirane objave. Ispod gumba za odabir kategorije moguće je upisati pojam za pretraživanje radova. Slika 17 prikazuje tablicu s područjima istraživanja u kojima je autor sudjelovao. Četiri stupca pored stupca s nazivom područja istraživanja pokazuju broj korištenja povezanog područja istraživanja prema kategoriji objave i s ukupnim brojem korištenja. Zadnji stupac ove tablice sadrži gumbe koji po odabiru otvaraju pregled radova u kojima je povezano područje istraživanja korišteno. Otvoreni pregled povezanih radova može se vidjeti na slici 18, te se on otvara u obliku skočnog prozora unutar stranice. Gumbi ”Institucije”, ”Suradnje” i ”Indeksi” u navigacijskom elementu gdje se nalaze i ”Članci” i ”Područja” koriste sličan prikaz prilagođen za odabrani resurs s mogućnošću pregleda povezanih radova.

Na slici 19 može se vidjeti primjer pregleda detalja objave. Svi prikazani detalji, osim onih specifičnih za odabrani rad, kao što je sadržaj objave, prikazani su u obliku poveznice koja korisnika vodi na stranicu za prikaz detalja. Stranica odabranog resursa trebala bi prikazivati relevantne podatne ovisno o resursu i povezanim podacima.

4.5 Vizualni prikaz podataka

Budući da su nam podaci graf baze vrlo povezani, podatke je moguće povezati i prilikom prikaza. Svaki čvor trebao bi voditi na stranicu s više informacija o odabranoj poveznici. Primjerice, poveznice s tipom ”ključna riječ”, trebale bi voditi na stranicu povezanu s odabranom ključnom riječi. Ovdje je moguće prikazati više in-

Speleološki objekti otoka Mljeta

Jezik	Engleski
Objavljeno	2011/11/18
Godina	2010

Tijekom biospeleoloških istraživanja otoka Mljeta Hrvatskoga biospeleološkoga društva, istraženi su novi speleološki objekti, u nekima su pronađeni novi podzemni prostori, a multidisciplinarni pristup istraživanju omogućio je nova saznanja o mljetskom podzemlju. Špilje i jame Mljeta različite su prema genezi, geološkoj podlozi u kojoj su nastale, morfologiji i špiljskoj fauni. U njima su ustanovljeni i fosilni ostaci te tragovi ljudskoga boravka. Prikazane su nove spoznaje o podzemlju Mljeta: Jama Međugrađen najdublja je, Galičnjak je najduža špilja, Žarnava garma i Jama za svetim Ilijom nastale su u brečama, sedrolike sige u Galičnjaku, fosilizirani beskralježnjak u Odisejevoj špilji itd. Neke od specifičnosti jesu i izrazita urušavanja nekih objekta, podzemne vodene površine i česte špilje na samoj morskoj obali – garme. Objekti u unutrašnjosti otoka često su na teško dostupnim područjima, a sve je manje domaćeg stanovništva koje zna gdje su njihovi ulazi. Buduća daljnja istraživanja uz suradnju s lokalnim stanovništvom prijeko su potrebna da bi se mljetsko podzemlje kvalitetno istražilo prije nego što lokacije objekata zauvijek padnu u zaborav.

Ključne riječi	Špilje, Jame, Speleologija, Mljet
Područja	Biologija
Kategorije	Stručni
Vrsta	Radovi u zbornicima skupova
Podvrsta	Cjeloviti rad (in extenso)
Recenzija	Domaća recenzija

Napisali Lukić, Marko, Pavlek, Martina, Jalžić, Vedran, Jalžić, Branko, Cvitanović, Hrvoje, Bilandžija, Helena, Miculinić, Kazimir

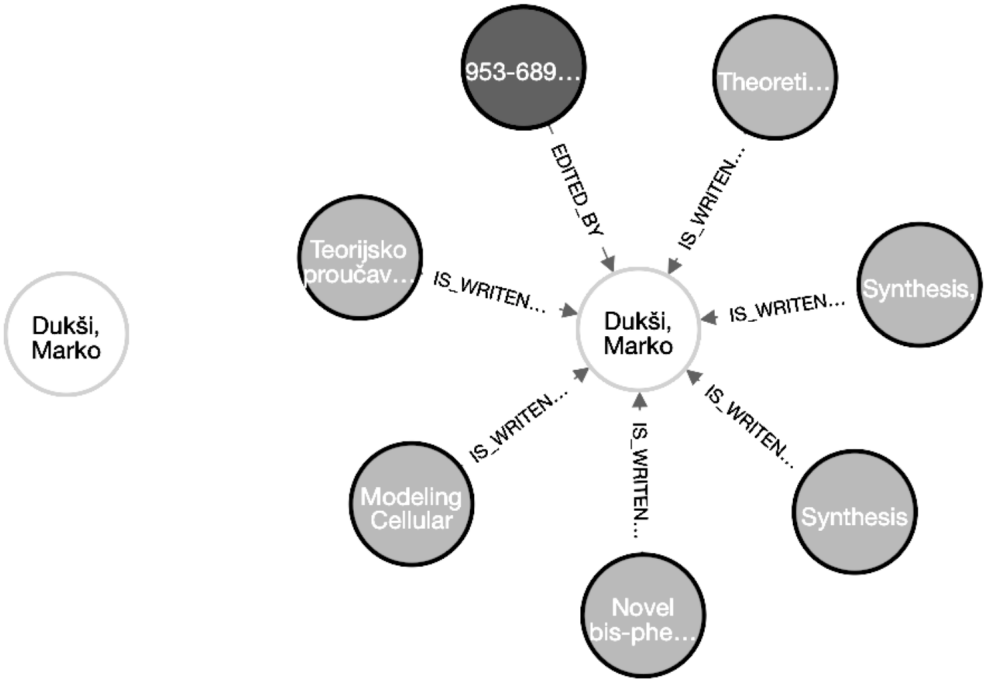
Pisano na Institut "Ruđer Bošković", Zagreb
Izvor ZBORNIK RADOVA SIMPOZIJA DANI BRANIMIRA GUŠIĆA – MLJET 2010

Mjesto Pomena, Hrvatska

Dio projekta 098-0982913-2874 - Geni i genomi: struktura, funkcija i evolucija (Helena Ćetković)

Sudjelovanje Predavanje

Slika 19: Stranica sa prikazom detalja povezanih sa objavom



Slika 20: Neo4j Desktop rezultat upita

formacija o pojavljivanju ove ključne riječi. U kojoj se vrsti radova najčešće prikazuje, koliko su kvalitetni radovi u ovom području prema broju indeksa u kojima se nalaze, koje su institucije koje najčešće koriste ovu ključnu riječ i tome slično. Osim HTML prikaza s vezama u obliku poveznica između resursa, prilično je intuitivan prikaz koji nudi mogućnost vizualizacije podataka u vidu čvorova i veza kao što je pokazano na primjeru Neo4j Desktop aplikacije.

D3 Javascript biblioteka omogućuje nam ovakvu vizualizaciju i mogućnost raznih naprednih opcija kod prikaza za izradu upita prema GraphQL aplikacijskom sučelju u realnom vremenu. U ovakvom prikazu, kada korisnik dva puta klikne na neki od čvorova, sve veze tog čvora dodaju se u postojeći upit te se time dopunjava vizualni pregled kao što je vidljivo na slici 20. S lijeve strane može se vidjeti početno stanje, što je rezultat pretraživanja jednog određenog čvora. S desne strane nalazi se rezultat upita koji povlači sve povezane čvorove prve razine kada se dva puta klikne na originalni čvor. Prikaz s čvorovima i vezama može biti prikladan kod vizualnog pretraživanja podataka i kada znamo koji put želimo pratiti u rezultatu upita. U protivnom je pregled svih podataka povezanih s određenim čvorom, primjerice člankom, prikladno prikazati

na HTML stranici uključujući i dodatne mogućnosti za korištenje poveznica na generalni prikaz povezan s odabranim čvorom ili pak kao nastavak na postojeći čvor. Ako se koristi generalni prikaz, nije potrebno pratiti od kojeg smo čvora došli na pregled trenutnog čvora, u protivnom, potrebno je zadržati stanje aplikacije prilikom navigacije između čvorova. Primjer prikaza podataka povezanih čvorova bio bi prikaz svih radova odabranog autora za određeno područje istraživanja. U tom slučaju potrebno je koristiti identifikator autora i identifikator područja istraživanja kako bi upit ograničili na čvorove odabranog autora i područja istraživanja.

5 Diskusija

5.1 Zahtjevnost sustava i moguće optimizacije

Za postavljanje sustava u produkcijsko okružje korištena su tri poslužitelja. Svaki od poslužitelja korišteni su preko servisa DigitalOcean koji nudi virtualne privante poslužitelje. Jedan poslužitelj koristi se za posluživanje SSL potpisanog prometa prema lokalnoj instanci Neo4j graf bazi. Drugi poslužitelj postavljen je na isti način sa SSL posrednikom, ali poslužuje ElasticSearch bazu podataka. Treći poslužitelj koristi se za posluživanje ReactJS aplikacije. Neo4j najzahtjevniji je od svih triju aplikacija. Potrebno je pripremiti poslužitelj koji može pohraniti cijeli kontekst upita i transakcije kako je naznačeno u dokumentaciji Neo4j graf baze. Kontekst upita i transakcije predstavlja mogućnost graf baze da podnese veću količinu podataka kao rezultat upita s obzirom na to da je kod dohvaćanja podataka potrebno pohraniti ih u memoriju poslužitelja.

ElasticSearch pretraživač nije toliko zahtjevan i podatke indeksira u vidu datotečnih indeksa datotečnog sustava poslužitelja što znači da ih nije potrebno unositi u memoriju prilikom izvođenja upita. Veća količina memorije potrebna je kod inicijalne sinkronizacije Neo4j graf baze i ElasticSearch pretraživača. Nakon sinkronizacije, veličina ElasticSearch poslužitelja može se smanjiti s obzirom na to da daljnja sinkronizacija isključivo uzima razlike za sinkronizaciju i ne radi se velikim količinama podataka kao što je to kod inicijalnog postavljanja. Za posluživanje ReactJS aplikacije koristi se najmanja veličina poslužitelja s obzirom na to da je za posluživanje potrebna mala instanca ExpressJS servera kao posrednika koji ima SSL certifikat i statične datoteke koje on poslužuje. Poslužitelji namijenjeni posluživanju Neo4j i ElasticSearch instanci postoje, ali su mnogo skuplji u odnosu na DigitalOcean generičke instance poslužitelja. Prednost dediciranih poslužitelja je i to da je održavanje poslužitelja obično napravljena od strane poduzeća koje nudi tu uslugu te se nije potrebno brinuti o sistemskim zadacima poslužitelja nego samo razvojem.

5.2 Proširivanje opsega podataka

Povezivanje podataka koji su i logički povezani omogućava nam jednostavniju vizualizaciju i bolje performanse kod analize podataka ovih razmjera. Jednostavnije je postići upite koji se u tradicionalnim relacijskim bazama zovu JOIN metode što je posebno prikladno kada želimo prikazati podatke povezane s podacima koje korisnik traži. Svi podaci u sustavima gdje znamo da će biti potrebno sagledati podatke iz svake perspektive, svakog pogleda podataka, preporučuje se pristup s graf bazom. Ona nam omogućava performantno pristupanje svim vezama i smjerovima u pregledu podataka bez obzira da li smo na njih računali kod izrade baze podataka. Jedino što je potrebno jest povezati podatke na odgovarajući način preko čvorova koji ih povezuju. Kod tradicionalnih relacijskih baza manje je potrebna priprema podataka, ali su mnogo sporiji odazivi na upite u kasnijim fazama razvoja u slučaju aplikacija za analizu podataka. Također je potrebno slaganje mnogo zahtjevnijih upita kako bi se postigla ista razina kompleksnosti rezultata kao što je to kod graf baze.

5.3 Mogućnost primjene strojnog učenja

Kako se radi o prilično ne strukturiranim podatcima određene je podatke potrebno klasificirati i obraditi kada se prepozna određena ponavljajuća greška u podacima ili iskoristiti naprednije tehnike određivanja strukture podataka u ovakvim tekstualnim zapisima. Jedan od primjera kako se može optimizirati prepoznavanje imena i prezimena u tekstualnim podacima jest i Sandford CoreNLP. Radi se o grupi alata za obradu prirodnog jezika. Jedan od alata može se koristiti za tokenizaciju teksta. Tokenizacijom dobivamo prijedloge moguće strukture rečenica u tekstu. Kod tekstova gdje imamo više autora u redu, moguće je dobiti dobru aproksimaciju o tome što je ime, prezime ili inicijal autora [35]. Kako se u Sandford CoreNLP radi o generalnoj primjeni alata za obradu prirodnog jezika, predlaže se izrada specijaliziranih alata koje bi bilo moguće naučiti prepoznavati autore u specifičnim slučajevima gdje se u tekstu pojavljuju redovi autora odvojeni raznim znakovima. Po strukturi teksta moguće je zaključiti što se koristi kao znak za odvajanje autora, primjerice znamo da će se taj znak ponavljati isti broj puta kao i broj autora manje jedan. Jedan od načina za implementaciju algoritama koje možemo nanovo koristiti za provjeru većeg broja podataka su neuron-

ske mreže. Dovoljnim brojem primjera ulaza i izlaza neuronsku mrežu možemo naučiti što je potrebno napraviti nad tekstualnim podacima kako bi se dobio željeni rezultat, odnosno razdvojeni autori iz teksta.

5.4 Pregled doprinosa rada

Ovim radom nastojalo se omogućiti drugim stranama otvoreno i jednostavno korištenje podataka. Podaci su povezani na osnovu onoga što predstavljaju u detaljima objave. Nije potrebno pripremati posebne upite za korištenje ovakvog otvorenog aplikacijskog programskog sučelja, osim u slučaju kada se podatke pretražuje generalnim putem, kao primjerice kod pretraživanja autora ili sličnog. Ako se ne radi o pretraživanju nego je poznato koje čvorove će se koristiti, graf baza je dovoljno učinkovita za posluživanje raznih upita. Za generalno pretraživanje predložena je ElasticSearch baza koja zbog učinkovitog indeksiranja općih tekstualnih podataka i visokih performansi kod pretraživanja istih. Kako se graf baza ne bi preopteretila predlaže se ograničavanje upita na više manjih upita umjesto jednog većeg kako se ne bi iskoristila sva memorija poslužitelja u jednom od upita. Ovaj pristup ne ograničava mogućnosti klijenata aplikacije s obzirom na to da postoje razne biblioteke koje optimiziraju pozive prema GraphQL aplikacijskom sučelju, uključujući i Apollo biblioteka. Ovakve biblioteke brinu se o tome da se što više upita spoji u jedan ako se radi o većem broju upita u kratkom vremenu. Rezultati upita tada se dopunjavaju iz lokalnog datotečnog sustava ili iz spremišta preglednika. Odabrano spremište podataka ažurira se kod svakog dohvaćenog rezultata nakon izvođenja upita na GraphQL aplikacijsko programsko sučelje. Umjesto preuzimanja svih podataka o svim radovima, predloženo je otvaranje podataka za korištenje u analizi akademskih radova u svrhu određivanja učinkovitosti i ostalih značajki povezanih s autorima, njihovim radovima, koautorima i ostalim povezanim podacima.

Kod određivanja akademske učinkovitosti poželjno je imati otvorene podatke o čimbenicima koji su nam potrebni za analizu, te su iz tog razloga podaci ovog sustava otvoreni na korištenje uz određene naputke kako ih iskoristiti. Uz površno znanje izrade Cypher upita koji se koriste u Neo4j bazi moguće je koristiti prilagođeno vizualno sučelje graf baze za osnovnu analizu podataka. U slučaju da se radi o naprednijoj analizi dovoljno je iskoristiti jednu od biblioteka koje nam omogućavaju integraciju sa

Neo4j graf bazom direktno ili preko GraphQL aplikacijskog programskog sučelja i pripremiti upite koji se potom prikazuju u klijentskoj aplikaciji. Nije potrebna priprema poslužiteljskog softvera za posluživanje upita s obzirom na to da su podaci otvoreni i da su poslužitelji graf baze već prilagođeni za obradu svih upita koje je moguće izvoditi na graf bazi.

Zaključak

Nestrukturirani podaci i nepristupačnost servisa koji pohranjuju podatke o akademskim radovima samo neki su od problema kod analize podataka ove vrste. Neki servisi nude preuzimanje i pregled podataka, ali su ograničeni količinom koju je moguće preuzeti. Drugi ne dopuštaju pristup bez određenih novčanih pretplata. Treći pak uopće ne podržavaju preuzimanje podataka. Bez obzira na izvor podataka na svakom servisu postoji određena razina greške koja postoji u podacima budući da nisu strukturirani i moderirani od treće strane.

Nije razumljivo očekivati potpunu točnost podataka, ali moguće je napraviti korisničko sučelje koje u dovoljnoj mjeri ograničava dodavanje nestrukturiranih podataka da se razina greške smanji. Ako dođemo do podataka, spajanje podataka iz više izvora je vrlo teško s obzirom na moguće greške u podacima i s obzirom na to da razni servisi različito tretiraju pohranu istih podataka. Potrebna je standardizacija podataka servisa koji nude informacije o akademskim radovima prije analize podataka kako bi oni bili potpuni. Nakon uspješnog prikupljanja podataka potrebno je analizirati prikupljeno i napraviti zajednički okvir svih podataka što se može postići raznim metodama testiranja očekivane strukture podatka u odnosu na ulazne podatke.

Daljnja obrada podataka zahtjeva korištenje naprednih algoritama za analizu i prepoznavanje uzoraka u tekstualnim podacima. Ako je uspješno postavljena struktura podataka, potrebno je podatke ograničiti u okvire već ustanovljenih struktura kako se ne bi odstupalo od postojeće strukture. Ovo je moguće napraviti pomno osmišljenim korisničkim sučeljem za unos podataka koje bi svaki od tipova podataka podrobno provjeravao prije unosa upozoravajući na nekonzistentnosti u formatu podataka. Ovime se veći dio posla prenosi na osobu koja te podatke unose, ali se ujedno i osigurava stabilna struktura podataka. Ukoliko se odlučimo na uključivanje dodatnih servisa i podataka iz raznih izvora kompleksnost se povećava utoliko što je potrebno prilagoditi podatke ne samo za unos u postojeću strukturu baze podataka već i prilagoditi ih s obzirom na postojeće podatke kako bi izbjegli redundantnost.

Predloženi su i neki od oblika općih algoritama za prikaz podataka iako je zbog otvorenosti i fleksibilnosti sustava moguće napraviti prilagođena sučelja i analize ovisno o slučaju. Fokus rada bio je u prilagođavanju strukture i otvaranju pristupačnog programskog sučelja. Podaci o znanstvenim radovima pa tako i povezana softverska rješenja, trebala bi biti otvorenog tipa. Na ovaj način u zajednici kojoj je u interesu diseminacija znanja zaista i potićemo kolaboraciju. Ovim radom pokazan je jedan od mogućih pristupa pri izradi aplikacijskog programskog sučelja namijenjenog korištenju od trećih strana. Poseban naglasak postavljen je na odabir prikladnih i performantnih tehnologija kako bi se što efikasnije veći dio posla prebacio na baze podataka. Primjer je prikazan na nacionalnoj razini sa trenutno dostupnim podacima.

Literatura

[1] Adobe Corporate Communications (2017). Flash and the future of interactive content. [Online; pristupljeno 02.09.2018.].

[2] Agregator hrvatskih repozitorija i arhiva (2018). Agregator hrvatskih repozitorija i arhiva. [Online; pristupljeno 04.09.2018.].

[3] Anthony, A., Nathaniel, M., and Ari, L. (2017). Fullstack react: The complete guide to reactjs and friends.

[4] Apollo (2018). Command line tool for development and production apollo workflows. [Online; pristupljeno 18.09.2018.].

[5] Blanchette, J. (2008). The little manual of api design.

[6] Bray, T. (2017). The javascript object notation (json) data interchange format. Technical report.

[7] Bulma (2018). Everything you need to create a website with bulma. [Online; pristupljeno 18.09.2018.].

[8] Bynens, M. (2018). What is v8? [Online; pristupljeno 03.09.2018.].

[9] Clarivate (2018). Web of science - terms of use. [Online; pristupljeno 19.09.2018.].

[10] Crockford, D. (2008). *JavaScript: The Good Parts: The Good Parts*. " O'Reilly Media, Inc.".

[11] CROSBI (2018). Uvjeti korištenja. [Online; pristupljeno 06.09.2018.].

[12] Facebook (2018a). Create react apps with no build configuration. [Online; pristupljeno 18.09.2018.].

[13] Facebook (2018b). Introducing jsx. [Online; pristupljeno 04.09.2018.].

[14] Fowler, S. (2017). *Microservices in Production: Standard Principles and Requirements*. O'Reilly Media.

[15] GraphAware (2018). Graphaware framework module for integrating neo4j with elasticsearch. [Online; pristupljeno 17.09.2018.].

[16] Haverbeke, M. (2014). *Eloquent JavaScript*. No Starch Press.

[17] Jansen, R. H. (2015). *Learning TypeScript*. Packt Publishing Ltd.

[18] L, R. (2016). *Express.js: Guide Book on Web Framework for Node.js*. CreateSpace Independent Publishing Platform.

[19] Lehmer, J. (2016). *Ten Step to Linux Survival: Essentials for Navigating the Bash Jungle*. O'Reilly Media.

[20] Let's encrypt (2018). Let's encrypt is a free, automated, and open certificate authority. [Online; pristupljeno 17.09.2018.].

[21] Microsoft (2018a). A starter template for typescript and react with a detailed readme describing how to use the two together. [Online; pristupljeno 18.09.2018.].

[22] Microsoft (2018b). tsconfig.json. [Online; pristupljeno 04.09.2018.].

[23] Module counts (2018). Module counts. [Online; pristupljeno 03.09.2018.].

[24] Neo4j (2018). Neo4j and graphql. [Online; pristupljeno 17.09.2018.].

[25] Nodej.js (2018). Node.js v10.9.0 documentation. [Online; pristupljeno 03.09.2018.].

[26] Node.js (2018). About node.js. [Online; pristupljeno 03.09.2018.].

[27] Open archives (2018). Oai-pmh registered data providers. [Online; pristupljeno 04.09.2018.].

[28] Palantir (2018). An extensible linter for the typescript language. [Online; pristupljeno 18.09.2018.].

[29] Redis (2018). Redis. [Online; pristupljeno 06.09.2018.].

[30] Riehle, D. and Gross, T. (1998). Role model based framework design and integration. In *ACM SIGPLAN Notices*, volume 33, pages 117–133. ACM.

[31] Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph databases*. " O'Reilly Media, Inc.".

[32] Sass (2018). Css with superpowers. [Online; pristupljeno 18.09.2018.].

[33] Shukla, P. and N, S. (2017). *Learning Elastic Stack 6.0*. Packt Publishing.

[34] Stackoverflow (2018). Developer survey results 2018. [Online; pristupljeno 03.09.2018.].

[35] Standford (2018). Stanford corenlp – natural language software. [Online; pristupljeno 18.09.2018.].

[36] Syed, B. (2017). *Typescript Deep Dive*. ARTPOWER International PUB.

[37] Techempower (2018). Best plaintext responses per second. [Online; pristupljeno 03.09.2018.].

[38] Topic, D. (2016). Moving to a plugin-free web. [Online; pristupljeno 02.09.2018.].

[39] Webpack (2018). Bundle your scripts. [Online; pristupljeno 18.09.2018.].

[40] Wieruch, R. (2017). *The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js*. CreateSpace Independent Publishing Platform.

Popis slika

Popis tablica

Sažetak

Nepristupačnost podataka o akademskim radovima predstavlja poseban problem za analizu i pregled podataka. Servisi koji poslužuju ovakve podatke obično nemaju sve potrebne podatke, ne prate određenu strukturu podataka ili imaju druga ograničenja poput broja podataka koje je moguće preuzeti, vremenski okviri u kojima je podatke moguće preuzeti, posebni zahtjevi na pristup podacima, zabrana izvođenja analize nad preuzetim podacima i tome slično. Cilj ovog rada bio demonstrirati jednostavnije i pristupačnije rješenje za korištenje podataka o akademskim radovima u svrhu analize i pregleda podataka.

Odabrane su tehnologije koje su efikasnije u radu s podacima koji su visoko povezani. Za pohranu podataka odabrana je graf baza u kojoj su veze jednako važne kao i podaci. Na ovaj način jednostavnije je i performantnije dohvaćanje povezanih podataka preko njihovih veza za razliku od tradicionalnih relacijskih baza podataka. Za pretraživanje podataka u graf bazi predložena je tehnologija ElasticSearch koja indeksira sve podatke graf baze. Indeksirani podaci koriste se za efikasno pretraživanje svih podataka. Generalno pretraživanje podataka jedna je od mana graf baza koja se na ovaj način može otkloniti. Za izradu web aplikacije odabrane su tehnologije kojima se olakšava razvoj. Podešeno je prepoznavanje tipova podataka u Javascript dinamičkom jeziku. Napravljen je pregled načina na koji je moguće iskoristiti napravljeno aplikacijsko programsko sučelje graf baze.

Predloženo je prikupljanje, obrada i pohrana podataka. Za prikupljanje podataka korišteni su skriptni jezici. Oni ubrzavaju razvoj privremenih sustava svojom pristupačnošću i visokom razinom apstrakcije. Obrada podataka napravljena je nad prikupljenim podacima u više koraka. Nakon analize prikupljenih podataka, svi podaci koje je bilo moguće dobiti iz podatkovnog izvora pretvoreni su u predloženi format. Opisane su i određene metode za ubrzavanje obrade podataka.

Za analizu podataka o akademskim radovima potrebno je uzeti u obzir što više izvora podataka. Na ovaj način podaci će biti cjeloviti. Preuzete podatke potrebno je prilagoditi zajedničkoj strukturi. Razlike u strukturi izvora mogu predstavljati problem

kod prilagodbe u postojeću strukturu. Za naprednije parsiranje podataka predlažu se metode strojnog učenja.

Sustav je predloženo otvoriti za korištenje kako bi potakli kolaboraciju i daljnji razvoj. Opisano je postavljanje sustava u produkcijsko okružje, osiguravanje prometa podataka i različiti načini za posluživanje podataka.

Ključne riječi: analiza podataka, graf baze, aplikacijska programska sučelja, dinamički jezici

Abstract

The lack of availability of information on academic papers is a problem, especially if they are to be used for analysis and data overview. Services that provide such information usually do not have all the information required, do not follow a particular data structure, or have other limitations such as the amount of data that can be downloaded, time frames within which data can be retrieved, data access requirements, disallowing data analysis on downloaded data, and the like. The aim of this paper was to provide a simpler and more accessible solution for using academic data for analysis and data overview.

Selected technologies are more efficient in working with data that is highly related. For data storage graph database was used where data relationships are as important as data. In this way, it is simpler and more efficient to retrieve connected data over their relationships than traditional relational databases. To search data in the graph database, it is suggested that ElasticSearch technology is used for indexing data. Indexed data is used to efficiently search all data. General data search is one of the downsides of graph databases that can be avoided in this way. For web application development, technologies that make development easier were selected. Data type detection was added to the Javascript dynamic language. An overview of integrations with created application programming interface was presented.

Data collection, processing and storage methods are suggested. Scripting languages were used to collect data. They accelerate the development of temporary systems with their accessibility and high level of abstraction. Collected data was processed in several steps. After analyzing the data collected, all the data that was available from the data source was converted into the proposed format. Some methods for accelerating data processing are also described.

For the analysis of academic data, it is necessary to take into account multiple data sources. This way, the data will be complete. The data collected must be adapted to its common structure. Differences in the source data structure can be a problem in adapting to the existing structure. For more advanced parsing of data, machine

learning methods are suggested.

The system was proposed to be opened for use to encourage collaboration and further development. System setup for production environment was described as well as data transfer security and various ways of serving data.

Keywords: data analysis, graph databases, application programming interfaces, dynamic languages