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 oktucanim upitom.

***Istorijat***

Alan Emtage, administrator sistema za departman informacionih tehnologija i student kanadskog Univerziteta McGill, u septembru 1990. godine objavljuje prvi Internet mehanizam pretrage (*eng. search engine*) pod nazivom *Archie*. *Archie* je predstavljao indeks FTP fajlova i njegovi korisnici su uz pomoć jednostavnih upita mogli da ih pretražuju.

Nekoliko godina i mehanizama pretrage kasnije, apsolutnu dominaciju u Internet svetu ima *Yahoo*, čija struktura predstavlja direktorijum koji se sastoji od Internet sajtova, prilikom čega su sajtovi organizovani hijerarhijski. Njegovi kreatori su studenti Stanford Univerziteta, Jerry Yang i David Filo.

1998. godine dvojica studenata, Larry Page i Sergey Brin, koji su takođe sa Stanford Univerziteta odlaze u *Yahoo* sa ponudom da im prodaju svoj biznis za milion dolara, međutim bili su odbijeni. Ono što tada *Yahoo* nije znao jeste da je odbio budućeg giganta koji i dan danas vlada Internetom

Larry i Sergey su želeli da „organizuju sve informacije sveta i da ih učine univerzalno pristupačnim i korisnim“, pa su napravili mehanizam pretrage zasnovan na *PageRank* algoritmu.

Uzmiajući u obzir prednosti korišćenja mehanizama pretrage i mnogobrojne projekte i pitanja koje studenti postavljaju prilikom njihove izrade, dolazi se do DebugIt-a, platforme koja obuhvata 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.

***Postojeća rešenja***

*Stack Overflow* je sajt, zamišljen da predstavlja biblioteku koja bi umesto knjiga na svojim policama imala sva moguća pitanja vezana za programiranje, sa najkvalitetnijim i vrlo detaljnim odgovorima. Predstavlja 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.

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

*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.

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

***Elasticsearch mehanizam pretrage***

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

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

***Struktura Elasticsearch mehanizma pretrage***

Nabrojati samo kao na slici.

***Index***

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

Postoje 2 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.
* ***inverted index*** - termini se mapiraju na dokumente u kojima se nalaze. To znači da za svaku reč postoji lista dokumenata u kojima se nalazi.

***Shard***

*Shard* predstavlja osnovnu gradivnu jedinicu *Elasticsearch* distribuirane arhitekture. Indeks može da sadrži veoma veliku količinu podataka, a bi upravljanje podacima bilo što efikasnije, ti podaci se obično podele u nekoliko *shard* komponenti.

Postoje 2 tipa *shard* komponenti:

* **primarni** (*eng. primary*) - sadrže originalne podatke i obavljaju indeksiranje i pretragu.
* **kopije** (*eng. replica*) - predstavljaju kopiju primarnih *shard* komponenti. Uz pomoć njih se uvodi redudantnost i poboljšavaju se performanse pretrage, jer omogućavaju izvršavanje paralelnih upita.

***Segment***

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

Prilikom indeksiranja dokumenta otprilike svake sekunde se kreira novi *segment*.

Sadržaj *segment* komponente se ne može menjati.

Što je više *segment* komponenti, time je pretraga sporija. Zbog toga se vrši spajanje *segment* komponenti istih veličina, počev od manjih.

***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, data node, ingest node, remote-eligible node, machine-learning node, transform node, coordinating node.

***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.

*Data mapping* je šema *index* komponente, odnosno, sadrži informacije o svim poljima i njihovima 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, kofigurišići imena polja i njihove tipove.

***Alias***

Aliasi se koriste za grupisanje indeksa, kada je na primer potrebno videti logove iz svih *index* komponenti na jednom mestu.

***Način rada Elasticsearch mehanizma***

Kako korisnci znaju od svih ponuđenih dokumenata koji je najbolji, odnosno, koji najviše odgovara njihovom upitu? *Elasticsearch* je rešenje našao u tome da svakom dokumentu dodeli *score*. 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. *Elasticsearch* ne dodeljuje *score* dokumentu, već „*da*“ ili „*ne*“, što označava da dokument odgovara ili ne odgovara upitu.
* **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*. Najčešće se koristi za upite nad tekstualnim poljima.
* **kombinovni upiti** (*eng. combining queries*) - kombinacija upita koja se naziva *compound query*, kombinuje prva dva tipa pretraga.

***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 posmatranju sinonima, *stop words* i tako dalje.

*Analyzer* modul izvršava *text analysis* proces i sastavljen je od tri tipa komponenti kroz koje tekst prolazi u toku *text analysis* procesa:

* ***character filters*** - koriste se za dodavanje, menjanje i brisanje karaktera iz niza karaktera kojim je predstavljen tekst na ulazu. *Analyzer* modul može imati nijedan ili više *character filter* komponenti.
* ***tokenizer*** - tekst predstavljen kao niz karakter 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. *Analyzer* modul može imati nijedan ili više *token filters* komponenti.

***Token filters***

*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.

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

* ***algorithmic stemmers*** - koriste set pravila koja se primenjuju na tokene. Primer bi bio skidanje prefiksa i sufiksa sa reči.
* ***dictionary stemmers*** - koriste rečnik koji je potrebno navesti pri implementaciji. Za svaku reč, odnosno token, traže koren u okviru datog rečnika.

*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 svakodnevno govoru, a za srpski jezik bi to bili: veznici, zamenice, predlozi, pomoćni glagoli i slično.

*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.

*Scoring* mehanizam se zasniva na *BM25* algoritmu. 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 ovkiru 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.

Prilikom pretrage, često se desi da korisnici pogrešno otkucaju reč ili nekoliko reči u celom upitu. Ipak, idalje žele da dobiju iste rezultate kao i u slučaju tačno otkucanih reči. *Elasticsearch* je to rešio korišćenjem *fuzzy query* koji koristi Levenštajnov algoritam udaljenosti (*eng. Levenshtein Distance Algorithm*) da bi izračunao udaljenost reči iz upita od onih koje se nalaze u *index* komponenti.

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.

Jedna od čestih grešaka pri kucanju je transponovanje slova. Na primer, u reči scoer 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 koji omogućava da se kod transponovanih slova distanca računa kao +1 na trenutnu vrednost distance.