|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  PRIRODNO-MATEMATIČKI FAKULTET  DEPARTMAN ZA MATEMATIKU I INFORMATIKU |  |

**Indeksiranje i pretraživanje naučno-istraživačkih časopisa otvorenog pristupa**

* Master rad -

**Mentor: Student:**

**dr Miloš Savić Jovan Vučetić**

**Novi Sad, 2020. godine**

***Predgovor***

Informacije predstavljaju najvredniji resurs modernog društva. Njihovim prikupljanjem i organizacijom formira se znanje, potrebno za izvršavanje različitih radnji korisnih čoveku, zbog čega je pristup dobrom izvoru informacija veoma važan. Nastanak i razvoj interneta vremenom je doveo do toga da se veliki broj radnji koje čovek vrši u moderno doba može izvršiti putem interneta. Radnje koje čovek najčešće obavlja korišćenjem interneta su svakako komunikacija i pretraživanje podataka. Korišćenjem savremenih tehnologija čovek je danas u mogućnosti da veoma lako pristupi velikom broju podataka iz raznih oblasti. Internet je od svog nastanka do danas postao najveća baza znanja poznata ljudskoj istoriji, te danas sadrži neverovatno velike količine podataka, zbog čega razmenjivanje znanja ima tendenciju potpunog prelaska na internet. Podaci na internetu uglavnom se nalaze u nestruktuiranoj formi u kojoj nisu naročito značajni čoveku. Takođe, kako je broj podataka dostupnih na internetu prevelik za obradu od strane čoveka, veoma je važno postojanje sistema za pretraživanje podataka, kako bi čovek mogao pristupiti željenim podacima na jednostavan način. Sistemi za web pretraživanje su gotovo nezaobilazni deo interneta u današnjem svetu. Korišćenjem sistema za web pretraživanje čovek ima mogućnost brzog i jednostavnog pristupa bilo kojim podacima dostupnim na internetu. Sistemi web pretraživanja mogu imati opštu namenu, primer ovakvih pretraživača su Bing ili Google koji imaju mogućnost pretraživanja svih dostupnih podataka na internetu bez ikakvih domenskih ograničenja. Takođe postoje i mnogi sistemi pretraživanja koji su usmereni na određene domene podataka koji se pretražuju. Glavni zadatak sistema za pretraživanje informaija je prikupljanje podataka i njihovo struktuiranje. Ovim procesom od potataka se dobijaju informacije koje korisnik može iskoristiti. Takođe, važan zadatak sistema pretraživanje jeste omogućavanje pristupa tim informacijama na osnovu korisničkog upita. Oblast informatike koja se bavi ovim temama jeste pretraživanje informacija.

Pretraživanje informacija podrazumeva pronalazak podataka koji se nalaze u nestruktuiranoj formi, iz velikih kolekcija podataka, na osnovu određenog korisničkog upita. Drugim rečima pretraživanje informacija obezbeđuje predstavljanje, skladištenje, organizaciju i pristup informacijama [1]. Ova oblast informatike nudi mnoge tehnike za brzu računarsku pretragu velikog obima podataka i njihovo grupisanje i rangiranje na osnovu važnosti za korisnika. Na ovim tehnikama se zasnivaju svi moderni pretraživači, što definiše i sam način korišćenja interneta danas.

U ovom radu predstavljeni su osnovni koncepti pretraživanja informacija kroz teorijsko poglavlje, te je kroz implementaciju manjeg sistema pretraživanja demonstrirana primena ovih koncepata.

Ovaj rad sastoji se iz ukupno četiri poglavlja:

* Uvod
* Dizajn i implementacija
* Demonstracija i testiranje
* Zaključak

U prvom poglavlju ovog rada opisani su teorijski osnovi nekih od osnovnih tema kojima se pretraživanje informacija bavi, kao što su: prikupljanje nestruktuiranih podataka u vidu web stranica, skladištenje podataka u indeks fajlove, kao i pretraživanje indeksiranih podataka na osnovu korisničkog upita.

Pored teorijskog poglavlja, ovo istraživanje obuhvata i implementaciju web aplikacije za pretraživanje informacija o naučnim radovima objavljenih u časopisu otvorenog pristupa. U drugom poglavlju ovog rada biće prikazan dizajn i implementacija ove web aplikacije sa svim njenim značajnim delovima.

Treće poglavlje baviće se demonstracijom kreirane aplikacije, i prikazom svih njenih funkcionalnosti. Ovo poglavlje će objasniti način upotrebe aplikacije i prikazati neke od razultata značajnih za ovo istraživanje.

Zaključak ovog rada, kao i ideje za potencijalna proširenja, prikazani su u četvrtom, finalnom poglavlju.

*Ovom prilikom želim da izrazim veliku zahvalnost dr Milošu Saviću, na prenetnom znanju u toku školovanja, kao i na velikoj pomoći prilikom izrade ovoga rada.*

*Zahvaljujem se i porodici i prijateljima na velikoj podršci u dosadašnjem školovanju.*

## Uvod

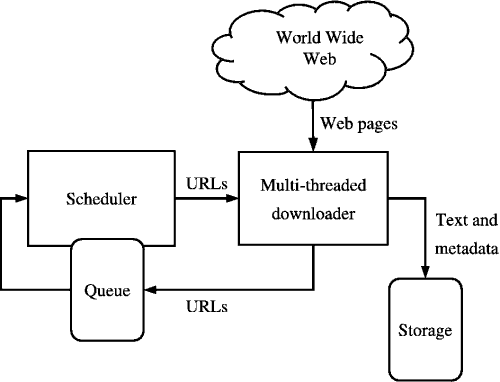
Pretraživanje informacija u savremenom svetu predstavlja jednu od najpopularnijih oblasti istraživanja. Kreiranje brzog i jednostavnog sistema za pretraživanje podataka koji se nalaze u nestruktuiranoj formi na webu predstavlja veliki izazov. Proces kreiranja ovakvog sistema za pretraživanje može se podeliti u tri glavne faze:

* Prikupljanje podataka
* Skladištenje prikupljenih podataka
* Pretraživanje podataka i njigovo rangiranje na osnovu korisničkog upita

Svaka od ove tri faze postavlja posebne izazove, koje je potrebno savladati kako bi se uspešno kreirao sistem za pretraživanje informacija. U nastavku ovog poglavlja objašnjeni su izazovi svake od ovih faza, kao i teorijski osnovi za rešenje problema koje te faze postavljaju.

* **Prikupljanje podataka**

World wide web predstavlja najveći izvor znanja poznat ljudskoj istoriji, te se kao takav najčešće koristi za prikupljanje podataka svih vrsta. Podaci na webu se najčešće nalaze u nestruktuiranim oblicima, kao što su tekstualni članci predstavljeni u formi web stranica koje za cilj imaju da podatke predstave korisniku na način koji će za čoveka biti intuitivan. Ipak kada je u pitanju rad sa velikim obimom podataka koje prikuplja i obrađuje računar, ovaj način predstavljanja podataka nije naročito pogodan. Ipak, kako struktuirani grupisani podaci nisu uvek dostupni za prikupljanje, prikupljanje podataka sa web stranica predstavlja jedini mogući izbor. Najčešća tehnika u prikupljanju podataka sa web stranica jeste korišćenje web crawler komponenti. Ove komponente koriste grafovsku strukturu world wide weba u cilju omogućavanja pretrage međusobno povezanih stranica bez ljudske interakcije. Suština web crawler komponente jeste pretraga html sadržaja web stranica iz kojeg je moguće prikupiti tražene podatke, kao i linkove ka daljim stanicama od interesa. Pretraga web sadržaja korišćenjem web crawlera može biti uopštena ili fokusirana na određeni opseg stranica. Opšta pretraga je proces koji najčešće vrše sistemi pretrage kao što su Google ili Bing. Jedna od najvažnijih osobina dobrog web pretraživača jeste rangiranje stranica prema njihovoj važnosti. Pionir ove oblasti bila je upravo kompanija Google koja je uvela inovacije u sve pretraživanja informacija korišćenjem Page Ranka, algoritma za određivanje „važnosti“ stranice na osnovu njenog položaja u usmerenom web grafu. Kada je u pitanju uopštena pretraga weba koju vrše sistemi za pretraživanje, web crawler komponenta analizira sve dostupne stranice na webu, te ih rangira prema važnosti. Ovaj proces se razlikuje kada je u pitanju usmerena pretraga. Ovakav tip pretrage fokusiran je na određeni domen znanja, kao što je jedan skup web stranica u okviru nekog web sajta. Web crawler komponenta ima za cilj pristupanje skupu međusobno povezanih stranica korišćenjem nekog od algoritama za obilazak grafova, kao i prikupljanje značajnih podataka tokom njihovog obilaska. Na slici 1.1.1. moguće je videti pojednostavljenu arhitekturu web crawler komponente.



Slika 1.1.1. Arhitektura web crawler komponente [1]

Kao što je moguće videti na slici web crawler sastoji se iz nekoliko osnovnih delova. Web crawler najčešće koristi više procesa prilikom preuzimanja i analize html stranica kako bi se sama pretraga ubrzala. Deo koji vrši preuzimanje i analizu html sadržaja stranicama na slici je označen kao „Multi-threaded downloader“. Kako bi se ovi procesi preuzimanja i skladištenja uspešno orkestrirali potrebana je komponenta koja ih pokreće i dodeljuje im linkove ka stranicama koje je potrebno posetiti, ova komponenta na slici je označena kao „Scheduler“. Takođe web crawler mora sadržati queue, odnosno kolekciju linkova ka stranicama koje je potrebno posetiti. Takođe, preporučeno je i korišćenje kolekcije stranica koje su već posećene kako bi se izbeglo posećivanje stranica koje su već posećene, što može izazvati i zaglavljivanje web crawlera u slučaju pojave konture u web grafu. Poslednji deo web crawler komponente jeste skladište podataka izvučenih za vreme analize prikupljenih web stranica.

Proces web pretraživanja započinje određivanjem skupa početnih stranica za obilazak. Prilikom ovog koraka je najvažnije pažljivo odabrati stranice iz kojih se putem linkova može stići do što većeg broja drugih stranica od značaja. Važnost ovog koraka se ogleda u tome da ukoliko se odabir početnih tačaka pretraživanja ne izvrši na adekvatan način, celokupno pretraživanje može biti neuspešno. Prilikom posete svake stranice, njen link se najpre prebacuje iz skupa neposećenih u skup posećenih stranica, nakon čega se sa stranice preuzimaju svi potrebni podaci. Poslednji korak koji se obavlja na svakoj stranici je izvlačenje linkova ka ostalim stranicama, i njihovo smeštanje u kolekciju neposećenih ukoliko stranice na tim linkovima već nisu posećene. Kraj procesa web pretraživanja nastupa u trenutku kada kolekcija neposećenih linkova više ne sarži ni jedan element, ili se ovaj proces prekine od strane orkestratora.

Kako bi web crawler komponenta imala smisao, potrebno je implementirati segment za analizu html sadržaja web stranice, kao i parsiranje podataka sa iste. Prilikom posete svake od stranica komponenta zadužena za prikupljanje podataka izvršava tri koraka: pristupanje traženim html elementima, parsiranje vrednosti elemenata u podatke i struktuiranje i čuvanje podataka. Kada su u pitanju prva dva koraka, oni uveliko zavise od same html strukture stranice, što je veoma često najveći izazov prilikom implementacije komponenti za prikupljanje podataka. Kako bi pravilno prikupljanje podataka sa web stranice bilo moguće, sama stranica mora imati jasno definisanu strukturu. Takođe kako bi se omogućilo lakše preuzimanje podataka, web stranice često koriste identifikacione html atribute kao što su id ili data atributi, pomoću kojih web crawler komponente mogu direktno pristupiti željenim html elementima i njihovim vrednostima. Nakon prikupljanja traženih podataka sa stranice, poslednji korak jeste struktuiranje i čuvanje istih. Ovaj proces počinje dizajnom strukture podataka, koja zavisi od samog cilja istraživanja. Ove strukture podataka često se nazivaju i modeli. Modeli u svojoj strukturi sadrže atribute čije vrednosti bivaju određene na osnovu prikupljenih podataka iz prethodnih koraka. Prilikom kreiranja modela, potrebno je dodeliti vrednost svim atributima koji su dostupni i neophodni za dalje istraživanje. Nakon što su novi podaci prikupljeni i struktuirani u modele, najčešća praksa jeste njihovo skladištenje, kako bi se omogućio kasniji pristup bez potrebe za ponovnom analizom web sadržaja.

Kao što je moguće videti iz prethodnih nekoliko pasusa, upotreba web crawler komponenti je veoma prost ali efikasan način pravilnog prikupljanja podataka koji se nalaze u formatu web stranica. Korišćenjem pogodnosti koje ova tehnika prikupljanja podataka pruža, nestruktuirani podaci sa weba se na jednostavan način mogu prikupiti i struktuirati, kako bi bili spremni za skladištenje i dalju upotrebu.

* **Skladištenje podataka**

Prilikom implementacije sistema za pretraživanje informacija veoma je važno pronaći adekvatan način skladištenja podataka. Kada je implementacija ovakvih sistema u pitanju, korišćenje klasičnih relacionih baza podataka često nije najbolja opcija. Razlog ovome jeste činjenica da su korisnički upiti, kao i sami podaci vrlo često nestruktuirani i nepotpuni, te nisu pogodni za skladišta podataka kao što su relacione baze. Takođe kod sistema za pretraživanje informacija veoma je važno rangiranje rezultata prema važnosti za korisnika, što nije moguće korišćenjem baza podataka. Kako bi se ove prepreke zaobišle pretraživači često koriste indekse, specijalne strukture podataka koje omogućavaju brzo pretraživanje podataka. Ove strukture podataka omogućavaju skladištenje nestruktuiranih podataka kao što su tekstualni podaci, i njihovo pretraživanje na osnovu korisničkog upita, koji ne mora biti potpun niti tačan kako bi se pretraživanje uspešno izvršilo. Skladištenje podataka u indeks zasniva se na takozvanim indeks termovima, odnosno ključnim rečima. Prilikom indeksiranja podataka, nad njihovim tekstom je moguće vršiti različite transformacije koje mogu služiti optimizaciji pretraživanja i rangiranju rezultata. Ove transformacije mogu obuhvatati uklanjanje dijalekata, delova teksta koji služe za formatranje, veznika, kao i svođenje reči na njihov koren i slično. U slučaju izvršavanja ovakvih transformacija veoma je važno primeniti iste i na svaki korisnički upit, kako bi se pretraga zaista optimizovla. Rezultati pretrage indeks fajlova su dokumenti, koji se rangiraju prema važnosti za korisnički upit. Ovakav format skladištenja podataka u svetu savremenih tehnologija vrlo često je povezan sa mašinskim učenjem, koje za cilj ima unapređivanje korisničkog iskustva brzinom pretrage i relevantnošću vraćenih rezultata. Postizanje ovog cilja korišćenjem mašinskog učenja zasniva se na automatizovanoj klasifikaciji i klasterizaciji dokumenata, kao i automatizovanom izvlačenju podataka i njihovom rangiranju na osnovu korisničkog upita. Takođe oblast nauke s kojom se ovakav tip skladištenja podataka bavi jeste procesiranje prirodnih jezika. Upotreba ove oblasti zasniva se na sintaksnoj i semantičkoj analizi teksta koji se indeksira. Ova analiza omogućava premošćavanje određenih pojava u jezicima koje mogu negativno da utiču na pretragu, kao što su na primer sinonimi ili homonimi. Takođem ovakva vrsta analize omogućava autokorekciju eventualnih grešaka u rečima korisničkog upita, kao i dopunjavanje upita na osnovu konteksta.

Proces indeksiranja obuhvata prijem podataka za indeksiranje, kreiranje dokumenata, njegovo popunjavanje na osnovu ključnih reči i na kraju smeštanje kreiranih dokumenata u indeks. Prvi korak u indeksiranju započinje kada komponenta za indeksiranje dobije ulazni podatak ili skup podataka. Nakon toga sledi kreiranje dokumenta, koji predstavlja strukturu za skladištenje podataka kada su u pitanju indeks strukture. Ukoliko je ulazni parametar indeksiranja sadrži kolekciju podataka, najčešća praksa jeste kreiranje dokumenta za svaki od elemenata tog skupa. Sledeći korak jeste popunjavanje dokumenata na osnovu ulaznog parametra i ključnih reči. Prilikom popunjavanja dokumenta moguće je dodeliti jednu ili više ključnih reči na osnovu kojih je moguća kasnija pretraga. Svaka ključna reč uz sebe ima određene vrednosti koje mogu biti različitog tipa. Ovaj tip skladištenja veoma podseća na skladištenje informacija u strukturama podataka kao što su mape. Nakon što se dokument kreira on biva smešten u indekser. Indekser koji za cilj ima kreiranje indeks fajlova i smeštanje dokumenata u iste. Indekser takođe u zavisnosti od svoje konfiguracije može vršiti različite operacije nad dokumentima u cilju optimizacije kasnije pretrage. Nakon što se indeks fajlovi kreiraju oni su otvoreni za pretraživanje dokumenata na osnovu prethodno definisanih ključnih reči.

* **Pretraživanje podataka i njihovo rangiranje na osnovu korisničkog upita**

Nakon što se indeks fajlovi kreiraju poslednji korak jeste omogućavanje pretrage podataka na osnovu korisničkog upita. Kada je pretraga podataka u pitanju najveći izazov jeste predviđanje relevantnosti dokumenta na osnovu korisničkog upita. Kroz istoriju pretraživanja informacija korišćene su razne vrste IR modela. Ovi modeli definišu kako su upiti i dokumenti struktuirani kao i na koji način se relevantnost dokumenata određuje.

Prva osmišljena vrsta IR modela jeste boolean model. Ovaj IR model se zasniva na binarnoj relevantnosti dokumenata. Ovo znači da dokument može biti samo relevantan ili irelevantan, što zavisi od korisničkog upita. Ovaj model omogućava konstrukciju korisničkog upita na osnovu termova, odnosno reči koje se pretražuju, kao i logičkih operatora I, ILI i NE. Korišćenjem termova i logičkih operatora korisniku se omogućava kreiranje korisničkog upita, koji za rezultat ima sve one dokumente koji zadovoljavaju taj upit, te se na taj način ostvaruje veoma efikasna pretraga podataka. Ipak uprkos svojoj efikasnosti, pretraživanje informacija korišćenjem boolean IR modela ima mnogobrojne nedostatke. Osnovni problem ovog modela pretraživanja jeste sama binarna relevatnost dokumenta, koja onemogućava da neki dokumenti budu relevantniji od drugih na osnovu upita. Iz ove činjenice takođe proizilazi da su svi relevantni dokumenti jednaki, te da korisnik nema uvid u to koji od ponuđenih dokumenata može biti od najveće koristi bez pregledanja kompletnog dokumenta. Pored ovoga, jedna od velikih mana ovog modela pretraživanja jeste činjenica da su rezultati pretrage često preveliki ili premali, što u oba slučaja uzrokuje loše korisničko iskustvo. Takođe još jedna od mana ovakvog pristupa jeste činjenica da se upiti u boolean IR modelu moraju predstaviti kao logički izrazi, što predstavlja prepreku korisnicima koji nemaju znanje o konstrukciji logičkih izraza. Zbog ove činjenice, odgovornost programera jeste da omogući korisnicima pretragu korišćenjem prirodnog jezika, koji potom biva preveden u logički izraz pre samog čina pretrage. Zbog svih navedenih mana koje boolean model ima, dolazi do potrebe za novim IR modelom koji omogućava rangirano pretraživanje korišćenjem prirodnog jezika pri konstruisanju korisničkog upita.

Rangirano pretraživanje predstavlja unapređenu verziju prvobitnog boolean pretraživanja. Osnovni princip ove vrste pretraživanja jeste ocenjivanje relevantnosti dokumenta na osnovu korisničkog upita. Vrednost relevantnosti dokumenta u ovakvom načinu pretraživanja više nije binarna, već se može kvantitativno odrediti. Ova činjenica unapređuje korisničko iskustvo u mnogim sferama. Najvažnija prednost ovakvog pristupa jeste mogućnost sortiranja rezultata pretrage po njihovoj relevantnosti. Osim toga, ovaj pristup rešava i problem prevelikog broja vraćenih rezultata, tako što omogućava vraćanje tačno određenog broja najznačajnijih dokumenata. Takođe ovakav način pretrage podržava i pretragu na osnovu korisničkog upita koji se piše u prirodnom jeziku, što veoma unapređuje korisničko iskustvo, ali i uklanja odgovornost sa programera kada je u pitanju prevođenje upita u logički izraz. Mera relevantnosti dokumenta u pretraživanju informacija često se naziva i skor. Određivanje ove mere predstavlja jedan od najvećih izazova kada je u pitanju rangirano pretraživanje. Računanje skora vrši se za svaki dokument posebno na osnovu nivoa poklapanja sa unetim korisničkim upitom. Osnovni pristup kalkulaciji relevantnosti jeste upotreba statističke relevantnosti, odnosno korišćenje učestalosti pojave termova iz upita u samom dokumentu. Ova metrika naziva se term frekvencija i zasniva se na pretpostavci da je dokument u kome se često pojavljuju ključne reči iz upita gotovo sigurno važan za korisnika. Mana ovog pristupa jeste činjenica da značaj dokumenta linearno raste sa brojem pojavljivanja terma iz upita u posmatranom dokumentu. Ovo znači da će dokument u kome se term iz upita pojavljuje pet puta, biti pet puta važniji od dokumenta u kome se isti term pojavljuje samo jednom, što ne predstavlja naročito precizan rezultat pretrage. Kao bolja alternativa za računanje term frekvencije koristi se log frekvencija. Ovaj način zasniva se na tome da se skor dokumenta za term frekvenciju računa kao:

Ovaj unapređeni način se sada može koristiti za računanje skora. Kako bi se izračunao skor, najpre je potrebno izračunati težinu svakog terma iz upita i dokumenata koji se pretražuju. Težina terma predstavlja značaj određenog terma za pretragu koja se vrši, i ona se računa kao log frekvencija terma. Nakon što su težine svih termova izračunate, njihova suma daje skor za svaki od dokumenata.

Kako bi se rangiranje dokumenata prema nivou relevantnosti dodatno unapredilo, osim same frekvencije pojavljivanja nekog terma u dokumentu u obzir je moguće uzeti i dokument frekvenciju, odnosno broj dokumenata u celokupnoj kolekciji koji sadrže term iz upita. Ovaj pristup zasniva se na činjenici da su termovi koji se pojavljuju u malom broju dokumenata uglavnom informativniji, te su dokumenti koji sadrže ove termove najčešće relevantni za korisnika. Kako bi izračunati skor bio obrnuto srazmeran sa brojem pojavljivanja nekog terma na nivou čitave kolekcije dokumenata, koristi se inverzna dokument frekvencija, koja se računa pomoću formule:

Gde N predstavlja ukupan dokumenata, a predstavlja broj dokumenata koji sadrže term t. S obzirom da se prilikom računanja skora, računa i težina reči u upitu, upotreba inverzne dokument frekvencije ne utiče na rezultate pretrage ukoliko se korisnički upit sastoji od samo jednog terma.

Kombinovanjem term frekvencije i inverzne dokument frekvencije dobija se najoptimalniji poznati način za računanje težine termova u korisničkom upitu i dokumentima. Ova formula glasi:

Gde je term frekvencija terma t u dokumentu d, N broj dokumenata, a broj dokumenata koji sadrže term t.

Nakon računanja težine svakog od termova svaki od dokumenata, kao i upit moguće je predstaviti pomoću vektora sačinjenog izračunatih težina. Ukoliko se i upit i dokument predstave na ovaj način, važnost dokumenta za zadati upit moguće je izračunati kao sličnost između ova dva vektora. Osnovni pristup računanju sličnosti ova dva vektora jeste računanje njihove inverzne distance. Ovaj način nije naročito pogodan zbog činjenice da Euklidova distanca među vektorima različite dužine uvek ima veliku vrednost. Bolja alternativa ovom pristupu jeste korišćenje ugla među vektorima, za računanje nivoa preklapanja između vektora upita i dokumenta. Kako bi ova mera bila srazmerna sličnosti vektora, koristi se kosinusna sličnost, čija vrednost raste sa opadanjem ugla između dva vektora. Kako bi se ovo računanje dodatno optimizovalo, vektori mogu biti normalizovani po svojoj dužini. Ova normalizacija vrši se tako što se svaka vrednost u vektoru podeli sa brojem tačaka tog vektora. Vršenjem ove operacije, dokumentima različitih dužina se obezbeđuje mogućnost poređenja. Računanje kosinusne sličnost vektora upita i vektora dokumenta normalizovanih prema dužini vrši se pomoću formule:

Gde predstavlja vektor upita, vektor dokumenta, i njihove dužine respektivno, a predstavlja vektorski proizvod. Ako se u obzir uzme da su težine svih termova veće ili jednake sa 0, izvodi se zaključak da će kosinusna sličnost uvek biti . Nakon računanja kosinusne sličnosti između upita i dokumenata, moguće je vratiti zadati broj dokumenata sa najvećom sličnošću.

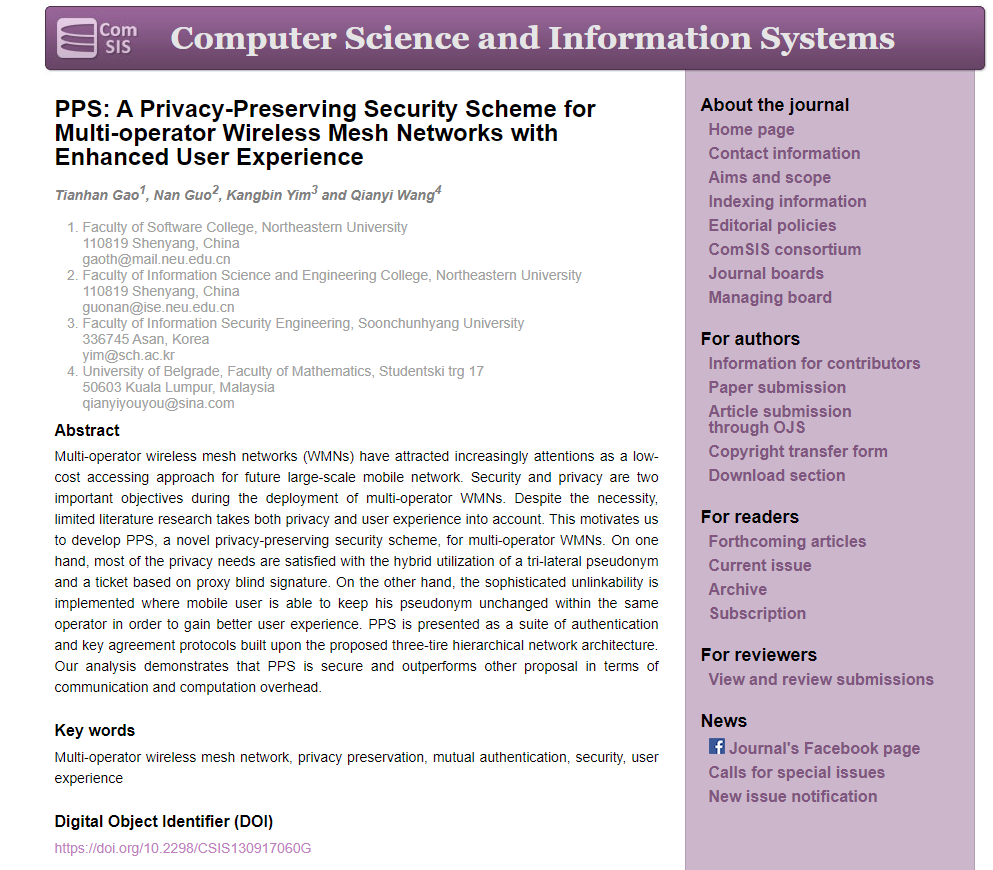
## Dizajn i implementacija

Kao što je pomenuto u prethodnom poglavlju, sistemi pretraživanja informacija mogu biti opšti ili usmereni na određenu oblast. Kako bi se dokazao koncept ovakvog pretraživanja, u okviru ovog rada implementirana je web aplikacija za pretraživanje naučnih radova, objavljenih u časopisima otvorenog pristupa. [PRIMER RANIJIH RADOVA NA OVU TEMU] Tokom ovog poglavlja biće prikazana i objašnjena high-level arhitektura ovog sistema, kao i arhitektura ključnih komponenti unutar sistema.

Ovaj sistem kreiran je iz dve glavne komponente: servera, koji je implementiran kao aplikacioni programski interfejs (API) i klijentske web aplikacije koja konzumira serverski API. Za potrebe implementacije serverske strane, korišćeno je Java razvojno okruženje Spring Boot. Glavni razlog korišćenja ovog razvojnog okruženja je njegova jednostavnost, kao i činjenica da je baziran na Java programskom jeziku, što omogućava korišćenje velikog broja open source biblioteka zasnovanih na tom programskom jeziku. Prilikom implementacije klijentske aplikacije korišćen je Angular 8, TypeScript razvojno okruženje. Odabir ove tehnologije je veoma prirodan ako se u obzir uzme da su web aplikacije današnjeg doba izrazito okrenute modernim razvojnim okruženjima za kreiranje klijetskih web aplikacija, kao što su Angular, React, VueJS i slični. Angular je zbog svoje jednostavnosti i veoma kvalitetno napisane dokumentacije veoma često prvi izbor prilikom kreiranja modernih single page aplikacija (SPA).

### Podaci

Za potrebe ovog istraživanja korišćeni su podaci o časopisima prikupljeni iz arhive časopisa *„Computer Science and Information Systems“ (ComSIS)* [Ref]. ComSIS je naučni časopis otvorenog tipa koji objavljuje originalne naučne radove kako teorijskog, tako i komercijalnog odnosno industrijskog tipa, usmerenih primarno na računarske nauke. Kako ovaj ovaj časopis ne nudi otvoren pristup skladištu podataka u vidu aplikacionog programskog interfejsa (API), jedini način pristupa podacima o naučnim radovima objavljenim u ovom časopisu jeste korišćenje web crawler komponente, koja je opisana kasnije u toku ovog poglavlja. Arhiva ovog naučnog časopisa predstavljena je kao jedna web stranica, koja u svom sadržaju poseduje linkove ka stranicama brojeva časopisa. Stranice brojeva časopisa u svom sadržaju poseduju linkove ka svim naučnim radovima objavljenim u tom broju časopisa. Sama web stranica naučnog rada prikazana je na slici (2.1.1). Kao što je moguće videti na slici web stranica na kojoj je predstavljen naučni rad, sadrži potake o naslovu rada, autorima, apstraktu kao i ključnim rečima u radu. Pored ovoga, stranica sadrži podatke i o godini u kojoj je rad objavljen, kao i link za preuzimanje kompletnog rada u vidu PDF dokumenta. Svi ovi podaci korišćeni su u fazi indeksiranja podataka o naučnim radovima.

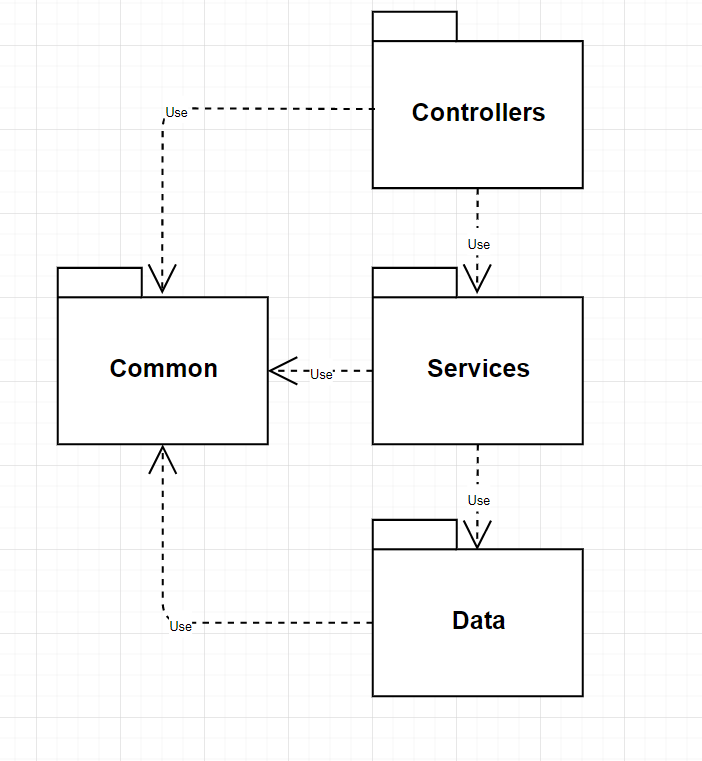


Slika 2.1.1. Primer izgleda stranice naučnog rada u ComSIS časopisu

### Arhitektura

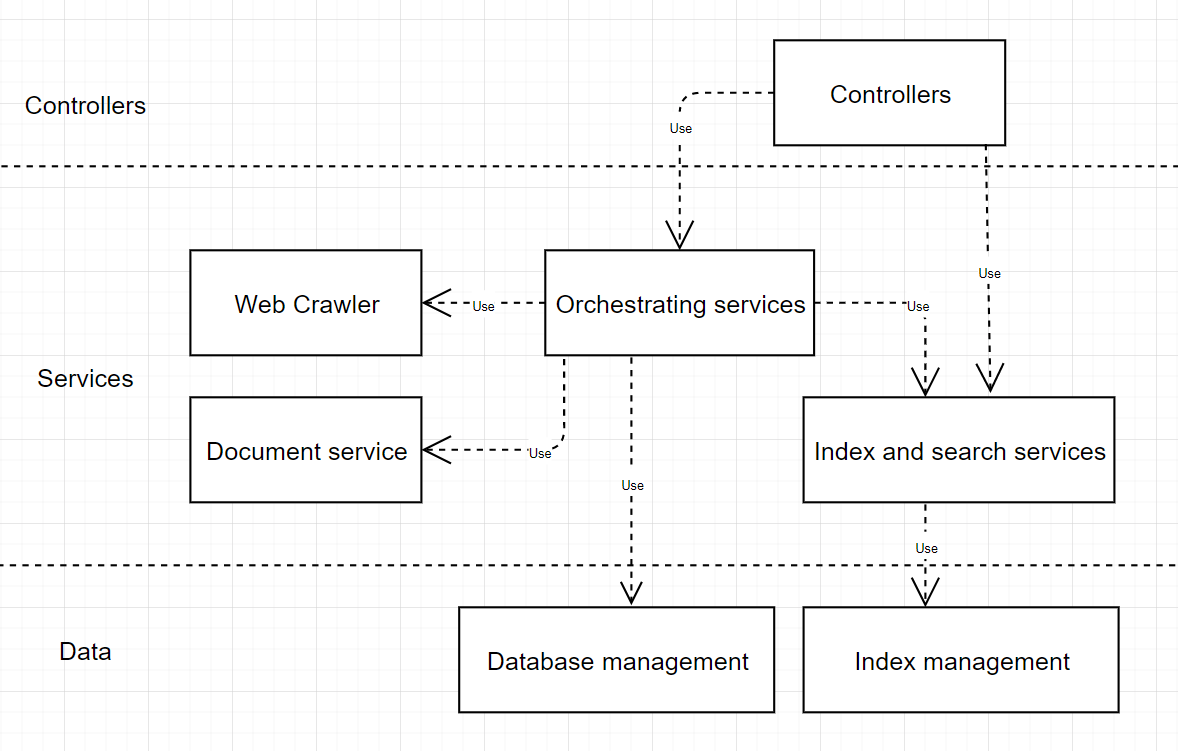
Kao što je pomenuto u ranijem delu ovog poglavlja, aplikacija kreirana u sklopu ovog rada se može podeliti na dve odvojene celine: klijentsku i serversku. Arhitektura klijentskog dela aplikacije je prilično trivijalna, ako se u obzir uzme činjenica da se kompletan rad aplikacije zasniva na serverskom delu, te da klijentska aplikacija samo šalje zahteve ka serveru i prikazuje dobijene rezultate. S ozbirom na to, arhitektura klijentske aplikacije neće biti detaljno opisana u sklopu ovog rada.

Serverski deo ovog sistema pretrage, kreiran je kao aplikacioni programski interfejs (API). Arhitekturu ove komponente moguće je jasno razdeliti na tri subkomponente: kontroler, servis i komponenta za rad sa podacima (Slika 2.2.1).



Slika 2.2.1. High-level arhitektura web api komponente

Svaka od pomenutih komponenti ima određen skup odgovornosti. Takođe kao što je moguće videti na slici, svaka od komponenti oslanja se i na komponentu „*Common*“ koja sadrži infrastrukturalne klase koje olakšavaju pisanje i održavanje koda. Komponente sa slike moguće je podeliti i na manje celine (Slika 2.2.2).



Slika 2.2.2. Slojevi unutar serverskog dela aplikacije

Na slici 2.2.2. moguće je uočiti podelu kompletnog servera na tri sloja: sloj kontrolera, sloj servisa i sloj podataka. Sloj kontrolera predstavlja ulaznu tačku web api komponente, te kao jedinu odgovornost ima prosleđivanje korisničkog poziva određenom servisu i obezbeđivanje povratne informacije korisniku. S obzirom da ni jedan od kontrolera iz ove komponente ne sadrži nikakvu biznis logiku, niti bilo kakve delove koda ključne za ovaj rad, klase unutar ove komponente neće biti detaljno opisani u sklopu ovog rada.

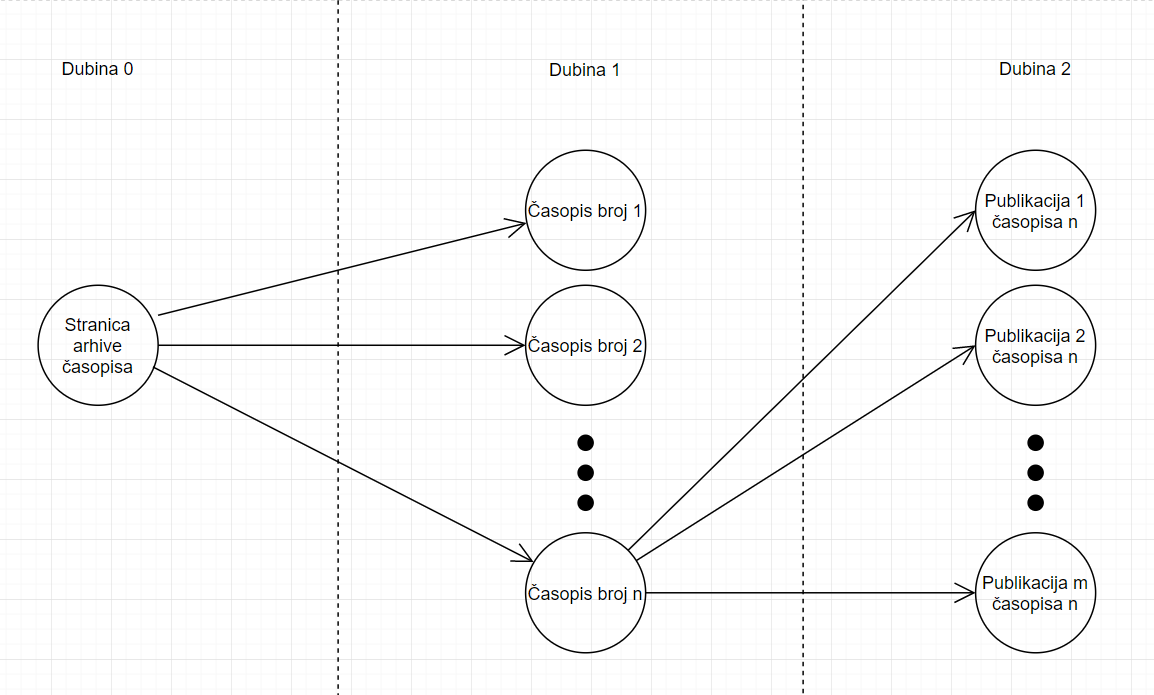
Sloj servisa je zasigurno najsloženiji sloj serverskog dela aplikacije. Komponenta servisnog nivoa koja poseduje najveći broj odgovornosti je svakako orkestrator. Orkestrator se sastoji od dva servisa: *PublicationService* i *AuthorService,* koji su zaduženi za prikupljanje i čuvanje podataka o publikacijama i autorima. Pri inicijalnom pokretanju aplikacije nad metodom *loadPublications* objekta klase *PublicationService.* Ovaj metod pokreće niz operacija čijim se izvršavanjem podaci o publikacijama i njihovim autorima preuzimaju i čuvaju u PostgreSQL bazi podataka. Celokupan proces prikupljanja podataka i njihovog čuvanja u bazu podataka opisan je nešto kasnije tokom ovog poglavlja. Pored metoda za učitavanje i čuvanje podataka, *PublicationService* sadrži i metod *indexPublications* koji za cilj ima preuzimanje prikupljenih podataka o publikacijama iz baze podataka, njihovo dopunjavanje i smeštanje u indeks fajlove. Druga značajna klasa orkestratora je *AuthorService* koja sadrži metod za smeštanje podataka o autoru u bazu, i metod za indeksiranje podataka o autorima. Prvi od ova dva metoda poziva se od strane klase *PublicationService* prilikom prikupljanja i čuvanja podataka u bazi.

Pored orkestratora, servisni nivo sadrži još i web crawler komponentu, koja je zadužena za prikupljanje podataka sa web stranica, kao i *DocumentService* koji je zadužen za preuzimanje PDF dokumenata o publikacijama sa web stranica i ekstraktovanje tekstualnog sadržaja iz njih, kako bi se omogućilo indeksiranje kompletnih naučnih radova. Poslednja komponenta servisnog sloja jesu servisi za indeksiranje i pretragu podataka o naučnim radovima i njihovim autorima. Svaka od pomenute tri komponente, biće detaljnije opisana kasnije u toku ovog poglavlja.

Kao što je navedeno u teorijskom uvodu ovog rada, u cilju kreiranja sistema za pretraživanje informacija potrebno je obezbediti načine za prikupljanje podataka, skladištenje podataka kao i njihovo pretraživanje i rangiranje. U nastavku ovog poglavlja biće opisani načini za prevazilaženje izazova u svakom od ovih koraka.

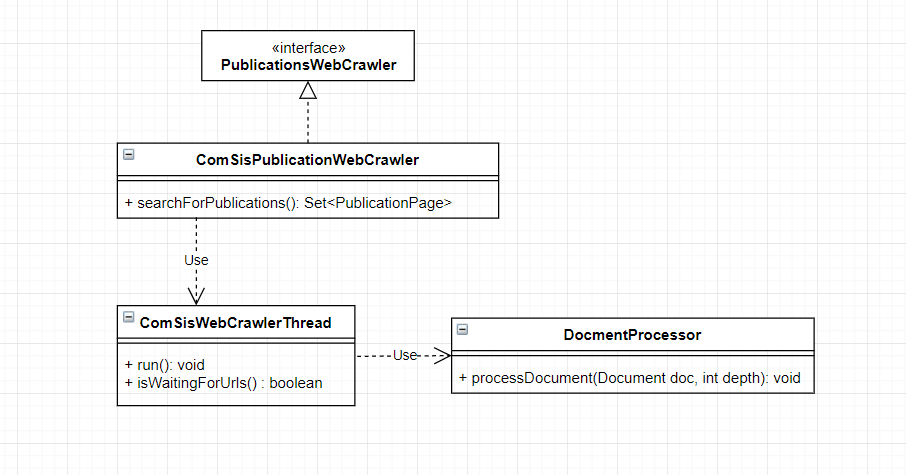
##### Prikupljanje podataka

Početna tačka ovog istraživanja jeste prikupljanje podataka o naučnim radovima. Za potrebe ovog rada podaci su prikupljani iz arhive časopisa Computer Science and Information Systems (ComSIS) [Ref], te je u cilju njihovog prikupljanja korišćena web crawler komponenta. Kako bi ovu komponentu bilo moguće implementirati najpre je potrebno utvrditi samu strukturu web sajta, kao i utvrditi skup početnih stranica sa kojih se pretraga može izvršavati. Nakon kraće analize ComSIS web sajta, moguće je uvideti jasnu i pravilnu strukturu linkovanja prikazanu na slici 2.2.3.



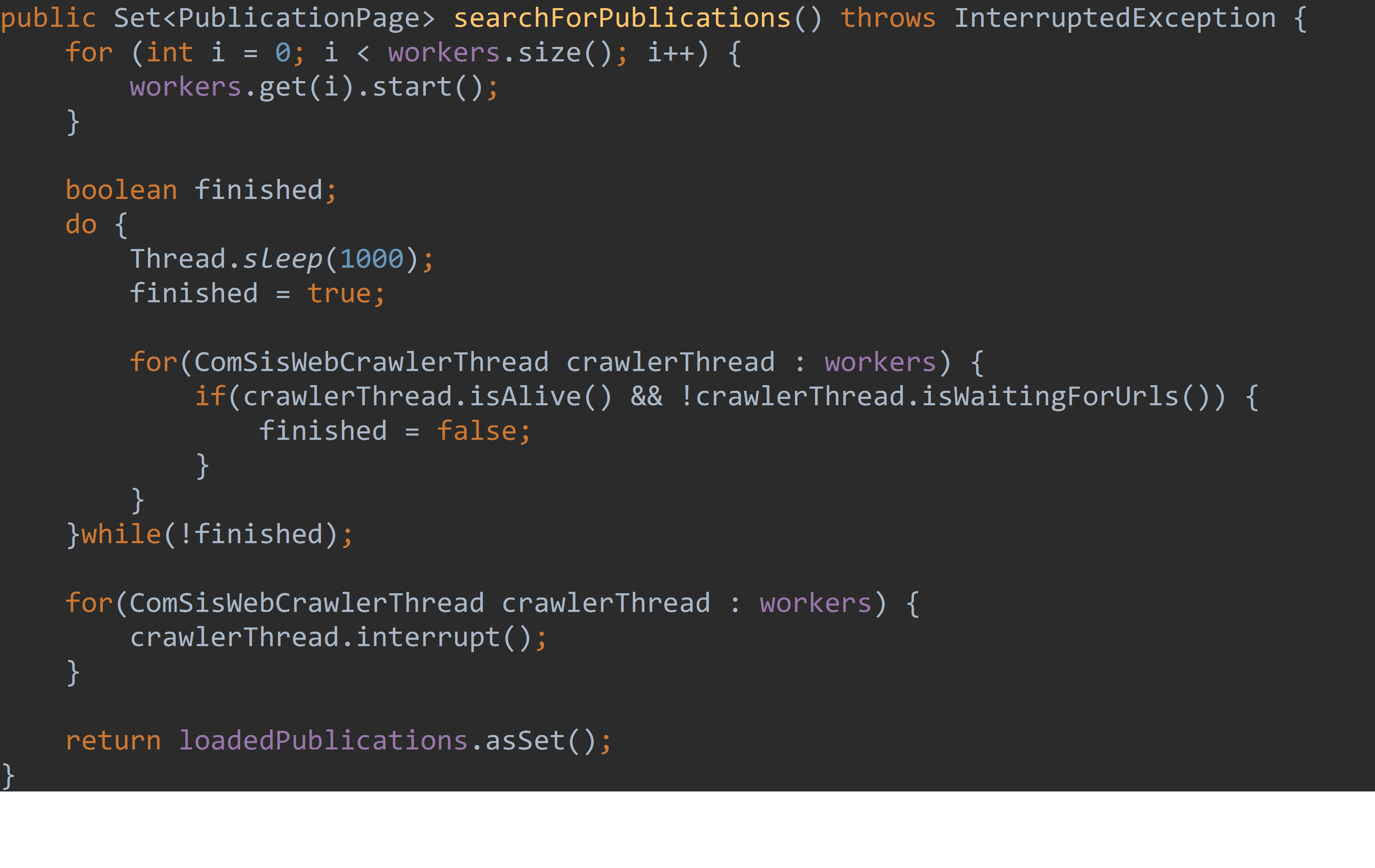
Slika 2.2.3. Struktura web sajta časopisa ComSIS

Nakon osvrta na strukturu sajta nameće se jasan zaključak da je najbolja početna tačka za komponentu pretraživanja stranica arhive časopisa, jer je sa te stranice putem linkova moguće pristupiti svim ostalim stranicama od interesa.



Slika 2.2.4. Struktura web crawler komponente

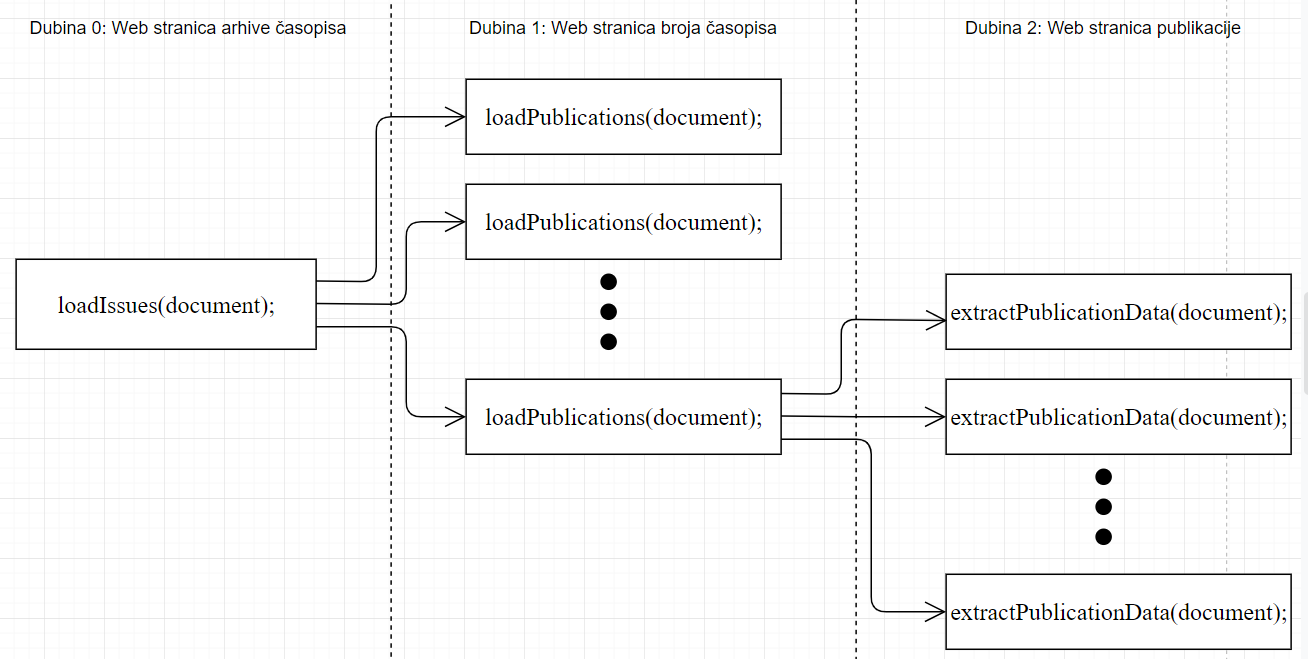
Sama struktura web crawler komponente prikazana je na slici 2.2.4. Kao što je moguće videti na dijagramu kalasa sa slike, konstruisan je interfejs *PublicationsWebCrawler* namenjen za korišćenje od strane eksternih komponenti. Klasa *ComSisPublicationWebCrawler* predstavlja implementaciju ovog interfejsa, i kako je propisano interfejsom sadrži samo jedan metod *searchForPublications* koja za cilj ima inicijalizaciju procesa za pretragu i njihovo upravljanje. Prilikom vršenja ovog istraživanja klasa *ComSisPublicationWebCrawler* konfigurisana je da inicijalizuje deset procesa pretraživanja koji se paralelno izvršavaju. Prilikom inicijalizacije ova klasa kreira instance skupa posećenih stranica, skupa parsiranih publikacija kao i red opsluživanja stranica koje tek trebaju biti posećene. Ova tri polja su veoma važna zbog toga što se prosleđuju svim procesima pretraživanja koji ih potom koriste za skladištenje izvučenih podataka i određivanje završetka pretrage. S obzirom na činjenicu da su ova tri polja deljena na nivou više procesa, veoma je važno da strukture podataka koje se koriste budu sinhronizovane, odnosno da se onemogući pristup strukturi od strane više procesa istovremeno. Klasa *ComSisPublicationWebCrawler* startuje deset instanci klase *ComSisWebCrawlerThread* i pokreće ih svake sekunde. Razlog zbog koga se nakon svake iteracije proces zaustavlja na jednu sekundu je zbog usporavanja slanja http zahteva. Ovaj korak je veoma važan jer u slučaju kreiranja prevelikog broja zahteva u kratkom vremenskom periodu, često dolazi do preopterećenja servera. Pored toga, mnoge web aplikacije poseduju zaštitu od ovakvog tipa „napada“, što rezultuje blokiranjem zahteva sa određene IP adrese. Takođe nakon svake od iteracija potrebno je proveriti da li je proces pretrage završen. Ovaj podatak se računa tako što program iterira kroz sve instance klase *ComSisWebCrawlerThread* i proverava da li je pretraga za njih završena. Ukoliko su svi procesi pretrage završeni, skup prikupljenih modela vraća se korisniku. Implementacija ovog procesa prikazana je na listingu 2.2.5.



Listing 2.2.5. Upravljanje web crawler procesima

Centralna klasa web crawler komponente jeste *ComSisWebCrawlerThread.* Ova klasa nasleđuje je naslednica klase *java.lang.Thread*, što omogućava objektima ove klase da izvršavaju kod paralelno sa drugim procesima istog ili drugog tipa. Ovakav pristup je veoma pogodan kada je u pitanju pretraživanje web sajtova koji često mogu biti veoma obimni, stoga se korišćenjem paralelne obrade stranica značajno skraćuje vreme pretraživanja sajta. Klasa *ComSisWebCrawlerThread*  sadrži *run* metod koji pokreće proces učitavanja web stranica i njhovo analiziranje i parsiranje. Prilikom poziva ovog metoda proces ulazi u beskonačnu petlju i na taj način omogućava da se proces izvršava dokle god ne bude prekinut od strane klase koja ga je pokrenula. U svakoj iteraciji ove petlje objekat najpre proverava red opsluživanja koji sadrži objekte klase *WebPageData*. Svaki od ovih objekata sadrži link ka stranici koja do sada nije posećena, kao i nivo na kojoj se ta stranica nalazi. Ukoliko ovaj red nije prazan sa njega će biti preuzet prvi objekat klase *WebPageData*, i na osnovu njegovog linka biće upućen zahtev za preuzimanje html sadržaja web stranice. Za potrebe ovog istraživanja preuzimanje html sadržaja vršeno je pomoću javine biblioteke Jsoup [RAZLOG]. Glavna klasa ove biblioteke takođe nosi naziv *Jsoup,* i ona omogućava preuzimanje html sadržaja u formi objekta clase *Document*, koja je takođe deo ove biblioteke. Poslednji korak svake iteracije jeste procesiranje dokumenta, koje izvršava klasa *DocumentProcessor* na osnovu prosleđenog dokumenta kao i nivoa na kome se nalazi stranica koja je učitana. Kako bi se omogućilo prekidanje procesa nakon završetka pretrage, *ComSisWebCrawlerThread* sadrži metod isWaitingForUrls, koji vraća vrednost true ukoliko proces pokušava da dobavi sledeći objekat klase *WebPageData* iz reda opsluživanja.

*DocumentProcessor* je klasa zadužena za procesiranje html dokumenata. Ova klasa sadrži statički metod *processDocument* koji kao ulazne parametre prima html dokument, kao i nivo na kome se taj dokument nalazi. Na osnovu prosleđenog nivoa procesor odlučuje o načinu parsiranja dokumenta. Ilustracija ovog procesa odlučivanja prikazana je na slici 2.2.6.



Slika 2.2.6. Procesiranje jsoup Document objekta na osnovu nivoa web stranice

Kao što je moguće videti na slici klasa *DocumentProcessor* sadrži tri metoda za parsiranje html dokumenta. Metod *loadIssues* koristi se prilikom obrade web stranice arhive časopisa. Ovaj metod sa html dokumenata prikuplja sve linkove koji odgovaraju formatu linkova ka brojevima časopisa, i potom na osnovu njih kreira nove instance klase *WebPageData* koje se potom smeštaju u red linkova koji još nisu posećeni. Metod *loadPublications* poziva se kada je dubina na kojoj se stranica nalazi jednaka broju jedan. Ovaj metod se izvršava skoro potpuno isto kao i prethodni. Prilikom pozivanja ovog metoda, sa stranice se parsiraju svi linkovi koji odgovaraju formatu linkova ka stranicama sa naučnim radovima, i na osnovu njih se kao i u prošlom slučaju kreiraju novi objekti tipa *WebPageData* koji se smeštaju u red opsluživanja. Kada je dubina stranice jednaka 2, unutar klase *DocumentProcessor* poziva se metod *extractPublicationData.* Prilikom poziva ovog metoda prvi korak koji se izvršava jeste pronalaženje vrednosti unutar html elemenata na osnovu unapred dodeljenih selektora, nastalim analizom html sadržaja stranica na kojima se nalaze publikacije. Sledeći korak koji se izvršava u okviru ovog metoda jeste kreiranje objekta klase *PublicationPage* na osnovu prikupljenih atributa, nakon čega se kreirani objekat smešta u kolekciju prikupljenih modela deljenu na nivou svih *ComSisWebCrawlerThread*  procesa.

Nakon završetka pretraživanja web stranica svi web crawler procesi bivaju prekinuti od strane objekta klase *ComSisPublicationWebCrawler.* Poslednji korak glavnog procesa jeste vraćanje kolekcije sa objektima klase *PublicationPage,* popunjene modelima prikupljenih sa web stranica. Klasa *PublicationPage* sadrži neprilagođene podatke o naučnom radu, koji se moraju prilagoditi ili dopuniti kako bi bili pogodni za kasnije čuvanje i indeksiranje.

Nakon što klasa *PublicationService* od web crawler komponente dobije skup objekata *PublicationPage*, svaki se dodatno obrađuje i pretvara u instancu klase *PublicationData.* Glavni razlog za ovu konverziju jeste činjenica da se podaci sakupljeni prilikom procesa pretrage ne nalaze u formatu koji je pogodan za skladištenje. Najbolji primer ovoga jeste sekcija autora kao i sekcija sa afilijacijama i email adresama autora. Kao što je moguće videti na slici 2.2.7., spisak autora je odvojen od sekcije sa njihovim afilijacijama, te je prostom selekcijom html elemenata stranice nemoguće povezati autore sa odovarajućom afilijacijom.



Slika 2.2.7. Primer spiska autora naučnog rada i njihovih afilijacija

Kako bi se ovaj problem prevazišao korišćena je klasa *AuthorsPageDataParser* koja najpre otklanja ostatke html tagova iz samih vrednosti nakon čega na osnovu spiska autora i njihovih referenci, za svakog autora kreira servisni model koji sadrži podatke o imenu autora, njegovoj email adresi i njegovoj afilijaciji. Drugi proces dopunjavanja podataka jeste konstruisanje url-a za preuzimanje dokumenta kombinovanjem adrese sajta dobijene iz konfiguracije i relativne putanje za preuzimanje dobijene sa stranici prilikom prikupljanja podataka.

Nakon što su svi podaci uspešno prikupljeni i parsirani sledi njihovo čuvanje u bazu podataka. Prilikom implementacije ovog projekta korišćena je PosgreSQL baza podataka. Glavni razlog korišćenja baze podataka umesto direktnog indeksiranja jeste mogućnost dobavljanja podataka iz više različitih izvora pre samog procesa indeksiranja, kao i skladištenje podataka u slučaju potrebe za ponovnim indeksiranjem podataka.

##### Skladištenje podataka

Drugi izazov pri implementaciji sistema za pretraživanje jeste obezbeđivanje adekvatnog načina za skladištenje podataka. Kao što je objašnjeno u teorijskom uvodu ovog rada, najbolji pristup rešavanju ovog problema jeste korišćenje indeks fajlova. Ovaj pristup omogućava brzu, rangiranu pretragu zasnovanu na korisničkom upitu, kojeg je moguće pisati u prirodnom jeziku. Servisni sloj sistema za pretraživanje informacija, kreiranog za potrebe ovog rada sadrži komponentu za indeksiranje i pretragu indeks fajlova. Za potrebe indeksiranja podataka, ovaj deo sistema sadrži dve klase *PublicationIndexService* i *AuthorIndexService,* zadužene za indeskiranje podataka o naučnim radovima i autorima respektivno. Ovi servisi se pokreću od strane orkestratora, koji im inicijalno prosleđuje sve prikupljene podatke iz baze podataka. Važno je napomenuti da ovi servisi ne obavljaju samo indeksiranje, već validiraju ulazne podatke, kreiraju indeks modele i prosleđuju rezultujuću kolekciju modela komponenti za rad sa indeksima koja se nalazi na sloju podataka. Ova komponenta sadrži dve glavne klase koje za odgovornost imaju kreiranje indeks fajlova: *PublicationIndexer* i *AuthorIndexer.* Kako je proces indeksiranja publikacija i njihovih autora gotovo isti, u toku toku ovog poglavlja detaljnije će biti opisan samo proces indeksiranja naučnih radova.

Proces indeksiranja podataka započinje nakon što metod *indexPublications* bude pozvan od strane orkestratora. Ovaj metod kao ulazni parametar prima kolekciju naučnih radova prikupljenih iz baze podataka. Na svakom od ulaznih naučnih radova vrši se validacija u sklopu koje se proverava da li nedostaje neki od podataka o naučnom radu, te se u slučaju da neki od podataka nedosaje posmatrana publikacija izbacuje iz kolekcije, kako bi se u indeks fajlovima nalazili samo naučni radovi sa svim popunjenim podacima. Nakon što se validacija uspešno izvrši, na osnovu validnih publikacija kreiraju se indeks modeli. Ovi modeli podakte o publikaciji predstavljaju na način koji je pogodniji za kreiranje dokumenata koji se smeštaju u indeks fajlove. Nakon što su modeli uspešno kreirani i prikupljeni u listu, aplikacija kreira novu instancu klase *PublicationIndexer* nad kojom se potom poziva metod *indexPublications* kome se kao ulazni parametar prosleđuje lista prikupljenih indeks modela.

Klasa *PublicationIndexer* predstavlja centralnu klasu kada je u pitanju skladištenje podataka o naučnim radovima u indeks fajlove. Kako bi se sam rad sa indeks fajlovima omogućio, za potrebe ovog rada korišćena je Lucene biblioteka za pretragu. Lucene je open-source biblioteka za pretraživanje informacija, koja omogućava lako i brzo dodavanje funkcionalnosti pretraživanja podataka u sve vrste aplikacija. Moderno vreme i velika količina podataka doneli su brojne biblioteke ovog tipa, kao što su Elasticsearch, Google Enterprise Search, Coveo i drugi. Glavni razlog za odabir Lucene biblioteke jeste veoma dobra dokumentacija kao i veliki broj naučnih radova i knjiga sa uputstvima za korišćenje ove biblioteke. Funckionalnosti koje Lucene pruža pri čuvanju podataka jeste kreiranje indeks fajlova na zadatoj putanji, izvršavanje analize ulaznog texta i njihovo smeštanje u kreirane indeks fajlove. Glavna klasa biblioteke Lucene za indeksiranje jeste *IndexWriter.* Konstruktor ove klase kao parametre prihvata objekat klase *Directory* koja predstavlja direktorijum u koji će indeks fajlovi biti smešteni. Pored ovoga, konstruktor klase *IndexWrite* kao parametar prima i konfiguracioni objekat koji određuje način analiziranja teksta, polisu brisanja dokumenata i slično. Nakon što je instanca klase *IndexWriter* uspešno kreirana, njoj se dodeljuju dokumenti koji se potom smeštaju u kreirane indeks fajlove.

Nakon što metod *indexPublications* bude pozvan sa ulaznom kolekcijom indeks modela za svaki od modela se kreira posebna instanca klase *org.apache.lucene.document.Document* koja se potom popunjava vrednostima iz indeks modela publikacije. Ovaj proces prikazan je na listingu 2.2.8.



Slika 2.2.8. Priprema dokumenta za indeksiranje

Metod *populateDocumentWithPublicationData* kao parametre prihvata indeks model publikacije i objekat klase *Document* koji predstavlja dokument koji će biti smešten u indeks fajl. Dokumentu se dodeljuju vrednosti pozivom metoda *add* nad instacom dokumenta, koja kao ulazne parametre prihvata naziv polja koje će dokument sadržati, vrednost tog polja, kao i enum koji označava da li se vrednost polja takođe čuva u indeksu. Kako je moguće videti na listingu 2.2.8. polja koja se čuvaju u indeks fajlovima su naslov, id, apstrakt, godina objavljivanja, ključne reči, tekst publikacije kao i njeni autori. Sva ova polja u kasnijoj fazi ovog rada biće dostupna za pretragu. Moguće je uočiti da se prilikom dodavanja tekstualnog sadržaja publikacije koristi *DocumentService* kako bi se dobavio sadržaj. Ovaj pristup odabran je umesto skladištenja tekstualnog sadržaja unutar indeks modela jeste činjenica da bi zbog velikog broja ulaznih modela došlo do nedostatka memorije za čuvanje kompletnog tekstualnog sadržaja. Takođe, na listingu je moguće videti i da je spisak autora publikacije serijalizovan, što omogućava pretragu naučnih radova po imenima autora, kao i po njihovim email adresama, afilijacijama i drugo.



Slika 2.2.9. Indeksiranje podataka o naučnim radovima

Nakon što se dokument popuni od strane ovog metoda, on biva dodat u instancu klase *IndexWriter* koja ga potom upisuje u kreirani indeks fajl. Ovaj proces se završava kada se dokumenti kreiraju za sve primljene modele i nakon što budu smešteni u indeks fajlove. Celokupan process indeksiranja podataka prikazan je na listingu 2.2.9. Nakon što se proces indeksiranja uspešno završi, veoma je važno zatvoriti instancu klase *IndexWriter* kako bi se oslobodili indeks fajlovi i na taj način se omogućilo kasnije čitanje iz njih. Nakon što se metod *indexPublications* uspešno izvrši indeks fajlovi za pretragu naučnih radova su sačuvani i spremni za pretragu.

##### Pretraživanje i rangiranje podataka

Sledeći korak u implementaciji ove aplikacije jeste kreiranje komponente za rangirano pretraživanje podataka. Kako bi se podaci o naučnim radovima i njihovim autorima uspešno pretražile, aplikacija pruža brojne metode kontrolera koji klijentu omogućavaju da uspešno dobavi podatke na osnovu zadatog upita. Svaki od metoda kontrolera odgovara tačno jednom polju pretrage, i svaki od njih prima dva ulazna parametra, od kojih prvi predstavlja sam upit u obliku teksta, dok trugi predstavlja maksimalni broj rezultata pretrage. Drugi parametar ovih metoda je opcioni te će se ukoliko klijent ne pruži ovaj podatak iskoristiti podrazumevana vrednost uzeta iz konfiguracije. Za potrebe pretrage naučnih radova koriste se metodi klase *PublicationSearchController,* dok se za potrebe pretrage autora koristi *AuthorSearchController.* Ove klase kao povratnu vrednost imaju kolekciju publikacija odnosno autora poređanu prema nivou sličnosti sa zadatim upitom. Kako je proces pretrage za naučne radove i autore gotovo isti, u toku ovog poglavlja detaljno će biti opisan samo implementacija procesa pretrage naučnih radova na osnovu korisničkog upita.

Nakon što se neki od metoda objekta klase *PublicationSearchController* pozove, on delegira poziv ka instanci klase *PublicationSearchService* koja sadrži metode za pripremu upita za pretragu, kao i mapiranje dobijenih rezultata na odgovarajuće modele koji će biti čitljivi klijentu. Nakon što se uputi poziv metodu klase *PublicationSearchService,* interno se poziva metod *searchPublications,* kome se kao parametri prosleđuju upit, maksimalni broj rezultata pretrage i polje po kome pretraga treba da se vrši. Ovaj metod potom delegira poziv instanci klase *PublicationIndexSarcher,* te vraća dobijene rezultate pretrage.

*PublicationIndexSarcher* i *AuthorIndexSarcher* su dve centralne klase za pretraživanje podataka iz indeks fajlova. Ove klase se nalaze u sloju podataka, i pretražuju indeks fajlove korišćenjem Lucene biblioteke. Prilikom inicijalizacije ove klase, u njenom konstruktoru se kreira instaca klase *IndexSearcher* na osnovu putanje do direktorijuma u kome se nalaze indeks fajlovi. Ovaj objekat kasnije se koristi za pretraživanje podata iz indeks fajlova. Glavni metod klase *PublicationIndexSarcher* jeste metod *search* koji kao ulazne parametre prihvata tekstualni upit, naziv polja koje se pretražuje, kao i maksimalni broj vraćenih rezultata.



Listing 2.2.9. Pretraga dokumenata iz indeks fajlova

Kompletan proces pretrage podataka iz indeks fajlova moguće je videti na listingu 2.2.9. Iz priloženog listinga može se videti da je prvi korak u pretraživanju kreiranje objekta klase *Query*. Ova klasa je deo Lucene biblioteke i kreira se korišćenjem instanceklase *QueryParser,* čijem se konstruktoru prosleđuje naziv polja za pretragu, kao i objekat koji označana način analiziranja teksta, te se nad ovom instancom poziva metod *parse* kome se prosleđuje tekstualni upit dobijen od strane klijenta. Nakon što se objekat klase *Query* uspešno kreira, on se zajedno sa maksimalnim brojem željenih rezultata prosleđuje metodu *search* klase *IndexSearcher*. Povratna vrednost ove pretrage je niz objekata klase *ScoreDoc* koja sadrži dokument koji je pronađen prilikom pretrage, kao i njegovu vrednost skora. Lucene biblioteka podrazumevano vraća niz ovih objekata prema sortiran prema vrednosti skora. Maksimalan broj elemenata ovog niza odgovara broju koji je prosleđen prilikom kreiranja *Query* objekta. Nakon obavljene pretrage i dobijanja rezultata, svaki od dobijenih elemenata parsira se u indeks model. Kao što je moguće videti na listingu 2.2.10 algoritam za parsiranje indeks modela na osnovu dokumenta je inverzan algoritmu za kreiranje dokumenta na osnovu indeks modela. Kao što je moguće videti, prilikom parsiranja liste autora poziva se metod getAutorsList, unutar koga se u cilju dobijanja kompletne liste autora publikacije vrši deserijalizacija teksta sačuvanog prilikom indeksiranja podataka. Na ovaj način se ostvaruje to da se podaci ne izgube prilikom pretrage, nego se vraćaju u istom formatu u kome su bili i pre samog čuvanja u indeks fajlove.



Listing 2.2.10. Formiranje indeks modela na osnovu pronađenog dokumenta

Nakon što su modeli uspešno kreirani usleđuje formiranje liste dobijenih modela i vraćanje rezultujuće liste. Kao što je to bio slučaj i pri procesu indeksiranja, veoma je važno da se na kraju procesa pretraživanja zatvori instanca klase za pretragu indeks fajlova, kako bi se fajlovi oslobodili za kasniju ponovnu pretragu. Krajnji korak ove pretrage jeste mapiranje servisnih modela publikacija u modele koji će biti čitljivi za klijenta, njihova serijalizacija i vraćanje http odgovora koji u svom sadržaju poseduje niz rezultujućih publikacija u json formatu.