



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Невена Глигоров

***DebugIt*: студентска платформа за
дискусију са интеграцијом
Elasticsearch претраге**

ДИПЛОМСКИ РАД
- Основне академске студије -


Нови Сад, 2024.

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска публикација
Тип записа, ТЗ:	Текстуални штампани документ
Врста рада, ВР:	Завршни-bachelor рад
Аутор, АУ:	Невена Глигоров
Ментор, МН:	доц. др Дуња Врбашки
Наслов рада, НР:	<i>DebugIt</i> : студентска платформа за дискусију са интеграцијом <i>Elasticsearch</i> претраге
Језик публикације, ЈП:	Српски (ћирилица)
Језик извода, ЈИ:	Српски / Енглески
Земља публикавања, ЗП:	Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2024.
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Факултет Техничких Наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад
Физички опис рада, ФО: <small>(поглавља/страна/ цитата/табела/слика/графика/прилога)</small>	6/41/44/0/21/0/0
Научна област, НО:	Електротехничко и рачунарско инжењерство
Научна дисциплина, НД:	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, ПО:	<i>Elasticsearch</i> механизам претраге, <i>text analysis</i> процес, <i>DebugIt</i> платформа
УДК	
Чува се, ЧУ:	Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	Потребно је место на коме ће студенти моћи да поставе и пронађу питања везана за имплементациони или дискусиони проблем. У ту сврху је креирана <i>DebugIt</i> платформа. Користећи <i>Elasticsearch</i> механизам претраге, студенти су у могућности да веома брзо и лако пронађу одговор на било које питање.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: др Драган Иветић, редовни професор
	Члан: др Ксенија Дорословачки, редовни професор
	Члан, ментор: др Дуња Врбашки, доцент
	Потпис ментора

KEY WORDS DOCUMENTATION

Accession number, ANO :		
Identification number, INO :		
Document type, DT :	Monographic publication	
Type of record, TR :	Textual material, printed	
Contents code, CC :	Bachelor thesis	
Author, AU :	Nevena Gligorov	
Mentor, MN :	Dunja Vrbaški, PhD, assist. prof.	
Title, TI :	Debuglt: student discussion platform with integration of Elasticsearch mechanism	
Language of text, LT :	Serbian (cyrillic script)/Serbian (latin script)	
Language of abstract, LA :	Serbian/English	
Country of publication, CP :	Serbia	
Locality of publication, LP :	Vojvodina	
Publication year, PY :	2024	
Publisher, PB :	Author reprint	
Publication place, PP :	Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad	
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	6/41/44/0/21/0/0	
Scientific field, SF :	Electrical and computer engineering	
Scientific discipline, SD :	Applied computer science and informatics	
Subject/Key words, S/KW :	Elasticsearch search engine, text analysis, Debuglt platform	
UC		
Holding data, HD :	Library of the Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad	
Note, N :		
Abstract, AB :	<p>There needs to be a place where students can post and find questions related to an implementational problem or a discussion. For that purpose, Debuglt platform was created. Using Elasticsearch search engine, students can find answers to any question quickly and easily.</p> <p>.</p>	
Accepted by the Scientific Board on, ASB :		
Defended on, DE :		
Defended Board, DB :	President:	Dragan Ivetić, PhD, prof.
	Member:	Ksenija Doroslovački, PhD, prof.
	Member, Mentor:	Dunja Vrbaški, PhD, assist. prof.
		Menthor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА	Лист/Листова:
		5/41

(Податке уноси предметни наставник - ментор)

Врста студија:	<input type="checkbox"/> Основне академске студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	др Милан Рапаић

Студент:	Невена Глигоров	Број индекса:	RA 7/2020
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	др Дуња Врбашки		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; - литература 			

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

<p align="center"><i>DEBUGIT: СТУДЕНТСКА ПЛАТФОРМА ЗА ДИСКУСИЈУ СА ИНТЕГРАЦИЈОМ ELASTICSEARCH ПРЕТРАГЕ</i></p>

ТЕКСТ ЗАДАТКА:

Имплементирати систем који ће омогућити дискусију међу студентима и наставницима на факултету у облику платформе за питања и одговоре. Посебно обратити пажњу на претаргу. Користити <i>.net</i> и <i>Elasticsearch</i> технологије. Објаснити начин функционисања коришћеног механизма претраге. Навести изазове у реализацији. Извести закључке и идентификовати могућа унапређења.

Руководилац студијског програма:	Ментор рада:

Примерак за: <input type="checkbox"/> - Студента; <input type="checkbox"/> - Ментора
--

Spisak skraćenica

FTP - *File transfer protocol*

QA - *questions and answers*

GB - *gigabytes*

CRUD - *create, read, update, delete*

TF - *term frequency*

IDF - *inverse document frequency*

Sadržaj

1. Uvod.....	8
2. Istorijat.....	9
3. Značaj i primena.....	12
3.1 Postojeća rešenja	12
4. <i>Elasticsearch</i> mehanizam pretrage.....	14
4.1 Šta je <i>Elasticsearch</i> mehanizam pretrage i kako je nastao?	14
4.2 Struktura <i>Elasticsearch</i> mehanizma pretrage.....	14
<i>Index</i>	14
<i>Shard</i>	15
<i>Segment</i>	16
<i>Cluster</i> i <i>node</i>	17
<i>Index template</i> i <i>data mapping</i>	18
<i>Alias</i>	18
4.3 Način rada <i>Elasticsearch</i> mehanizma pretrage.....	19
<i>Text analysis</i>	20
<i>Stemming</i>	21
<i>Stop words</i>	22
<i>Synonyms</i>	22
<i>Scoring mehanizam</i>	23
<i>Fuzzy search</i>	24
5. <i>DebugIt</i>	27
5.1 Detalji implementacije	27
Konfiguracija <i>Elasticsearch</i> mehanizma i <i>index</i> komponenti.....	27
Implementacija <i>CRUD</i> metoda.....	29
Implementacija pretrage	30
Primeri korisničkog interfejsa	31
5.2 Izazovi u realizaciji.....	35
5.3 Potencijalna unapređenja	36
6. Zaključak.....	37
Literatura	38
Podaci o kandidatu.....	41

1. Uvod

Internet je ogromno more prepuno informacija. Problem pronalaska odgovarajućeg skupa informacija je skoro nerešiv bez upotrebe pomoćnih alata, odnosno, mehanizama pretrage. Mehanizmi pretrage predstavljaju moćan alat uz pomoć kojeg se korisnicima nudi skup informacija koje imaju najviše sličnosti sa otkucanim upitom.

Platforma *DebugIt* predstavlja riznicu najraznovrsnijih studentskih implementacionih i diskusionih pitanja i odgovara. Cilj ove platforme je da uz pomoć *Elasticsearch* mehanizam pretrage studentima olakša pronalaženje pitanja i odgovora koji su im potrebni.

U poglavlju 2 se nalazi kratka istorija najznačajnijih mehanizama pretrage, zajedno sa informacijama o njihovim kreatorima i načinima rada samih mehanizama.

U narednom poglavlju se nalazi opis problema i ideje za kreiranje i korišćenje *DebugIt* platforme, uz naznačene benefite njenog korišćenja.

Poglavlje 4 sadrži teorijske osnove *Elasticsearch* mehanizma pretrage, sa detaljno objašnjenom strukturom i načinom rada mehanizma.


Potom se u poglavlju 5 nalazi način rada *DebugIt* platforme. Navedene su korišćenje tehnologije, primeri koda sa potrebnim konfiguracijama i primeri interfejsa.

Na kraju su dati zaključci.

2. Istorijat

Alan Emtidž, administrator sistema na Departmanu za Informacione tehnologije i student kanadskog Univerziteta McGill, čije je sedište u Montrealu, u septembru 1990. godine objavljuje prvi Internet mehanizam pretrage (*eng. search engine*) pod nazivom *Archie*. Ime *Archie* potiče od reči „*Archive*“, po uzoru na dugačak i mukotrpan proces pretrage kroz koji je Alan prolazio kako bi razvio svoj softver [1].

Archie korisnički interfejs je poprilično jednostavan, kao što je prikazano na slici 1, uzimajući u obzir da je njegova glavna namena pretraživanje fajlova na FTP sajtovima koji se nalaze na Internetu.

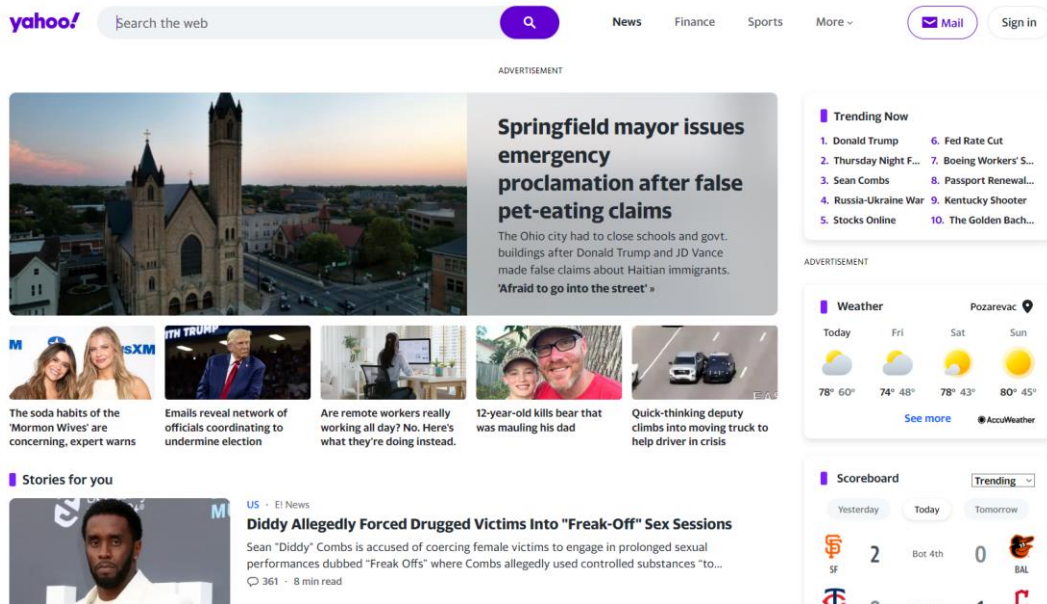


Slika 1. Archie interfejs [2]

Univerzitet je pred Alana postavio težak zadatak u to vreme, a to je da kreira softver uz pomoć kog je moguće povezati se na Internet. Novokreirani *Archie* je predstavljao indeks FTP fajlova i njegovi korisnici su uz pomoć jednostavnih upita mogli da ih pretražuju. Ovi upiti su morali da budu dosta precizni, kako bi korisnici dobili ono što su želeli, pošto *Archie* nije imao podršku za upite napisane prirodnim jezikom (*eng. natural language*), niti je indeksirao sadržaj fajlova. Kako nije imao mogućnost da da korisnicima informaciju o tome šta se nalazi u fajlovima koji su im ponuđeni kao rezultat pretrage, korisnici su te fajlove morali da skinu (*eng. download*) na svoj računar i pogledaju njihov sadržaj.

Do 1993. *Archie* je postao veoma popularan i korišćen od strane drugih univerziteta širom sveta [3].

Nekoliko godina i mehanizama pretrage kasnije, apsolutnu dominaciju u Internet svetu ima *Yahoo*. Bilo da je ime *backronym* za „*Yet Another Hierarchically Organized Oracle*“ ili „*Yet Another Hierarchical Officius Oracle*“, njegova struktura je jednoznačna, takva da predstavlja direktorijum koji se sastoji od Internet sajtova, prilikom čega su sajtovi organizovani hijerarhijski. Primer interfejsa videti na slici 2 [4].

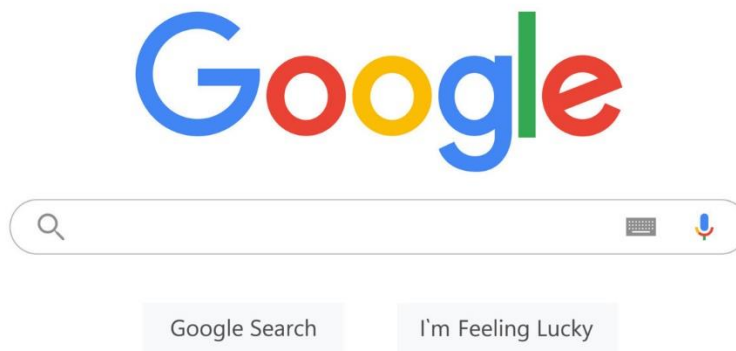


Slika 2. Yahoo! interfejs

Yahoo se temelji na ideji dvojice studenata sa Stanford univerziteta, Džeri Jang i Dejvid Filo, čija je zamisao bila da naprave direktorijum pod nazivom „Jerry and David's Guide to the World Wide Web“, uz pomoć kojeg bi navigirali do svojih omiljenih sajtova.

Zahvaljujući internet pregledaču Navigator koji je postavio link do njihovog projekta na vrh svoje stranice, direktorijum koji su napravili Džeri i Dejvid polako postaje sve popularniji, tačnije, postaje toliko popularan da odlučuju da svoju ideju pretoče u biznis.

Iako je postojalo dosta drugih mehanizama pretrage, poput: Lycos, AskJeeves, Excite, AOL, AltaVista, Infoseek i drugih, Yahoo je stajao na samom vrhu i niko mu nije bio ni blizu. 1998. godine dvojica studenata odlaze u Yahoo sa ponudom da im prodaju svoj biznis za milion dolara, međutim Yahoo ih je odbio. Ono što tada Yahoo nije znao jeste da je odbio budućeg giganta koji i dan danas vlada Internetom, čiji primer interfejsa je moguće videti na slici 3 [5].



Slika 3. Google interfejs [6]

Google je polako postajao glavni protivnik *Yahoo* kompanije i *Yahoo*, imajući to u vidu, ponudio je *Google* kompaniji da integriše njihov mehanizam pretrage u okviru svoje Internet stranice. Ovim potezom, *Google* je dobio promociju od strane kompanije *Yahoo*, kojoj to nije bila zamisao, i time se polako peo ka samom vrhu.

Leri Pejđž i Sergej Brin su bili studenti Stanford Univerziteta kada su napravili prvi mehanizam pretrage koji je radio po principu analiziranja linkova tipa *backlink* (linkovi u okviru stranica koji upućuju na druge stranice) i na osnovu njih je određivao koliko su relevantne pojedine stranice na Internetu. Projekat je dobio naziv *Backrub*, upravo zbog načina rada ovog mehanizma.

Želeli su da „organizuju sve informacije sveta i da ih učine univerzalno pristupačnim i korisnim“ i u tom duhu su *Backrub* preimenovali u *Google*, što predstavlja matematički izraz za broj 1 koji je praćen sa 100 nula (1 googol) [7].

Osnova *Google* mehanizma pretrage se zasniva na *PageRank* algoritmu, koji koristi opisani koncept *Backrub* mehanizma. Drugim rečima, ovaj algoritam istražuje koliko drugih sajtova referencira neku stranicu da bi odredio koliko je ta stranica zaista relevantna.

Naročitu pažnju velikog broja kompanija je privukao *Google* algoritam za reklame. Algoritam koristi reči koje korisnik unese tokom pretrage kako bi oforimo skup reklama koje mu treba prikazati [7][8][9].

Kada je potrebno nešto pretražiti, ljudi će za to obično reći da je to potrebno „izguglati“, što već dovoljno govori o tome gde se danas nalazi *Google* i njihov mehanizam pretrage. Pored ovog mehanizma, među popularnim mehanizmima su još i: *Microsoft Bing*, *Yahoo*, *Baidu* i drugi.

3. Značaj i primena

Na fakultetima širom sveta prolazi i smenjuje se bezbroj studenata. Svaki od njih je tokom svojih studentskih dana uradio barem nekoliko projekata, što na zahtev predmetnog profesora, što iz radoznalosti i želje za unapređivanjem. Iako različiti, jedna stvar je zajednička svim studentima, a to je da uvek pitaju za savete i iskustva nekog od svojih starijih kolega. Iz tih razloga su tokom godina nastajali različiti *Github* repozitorijumi, *DropBox* linkovi, kompresovani folderi i slično, puni materijala uz pomoć kojih je lakše položiti ispite.

Ideja podela materijala novim generacijama jeste vrlo dobra i odlično se pokazala kada je u pitanju teorijski deo ispita. Imajući to u vidu, nameće se jedno vrlo prosto pitanje, šta se dešava sa projektima koje su starije kolege radile? Projekti ne treba da se recikliraju godinama, zbog čega profesori i asistenti svake godine smišljaju nove projekte za nove generacije studenata i time već urađeni projekti postaju u dovoljnoj meri neupotrebijivi.

Međutim, svaki od tih projekata je prilikom izrade imao po neki problem (*eng. bug*), a dosta njih je imalo slične ili iste probleme. Studenti koji su radili na njima su možda potrošili malo više vremena dok su pronašli adekvatno rešenje u moru informacija koje se danas nalaze na Internetu. Iz generacije u generaciju, studenti imaju slične probleme prilikom izrade svojih projekata bez uvida u to da li je neko od njihovih ili starijih kolega imao sličan problem, ali i bez načina kako ga je rešio.

Jedan od potencijalnih rešenja koje se koristi trenutno jeste da se uvede po jedan dokument sa pitanjima i odgovorima (*eng. QA sheet*) za svaku generaciju i svaki predmet. U okviru ovog dokumenta, studenti bi opisivali problem na koji su naišli i onda bi čekali odgovor nekog od asistenata. Ovo rešenje je dobro na generacijskom nivou, jer su onda svi studenti iste generacije u mogućnosti da vide odgovor na problem sa kojim se susreo neki od njihovih kolega. Glavni nedostatak ovakvog pristupa jeste što se taj dokument briše i pravi novi, za svaki predmet i za svaku generaciju ponovo.

Umesto dokumenata sa pitanjima i odgovorima, mnogo bolje rešenje je da postoji platforma koja bi obuhvatila sva implementaciona i diskusiona pitanja studenata, nezavisno od predmeta i tehnologija. Na ovakav način se postiže:

- Svaki student može da pogleda bilo koje pitanje i njegove odgovore, bez obzira na to kada je postavljeno.
- Studenti, asistenti i profesori mogu da odgovore na postavljena pitanja. Na takav način student koji je postavio pitanje brže dolazi do odgovora koji mu je potreban, a asistenti i profesori postaju rasterećeniji od gomile pitanja koje bi inače dobili.
- Najčešći problemi koji se javljaju tokom izrade projekata će vrlo brzo imati odgovore i neće biti potrebe za dupliranjem istih pitanja.
- Studenti se navikavaju da rešenje koje su pronašli prilagode svom kodu.

3.1 Postojeća rešenja

Stack Overflow je sajt, zamišljen da predstavlja biblioteku koja bi umesto knjiga na svojim policama imala sva moguća pitanja vezana za programiranje. Najkvalitetniji i vrlo detaljni odgovori bi se nalazili na njihovim stranicama. Programeri širom sveta su upravo oni koji omogućuju da se ovaj cilj postigne, pomažući potpunim strancima na Internetu, nudeći im odgovore za opisane probleme [10].

Stack Overflow je samo jedan od 173 sajta koja se nalaze u okviru *Stack Exchange* mreže, koju na mesečnom nivou poseti preko 100 miliona ljudi. Trenutno predstavlja najmoćniju mrežu za pronalaženje odgovora, sa *Stack Overflow* kao vodećim sajtom. [11].

Discuss The Elastic Stack predstavlja skup „diskusionih foruma za *Elasticsearch*, *Beats*, *Logstash*, *Kibana*, *Elastic Cloud* i ostale proizvode u okviru *Elastic* ekosistema“ [12].

Slično kao i *Stack Overflow*, ovaj sajt je namenjen za rešavanje problema nastalih prilikom implementacije rešenja, koja su usko vezana za *Elastic* domen.

GitHub community je sajt namenjen *GitHub* korisnicima, kao mesto na kome mogu da dobiju odgovor na postavljeno pitanje i da pritom nauče nešto novo ili čak dobiju inspiraciju za neki novi projekat [13].

Osim navedenih sajtova, koriste se i: Dev.to, Experts Exchange, Code Project i slični.

4. *Elasticsearch* mehanizam pretrage

4.1 Šta je *Elasticsearch* mehanizam pretrage i kako je nastao?

Elasticsearch je distribuirani, *open source* mehanizam za pretragu i analitiku čiji temelj predstavlja *Apache Lucene* biblioteka. Omogućava skladištenje, pretragu i analiziranje velike količine podataka u realnom vremenu.

Elasticsearch, danas jedan od najpopularnijih mehanizama pretrage, nastao je kada je Šej Benon, suosnivač *Elastic* kompanije, želeo da kreira mehanizam pretrage koji bi njegova žena mogla da koristi za svoje recepte za kuvanje [14].

4.2 Struktura *Elasticsearch* mehanizma pretrage

Index

Elasticsearch koristi svoju ugrađenu NoSQL bazu koja podatke čuva kao JSON dokumente. Dokumenti predstavljaju osnovnu jedinicu podataka koju je moguće grupisati u *index*, koncept sličan tabeli kod relacionih baza podataka. Da bi se dokumenti grupisali u *index*, moraju imati slične karakteristike [15].

Postoje dva tipa strukture podataka koje se koriste kod mehanizama pretrage u cilju čuvanja i organizovanja podataka:

- ***forward index*** - Dokumenti se mapiraju na termine koje sadrže. To znači da se čuva lista svih reči koje se nalaze u svakom dokumentu. Ova struktura je dobra kada je u pitanju indeksiranje pošto se svaka reč nadovezuje na prethodnu, ali s druge strane nije previše efikasna kada je reč o pretrazi na osnovu termina. Na slici 4 se nalazi primer koji prikazuje kako se korišćenjem *forward index* strukture podataka smešta pet dokumenata i njihov sadržaj. Ukoliko korisnik želi da pronađe sve dokumente u kojima se pominje reč „*Elasticsearch*“, mehanizam bi morao da prođe kroz sve dokumente kako bi korisniku vratio rezultat „dokument1, dokument4“.

Dokument	Opis
dokument1	Elasticsearch
dokument2	Kibana
dokument3	Engine
dokument4	Elasticsearch
dokument5	Database

Slika 4. *Forward index*

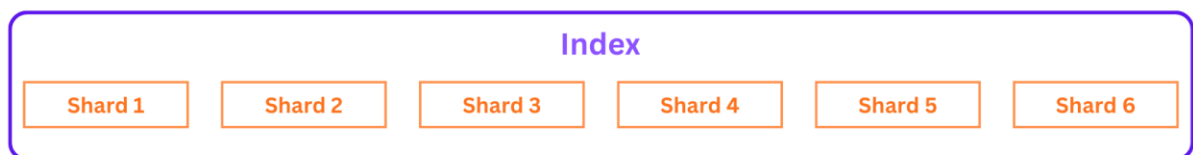
- ***inverted index*** - Termini se mapiraju na dokumente u kojima se nalaze. To znači da za svaku reč postoji lista dokumenata u kojima se nalazi. Kod ovakve strukture je indeksiranje sporije, pošto je za svaku reč potrebno proveriti da li već postoji u *index* komponenti, ali je pretraga veoma brza, čak i kada postoji veoma velika količina podataka iz razloga što se podaci već unapred čuvaju kao upiti (*eng. queries*). *Elasticsearch* mehanizam pretrage koristi upravo ovu strukturu za smeštanje svojih podataka. Na slici 5 se nalazi primer koji prikazuje kako se korišćenjem *inverted index* strukture podataka smešta pet dokumenata i njihov sadržaj. Ukoliko korisnik želi da pronađe sve dokumente u kojima se pominje reč „Elasticsearch“, mehanizam odmah može da vrati rezultat „dokument1, dokument4“ upravo iz razloga što se podaci čuvaju kao upiti [16].

Opis	Dokument
Elasticsearch	dokument1, dokument4
Kibana	dokument2
Engine	dokument3
Database	dokument5

Slika 5. Inverted index

Shard

Shard predstavlja osnovnu gradivnu jedinicu *Elasticsearch* distribuirane arhitekture. *Index* može da sadrži veoma veliku količinu podataka, a da bi upravljanje podacima bilo što efikasnije ti podaci se obično podele u nekoliko *shard* komponenti [17]. Na slici 6 se nalazi primer *index* komponente podeljene na šest *shard* komponenti.



Slika 6. Shards in index

Postoje 2 tipa *shard* komponenti:

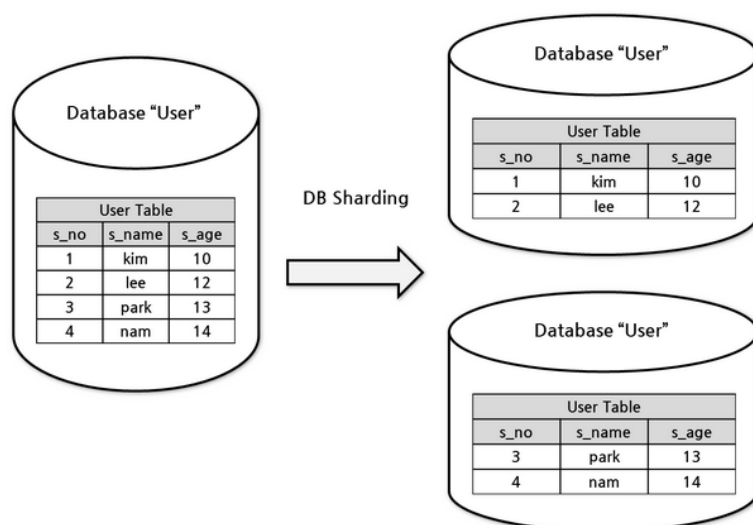
- **primarni** (*eng. primary*) - Sadrže originalne podatke i obavljaju indeksiranje i pretragu. Broj primarnih *shard* komponenti se definiše prilikom kreiranja *index* komponente i ne može se naknadno menjati.
- **kopije** (*eng. replica*) - Predstavljaju kopiju primarnih *shard* komponenti. Uz pomoć njih se uvodi redundantnost i poboljšavaju se performanse pretrage jer omogućavaju izvršavanje paralelnih upita. Broj kopija se može dinamički menjati, prilikom čega se automatski balansira broj kopija među *node* komponentama u *cluster* komponenti. Preporuka je da svaka primarna *shard* komponenta ima svoju kopiju [18].

Broj *shard* komponenti jedne *data node* komponente je proporcijalan *heap* memoriji te *node* komponente. *Node* koji ima 30GB *heap* memorije treba da ima najviše 600 *shard* komponenti, a što ih je manje, time bolje. Korišćenjem previše *shard* komponenti se bespotrebno zauzima memorija i time usporava pretraga, indeksiranje zahteva i druge operacije [19].

Najveća efikasnost pretrage se postiže ukoliko se upiti izvršavaju na više raličitih *shard* komponenti paralelno, prilikom čega se sve *shard* komponente nalaze u okviru različitih *node* komponenti [18].

Prednosti korišćenja *shard* komponenti:

- **horizontalno skaliranje** - *Sharding* je isto što i horizontalno particionisanje, a to je horizontalna podela podataka u više baza. To znači da se jedan set podataka sa svim poljima prebaci u jednu bazu, a drugi set podataka sa svim njegovim poljima prebaci u drugu bazu, kao što je prikazano na slici 7. Zahvaljujući tome *cluster* može da se skalira horizontalno prilikom priliva novih podataka.



Slika 7. Sharding [20]

- **bolje performanse pretrage** - Upiti se mogu vršiti nad više *shard* komponentata u isto vreme, što je mnogo brže od izvršavanja upita nad *index* komponentom koji sadrži samo jednu *shard* komponentu.
- **veća dostupnost** - *Node* komponente mogu da prestanu da rade iz mnogo razloga, bilo zbog nedostatka resursa, problema sa konekcijom ili nekih konfiguracionih grešaka. Praksa je da se prilikom raspodele *shard* komponenti po *node* komponentama, u istoj *node* komponenti ne nađe primarna *shard* komponenta i njena kopija [20]. To znači da iako neki node otkáže, podaci i dalje ostaju dostupni krajnjem korisniku, jer se sve operacije preusmeravaju na one *shard* komponente koje su dostupne [17].

Segment

Svaka *shard* komponenta se sastoji od više *segment* komponenti, koje imaju *inverted index* strukturu podataka.

Prilikom indeksiranja dokumenta, postojeće *segment* komponente se učitaju u memoriju i otprilike svake sekunde se kreira novi *segment*. Ovaj *segment* nije trajno sačuvan u memoriji, tako da postoji opasnost od gubitka

informacija iz njega sve dok *Elasticsearch* ne izvrši *flush* naredbu. Kreiranjem nove *segment* komponente se osvežava pretraga kako bi podaci koji su u njemu sačuvani mogli da se koriste.

Sadržaj *segment* komponente se ne može menjati. To znači da svaki put kada se ažurira dokument, taj dokument se zapravo označi kao obrisani i biva obrisani prilikom spajanja *segment* komponenti, a umesto njega se indeksira novi sa navedenim izmenama.

Što je više *segment* komponenti, time je pretraga sporija. Zbog toga se vrši spajanje *segment* komponenti istih veličina, počev od manjih. Kada nastane dovoljan broj većih *segment* komponenti iste veličine, onda se vrši njihovo spajanje [21].

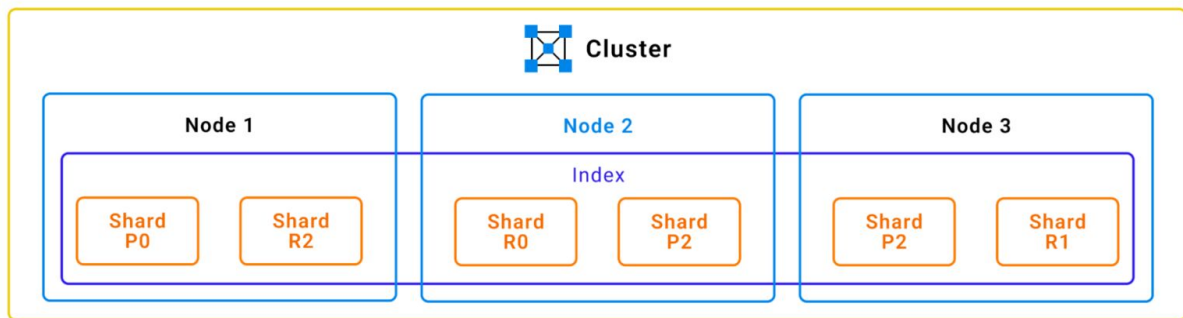
Cluster i node

Node predstavlja server na kome se izvršava jedna *Elasticsearch* instanca. Jedna ili više *node* komponenti povezanih zajedno čine *cluster*. Svaki *node* u okviru jedne *cluster* komponente zna za ostale *node* komponente, što omogućava preusmeravanje zahteva ka odgovarajućoj *node* komponenti.

Node komponente imaju svoje uloge (*eng. roles*) i na osnovu toga se razlikuje više *node* tipova:

- **master-eligible node** - ova uloga omogućava da *node* može biti izabran za *master node*, čije je zaduženje da upravlja celokupnom *cluster* komponentom. **Master node** je zadužen za kreiranje i brisanje *index* komponenti, praćenje koje *node* komponente su u okviru njegove *cluster* komponente i odlučuje koje *shard* komponente će biti dodeljene kojim *node* komponentama.
- **data node** - sadrži podatke i izvršava *CRUD* operacije, pretragu i agregaciju.
- **ingest node** - ima mogućnost korišćenja *ingest pipeline* procesora, uz pomoć kojih se sekvencijalno vrše transformacije nad podacima pre njihovog indeksiranja [22].
- **remote-eligible node** - može se ponašati kao krajnji korisnik.
- **machine-learning node** - *node* koji se mora dodati kako bi moglo da se koristi mašinsko učenje, a može ih biti i više.
- **transform node** - *node* koji se mora dodati kako bi mogle da se koriste transformacije nad podacima. Transformacije omogućavaju kreiranje sumariзовanih *index* komponenti od postojećih [23].
- **coordinating node** - vrši koordinaciju između dve faze pretrage. Prva faza je *scatter* faza i u njoj *coordinating node* prosleđuje zahtev onim *node* komponentama koje imaju tražene podatke. Nakon što *data node* komponente izvrše zahtev, rezultat vraćaju *coordinating node* komponenti. Sledi *gather* faza u kojoj *coordinating node* uzima sve rezultate koje je dobio i stapa ih u jedan. Svaki *node* je implicitno *coordinating node*, međutim ukoliko postoji neki *node* kome nije eksplicitno navedena nijedna uloga, taj *node* će onda jedini biti *coordinating node* [24].

Na slici 8 se nalazi primer jednog distribuiranog sistema, koji se sastoji od tri servera, odnosno, *node* komponente. *Shard* komponente su raspoređene tako da ne postoje primarna i kopija *shard* komponente u istoj *node* komponenti. *Shard* komponente označene slovom P označavaju primarne, a slovom R kopije *shard* komponenti.



Slika 8. Cluster nodes with shards [25]

Index template i data mapping

Index template je skup podešavanja koja se primenjuju nad *index* komponentom prilikom njenog kreiranja. U okviru njega se definišu brojevi primarnih *shard* komponenti i njihovih kopija, *data mapping*, prioritet i ostalo [26].

Data mapping je šema *index* komponente, odnosno, sadrži informacije o svim poljima i njihovim tipovima koje dokumenti u okviru te *index* komponente imaju. Postoje dva tipa za *data mapping*:

- **statički** - unapred su definisana polja i tipovi,
- **dinamički** - prilikom indeksiranja dokumenata, *Elasticsearch* automatski osvežava (eng. *update*) šemu, kofigurišući imena polja i njihove tipove.

Korišćenjem statičkog *data mapping* tipa, može se definisati *index* koji bi imao polja i tipove redom: *ime: string*, *prezime: string* i *brojGodina: int*, kao što je prikazano na Listingu 1 sa leve strane. Kada bi stigao dokument koji se nalazi sa desne strane na istom listingu, bio bi odbijen, zato što se tip vrednosti za polje „*brojGodina*“ ne poklapa sa definisanim u šemi *index* komponente.

Ukoliko bi se umesto statičkog *data mapping* tipa koristio dinamički, onda bi dokument sa desne strane bio prihvaćen, pri čemu bi se napravila izmena u šemi *index* komponente, gde bi se za tip vrednosti za polje „*brojGodina*“ prihvatao i *int* i *string* [27].

<pre>{ "ime": "Marko", "prezime": "Markovic", "brojGodina": 35 }</pre>	<pre>{ "ime": "Marko", "prezime": "Marko", "brojGodina": "35" }</pre>
--	---

Listing 1: Dva dokumenta sa različitim tipovima za polje „*brojGodina*“

Alias

Aliasi se koriste za grupisanje *index* komponenti. Na listingu 2 se nalazi primer kreiranja aliasa, gde se nalaze svi logovi vezani za *index* komponente: *user-logs*, *question-logs* i *comment-logs* [28].

```

POST _aliases
{
  "actions": [
    {
      "add": {
        "indices": ["user-logs", "question-logs", "comment-logs"],
        "alias": "logs"
      }
    }
  ]
}

```

Listing 2: Alias, kao skup više indeksa

4.3 Način rada *Elasticsearch* mehanizma pretrage

Osnovna namena bilo kog mehanizma pretrage, pa tako i *Elasticsearch*, jeste da korisniku ponudi rezultat sa svim dokumentima koji odgovaraju otkucanom upitu. Postavlja se pitanje kako korisnici znaju od svih ponuđenih dokumenata koji je najbolji, odnosno, koji najviše odgovara? *Elasticsearch* je rešenje našao u tome da svakom dokumentu dodeli *score*. Što je *score* veći, to dokument ima veću relevantnost za korisnika. Dokumenti kojima je dodeljena 0 za *score* nemaju nikakvu relevantnost za korisnika i oni se ne ubrajaju u krajnji rezultat. U zbirnom rezultatu se dokumenti ređaju u opadajućem redosledu na osnovu *score* vrednosti.

Moguće je koristiti različite tipove upita, odnosno prertrage, među kojima se najčešće koriste:

- **struktuirana pretraga** (*eng. structured search*) - koristi se za upite nad podacima koji su inherentni, odnosno koji predstavljaju celinu, kao što su: datum, vreme i brojevi. Ovi upiti se koriste za traženje dokumenata kod kojih odgovarajuća polja imaju: stopostotno poklapanje, nalaze se u granicama od do i slično. Razlog tome je što *term level queries* koji se koriste za pretragu nemaju *text analysis* fazu. Postoji mogućnost vršenja pretrage i nad tekstualnim podacima, ali samo u slučaju podataka čije su vrednosti kratke i precizne, poput oznaka (*eng. tags*) objava. *Elasticsearch* ne dodeljuje *score* dokumentu, već „da“ ili „ne“, što označava da dokument odgovara ili ne odgovara upitu. Na listingu 3 se nalazi primer dobavljanja dokumenata postavljenih pre najviše sedam dana iz *Question index* komponente.

```

var response = _elasticClient.Search<Question>(s => s
    .Query(q => q
        .Bool(b => b
            .Filter(f => f
                .DateRange(r => r
                    .Field(ff => ff.PostedOn)
                    .GreaterThanOrEquals(DateTime.UtcNow.AddDays(-7))
                )
            )
        )
    );

```

Listing 3: Struktuirana pretraga

- **nestruktuirana pretraga** (*eng. unstructured search*) - cilj ovog tipa pretrage je da na osnovu otkucanog upita pronađe dokumente u kojima postoji najviše poklapanja, tako što se dokumentima dodeljuje *score*.

Koristi *full text queries*, koji za razliku *term level queries* ima *text analysis* fazu u kojoj se sve reči u upitu porede sa onim u *index* komponenti. Najčešće se koristi za upite nad tekstualnim poljima. Na listingu 4 se nalazi primer upita gde je potrebno pronaći sva dokumenta iz *Question index* komponente, kod kojih polje *Title* sadrži reč „implementacija“.

```
var response = _elasticClient.Search<Question>(s => s
    .Query(q => q
        .Match(m => m
            .Field(f => f.Title)
            .Query("implementacija")
        )
    )
);
```

Listing 4: Nestrukturirana pretraga

- **kombinovni upiti** (eng. *combining queries*) - kombinacija upita koja se naziva *compound query*. Na listingu 5 se nalazi primer kombinovanog upita koji kombinuje upite sa listinga 3 i listiga 4. Iz *Question index* komponente se dobavljaju svi dokumenti kod kojih polje *Title* sadrži reč „implementacija“. Nakon toga se vrši filtracija dokumenata, tako da krajnji rezultat sadrži samo ona pitanja koja su postavljena u poslednjih sedam dana [29].

```
var response = _elasticClient.Search<Question>(s => s
    .Query(q => q
        .Bool(b => b
            .Must(m => m
                .Match(mm => mm
                    .Field(f => f.Title)
                    .Query("implementacija")
                )
            )
        )
        .Filter(f => f
            .DateRange(r => r
                .Field(ff => ff.PostedOn)
                .GreaterThanOrEquals(DateTime.UtcNow.AddDays(-7))
            )
        )
    )
);
```

Listing 5: Kombinovani upit

Text analysis

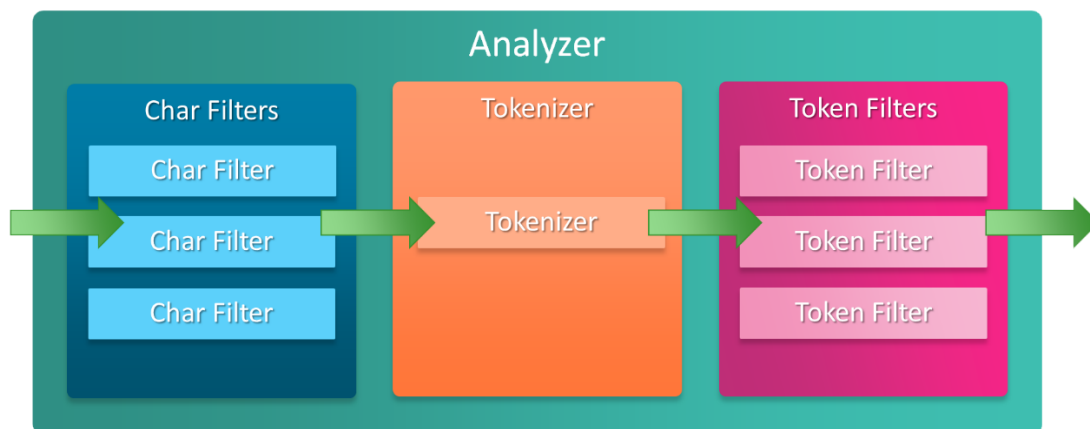
Prilikom čuvanja i pretraživanja podataka, sva tekstualna polja prolaze kroz *text analysis* proces, koji ima dve funkcije:

- **tokenizacija** (eng. *tokenization*) - proces u kome *tokenizer* komponenta vrši deljenja teksta na tokene, najčešće reči, na osnovu nekog definisanog pravila.
- **normalizacija** (eng. *normalization*) - proces u kome se vrše transformacije tokena. Transformacije mogu biti različite, od *stemming* procesa do posmatranja sinonima, specijalnih reči i tako dalje [30].

Analyzer modul izvršava proces analize teksta. *Elasticsearch* ima nekoliko ugrađenih tipova *analyzer* modula i oni imaju podršku za rad za različite jezike. Osim njih, moguće je kreirati i sopstveni *analyzer* modul, kombinujući komponente.

Analyzer modul je sastavljen od tri tipa komponenti kroz koje tekst prolazi u toku analize, a primer njegove strukture je moguće videti na slici 9:

- **character filters** - koriste se za dodavanje, menjanje i brisanje karaktera iz niza karaktera kojim je predstavljen tekst na ulazu. Izvršavaju se redom, jedan za drugim. Primer primene bi bio uklanjanje *HTML* tagova iz teksta. *Analyzer* modul može imati nijedan ili više *character filter* komponenti.
- **tokenizer** - na ulazu dobija tekst predstavljen kao niz karakter koji deli na tokene, najčešće to budu reči, na osnovu definisanog pravila. Tekst se može podeliti na tokene posmatranjem: razmaka, određenog slova, šablona i slično. *Analyzer* modul ima tačno jedan *tokenizer*.
- **token filters** - na ulazu dobijaju niz tokena. U taj niz je moguće dodati novi, izmeniti ili obrisati postojeći token, u cilju što bolje normalizacije teksta. Izvršavaju se redom, jedan za drugim. *Analyzer* modul može imati nijedan ili više *token filters* komponenti [31].



Slika 9. Struktura analyzer modula [32]

U okviru *text analysis* procesa je moguće koristiti i *normalizer* modul koji, za razliku od *analyzer* modula, na izlazu daje samo jedan, a ne niz tokena, što znači da nema *tokenizer* komponentu. Još jedna razlika između ova dva modula je što *normalizer* modul ima manji skup *character* i *token* filtera. Koristi se za polja koja su tipa *keywords* [33].

Stemming

Stemming je proces u kome se reč redukuje na svoj koren. Koren na koji je reč redukovana ne mora biti prava reč, odnosno, ne mora ni zaista biti koren te reči. Bitno je da se postigne da iste reči napisane u različitom obliku budu prepoznate kao iste.

Koliko dobre rezultate će *stemming* proces dati uveliko zavisi od jezika nad kojim se primenjuje, ali i od tipa *stemmer token filter* komponente koja se koristi.

Stemming token filters su zaduženi za *stemming* i postoje dve vrste:

- **algorithmic stemmers** - koriste skup pravila koja se primenjuju na tokene. Primer bi bio skidanje prefiksa i sufiksa sa reči. Lako se podešavaju i koriste vrlo malo memorije, a pritom su brži od *dictionary stemmers*. Međutim, ovaj tip *stemming token filters* komponenti ne radi dobro sa rečima koje ne sadrže koren u svom obliku.
- **dictionary stemmers** - koriste rečnik koji je potrebno navesti pri implementaciji. Za svaku reč, odnosno token, traže koren u okviru datog rečnika. Ovim se rešava problem pronalaženja korena za izuzetke, ali i reči koje deluju kao da imaju zajednički koren, ali je međusobna smisao tih reči potpuno različita. Iako se ova opcija za *stemming token filters* čini kao bolja, izbegava se, zato što je potrebno naći provereno dobar rečnik, koji ima što veći broj reči, a takvi su često retko dostupni. Osim toga, da bi rečnik mogao da se koristi, potrebno je da bude celokupno učitao u RAM memoriju, što znači da je potrebno učitati sve njegove reči, prefikse i sufikse, što poprilično usporava *stemming* proces.

Bez obzira na to koji *stemming token filter* se koristi, potrebno je isti primeniti na tekst i prilikom upisivanja u *index* i prilikom pretrage, s obzirom da *stemming* menja oblik reči [34].

Stop words

Stop words predstavljaju skup reči koje ni na koji način ne doprinose pretrazi kako bi njen rezultat bio bolji, pa se iz tih razloga ignorišu. To su reči koje se često koriste u svakodnevnom govoru, a za srpski jezik bi to bili: veznici, zamenice, predlozi, pomoćni glagoli i slično. *Stop words* se koriste uz pomoć *stop token filter* komponente. *Elasticsearch* nudi listu predefinisanih *stop words* za više jezika [35][36].

Synonyms

Synonyms predstavljaju jednu od ključnih stvari za uspešnu pretragu. Njihovim korišćenjem se poboljšava rezultat pretrage, jer je moguće naći i one dokumente koji sadrže sinonime umesto otkucanih reči u upitu.

Synonyms se čuvaju u *synonyms sets* u vidu *synonym rules* celina. *Synonym rules* definišu skup reči koje su sinonimi [37]. Svaki *synonyms set* je potrebno sačuvati u okviru *Elasticsearch* mehanizma kako bi mogao da se koristi.

Postoje tri načina na koje je moguće čuvati *synonyms sets*:

- **synonyms API** - najfleksibilniji način. Sve izmene nad *synonyms set* su dinamičke i *analyzer* koji referencira taj *set* se sam automatski učitava pri svakoj izmeni.
- **synonyms file** - kreiranje fajlova koji sadrže *synonyms set*. Potrebno je da svaki *node* u *cluster* komponenti sadrži ovaj fajl. Sa svakom izmenom fajla potrebno je okačiti ažurirani fajl za svaki *node* u *cluster* komponenti i ponovo učitati *analyzer* koji se koristi.
- **inline** - sinonimi se direktno dodaju u *token filter*. Nije preporučljivo.

Prilikom korišćenja *synonyms* potrebno je voditi računa o tome da postoji *synonyms set* koji neka *index* komponenta referencira. U suprotnom, *index* komponenta će biti parcijalno kreirana i neće moći da se koristi. Kod kreiranih *synonyms set* treba voditi računa da su sve *synonym rules* celine dobro napisane, u suprotnom *analyzer* neće moći da se učita.

Potrebno je definisati *token filter* za korišćenje *synonyms* i za tu svrhu postoje dva *token filter* tipa:

- ***synonym graph*** - koristi se kada jedna reč ima više različitih sinonima.
- ***synonym*** - ne preporučuje se za slučajeve kada jedna reč ima više različitih sinonima [38].

Scoring mehanizam

Prilikom pretrage, *Elasticsearch* svakom dokumentu, na osnovu upita koji je korisnik uneo, dodeljuje *score* vrednost. Uz pomoć te vrednosti se određuje relevantnost dokumenta za krajnjeg korisnika i vrši se sortiranje dokumenata, koja se nalaze u krajnjem rezultatu, u opadajućem redosledu.

Scoring mehanizam se zasniva na *BM25* algoritmu [39]. Ovaj algoritam posmatra tri parametra prilikom određivanja *score* vrednosti dokumenta:

- ***term frequency (TF)*** - broj puta koji se određena reč iz upita nalazi u dokumentu. Što je reč češća u okviru dokumenta, to je *TF* vrednost veća i dokument ima veću relevantnost.
- ***inverse document frequency (IDF)*** - određuje koliko je reč iz upita relevantna na osnovu toga koliko se često pominje u svim dokumentima u okviru *index* komponente. Što je reč češće prisutna u dokumentima, to je manje relevantna i dobija manju *IDF* vrednost i obrnuto, što se ređe pojavljuje smatra se više relevantnom i dobija veću *IDF* vrednost.
- ***field length normalization*** - posmatra dužinu polja u kome je pronađena reč iz upita. Što je polje duže, to je ova vrednost manja. Kraća polja dokumenata u kojima je pronađena reč dobijaju veću vrednost za ovaj parametar.

Ovi parametri se računaju za svaku reč koja se nalazi u upitu. Formula koja se koristi za računanje ukupne *score* vrednosti dokumenta je prikazana na slici 10.

$$\sum_i^n IDF(q_i) \frac{f(q_i, D) * (k1 + 1)}{f(q_i, D) + k1 * (1 - b + b * \frac{fieldLen}{avgFieldLen})}$$

Slika 10. *BM25* formula [39]

Korišćenjem *Explain API* koji nudi *Elasticsearch*, moguće je bolje razumeti način dodeljivanja vrednosti paramterima. Na slici 11 se nalazi primer za *Explain API endpoint* i deo rezultata [40].

Slika 11. Explain API

Ukoliko bi se postavila granica za maksimalnu vrednost koju dokument može da ima, desilo bi se sledeće:

„Mislim da bi dobio *false* relevantne vrednosti. Recimo da u okviru iste *index* komponente vršiš pretragu dva puta. Nakon prve pretrage, *score* vrednosti za relevantnost su u rasponu od .0001 do 1.5000 i to skalira da bi bilo u opsegu od 0 do 1. Za drugu pretragu, *score* vrednosti za relevantnost su u rasponu od 0.0500 do 3.5000 i to se skalira na opseg od 0 do 1. Sada si dobio dva seta rezultata - možeš pomisliti da dva rezultata koja imaju najveću *score* vrednost iz obe pretrage, pošto je sve skalirano na raspon od 0 do 1, imaju istu vrednost - ali nemaju. Veštački si skalirao rezultate tako da se uklapaju u opseg vrednosti od 0 do 1. Time si izgubio originalne *score* vrednosti za relevantnost uz pomoć kojih bi moglo da se vrši poređenje najboljih rezultata dve pretrage, pošto skaliranje vrednosti na opseg od 0 do 1 nije konstantno za sve pretrage.“ - Bob Sandiford, *Principal Engineer* [41].

Fuzzy search

Prilikom pretrage, često se desi da korisnici pogrešno otkucaju reč ili nekoliko reči u celom upitu. Ipak, i dalje žele da dobiju iste rezultate kao i u slučaju tačno otkucanih reči. *Elasticsearch* je to rešio korišćenjem *fuzzy query*, koji se može koristiti kao zaseban *query* ili kao parametar unutar *match query*. *Fuzzy query* koristi Levenštajnov algoritam udaljenosti (eng. *Levenshtein Distance Algorithm*) da bi izračunao udaljenost reči iz upita od onih koje se nalaze u *index* komponenti i time proverio da li je nastala greška u kucanju ili se radi o skroz različitim rečima [42].

Levenštajnov algoritam udaljenosti računa distancu između reči tako što poredi karaktere reči redom. Za svaki karakter koji se razlikuje dodaje se vrednost 1 na ukupnu vrednost distance. Na slici 12 se nalazi primer računanja Levenštajnov distance za reči „*elastikseerch*“ i „*elasticsearch*“. Distanca se ne menja u svakoj koloni u kojoj su slova u toj koloni ista. Za svako različito slovo, na trenutnu vrednost distance se dodaje vrednost 1.

Otkucana reč	E	L	A	S	T	I	K	S	E	E	R	C	H
Reč u indeksu	E	L	A	S	T	I	C	S	E	A	R	C	H
Levenštajnova distanca	0	0	0	0	0	0	1	1	1	2	2	2	2

Slika 12. Levenštajnova distanca

Levenštajnova distanca se računa za svaku reč u upitu posebno, ukoliko se pretraga vrši za tekstualno polje.

Ukoliko se u okviru *Elasticsearch* mehanizma koristi *match* ili *multi_match query*, *fuzzy query* se primenjuje korišćenjem *fuzziness* parametra u okviru pretrage. Ovaj parametar određuje kolika je maksimalna dozvoljena distanca i prima dva tipa vrednosti: *AUTO* ili broječanu vrednost. Na slici 13 se nalazi primer upita koji koristi *fuzziness* parametar. Prva reč u upitu ima distancu 0, dok druga ima distancu 1.



```
GET question-index/_explain/-7
{
  "query": {
    "multi_match": {
      "query": "dependency injection",
      "fuzziness": "AUTO", // "fuzziness": "2",
      "fields": ["description"]
    }
  }
}
```

```
1 * {
2   "_index": "question-index",
3   "_id": "-7",
4   "matched": true,
5   "explanation": {
6     "value": 4.149836,
7     "description": "sum of:",
8     "details": [
9       {
10        "value": 2.2132459,
11        "description": "weight(description:dependency in 1) [PerFieldSimilarity], result of:",
12        "details": [
13          {
14            "value": 2.2132459,
15            "description": "score(freq=1.0), computed as boost * idf * tf from:",
16            "details": [
17              {
18                "value": 2.2,
19                "description": "boost",
20                "details": []
21              },
22              {
23                "value": 2.235923,
24                "description": "idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:",
25                "details": [
26                  {
27                    "value": 1,
28                    "description": "n, number of documents containing term",
29                    "details": []
30                  },
31                  {
32                    "value": 13,
33                    "description": "N, total number of documents with field",
34                    "details": []
35                  }
36                ]
37              }
38            ]
39          }
40        ]
41      }
42    ]
43  }
44 }
```

Slika 13. Fuzzy query

Ukoliko se koristi vrednost *AUTO*, onda se na osnovu dužine reči određuje maksimalna dozvoljena distanca i to:

- distanca 0 - za reči koje imaju 2 ili manje karaktera,
- distanca 1 - za reči koje imaju između 2 i 6 karaktera,
- distanca 2 - za reči koje imaju 6 i više karaktera [43].

Vrednost *AUTO* pruža veću fleksibilnost, za razliku od korišćenja precizne broječne vrednosti koja bi predstavljala maksimalnu distancu za sve reči, bez obzira na njihovu dužinu.

Jedna od čestih grešaka pri kucanju je transponovanje slova. Na primer, u reči „*elasticsearch*“ koristeći Levenštajnov algoritam, distanca bi bila 2. S obzirom da je očigledno kod ovakvog tipa grešaka koja reč je u pitanju, poželjno je da se zbog njih ne dobija veća distanca, a to se postiže korišćenjem *transpositions* parametra.

Transpositions parametar omogućava da se kod transponovanih slova distanca računa kao +1 na trenutnu vrednost distance.

Na slici 14 se nalazi primer korišćenja *transpositions* parametra uz *fuzziness*. *Tags* polje je *keyword* polje, što znači da se vrednosti ovog polja posmatraju u celini, takve kakve su, pa samim tim će se i otkucani upit nad ovim poljem posmatrati kao jedna celina. Imajući to u vidu, prvo što se primeti jeste da fali „-“ između reči „*edependency*“ i „*injection*“, što daje vrednost 1 za distancu. Slova „d“ i „e“ su transponovana, pa zahvaljujući *transpositions* parametru, ova distanca će se računati kao +1 na trenutnu, što daje ukupnu Levenštajnovu distancu dva. Pošto je *fuzziness* podešen na *AUTO* i upit ima sigurno više od 6 slova, ispoštovano je da je maksimalna dozvoljena distanca dva, pa je pronađena sličnost između reči iz upita i dokumenta i dokument se smatra relevantnim i dodeljuje mu se određeni *score*.

```

GET question-index/_explain/-7
{
  "query": {
    "fuzzy": {
      "tags": {
        "value": "edependency injection",
        "fuzziness": "AUTO",
        "transpositions": true
      }
    }
  }
}

1- {
2  "index": "question-index",
3  "id": "-7",
4  "matched": true,
5  "explanation": {
6    "value": 2.7215462,
7    "description": "weight(tags:dependency-injection in 1) [PerFieldSimilarity], result of:",
8    "details": [
9      {
10       "value": 2.7215462,
11       "description": "score(freq=1.0), computed as boost * idf * tf from:",
12       "details": [
13         {
14           "value": 1.98,
15           "description": "boost",
16           "details": []
17         },
18         {
19           "value": 2.2335923,
20           "description": "idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:",
21           "details": [
22             {
23               "value": 1,
24               "description": "n, number of documents containing term",
25               "details": []
26             },
27             {
28               "value": 13,
29               "description": "N, total number of documents with field",
30               "details": []
31             }
32           ]
33         }
34       ],
35       "value": 0.6153846

```

Slika 14. Fuzzy query - transpositions

Osim *fuzziness* i *transpositions* parametara, moguće je koristiti i parametre poput:

- **max expansions** - maksimalan broj dokumenata koji mogu da se nađu u rezultatu,
- **prefix length** - broj početnih karaktera koje će *fuzzy query* ignorisati,
- **rewrite** - za menjanje *score* vrednosti rezultata [42].

5. DebugIt

DebugIt je studentska QA platforma sa integrisanim *Elasticsearch* mehanizmom pretrage. Predstavlja riznicu najraznovrsnijih studentskih pitanja, od diskusionih do implementacionih, i njihovih rešenja.

Zajednicu ove platforme čine studenti, asistenti i profesori. Studenti bi uz pomoć ove platforme mogli da pronađu odgovor na bilo koji problem ili nejasnoću koristeći jedan od trenutno najpopularnijih i najbržih mehanizama pretrage, *Elasticsearch*. Ukoliko slično pitanje nije postavljeno, studenti su u mogućnosti da ga postavе i vrlo brzo dobiju odgovor na njega. Osim pretraživanja i postavljanja pitanja, mogu na njih i da odgovaraju i time pruže brzu pomoć nekom od svojih kolega, a asistenti i profesori su tu da im u tome pripomognu.

Svako pitanje i odgovor je moguće označiti („lajkovati“ ili „dislajkovati“). Time se daje do znanja koliko ljudi je pitanje, ili odgovor, smatralo korisnim.

Sve platforme sličnog tipa se danas zasnivaju na kreiranju zajednica programera širom sveta koji pomažu jedni drugima i uz pomoć kojih sakupljaju najrazličitija pitanja, opšteg ili specifičnog domenskog karaktera. *DebugIt* platforma predstavlja jedno od odličnih početnih mesta za mlade programere koji tek stupaju u taj svet.

Rešenje je razvijano uz korišćenje sledećeg skupa tehnologija:

- **ASP.NET** - *Backend* deo aplikacije.
- **Elasticsearch** - Mehanizam za efikasnu pretragu.
- **NEST** - Oficijalna *.NET* biblioteka uz pomoć koje se vrši integracija *Elasticsearch* mehanizma.
- **Elasticsearch DBMS** - Distribuirana, skalabilna baza u kojoj se čuvaju podaci, radi što brže i relevantnije pretrage.
- **Kibana** - Alat za vizualizaciju podataka, analitiku i monitoring aplikacije koja koristi *Elasticsearch*.
- **Docker** - Podizanje *Elasticsearch* i *Kibana* instanci i konfiguraciju broja *node* komponenti u *cluster* komponenti.
- **Angular** - *Frontend* deo aplikacija.

5.1 Detalji implementacije

Konfiguracija *Elasticsearch* mehanizma i *index* komponenti

Konfiguracija *Elasticsearch* mehanizma je prikazana na listingu 6. Port 9200 je port za *Elasticsearch*, definisan u okvir *docker-compose.yml* fajla. Mora postojati podrazumevani *index* u koji će se smeštati podaci. Ukoliko nijedan drugi nije kreiran ili naveden, i u te svrhe se koristi „*elasticsearch-demo*“.

```
var settings = new ConnectionSettings(new
    Uri("http://localhost:9200")).DefaultIndex("elasticsearch-demo");
var client = new ElasticClient(settings);
```

Listing 6: Konfiguracija *Elasticsearch* mehanizma

Aplikacija sadrži tri glavne klase: *User*, *Question* i *Comment*. Za svaku od njih je potrebno kreirati poseban *index* u kome će se nalaziti podaci sa poljima navedenim u tim klasama.

U narednih nekoliko listinga će biti prikazana sva podešavanja koja je potrebno napraviti da bi se kreirao *index* koji se koristi za pretragu.

Na listingu 7 se nalazi primer podešavanja koja se odnose na podešavanja strukture *index* komponente. Najpre je potrebno odrediti ime za *index*, nakon čega se postavlja broj primarnih *shard* komponenti i njihovih kopija na isti broj, odnosno dva, zato što je potrebno da svaka primarna *shard* komponenta ima svoju kopiju. Naredno podešavanje se odnosi na maksimalnu veličinu *segment* komponente koja može nastati spajanjem. Poslednje podešavanje na listingu se odnosi na ispis *log* zapisa koji treba da sadrže upite korišćene za pretrage kojima je trebalo više nego što je očekivano da se izvrše. Ovi *log* zapisi se upisuju u *Elasticsearch slow log*.

```
var response = client.Indices.Create("question-index", q => q
    .Settings(s => s
        .NumberOfShards(2)
        .NumberOfReplicas(2)
        .Setting("index.merge.policy.max_merged_segment", "10mb")
        .Setting("index.search.slowlog.threshold.fetch.warn", "1s")
```

Listing 7: Podešavanja index komponente

Na listingu 8 se nalazi podešavanje vezano za *text analysis* fazu. Postavlja se standardni *tokenizer*.

```
.Analysis(a => a
    .Tokenizers(t => t
        .Standard("serbian_latin_tokenizer")
    )
```

Listing 8: Tokenizer

Na listingu 9 se nalaze *token filter* komponente, koje se koriste u *text analysis* fazi. Svaka od komponenti ima jednu od sledećih uloga, navedenih redom kako se pojavljuju u kodu:

- Sav tekst se prebacuje u mala slova.
- Slova poput š, č, ć se posmatraju kao s i c, kako bi se omogućila veća fleksibilnost pretrage za korisnike koji ne koriste navedena slova prilikom kucanja.
- *Token filter* za *stop words*. Učitava spisak veznika, zamenica, predloga, pomoćnih glagola i sličnih reči iz priloženog fajla koji se nalazi u konfiguracionom folderu.
- *Token filter* za *stemming*. Koristi se algoritamski *stemmer* prilagođen za srpski jezik.
- *Token filter* za *synonyms*. Učitava *synonym set* iz fajla koji se nalazi u konfiguracionom folderu.

```
.TokenFilters(tf => tf
    .Lowercase("serbian_lowercase")
    .AsciiFolding("serbian_ascii_folding", afd => afd)
    .Stop("serbian_stop_words", st =>
st.StopWordsPath("serbianStopwords.txt"))
    .Stemmer("serbian_stemmer", st => st.Language("serbian"))
    .SynonymGraph("synonym", sy => sy.SynonymsPath("serbianSynonyms.txt"))
)
```

Listing 9: Token filters

Na listingu 10 se nalazi podešavanje *analyzer* komponente koja će se koristiti prilikom *text analysis* procesa. Definisan je *custom analyzer* koji radi isto kao i *standard analyzer*, ali je prilagođen za rad za srpski jezik. U okviru njega se definišu *tokenizer* i sve *token filter* komponente navedene na prethodna dva listinga.

```

        .Analyzers(an => an
            .Custom("serbian_latin_analyzer", ca => ca
                .Tokenizer("serbian_latin_tokenizer")
                .Filters("serbian_lowercase", "serbian_ascii_folding",
                    "serbian_stop_words", "serbian_stemmer",
                    "synonym")
            )
        )
    )
}

```

Listing 10: Analyzer

Nakon što su izvršena sva podešavanja koja su potrebna da bi *text analysis* proces dobro radio, potrebno je mapirati polja klase na attribute koji će se naći u odgovarajućoj *index* komponenti, kao što je prikazano na listingu 11. Osim mapiranja, potrebno je navesti i kog tipa su polja koja se nalaze u klasi, kako bi *Elasticsearch* znao kako da ih posmatra prilikom pretrage. Pošto se u okviru aplikacija pretraga radi po poljima *Title*, *Description* i *Tags*, samo ova polja su i eksplicitno navedena. Za svako tekstualno polje je potrebno navesti koji *analyzer* se koristi za njih.

```

        .Map<Question>(m => m
            .AutoMap()
            .Properties(p => p
                .Text(t => t
                    .Name(n => n.Title)
                    .Analyzer("serbian_latin_analyzer")
                )
                .Text(t => t
                    .Name(n => n.Description)
                    .Analyzer("serbian_latin_analyzer")
                )
                .Keyword(t => t
                    .Name(n => n.Tags)
                )
            )
        )
    );

```

Listing 11: Mapiranje polja

Ovim su zaokružena osnovna podešavanja koja je potrebno napraviti kako bi *index* bio konfigurisan tako da se može koristiti za pretragu. *Index* komponente koje odgovaraju klasama *User* i *Comment* imaju dosta jednostavniju konfiguraciju, s obzirom da se za sad pretraga vrši samo u okviru *question-index* komponente.

Prilikom pokretanja programa, *Elasticsearch* će prvo proveriti da li već postoje navedene *index* komponente. Ukoliko ne postoje, kreiraće ih.

Implementacija *CRUD* metoda

Svaka klasa ima svoj *service* koji nasleđuje *ElasticsearchService<T>*, generičku klasu koja sadrži implementaciju *CRUD* metoda.

U okviru *ElasticsearchService<T>*, najpre je potrebno instancirati objekat klase *ElasticClient*, kao što je prikazano na listingu 12 i potom injektovati tu vrednost u samom konstruktoru.

```
private readonly ElasticClient _elasticClient;
```

Listing 12: ElasticClient

Na listinzima 13, 14, 15, 16 i 17 se nalaze primeri implementacije *CRUD* metoda. U svakoj od metoda se eksplicitno naglašava kojoj *index* komponenti je potrebno pristupiti, tako što se koristi naziv klase, prilikom čega su sva slova konvertovana u mala slova i dodat je sufiks „-index“.

```
public async Task<T> GetDocumentAsync(int id)
{
    var res = await _elasticClient.GetAsync<T>(id, d =>
        d.Index(typeof(T).Name.ToLower() + "-index"));
    return res.Source;
}
```

Listing 13: GetDocumentAsync(int id)

```
public async Task<IEnumerable<T>> GetAllDocuments()
{
    var res = await _elasticClient.SearchAsync<T>(s => s
        .Index(typeof(T).Name.ToLower() + "-index")
        .MatchAll()
        .Size(10000));
    return res.Documents;
}
```

Listing 14: GetAllDocuments()

```
public async Task<string> CreateDocumentAsync(T document)
{
    var res = await _elasticClient.IndexAsync(document, d =>
        d.Index(typeof(T).Name.ToLower() + "-index"));
    return res.IsValid ? "Document created successfully" : "Failed to create document";
}
```

Listing 15: CreateDocumentAsync(T document)

```
public async Task<string> UpdateDocumentAsync(T document)
{
    var res = await _elasticClient.UpdateAsync(new DocumentPath<T>(document), d => d
        .Index(typeof(T).Name.ToLower() + "-index")
        .Doc(document)
        .RetryOnConflict(3));

    return res.IsValid ? "Document updated successfully" : "Failed to update document";
}
```

Listing 16: UpdateDocumentAsync(T document)

```
public async Task<string> DeleteDocumentAsync(int id)
{
    var res = await _elasticClient.DeleteAsync<T>(id, d =>
        d.Index(typeof(T).Name.ToLower() + "-index"));
    return res.IsValid ? "Document deleted successfully" : "Failed to delete document";
}
```

Listing 17: DeleteDocumentAsync(int id)

Implementacija pretrage

U okviru *QuestionService* klase se pored opisanih nalazi i metoda, prikazana na listingu 18, koja se koristi za pretragu. Kao i u prethodnim metodama, prvo se navodi *index* u okviru kog se vrši pretraga. U okviru *.Query()*

parametra se koristi `.MultiMatch()` parametar koji omogućuje da se pretraga izvršava nad više polja. Drugim rečima, ključne reči za pretragu se mogu naći samo u nekim od ovih polja, a mogu i u svim. Iz tog razloga se posmatraju sva navedena polja. S obzirom da su sve tekstualne vrednosti u poljima napisane malim slovima, isto to je potrebno uraditi i sa samim upitom. Greške u upitu nastale pri kucanju se ignorišu korišćenjem `.Fuzziness()` i `FuzzyTranspositions()` parametara. U `res` promenljivoj će se nalaziti `ISearchResponse<Question>` koji u sebi sadrži prvih hiljadu dokumenata, a uz pomoć `res.Documents` se vrši deserijalizacija rezultata u objekte `Question` klase.

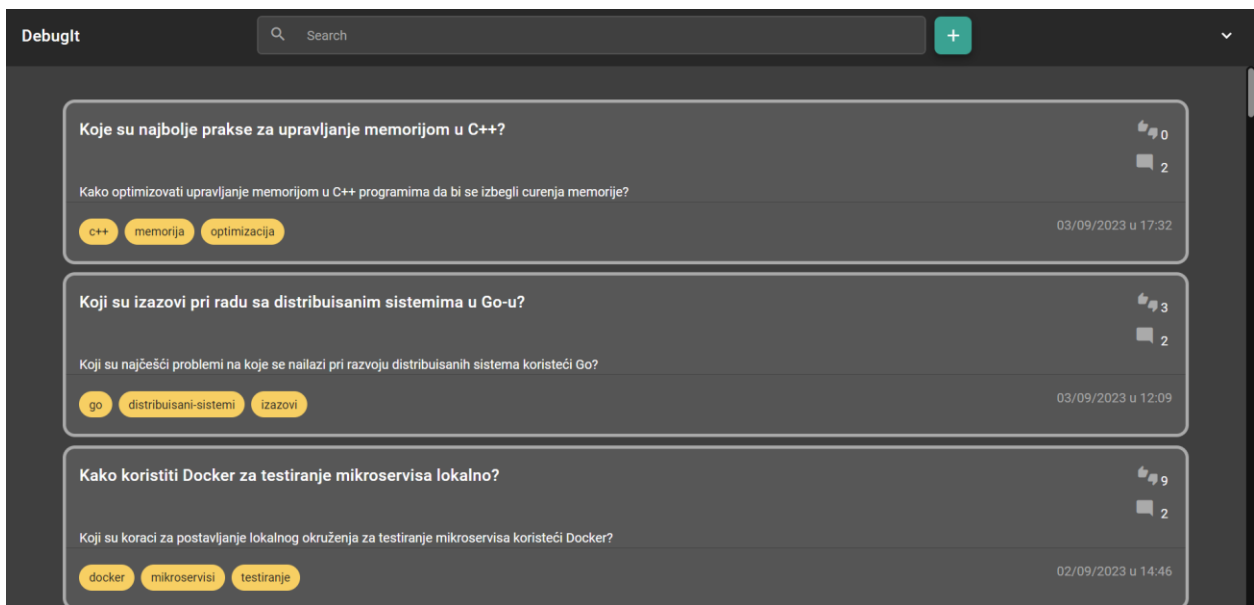
```
public async Task<IEnumerable<Question>> SearchAsync(string query)
{
    var res = await _elasticClient.SearchAsync<Question>(s => s
        .Index("question-index")
        .Query(q => q
            .MultiMatch(m => m
                .Query(query.ToLower())
                .Fuzziness(Fuzziness.Auto)
                .FuzzyTranspositions(true)
                .Fields(f => f
                    .Field(ff => ff.Title)
                    .Field(ff => ff.Description)
                    .Field(ff => ff.Tags)
                )
            )
        )
        .Size(1000));
    return res.Documents;
}
```

Listing 18: `SearchAsync(string query)`

Ukoliko *endpoint URL* koji se koristi za pretragu izgleda ovako: `[HttpGet("search/{query}")]`, potrebno je voditi računa i o *endpoint* pozivu na *frontend* delu. Na primer, korisnik želi da pretraži sva pitanja vezana za `C#`. Kako je `#` specijalni URI karakter, osim `C#` pitanja, dobiće i pitanja vezana za `C`, zato što bi `#` bila izbačena iz upita. Da bi se to izbeglo, specijalne *URI* karaktere je potrebno enkodovati uz pomoć `encodeURIComponent()` funkcije na frontu.

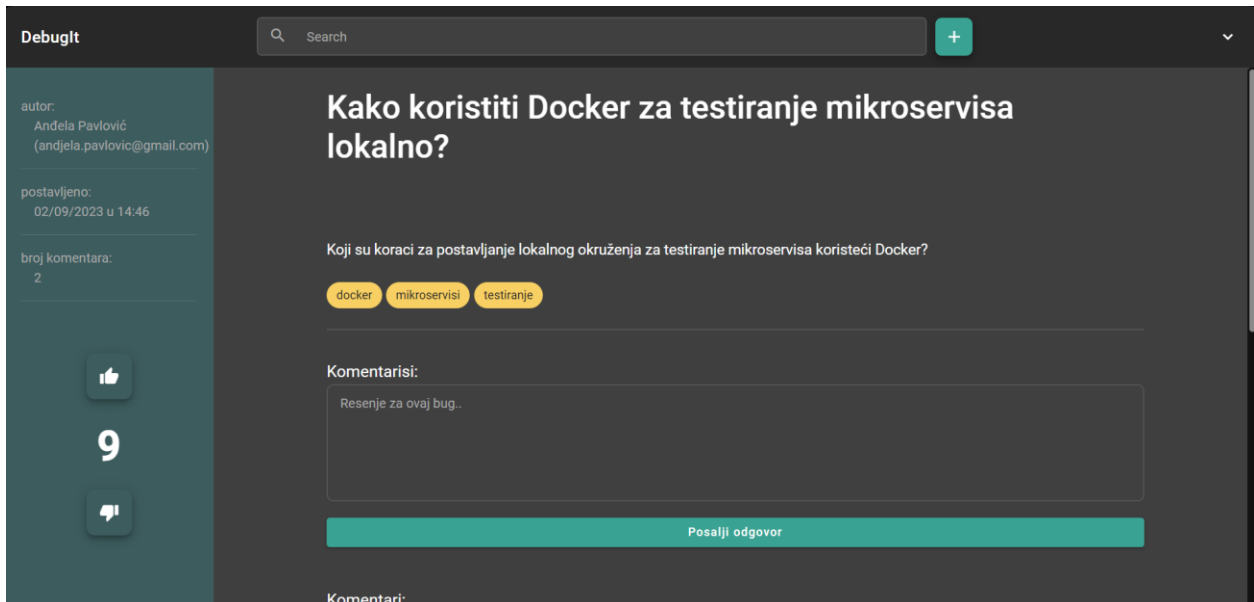
Primeri korisničkog interfejsa

Na slici 15 se nalazi osnovna stranica koji sadrži pitanja sortirana po datumu, tako da se najnovija pitanja uvek pojavljuju prva i tako postaju više uočljiva, što je pogotovu korisno kada još uvek nemaju nijedan odgovor.

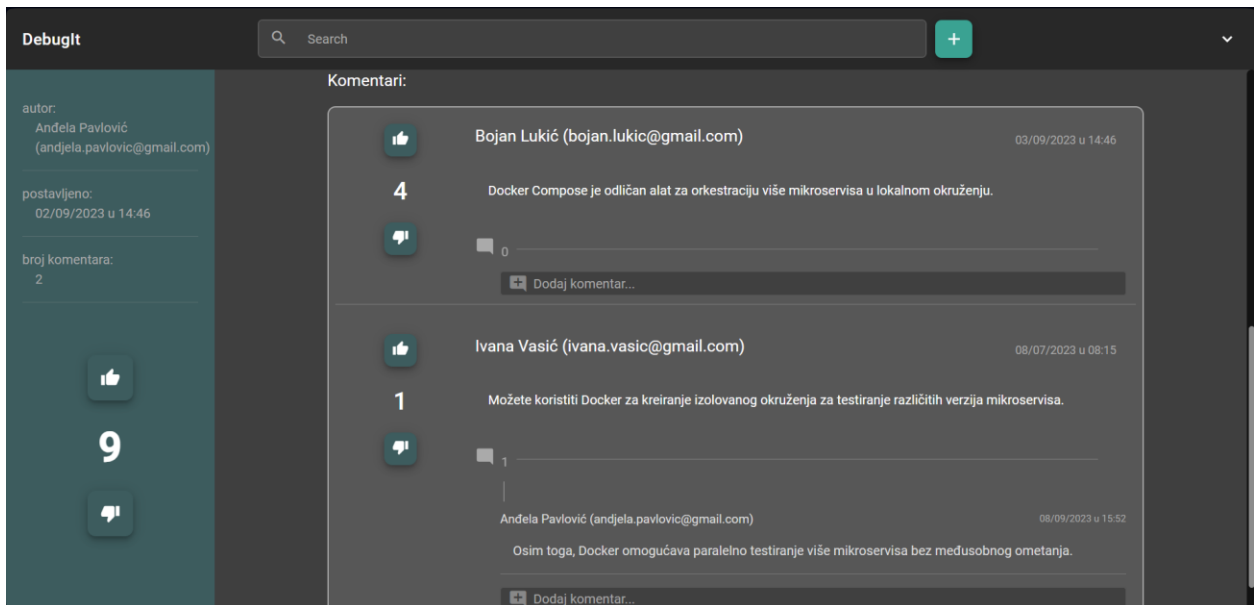


Slika 15. Landing page

Na slikama 16 i 17 se nalazi prikaz detalja pitanja i svih odgovora. Sa leve strane se nalaze osnovne informacije: ime i *email* adresa autora, datum postavljanja, broj komentara i broj lajkova. Time se daje do znanja ostalim korisnicima koliko se pitanje smatra korisnim. Moguće je lajkovati i komentare i tako staviti do znanja ostalim korisnicima koji komentar predstavlja potencijalno najbolji odgovor na postavljeno pitanje. Komentari mogu imati potkomentare.

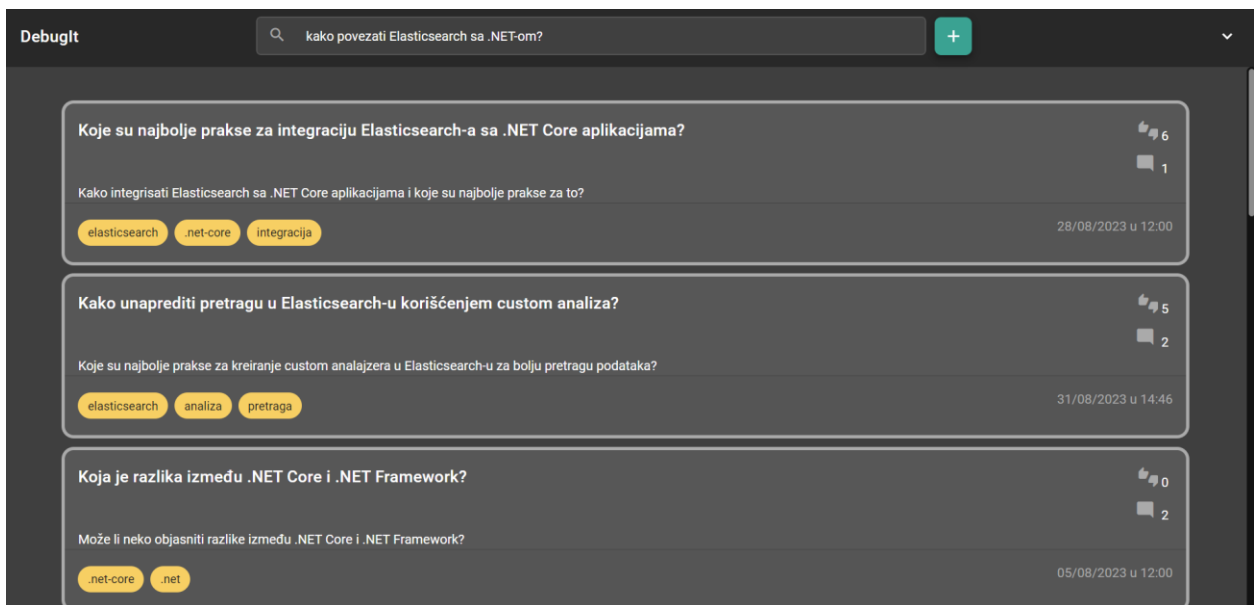


Slika 16. Detalji pitanja

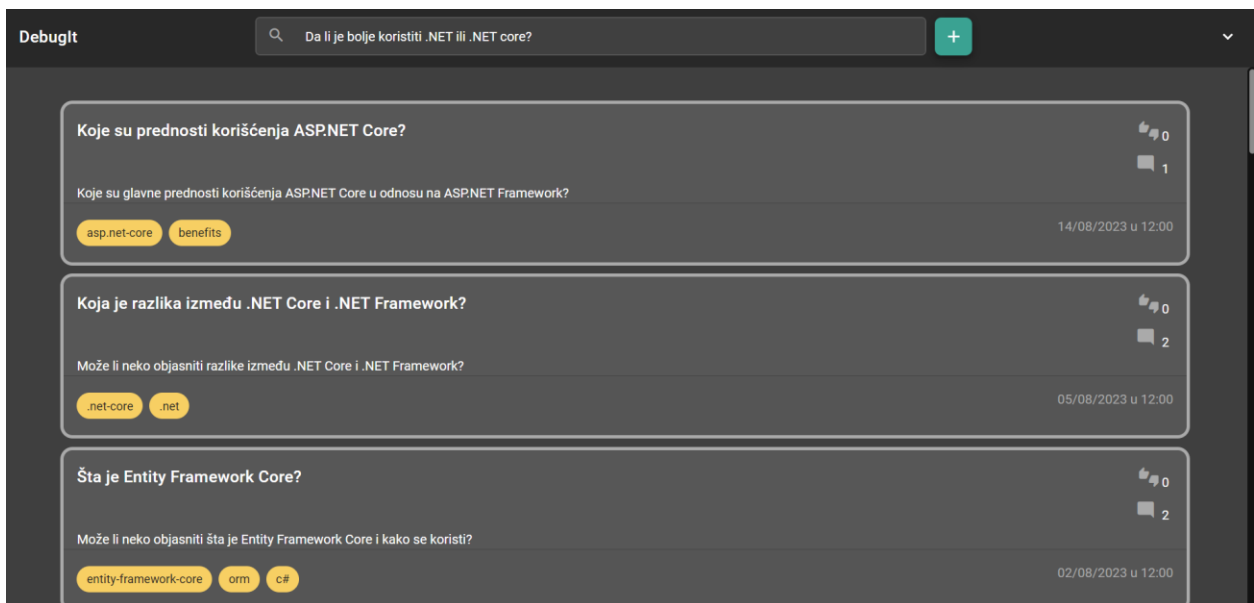


Slika 17. Odgovori na pitanje

Na slikama 18 i 19 se nalaze primeri pretrage sa rezultatima sortiranim tako da se na vrhu nalazi najrelevantniji.

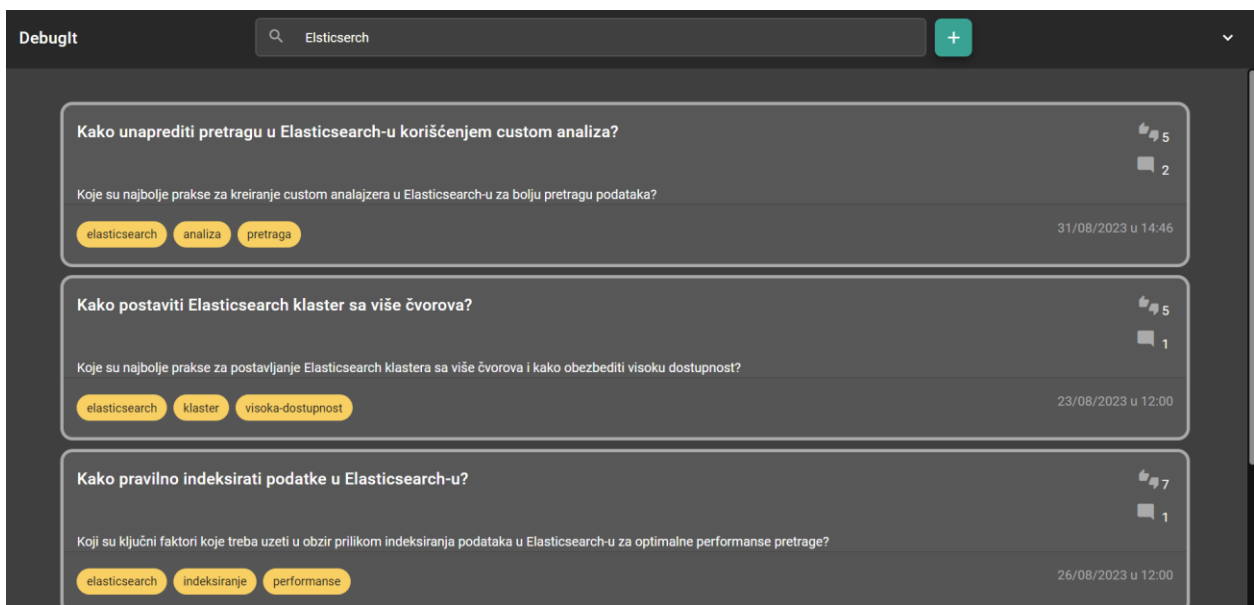


Slika 18. Primer pretrage 1



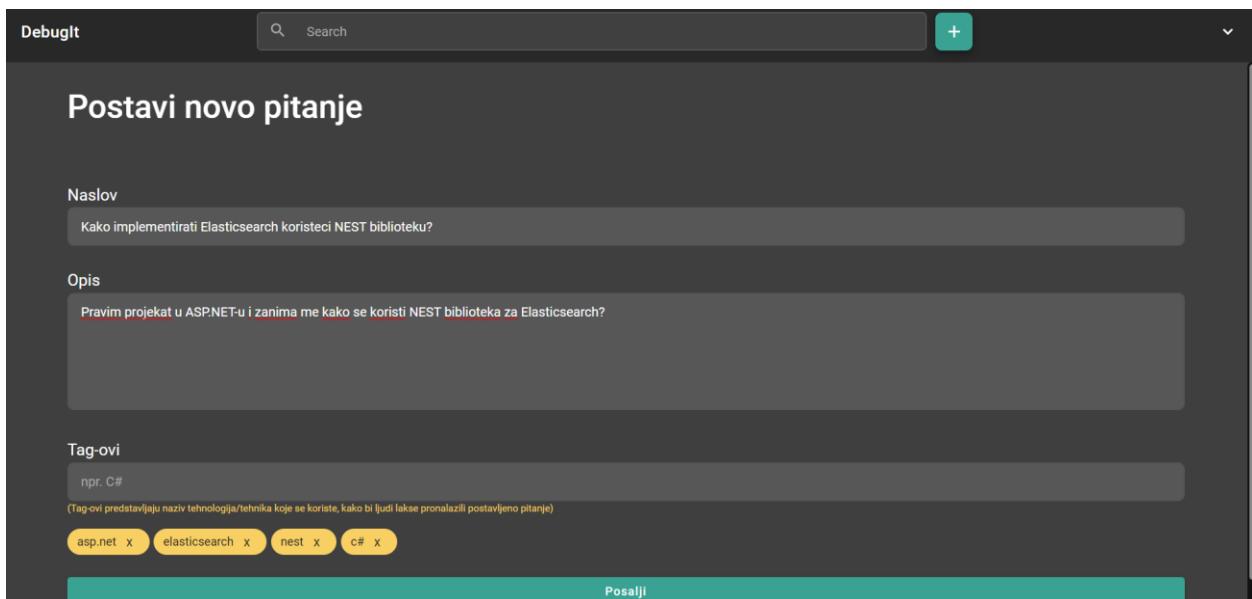
Slika 19. Primer pretrage 2

Na slici 20 se nalazi primer pretrage u čijem upitu postoji greška pri kucanju.



Slika 20. Primer greške pri kucanju

Na slici 21 se nalazi primer kreiranja novog pitanja. Potrebno je navesti tagove koji predstavljaju kratke opise tehnologija ili tehnika za koje je pitanje vezano. Uz pomoć njih se poboljšava rezultat pretrage.



DebugIt

Search

Postavi novo pitanje

Naslov

Kako implementirati Elasticsearch koristeći NEST biblioteku?

Opis

Pravim projekat u ASPNET-u i zanima me kako se koristi NEST biblioteka za Elasticsearch?

Tag-ovi

npr. C#

(Tag-ovi predstavljaju naziv tehnologija/tehnika koje se koriste, kako bi ljudi lakše pronalazili postavljeno pitanje)

asp.net x elasticsearch x nest x c# x

Posalji

Slika 21. Kreiranje novog pitanja

5.2 Izazovi u realizaciji

Što su bolje konfigurisana podešavanja za *text analysis* proces, to će pretraga davati bolje rezultate. Pod tim se misli na odabir odgovarajućih *token filters* i *analyzer* komponenti.

DebugIt aplikacija koristi *synonyms set* predstavljen putem fajla. Postoje tri načina čuvanja *synonyms set* i ovo nije najbolji način, iz razloga što je sa svakim ažuriranjem fajla potrebno okačiti ažurirani fajl svakoj *node* komponenti u *cluster* komponenti i ponovo učitati *analyzer*. Najbolje rešenje bi bio *synonyms API* pristup. Osim toga, mogao bi se kreirati AI mehanizam koji bi mogao da na osnovu dobijene reči prepozna sinonime.

Stop words poboljšavaju i ubrzavaju pretragu tako što se reči koje ne donose značaj pretrazi ignorišu. Trenutno rešenje jeste korišćenje fajla sa pobrojanim rečima, kojih zasigurno ima mnogo više, ali bi bilo vrlo mukotrpno pronaći ih sve i zapisati.

Za *stemmer* je izabran *algorithmic stemmer* koji ima podršku za rad za srpski jezik. Predstavlja set pravila koja se primenjuju na token. Srpski jezik se sastoji od dosta glasovnih promena i izuzetaka. To znači da *algorithmic stemmer* ne daje dovoljno dobre rezultate, uzimajući u obzir da uglavnom skida prefikse i sufikse sa reči. Primer problema bi bio ukoliko bi korisnik otkucio reči „koristiti” i „korišćenje”. Iako su iste reči u pitanju, zbog glasovnih promena bi imale drugačiji koren, pa bi korisniku bio ponuđen i drugačiji rezultat. Drugo potencijalno rešenje bi bilo korišćenje *dictionary stemmer*. Međutim, ni ovo nije ni približno dobro rešenje jer je veoma teško naći najnovije izdanje rečnika i pritom korišćenje rečnika zahteva učitavanje njegovog celokupnog sadržaja u *RAM* kako bi mogao da se koristi. Nijedno od dva postojeća rešenja nije najbolje moguće kada je u pitanju traženje korena reči srpskog jezika. Potencijalno rešenje bi bilo uvođenje AI mehanizma koji bi bio istreniran da prepoznaje korene reči.

5.3 Potencijalna unapređenja

Pored izazova koji su navedeni u glavi 5.3, postoje još razna moguća poboljšanja, od kojih bi neka bila:

- prelazak na *Elastic.Clients.Elasticsearch* biblioteku - *NEST* biblioteka, iako oficijalna biblioteka za *Elasticsearch*, više neće imati nove *features*. Razlog tome je što se *NEST* biblioteka koristi za *Elasticsearch* verzije 7.X, dok se za nove verzije, odnosno 8.X verzije, koristi nova biblioteka *Elastic.Clients.Elasticsearch*. Dugoročno gledano, bilo bi bolje preći na novu biblioteku, s obzirom da će sve novine koje uvodi *Elasticsearch* biti dostupne u okviru nje [44].
- dodavanje *normalizer* komponente - *normalizer* komponenta je slična *analyzer* komponenti. Ključna razlika je u tome što ne poseduje *tokenizer* komponentu, što znači da vraća samo jedan token. Koristi se prilikom indeksiranja i pretrage polja koja su tipa *keyword*, što znači da bi se pretraga u okviru *Tags* polja mogla poboljšati.

6. Zaključak

Većina Internet sajtova ima neki oblik pretrage, za koju se najčešće koristi neki od mehanizama kako bi se njeni rezultati poboljšali. Zahvaljujući mehanizmima pretrage, korisnici na veoma jednostavan način mogu doći do informacija koje su im potrebne. Osim dobijanja odgovarajućeg skupa informacija, jedan od glavnih zahteva je i brzina izvršavanja pretrage. *Elasticsearch*, poštujući oba, predstavlja odličan primer mehanizma pretrage koji zasigurno podiže kvalitet sajtova koji ga koriste za čitav nivo.

Mogućnost postavljanja i prikupljanja pitanja u okviru *DebugIt* platforme ne bi imala smisla bez odgovajućeg načina da se vrši njihovo pretraživanje, u čemu upravo pripomaže *Elasticsearch* mehanizam pretrage. Studenti se, koristeći ovu platformu, od samog početka navikavaju na istraživanje i postavljanje pitanja kada im je potrebna pomoć, ali i na pomaganje kolegama. Osim što će pročitati relevantne stvari vezane za problem koji imaju, zasigurno će u toku svog istraživanja pročitati i neke dodatne informacije koje će im u nekom momentu biti korisne. Na takav način se dosta poboljšava proces učenja.

Svi studenti jednog fakulteta čine jednu ogromnu zajednicu i kao takvoj joj je potrebno mesto u okviru koje će moći da se vode diskusije, što se upravo postiže korišćenjem *DebugIt* platforme.

Literatura

- [1] Capitol Technology University, „Alan Emtage Creator of Archie, the World's First Search Engine“, <https://www.capttechu.edu/blog>, Artikel o kreatoru prvog mehanizma pretrage, pristupano: avgust 2024.
- [2] LinkedIn, „Archie: The first internet search engine“, <https://media.licdn.com/dms/image>, Slika Archie interfejsa, pristupano: avgust 2024.
- [3] Stackscale Group Aire, „Archie, the first Internet search engine“, <https://www.stackscale.com/blog>, Artikel o prvom mehanizmu pretrage, pristupano: avgust 2024.
- [4] DBPedia, „History of Yahoo!“, <https://dbpedia.org>, Kratka istorija o nastanku Yahoo! mehanizma pretrage, pristupano: avgust 2024.
- [5] Chatri Sityodtong, „Lessons From Yahoo's Fall: \$125b to \$5b“, <https://chattrisityodtong.com/blog/entrepreneurship>, Artikel o tome kako je Google zamenio Yahoo!, pristupano: avgust 2024.
- [6] Vecteezy, <https://static.vecteezy.com>, Google interfejs, pristupano: avgust 2024.
- [7] Google, „From the garage to the Googleplex“, <https://about.google>, Nastanak Google kompanije, pristupano: avgust 2024.
- [8] BusinessHistory, „Yahoo!: The first king of the internet“, <https://businesshistory.domain-b.com/focus>, Nastanak i razvijanje Yahoo! mehanizma pretrage, pristupano: avgust 2024.
- [9] Search Engine Land, „A guide to Google: Origins, history and key moments in search“, <https://searchengineland.com/guide>, Istorija Google kompanije i njihovog mehanizma pretrage, pristupano: avgust 2024.
- [10] Stack Overflow, „Welcome to Stack Overflow“, <https://stackoverflow.com>, Upoznavanje sa Stack Overflow sajtom, pristupano: avgust 2024.
- [11] Stack Exchange, „The world's largest programming community is growing“, <https://stackexchange.com>, O Stack Exchange mreži, pristupano: avgust 2024.
- [12] Elastic, „About Discuss the Elastic Stack“, <https://discuss.elastic.co>, O Discuss the Elastic Stack sajtu, pristupano: avgust 2024.
- [13] GitHub Community, „Overview“, <https://github.com/community>, Opis čemu je namenjen GitHub Community, pristupano: avgust 2024.
- [14] Elastic, „Recipes, Elasticsearch. More recipes, morphood + Elasticsearch“, <https://www.elastic.co/elasticon>, Kratak pregled za Webinar, pristupano: avgust 2024.
- [15] Jay Gopalakrishnan, „Elasticsearch: What It Is, How It Works, And What It's Used For“, <https://www.knowi.com/blog>, Informacije o tome šta je i kako radi Elasticsearch, pristupano: avgust 2024.
- [16] GeeksForGeeks, „Difference between Inverted Index and Forward Index“, <https://www.geeksforgeeks.org>, Opis razlike između inverted i forward index strukture podataka sa primerima, pristupano: avgust 2024.
- [17] Opster Team, „Elasticsearch Shards“, <https://opster.com/guides/elasticsearch>, Primeri i dobre prakse korišćenja shard komponenti, pristupano: avgust 2024.
- [18] Opster Team, „Understanding Shards in Elasticsearch“, <https://opster.com/guides/elasticsearch>, Osnove shard komponenti u Elasticsearch mehanizmu pretrage, pristupano: avgust 2024.

- [19] Elastic, „Size your shards“, <https://www.elastic.co/guide/en/elasticsearch/reference/7.17>, Preporuke o odabiru broja shard komponenti, pristupano: avgust 2024.
- [20] Nader Medhat, „Understand Database Sharding The Good and Ugly“, <https://nadermedhatthoughts.medium.com> , Objašnjenje sharding procesa i slika kao primer, pristupano: avgust 2024.
- [21] DrTech, „Understanding Segments in Elasticsearch“, <https://stackoverflow.com/questions>, Odgovor na postavljeno pitanje koji objašnjava ulogu segment komponenti u Elasticsearch mehanizmu, pristupano: avgust 2024.
- [22] Elastic, „Ingest pipelines“, <https://www.elastic.co/guide/en/elasticsearch>, Teorija i primeri, pristupano: avgust 2024.
- [23] Elastic, „Transforming data“, <https://www.elastic.co/guide/en/elasticsearch>, Teorija, pristupano: avgust 2024.
- [24] Elastic, „Nodes“, <https://www.elastic.co/guide/en/elasticsearch>, Objašnjenje node komponenti u Elasticsearch mehanizmu pretrage, pristupano: avgust 2024.
- [25] Sematext, „How to Find and Fix Elasticsearch Unassigned Shards“, <https://sematext.com>, Slika kao primer strukture cluster, node, shard i index komponenti, pristupano: avgust 2024.
- [26] Elastic, „Index templates“, <https://www.elastic.co/guide/en/elasticsearch>, Teorija i primeri, pristupano: avgust 2024.
- [27] logz.io, „Elasticsearch Mapping: The Basics, Updates & Examples“, <https://logz.io/blog/elasticsearch-mapping>, Objašnjenja za data mapping sa primerima, pristupano: avgust 2024.
- [28] Elastic, „Aliases“, <https://www.elastic.co/guide/en/elasticsearch/reference/7.17>, Teorija i primeri , pristupano: avgust 2024.
- [29] Elastic, „Writing queries“, <https://www.elastic.co/guide/en/elasticsearch/client/net-api/7.17>, Pisanje upita korišćenjem NEST biblioteke , pristupano: avgust 2024.
- [30] Opster Expert Team - Madhusudhan, „Elasticsearch Text Analyzers - Tokenizers, Standard Analyzers, Stopwords and more“, <https://opster.com/guides/elasticsearch>, Način rada Elasticsearch Analyzer mehanizma, pristupano: septembar 2024.
- [31] Elastic, „Anatomy of an analyzer“, <https://www.elastic.co/guide/en/elasticsearch>, Struktura analyzer modula, pristupano: septembar 2024.
- [32] Elastic, „Writing analyzers“, <https://www.elastic.co/guide/en/elasticsearch>, Slika koja prikazuje strukturu analyzer modula, pristupano: septembar 2024.
- [33] Elastic, „Normalizers“, <https://www.elastic.co/guide/en/elasticsearch>, Objašnjenja vezano za normalizer, pristupano: septembar 2024.
- [34] Elastic, „Stemming“, <https://www.elastic.co/guide/en/elasticsearch>, Objašnjen način rada stemming procesa, pristupano: septembar 2024.
- [35] Opster Team, „Elasticsearch Stop Words“, <https://opster.com/guides/elasticsearch>, Objašnjenje sa implementacijom za stop words, pristupano: septembar 2024.
- [36] Elastic, „Stop token filter“, <https://www.elastic.co/guide/en/elasticsearch>, Stop words token filter, pristupano: septembar 2024.
- [37] Elastic, „Synonyms APIs“, <https://www.elastic.co/guide/en/elasticsearch>, Synonyms APIs kao način čuvanja synonyms sets, pristupano: septembar 2024.
- [38] Elastic, „Search with synonyms“, <https://www.elastic.co/guide/en/elasticsearch>, Značaj i korišćenje synonyms kod pretrage, pristupano: septembar 2024.

- [39] Elastic, „Practical BM25 - Part 2: The BM25 Algorithm and its Variables“, <https://www.elastic.co/blog>, Princip rada BM25 algoritma u okviru Elasticsearch mehanizma, pristupano: septembar 2024.
- [40] Opster Team, „Understanding Elasticsearch Scoring and the Explain API“, <https://opster.com/guides/elasticsearch>, Način rada scoring mehanizma u Elasticsearch mehanizmu pretrage, pristupano: septembar 2024.
- [41] Elastic, „Limiting the relevancy score of all the searched documents“, <https://discuss.elastic.co>, Primer koji objašnjava šta se desi kada se ograniči score vrednost, pristupano: septembar 2024.
- [42] CodeCurated, „How to Handle Typos in Elasticsearch Using Fuzzy Query“, <https://codecurated.com/blog>, Način korišćenja za fuzzy query, pristupano: septembar 2024.
- [43] Hernan Velasquez, „Understanding and tuning fuzzy queries in Elasticsearch by example“, <https://dev.to>, Objašnjenje za fuzzy query, pristupano: septembar 2024.
- [44] Elastic, „Release notes v8.0.0“, <https://www.elastic.co/guide/en/elasticsearch>, Prelazak na novu Elasticsearch verziju i menjanje biblioteke koja se koristi za .NET, pristupano: septembar 2024.

Podaci o kandidatu

Kandidat Nevena Gligorov je rođena 2001. godine u Kraljevu. Završila je srednju školu u Kraljevu, 2020. godine. Fakultet Tehničkih Nauka u Novom Sadu je upisala 2020. godine. Ispunila je sve obaveze i položila je sve ispite predviđene studijskim programom sa prosečnom ocenom od 9.62.