Integracija sustava pretraživanja i Adobe Experience Manager-a

Malić, Tadija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tadija Malić

INTEGRACIJA SUSTAVA PRETRAŽIVANJA I ADOBE EXPERIENCE MANAGERA

DIPLOMSKI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tadija Malić

Matični broj: 45287/16–R

Studij: Informacijsko i programsko inženjerstvo

INTEGRACIJA SUSTAVA PRETRAŽIVANJA I ADOBE EXPERIENCE MANAGERA

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2019.

Tadija Malić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada korištene su etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj rad bavi se programskim sustavom *Adobe Experience Manager*a s posebnim osvrtom na arhitekturu i skup tehnologije koje *AEM* koristi. Za potrebe praktičnog dijela rada pobliže će biti objašnjena svaka pojedina tehnologija. Bit će prikazani detalji različitih sustava pretraživanja kao što su *Apache Lucene*, *Apache Solr* i *Elasticsearch*. Pored toga autor će se posvetiti načinima implementacije navedenih sustava s *Adobe Experience Manager*om na primjeru aplikacije za *online* prodaju – *We Retail*. U praktičnom dijelu bit će testirani razni slučajevi korištenja takvih sustava pretraživanja kao i njihova međusobna usporedba. Svi zaključci i ideje za buduće nadogradnje bit će opisane u zaključnim poglavljima rada.

Ključne riječi: *adobe experience manager, apache lucene, elasticsearch, apache solr*, sustav pretraživanja

Sadržaj

1. Uvod

Tehnologija je sastavni svakodnevnice. Dok još uvijek postoje pojedinci koji se na sve načine trude izbjeći dodir s tehnologijom, za mnoge je to neophodno. Tehnologija nije neophodna samo kako bi pojedinac ili organizacija opstala na konkurentskom tržištu, nego i za samo preživljavanje. Gotovo da ne postoji medicinska ustanova u modernom svijetu koja ne bazira svoje liječenje na rezultatima koje im generira ili prikazuje tehnologija. Osim toga, dokazane metode liječenja na koje se liječnici često pozivaju ostvarene su pomoću tehnologije.

Najbolji predstavnik tehnologije u današnjem svijetu je internet. Često se može čuti kako je internet sastavni dio života većine ljudi na kugli zemaljskoj. Prema istraživanju *Internet World Stats*a (u nastavku *IWS*) ova hipoteza je točna ali ne i u razdoblju do lipnja 2017. Tada se dogodilo prvo mjerenje *IWS*-a koje je pokazalo kako više od 50% svjetskog stanovništva koristi internet [1]. Posljednje mjerenje, dogodilo se u ožujku 2019. Tadašnje istraživanje je pokazalo kako se internetom koristi 56,8% stanovnika svijeta [2]. Potrebno je naglasiti kako su na ovakve rezultate istraživanja utjecale činjenice da na svijetu još uvijek postoje područja koja nemaju pitku vodu, da je velika razlika između bogatih i siromašnih, da postoji velika razlika u infrastrukturi i edukaciji u zemljama trećeg svijeta u odnosu na ostatak zemalja i sl., ali bez obzira na spomenute čimbenike utjecaja na istraživanje, 56,8% je i dalje vrlo značajan udio. Točnije, u vrijeme istraživanja se radilo o brojci od 4,383,810,342 ljudi [2].

Pored svega navedenog, svjedoci smo svakodnevnog napretka tehnologije koja se danas nalazi u strukturi gotovo svake organizacije. Kako je uznapredovao razvoj tehnologija u vrlo kratkom vremenu pokrenule su se i razvile mnoge stvari koje čovječanstvo nije očekivalo. Jedna od značajnijih promjena je važnost informacije. Informacija je danas značajnija nego što je bila prije par desetaka godina jer se pomoću današnjih resursa informacije mogu iskoristiti efikasnije nego što je to bio slučaj ranije. Informaciju tvori više podataka, a s većom količinom podataka ona postaje jasnija. Brojne su metode analize i iskoristivosti informacije, bilo to u pozitivne ili negativne svrhe za čovječanstvo.

Kako bi jasnoća informacije bila veća, potrebno je više podataka. Tako se danas organizacije trude prikupiti što više podataka iz više izvora (govor, napisana riječ, ponašanje i sl.). Količina podataka se iz dana u dan znatno povećava čemu uvelike pridonosi razvoj *Internet of Things*a. Prema *Forbesovim* podacima iz svibnja 2018. dnevno se generira oko 2,5 kvintilijuna bajtova podataka [3]. Tome pridonose i sljedeća mjerenja [4]:

- 500 milijuna *tweetova* se objavi u jednom danu
- 294 milijarde poslanih emailova dnevno
- 4 petabajta sadržaja se kreira na *Facebook*u dnevno
- 4 terabajta podataka se stvori od povezanih automobila dnevno
- 65 milijardi poruka dnevno se pošalje preko *WhatsApp*a
- 5 milijuna pretraga korisnika dnevno

Posve je očekivano da tolika količina podataka narušava performanse svake aplikacije pri pretraživanju podataka od strane korisnika te pri bilo kakvom drugom obliku manipulacije podacima. Korisnicima je iznimno važno vrijeme izvršavanja za koje imaju jako malu toleranciju. Vrijeme izvršavanja određenog zahtjeva na aplikaciji smatra se ključnim čimbenikom u ocjeni kvalitete rada aplikacije [5]. *Marissa Mayer* je 2006. godine objavila da pola sekunde čekanja na rezultate tražilice na *Google*u rezultira 20%-tnim smanjenjem prometa, iako tako korisnici dobiju više relevantnih informacija [6].

Kako bi se to spriječilo osmišljeni su sustavi za napredno pretraživanje. Tako danas sve veće organizacije koriste napredne sustave pretraživanja da bi svoje korisnike učinili zadovoljnima. Kao najpopularniji sustavi pretraživanja danas, nameću se *Apache Solr* i *Elasticsearch*. Oba navedena sustava su izgrađena povrh *Apache Lucene* tehnologije iako je nastala davne 2000.

Napredni sustavi pretraživanja se mogu implementirati u gotovo svim poznatim okvirima i programskim jezicima. U ovom radu bit će prikazana njihova implementacija u *Adobe Experience Manageru* (u daljnjem tekstu *AEM*).

Zašto baš *AEM*? Razloga je mnogo, a najvrjedniji bi bio porast u broju korisnika svake godine koji dovoljno govori o kvaliteti ovog sustava. Još jedan od pokazatelja relevantnosti ovog sustava u tehnološkom svijetu je podatak kako je *AEM* u vrhu *Gartnerove* liste tehnologija za upravljanje web sadržajem u posljednjih osam godina [7].

2. Metode i tehnike rada

U ovom radu korištene su različite metode i tehnike rada, kao i mnogi alati pomoću kojih su ostvareni rezultati iz zaključka rada. Istraživanje je rađeno prema primjeru aplikacije *We Retail* - referentnim primjerom aplikacije u *Adobe Experience Manageru*. Da bi rad nad spomenutom aplikacijom bio moguć, potrebno je instalirati instancu *AEM* poslužitelja iz direktorija u kojemu se nalazi licenca za korištenje *Adobe Experience Manager* tehnologije.

Razlozi zbog kojih je izabrana *We Retail* aplikacija su sljedeći:

- Realan sadržaj;
- Aplikacija *online* trgovine što odlično predstavlja aplikaciju u kojoj je pretraga osnovna funkcionalnost;
- Moderan sadržaj koji predstavlja sve današnje potrebe korisnika online trgovina;
- Rađena na najboljim praksama razvoja aplikacija u *AEM*-u;
- Lako proširiva i prilagodljiva;
- Velika količina sadržaja;
- Projekt otvorenog koda.

Za razradu teme korištena je dokumentacija svih tehnologija koje su opisane u radu kao i knjige napisane sa svrhom razumijevanja navedenih sustava pretraživanja. Osim toga osmišljeni su realni scenariji poslovnih slučajeva kojima ovo istraživanje može biti od velike koristi. Sve navedeno nalazi se u popisu literature.

Kako bi praktični dio istraživanja bio moguć, bio je potreban skup sljedećih alata:

- Integrirano razvojno okruženje (eng. *Integrated development environment – IDE*) za razvoj *Adobe Experience Manager* – korišteni *IntelliJ*, *Eclipse, Visual Studio Code, Typora;*
- Programski jezik *Java;*
- Sustav verzioniranja *Git* i verzija s grafičkim sučeljem – *GitHub;*
- *iTerm* konzola za rad sa *Solrom* i *Gitom;*
- *AEM* licenca i instanca poslužitelja;
- *Apache Lucene;*
- *Apache Solr;*
- *Elasticsearch* i pripadni alati;
- *JUnit, OSGi, Sling*, integrirani *Jetty* poslužitelj i ostale zavisnosti potrebne za pravilan razvoj, rad i testiranje funkcionalnosti *AEM*-a.

Kako bi odredili razliku između sustava pretraživanja usporedit će se sljedeći parametri:

- Osnovna tehnologija;
- Skalabilnost;
- Sustavi za povezivanje podataka i analitiku;
- Procesiranje sadržaja;
- Indeksiranje;
- Funkcionalnost upita;
- Relevantnost pretrage;
- Sigurnost;
- Korisničko sučelje;
- Administracija, nadzor i održavanje.

Za takvo istraživanje je pripremljen poseban skup podataka tekstualnih i numeričkih vrijednosti, parametara te postojećeg sadržaja i oznaka.

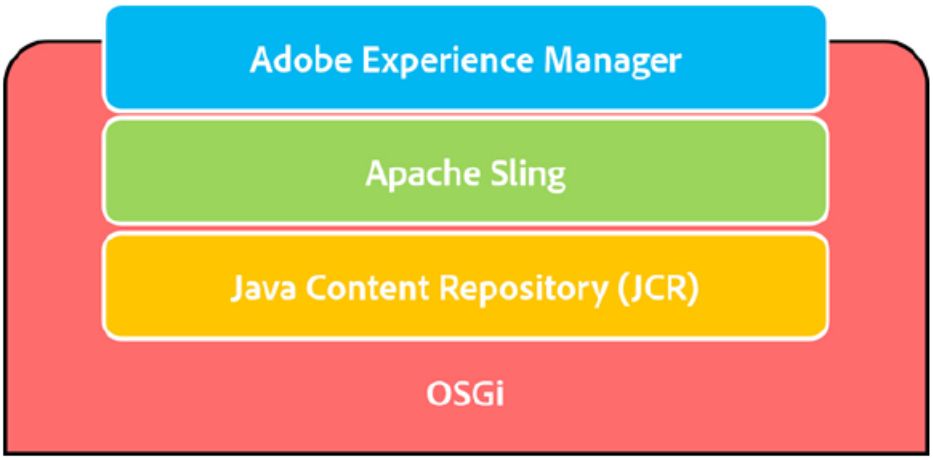
Istraživanje o brzini izvođenja upita izvodi se klasičnom matematičkom metodom – *početno vrijeme – završno vrijeme = vrijeme trajanja*. Prije praktične primjene autor je prikupio povijesne podatke iz izvora koji se nalaze u literaturi pa se radi i o povijesnoj metodi korištenoj za ovo istraživanje. Važnosti istraživanja pridonijeli su kvantitativni podaci kojima je mjerena brzina izvršavanja upita (zadano mjerenje *AEM*-a preko *Explain Query* alata). Vrijednosti su poredane po veličini, a apsolutna nula ne postoji.

3. Adobe Experience Manager

Adobe Experience Manager je sustav za upravljanje sadržajem koji se koristi za izradu web stranica, servisa sadržaja i obrazaca. Sustav je baziran na klijent – poslužitelj arhitekturi s više funkcija na razini infrastrukture i aplikacijskoj razini. Korištenjem funkcijskih blokova na infrastrukturnoj i aplikacijskoj razini izrađuju se prilagodljive aplikacije. Na infrastrukturnoj razini *AEM* sadrži:

- *Web* aplikacijski poslužitelj: *AEM* se može isporučiti kao samostalna aplikacija ili kao dio drugog aplikacijskog poslužitelja kao npr *WebLogic*-a ili *WebSphere*. Samostalna aplikacija podrazumijeva integrirani *Jetty* poslužitelj,
- Okvir *Web* aplikacije: *AEM* se sastoji od integriranog *Sling* okvira koji olakšava pisanje *REST* aplikacija i aplikacija orijentiranih na sadržaj,
- Repozitorij sadržaja: *AEM* se sastoji od *Java* sadržajnog repozitorija (eng. *Java Content Repository*), skraćeno *JCR*. To je hijerarhijska baza podataka koja je dizajnirana tako da omogućuje rad sa strukturiranim ali i polu-strukturiranim podacima.

AEM je *Java* web aplikacija koja se bazira na više različitih tehnologija. Tehnologije su prikazane u sljedećem dijagramu:



Slika 1 - *AEM* - skup tehnologija (Prema: Adobe Systems Incorporated, 2017.)

Apache Sling baziran na *REST* principima je tehnologija koja služi za upravljanje i spremanje sadržaja. *JCR* predstavlja bazu podataka, a *OSGi* (eng. *Open Service Gateway Initiative*) omogućava svakoj aplikaciji da bude izgrađena i pokrenuta kao *OSGi* paket (eng. *bundle*).

3.1. Povijest *Adobe Experience Managera*

Za potrebe ovog rada koristit će se najnovija stabilna verzija *AEM*-a – 6.4. Zbog boljeg uvida u značenje verzije 6.4, u nastavku će biti prikazane promjene na *AEM*-u kroz povijest.

Tablica 1 - Povijest verzija *AEM*-a

Naziv proizvoda	Vrijeme izlaska na tržište	Osnovne značajke
Day CQ 1.0	2000	1. Pokrenuta švicarska tvrtka *Day Software* 2. *CGI* Skripte za *Netscape Enterprise Poslužitelj*
Day CQ 2.0	2000	1. Samostalna aplikacija 2. Programski jezik C 3. *HTTP* Poslužitelj 4. Spremnik baziran na datotečnom sustavu
Day CQ 3.0	2000	1. *Communique* 2 napisan kao *Java Web* aplikacija 2. Odvojen *HTTP* poslužitelj 3. *Content Bus* spremnik, konektori za različite vrste datoteka, baza i sl.
Day CQ 3.5	2002	1. Riješena velika količina prijašnjih grešaka
Day CQ 4.0	2005	1. *Content Bus* spremnik preko *JCR*-a 2. Dvije *Web* aplikacije: *CRX* i *CQ*
Day CQ 4.1	2006	1. Riješena velika količina prijašnjih grešaka
Day CQ 4.2	2008	1. Riješena velika količina prijašnjih grešaka
Day CQ 5.0	2008	1. Kompletni kod je ponovo napisan
Day CQ 5.0	2009	1. Riješena velika količina prijašnjih grešaka
Day CQ 5.0	2010	1. Riješena velika količina prijašnjih grešaka
Adobe Day CQ 5.4	2011	1. *Adobe* preuzima *Day Software* i prva verzija *CQ*-a izlazi pod *Adobe*jem 2. Dodana mogućnost upravljanja *Web* stranicama preko mobilnih uređaja 3. Dodana mogućnost upravljanja *CQ Workflow* ulaznim spremnikom poruka preko *iPhone*/*iPad* uređaja 4. Dodana mogućnost autoriziranja preko *iPad*a

		5. Kampanje 6. Izvještaji 7. *CRX* Klasteri 8. Anotacije 9. Video komponenta 10. *Carousel* komponenta 11. Rad s radnim tijekovima 12. Forumi 13. Zajednička konzola 14. Profili
Adobe Day CQ 5.5	2012	1. Nova arhitektura bazirana na *OSGi*ju 2. Nove funkcionalnosti natrag/naprijed pri uređivanju sadržaja 3. Novi okvir za analitiku integriran kao *Adobe SiteCatalyst* 4. Nova integracija s *Adobe Search&Promote* platformom 5. Unaprijeđena integracija s *Adobe Test&Target* platformom 6. Unaprijeđen *Campain Manager UI* 7. Integracija s *Adobe Scene7* 8. *XMP* metapodaci za određene formate datoteka 9. *CQ* DAM *Proxy* integriran s *InDesign* poslužiteljom 10. *CQ DAM Proxy* i *ProxyWorkers* 11. Vanjski poslužitelj za promatranje i upravljanje preko *Java Management* Ekstenzije (*JMX*) 12. Dodana mogućnost izvoza stranica
Adobe Experience Manager (AEM) 5.6	2013	1. *Adobe* preimenuje *CQ5* s nazivom *Adobe Experience Manager* 2. Uveden *Touch UI* 3. Upravljanje poslovima (eng. *Offloading Jobs*)
Adobe Experience Manager (AEM) 6.0	2014	1. *CQSE* poslužitelj zamijenjen s *Jetty*jem 2. Novi *HTML Templating* jezik (*HTL*) – prije poznat kao *Sightly* 3. *Jackrabbit Oak (aka Jackrabbit 3*) 4. Unaprijeđen *CRX* repozitorij sa *CRX* 2.0 na *CRX* 3.0 5. Prijašnji način obrnutog repliciranja je označen kao zastarjeli i uveden je novi *UGC* (User Generated Content)

		6. *Touch UI* postaje funkcionalan 7. Nova implementacija *Social* komponente
Adobe Experience Manager (AEM) 6.1	2015	1. Unaprijeđeno *Touch UI* sučelje za autore 2. Uvedena *Cold standby* topologija za oporavak pri greškama 3. Uvedeni prolazni tijekovi rada (eng. *workflows*) 4. Dodana mnoga poboljšanja za *Touch UI* sučelje
Adobe Experience Manager (AEM) 6.2	2016	1. Dodana mnoga poboljšanja korisničkog sučelja 2. Dodana stranica za upravljanje redovima 3. Predstavljeni *Content Fragmenti* 4. Predstavljeni okviri s mogućnošću uređivanja (eng. *Editable Templates*) za template autore
Adobe Experience Manager (AEM) 6.3	2017	1. Uvedene *Core* komponente 2. *Experience Fragmenti* 3. Stabilna mrežna revizija *Clean up*-a 4. Masivni radni tijek (eng. *Bukl Workflow*) 5. Poboljšane performanse i skalabilnost 6. *3D* Sadržaj 7. *Asset Templates* 8. Poboljšana kvaliteta formata datoteka i upravljanja bojama
Adobe Experience Manager (AEM) 6.4	2018	1. Sustav stiliziranja 2. Kratki kodovi 3. *Experience Fragmenti* u izgradivim blokovima s mogućnošću integracije s *Adobe Target* platformom 4. Strukturirani sadržaj 5. Mogućnost uređivanja radnih tijekova kroz *Touch UI* 6. Izvoz i uvoz metapodataka 7. Kaskadna pravila za metapodatke 8. *AEM* pametni tagovi i pametno prevođenje pri pretraživanju 9. Izvještaj dokumenata (eng. *Asset Reports*)

(Prema: *Ankur Ahlawat,* 2016.)

3.2. Granite

Granite je platforma za izgradnju robusnih i skalabilnih aplikacija. Podržava tzv. otvorenu arhitekturu koja je bazirana na „otvorenim standardima“ (*JCR* i *OSGi*) i projektima otvorenog koda (*Apache Sling* i *Apache Jackrabbit*). *Granite* je *Adobeov* otvoreni skup tehnologija za web, a *AEM* je izgrađen na *Granite*u. *Granite* je projekt otvorenog razvoja (eng. *Open development*), ali ne i otvorenog koda (eng. *Open source*).

Otvoreni razvoj projekta je relativno novi pojam koji je u praksi prvi put primijenjen u *Apache*u, a zatim i u *Adobe*u. Radi se o strategiji projekt menadžera za razvoj softvera koji je otvoren za sve timove unutar tvrtke. Svaki tim može prijaviti problem na projektu ili dati prijedlog za poboljšanje, a moguće je i raditi promjene direktno u kodu, iako je projekt implementiran od strane drugih timova. Tako uspostavlja se mnogo veća kontrola napisanog koda iako on nije javno dostupan. U tvrtkama često određeni timovi imaju pauze između prelaska s projekta na projekt ili se intenzitet projekta s vremenom smanjuje. Također neki programeri svoj posao odrađuju brže od drugih pa tu postoji prilika za sudjelovanje u poboljšanju projekata drugih timova. Projekti otvorenog razvoja mogu biti i projekti otvorenog koda, ali to nije uvijek slučaj.

S tehničkog gledišta *Granite* omogućava sljedeće:

- Pokretač aplikacije: *QuickStart* aplikacija omogućava pokretanje samostalne *Java* aplikacije za isporuku (eng. *Deployment*) u postojećem kontejneru servleta ili aplikacijskom poslužitelju;
- *OSGi* okvir u kojemu se događa isporuka;
- Podupiruće servise pri pokretanje aplikacije: servis za *dnevnik rada*, *Http* servis, različite *admin* servise, deklarativni servisi, servisi *meta*-podataka;
- Poseban okvir za dnevnike rada koji podržava sljedeća aplikacijska programska sučelja (u nastavku rada API) za zapis logova: *SLF4J*, *Log4F*, *Apache Commons Logging* i *OSGi Log Servis;*
- *JCR API* Specifikacija bazirana na *Apache Jackrabbit*-u i *OSGI*-ju;
- *Apache Sling Web* okvir.

Sljedeća slika prikazuje dijagram upotrebe *Granite* platforme i *AEM*-a.

Slika 2 - *AEM* i *Granite* usporedba (Prema: *Adobe Systems Incorporated*, 2017.)

Iz slike je lako zaključiti kako je *Granite* sloj koji je arhitekturalno ispod *AEM*-a i podržava njegove pozadinske funkcije.

Jedan od brojnih okvira koje pruža *Granite* platforma je i *Granite* korisničko sučelje (eng. *Granite UI*).

Granite korisničko sučelje

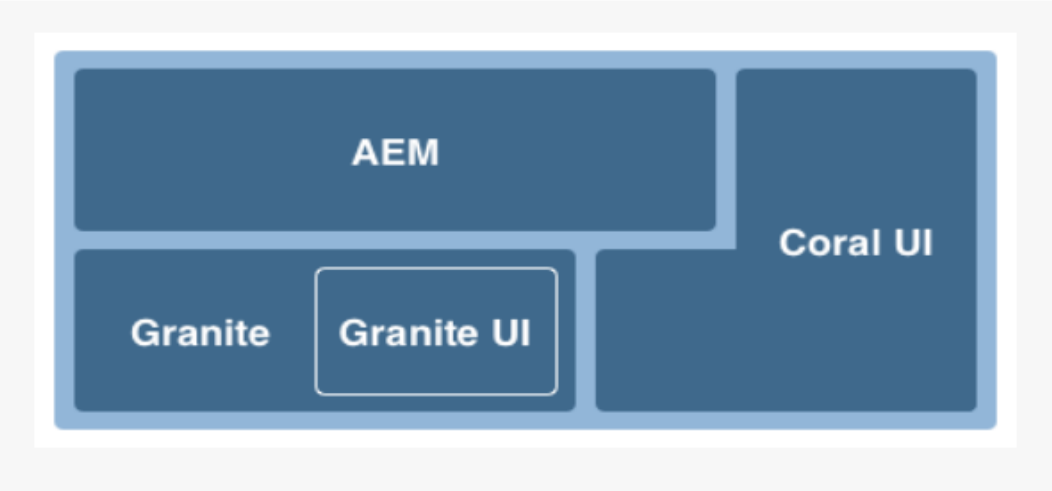
Osnovni ciljevi *Granite* korisničkog sučelja su:

- Omogućavanje granularnih dijelova korisničkog sučelja (eng. *Widget*),
- Implementacija koncepta korisničkog sučelja kao i najboljih praksi,
- Proširivo i nadogradivo administratorsko korisničko sučelje.

Ciljevi se zapravo podudaraju sa zahtjevima *Granite* korisničkog sučelja, a to su sljedeći:

- Mobilni pristup je na prvom mjestu,
- Proširivost,
- Laka nadogradnja.

Kako bi čitateljima bilo lakše shvatiti kako se *Granite* korisničko sučelje uklapa u dosadašnju arhitekturu, slijedi prikaz:



Slika 3 - *Granite* korisničko sučelje u trenutnoj arhitekturi (Prema: *Adobe Systems Incorporated*, 2017.)

Granite korisničko sučelje koristi arhitekturu baziranu na *REST*-u. Služi za implementaciju komponenti kao biblioteka koje su namijenjene za korištenje u *web* aplikacijama baziranim na sadržaju. U *Granite UI*-u postoji standardizirano korisničko sučelje s granularnim dijelovima koje je proširivo. Dizajniran je kako za mobilna, tako i za stolna računala.

Coral UI

Najnovija verzija *AEM*-a je *AEM* 6.5, dok se trenutno preporučuje korištenje verzije *AEM* 6.4 koja je posljednja stabilna verzija. Kada bi bilo riječi o najvećoj promjeni koja se dogodila u povijesti *AEM*-a, mnogi bi se složili da se radi o prelasku na verziju *AEM* 5.6. Tada je *Adobe* izbacio popuno novi sustav baziran većinom na *Java* kodu i *XML* datotekama, koji su u mnogome zamijenili dotadašnji *JavaScript*. Osim jezika koji djeluju u pozadini, u potpunosti se promijenio izgled sučelja *AEM* poslužitelja, kao i način na koji funkcionira tehnologija. Novi sustav se naziva *Touch UI*, a dotadašnji se nazivao *Classic UI*.

S *Touch UI*-om stigla je nova implementacija *Adobe*jevog vizualnog stila za *Touch UI* koja nudi konzistentnost korisničkog iskustva na svakoj *AEM* stranici. Spomenuta implementacija naziva se *Coral UI* (*CUI*). *CUI* je zapravo biblioteka koja je dostupna *AEM* korisnicima za izgradnju *web* aplikacija i sučelja u granicama u kojima im to dopušta licenca.

Coral UI neće biti detaljno objašnjavan jer se jako rijetko modificira, ali ima čestu upotrebu.

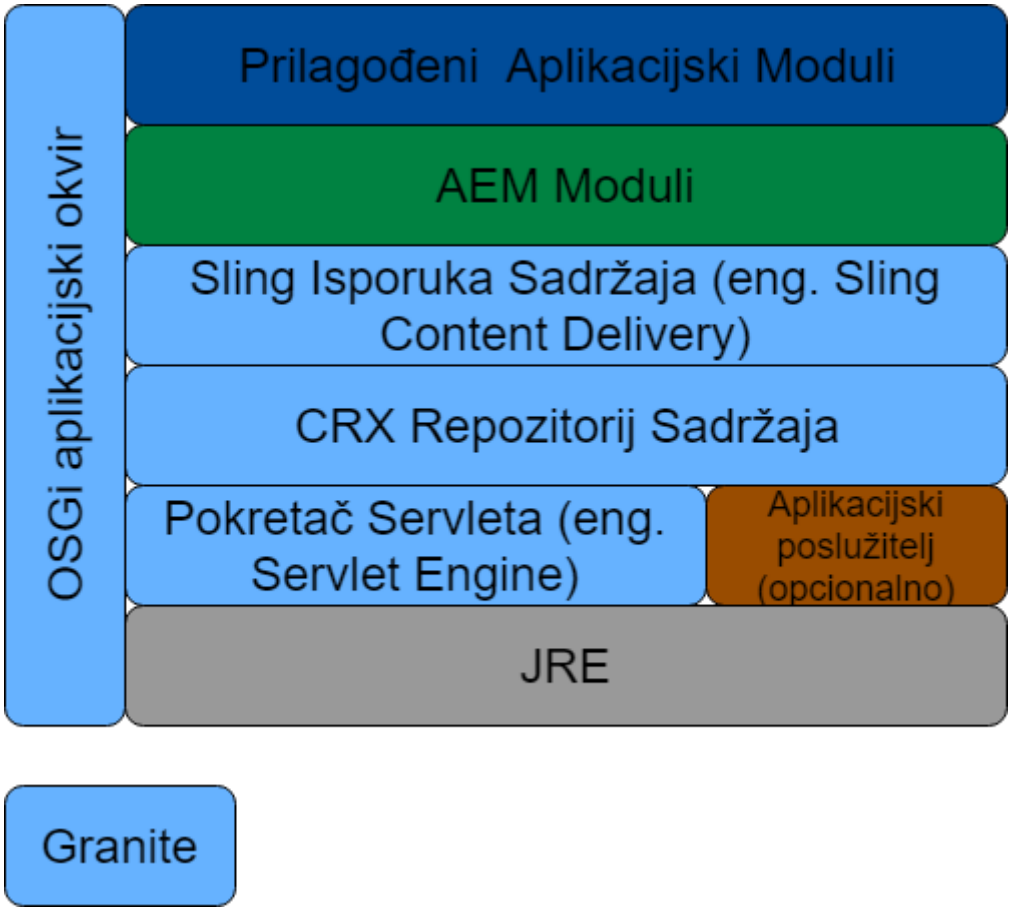
3.3. Open Service Gateway initiative

OSGi (eng. *Open Service Gateway initiative*) je specifikacija za pisanje modularnih sustava i platformi u *Java* programskom jeziku koji implementiraju modele dinamičkih komponenti, što ne postoji u samostalnom *Java* Virtualnom Uređaju (kraće JVM). Aplikacije ili komponente su u formama paketa (eng. *Bundle*) koji se mogu posebno isporučiti (eng. *Deploy*), mogu biti odvojeno instalirane, pokrenute, zaustavljane, nadograđivane i izbrisane bez potrebe za ponovnim pokretanjem cijelog sustava. Cijeli proces manipulacije s *Java* paketima (klasama) vrlo je detaljno specificiran. Upravljanje životnim ciklusom aplikacije izvršava se preko *API*-ja koji omogućavaju udaljeno preuzimanje.

Dakle, smisao *OSGi*-ja je omogućavanje je modularnog okruženja u kojemu je suradnja pojednostavljena jer se aplikacije mogu izvršavati i implementirati kao manji paketi.

Sadržaj aplikacije u potpunosti je spremljen u repozitorij sadržaja što znači da se oporavak aplikacije (vraćanje na neku od starijih verzija) odvija na razini repozitorija. Kao dio aplikacijskog pokretanja (eng. *application runtime*) pokreće se *Apache Sling*, aplikacijski okvir baziran na *REST* principima koji omogućava rad s cijelim repozitorijem koristeći *HTTP* i druge protokole.

Kako bi cijela slika *OSGi*-ja bila jasnija, slijedi grafički prikaz.



Slika 4 - *OSGi* arhitektura u *AEM*-u (Prema: *Adobe Systems Incorporated*, 2017.)

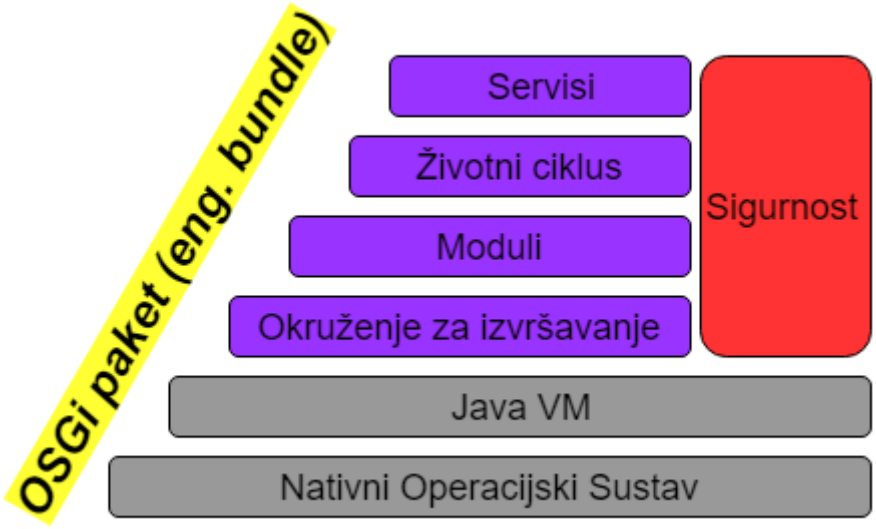
Apache Felix je implementacija *OSGi*-ja otvorenog koda koja se koristi u *AEM-u*.

Open Service Gateway initiative arhitektura

OSGi je specifikacija s visokom razinom apstrakcije, pa čak i ako imate iskustvo rada u *AEM*-u jer je to jedan od procesa koji djeluje iz pozadine i često se poistovjećuje s *AEM*-om. Ono što je važno jest da *OSGi* pruža modularnosti i time omogućava lako uočavanje koji paketi su vanjski, a koji se razvijaju unutar *AEM*-a. Tako je znatno olakšan rad s paketima. Također, uvodi pojam servisa u *AEM* pomoću kojih je ostvarena slaba povezanost (eng. *loosley coupled*).

Što se tiče poslovne strane, *OSGi* također pridonosi poboljšanju smanjenja potrebe za dodatnim operacijskim troškovima jer integrira više uređaja i mrežnih okruženja.

Slijedi prikaz opisanog na slici.



Slika 5 - *OSGI* slojevi (Prema: *Adobe Systems Incorporated*, 2017.)

OSGi paketi su *JAR* komponente s dodatnom manifest datotekom u zaglavlju. Servisi sadrže poslužiteljsku stranu okvira i tu se odvija upravljanje servisima. Životni ciklus je sloj koji upravlja i prikazuje stanje svakog *OSGi* paketa. On služi kako bi se instalirali ili izbrisali objekti te zaustavljali i ponovno pokretali. Moduli predstavljaju prostor *OSGi* paketa, sadrže sve ono s čime se manipulira u sloju životnog ciklusa. Sigurnost je opcionalan sloj koji kada se aktivira, validira svaki *OSGi* paket i kontrolira dozvole pristupa komponentama. Okruženje za izvršavanje je posljednji sloj u kojemu djeluje *OSGi* paket, a odabran je tako da odgovara hardveru i operacijskom sustavu.

OSGi paketi su implementirani u hijerarhiju pomoću *Composite* uzorka dizajna.

Open Service Gateway initiative paketi

Svaki od paketa je kolekcija čvrsto povezanih klasa s dinamičkim učitavanjem, *JAR* datoteka i konfiguracijskih datoteka koje eksplicitno deklariraju vanjske ovisnosti (eng. *dependency*). *OSGi* paketi sadrže i dodatnu datoteku manifesta. *OSGi* metapodaci su sadržani u zaglavlju *META-INF* odnosno manifest datoteke. Dodatne informacije su sljedeće:

- Naziv *OSGi* paketa
  - Simbolično ime koje *OSGi* koristi da utvrdi jedinstveni identitet paketa;
  - Deskriptivno ime koje je čovjeku razumljivo;
- Verzija paketa;

    - Lista ulaznih i izlaznih servisa paketa;
- Opcionalne dodatne informacije poput:
    - Minimalna *Java* verzija koju paket zahtjeva;
    - Vendor paketa;
    - Autorska prava;
    - Kontakt, adresa i sl.;

OSGi paketi su slabo povezani (eng. *loosley coupled*).

Svaki novi paket automatski se instalira u *OSGi* kontejner koji pruža automatsko prepoznavanje vanjskih zavisnosti.

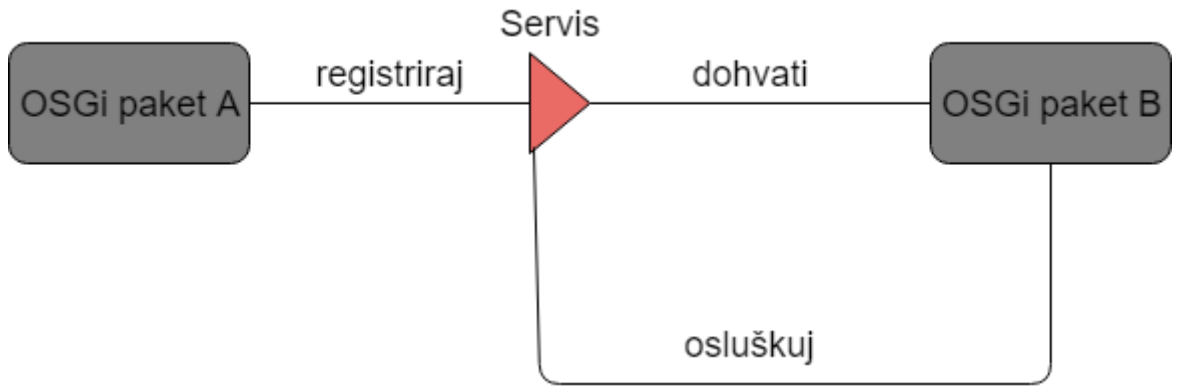
Moduli

O modularnosti je već bilo riječi u uvodu ovog poglavlja. Razlika između *OSGi* paketa i paketa u *Javi* je u vidljivosti sadržaja, tj. u *JAR* datoteci sadržaj je vidljiv svim drugim *JAR* datotekama. U *OSGi*-ju je sadržaj *JAR* datoteke skriven osim u slučaju kada je eksplicitno izvezen. Ako jedan *OSGi* paket želi koristiti drugu *JAR* datoteku, mora eksplicitno izvesti dijelove koji su joj potrebni, inače nema dijeljenja sadržaja.

Skrivanje koda (sadržaja *JAR* datoteka) pruža mnoge prednosti kao što su mogućnost korištenja više verzija iste biblioteke unutar iste *Java VM*. Dijeljenje koda je omogućeno djelomično samo kako bi podržalo *OSGi* pakete, ali cijela suradnja između *OSGi* paketa morala bi se odvijati preko servisa.

Servisi

OSGi paket može napraviti objekt i registrirati ga u registrator servisa u jednom ili više sučelja. Drugi paketi mogu pristupiti registratoru, pregledati listu svih objekata te ih koristiti i osluškivati pojavljivanje i nestajanje servisa. Postoje i filteri koji mogu prikazivati samo servise od interesa za određeni paket, a registracija servisa zahtjeva određena pravila i atribute.



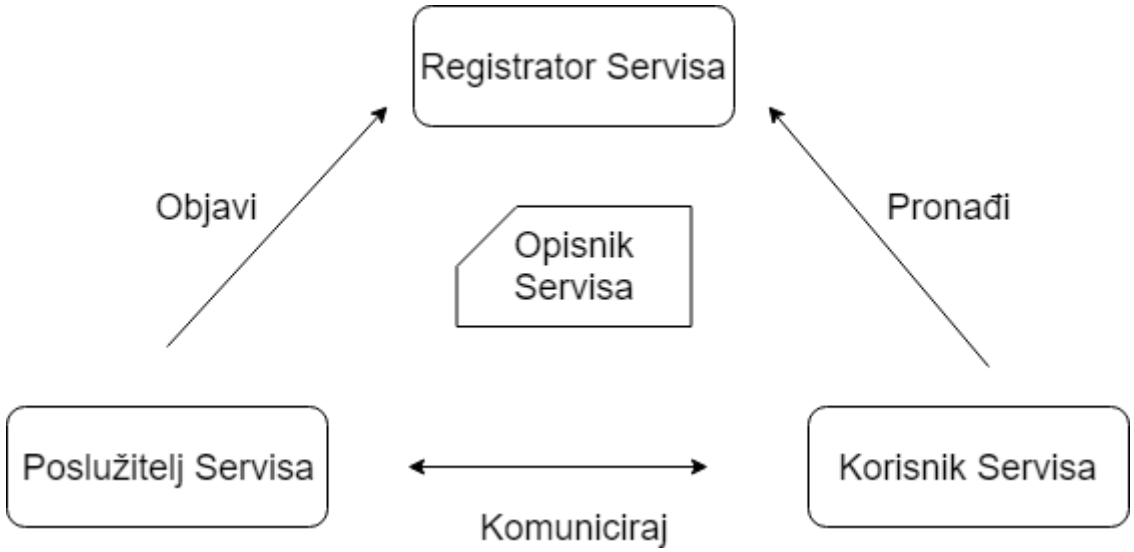
Slika 6 - *OSGi* paketi – komunikacija (Prema: *Adobe Systems Incorporated*, 2017)

Servisi su dinamični. To znači da svaki paket može „donijeti odluku“ o odjavljivanju određenog servisa iz registratora dok ga drugi paketi još uvijek koriste. Tada svi paketi koji koriste taj određeni servis (objekt) moraju postići da sve reference na taj isti servis budu odbačene.

Model registracije servisa

OSGi pruža servisno orijentirani komponentni model koji koristi objavi-pronađi-poveži (eng. *publish/find/bind*) mehanizam. To bi značilo da paket funkcionira na način „trebam X“ i tada je X uvezen (eng. *inject*).

Budući da ne postoji slušač na koji se paket može osloniti prilikom zahtjeva za objektom, postoji dinamički pregled servisa ostvaren preko *Whiteboard registry* uzorka koji funkcionira na sljedeći način:



Slika 7 - OSGi model registracije servisa (Prema: Adobe Systems Incorporated, 2017.)

Umjesto registracije „slušač – korisnik“ objekta uz poslužitelja servisa, korisnik kreira samo objekt koji implementira sučelje *OSGi* slušača koji sadrži metodu koja će biti pozvana svaki put kada se objekt od interesa pojavi u registratoru. Kada se pojavi objekt od interesa, poslužitelj servisa poziva sve objekte istog tipa i i paralelno s tim sve potrebne metode. Cijeli teret održavanja odnosa između poslužitelja servisa i korisnika servisa je prebačen na *OSGi* okvir. Prednost toga jest što je *OSGi* okvir uvijek svjestan statusa i životnog ciklusa sadržanih paketa te automatski deregistrira sve servise paketa jednom kada se taj paket zaustavi.

Prednosti Open Service Gateway initiative servisa

Životni ciklus servisa traje koliko i životni ciklus paketa koji ga isporučuje. Prednosti korištenja *OSGi* servisa su sljedeće:

- Za rad servisa potrebno je malo memorijskih resursa;
- Pretraga je bazirana na nazivu servisa;
- Metode se pozivaju (eng. *invoke*) direktno;
- Dobra praksa dizajna aplikacija;
- Odvojena sučelja od implementacije;
- Osigurava ponovno korištenje, održivost, slabu povezanost i kasno povezivanje;

Deklarativni servisi

Deklarativni servisi dio su *OSGi* kontejnera i pojednostavljuju izradu komponenti koje objavljuju ili referenciraju *OSGi* servise.

Prednosti korištenja deklarativnih servisa:

- Nema potrebe za pisanjem koda koji objavljuje ili konzumira servise;
- Objavljene komponente se ne objavljuju u trenutku objave (klasa implementacije se ne objavljuje odmah), nego u trenutku kada ih klijent zahtjeva;
- Komponente imaju svoj životni ciklus povezan sa životnim ciklusom *OSGi* paketa;
- Komponente mogu automatski dobiti konfiguracijske podatke kroz konfiguracijskog administratora (eng*. Configuration Admin*);

Komponente se deklariraju preko *XML* konfiguracijskih datoteka. Te datoteke su popisane u zaglavlju manifest datoteke određenog *OSGi* paketa.

Deklarativni servisi su dobra alternativa sljedećim aktivnostima:

- Pisanju klase *Aktivatora;*
- Registraciji paketa u *OSGi* okvir;
- Korištenju pratitelja promjena na servisu;

Deklarativni servisi čitaju opis iz pokrenutog *OSGi* paketa. Taj opis su spomenute *XML* datoteke koje definiraju set komponenti za taj paket.

Isporuka

OSGi paketi se isporučuju (eng. *deploy*) na *OSGi* okvir – na izvršno okruženje *OSGi* paketa (eng. *bundle runtime environment*). To je zajedničko okruženje svih paketa jer su svi pokrenuti na istoj *VM* i mogu dijeliti kod. Koriste se eksplicitna učitavanja i izvozi da se povežu *OSGi* paketi kako se u okvir ne bi učitavale klase. *API* je jednostavan i omogućava instalaciju, pokretanje, zaustavljanje i ažuriranje paketa, kao i enumeraciju korištenja servisa.

Tehnološke prednosti Open Service Gateway initiative-a

Generalno, *OSGi* donosi sljedeće benefite:

- **Reducirana kompleksnost** – *OSGi* paketi kao moduli ograničavaju vidljivost drugih *OSGi* paketa, ali zadržavaju jednostavnost međusobne komunikacije tih paketa, tj. modula preko servisa.

- **Ponovna upotreba** – *OSGi* model komponenti olakšava korištenje vanjskih komponenti u aplikaciji. Danas postoje mnoge komponente otvorenog koda koje su prilagođene da se mogu koristiti kao *OSGi* komponente. Slično je i s komercijalnim bibliotekama.
- **Stvarni svijet** – *OSGi* okvir je dinamičan. Omogućava konstantnu promjenu servisa i *OSGi* paketa. Tradicionalni *Java* programeri rijetko vide prednost stalne promjene servisa, međutim pokazalo se kako razvoj aplikacija kao i zahtjevi klijenata postaju sve dinamičniji pa su stoga brze promjene i isporuke servisa velika prednost. Primjerice, postoji servis koji instalira uređaj u mrežu. Ako je taj uređaj detektiran, servis je registriran. Ako se uređaj isključi iz mreže, servis se de-registrira. Postoji mnogo sličnih poslovnih slučajeva gdje se dinamički model servisa pokazao korisnim. Ovo ne reducira samo pisanje koda, nego pruža i globalnu vidljivost, alate za ispravljanje pogrešaka (eng. *debugging*) itd... Pisanje koda u tako dinamičnom okruženju se čini vrlo zahtjevno, ali zapravo je većina posla već sadržana u gotovim klasama i programskim okvirima. *OSGi* paketi komuniciraju preko servisa tako da se servisi dijele kao objekti među paketima.
- **Jednostavna isporuka** – *OSGi* nije samo standard za razvoj komponenti, nego i za njihovu instalaciju i upravljanje. *OSGi* sadrži agenta preko kojeg se *OSGi* tehnologija može jednostavno integrirati u postojeće i buduće tehnologije.
- **Dinamično ažuriranje** – Omogućeno brzom instalacijom, deinstalacijom, pokretanjem i zaustavljanjem *OSGi* paketa bez narušavanja funkcioniranja ostatka sustava.
- **Prilagodljivost** – upravljanje *OSGi* paketima koji moraju funkcionirati u sustavu i ako njihove vanjske ovisnosti nisu trenutno dostupne. Paketi osluškuju servise i prilagođavaju se dobivenim informacijama.
- Transparentnost, verzioniranje, jednostavnost, mala potreba za memorijom (*OSGi* 4 okvir se može implementirati sa 300 KB *jar* datotekom), brzina, sigurnost, mogućnost pokretanja u *Java VM*, široka uporaba i podrška ključnih kompanija u svijetu tehnologije (*Oracle, IBM, Samsung, Nokia, Motorola, NTT, Simens, Hitachi, Deutsche Telekom, Redhat, Ericsson* i mnoge druge).

Komponente u Open Service Gateway initiative-u

Komponente su osnovni blokovi izgradnje *OSGi* aplikacije. Komponente se nalaze u *OSGi* paketima. Jedan paket sadrži jednu ili više komponenti.

Komponente su *Java* objekti kojima upravlja *OSGi* kontejner. Komponente mogu isporučivati servise, ali i konzumirati jedan ili više servisa preko sučelja. Komponente mogu biti objavljene kao servisi, ali mogu imati i ovisnosti o drugim komponentama (servisima). Te ovisnosti ovise o životnom ciklusu paketa u kojima se nalaze pa će komponenta biti aktivirana od strane *OSGi* kontejnera samo ako su aktivne sve komponente od potrebnih zavisnosti (eng. *dependency*). Svaka komponenta ima svoju klasu implementacije i može implementirati druga javna sučelja.

Da bi se paket smatrao komponentom, potrebno mu je sljedeće:

- *XML* datoteka koja opisuje servis koji paket pruža i sve njegove zavisnosti o drugim servisima;
- Manifest datoteku sa zaglavljem u kojemu je deklarirano kako se paket ponaša kao komponenta;
- Metode „*Activate*“ i „*Deactivate*“ u implementacijskoj klasi;
- Deklarativni servis koji će upravljati komponentom;

3.4. Java Content Repository

Java repozitorij sadržaja (eng*. Java Content Repository*), u nastavku teksta *JCR*, je baza podataka koja podržava strukturirane i nestrukturirane podatke, verzioniranje i promatranje. Svi podaci uključujući *HTML*, *CSS*, *JavaScript* ili *Javu*, slike i video sadržaje su spremljeni u *JCR* bazu podataka. Izgrađena je *Apache Jackrabbit Oak* tehnologijom koja je otvorenog koda. *AEM*-ova implementacija *JCR*-a poznata je kao ekstremni repozitorij sadržaja (eng. *Content Repository eXtreme*), u nastavku teksta *CRX*. *CRX* je kao tehnologija izbačen iz *AEM*-a, a na njegovom mjestu je *Granite* platforma koja koristi *Apache Jackrabbit Oak* tehnologiju.

Prednosti korištenja *JCR*-a:

- *JCR* pruža generičke aplikacijske podatke za strukturirani, ali i nestrukturirani sadržaj. Dok je datotečni sustav pogodan za spremanje nestrukturiranih podataka, hijerarhijskog sadržaja, baza podataka je pogodna za spremanje strukturiranog sadržaja. *JCR* je kombinacija obje navedene arhitekture;
- *JCR* podržava imenski prostor (eng. *namespace*). To sprječava koliziju u nazivima različitih podataka. Imenski prostor u *JCR*-u je definiran s prefiksom i odvojen dvotočkom. Npr. *jcr:title*.

Razlike između *JCR*-a i relacijske baze podataka su sljedeće:

- Hijerarhija – sadržaj se može spremati prema zahtjevima klijenta. Mogu se raditi grupacije sadržaja i različite vrste podataka se mogu spremati u iste grupe. Tako se postiže laka navigacija;
- Fleksibilnost – sadržaj se može prilagoditi ili razviti tako da bude bez scheme ili potpuno restriktivan;
- Koristi standardni *Java API;*
- Mjesto gdje je informacija zaista spremljena je apstraktno;
- Podržava pisanje upita i za pretragu teksta (eng. *full-text search*).

Mogućnosti *JCR*-a su sljedeće:

- Upit pisan u *SQL-u, JQOM*-u ili *XPath*-u;
- Uvoz/izvoz baze podataka (*XML*);
- Referencijalni integritet (osigurana postojanost svih podataka u bazi);
- Autentikacija;
- Kontrola pristupa;
- Verzioniranje;
- Promatranje (analiza);
- Zaključavanje i transakcije (eng. *locking and transactions*) (*JTA*) – konkurentnost u transakcijama, omogućava istovremeni rad više korisnika.

JCR pruža i određene servise:

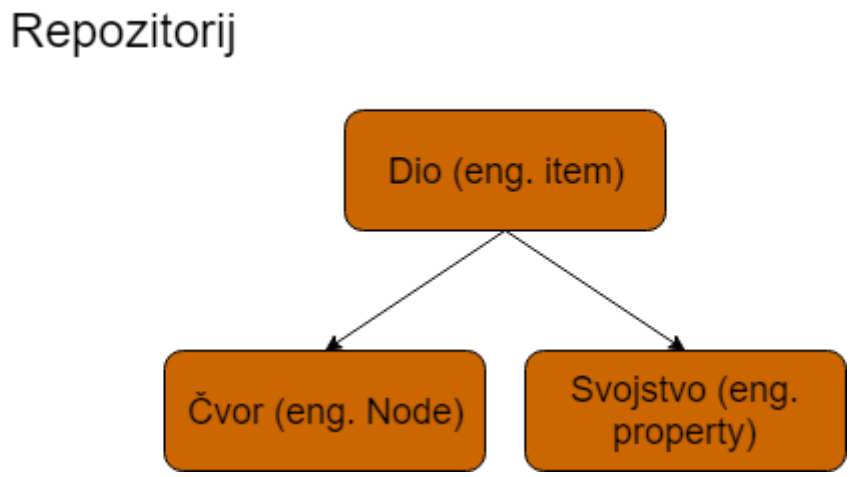
- Verzioniranje na *Author* instanci;
- Pretraživanje cijelog teksta (eng. *full-text search*);
- Upravljanje kontrole pristupa;
- Kategorizacija sadržaja;
- Nadzor događaja nad sadržajem.

Struktura Java Content Repositoryja

JCR repozitorij se sastoji od mnogo različitih dijelova. Svaki dio mora biti ili čvor (eng. *node*) ili svojstvo (eng. *property*). Čvor može imati 0 ili više čvorova koji su mu djeca (eng. *child nodes*). Svojstvo ne može imati podčvorove („djecu“), ali može sadržavati 0 ili više vrijednosti zapisanih u toj postavci. Čvorovi čine strukturu spremljenih podataka, dok se ti podaci zapravo nalaze u vrijednostima svojstava.

Dodatna prednost *JCR*-a je već spomenuta podrška za imenske prostore.

Struktura je prikazana na sljedećoj slici:



Slika 8 - *JCR* repozitorij – struktura (Prema*: Adobe Systems Incorporated*, 2017.)

Podaci u *JCR*-u su spremljeni kao stablo čvorova s pripadajućim svojstvima.

Čvorovi

Čvorovi mogu imati jedan ili više tipova koji su im pridruženi. Tipovi određuju vrste svojstava u tom čvoru, broj i vrstu podčvorova te određene karakteristike u ponašanju čvor ova. Čvorovi preko posebnih referenci (svojstava) mogu imati pokazivače i na druge čvorove. Čvorove je moguće i verzionirati. Tako se može pratiti povijest spremljenih čvorova i mogu se spremati stare verzije dokumenata.

Svojstva mogu imati jednu ili više vrijednosti. Svaka vrijednost ima jedno od 12 različitih tipova. Tipovi svojstava podrazumijevaju osnovne tipove spremljenih podataka poput *Stringa*, numeričkih vrijednosti, *Booleana*, binarnih zapisa i podataka, kao i tipove koji sadrže pokazivače na druge čvorove.

3.4.1.1. Tipovi čvorova

Tipovi čvorova omogućuju strukturu čvorova i svojstava u bazi podataka tako da definiraju za svaki čvor njegove obavezne i dozvoljene podčvorove. Primarni tipovi čvorova definiraju osnovne karakteristike čvora - *jcr:primaryType*. Miješani tipovi čvorova (eng. *Mixin node type*) koriste se kako bi čvorovima dodali dodatne karakteristike vezane za specifičnosti funkcija repozitorija ili metapodataka.

Čvor može imati jedan ili više super tipova. Čvor koji ima super tip naziva se podtip i označava nasljeđivanje karakteristika svog supertipa. Karakteristike koje nasljeđuje su sljedeće:

- Definicije svojstava;
- Definicije podčvorova;
- Ostale atribute kao npr. miješanje tipove.

3.5. Apache Sling

Apache Sling je web aplikacijski okvir za aplikacije koje sadržaj stavljaju na prvo mjesto. Koristi *JCR* (*Apache Jackrabbit Oak*) za spremanje i upravljanje sadržajem. *Apache Sling* je baziran na *REST* principima i pomaže izgradnji aplikacija kao *OSGi* paketa.

Prednosti *Apache Slinga*:

- Projekt otvorenog koda;
- Baziran na *REST* principima;
- Aplikacije su izgrađene kao serije *OSGi* paketa;
- Orijentiran na resurse (svaki resurs ima svoj *URL*) i mapira svaki resurs na *JCR* čvor.

Sling je orijentiran na resurs, tj. baziran na pretpostavci da je sve resurs. Svaki resurs ima svoj *URL* i mapiran je na *JCR* čvor. Primljeni zahtjev najprije se pretvara u resurs, a onda se na temelju tog resursa bira *Servlet* ili skripta koja obrađuje dobiveni zahtjev. Servleti i skripte su također resursi kojima je moguće pristupiti njihovom putanjom resursa. S gledišta razvojnog programera ovo je od velike važnosti jer mu pruža mogućnost dohvaćanja svakog *Servleta*, skripte, filtera, obrade grešaka (eng. *error handlera*) i drugih mogućnosti iz klase *ResourceResolver*.

REST arhitektura

REST je stil softverske arhitekture za distribuirane sustave kao što je *World Wide Web*. Sustavi koji koriste *REST* arhitekturu komuniciraju putem *Hyper Text Transfer* Protokola (*HTTP*) koristeći *GET*, *POST, PUT, DELETE* i druge metode. Sučelje *REST*-a uključuje kolekciju resursa s njihovim identifikatorima. Npr. */trening/english*.

REST je arhitekturalni stil za mrežne aplikacije koji inducira ista arhitekturalna svojstva na kojima postoji *World Wide Web*.

Svakom resursu je moguć pristup preko njegove adrese pa se tako mijenja stanje resursa (čitanje i ažuriranje). Najbolji primjer *REST* arhitekture je *Web*, gdje resursi imaju svoje *URL*-ove i jedinstveno *HTTP* sučelje.

Sljedeća su svojstva *REST* arhitekturalnog stila:

- Dobre performanse i mrežna učinkovitost;
- Skalabilnost;
- Jednostavnost sučelja;
- Brza promjena stanja;
- Vidljivost komunikacije;
- Prenosivost komponenti (portabilnost);
- Sigurnost koju pruža mala vjerojatnost unutrašnjih grešaka.

Općenito, okvir kao što je *HTTP* je sve što je potrebno da bi se napravila distribuirana aplikacija. *HTTP* je vrlo bogat aplikacijski protokol koji omogućava rad sa sadržajem i distribuirano keširanje – privremeno spremanje (eng. *caching*).

Prednosti *REST* arhitekture

Slijede prednosti *REST* arhitekture:

- Koriste se dobro dokumentirane, provjerene i poznate tehnologije i metodologije
- Baziran je na resurs, a ne na metode
- *URI* je globalno vidljiv
- Nema više vrsta protokola
- Nema definiran format za odgovor na zahtjev
- Nasljeđuje *HTTP* sigurnosni model
- Laka je zabrana (ograničavanje) metoda za određene *URI*-ove preko konfiguracije vatrozida

Razumijevanje obrade zahtjeva u *Slingu*

Svi resursi u *Slingu* su u formi stabla. Resurs je najčešće mapiran na *JCR* čvor, ali može biti mapiran i na datotečni sustav ili bazu podataka. Osnovna svojstva koja sadrži većina resursa su sljedeća:

- Putanja – uključuje imena svih resursa od početka datotečnog sustava razdvojenih kosom crtom, slično kao *URL* putanja ili putanja u datotečnom sustavu.

- Naziv – zadnji naziv u putanji resursa
- Tip resursa – svaki resurs ima svoj tip koji se koristi u *Servletu* ili skripti kako bi se pronašla odgovarajuća skripta ili *Servlet* koji obrađuje primljeni zahtjev

U *Slingu* glavni fokus nije na sadržaju ili na tipu sadržaja, nego vodi li primljeni *URL* do resursa koji vodi do Skripte (*Servleta)* kako bi se izvršilo renderiranje sadržaja. Ovo olakšava rad autorima sadržaja čiji se zahtjevi mogu lako prilagođavati.

Sling resursi mogu biti registrirani kao *OSGi* servisi. *Sling* aplikacije koriste skripte ili *Java* servlete kako bi obradile *HTTP* zahtjeve. *GET* metoda označava uobičajeno ponašanje pa nisu potrebni dodatni parametri. Dekompozicijom *URL*-a *Sling* utvrdi *URL* do resursa i ostale informacije koje mogu koristiti pri obradi resursa.

POST metodu obrađuje *Sling PostServlet*. *PostServlet* omogućava zapisivanje novih podataka direktno iz *HTML*-a u *JCR*.

3.5.1.1. Osnovni koraci u obradi zahtjeva

Svaki dio sadržaja u *JCR* repozitoriju je *HTTP* resurs pa je *URL* zahtjev stiže direktno na sadržaj, a ne na proces koji taj *URL* zahtjev procesira. Nakon što je sadržaj određen, određuje se skripta/*Servlet* prema prioritetima:

- Svojstva sadržaja/resursa;
- *HTTP* metoda korištena u zahtjevu;
- Konvekcija naziva u *URL*-u.

Za svaki *URL* zahtjev se izvršavaju sljedeći koraci:

- Dekompozicija *URL*-a;
- Pretraga za datotekom na koju indicira *URL;*
- Obrada resursa;
- Obrada odgovarajuće skripte/servleta;
- Izrada lanca za renderiranje;
- Pokretanje lanca za renderiranje.

3.5.1.2. Dekompozicija *URL*-a

Primjer *URL*-a: http://myhost/tools/spy.printable.a4.html/a/b?x=12

Tablica 2- Proces dekompozicije URL-a

Protokol	Domaćin (eng. host)	Putanja do sadržaja	Selektor(i)	Ekstenzija		Sufiks		Parametri
http://	myhost	tools/spy	printable.a4	html	/	a/b	?	x=12

(Prema: *Adobe Systems Incorporated*, 2017.)

U sljedećoj tablici opisane su komponente dekomponiranog URL-a.

Tablica 3 - Opis komponenti *URL*-a

protokol	*HTTP* protokol
domaćin (eng. host)	naziv web stranice
putanja do sadržaja	putanja do sadržaja koji će biti renderiran. Koristi se u kombinaciji s proširenjem
selektor(i)	alternativna metoda renderiranja sadržaja
proširenje (eng. extension)	format sadržaja i specifikacija skripte koju je potrebno renderirati
sufiks	može se koristiti kao dodatna specifikacija
parametri	parametri koje zahtjeva sadržaj

(Prema: *Adobe Systems Incorporated*, 2017.)

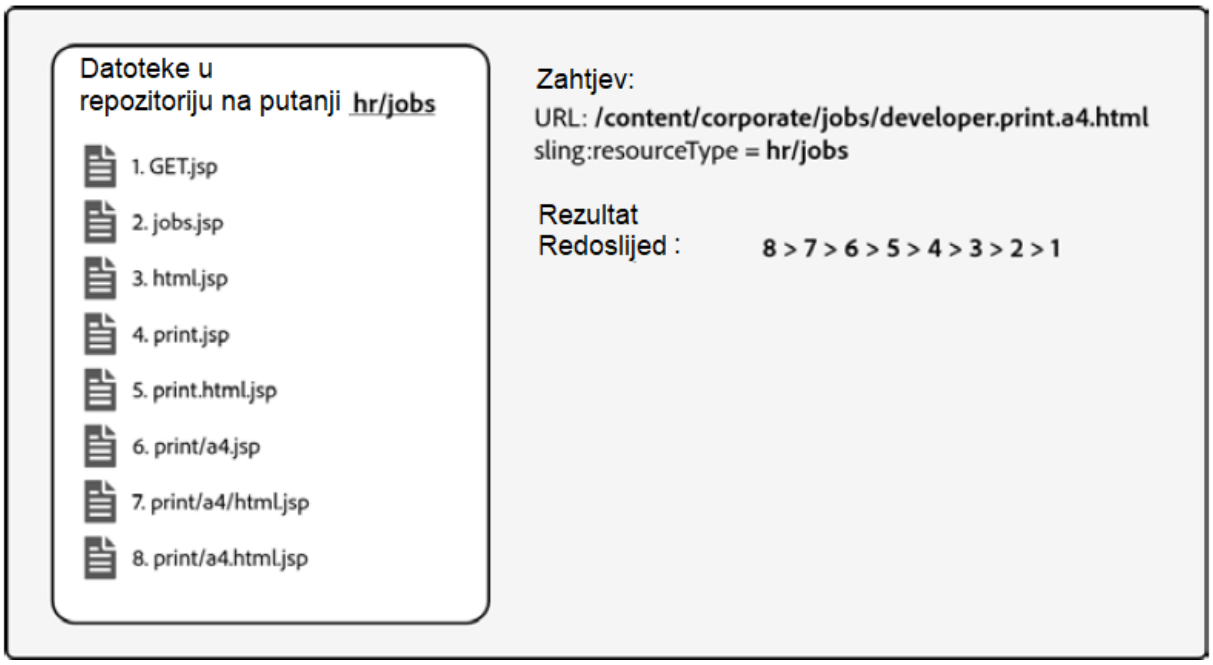
3.5.1.3. Mapiranje zahtjeva resursu

Jednom kada je *URL* dekomponiran, čvor sadržaja je lociran iz putanje od sadržaja. Taj čvor je identificiran kao resurs i obavlja određene radnje kako bi mapirao zahtjev. To je vidljivo na primjeru sljedećeg *URLa*: *http://myhost/tools/spy.html*

- *Sling* provjerava postoji li čvor na lokaciji specificiranoj u *URL*-u. U ovom primjeru to je čvor *spy.html*
- Ako čvor ne postoji na toj lokaciji, iz pretrage se izuzima ekstenzija (proširenje) pa se traži čvor na lokaciji *spy*
- Ako čvor ni tada nije pronađen, *Sling* vraća *HTTP* kod 404 – resurs nije pronađen
- Ako je čvor pronađen, iz svojstva *resourceType* se locira skripta/servlet koji je potrebno pokrenuti za generiranje sadržaja

Sve skripte su spremljene na */apps* ili */libs* mapi i pretražuju se tim istim redoslijedom. Ako skripta nije pronađena ni u jednoj od navedenih mapa, renderira se unaprijed zadana skripta za takve slučajeve.

U slučaju da je pronađeno više skripti koje odgovaraju lokaciji u svojstvu pronađenog čvora, renderira se ona skripta koja najbolje odgovara pretrazi.



Slika 9 - *Sling* proces određivanja skripte (Prema*: Adobe Systems Incorporated*, 2017.)

Više selektora čini pretragu točnijom. Kada se pronađe željeni čvor, njegovo svojstvo *resourceType* je npr *hr/jobs*. *ResourceType* prikaziva gdje se nalazi skripta koja će prikazati traženi sadržaj. U situaciji s prethodne slike pri slanju zahtjeva na *URL:*

„/content/corporate/jobs/developer.print.a4.html“ pri čemu je *sling:resourceType* postavljen na vrijednost *hr/jobs* izvršava se pretraga na putanji hr/jobs i traže se sljedeće skripte:

„*print/a4.html.jsp*“, ako nije pronađeno tada se pokušava naći „*print/a4/html.jsp*“, zatim „*print/a4.jsp*“ i tako dalje redosljedom do br. 1 sa slike. Ako navedena skripta ne postoji, tada servlet vraća vrijednost 404 (eng. *Not Found*).

3.5.1.4. Prilagođavanje resursa

Kako bi dvije različite klase radile skupa, u *AEM*-u se koristi uzorak dizajna *Adapter*. *Sling* nudi gotovi *Adapter* uzorak za pretvaranje objekata koji koriste *Adaptable* sučelje. Sučelje pruža *adaptTo()* metodu koja pretvara objekt u tip koji je metodi dan kao argument.

Spomenuta metoda često se koristi za pretvaranje resursa u čvor:

```
Node node = resource.adaptTo(Node.class);
```

3.5.1.5. Rad sa Sling Modelima

Sling Modeli dopuštaju mapiranje *Java* objekata u *Sling* resurse. *Sling* modeli su:

- Potpuno bazirani na anotacijama
- Preporučuje se korištenje standardnih anotacija
- Izdvojivi
- Podržavaju svojstva resursa, *SlingBinding*-e, *OSGi* servise i sl.
- Mogu prilagođavati (eng. *adapt*) više objekata
- Podržavaju i klase i sučelja
- Rade s postojećom *Sling* infrastrukturom

Sling Modeli koriste se u sljedećim situacijama:

- Ukoliko postoji objekt modela (kao *Java* klasa ili sučelje)
- Kada je objekt potrebno koristiti s *AEM*-om bez izrade vlastitih adaptera
- Ukoliko je potrebno mapirati *POJO* klase u *Sling Resurs*

Sling Modeli su *OSGi* paketi koji su bazirani na anotacijama i tako omogućavaju postizanje istih funkcionalnosti kao *OSGi* servisi uz manje koda.

```
@Model(adaptables=Resource.class)
Public interface MyModel {

      @Inject
      String getPropertyName();
}
```

Ovaj model učitava svojstvo *propertyName* iz resursa u *Java* kod. Dovoljno je staviti *@Inject* anotaciju i za ovo svojstvo će se pronaći odgovarajući resurs i taj resurs će se prilagoditi (eng. *adapt*) u *ValueMap* objekt. Tada se iz tog objekta uzimaju svojstva potrebna u *Java* kodu.

Klijentski kod ne mora znati da su *Sling Modeli* korišteni, dovoljno je da koristi *Sling Adapter* okvir:

```
MyModel model = resource.adaptTo(MyModel.class);
```

Postoje brojne druge anotacije kojima je lako manipulirati i pozivati resurse u *Java* kod što značajno olakšava proces manipulacije *JCR* čvorovima i svojstvima. Također, moguće je izrađivati prilagođene anotacije preko *org.apache.sling.models.spi.Injector* sučelja.

3.6. HTML Templating Language

HTML Template Language (kraće HTL) je *Adobe*jev jezik za predloške (eng. *template*). To je preporučeni jezik za razvoj komponenti prilikom korištenja *AEM-a*. U ranijim verzijama *AEM*-a se koristio *JSP*, ali *HTL* se pokazao kao bolji izbor pa je nova referentna stranica *Adobe*ja za *AEM – We Retail* napisana u *HTL*-u.

HTL definira *HTML* rezultate specificiranjem prezentacijske logike u koju se učitavaju dinamičke vrijednosti iz *Java* koda. *HTL* se razlikuje od ostalih jezika za pisanje predložaka po sljedećem:

- ***HTL* je *HTML5***: Svaka *HTL* datoteka je valjana *HTML* datoteka. Sve specifičnosti *HTL* jezika su specificirane u „*data*“ atributu ili u *HTML* tekstu. Bilo koji *HTML* editor može otvoriti *HTL* datoteke.
- **Razdvajanje interesa** (*Web* dizajner protiv *Web* razvojnog programera): *Web* dizajneri mogu razvijati samo dizajn *HTML*-a, a sva poslovna logika odvojena je u *Java* klase koje razvojni programeri mogu pozvati u gotovim predlošcima izrađenim od strane dizajnera.
- **Zadana sigurnost**: *HTL* automatski filtrira i izbjegava mogućnosti da tekst bude direktno na prezentacijskom sloju kako bi spriječio *XSS* sigurnosnu slabost (*cross-site scripting*).
- **Kompatibilnost s *JSP* ili *ESP* tehnologijama**

Dva osnovna cilja *HTL*-a:

- Pojednostavljen proces razvoja
  - Manje koda
  - Čišći kod
  - Intuitivniji kod
  - Efektivniji kod
- Sigurnost
  - XSS zaštita
  - Zadana sigurnost
  - Automatska zaštita

- Osjetljivost na sadržaj

Dva su osnovna tipa sintakse u *HTL*-u. Prvi tip su *HTL* Blok Iskazi – definiraju strukturu unutar predloška i postoje u *HTML* data atributu koji je u *HTML*-u 5 namijenjen za korištenje od strane vanjskih aplikacija (tehnologija). Svi *HTL* specifični atributi definiraju se s prefiksom *data-sly.*

Drugi tip sintakse su *HTL* izrazi. Delimitirani su znakovima *${* i *}*. Pri svakom pokretanju ove vrijednosti se validiraju prije nego što se priključe *HTML*-ovom toku.

4. Apache Lucene

Lucene je *Java* biblioteka koja služi za pretraživanje teksta u aplikacijama i Web stranicama. Napisao ju je *Doug Cutting* godine 1999., dok 2001. prelazi u *Apache Software Foundation* kao *Java* projekt otvorenog koda. Ona nudi mnoge detalje implementacije sustava pretraživanja o kojima će više biti napisano u nadolazećem tekstu.

Apache Lucene funkcionira na temelju sljedećeg dijagrama:

Slika 10 - *Apache Lucene Dijagram* (Prema: *Edwood Ng*, 2015.)

Dohvaćanje informacija je mehanizam koji je neovisan o *Lucene*u. To mogu biti različite vrste informacija u različitim strukturama i iz različitih izvora:

- *Web*
- *XML*
- *TXT*
- *JSON*
- *PDF*

Dodavanje informacija indeksu odnosi se na dodavanje prikupljenih podataka i informacija indeksima (što se naziva indeksiranje) i njihovim spremanjem u indeks. Indeksiranje je u potpunosti ovisno i odvija se preko *Lucene*a.

Pretraga informacija odnosi se na pretragu ranije spomenutog indeksa u kojemu su informacije spremljene. Pretraga se također odvija preko *Lucene*a.

4.1. Dodavanje informacija

Terminologija koju *Lucene* koristi je sljedeća:

Spremište podataka (dokumenata) se naziva indeks, dok se proces spremanja dokumenata naziva indeksiranje. Skup podataka naziva se dokument. Svaki dokument se sastoji od više polja, a svako polje ima tri atributa. Ti atributi su naziv, tip i vrijednost (string ili binarni zapis).

Najbolje je navedeno prikazati na primjeru aplikacije pretraživanja vijesti na portalima, gdje bi jedan dokument mogao biti sastavljen od sljedećih podataka:

- Naslov
- Datum
- Autor
- Sadržaj

Kada se indeks kreira, moguće ga je pretraživati slanjem upita. *Lucene* funkcionira na način kreiranja tzv. invertiranog Indeksa. U nastavku su prikazani dokumenti i tablica indeksa koji bi se kreirao za njih.

Document id 1: Lucene

Document id 2: Lucene and Solr

Document id 3: Solr extends Lucene

Tablica 4- Lucene dokument i tablica indeksa

and	2
extends	3
Lucene	1, 2, 3
Solr	2, 3

(Izvor: *Lucene* 4 *Cookbook* - *Edwood Ng*, 2015.)

U prikazanom indeksu je lako pretražiti riječi jer su one sortirane abecedno, a onda će se kao rezultat prikazati svi dokumenti gdje se ta riječ spominje.

Analizatori

Analyzer klase analiziraju tekst i tokeniziraju ga u ključne riječi kako bi omogućile brzo pretraživanje. Analizirati se može samo običan tekst, pa je sve druge formate poput *HTML*-a, *XML*-a, *JSON*-a, *PDF*-a i sl., potrebno najprije pretvoriti u običan tekst. Npr. za tekst:

„Lucene is an information library written in Java“, analyzer će stvoriti sljedeće tokene:

{*Lucene*}, {*is*}, {*an*}, {*information*}, {*library*}, {*written*}, {*in*}, {*Java*}.

Tokenizacija je proces koji ovisi o vrsti *Analyzer* potklase. Svaki analyzer sastoji se od tri osnovne komponente.

- ***Character filter*** – prije samog procesa tokenizacije brišu se svi nepotrebni znakovi poput *HTML* markup-ova, korisnički definiranih uzoraka teksta, pretvaranje specijalnih znakova i sl.
- ***Tokenizer*** – stvaranje tokena od ključnih riječi
- ***Token filter*** – različite manipulacije tokeniziranim podacima.
    - *Stemming* – brisanje prefiksa i sufiksa riječi čiji korijeni ne moraju nužno biti gramatički ispravni;
    - *Stop words filtering* – brisanje riječi poput *a, the, is* i sl.;
    - *Text normalization* – uklanjanje nepotrebnih riječi znakova (više razmaka za redom i sl.);
    - *Synonym expansion* – dodavanje sinonima od ključnih riječi.

Razlike u komponentama *analyzera* u glavnom ovise o vrsti potklase koja nasljeđuje *Analyzer*. Postoji više vrsta *Analyzer* potklasa, a najčešće korištene su sljedeće potklase koje će biti potkrijepljene na primjeru *String-a:*

„Lucene is mainly used for information retrieval and you can read more about it at lucene.apache.org.“

4.1.1.1. WhitespaceAnalyzer

WhitespaceAnalyzer klasa razdvaja *String* prema razmacima u tekstu. Ovo je jedna od najjednostavnijih vrsta klase *Analyzer*.

To izgleda ovako:

[Lucene] [is] [mainly] [used] [for] [information] [retrieval] [and] [you] [can] [read] [more] [about] [it] [at] [lucene.apache.org.]

Posebnu pozornost treba obratiti na *URL* unutar *String*a koji je ostao nepromijenjen.

4.1.1.2. SimpleAnalyzer

SimpleAnalyzer klasa razdvaja *String* slično kao *WhitespaceAnalyzer*, samo što je u ovom slučaju razdvojnik svaki znak koji nije slovo u engleskoj abecedi. Također, tekst se pretvara u mala slova.

[lucene] [is] [mainly] [used] [for] [information] [retrieval] [and] [you] [can] [read] [more] [about] [it] [at] [lucene] [apache] [org]

Posebnu pozornost potrebno je obratiti na *URL* unutar *String*-a koji je u ovom slučaju razdvojen jer znak „.“ nije slovo engleske abecede.

4.1.1.3. StopAnalyzer

StopAnalyzer analizator ima iste funkcije kao i *SimpleAnalyzer* s dodatnom funkcionalnosti uklanjanja riječi s malim značenjem.

Ovaj analizator je također prikladan kada se radi o tekstu na engleskom jeziku ili nekom drugom tekstu koji ne sadrži specijalne znakove, brojeve, šifre i sl. Postoje definirane riječi bez ili s malo značenja (eng. *stopwords*) koje se uklanjaju iz teksta pri tokenizaciji, ali te se riječi mogu prilagoditi prema potrebama.

[lucene] [mainly] [used] [information] [retrieval] [you] [can] [read] [more] [about] [lucene] [apache] [org]

Posebnu pozornost zahtjeva broj tokena koji je znatno manji uklanjanjem riječi bez značenja kada su samostalne.

4.1.1.4. StandardAnalyzer

Ovdje je riječ o tokenizaciji baziranoj na gramatici, pretvaranje riječi u mala slova i uklanjanje riječi bez određenog ili s malim značenjem. Također, uklanjaju se interpunkcije u *Stringu*. Ovaj analizator je jako koristan pri izdvajanju naziva tvrtki, e-mail adresa, šifri, modela i sl.

[lucene] [mainly] [used] [information] [retrieval] [you] [can] [read] [more] [about] [lucene.apache.org]

Može se primijetiti kako je ovaj analizator izgrađen povrh *StopAnalyzer*a. Vrlo je koristan i za generalnu upotrebu jer ne razdvaja šifre, brojeve i sl. na različite tokene.

4.1.1.5. SnowballAnalyzer

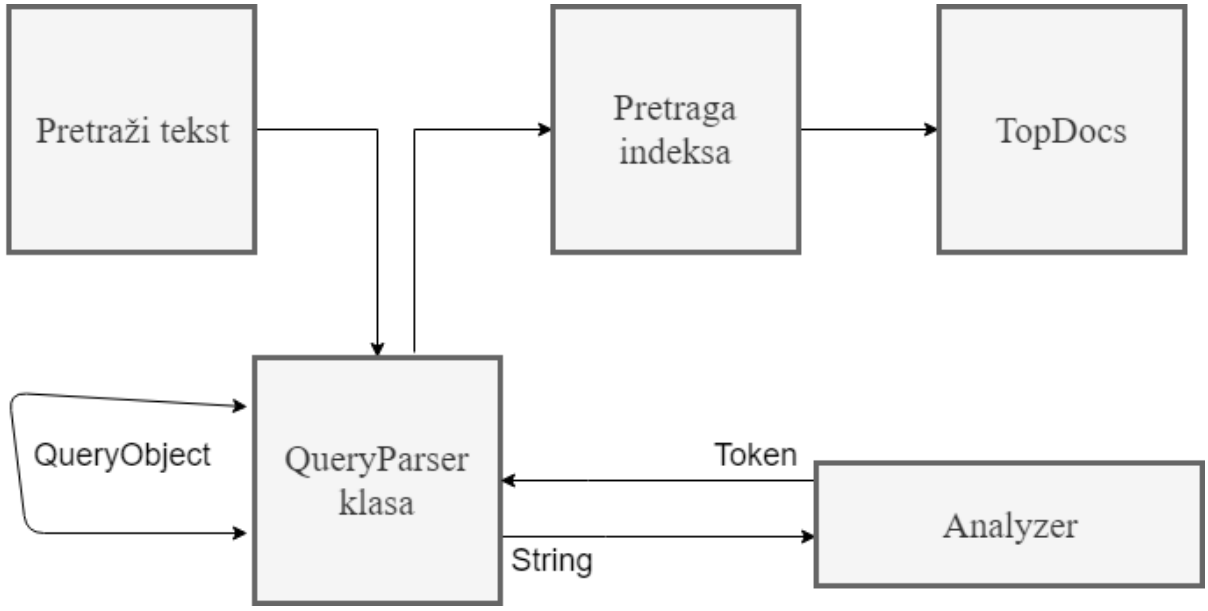
StandardAnalyzer s dodatkom *Stemminga* – uklanjanje sufiksa i prefiksa riječi. Važno je naglasiti da je ovaj analizator jako agresivnog ponašanja i da neke riječi gube svoj smisao i nemaju gramatičko značenje.

[lucen] [is] [main] [use] [for] [inform] [retriev] [and] [you] [can] [read] [more] [about] [it] [at] [lucene.apache.org]

Od *Lucene* 5.0 verzije se *SnowballAnalyzer* smatra zastarjelim i preporučuje se korištenje *Lucene Porter Stemmer* klase, međutim za neke upotrebe *SnowballAnalyzer* je i dalje prikladniji jer se radi o većoj redukciji riječi.

4.2. Pretraga informacija

Pretraga se vrši preko upita s posebnom sintaksom specifičnom za *Lucene* i sustave bazirane na *Lucene*u. Da bi pretraga preko upita bila moguća, potrebno je tekst koji tražimo pretvoriti u *QueryObject*, pri tome pomaže *QueryParser* klasa. Kako funkcionira pretraga?



Slika 11 - Dijagram pretrage informacija (Prema: *Edwood Ng*, 2015.)

Dijagram prikazuje procese koji se odvijaju pri pretrazi teksta. Kada tekst stigne do *QueryParser*.*parse*(*String*) metode, analyzer taj tekst pretvara u set tokena. Zatim se ti isti tokeni pretvaraju u *QueryObject*-e i šalju se u *IndexSearcher* kako bi se pokrenulo pretraživanje. *IndexSearcher* vraća objekt *TopDocs* iz kojega je onda moguće dobiti rezultate i statističke analize, te mjerenja tih rezultata. Rezultati su sortirani prema relevantnosti.

4.3. Sustav pretraživanja u Adobe Experience Manager-u

AEM-ov sustav pretraživanja funkcionira na *Apache Lucene* tehnologiji. Pretraga u *AEM*-u izvodi se nad *Apache Jackrabbit Oak* (u nastavku *Oak*) tehnologijom koja nudi moderan, skalabilan, efikasan i hijerarhijski repozitorij sadržaja. Da bi se sustav pretraživanja unaprijedio, kao i kod tradicionalnih baza podataka, potrebno je dodati indekse. Indeksi se u *Oak*-u kreiraju izradom čvorova i dodavanjem svojstava unutar */oak:indexes* čvora.

AEM alati za indeksiranje

Kao dio *Adobe Consulting Services Commons* alata, nude se dva alata koja pomažu administratorima pri radu s upitima.

1. *Explain Query* – alat koji pomaže administratorima pri razumijevanju upita
2. *Oak Index Manager* – web sučelje za održavanje i pregled postojećih indeksa.

Navedenim alatima može se pristupiti na sljedeći način:

Tools -> Operations -> Dashboard -> Diagnosis na instanci *AEM* poslužitelja.

Indeksi se mogu kreirati i preko *OSGi* konfiguracije kojoj se pristupa preko w*eb* konzole pod nazivom „*Ensure Oak Property Index*“.

Iako *Explain Query* nudi detaljne informacije o upitima, ponekada je potrebno saznati detaljnije informacije o cijelom procesu i o upitu koji se izvršio. Pri tome pomažu dnevnici rada koji su dio *AEM*-a. Način na koji se omogućava logiranje za upite je sljedeći:

- Na adresi *http://serveraddress:port/system/console/slinglog* potrebno je dodati novi logger tipa *DEBUG*. Izvršna datoteka se postavi na *logs/queryDebug.log.* Osim toga svi će *DEBUG* događaji biti zapisani u jednu datoteku. Nakon izvršavanja upita potrebno je sve promijenjene postavke vratiti na *INFO logger*.

Važno je spomenuti kako se konfiguracija indeksa može vidjeti osim u *CRXu* ili izvezenom paketu sadržaja, kao *JSON* na adresi *http://serveraddress:port/oak:index.tidy.-1.json*.

Apache Jackrabbit Oak

Oak se koristi kao pozadinska tehnologija za *AEM*-ov repozitorij sadržaja (*CRX*). *Oak* je nasljednik *Jackrabbit 2* tehnologije koja se koristila u ranijim verzijama *AEM*-a. Dolaskom nove verzije, s korisničke strane se gotovo ništa nije promijenilo. S druge strane, pri implementaciji *AEM*-a koji se bazira na *Oak* tehnologiji, dogodile su se značajne promjene:

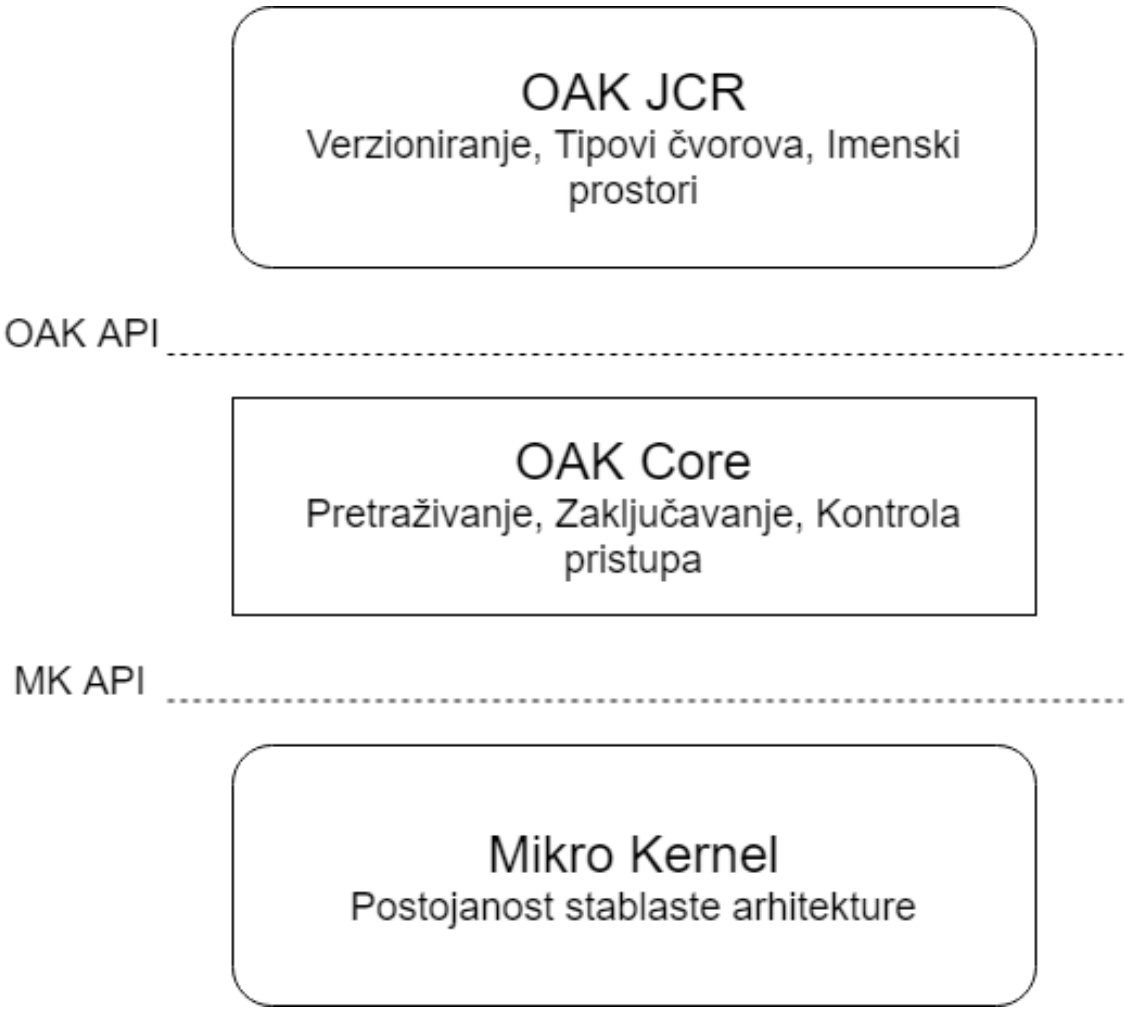
- *Oak* ne stvara indekse automatski., nego se moraju stvarati prilagođeni indeksi kada postoji potreba za tim.
- U *Jackrabbit*-u *2* sesija je uvijek reflektirala posljednje stanje repozitorija, dok kod *Oak*-a sesija predstavlja zadnje stabilno stanje iz vremena kada je sesija dohvaćena. Ovo je nužno zbog *MVC* modela na kojemu je *Oak* tehnologija bazirana.
- Oak ne podržava iste nazive čvorova „djece“.

4.3.1.1. Arhitektura

Oak implementira *JSR-283* specifikaciju čiji su osnovni principi dizajna sljedeći:

- Podrška za velike repozitorije
- Više grupa čvorova za bolju pristupačnost
- Bolje performanse u radu s velikim repozitorijima
- Podrška za više čvorova „djece“
- Podrška za kontrolu pristupa sadržaju

Arhitekturni koncept je sljedeći:



Slika 12 - *Apache Jackrabbit Oak* Arhitekturni koncept

Oak JCR je sloj s glavnom zadaćom pretvaranja semantike u strukturu stabla. Osim toga ovaj sloj implementira *JCR API*.

Oak Core je sloj koji dodaje nekoliko arhitekturnih slojeva:

- Kontrola razina pristupa
- Pretraga i indeksiranje
- Promatranje

Mikro Kernel ili spremište je sloj koji je zadužen za implementiranje strukture stabla, prenosivost i mehanizam grupiranja. Trenutno postoje dvije implementacije spremišta dostupne u svakoj verziji *AEM*-a novijoj od 6.0. To su *Tar Storage* i *MongoDB Storage*.

Tar Spremište koristi *tar* datoteke. Sadržaj se sprema kao dio većih segmenata, a dnevnici prate trenutno stanje repozitorija. Postoji nekoliko osnovnih principa nad kojima je izgrađeno *Tar* Spremište:

- Nepromjenjivi Segmenti (eng. *Immutable Segments*) – svaki segment ima jedinstveni identifikator i nepromjenjiv je što olakšava privremeno spremanje (eng. *caching*) često pristupljenih segmenata;
- Lokalnost – čvorovi sa svojom djecom su obično sadržani unutar istog segmenta što omogućava brzu pretragu s visokom stopom točnosti;
- Kompaktnost – formatiranje podataka je optimizirano tako da se što manje sadržaja mijenja, a što više ostaje u privremenom spremniku.

Mongo Spremište funkcionira na temelju činjenice da cijelo stablo čvorova zadržava u istom obliku, a svaki čvor predstavlja poseban dokument. To omogućava učinkovitu reviziju repozitorija. Za svako ažuriranje sadržaja kreira se nova revizija. Revizija je zapravo *String* objekt koji se sastoji od tri elementa:

- Milisekunde preuzete sa sustava s kojeg je revizija napravljena,
- Brojača koji određuje koliko je revizija kreirano s istog sustava,
- Jedinstvenog identifikatora čvora gdje je kreirana revizija.

Osim revizije, postoji podrška za kreiranjem grana (eng. *branches*) pomoću kojih se može vršiti verzioniranje.

MongoDB spremište dodaje nove podatke svaki put kada se dogodi modifikacija dokumenta. Međutim, stare podatke briše samo ako je to eksplicitno zahtijevano. U suprotnom sprema podatke sve do određenog praga.

Postoji i informacija o tome koje su grupe trenutno aktive, a koje ne.

Oak upiti i indeksiranje

Kako je ranije spomenuto, ako nije postavljen indeks, pretraga će biti spora kao i vrijeme čekanja na uspostavljanje *AEM* instance. U slučaju da *Oak* primi upit za koji ne postoji indeks, javlja se sljedeća greška:

```
*WARN* Traversed 1000 nodes with filter Filter(query=select ...) consider
creating an index or changing the query
```

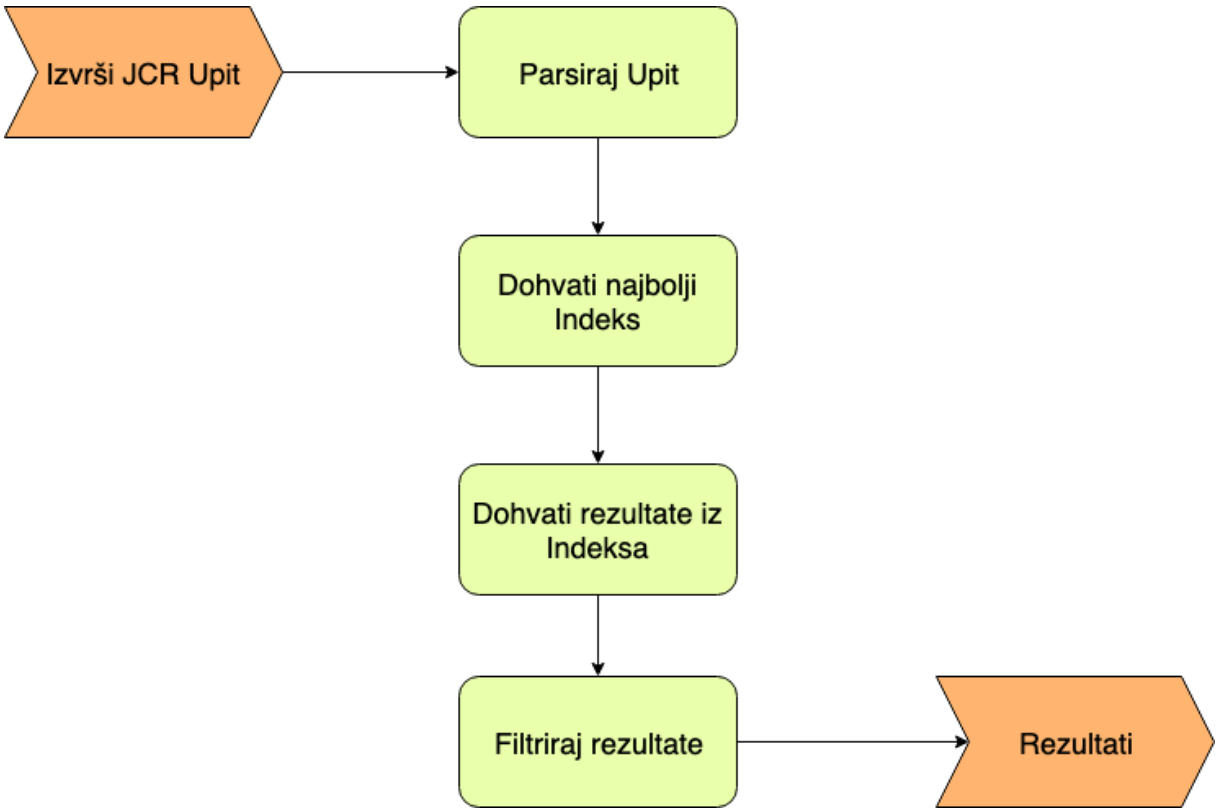
Podržani jezici za pisanje upita su sljedeći:

- *XPath* (preporučeno)
- *SQL*-2
- *SQL* (zastarjelo)
- *JQOM*

Tipovi indeksa i izračun troškova

Apache Oak dopušta kreiranje različitih tipova indeksa u repozitoriju. Indeks svojstava (eng. *Property Indeks*) je jedan od indeksa kojeg je moguće kreirati u repozitoriju. Također, podrška za implementaciju *Apache Lucenea* i *Solra* je zadana, a obje tehnologije podržavaju kreiranje indeksiranja potpunog teksta (eng. *fulltext indexing*). Ako ne postoji odgovarajući indeks, koristi se *obrnuti indeks* (eng. *Traversal Indeks*). Tada se sadržaj ne indeksira, nego se čvorovi izdvajaju jedan po jedan i provjerava se njihovo podudaranje s upitom. Ako postoji više indeksa koji odgovaraju upitu, svaki od indeksa računa cijenu izvršavanja upita, a onda *Oak* odlučuje koji će se indeks koristiti (onaj s najmanjom cijenom).

Cijeli proces ima sljedeći tijek:



Slika 13 - Proces od slanja upita do prikaza rezultata

Upit se najprije parsira u *Abstract Syntax Tree*, a nakon toga se upit provjerava pa pretvara u *SQL-2* sintaksu. *SQL-2* je nativni jezik *Apache Oak*-a. Nakon toga svaki indeks procjenjuje trošak izvršavanja takvog upita. Rezultat indeksa koji daje najmanji trošak se dohvaća, filtrira i provjerava ima li trenutni korisnik pristup rezultatu pretrage i zadovoljava li rezultat u potpunosti upit.

Konfiguracija indeksa

Indeksiranje velikih repozitorija zahtjeva više vremena nego što programeri na prvu očekuju. Isto vrijedi i za ponovno indeksiranje sadržaja (eng. *reindexing*), odnosno proces kada radimo izmjene u indeksu jer ga je potrebno ponovno izvršiti. Sporo vrijeme izvršavanja se posebno ističe kada se radi o *MongoDB* spremištu i indeksiranju potpunog teksta. Tip čvora indeksa mora biti *oak*:*QueryIndexDefinition*.

4.3.1.2. Indeks svojstva (eng. *Property Index*)

Indeks svojstva je koristan za upite koji imaju ograničenja na određena svojstva, a ne odnose se na indekse potpunog teksta. Indeks se postavlja tako da se u *CRXDE*-u kreira novi čvor pod čvorom *oak:index*. Čvor je potrebno nazvati *PropertyIndex* i postaviti mu tip na *oak:QueryIndexDefinition*. Potrebno je na čvor dodati i svojstva u skladu s tim što je cilj pretrage, npr. moguće je postaviti sljedeća dva svojstva:

- *type: property (string),*
- *propertyNames: jcr:uuid (Name).*

U navedenom primjeru indeksirano jcr:uuid svojstvo.

Tip indeksa je postavljen na *PropertyIndex* jer je cilj napraviti Indeks Svojstva.

PropertyNames svojstvo označava koje će se svojstvo spremati u indeks. Ako ovo svojstvo ne postoji, tada će se spremati naziv (eng. *name*) čvora koji se sprema. U navedenom primjeru se sprema *jcr:uuid* koji ima svrhu prikazati jedinstveni identifikator čvora koji je dodan u indeks.

Jedinstvena zastavica (eng. *unique flag*) se postavlja na *true* kako bi se dodala jedinstvenost za svojstva unutar indeksa.

Svojstvo ***declaringNodeTypes*** se postavlja ako želimo da se indeks primjenjuje samo za određeni tip svojstva.

Svojstvo ***reindexing*** pokreće reindeksiranje cijelog sadržaja kada je postavljeno na true vrijednost.

Postoji i Indeks Redoslijeda (eng. *Order Indeks*) koji je proširenje Indeksa Svojstva ali je zastarjelo i *AEM* 6.4 verziji se koristi *Lucene* Indeks Svojstva o kojemu će više govora biti u nastavku.

4.3.1.3. *Lucene* indeks potpunog teksta (eng. *Lucene Fulltext Index*)

Ovaj indeks dostupan je u svim verzijama *AEM*-a novijim od 6.0. Kada je Indeks Potpunog Teksta konfiguriran, tada svi upiti s uvjetom potpunog teksta koriste Indeks Potpunog Teksta, čak i ako postoje drugi uvjeti koji su indeksirani ili određene zabrane na putanjama. Ako navedeni indeks nije konfiguriran, upiti s uvjetom indeksiranja punog teksta neće ispravno raditi. Indeks se ažurira preko pozadinske asinkrone dretve pa će neke pretrage s uvjetom potpunog teksta biti nedostupne na određeno kratko vrijeme dok se ne izvrše pozadinski procesi.

Lucene indeks potpunog teksta se konfigurira na sljedeći način:

- Kreirati čvor *LuceneIndex* podčvorom oak:indeks tipa *oak:QueryIndexDefinition,*

- Dodati sljedeća svojstva: *type: lucene (string), async: async (string),*
- Spremiti promjene.

Postoje sljedeće opcije:

- Tip svojstva koji specificira tip indeksa i u ovom slučaju mora biti postavljen na vrijednost „*lucene*“;
- *Async* svojstvo mora biti postavljeno na *async*. Ovo svojstvo šalje sve podatke o ažuriranju na pozadinsku dretvu o kojoj je bilo govora na početku poglavlja;
- Svojstvo *includePropertyTypes* definira koji će podskup svojstava biti uključen u indeks;
- Svojstvo *excludePropertyNames* definira listu naziva svojstava koja trebaju biti izostavljena iz indeksa;
- Svojstvo *reindex* kada je postavljeno na vrijednost true pokreće re-indeksiranje sadržaja.

4.3.1.4. *Lucene* Indeks Svojstva (eng. *Lucene Property Index*)

Počevši od verzije *Oak*-a 1.0.8, *Lucene* se može koristiti za kreiranje indeksa koji sadrže svojstva. Kako bi pretraga koristila *Lucene* Indeks Svojstva, obavezno je u indeks čvor postaviti svojstvo *fulltextEnabled* na vrijednost false. Primjer može biti sljedeći upit:

```
Select * from [nt:base] where [alias] = '/admin'
```

Da bi za navedeni upit bio kreiran indeks, potrebno je dodati novi čvor pod *oak:index* čvor:

- *Name: LucenePropertyIndex*
- *Type: oak:QueryIndexDefinition*

Kada je čvor kreiran, potrebno je dodati sljedeća svojstva:

- *type: lucene (string)*
- *async: async (string)*
- *fullTextEnabled: false (Boolean)*
- *includePropertyNames: [„alias“] (string)*

U usporedbi sa standardnim Indeksom Svojstva, *Lucene* Indeks Svojstva je uvijek postavljen na *async* stanje, što znači da rezultati indeksa neće uvijek predstavljati najnovije stanje repozitorija.

4.3.1.5. *Lucene* Analizatori

O analizatorima je bilo riječi ranije u ovom poglavlju, a sada će biti objašnjeno kako se analizatori koriste u *AEM*-u.

Analizatori se koriste pri indeksiranju dokumenata, ali i pri pretrazi rezultata upita. U *AEM*-u se analizatori konfiguriraju na čvoru *analyzers* tipa *nt:unstructured* koji je podčvor od čvora indeksa. Za korištenje zadanih analizatora potrebno je sljedeće:

- Locirati indeks na kojemu se želi primijeniti analizator
- Na čvoru željenog indeksa potrebno je kreirati čvor naziva „*default*“ i tipa *nt:unstructured*
- Dodati svojstva:
    - *Name: class*
    - *Type: String*
    - *Value: org.apache.lucene.analysis.standard.StandardAnalyzer* (ako se želi koristiti Standardni Analizator)
- Ako je potrebno, moguće je dodati datoteku s riječima koje je potrebno filtrirati analizatorom. Tada se kreira novi čvor naziva default i tipa nt:unstructured sa sljedećim svojstvima:
    - *Name: stopwords*
    - *Type: nt:file*

Od analizatora se može stvoriti kompozicija tako da se specificiraju tokenizeri, token filteri i filteri riječi. Navedeno je moguće postići kreiranjem čvorova „djece“ na čvoru analizatora koji predstavljaju filtere, tokenizere i sl.

5. Apache Solr

Apache Solr (u nastavku *Solr*) je projekt *Apache Lucenea* koji se ponaša kao poslužitelj za pretragu (eng. *Enterprise Search Server*) i napisan u *Javi*.

Solr je nastao 2004, a napisala ga je tvrtka *CNET Networks* kao interni projekt koji je poslužio kao proširenje sustava pretraživanja službene stranice spomenute tvrtke. Godine 2007. *CNET Networks* odlučue objaviti izvorni kod *Solr*a koji poklanja tvrtki *Apache Software Foundation*. Nakon toga *Apache* nastavlja s objavom sljedećih *release* verzija softvera koji postaje jako popularan s obzirom na svakodnevni rast podataka prikazan u uvodu rada.

Povijest *Apache Solr*a

Značajne promjene *Solr*a kroz povijest:

Tablica 5 - Povijest Apache Solra

Godina	Verzija	Promjene
2004	*Solr*	Nastao kao interni projekt *CNET Networks* firme
2006	*Apache Solr*	*Apache* preuzima izvorni kod i *Solr* ulazi u period inkubacije
2007		*Solr* prelazi iz faze inkubacije u samostalni top-level projekt (*TLP*)
2008	*Apache Solr 1.3*	Prva službena (eng. *release*) verzija s mogućnošću distribuirane pretrage i poboljšanih performansi
2009	*Apache Solr 1.4*	Nastaje *Lucidwork* – tvrtka koja nudi edukaciju i podršku za korištenje *Solr*a. Uvode se poboljšanja pri indeksiranju, pretrazi, obradi više vrsta dokumenata, uvođenje klastera i sl. Nastaje i napredna navigacija (faceted navigation) pri pretrazi
2010		Nastaje spajanje *Lucenea* i *Solr*a. *Solr* postaje pod-projekt *Lucene*a
2011	*Apache Solr 3.1*	Mijenja se verzija *scheme* kako bi se podudarala s verzijom *Lucene*a
2012	*Apache Solr 4.0*	*SolrCloud* funkcionalnost
2015	*Apache Solr 5.0*	Prvo službeno izdanje gdje je *Solr* samostalna aplikacija. Prestaje podrška za razvoj *Solr*a kao *war* datoteke. Nastaje okvir za autentikaciju i autorizaciju
2016	*Apache Solr 6.0*	Novi *JDBC* upravljački program, podrška za *StreamExpression* izraze, podrška za paralelne *SQL* upite preko *SolrCloud* kolekcije
2017	*Apache Solr 7.0*	Podrška za *Math* alat, auto-skalabilnost

2019	*Apache Solr 8.0*	Mogućnost osluškivanja i slanja *HTTP* zahtjeva. *Admin UI* prijava podržava *BasicAuth* i *Kerberos*

Funkcionalnosti *Solr*a

Solr podržava različite načine komunikacije i vrste datoteka poput *HTTP*-a, *XML*-a i *JSON*-a. U potpunosti je baziran na *Apache Lucene* tehnologiji sa značajnim razlikama poput one da je *Lucene* samo *Java* biblioteka, a ne poslužitelj ili *web* pretraživač te nema konfiguracijske datoteke. Više o razlikama će biti objašnjeno u nastavku poglavlja. Osim pružanja rezultata pretrage, Solr sadrži sljedeće funkcionalnosti:

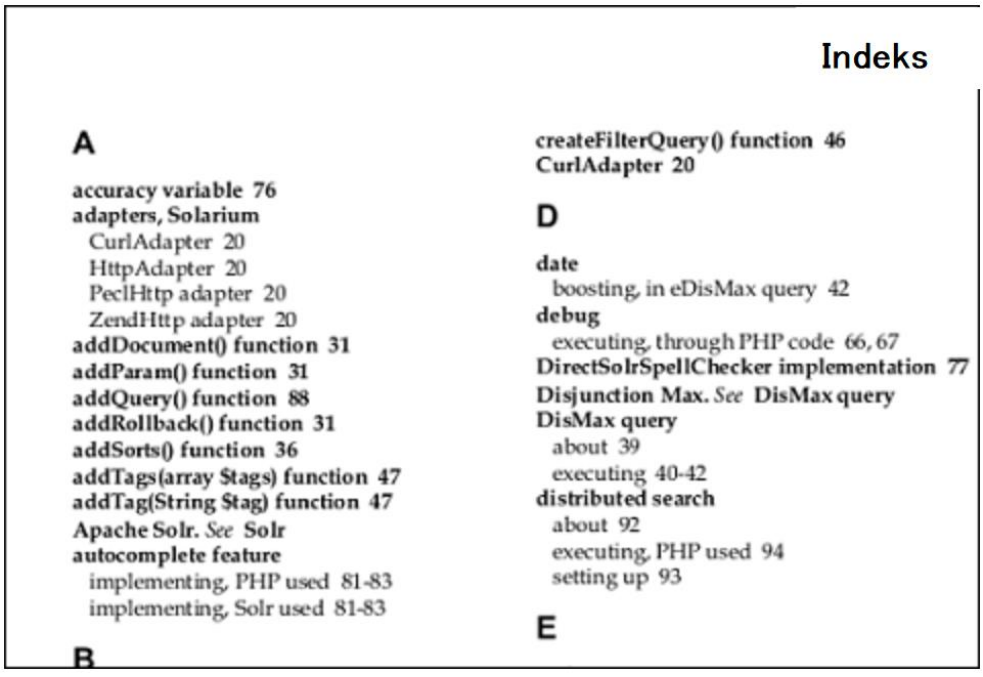
- Poslužitelj koji komunicira preko *HTTP* protokola s više različitih formata poput *XML*-a i *JSON*-a
- Konfiguracijske datoteke za indekse
- Više tipova privremene (eng. *cache*) memorije za poboljšanje performansi
- Interaktivno *web* sučelje
    - Statistika i performanse pretrage
    - Statistika indeksa
    - Dijagnostički alati za debugiranje analize teksta
    - I ostalo
- *Faceting* (grupiranje, filteri i sl.)
- Parser upita *eDisMax*
- Distribuirana podrška za pretragu, replikaciju indeksa, skaliranje i sl.
- Konfiguracija klastera pomoću *ZooKeeper* alata
- *Solaritis* – prototipiranje i demonstracija funkcionalnosti *Solr*a

Povezanost *Solr*a i *Lucene*a

Od verzije 3.0 ustanovljena je velika povezanost *Solr* i *Lucene* tehnologija. Repozitorij izvornog koda, programeri, mailing liste i release verzije su zajedničke. Mnogi *Solr* smatraju najboljom verzijom *Lucene*a. To se postiže jednostavnošću kojom Solr koristi Lucene preko konfiguracijskih datoteka, *HTTP* parametara i sl.

Razlike između navedenih tehnologija su prvenstveno u razini apstrakcije gdje *Lucene* pruža više slobode pri potrebi za prilagođenim rješenjima i za čije je korištenje potrebno poznavanje *Java* programskog jezika, dok *Solr* može jednostavno koristiti i netko tko nije upoznat s gore spomenutim. Kao što je spomenuto, *Lucene* je biblioteka, dok *Solr* to nadizali na mnoge načine, a u prvom redu činjenicom da nudi cijelu infrastrukturu za pretraživanje poput poslužitelja, brojnih analiza, statistike i gotovih funkcionalnosti spremnih za korištenje.

Kako je *Solr* baziran na *Lucene*u, podrazumijeva se da u pozadini djeluju svi principi isti kao za *Lucene*. Primjer toga je invertirani indeks. Invertirani indeks je najlakše poistovjetiti sa zadnjim stranicama knjige gdje su zapisani osnovni termini i stranice na kojima se ti termini pojavljuju. Tako se i u invertiranom indeksu mogu lako pronaći dokumenti na kojima se pojavljuje termin koji se pretražuje.



Indeks

A

accuracy variable 76
adapters, Solarium
  CurlAdapter 20
  HttpAdapter 20
  PeclHttp adapter 20
  ZendHttp adapter 20
addDocument() function 31
addParam() function 31
addQuery() function 88
addRollback() function 31
addSorts() function 36
addTags(array $tags) function 47
addTag(String $tag) function 47
Apache Solr. *See* Solr
autocomplete feature
  implementing, PHP used 81-83
  implementing, Solr used 81-83

B

createFilterQuery() function 46
CurlAdapter 20

D

date
  boosting, in eDisMax query 42
debug
  executing, through PHP code 66, 67
DirectSolrSpellChecker implementation 77
Disjunction Max. *See* DisMax query
DisMax query
  about 39
  executing 40-42
distributed search
  about 92
  executing, PHP used 94
  setting up 93

E

Slika 14 - Primjer Indeksa (Prema*: Kumar J. - Apache Solr Search Patterns*, 2015.)

Detalji invertiranog indeksa su objašnjeni u poglavlju *Apache Lucene*, kao i ostale funkcionalnosti. Za sve pojedinosti o radu *Solr*a u pozadini, moguće se osvrnuti na poglavlje *Apache Lucene*.

5.1. Razlika između *Solra* i baza podataka

Početnici u radu s *Lucene*om i *Solr*om često se pitaju koja je razlika između *Solr*a i baze podataka. Također, popularno je pitanje postoji li potreba za bazom podataka ako koristimo *Solr*.

Najpoznatija vrsta baza podataka danas su relacijske baze podataka. Njihova popularnost temelji se na modelu (organizacijskoj strukturi) podataka. Model se bazira na vezama između entiteta koji su identificirani preko ključeva, a podaci se mogu dohvatiti slanjem upita. Spomenuti pristup se čini prilagodljiv jer je moguće dohvatiti gotovo svaku informaciju iz tog modela pisanjem jednog upita.

Taj pristup je prilagodljiv, ali ne i skalabilan. Poteškoće nastaju kada je u pitanju više milijuna podataka i dokumenata, što je uobičajeno za bilo koju aplikaciju koja se bazira na pretrazi. Zahtjevi za niskom latencijom pri odgovoru od baze podataka su visoki. Tu se pojavljuje *Lucene*ov model podataka orijentiran prema dokumentima, koji su analogni jednoj tablici baze podataka.

MongoDB je baza podataka koja je također orijentirana na dokument. Sličnosti *Solr*a i *MongoDB*-a su sljedeće:

- Bazirani na dokumentima;
- Ne koriste relacijske modele podataka;
- Unaprjeđuju funkcionalnosti upita;
- *Solr* implementira pretragu baziranu na *Lucene*u, a MongoDB bazira pretragu na tekstu;
- Ocjenjuju relevantnost pretrage;
- Mogu koristiti analizatore;
- Mogu pojačati važnost određenog polja (termina).

Dokumenti u *MongoDB*-u su u hijerarhiji što otežava posao pri pisanju upita, dobivanju relevantnih rezultata i brzine. *Luceneovi* dokumenti su jednostavna tablica koja podržava više vrijednosti po jednom polju. S druge strane, problem kod *Lucene*a i *Solr*a predstavljaju ujedinjeni upiti (eng*. join queries*), pa se u pravilu koriste što manje kako bi se iskoristila skalabilnost koju tehnologije pružaju.

Već površnim pregledom *Solr*a lako je za uočiti kako postoje mnoge funkcionalnosti koje relacijske baze podataka nemaju. Neke od njih su sortiranje rezultata prema točnosti, isticanje rezultata, pregled sintakse pri pisanju upita, automatska nadopuna upita pri pisanju i sl. Jako je važno istaknuti i *faceting* koji je moguće postići s bazama podataka, ali na mnogo teži način.

Kao što je spomenuto na početku poglavlja, na prvi pogled se čini kao da je podatke moguće spremiti u *Solr* , a odgovore na upite dobiti brzo i efikasno. To nas navodi na zaključak da se *Solr* može koristiti bez baze podataka. Međutim to nije točno, preporuka je da se *Solr* uvijek koristi kao nadopuna bazama podataka. Postoje jasni razlozi za to. Baze podataka daju bolje rezultate pri *ACID* (eng. *Atomicy, Consistency, Isolation, Durability*) transakcijama, efikasnosti pri dodavanju i ažuriranju podataka, mijenjaju strukture tablica, kontrolu pristupa više korisnika i često nema potrebe za integracijom s drugim alatima. Najvažnija od navedenih prednosti baza podataka bila bi činjenica da je model prilagodljiv.

5.2. Uvod u rad sa *Solrom*

Kako bi radili sa *Solr*om, potrebno ga je preuzeti i raspakirati. U raspakiranom direktoriju se nalaze datoteke sa sljedećim značenjima:

- *contrib* – direktorij ekstenzija Solra
  - *analysis-extras* – u ovom direktoriju se nalazi nekoliko analizatora teksta koji imaju velike zavisnosti. Osim toga tu su nekolike *ICU* (*International Components for Unicode*) – *unicode* klase za podršku više jezika s kineskim i poljskim *stemming*-analizatorom.
  - *clustering* – u ovom direktoriju nalazi se alat za kreiranje klastera rezultata pretrage.
  - *dataimporthandler* – vrlo popularan modul koji uvozi podatke u *Solr* iz baze podataka ili nekog drugog izvora.
  - *extraction* – modul za integraciju s *Apache Tika* alatom – okvir za izdvajanje teksta iz poznatih formata datoteka.
  - *langid* – modul za prepoznavanje jezika podataka prije nego što su indeksirani.
  - *map-reduce* – Alati za rad s *Handoop Map*-Resursima – distribuiranje velikih količina podataka na više uređaja i klastera.
  - *morphlines-core* – okvir za rad s dokumentima u *Solr*u.
  - *uima* – biblioteka za integraciju s *Apache UIMA* – okvirom za izdvajanje metapodataka iz teksta.

    - *velocity* – u ovom direktoriju će se nalaziti jednostavni *UI* za pretragu baziran na *Velocity* jeziku za kreiranje okvira.
- *dist* – ovdje se nalaze *Solr core* i *contrib JAR* datoteke. *Core JAR* datoteka se koristi za integraciju *Solr*a u aplikaciju. *SolrJ JAE* i /*solrj-lib* su potrebni za izgradnju klijentskih aplikacija za *Solr* baziranih na *Javi*.
- *docs* – dokumentacija i „*Javadocs*“ za sadržaj vezan uz javnu *Solr* stranicu, upute za rad sa *Solr*om i *Solr API*.
- *example* – ogledni primjer
    - *example-DIH – DataImportHandler* konfiguracijske datoteke za primjer postavki *Solr*a.
    - *exampledocs* – ogledni dokumenti koji će biti indeksirani u oglednom primjeru. Osim toga tu je i *post.jar* program koji šalje dokumente na *Solr*.
- *server* – svi dokumenti potrebni da se *Solr* pokrene kao poslužitelj se nalaze u ovom direktoriju.
    - *context – Jetty Web* aplikacijska konfiguracija za uspostavljanje *Solr*a.
    - *etc* – konfiguracija *Jetty* poslužitelja. Ovom direktoriju se pristupa kada se želi primijeniti zadani port (8983) na 80 – zadani *HTTP* port.
    - *logs* – dnevnici rada uvijek pružaju korisne informacije i nalaze se u ovom direktoriju
    - *resources* – konfiguracijska datoteka za *Log4j* – upravljanje informacijama koje se prikazuju u logovima
    - *solr* – konfiguracijske datoteka za pokretanje *Solr*a. *solr.xml* odnosi se na generalnu konfiguraciju *Solr*a kao i *zoo.cfg* koji je potreban za rad *SolrCloud*a. U pod direktoriju /*congifsets* nalaze se ogledni primjeri konfiguracija.
    - *webapps* – iz ovog direktorija Jetty isporučuje *Solr*. Ovdje se nalazi *Solr* war datoteka koja sadrži kompilirani *Solr*ov kod i sve jar datoteke koje predstavljaju njegove ovisnosti.
    - *solr-webapp* – ovo je direktorij u koji *Jetty* isporučuje neraspakiranu *WAR* datoteku.

Kako bi pokrenuti Solr, potrebno je iz bin direktorija pokrenuti sljedeću skriptu:

```
./solr start -e techproducts
```

Tako bi bio pokrenut jedan od oglednih primjera *Solr*a – *techproducts*. Tada se *techproducts* kolekcija kreira preko *API* poziva, a nakon toga se događa preusmjeravanje na *URL*:

```
http://localhost:8983/solr
```

Pritom se prikazuje početna stranica za rad sa *Solr*-om na kojoj su intuitivnim sučeljem prikazane sve mogućnosti i analize koje nudi *Solr*. Za zaustavljanje *Solr*a pokreće se sljedeća naredba:

```
./solr.stop
```

Rad sa *Solr*om u *AEM*-u

Za rad sa *Solr*om u *AEM*-u potrebno je sljedeće:

- Preuzimanje i raspakiranje *Solr* paketa
- Kreiranje *Solr* dijelova (eng. *shards*) kao direktorija unutar direktorija u kojemu je *Solr* paket raspakiran
  - `<solrunpackdirectory>\aemsolr1\node1`
  - `<solrunpackdirectory>\aemsolr2\node2`
- Lociranje ogledne instance *Solr*a
- Kopiranje sljedećih direktorija u kreirane dijelove (*aemsolr1\node1* i *aemsolr2\node2*)
  - context
  - etc
  - lib
  - resources
  - scripts
  - solr-webapp
  - webapps
  - start.jar
- Kreiranje novog direktorija „*cfg*“ u svakom od dijelova
- Spremanje *Solr* i *Zookeeper* konfiguracijskih datoteka u „*cfg*“ direktorij
- Prvi dio *Solr*a se pokreće *Zookeeperom* pokretanjem sljedeće komande u *aemsolr1\node1* direktoriju
  - `java -Xmx2g -Dbootstrap_confdir=./cfg/oak/conf -Dcollection.configName=myconf -DzkRun -DnumShards=2 -jar start.jar`
- Drugi dio *Solr*a se pokreće iz *aemsolr2\node2* direktorija sljedećom komandom:
  - `java -Xmx2g -Djetty.port=7574 -DzkHost=localhost:9983 -jar start.jar`

- Nakon što su oba dva dijela *Solr*a pokrenuta, potrebno je testirati povezanost pristupanjem *Solr* sučelja na sljedećoj adresi:

`http://localhost:8983/solr/#/`

- Pokrenuti *AEM* instancu i *configManager* u *Web* konzoli
- Na konfiguraciji ***Oak Solr remote server configuration*** postaviti sljedeće:
    - `Solr HTTP URL: http://localhost:8983/solr/`
- Izabrati „*Remote Solr“* opciju u listi ispod ***Oak Solr*** poslužitelj opcije
- Prijaviti se u *CRXDE* sa administratorskim računom
- Kreirati čvor naziva solrIndex pod *oak:index* čvorom i postaviti sljedeće atribute:
    - type:solr (tip String)
    - async:async (tip String)
    - reindex:true (tip Boolean)
- Spremiti promjene.

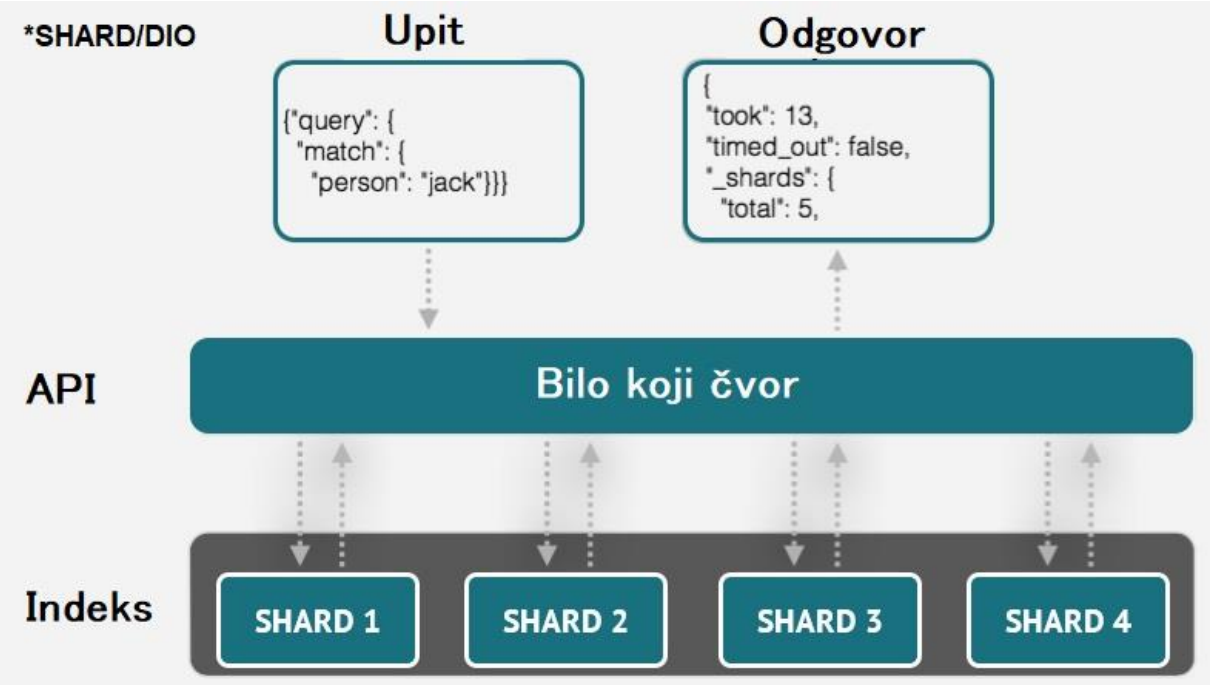
6. Elasticsearch

Elasticsearch je, poput *Solr*a, projekt otvorenog koda razvijen povrh *Apache Lucene*a. Projekt je napisan u *Javi*, a *Lucene* se koristi interno za indeksiranje sadržaja i pretragu. Ideja *Elasticsearcha* je ponuditi mogućnosti *Lucene*a korisnicima koji ne moraju nužno imati znanja u programiranju. To je postignuto redukcijom apstrakcije intuitivnim korisničkim sučeljem koje prikriva kompleksnosti *Lucene*a.

Elasticsearch je:

- Distribuirani sustav koji sprema dokumente u realnom vremenu gdje je svako polje indeksirano i moguće ga je pretražiti;
- Distribuirani sustav za pretragu koji pruža analitiku u realnom vremenu;
- Sustav sa sposobnosti skaliranja na stotine poslužitelja i petabajta strukturiranih i nestrukturiranih podataka.

Jednostavnost *Elasticsearcha* sadrži se u jednostavnom *REST API*-ju. *Elasticsearch* se tako može koristiti u bilo kojem programskom jeziku, sve što je potrebno je poslati podatke preko *HTTP*-a u *JSON* formatu kako bi se indeksirali, pretraživali i održavali u *Elasticsearch* klasteru. Osim JSON formata, podržan je i YAML koji se uglavnom koristi za pisanje konfiguracijskih datoteka.



Slika 15 - *Elasticsearch* tijek pretrage

6.1. Povijest nastanka *Elasticsearcha*

Postoji zanimljiva priča o nastanku *Elasticsearcha*. Shay Banon je bio nezaposleni programer koji se odselio u London jer je njegova supruga studirala kulinarstvo. Dok je tražio posao, bio je okupiran s ranijim verzijama *Lucene*a kako bi za svoju suprugu napravio aplikaciju za pretragu recepata za jela.

Budući da je rad direktno s *Lucene*om zahtjevan, Shay je odlučio napraviti novi apstrakcijski level kako bi *Java* programeri lakše mogli dodati pretragu svojim aplikacijama. Objavio je to kao svoj prvi projekt otvorenog koda – *Compass*.

Shay se nakon toga zaposlio u tvrtki koja je trebala visoke performance pretrage u realnom vremenu. Pretraga je morala biti distribuirana pa je odlučio doraditi biblioteke *Compass*a i pretvoriti ih u odvojeni poslužitelj nazvan *Elasticsearch*.

Godine 2010. govori se o prvom službenom izdanju *Elasticsearcha*. Otada je *Elasticsearch* jedan od najpopularnijih repozitorija na *GitHubu* s preko 300 suradnika. Nakon navedenih događaja, cijela tvrtka preusmjerila je svoje poslovanje u razvoj *Elasticsearch*a kako bi omogućila nove funkcionalnosti i komercijalnu podršku. Jedini uvjet je bio da *Elasticsearch* zauvijek ostane projekt otvorenog koda koji je dostupan svima. *Shay*-ova žena još uvijek čeka svoju aplikaciju za pretragu recepata.

6.2. Povezanost *Elasticsearch*a i *Apache Lucene*a

Iako je *Elasticsearch* razvijen povrh *Lucene*a, postoje određene razlike u razvoju pomoću te dvije tehnologije. Glavne razlike između *Elasticsearcha* i *Lucene*a su sljedeće:

- *Elasticsearch* je poslužitelj, a *Lucene* biblioteka
- *Elasticsearch* je razvijen povrh *Lucene*a i omogućava *REST API* baziran na *JSON*-u koji omogućava rad s funkcionalnostima *Lucene*a
- *Elasticsearch* je distribuirani sustav što *Lucene* nije, niti je tako razvijen
- *Elasticsearch* pruža funkcionalnosti koje podržavaju postojeće *Lucene* funkcionalnosti, ali sami *Lucene* ih ne sadrži. To su rad s dretvama, redovima, čvorovima, klasterima, nadzorom *API*-ja, nadzorom podataka, upravljanjem klasterima i sl.

- Budući da je *Elasticsearch* distribuirani sustav, omogućava repliciranje podataka, što znači da više kopija podataka može postojati u jednom klasteru. To omogućava visoku dostupnost podataka
- *Elasticsearch* nudi *JSON* sučelje za čitanje i pisanje upita na *Lucene* omogućava pisanje kompleksnih upita bez poznavanja *Lucene* sintakse
- *Elasticsearch* korisnike oslobađa sheme, tj polja (parovi naziva i vrijednosti) ne moraju biti unaprijed definirani. Kada se podaci indeksiraju, *Elasticsearch* automatski može napraviti shemu

6.3. Uvod u rad s *Elasticsearchom*

Vrlo je lako uočiti da je *Elasticsearch* jako sličan *Solr*u prema navedenim opisima tehnologija. Tehnologije jesu slične, između ostalog baziraju se na istoj tehnologiji – *Apache Lucene*. Međutim postoje određene razlike koje će biti specificirane u sljedećem poglavlju. Budući da je opis funkcionalnosti sličan onima iz *Solr*a, kako isti tekst ne bi bio ponavljan, autor je odlučio prikazati rad *Elasticsearch*a kroz primjer kako bi se dobio bolji uvid u smisao takve tehnologije.

Radi se o primjeru sa stranicom o blogovima. Jedna stranica može imati mnogo objavljenih blogova, te je potrebno omogućiti njihovu efikasnu pretragu. Prva ideja bila bi implementirati pretragu ključnih riječi. Ako korisnik traži riječ „izbori“, potrebno je prikazati sve blogove koji sadrže navedenu riječ.

Elasticsearch će omogućiti ovu jednostavnu pretragu, međutim ona mora biti robusna, relevantna, vraćati rezultate brzo i efikasno. Također, ponekada korisnik ne zna što točno traži pa je potrebno uključiti pomoć pri pretrazi, uočavanje grešaka pri pisanju, omogućavanje prijedloga pretrage, prikaz rezultata u kategorijama i sl.

Ako na stranici postoji velik broj blogova koje sadrže riječ izbor, pretraga bi bila spora. Taj problem *Elasticsearch* rješava iskorištavanjem brzine *Apache Lucene*a koji indeksira sve podatke. O indeksima i indeksiranju je bilo riječi ranije, a kod *Elasticsearcha* je princip isti. Radi se o invertiranom indeksu. Invertiranim indeksom moguće je riječ „izbori“ pronaći prema oznaci, dok bi obična *SQL* pretraga uspoređivala svaku riječ svakog bloga s riječi „izbori“. Osim brzine, invertirani indeks nudi i relevantnost. Pretragom određene oznake u indeksu se ne dobije samo mjesto (blog) u kojemu se ta riječ pojavljuje, nego i broj puta koliko se ta riječ pojavljuje na tom mjestu. Ovo je jako važna funkcionalnost, jer pojavljivanje određene riječi u velikom broju dokumenata, govori o niskoj relevantnosti te riječi. Tako za pretragu izraza „*Elasticsearch in Action*“, riječ „in“ pojavljuje se mnogo puta pa nije relevantna, a ako dokument sadrži riječ „*Elasticsearch*“, zajedno s npr. 100 drugih dokumenata, tada je taj dokument dobar kandidat za konačan rezultat jer se relevantnost povećala.

Sljedeći veliki izazov je omogućiti korisnicima da se blogovi u kojima riječ „izbori“ ima značajnu ulogu (npr. naslov) prikazuju prije drugih blogova. *Elasticsearch* taj problem rješava dodjeljivanjem ocjene relevantnosti svakom od dokumenata.

Slika 16 - Pojava jednog termina pretrage više puta obično rangira dokument više u hijerarhiji

Zadano ponašanje *Elasticsearch*a je da ocjenu dokumenta računa na bazi *TF-IDF* (eng*. term frequency-inverse document frequency*) algoritma.

- Učestalost termina – što se više puta termin pojavi u dokumentu, veća je ocjena
- Učestalost inverznog indeksa – Težina određenog termina je veća ako termin nije uobičajen u drugim dokumentima

Osim navedenog algoritma, *Elasticsearch* dopušta nadograđivanje procesa računanja ocjene pa tako korisnik može postaviti da mu je u pretrazi pronalazak termina u naslovu bloga važniji od pronalaska termina u tekstu bloga. Također, moguće je primijeniti pravilo kako pronalazak cijelog termina znači više nego pronalazak samo dijela tog termina ili kako pojava termina u blogu koji su korisnici više puta označili sa „sviđa mi se“.

Korisne funkcionalnosti bile bi i dodavanje funkcionalnosti koja korisniku oprašta greške pri pisanju preko tzv. *fuzzy* upita, prikaz statistike koja pomaže korisnicima koji nisu sigurni što bi pretraživali, pomoć pri pisanju s prijedlozima termina i sl.

Iako je ustanovljeno da je *Elasticsearch* dobar način za brzo i kvalitetno pronalaženje rezultata pretrage, ipak se radi samo o sustavu pretraživanja koji ne bi trebao postojati samostalno. Kao i svakom drugom spremištu podataka, potrebno je pronaći način kako te podatke unijeti u *Elasticsearch* te kako korisnicima napraviti sučelje na kojemu će pretraživati sadržaj i dobivati rezultate. Prema tome, razlikujemo tri tipična scenarija u kojima se koristi *Elasticsearch*:

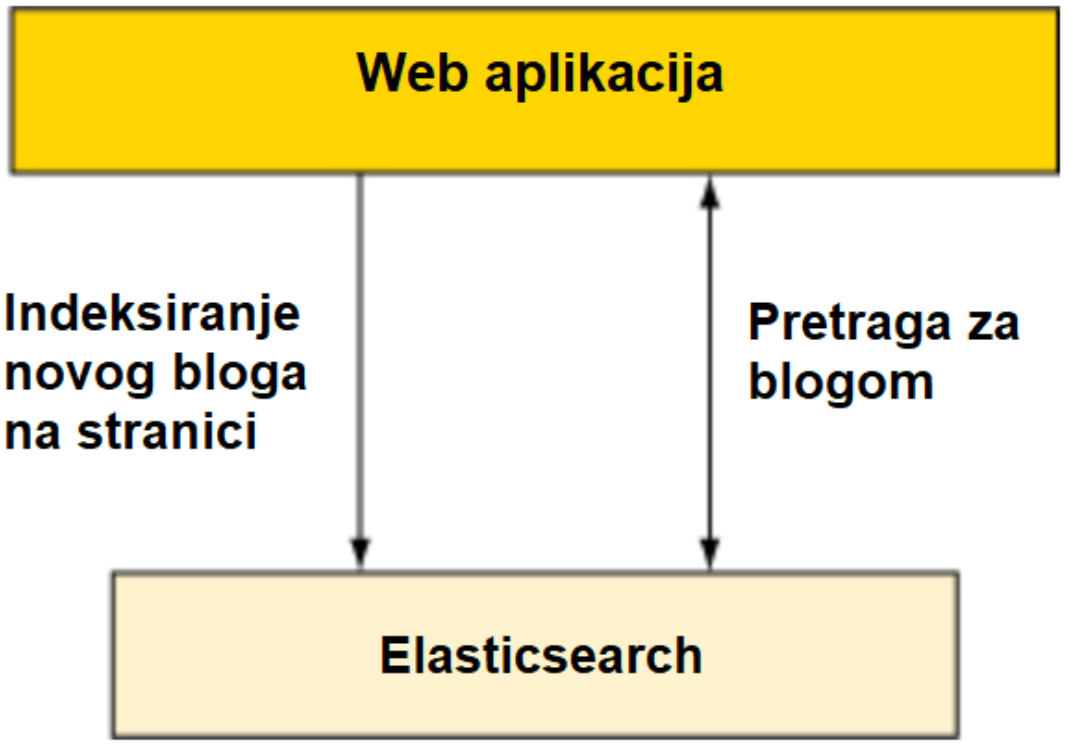
- ***Elasticsearch* kao primarni dio (osnova) aplikacije** – primjer može biti spomenuta stranica s blogovima i pretragom među njima. *Elasticsearch* se može koristiti kao spremište podataka i kao servis za slanje upita.
- **Dodavanje *Elasticsearch*a na postojeći sustav pretraživanja** – postojeći sustav se proširio i pretraga ne može funkcionirati kao ranije. Postoji više vrsta dizajna primjenjivih u ovakvim situacijama.
- ***Korištenje Elasticsearc-a uz dodatak gotovih rješenja iz Elasticsearch zajednice*** – postoji cijeli ekosistem izgrađen kao podrška *Elasticsearch*u kako bi se smanjila potreba za korisničkim razvojem aplikacija podrške.

Elasticsearch kao primarni dio (osnova) aplikacije

Tradicionalni sustavi pretraživanja nisu se pokazali dobrima kao jedino i osnovno spremište podataka u aplikacijama. To se dogodilo jer nisu omogućavali izdržljivost i pregled statistike nad spremljenim podacima.

Elasticsearch nije tradicionalni, nego moderni sustav pretraživanja koji nudi dugotrajnu očuvanost podataka, statistiku i druge funkcionalnosti. Iz tog razloga, pri planiranju arhitekturalnog dizajna aplikacije, dobro je razmisliti o opciji postavljanja *Elasticsearch*a kao jedinog spremišta podataka. Naravno, to neće biti dobra arhitektura u svim situacijama, posebno ako se radi o aplikacijama koje rade mnogo ažuriranja nad podacima koje je svaki put potrebno ponovno indeksirati. Tada je *Elasticsearch* koristan u kombinaciji s drugim spremištima podataka. Također, pri korištenju *Elasticsearch*a kao jedinog spremišta podataka, kao i drugih *NoSQL* spremišta podataka, potrebno je voditi posebnu brigu o pričuvama (eng. backup) podataka.

Na primjeru stranice s blogovima, *Elasticsearch* kao primarni dio aplikacije bi izgledao ovako:



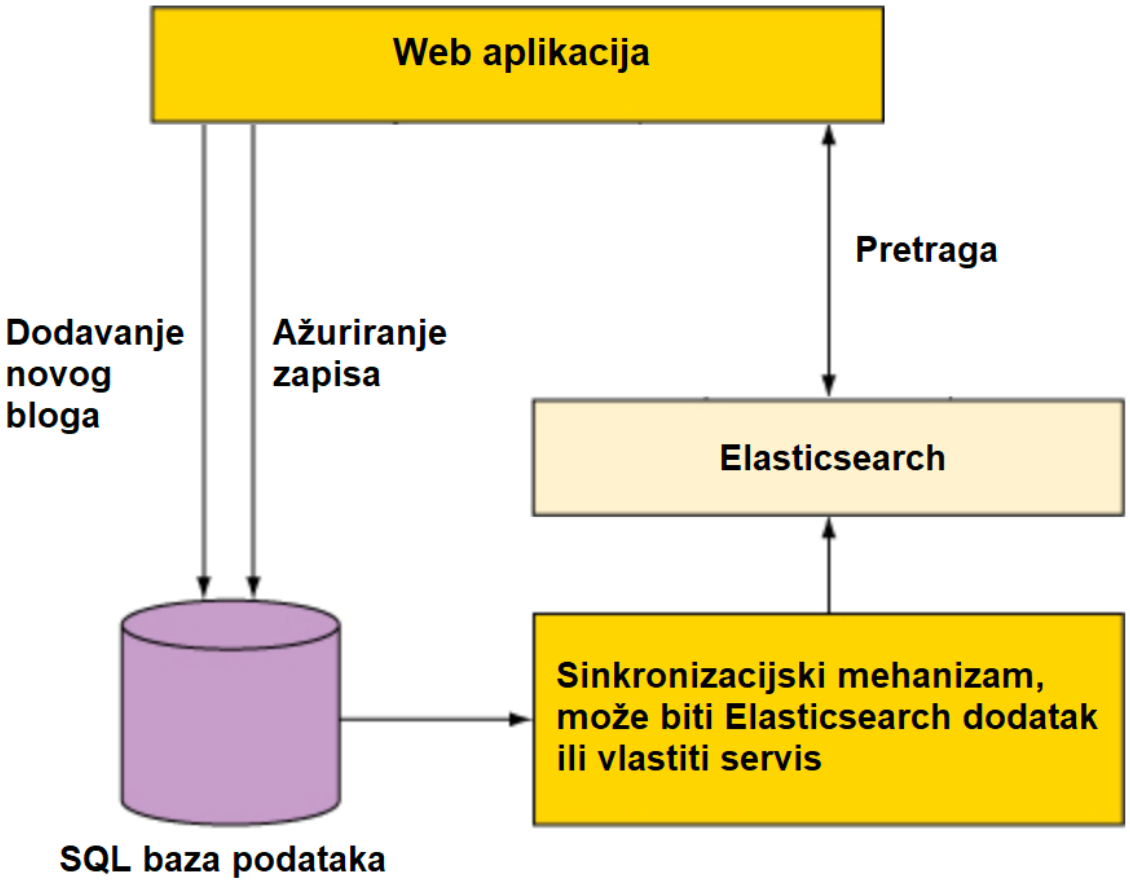
Slika 17 - *Elasticsearch* kao primarni dio aplikacije (Izvor: *Hinman M. L., Gheorghe R., Russo R. – Elasticsearch in Action*, 2015.)

Važno je replicirati podatke na više različitih poslužitelja kako ne bi ovisili samo o funkcioniranju jednog poslužitelja. *Elasticsearch* može unaprijediti gotovo svaku aplikaciju, ali potrebno je razmisliti je li arhitekturalno potrebno još jedno spremište podataka ili *Elasticsearch* može biti jedini. Više o tome u tekstu koji slijedi.

Dodavanje *Elasticsearch*a na postojeći sustav pretraživanja

Kao što je naglašeno, *Elasticsearch* ponekad nije dovoljan kao samostalni sustav spremanja podataka. To je u prvom redu zato što ne nudi mogućnosti transakcija i veza među entitetima kao baze podataka. Još jedan od mogućih čestih razloga je sprječavanje posljedica od naslijeđenog projekta koji je arhitekturalno loše koncipiran. Ako je projekt uvelike ovisan o sustavu pretraživanja, a spremište podataka mu je relacijska baza, problem je mijenjati cijeli dizajn kako bi se mijenjalo spremište podataka u *Elasticsearch*. Takve situacije se mogu spasiti dodavanjem *Elasticsearch*a kao potporu postojećem spremištu podataka.

Jednostavno pitanje koje se nameće uvijek kada nad istom aplikacijom djeluju dva ili više spremišta podataka jest kako sinkronizirati sadržaj među njima. Ovisno o tome kako je sadržaj spremljen i kako mu se pristupa, postoji *Elasticsearch* dodatak koji je moguće isporučiti kako bi dva entiteta održao sinkroniziranima. Slijedi grafički prikaz:



Slika 18 - *Elasticsearch* u istom sustavu s drugim spremištem podataka

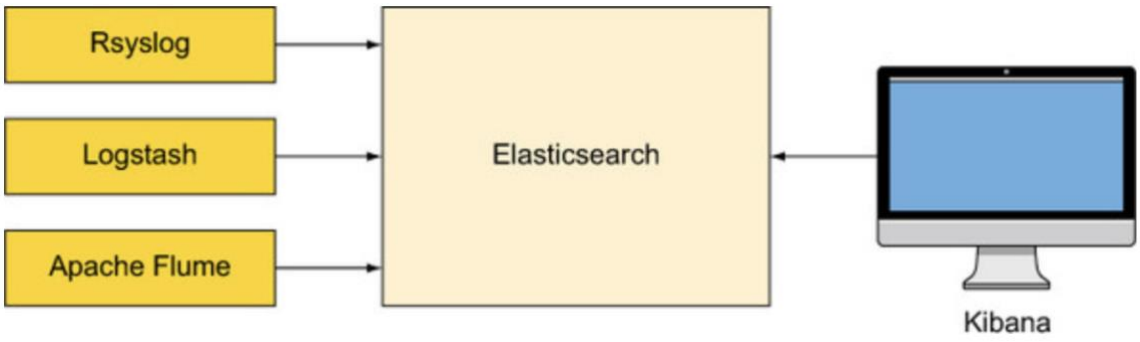
Kao što slika prikazuje, pri instaliranju *Elasticsearch*a na aplikaciji koja već sadrži *SQL* bazu podataka, svaki postojeći, kao i naknadno dodani, podatak se automatski indeksira i spreman je za pretragu. Dodavanje novih podataka i ažuriranje postojećih odvija se na *SQL* bazi podataka, dok *Elasticsearch* služi samo za pretragu.

Prije integracije *Elasticsearch*a potrebno je proučiti postojeću zajednicu gotovih alata jer ih postoji mnogo. Više o tome u idućem poglavlju.

Korištenje Elasticsearcha uz dodatak gotovih rješenja iz *Elasticsearch zajednice*

Elasticsearch zajednica je toliko napredovala da u nekim situacijama nije potrebno napisati niti jednu liniju koda kako bi se omogućilo optimalno pretraživanje. Pri tome pomažu mnogi alati izrađeni isključivo kao podrška *Elasticsearch*u.

Recimo da je cilj postaviti sustav zapisa prema ponašanju aplikacije (eng. *logging system*). Sustav će biti memorijski zahtjevan, bit će ga moguće pretraživati i analizirati velik broj događaja.



Slika 19 - *Elasticsearch* u sustavu zapisa podataka pomoću alata koji podupiru rad *Elasticsearch*a

Kao što je prikazano na slici 20, da bi procesirali zapise o ponašanju aplikacije na *Elasticsearch*, moguće je koristiti alate za zapise (eng. *logging tools*) poput *Rsyslog*a, *Logstash*a ili *Apache Flume*a. Da bi ti zapisi mogli biti pretraživani u vizualnom sučelju, moguće je koristiti još jedan od *Elastic* alata - *Kibanu*.

Instalacija *Elasticsearch*a je jednostavna na *Unix OS*-u. Preko *Homebrew* alata instalacija se postiže sljedećom naredbom:

```
brew install elasticsearch
```

Da bi potvrdio rad *Elasticsearch*a, korisnik može vidjeti zapise koji se generiraju pri pokretanju u direktoriju /var/log/elasticsearch/.

Zadani port na kojemu se pokreće *Elasticsearch* je 9300, a *Elasticsearch* svakom korisniku daje defaultno ime koje se može promijeniti u konfiguracijama.

Budući da *Elasticsearch* nudi *Java API*, rad s njim u *AEM*-u je jednostavan. Za rad su potrebne sljedeće ovisnosti u *Maven* repozitoriju:

```
<dependency>
    <groupId>org.elasticsearch</groupId>
    <artifactId>elasticsearch</artifactId>
    <version>2.1.1</version>
</dependency>
<dependency>
    <groupId>org.apache.servicemix.bundles</groupId>
    <artifactId>org.apache.servicemix.bundles.elasticsearch</artifactId>
    <version>2.1.1_1</version>
</dependency>
```

Potrebno je razviti *search box* komponentu koja će slati upite na *Elasticsearch* i prikazivati rezultate korisnicima. Podaci mogu stizati s *AEM*a, ali i od drugih izvora. Postoje dva načina kako poslati podatke iz *AEM*a u instancu *Elasticsearch*a. Prvi je baziran na korištenju *OSGi* događaja (eng. *OSGi events*) i *Sling* poslova (*Sling jobs*), a drugi se bazira na *replikacijskim agentima* (eng. *Replication Agents*). Da bi se spojili na *Elasticsearch*, potrebno je izraditi posebni *Java Klijent*, ali više o tome će biti prikazano u praktičnom dijelu rada.

7. Usporedba sustava pretraživanja *Apache Solr* i *Elasticsearch*

Usporedba će biti bazirana na sljedećih deset kriterija koje bi svatko tko implementira sustave pretraživanja u svoju aplikaciju morao uzeti u obzir. Ocjene se kreću u rasponu 1 – 10 za svaku kategoriju. Ocjene i objašnjenja dodjeljuje autor s obzirom na zaključke donesene temeljem izrade ovog projekta, *Java* programskom jeziku, *AEM* tehnologiji i istraživanju dostupnih resursa, te ona ne moraju nužno biti primjenjiva za generalne slučajeve.

7.1. Osnovna tehnologija

Prvi korak pri korištenju usluga sustava pretraživanja je istražiti osnovnu tehnologiju na kojoj su bazirani. Pri tome je potrebno u obzir uzeti trošak korištenja tehnologije s obzirom na potrebno znanje. Dakle, tehnologije otvorenog koda su obično besplatne, ali ako programer nema znanje potrebno za implementaciju sustava, možda je bolje koristiti komercijalne proizvode jer nude bolju podršku pri radu.

Tablica 6 - Osnovna tehnologija

Naziv	***Solr***	***Elasticsearch***
Ocjena	**7**	**8**
Objašnjenje	• *Solr* je projekt otvorenog koda koji posjeduje detaljnu i dobro razrađenu dokumentaciju, međutim u dokumentaciji nedostaju kvalitetni primjeri *Java* koda, većina primjera je vezana za *CURL* zahtjeve. • *Solr* je baziran na *Apache Lucene* tehnologiji i sa svakom novom verzijom *Lucene*-a, razvija se i nova verzija *Solra*. *Apache Lucene* je također projekt otvorenog koda.	• *Elasticsearch* je projekt otvorenog koda koji ima odličnu dokumentaciju za *Java* programski jezik, osim u slučaju kada ona ne postoji. Dakle, većina primjera rada s *Elasticsearch High Java* klijentom je dobro dokumentirana i sadrži primjere rada u *Javi*. Međutim, pri specifičnim implementacijama potrebno se koristiti starom verzijom Java klijenta koja nema dokumentirane primjere u Java programskom jeziku, nego u *CURL*-u. • *Elasticsearch* je također baziran na *Apache Lucene* tehnologiji, ali nije razvijan od strane *Apache*-a.

	• Postoji velik broj vanjskih zavisnosti što nije idealno za implementaciju s *OSGi*-jem jer može doći do problema s različitim verzijama među zavisnostima. • Za rad sa *Solrom* potrebno je dobro poznavanje *XML* konfiguracija i velika količina uloženog vremena na definiranje shema potrebnih za kvalitetan rad.	• Postoji velik broj vanjskih zavisnosti što nije idealno za implementaciju s *OSGi*-jem jer može doći do problema s različitim verzijama među zavisnostima. • Za rad s *Elasticsearchom* je potrebno znatno manje vremena za početnike zbog pristupa koji ne zahtijeva *XML* shemu.

7.2. Skalabilnost

Moderni sustavi pretraživanja često imaju mogućnost skaliranja na milijune ili milijarde podataka, osnovno pitanje je koliko je skaliranje zadano, a za koliko je potrebno plaćati dodatne memorijske resurse. Za ovu kategoriju je također potreba i dobra procjena kolike će naš projekt imati potrebe za skaliranjem.

Tablica 7 - Skalabilnost

Naziv	***Solr***	***Elasticsearch***
Ocjena	**9**	**9**
Objašnjenje	• *Solr* ima definirane limite koji su 2 milijarde dokumenata po jednom indeksu. Ostala ograničenja ne postoje, međutim performanse se značajno smanjuju pri neuobičajeno velikim količinama podataka. Implementacija je u potpunosti prepuštena korisniku, kao i odluka o tome koliko je dokumenata, dijelova, polja itd... razumno u korelaciji s brzinom izvođenja programa. U takvim situacijama se obično dorađuje format podataka, a izbacuju se dinamička polja jer su ona obično naznaka lošeg formata podataka i skupa operacija.	• U *Elasticsearchu* su granice točno određene, ako se ne poštuju dođe do pogoršanja performansi. Npr. maksimalni broj dokumenata po jednom dijelu (eng. *shard*) je 2 milijarde. Ne preporučiva se korištenje velikih dokumenata, nego veći broj manjih dokumenata. Maksimalni broj polja unutar dokumenta se može definirati u konfiguracijskoj datoteci. Također, ne preporučuje se vraćanje velikih skupova rezultata, ako je to baš potrebno, preporučuje se korištenje *Scroll API*-ja. To su preporuke koje postoje u trenutnoj dokumentaciji.

	• Jako je teško doseći limite u produkcijskoj aplikaciji.	• Jako je teško doseći limite u produkcijskoj aplikaciji. Ako se to dogodi, vjerojatno se radi o pogrešnom korištenju tehnologije.

7.3. Sustavi za povezivanje podataka i analitiku

Vrsta podataka odnosi se na mogućnost uvoza više vrsti podataka, odnosno podatak direktno iz više različitih izvora. Ti izvori mogu biti datotečni sustavi, relacijske baze podataka, društvene mreže, sustavi za verzioniranje, Web stranice, sustavi upravljanja sadržajem (eng. *Content Management Systems – CMS*), srodne aplikacije i sl.

Tablica 8 - Sustavi za povezivanje podataka i analitiku

Naziv	***Solr***	***Elasticsearch***
Ocjena	**10**	**10**
Objašnjenje	• Datotečni sustavi • Relacijske baze podataka • *CMS*-ovi (*Adobe Experience Manager*) • Aplikacije za suradnju (npr. *Jira, Kafka, IBM* konektori i sl.) • Ostali repozitoriji • *Web* pretraživači	• Datotečni sustavi • Relacijske baze podataka • *CMS*-ovi (*Adobe Experience Manager*) • Aplikacije za suradnju (npr. *Jira, Kafka, IBM* konektori i sl.) • Ostali repozitoriji • Web pretraživači

7.4. Procesiranje sadržaja

Procesiranje sadržaja je kritična funkcionalnost svakog sustava pretraživanja. Samim time se nude brojne opcije pri obradi i procesiranju sadržaja. Slijede važne karakteristike:

- Spajanje podataka – često je potrebno pretraživati podatke iz više izvora. Jako je važno podatke prije indeksiranja normalizirati, tj. odrediti jedan format u kojeg će biti pretvarani svi podaci kako bi bili jednakog formata pri indeksiranju.

- Taksonomija – taksonomija bazirana na kategorizaciji dokumenata i korištenju kompleksnih pravila koja omogućavaju tvrtkama da bolje predstave svoje podatke indeksu zbog relevantnosti. Primjer toga može biti kreiranje rječnika (eng. *dictionary*) koji pomažu razumijevanju konteksta. Npr riječ „*developer*“ u engleskom jeziku može značiti „*software developer*“, ali može značiti i „*real estate developer*“.
- Prepoznavanje entiteta i podataka – mogućnost sustava pretraživanja da prepozna strukturu ili uzorak podataka kao što su npr. datum, broj mobitela, email adresa, poštanski broj, registarske oznake i sl. Ali ne samo standardne podatke, nego i da se prilagodi prepoznavanju bilo kojih podataka koji pripadaju određenom uzorku ili regularnom izrazu (eng. *REGEX expression*).
- Rad s nestrukturiranim podacima – u svakoj vrsti podataka se mogu pronaći uzorci pa je važna i dobra analiza s korisničke strane. Za rad s nestrukturiranim podacima obično se koriste pravila ponašanja i strojno učenje (za statističku analizu sadržaja, potencijalno skupe operacije za veliku količinu sadržaja pri kojoj može pomoći „*Handoop“* i sl.).
- Normalizacija podataka – svaki sustav pretraživanja mora imati mogućnost normalizacije različitog sadržaja jer se uspoređivati mogu samo podaci istog formata.

Primjer: bez kvalitetnog procesiranja sadržaja, može doći do sljedeće situacije:

Tablica 9 - Primjer potrebe za kvalitetnim procesiranjem sadržaja

Dokument 1	**Dokument 2**	**Dokument 3**
Prezime i Ime	Ime	Puno Ime
HR	Hrvatska	Republika Hrvatska
ZG (limit na dva znaka)	Zagreb (limit nije postavljen)	Grad Zagreb (limit na 35 znakova)

Tablica 10 - Procesiranje sadržaja

Naziv	***Solr***	***Elasticsearch***
Ocjena	**10**	**10**

Objašnjenje	• Postoji opcija spajanja indeksa preko „*IndexMergeTool“* alata. Za to je potrebno imati usklađene indekse, dakle sheme im moraju biti jednake i na jednak način moraju analizirati polja u dokumentima. • U ovom primjeru taksonomija, klasifikacija i grupiranje nisu imali veliku ulogu jer se radilo o jednostavnijem skupu podataka gdje je svaki dokumente predstavljao stranicu ili sliku sa svojim svojstvima kao poljima indeksa. Nekada to nije slučaj jer postoje industrijski definirani standardi, nepotpuni podaci i sl. U *Solr* postoje mehanizmi za stvaranje taksonomije podataka kao i posebno grupiranje unutar taksonomije. • *Solr* podržava rad s entitetima u tekstu. Ova opcija je specifična za svaki poslovni slučaj pa postoji mnogo načina kako to ostvariti (korištenje akronima, fraza, formata podataka i sl.). • *Solr* podržava uvođenje pravila i unaprjeđenje relevantnosti preko strojnog učenja. • Zbog striktno definirane sheme sa Solrom je teže raditi za početnike, ali jednom kada se indeksiranje i dohvaćanje rezultat uspješno postavi, mogućnost za pogrešku je znatno manja i promjene su lako izvedive.	• Ne postoji konkretan alat koji spaja indekse, ali se to može učiniti tako da se postavi opcija „*max_num_segments*=1“ u konfiguraciju i tada se pokrene *forcemerge* funkcija. Drugi način je reindeksiranje sadržaja. • Taksonomija, grupiranje i klasifikacija su opcije koje postoje u *Elasticsearchu*. Nema razlike sa *Solr*-om. • U Elasticsearchu postoji *Annotated Text Plugin* – dodatak koji omogućava prepoznavanje entiteta u tekstu na vrlo efikasan način. U ovom radu nije bilo potrebe za korištenjem entiteta jer to nije čest slučaj u praksi. Postoji mogućnost označavanja entiteta u tekstu koji su povezani s upitom, pozicioniranja na takav tekst i sl. • *Elasticsearch* podržava uvođenje pravila i unaprjeđenje relevantnosti preko strojnog učenja. Postoji mogućnost uvođenja modificiranih pravila na bazi analize strojnog učenja o anomalijama u povijesnim podacima. • S *Elasticsearchom* je jednostavno započeti rad zbog načina prepoznavanja svake vrste podataka koju dobije bez unaprijed definirane sheme. Dojam stečen kroz ovaj projekt je da se s *Elasticsearcom* konfiguracije mijenjaju kroz programski kod, dok je u *Solru* to isključivo preko *XML* konfiguracije.

7.5. Indeksiranje

Kao osnovni koncept na kojemu se temelji rad sustava pretraživanja, važna je brzina indeksiranja sadržaja. Osim brzine važna je i dostupnost sadržaja, tj. brzina kojom možemo pregledati sadržaj dostupan u indeksu. Sljedeća važna stavka je latentnost indeksiranja, odnosno vrijeme koje je potrebno između indeksiranja i vremena kada je sadržaj dostupan u za pretragu. Važnu stavku pri indeksiranju čine i dinamička polja. Pomoću dinamičkih polja ne postoji stalna potreba za reindeksiranjem (ponovnim indeksiranjem cijelog sadržaja), koje je zahtjevan proces i s memorijskog aspekta, ali i s aspekta brzine. Pomoću dinamičnih polja, osim slobode u indeksiranju sadržaja korisnik dobiva i mogućnost da se ponovno indeksiraju samo promijenjena polja u indeksu.

Brzinu indeksiranja je teško izmjeriti jer se implementacija razlikuje u sustavima pretraživanja koji se uspoređuju. Važno je spomenuti da je brzina indeksiranja zadovoljavajuća za *AEM* podatke koji su kroz ovaj rad testirani, te da se brzina uvijek može postavkama unaprijediti kod oba sustava pretraživanja.

Tablica 11 - Indeksiranje

Naziv	***Solr***	***Elasticsearch***
Ocjena	**10**	**10**
Objašnjenje	• Ako postoje visoke latencija pri indeksiranju, problem najčešće leži u fizičkoj memoriji. • Postoji mogućnost postavljanja dinamičkih polja u konfiguracijskoj datoteci.	• Latencija indeksa ovisi o postavkama. Do visoke latencije može doći ako se ne obrati pozornost na korištenje *BulkIndexRequest* zahtjeva i na raspored podataka zbog važnosti jednakog rasporeda vremena čekanja između klastera. • Dinamička polja su zadano ponašanje koje se može dodatno isključiti na razini objekta ili dokumenta.

7.6. Funkcionalnost upita

Također jedna od osnovnih funkcionalnost sustava pretraživanja koji mora podržavati i optimizirati različite vrste upita potrebne za poslovne slučajeve.

Sljedeće funkcionalnosti bi sustav pretraživanja trebao podržavati:

- Brzina upita
- Pretraga ključnih riječi
- *Boolean* pretraga (kombinacija ključnih riječi preko operatora)
- Posebni znakovi (eng. *wildcards*)
- *XML* pretraga
- Pozicioniranje (eng. *facets*) rezultata
- Sortiranje
- Intervali pretrage
- *Stemming* i *lemming* opcije
- Sinonimi
- Prijedlozi pretrage i automatska nadopuna upita

Tablica 12 - Funkcionalnost upita

Naziv	***Solr***	***Elasticsearch***
Ocjena	**10**	**10**
Objašnjenje	• *Solr* podržava sve nabrojene funkcionalnosti.	• *Elasticsearch* podržava sve nabrojene funkcionalnosti.

7.7. Relevantnost pretrage

Relevantnost pretrage se preslikava u značaju prvih prikazanih rezultata za korisnika. Međutim to je uvijek težak posao jer se razmišljanja, pa tako i napisani termini, korisnika znatno razlikuju. Tu veliki značaj ima poboljšavanje relevantnosti pretrage u skladu s analizom korisničkih termina pretrage i potreba. To se može ostvariti na sljedeći način:

- Unaprjeđenje relevantnosti (eng. *Relevancy boosting*) – prioritiziranje rezultata bazirano na specifičnim kriterijima.
- Dodjeljivanje bodova dokumentima – bazirano na povijesnim podacima pretrage
- Sloboda u mijenjanju rezultata

Nije primijećena razlika između *Solra* i *Elasticsearcha* u ovom području. Oba sustava pretraživanja nude visoku razinu slobode pri određivanju rezultata kao i u prioritiziranju polja ili dokumenata. U praktičnom slučaju koji je obrađivan u ovom radu, nije bilo potrebe za mijenjanjem rezultata iako je unaprjeđenje polja testirano.

I *Solr* i *Elasticsearch* nude ispis dnevnika rada (eng. *log*) čijom se analizom mogu iščitati zaključci o relevantnosti pretrage pa u skladu s tim i mijenjati rezultati. Osim dnevnika rada, moguće je pratiti i dnevnike klikova na stranici te korisničkih kretnji. Tu je važno uočiti broj pokušaja potrebnih za pronalazak ispravnih rezultata. Postoje brojne tehnike rada s velikim količinama podataka (eng. *big data*), strojnog učenja i prediktivne analize koji mogu pomoći pri problemima s relevantnosti pretrage.

7.8. Sigurnost

Kao i u svakom drugom području *IT*-ja, sigurnost je ključni faktor uspjeha sustava pretraživanja. Kada god postoji više repozitorija rada poseban je naglasak na autorizaciji, autentikaciji i definiranju korisničkih uloga u jednu shemu.

Tablica 13 - Sigurnost

Naziv	***Solr***	***Elasticsearch***
Ocjena	**8**	**9**
Objašnjenje	• Autentikacija – definirana u security.json datoteci. Može se postaviti i preko *Basic authentication API*-ja. Korisničko ime i lozinka se šalju u običnom tekstu pa je potrebno osigurati *SSL*. • Autorizacija – preko *Rule-Based Authorization Plugina*. • Nije pronađeno konkretno rješenje za sigurnost definiranu na razini dokumenta. • Sigurnost na razini polja se postiže kopiranjem i sakrivanjem polja u indeksu bazirano na ulozi korisnika.	• Autentikacija – definirana u *„xpack.security.authc.realms“* postavkama unutar *elasticsearch.yaml* datoteke. Podržava postojeće sustave za autentikaciju kao *LDAP* (bez podrške za pod grupe), *PKI, Actrive Directory, Kerberos* i sl. • Autorizacija – postoji *Rule-based* i *attribute-based autorizacija*. Npr. korisnik može vidjeti dokumente ako ima određenu ulogu, ali i na temelju atributa koji su dodijeljeni njemu i dokumentu. To pojačava sigurnost na razini dokumenta. • Također postoje sigurnosni mehanizmi na razini polja.

		• Postoje točne definirani limiti kada se radi o sigurnosti. Npr. vanjski dodaci se mogu koristiti, ali nisu podržani po službenoj dokumentaciji *Elasticsearcha* jer se sigurnost vanjskih izvora ne može osigurati.

7.9. Korisničko sučelje

Korisničko sučelje je jednako važno kao i pozadinska konfiguracija. Činjenica da postoji intuitivno korisničko sučelje za nadzor podataka od početka razvoja znatno olakšava posao implementacije. Kod oba promatrana sustava pretraživanja je potrebno razviti korisničko sučelje jer ne postoje gotove varijante. Postoje samo vanjske implementacije korisničkih sučelja koje su besplatne.

7.10. Administracija, nadzor i održavanje

Tablica 14 - Administracija, nadzor i održavanje

Naziv	***Solr***	***Elasticsearch***
Ocjena	**10**	**9**
Objašnjenje	• Postoji administracijsko sučelje koje pruža detaljnu analizu indeksa, testiranje upita i prikaz memorijskih resursa.	• Postoji intuitivno administracijsko sučelje koje se naplaćuje nakon 14 dana korištenja.

7.11. Korištenje *Apache Lucenea* kao samostalne tehnologije

Korištenje *Apache Lucenea* kao samostalne tehnologije nudi najveću slobodu pri implementaciji sustava pretraživanja, međutim ne preporučuje se zbog kompleksnosti implementacije. Gotovo da je potrebno napraviti cijeli sustav pretraživanja pa ga tek onda koristiti jer je *Lucene* samo biblioteka koja se ne može samostalno koristiti. Pri tome pomažu sustavi pretraživanja koji implementiraju *Lucene* kao što su *Apache Solr* i *Elasticsearch* preko čijih *API*-ja i konfiguracijskih datoteka se može postići većina funkcionalnosti *Lucenea*. *Solr* i *Elasticsearch* se prilagođavaju novim funkcionalnostima koje *Lucene* pruža svakom novom verzijom.

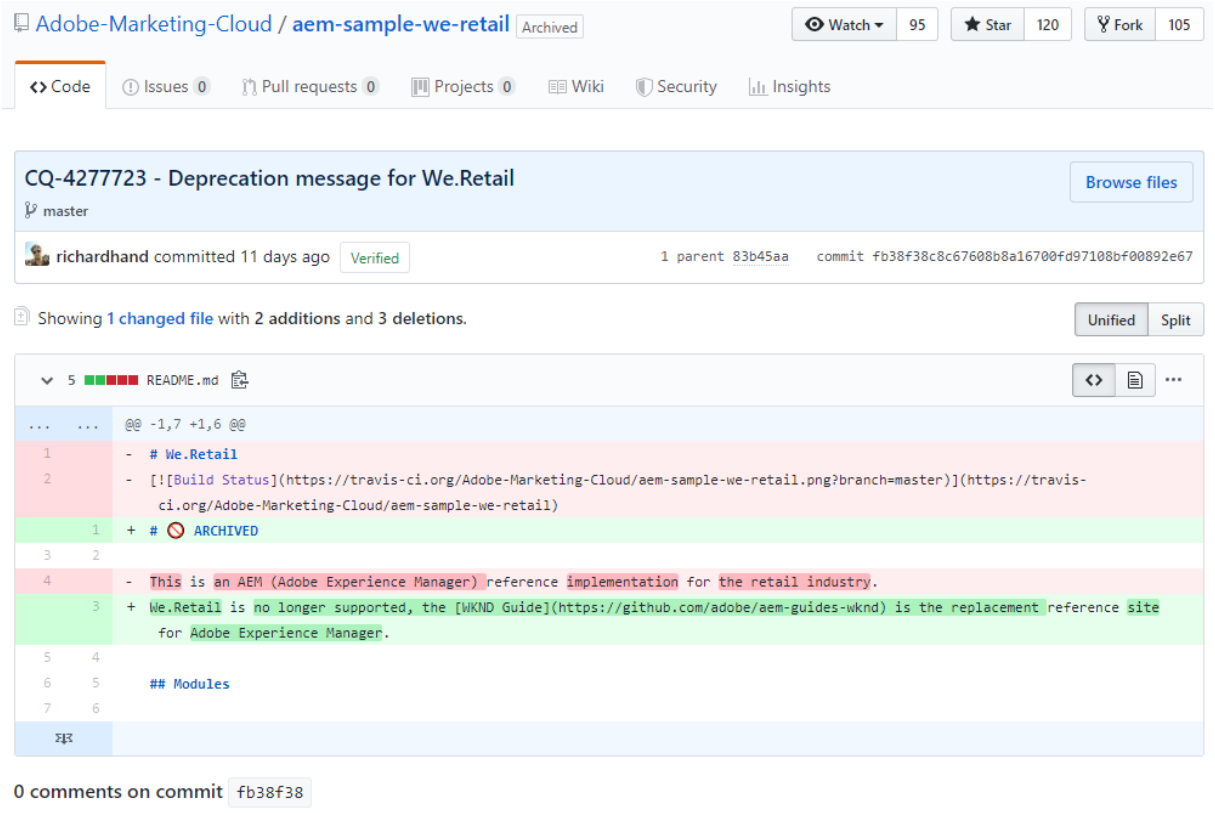
Samostalna *Apache Lucene* biblioteka može biti pogodna kada je potrebno funkcionalnost pretrage implementirati u računalnu aplikaciju (eng. *Desktop application*) ili kada su zahtjevi za pretragom jako specifični pa je potrebno koristiti *Lucene API*-je niske razine jer *Solr* i *Elasticsearch* ne nude potrebnu razinu slobode za modifikacije sustava pretraživanja.

Adobe Experience Manager nudi integriranu pretragu preko *Lucene* biblioteke koja odlično odgovara zahtjevima specifične baze podataka u *AEM*-u. Tako je moguće indeksirati i pretraživati podatke koji se nalaze u bazi podataka na brz i efikasan način uz jednostavnu implementaciju pravila *Lucene*-a.

8. Aplikacija *We Retail*

Osnovni podaci o aplikaciji *We Retail* kao i razlozi izbora te aplikacije izneseni su u poglavlju *Metode i tehnike rada*. U ovom poglavlju bit će govora o tehničkim karakteristikama aplikacije čiji je izvorni kod dostupan na sljedećem *URL*-u: *https://github.com/Adobe-Marketing-Cloud/aem-sample-we-retail*.

*Napomena: aplikacija We Retail je pred kraj pisanja ovog rada proglašena zastarjelom i umjesto nje se preporučuje korištenje aplikacije **aem-guides-wknd**. To znači da se We Retail i dalje može koristiti, ali nove funkcionalnosti koje dolaze s AEM 6.5 verzijom neće biti implementirane u We Retail i da prestaje podrška od strane Adobe razvojnog tima.*



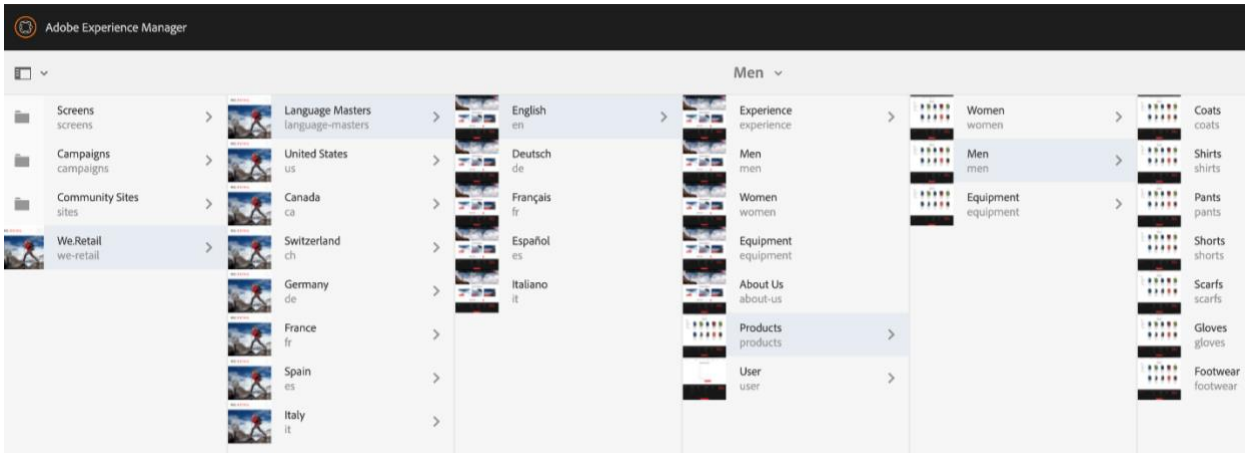
Slika 20 - *We Retail* proglašena zastarjelom

We Retail je online trgovina popraćena najboljim metodama razvoja softvera u *AEM*-u. Razvijena je od strane *Adobe* razvojnog tima. Sastoji se od mnogih dijelova koji će biti predstavljeni u nastavku:

- Početna stranica
- Muška oprema

- Ženska oprema
- Oprema
- O nama
- Proizvodi
- Zajednica

Na sljedećoj slici je prikaz hijerarhije stranice.



Slika 21 - *We Retail* hijerarhija stranica

Prikazana hijerarhija tipična je za *AEM* stranice. *Language Masters* označava općenita svojstva svih stranica, a sve ostale stranice su samo kopije na drugim jezicima koje proizlaze iz *Language Masters* stranice. Dakle, svaka stranica nasljeđuje sva svojstva *Language Masters* stranica, a to nasljeđe se može jednostavno prekinuti u svojstvima svake od stranica ako je to potrebno.

Lako je uočiti kako već postoji mnogo podataka za koje je potrebna pretraga. O samoj pretrazi u *We Retail* stranici će biti riječi kasnije. Slijedi prikaz komponenti na stranicama.

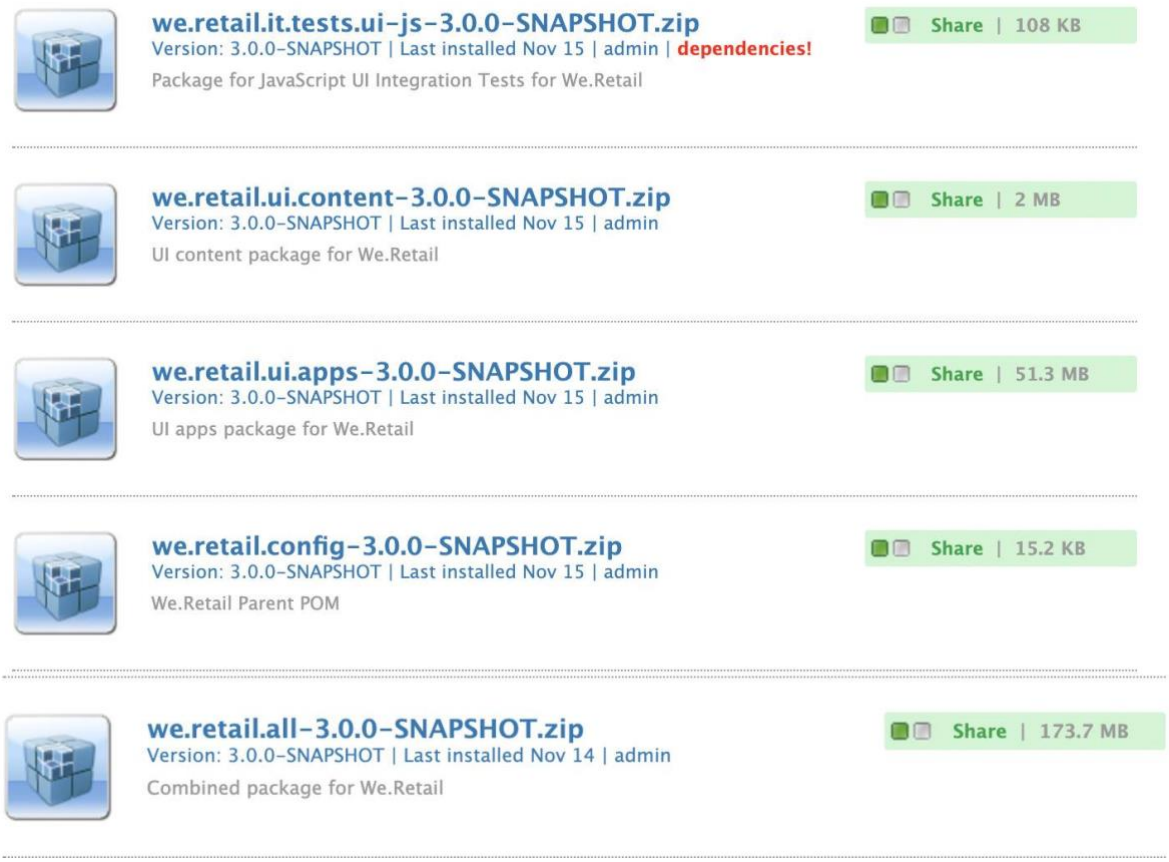
Svaka od navedenih stranica se sastoji od više komponenti. Popis komponenti slijedi:

- *Breadcrumb*
- *Category Teaser*
- *Content Fragment*
- *Experience Fragment*
- *Hero Image*
- *Image*
- *Layout Container*

- *Link Button*
- *Link*
- *Mini Shopping Cart*
- *Navigation*
- *Order Details*
- *Order History*
- *Product*
- *Product Grid*
- *Product Recommendations*
- *Shopping Cart*
- *Shopping Cart Prices*
- *Site Feature*
- *Social Media Sharing*
- *Text*
- *Title*

Sve navedene komponente su osnovne komponente *AEM*-a (eng*. Core Components*) koje se mogu jednostavno uključiti u bilo koju aplikaciju u *AEM*-u zbog mogućnosti nasljeđivanja s putanje */apps/core/wcm/components*. Cijela *We Retail* stranica sastavljena je od takvih komponenti uz par okvira i fragmenata. Osim osnovnih komponenti s *AEM* instancom postoje zadani *OSGi* paketi, a s *We Retail* projektom ostvaren je *Core* paket (eng. *bundle*).

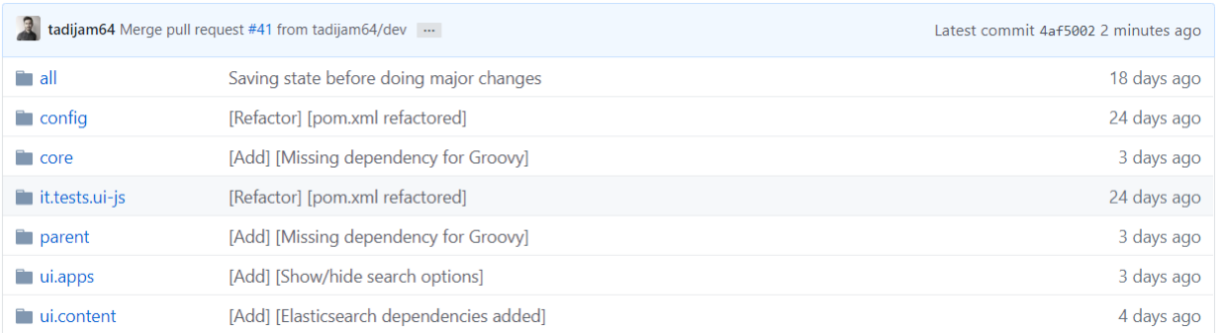
Preko *Maven* alata za izgradnju aplikacije, na *AEM* instancu su isporučeni sljedeći paketi:



Slika 22 - Paketi isporučeni na *AEM* poslužitelj

Svi moduli se isporučuju kao paketi na *AEM* instancu, osim *core* modula koji je isporučen kao *OSGi* paket.

Cijeli projekt podijeljen je na module.



Slika 23 - *Moduli projekta*

- *all* – osnovni modul koji spaja ostale module preko *Maven* postavki kako bi se ponašale kao dio jednog projekta.
- *config* – generalna konfiguracija projekta, postavljanje opcija za izgradnju i pokretanje programa, izbacivanje određenih datoteka iz procesa izgradnje projekta, git postavke i sl.
- *core* – poslovna logika, osnovni paket (eng. *bundle*) koji upravlja ostalim paketima, vanjskim zavisnostima i sl.
- *it.tests.ui-js* – testovi korisničkog sučelja
- *parent* – definicija svojstava koja se koriste u konfiguraciji svakog modula, upravljanje vanjskim zavisnostima, njihovim verzijama i sl.
- *ui.apps* – aplikacijski modul koji podrazumijeva popis komponenti i njihovog korisničkog sučelja
- *ui.content* – sadržaj projekta poput slika, dokumenata, kataloga i sl.

9. Implementacija

Za proces razvoja istraživanja napravljena je ploča rada na *Trello*-u. Cijeli proces odrađenih zadataka je kronološki poredan i javno dostupan na sljedećoj adresi: *https://trello.com/b/2UHVPdmz/masters-thesis-search-engines-in-adobe-experience-manager*.

Proces razvoja aplikacije dostupan je na *GitHub* repozitoriju *search-engines-comparison-on-we-retail* na *URL*-u*:* *https://github.com/tadijam64/search-engines-comparison-on-we-retail*. Glavne i stabilne (eng. *release*) verzije projekta nalaze se na *master* grani (eng. *branch*). Sve funkcionalnosti koje su se u određenom trenutku razvijale isporučivane su na *dev* (eng. *development*) granu. Na popisu objava (eng. *commits*) može se vidjeti detaljan kronološki razvoj projekta. Svaka nova funkcionalnost razvijana je na posebnoj grani određenoj samo za razvoj te funkcionalnosti. Grane su na kraju projekta obrisane zbog bolje preglednosti, ali se njihov popis može vidjeti u pogledu na kronološka spajanja (eng. *pull requests*) s granom *dev*.

9.1. Konfiguracija projekta

Prvi korak je postavljanje projekta u početnu fazu kako bi bio spreman za rad. To se odnosi na pokretanje projekta, nadogradnju, proces izgradnje (eng. *build process*), promjene i slično. Za to je potrebno uskladiti *Maven* postavke *pom.xml* datoteke te napraviti određene promjene u *filter.xml* datoteci projekta. Slijedi prikaz i opis promjena:

Sljedeći dio koda je potrebno ukloniti iz projekta (*parent/pom.xml*).

```
<plugin>
    <groupId>org.apache.rat</groupId>
    <artifactId>apache-rat-plugin</artifactId>
    <configuration>
        <excludes combine.children="append">
            <!--  Used by maven-remote-resources-plugin -->
            <exclude>src/main/appended-resources/META-INF/*</exclude>
            <!--  Generated by maven-remote-resources-plugin -->
            <exclude>velocity.log</exclude>
            <!-- don't check anything in target -->
            <exclude>target/*</exclude>
            <!-- README files in markdown format -->
```

```
            <exclude>README.md</exclude>
  <!-- Ignore files generated by IDE plugins e.g. maven-eclipse-plugin -->
            <exclude>maven-eclipse.xml</exclude>
            <!-- Ignore VLT .content.xml files + dialog configurations -->
            <exclude>**/jcr_root/**/*.xml</exclude>
            <!-- Exclude all svg files -->
            <exclude>**/*.svg</exclude>
            <!-- Ignore auto-generated VLT file -->
            <exclude>**/META-INF/vault/settings.xml</exclude>
            <!-- Ignore .vlt files -->
            <exclude>**/.vlt</exclude>
            <!-- Exclude all .properties files -->
            <exclude>**/*.properties</exclude>
            <!-- Ignore .vltignore files -->
            <exclude>**/.vltignore</exclude>
            <!-- Ignore vendor files -->
            <exclude>**/clientlibs/vendor/**/*</exclude>
            <!-- Exclude all JSON files -->
            <exclude>**/*.json</exclude>
            <!-- Generated for release source archives -->
            <exclude>DEPENDENCIES</exclude>
            <!-- .rej files from svn/patch -->
            <exclude>**/*.rej</exclude>
            <!-- Jenkins configuration file -->
            <exclude>Jenkinsfile</exclude>
        </excludes>
    </configuration>
    <executions>
        <execution>
            <phase>verify</phase>
            <goals>
               <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>


<plugin>
    <groupId>org.apache.rat</groupId>
    <artifactId>apache-rat-plugin</artifactId>
```

```
    <version>0.12</version>
</plugin>
```

Kod koji je uklonjen odnosi se na *Apache Rat* dodatak kojim *Adobe* zabranjuje promjenu nad datotekama projekta bez *Adobe* licence u zaglavlju svake datoteke. Budući da se u radu razvijaju nove funkcionalnosti, navedeni dodatak je izbrisan kako bi to bilo moguće. Sve navedeno sukladno je s licencom koju je *Adobe* izdao za *We Retail* projekt. Cijela verzija licence je dostupna unutar projekta pod nazivom *LICENSE*.

Kod se pokreće preko *Maven* alata za izgradnju aplikacija pomoću faza -*PautoInstallPackage* – za instalaciju komponenti i promjena na sučelju, te -*PautoInstallBundle* za izgradnju *Core* modula tj. poslovne logike. Navedene faze se pokreću s ciljevima *Clean* i *Install*.

Kako bi projekt bio spreman za razvoj, potrebno je uskladiti *Maven* postavke.

```
<subPackage>
        <groupId>com.adobe.cq</groupId>
        <artifactId>core.wcm.components.all</artifactId>
        <filter>true</filter>
</subPackage>

<dependency>
    <groupId>com.adobe.cq</groupId>
    <artifactId>core.wcm.components.all</artifactId>
    <type>zip</type>
    <scope>provided</scope>
</dependency>
```

U glavnu parent/pom.xml konfiguracijsku datoteku potrebno je dodati core.wcm.components.all vanjske zavisnost kako bi se mogle nasljeđivati osnovne, gotove komponente *AEM*-a. Osnovne komponente programirane su poštujući *Adobe*jeve i *Javine* konvekcije pisanja koda i preporučuje se njihovo korištenje kada je god to moguće. U ovom projektu će se naslijediti osnovna komponenta pretrage (eng. *search*) nad kojom će se razvijati dodatne funkcionalnosti.

```
<embeddeds>
    <embedded>
        <groupId>com.adobe.cq.sample</groupId>
        <artifactId>we.retail.core</artifactId>
```

```
        <target>/apps/weretail/install</target>
    </embedded>
</embeddeds>
```

U osnovnu konfiguracijsku datoteku potrebno je dodati *we.retail.core* modul kako bi se omogućila isporuka paketa (eng. *bundle*) na instancu *AEM* poslužitelja.

```
<filter root="/apps/weretail"/>
```

Kako bi se u paket isporučila aplikacije, korijensku putanju do aplikacije potrebno je postaviti u *filter.xml* konfiguracijsku datoteku.

9.2. Priprema aplikacije za implementaciju sustava pretraživanja

U nastavku će biti prikazan kod komponente za pretragu – *search*. Preko *search* komponente će biti pretraživan sadržaj aplikacije.

content.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
xmlns:cq="http://www.day.com/jcr/cq/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0"
    jcr:description="Search component for the modal dialog"
    jcr:primaryType="cq:Component"
    jcr:title="We.Retail Search"
    sling:resourceSuperType="core/wcm/components/search/v1/search"
    componentGroup="We.Retail.Structure"/>
```

search.html:

```
<section class="cmp-search" role="search"
     data-sly-use.search="com.adobe.cq.wcm.core.components.models.Search"
     data-cmp-is="search"
     data-cmp-min-length="${search.searchTermMinimumLength}"
     data-cmp-results-size="${search.resultsSize}">
  <form class="cmp-search__form" data-cmp-hook-search="form"
     method="get" action="${currentPage.path @
addSelectors=['mysearchresults'], extension='json', suffix =
search.relativePath}"
```

```
    autocomplete="off">
    <div class="cmp-search__field">
        <i class="cmp-search__icon" data-cmp-hook-search="icon"></i>
        <span class="cmp-search__loading-indicator" data-cmp-hook-
search="loadingIndicator"></span>
        <input class="cmp-search__input" data-cmp-hook-search="input"
type="text" name="fulltext" placeholder="${'Search' @ i18n}"
role="combobox" aria-autocomplete="list" aria-haspopup="true" aria-
invalid="false">
        <button class="cmp-search__clear" data-cmp-hook-search="clear">
            <i class="cmp-search__clear-icon"></i>
        </button>
    </div>
  </form>
  <div class="cmp-search__results" data-cmp-hook-search="results"
role="listbox" aria-multiselectable="false"></div>
  <sly data-sly-include="itemTemplate.html"/>
</section>
```

Najprije je unutar *content.xml*-a u podebljanom dijelu koda označeno kako ova *search* komponenta nasljeđuje osnovnu, gotovu search komponentu *AEM*-a iz core modula. Nad tom komponentom se dodaje *search.html* datoteka pomoću koje autor poziva *SearchServlet* - modificirani *servlet*. U datoteci *search.html* se koristi logika iz osnovnog *core* modela *search*.

core/src/main/java/we/retail/core/servlets/SearchServlet.java:

```
package we.retail.core.servlets;

import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.HttpConstants;
import org.apache.sling.api.servlets.ServletResolverConstants;
import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
import org.osgi.service.component.annotations.Component;

import javax.servlet.Servlet;
import javax.servlet.ServletException;
import java.io.IOException;

@Component(service = Servlet.class, immediate = true,
  property = {
```

```
      ServletResolverConstants.SLING_SERVLET_METHODS + "=" +
HttpConstants.METHOD_GET,
      ServletResolverConstants.SLING_SERVLET_SELECTORS + "=" +
"mysearchresults",
      ServletResolverConstants.SLING_SERVLET_EXTENSIONS + "=" + "json"
})
public class SearchServlet extends SlingSafeMethodsServlet {


    protected void doGet(SlingHttpServletRequest request,
SlingHttpServletResponse response) throws ServletException, IOException {
        response.getWriter().write("search results...");
    }
}
```

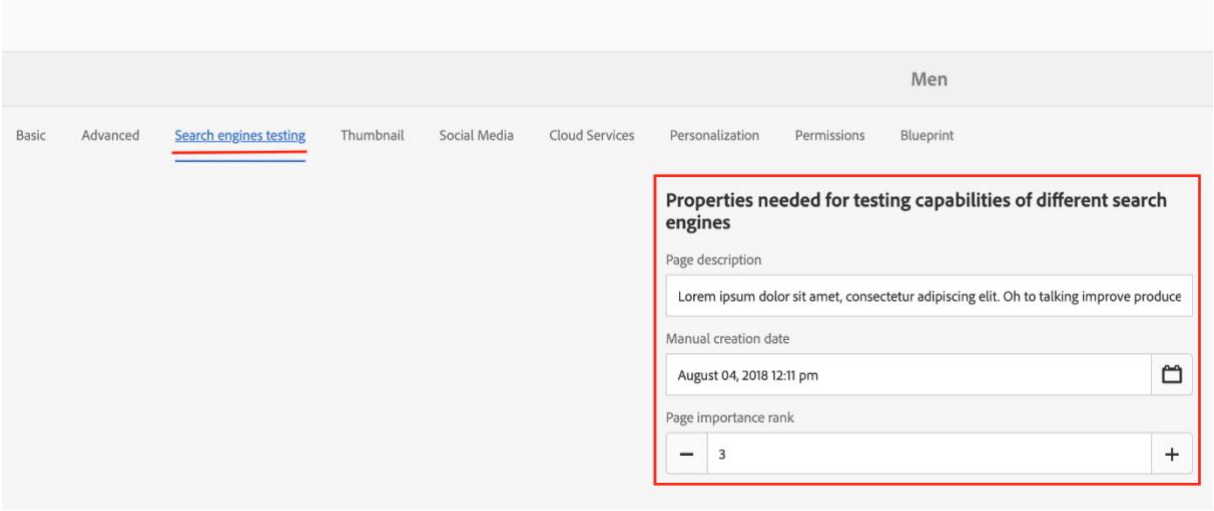
Prethodno prikazanim kodom je ostvareno sljedeće:

- Naslijeđena je osnovna komponenta pretrage;
- Komponenta pretrage je modificirana da koristi novo kreirani servlet za pretragu tako što joj je dodan potrebni selektor element u obrazac;
- Napravljeni servlet preko istog selektora obrađuje poziv iz komponente pretrage te vraća odgovor "*search results*...".

Budući da se koristi projekt koji je napravljen prema najboljim programerskim praksama *AEM*-a, implementirana pretraga već koristi indeksirano pretraživanje. Kako bi izbjegli korištenje indeksa, njih je potrebno onemogućiti, odnosno isključiti iz pretrage. Osim toga, kako bi sustavi pretraživanja bili ispravno testirani, potrebno je omogućiti različite tipove podataka koji će spremati različite vrste sadržaja. Za to su izrađena posebna svojstva koja su dodana na svaki dio sadržaja (eng. *asset*) i na postavke svake stranice. Navedena svojstva su sljedeća:

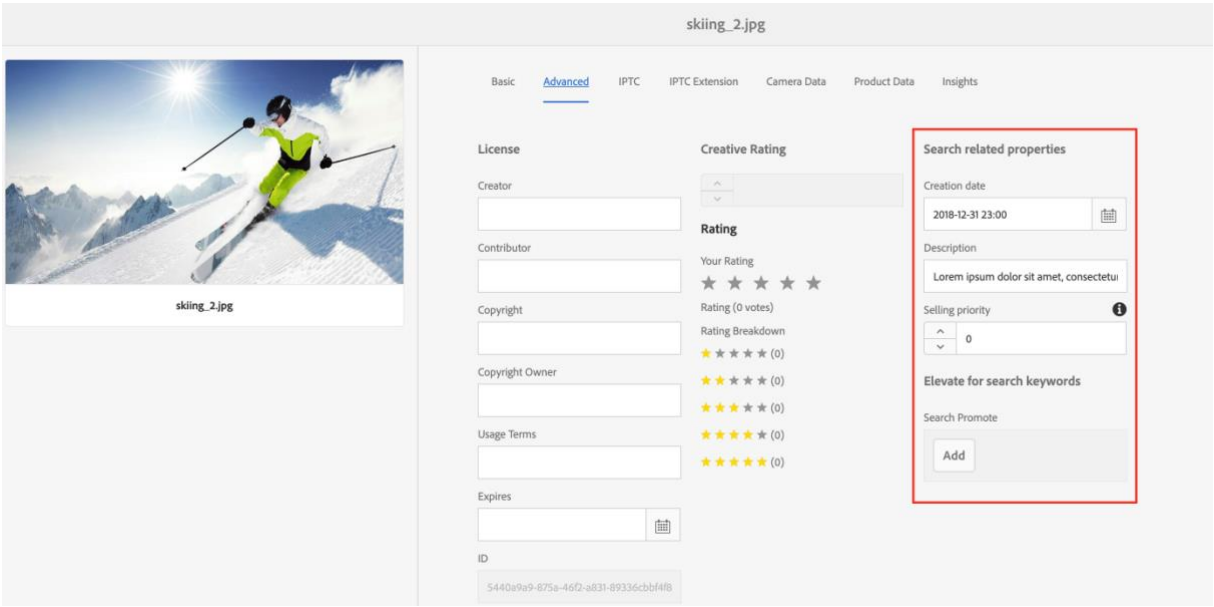
- *pageDescription/description* – opis stranice/sadržaja tipa *String*
- *creationDate* – datum izrade stranice/sadržaja tipa *Date*
- *sellingPriority* – prioritet prodaje stranice ili proizvoda sa slike tipa *Decimal*.

Na ovaj način u istraživanje ulaze tri najčešće korištena tipa podataka čije će se vrijednosti pretraživati u svojstvima sadržaja i svojstvima stranica. Izgled dodanih svojstava na stranici je sljedeći:



Slika 24 - Dodana svojstva za pretraživanje na stranice

Izgled dodanih svojstava na sadržaju (slikama, dokumentima i sl.) ima sljedeći izgled:



Slika 25 - Dodana svojstva za pretraživanje na sadržaj (eng. *asset*)

Sva dodana svojstva su referencirana u *i18n* datoteci, a nakon toga se samo koriste u kodu preko varijabli. Tako je za moguća buduća proširenja dodana mogućnost lake implementacije prijevoda na druge jezike. Putanja do konfiguracije za elemente sadržaja mora biti dodana i u *filter.xml* datoteku kako bi bila dio izvršavanja aplikacije.

Novo dodanim svojstvima potrebno je pridružiti vrijednosti. Budući da se *We Retail* projekt sastoji od preko stotinu stranica, slika, dokumenata i sl., proces dodavanja vrijednosti svakom svojstvu na svakoj stranici i svakom elementu sadržaja kroz *AEM* je izuzetno zahtjevan. Najbolji način za postizanje navedenog je izvršavanje *Groovy* skripte direktno na autor instanci poslužitelja.

Slijedi prikaz napisane *Groovy* skripte:

```
import org.apache.sling.api.resource.ModifiableValueMap
import org.apache.sling.api.resource.Resource
import org.apache.sling.api.resource.ResourceResolver

import javax.jcr.Node
import com.day.cq.wcm.api.Page;
import groovy.transform.Field;

// Warning: comment the following like in order to see test run.
Uncommented line will apply the changes!
session.save();

//Three random examples of properties
@Field descriptionList = ["Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Oh to talking improve produce in limited offices fifteen
an. Wicket branch to answer do we. Place are decay men hours tiled.",
                          "Lorem ipsum dolor sit amet, consectetur
adipiscing elit. On no twenty spring of in esteem spirit likely estate.
Continue new you declared differed learning bringing honoured. At mean mind
so upon they rent am walk. Shortly am waiting inhabit smiling he chiefly of
in. Lain tore time gone him his dear sure.",
                          "Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sportsman delighted improving dashwoods gay instantly
happiness six. Ham now amounted absolute not mistaken way pleasant
whatever. At an these still no dried folly stood thing. Rapid it on hours
hills it seven years. If polite he active county in spirit an. Mrs ham
intention promotion engrossed assurance defective. Confined so graceful
building opinions whatever trifling in. Insisted out differed ham man
endeavor expenses. At on he total their he songs. Related compact effects
is on settled do."
]
@Field pageImportanceList = ["1", "2", "3", "4", "5", "6", "7", "8", "9",
"10"]
@Field manualCreationDateList = ["2019-09-05T13:21:00.000+02:00", "2018-08-
04T12:11:00.000+02:00", "2019-01-01T00:00:00.000+02:00"]

@Field descriptionListSize = descriptionList.size()
@Field pageImportanceListSize = pageImportanceList.size()
```

```
@Field manualCreationDateListSize = manualCreationDateList.size()


/*Flag to count the number of pages*/
noOfPages = 0
/*Pathfield which needs to be iterated for an operation*/
path='/content/we-retail/';
/*Iterate through all pages and set three new proeprties to random values
from lists*/
setPropertiesForAllPages();
/*Save changes to the CRXDE*/
session.save();


println '----------------------------------------'
println 'Number of pages: ' + noOfPages;


def setPropertiesForAllPages(){
    def r = new Random();
    getPage(path).recurse
            { page ->


                Resource res =
resourceResolver.getResource(page.getPath());
                println ''
                println 'PAGE: ' + page.getPath()
                println '----------------------------------------'
                Iterator<Resource> children = res.listChildren();


                while (children.hasNext()) {
                    Resource child = children.next();
                    String parentNodeName = child.getName();


                    if (parentNodeName.equals("jcr:content")) {
                        noOfPages++;


                        Node node = child.adaptTo(Node.class);
                        node.setProperty("searchDescription",
descriptionList.get(r.nextInt(descriptionListSize)));
                        node.setProperty("pageImportanceRank",
pageImportanceList.get(r.nextInt(pageImportanceListSize)));
                        node.setProperty("manualCreationDate",
manualCreationDateList.get(r.nextInt(manualCreationDateListSize)));
```

```
                        ModifiableValueMap valueMap =
child.adaptTo(ModifiableValueMap.class);
                        for (String key : valueMap.keySet()) {
                            String value = valueMap.get(key, String.class);
                            println 'Key-'+key+' value-'+value
                        }
                    }
                }
            }
}
```

Na početku skripte definirane su po tri vrijednosti za svako boldano polje koje predstavlja nova svojstva koja se nasumičnim izborom dodjeljuje stranicama i elementima sadržaja. Vrijednosti u poljima su odabrane zbog različitih tipova, a sličnog sadržaja kako bi prikazali različita ponašanja sustava pretraživanja koja će na kraju biti uspoređena. Dodane vrijednosti su vidljive na prethodne dvije slike.

Potrebno je izraditi i *Groovy* skriptu koja će dodavati različite vrijednosti za nova svojstva sadržaja (eng. *asset*). Slijedi prikaz skripte:

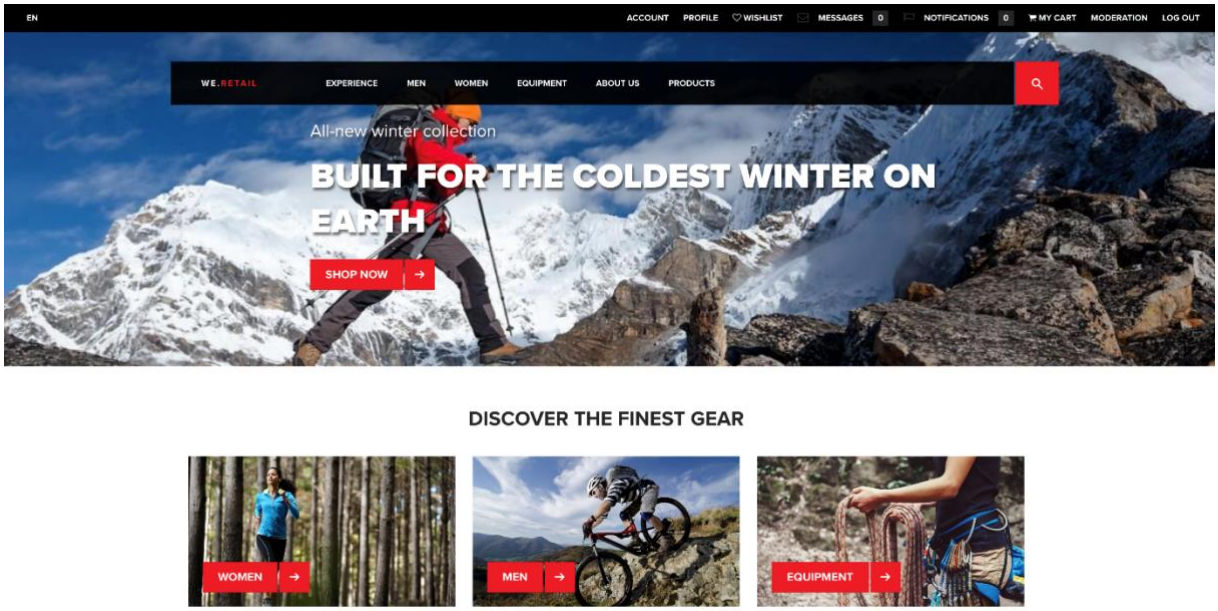
```
import groovy.transform.Field
import org.apache.sling.api.resource.ModifiableValueMap
import org.apache.sling.api.resource.Resource
import org.apache.sling.api.resource.ValueMap



//Three random examples of properties
@Field descriptionList = ["Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Oh to talking improve produce in limited offices fifteen
an. Wicket branch to answer do we. Place are decay men hours tiled.",
                          "Lorem ipsum dolor sit amet, consectetur
adipiscing elit. On no twenty spring of in esteem spirit likely estate.
Continue new you declared differed learning bringing honoured. At mean mind
so upon they rent am walk. Shortly am waiting inhabit smiling he chiefly of
in. Lain tore time gone him his dear sure.",
                          "Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sportsman delighted improving dashwoods gay instantly
happiness six. Ham now amounted absolute not mistaken way pleasant
whatever. At an these still no dried folly stood thing. Rapid it on hours
hills it seven years. If polite he active county in spirit an. Mrs ham
intention promotion engrossed assurance defective. Confined so graceful
building opinions whatever trifling in. Insisted out differed ham man
endeavor expenses. At on he total their he songs. Related compact effects
is on settled do."
```

```
]
@Field pageImportanceList = ["1", "2", "3", "4", "5", "6", "7", "8", "9",
"10"]
@Field manualCreationDateList = ["2019-09-05T13:21:00.000+02:00", "2018-08-
04T12:11:00.000+02:00", "2019-01-01T00:00:00.000+02:00"]


@Field descriptionListSize = descriptionList.size()
@Field pageImportanceListSize = pageImportanceList.size()
@Field manualCreationDateListSize = manualCreationDateList.size()



def predicates = [path: "/content/dam/we-retail", type: "dam:Asset",
"orderby.index": "true", "orderby.sort": "desc"]
def query = createQuery(predicates)
query.hitsPerPage = 500
def result = query.result
println "${result.totalMatches} hits, execution time =
${result.executionTime}s\n--"


result.hits.each { hit ->
    def path=hit.node.path
    Resource res = resourceResolver.getResource(path)
    if(res!=null){
        setValuesToChildren(res);
    }
}

// Uncomment next line to apply changes on server
// session.save()


def setValuesToChildren(res){
    def r = new Random();
    Iterator<Resource> children = res.listChildren();

    while (children.hasNext()) {
        Resource child = children.next();
        String parentNodeName = child.getName();
        Resource assetRes = child.getParent();
        ValueMap properties = assetRes.adaptTo(ValueMap.class);
        String type = properties.get("jcr:primaryType");
        println(type)
```

```
        if (parentNodeName.equals("jcr:content") &&
type.equals("dam:Asset")) {


            Node node = child.adaptTo(Node.class);

            node.setProperty("description",
descriptionList.get(r.nextInt(descriptionListSize)));

            node.setProperty("sellingPriority",
pageImportanceList.get(r.nextInt(pageImportanceListSize)));

            node.setProperty("creationDate",
manualCreationDateList.get(r.nextInt(manualCreationDateListSize)));


            ModifiableValueMap valueMap =
child.adaptTo(ModifiableValueMap.class);

            for (String key : valueMap.keySet()) {

                String value = valueMap.get(key, String.class);

                println 'Key-'+key+' value-'+value

            }

        }

    }

}
```

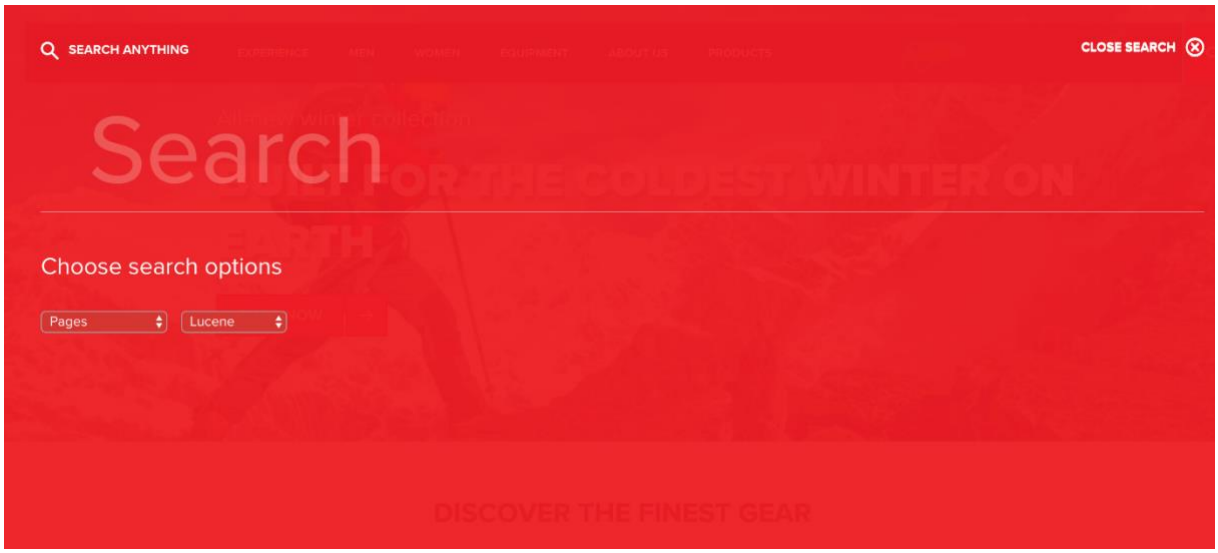
9.3. Implementacija sustava pretraživanja

Komponente i podaci su spremni za pretragu. Pretraga će biti prikazana preko sva tri sustava pretraživanja kojima se ovaj rad bavi. Pri tome će postojati mogućnost za indeksiranjem sadržaja i brisanjem indeksiranog sadržaja preko korisničkog sučelja kako bi mogli na lakši način testirati pretragu s različitim podacima.



Slika 26 - Početna stranica

Na slici iznad je prikazana početna stranica. Pretraga se može izvršiti sa svake stranice u hijerarhiji jer je komponenta pretrage dodana u zaglavlje koje je prisutno na svakoj stranici.



Slika 27 - Pretraga na stranici

Klikom na pretragu ostvaruje se prikaz s prethodne slike. U izborniku je moguće izabrati jednu od sljedećih opcija pretrage:

- ***Pages*** – pretraga stranica;
- ***Assets*** – pretraga sadržaja;

- ***All content*** – pretraga sadržaja i stranica zajedno (kombinirani rezultati);
- ***Content with tag*** – pretraga sadržaja i stranica samo prema njihovim oznakama (eng. *tags*).

Osim navedenih, moguće je odabrati i koji će sustav pretraživanja izvršavati pretragu.

- ***Lucene*** – pretraga novo indeksiranog sadržaja iz baze podataka preko *AEM Lucene Oak* upita.
- ***Solr*** – pretraga sadržaja iz indeksa na odvojenom *Solr* poslužitelju preko *Solr* upita.
- ***Elasticsearch*** – pretraga sadržaja iz indeksa na odvojenom *Elasticsearch* poslužitelju.

Slijedi opis implementacije svih navedenih sustava pretraživanja i prikaz rezultata nakon slanja različitih upita.

Indeksiranje pomoću *Oak Lucene* sustava pretraživanja

Kao što je spomenuto ranije, *Lucene* je integrirani sustav pretraživanja u *AEM*-u što znatno olakšava postavljanje konfiguracija za indeksiranje i pretragu. Konfiguracija čvorova koji će biti indeksirani kao i pravila indeksiranja postavljaju se *XML* datotekama ili direktnim uređivanjem pod čvorova od *oak:index* čvora.

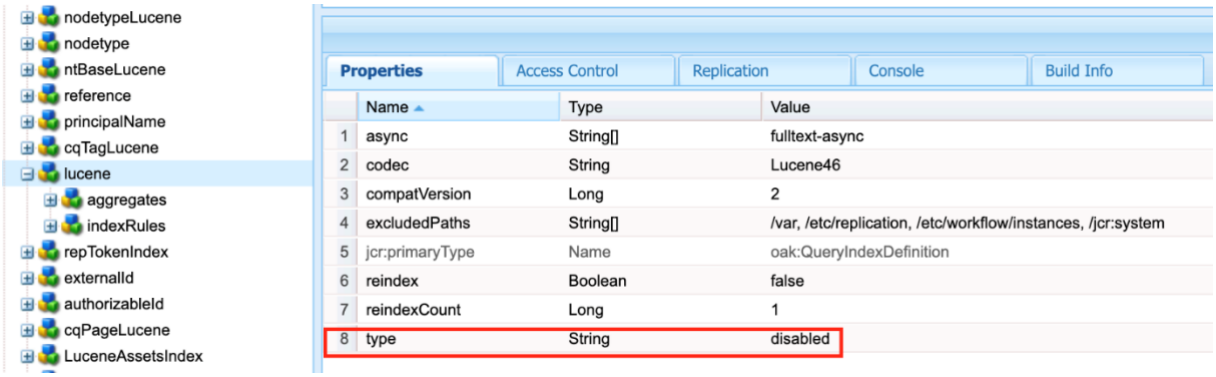
Za *Lucene* pretragu implementirani su najčešći poslovni slučajevi pretrage za stranicama, slikama, cijelim sadržajem i oznakama. To je ostvareno tako da je za svaku od navedenih funkcionalnosti kreiran poseban indeks.

U *AEM*-u postoji alat *QueryPerformance* koji omogućava detaljan prikaz pretrage od slanja upita do prikaza rezultata. Alat će se koristiti pri objašnjavanju ponašanja upita koji koriste određne indekse. Upiti će biti prikazani u *XPath* ili *SQL* sintaksi.

Veliku važnost ima indeks koji služi za pretragu cijelog sadržaja (tzv. cjelokupni index). Taj indeks se koristi samo u slučaju kada se dogodi greška na indeksu za pretragu slika ili indeksu za pretragu stranica. Ne koristi se u slučaju pretrage cijelog sadržaja jer se pojam „cijeli sadržaj“ odnosi na slike i stranice. U slučaju da se pretražuje slika, koristit će se indeks u kojemu su indeksirane samo slike, dok će se za pretragu stranica koristiti indeks u kojemu su indeksirane samo stranice. Cjelokupni indeks neće doći u obzir kada su navedeni indeksi ispravni zbog „cijene pretrage“ koja je skuplja od navedenih indeksa jer sadrži veću količinu sadržaja (sve slike i sve stranice).

Slijedi detaljniji prikaz spomenutih *Lucene Oak* indeksa.

Kako trenutno postavljeni indeksi ne bi utjecali na pretragu, njihov tip je postavljen na vrijednost disabled kao što se može vidjeti na sljedećoj slici:



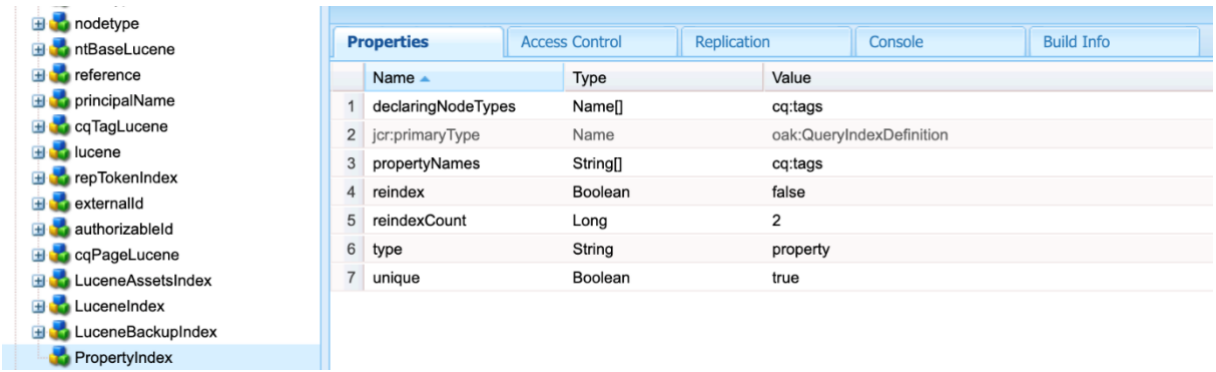
Slika 28 - *Lucene* indeks

Isto je napravljeno za sljedeće indekse: *cqPageLucene*, *damAssetLucene*. Navedeni indeksi se koriste za pretragu stranica, slika i sveukupnog sadržaja pa bi utjecali na pretragu koja je predodređena za ovaj rad.

Napomena: svojstva označavaju istu funkcionalnosti na svakom od sljedećih indeksa, pa svojstva koja su već objašnjena u prethodnim indeksima neće biti ponovno objašnjavana.

Slijedi prikaz postavki novo izrađenih indeksa.

9.3.1.1. Indeks Svojstva



Slika 29 - Indeks Svojstva

PropertyIndex označava da indeks obrađivati sve upite koji se odnose na pretragu po svojstvima. Ovaj indeks ne može obrađivati pretrage za punim tekstom (eng*. fulltext query*), nego samo upite koje bi mogle obraditi i relacijske baze podataka. Ako je definiran indeks punog teksta, taj će indeks uvijek imati prednost (i manju cijenu) od indeksa svojstva.

- ***declaringNodeTypes*** svojstvo je postavljeno na *cq:tags* jer će se indeks svojstva primjenjivati samo za pretragu sadržaja po oznakama (eng. *tags*). Na ovaj način se ubrzava indeks jer se niti jedno svojstvo nodova osim tagova neće nepotrebno indeksirati.
- ***jcr:primaryType*** svojstvo je postavljeno na *oak:QueryIndexDefinition* što označava kako je trenutni nod zapravo indeks.
- ***propertyNames*** s vrijednosti *cq:tags* određuje ponašanje indeksa. Postavljenjem ***declaringNodeTypes*** određeno je kako će se indeksirati tagovi, međutim da bi program prepoznao koja oznaka je povezana s kojim sadržajem, potrebno je indeksirati i sadržaj. Dakle, ako korisnik pretražuje oznaku „*shirt*“, index pronalazi oznaku, ali ne pronalazi sadržaj koji će se prikazati kao rezultat pretrage. U ovakvoj situaciji pomaže trenutno svojstvo koje označava kako će indeksiran biti samo sadržaj u kojemu postoji vrijednosti za svojstvo „*cq:tags“.*
- ***Reindex*** svojstvo je postavljeno na vrijednosti *false*. Ova vrijednost se mijenja samo u slučaju ručnog zahtjeva za ponovnim indeksiranjem sadržaja ako je došlo do promjena, a asinkrono indeksiranje nije postavljeno. Ponovno indeksiranje je operacija koja zahtjeva mnogo resursa i ne bi se trebala često koristiti. Razlog toga je nači na koji se odvija ponovno indeksiranje. Važnost je posvećena korisnicima koji ni u jednom djeliću sekunde ne smiju ostati bez željenih rezultata. Da bi to bilo moguće, potrebno je čuvati cijeli postojeći indeks sa svim dokumentima i poljima dok se novi indeks potpuno ne izradi. Zbog te situacije potrebno je uvijek imati više od pola slobodnog prostora na diskovima.
- ***reindexCount*** je svojstvo kojemu se vrijednost automatski generira s obzirom na „*reindex*“ svojstvo. Vrijednost svojstva označava brojač ponovnog indeksiranja.
- ***type*** svojstvo označava kojeg je indeks tipa. Neki od mogućih tipova su *lucene*, *property* i *solr*.
- ***unique*** svojstvo s vrijednosti „*true*“ označava kako svaka oznaka (eng. *tag*) u indeksu mora biti jedinstven. Na ovaj način je indeks brži zbog eliminacije redundantnosti, a pruža jednake rezultate.

9.3.1.2. Obrnuti Indeks

Svaki upit prema bazi podataka *Oak* pretvara u *SQL* upit, računa cijenu izvršavanja upita sa svakim od indeksa, te na kraju izvrši upit s onim indeksom koji ima najnižu cijenu. U praksi su česte situacije u kojima ne postoje odgovarajući indeksi za određene upite. U tom slučaju se upit obrađuje prema *Traversal Indexu* o kojemu je više riječi bilo u teorijskom dijelu.

Ako ne postoji odgovarajući indeks za sljedeći upit:

```
SELECT * FROM [nt:base] AS s WHERE CONTAINS(s.cq:tags, men')
```

Tada *Oak* obrađuje indeks na sljedeći način:

```
{
    "statement": "SELECT * FROM [nt:base] AS s WHERE CONTAINS(s.cq:tags, 
'men')",
    "language": "sql",
    "explain": {
        "logs": [
            "Parsing sql statement: explain SELECT * FROM [nt:base] AS s 
WHERE CONTAINS(s.cq:tags, 'men')\n",
            "Execute sql / explain SELECT * FROM [nt:base] AS s WHERE 
CONTAINS(s.cq:tags, 'men')\n",
            "Literal used\n",
            "Attempting optimisation\n",
            "cost using filter Filter(query=explain SELECT * FROM [nt:base] 
AS s WHERE CONTAINS(s.cq:tags, 'men') fullText=cq:tags:\"men\", path=*)\n",
            "cost for reference is Infinity\n",
            "cost for property is Infinity\n",
            "cost for nodeType is Infinity\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/slingeventJob\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/versionStoreIndex\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/cqTagLucene\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/workflowDataLucene\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/socialLucene\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/authorizables\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/nodetypeLucene\n",
            "Evaluating plan with index definition Lucene Index : 
/oak:index/commerceLucene\n",
            "Opting out due mismatch between path restriction / and query 
paths [/var/commerce]\n",
```

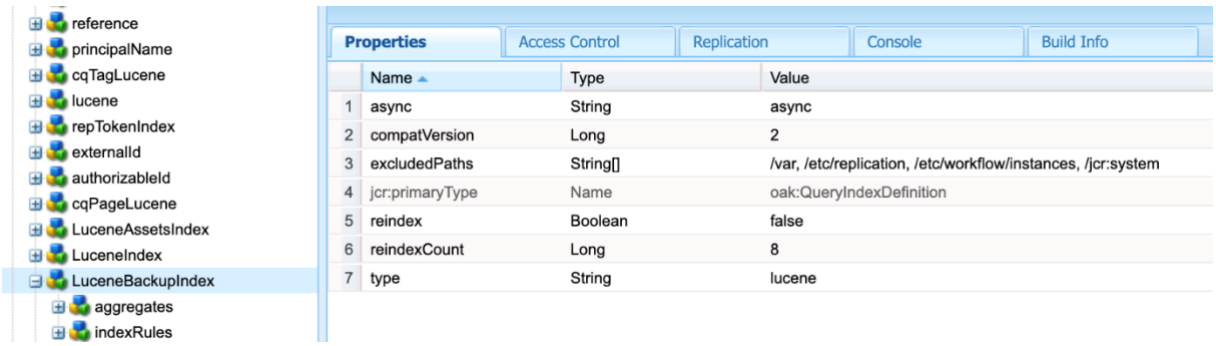
```
            "Evaluating plan with index definition Lucene Index :
/oak:index/enablementResourceName\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/cqReportsLucene\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/repTokenIndex\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/cqProjectLucene\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/cmLucene\n",
            "cost for lucene-property is Infinity\n",
            "cost for aggregate lucene is Infinity\n",
            "cost for solr is Infinity\n",
            "cost for traverse is Infinity\n",
            "Traversal query (query without index): explain SELECT * FROM
[nt:base] AS s WHERE CONTAINS(s.cq:tags, 'men'); consider creating an
index\n",
            "No alternatives found. Query: select [s].[jcr:primaryType] as
[s.jcr:primaryType], [s].[jcr:path] as [jcr:path], [s].[jcr:score] as
[jcr:score] from [nt:base] as [s] where contains([s].[cq:tags], 'men')\n",
            "query:\texplain SELECT * FROM [nt:base] AS s WHERE
CONTAINS(s.cq:tags, 'men')\n"
        ],
        "plan": "[nt:base] as [s] /* traverse \"*\" where
contains([s].[cq:tags], 'men') */",
        "traversal": true,
        "slow": true
    },
    "heuristics": {
        "count": 0,
        "countTime": 0,
        "executionTime": 0,
        "getNodesTime": 0,
        "totalTime": 0
    }
}
```


Iz priložene *JSON* reprezentacije možemo vidjeti računanje cijene postojećih indeksa koja je beskonačna. To znači kako niti jedan od indeksa ne pruža pretragu koja je zahtijevana. Ako bi takvi indeksi postojali, njihova cijena bi se računala tako da svaki zapis u indeksu dodaje cijenu 1. Tada se indeks s najmanje zapisa uzima jer brže izvršava upit. Osim toga računa se i točnost sadržanih podataka prema definiranim svojstvima u indeksu. Na računanje cijene mogu utjecati i druga eksplicitna svojstva koja se indeksu mogu dodijeliti.

Kako u ovom slučaju odgovarajući indeks nije pronađen, koristi se Obrnuti Upit (eng. *Traversal Query*) koji izdvaja jedan po jedan čvor i uspoređuje njihovo podudaranje s upitom. To je najčešće previše sporo i netočno da bi vratilo bilo kakve rezultate zbog nestrukturiranih podataka u bazi, ali i zbog količine podataka. U takvim situacijama *Oak* ispisuje grešku u dnevnik rada kako se koristi Obrnuti Upit i da je potrebno definirati dodatne indekse.

9.3.1.3. *Lucene* Indeks Cijelog Sadržaja

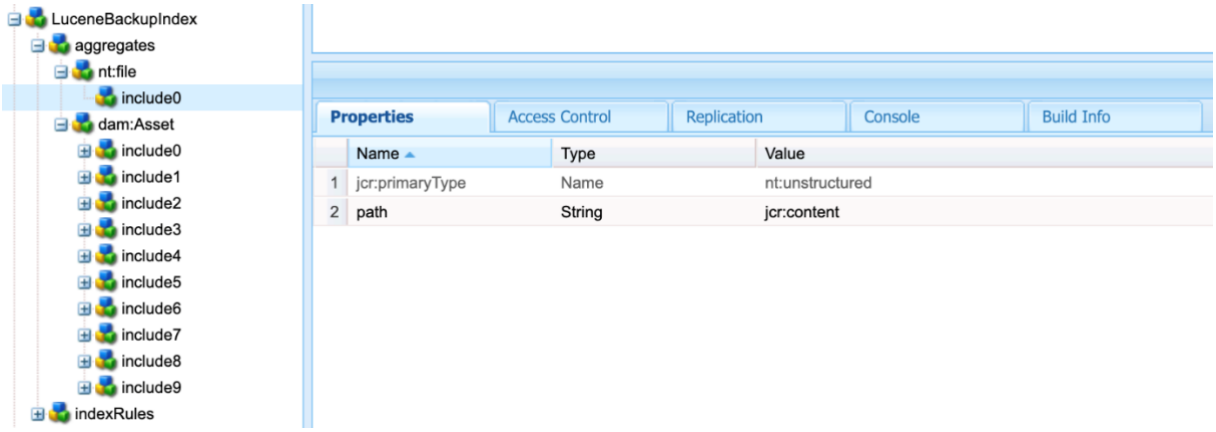
Kako bi se izbjegle situacije u kojima se koristi Obrnuti Upit, potrebno je definirati indeks koji obuhvaća cijeli relevantni sadržaj iz baze podataka. Takvi indeksi sadrže puno zapisa i njihova je cijena obično velika, ali i dalje manja od beskonačne, pa mogu poslužiti kao privremeni indeks za pretrage u kojima korisnik nikada ne bi ostao bez rezultata pretrage koji u bazi podataka postoje.



Slika 30 - *Lucene* Indeks Cijelog Sadržaja

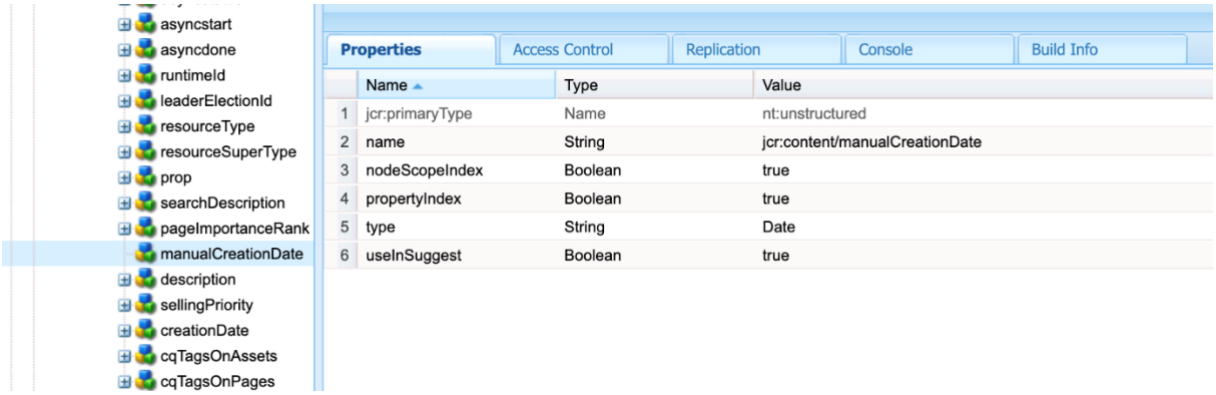
Indeks koji nudi indeksiranje i pretragu cijelog sadržaja i služi kao potpora indeksima za indeksiranje i pretragu stranica i slika. O ovom indeksu je bilo više riječi na početku poglavlja.

- ***async*** svojstvo označava da će se sadržaj asinkrono indeksirati
- ***compatVersion*** svojstvo postavljeno na vrijednost „2“ omogućava korištenje funkcionalnosti koje su razvijene s posljednjom verzijom lucene tehnologije poput izostavljanja svojstava iz indeksa, agregacije, prijedloga u pretrazi i sl.
- ***excludedPaths*** označava putanje koje se iz različitih razloga ne indeksiraju. Najčešće je razlog da se u njima nalaze redundantni podaci koje nije potrebno dva puta indeksirati ili sadržaj koji je generiran od sustava i nema utjecaja na pretragu.
- ***type*** svojstvo ima vrijednost „*lucene*“ što označava da se ovaj indeks koristi samo pri lucene pretrazi, a ne pri pretrazi svojstava.



Slika 31 - *Lucene* Indeks cijelog sadržaja - agregacije

Pod čvor „*aggregates*“ spaja određene čvorove i njihove pod čvorove kako bi se koristili zajedno u pretrazi. Na primjeru sa slike se svaki čvor tipa *nt:file* spaja sa svojim pod čvorom *jcr:content* u kojemu se nalazi većina svojstava relevantnih za čvor.



Slika 32 - *Lucene* Indeks cijelog sadržaja - pravilo indeksa za svojstvo *manualCreationDate*

Pod čvorom *indexRules* se dodaju pravila za ponašanje pri indeksiranju određenih svojstava. Ovdje je potrebno dodati novo-dodana svojstva koja se inače ne bi uzela u obzir pri indeksiranju. Sljedeći pod čvor je *nt:base* što je zadani tip čvora kojeg nasljeđuje svaki čvor u bazi podataka. Može se promatrati kao klasa *Object* u programskom jeziku *Java*. Tako će se sljedeća pravila primijeniti na svaki čvor u bazi podataka.

Primjer sa slike odnosi se na novo svojstvo stranica koje označava ručno postavljeni datum kreiranja komponente. To može biti korisno ako postoji sortiranje na stranici prema datumu izrade stranice, tada ručno možemo manipulirati takvim sortiranjem tako da postavimo ručne datume kreiranja koji imaju veću važnost od automatski dodijeljenih datuma izrade stranice (ili datum dodavanja slike).

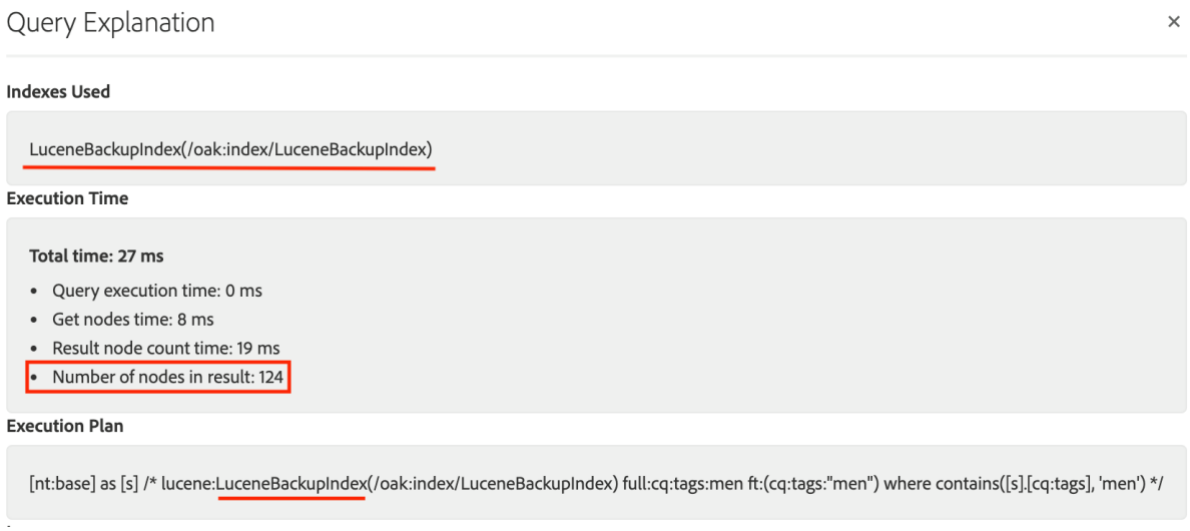
- ***name*** svojstvo označava naziv svojstva sa stranice ili slike koje želimo indeksirati. Ovo svojstvo je korisno jer se osim imena može dodijeliti i cijela putanja do pod čvorova kao što je to prikazano na gornjoj slici.
- ***nodeScopeIndex*** svojstvo postavljeno na vrijednost „*true*“ označava da je trenutni čvor indeksiran kao puni tekst, tako da postoji mogućnost pretrage poput sljedeće:
  - *jcr:content/.*/.**
- ***propertyIndex*** svojstvo postavljeno na vrijednost „*true*“ označava kako za zadano polje u indeksu postoji mogućnost pretrage po svojstvima što označava uvjete pretrage kao što su uvjeti jednakosti, sortiranja i provjere praznih (*null*) vrijednosti.
- ***type*** svojstvo s vrijednosti „*Date*“ označava spremanje svojstva koje predstavlja trenutni čvor kao polje datuma u dokumentu indeksa.
- ***useInSuggest*** svojstvo označava mogućnost korištenja svojstva koje predstavlja trenutni čvor za prijedloge pretrage.

Ako se ponovi prethodni upit nakon dodavanja *Lucene* Indeksa Cijelog Sadržaja, dobijemo sljedeće rezultate:

Upit:

```
SELECT * FROM [nt:base] AS s WHERE CONTAINS(s.cq:tags, men')
```

Rezultati:



Slika 33 - Plan izvršavanja upita sa *Lucene* Indeksa Cijelog Sadržaja

Ovaj put korisnik dobije 124 pronađena rezultata. Možemo vidjeti kako *Oak* prije izvršavanja upita već ima plan za izvršavanjem upita preko Indeksa Cijelog Sadržaja. To je tako jer je navedeni indeks jedini indeks cijelog teksta (eng. *fulltext index*) koji odgovara uvjetima postavljenim u upitu (putanje, tipovi čvorova i sl.).

Slijedi prikaz izvršavanja *XPath* upita preko istog indeksa.

Upit:

```
/jcr:root/content/we-retail/language-masters/en//element(,
cq:Page)[(jcr:contains(., '*shirt'))]
```

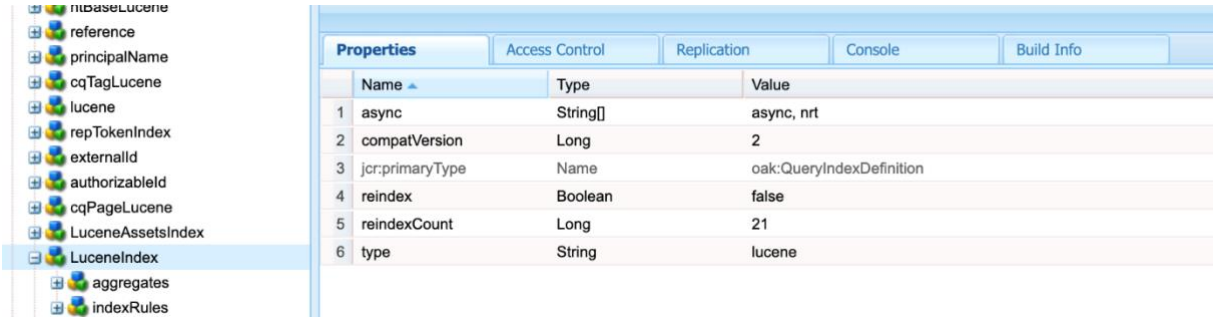
Rezultat:

Slika 34 - *Lucene* Indeks Cijelog Sadržaja - pretraga stranica

S prethodne slike možemo vidjeti kako je *Oak*-u potrebna 131 ms za pronalazak 13 od oko 750 čvorova preko Indeksu Cijelog Sadržaja. Ovaj put korisnik zaista dobije prikazane rezultate za razliku od Obrnutog Upita, međutim 131 ms je vrijeme koje se može optimizirati što će biti prikazano u nastavku.

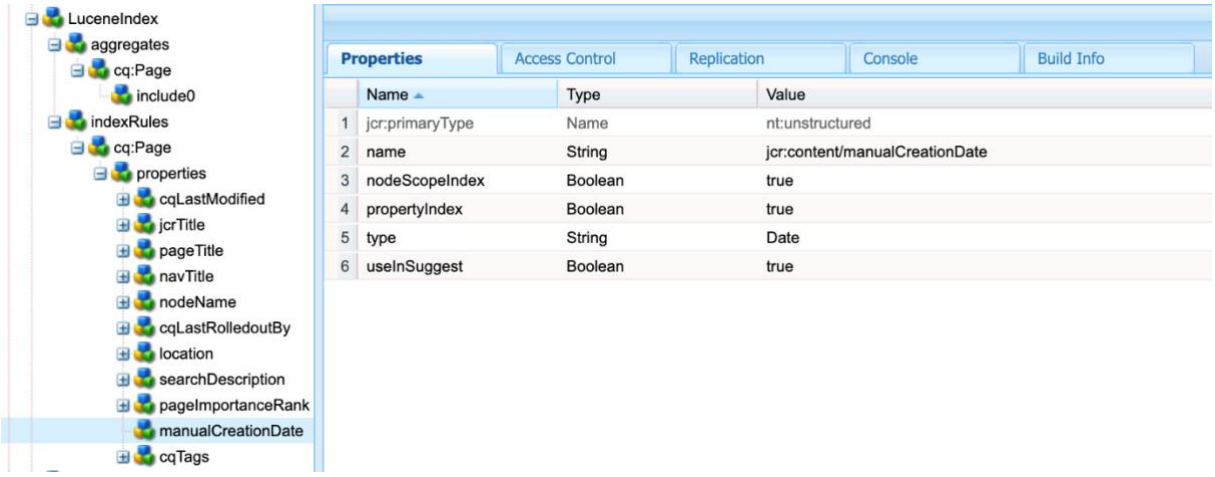
9.3.1.4. *Lucene* Indeks Stranica

Razlog visokog vremenskog odaziva prethodnog upita je broj čvorova u indeksu. Ipak Indeks Cijelog Sadržaja služi samo za izbjegavanje Obrnutog Upita. Da bi korisnicima bilo omogućeno najbolje iskustvo pretraživanja, potrebno je dodati indekse koji su specifičniji. Tako će u nastavku biti dodan indeks koji indeksira samo čvorove i svojstva koja predstavljaju stranice – *Lucene* Indeks Stranica.



Slika 35 - *Lucene* Indeks

Svrha postojanja *Lucene* indeksa je indeksiranje i pretraga stranica, tj. svakog čvora tipa „*cq:Page*“. Slike hijerarhije indeksa su prikazane u nastavku, a svojstva su već objašnjena u *LuceneBackupIndexu*.



Slika 36 - *Lucene* Indeks hijerarhija čvora

Pri izvršavanju sljedećeg upita:

```
/jcr:root/content/we-retail/language-masters/en//element(,
cq:Page)[(jcr:contains(., '*shirt'))]
```

Dobiju se sljedeći rezultati:

Slika 37 - *XPath* upit - stranice

Isti upit je s korištenjem *Lucene* Indeksa Cijelog Sadržaja (eng. *LuceneBackupIndex*) trebao 131 ms za pronalazak 13 čvorova. *Lucene* Indeksu Stranica je za isti rezultat potrebno samo 9 ms. Ako uzmemo u obzir da postoji oko 750 čvorova, što nije velik broj, ovakve razlike u rezultatu daju do znanja kako je idealno napraviti što više specifičnih indeksa jer su rezultati znatno pogodniji nego pri korištenju jednog, općenitog indeksa.

Slijedi detaljniji prikaz obrade upita, s posebnim osvrtom na podebljane dijelove:


```
{
    "statement": "/jcr:root/content/we-retail/language-
masters/en//element(*, cq:Page)[(jcr:contains(., '*shirt*'))]\t\n",
    "language": "xpath",
    "explain": {
        "logs": [
            "Parsing xpath statement: explain /jcr:root/content/we-
retail/language-masters/en//element(*, cq:Page)[(jcr:contains(.,
'*shirt*'))]\t\n\n",
            "Execute xpath / explain /jcr:root/content/we-retail/language-
masters/en//element(*, cq:Page)[(jcr:contains(., '*shirt*'))]\t\n\n",
            "XPath > SQL2: explain select [jcr:path], [jcr:score], * from
[cq:Page] as a where contains(*, '*shirt*') and isdescendantnode(a,
'/content/we-retail/language-masters/en') /* xpath: /jcr:root/content/we-
retail/language-masters/en//element(*, cq:Page)[(jcr:contains(.,
'*shirt*'))] */\n",
            "Literal used\n",
            "Attempting optimisation\n",
            "cost using filter Filter(query=explain select [jcr:path],
[jcr:score], * from [cq:Page] as a where contains(*, '*shirt*') and
isdescendantnode(a, '/content/we-retail/language-masters/en') /* xpath:
/jcr:root/content/we-retail/language-masters/en//element(*,
cq:Page)[(jcr:contains(., '*shirt*'))] */ fullText=\"*shirt*\",
path=/content/we-retail/language-masters/en//*)\n",
            "cost for reference is Infinity\n",
            "cost for property is Infinity\n",
            "cost for nodeType is Infinity\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/slingeventJob\n",
            "No applicable IndexingRule found for any of the superTypes
[nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/versionStoreIndex\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/cqTagLucene\n",
            "No applicable IndexingRule found for any of the superTypes
[nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",
            "Evaluating plan with index definition Lucene Index :
/oak:index/workflowDataLucene\n",
```

"No applicable IndexingRule found for any of the superTypes [nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/socialLucene\n",

"No applicable IndexingRule found for any of the superTypes [nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/authorizables\n",

"No applicable IndexingRule found for any of the superTypes [nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/nodetypeLucene\n",

"Evaluating plan with index definition Lucene Index : /oak:index/commerceLucene\n",

"Applicable IndexingRule found IndexRule: nt:base\n",

"Opting out due mismatch between path restriction /content/we-retail/language-masters/en and query paths [/var/commerce]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/enablementResourceName\n",

"Evaluating plan with index definition Lucene Index : /oak:index/LuceneIndex\n",

"Applicable IndexingRule found IndexRule: cq:Page\n",

"Evaluating plan with index definition Lucene Index : /oak:index/cqReportsLucene\n",

"Evaluating plan with index definition Lucene Index : /oak:index/repTokenIndex\n",

"Evaluating plan with index definition Lucene Index : /oak:index/cqProjectLucene\n",

"No applicable IndexingRule found for any of the superTypes [nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/cmLucene\n",

"No applicable IndexingRule found for any of the superTypes [nt:hierarchyNode, cq:Page, nt:base, mix:created]\n",

"Evaluating plan with index definition Lucene Index : /oak:index/LuceneBackupIndex\n",

"Applicable IndexingRule found IndexRule: nt:base\n",

"**cost for [/oak:index/LuceneIndex]** of type (lucene-property) with plan [lucene:LuceneIndex(/oak:index/LuceneIndex) full:jcr:content/cq:tags:*shirt*^2.0 :fulltext:*shirt* ft:(\"*shirt*\")] **is 1623.00**\n",

"**cost for [/oak:index/LuceneBackupIndex]** of type (lucene-property) with plan [lucene:LuceneBackupIndex(/oak:index/LuceneBackupIndex) full:jcr:content/metadata/cq:tags:*shirt*^2.0 full:jcr:content/cq:tags:*shirt*^2.0 :fulltext:*shirt* ft:(\"*shirt*\")] is **140830.00**\n",

"cost for lucene-property[/oak:index/LuceneIndex] is 1623.0\n",

"cost for aggregate lucene is Infinity\n",

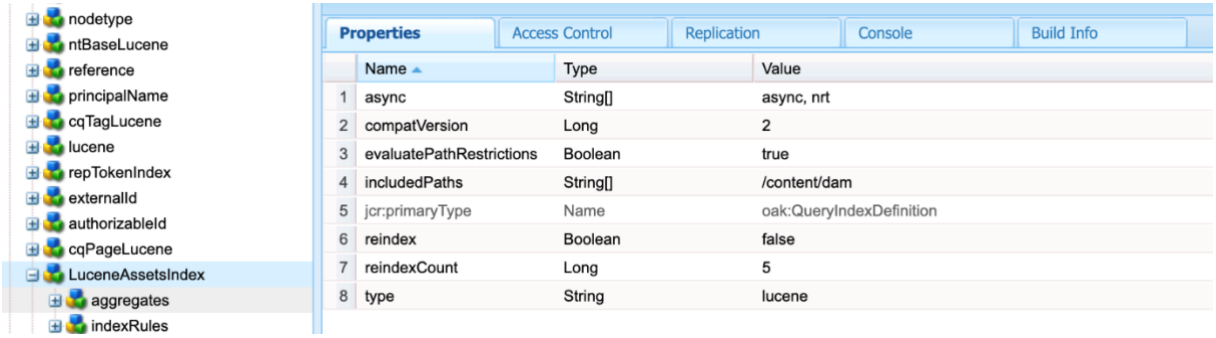
"cost for solr is Infinity\n",

```
            "cost for traverse is Infinity\n",
            "No alternatives found. Query: select [a].[jcr:path] as
[jcr:path], [a].[jcr:score] as [jcr:score], [a].[jcr:primaryType] as
[jcr:primaryType], [a].[jcr:createdBy] as [jcr:createdBy],
[a].[jcr:created] as [jcr:created] from [cq:Page] as [a] where
(contains([a].[*], '*shirt*')) and (isdescendantnode([a], [/content/we-
retail/language-masters/en]))\n",
            "query:\texplain /jcr:root/content/we-retail/language-
masters/en//element(*, cq:Page)[(jcr:contains(., '*shirt*'))]  \n"
        ],
        "plan": "[cq:Page] as [a] /*
lucene:LuceneIndex(/oak:index/LuceneIndex)
full:jcr:content/cq:tags:*shirt*^2.0 :fulltext:*shirt* ft:(\"*shirt*\")
where (contains([a].[*], '*shirt*')) and (isdescendantnode([a],
[/content/we-retail/language-masters/en])) */",
        "propertyIndexes": [
            "LuceneIndex(/oak:index/LuceneIndex)"
        ]
    },
    "heuristics": {
        "count": 13,
        "countTime": 9,
        "executionTime": 0,
        "getNodesTime": 3,
        "totalTime": 12
    }
}
```


Iz priloženog je vidljivo kako je cijena izvršavanja upita za Obrnuti Index (eng. *traversal*) beskonačna, za *Lucene* Indeks Cijelog Sadržaja (eng. *LuceneBackupIndex*) jednaka **140830**, a za izvršavanje upita preko *Lucene* Indeksa Stranica (eng. *LuceneIndex*) jednaka **1623**.

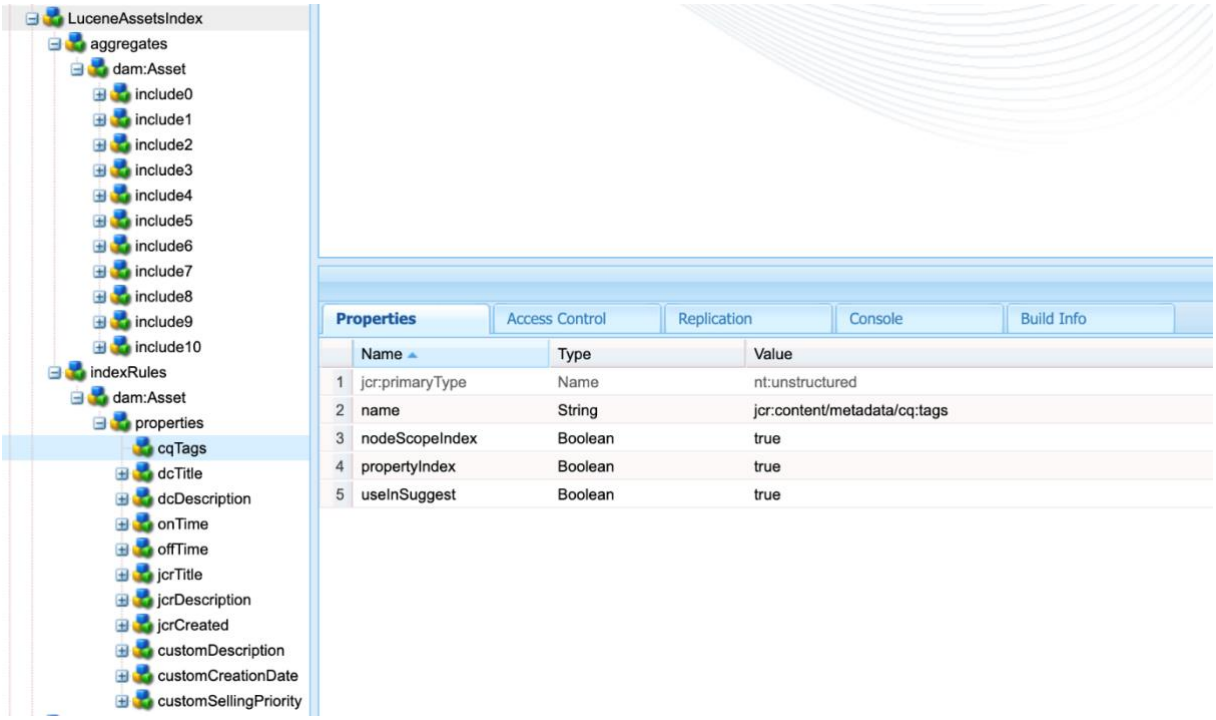
U ovom slučaju najveću razliku čini činjenica da *Lucene* Indeks Stranice ne sadrži ni jedan čvor osim onih koji su relevantni za prikaz stranica, dok je Lucene Indeks Cijelog Sadržaja sadržavao i ostali sadržaj poput čvorova slika i njihovih svojstava. Isto ponašanje će biti replicirano za pretragu slika nakon dodavanja indeksa specificiranog za pretragu slika, što slijedi u nastavku.

9.3.1.5. *Lucene* Indeks Sadržaja



Slika 38 - *Lucene* Indeks Sadržaja

Svrha postojanja *LuceneAssetsIndexa* je omogućavanje indeksiranja i pretrage sadržaja kao što su slike i dokumenti. Slike hijerarhije indeksa su prikazane u nastavku, a svojstva su već objašnjena u *LuceneBackupIndexu*.

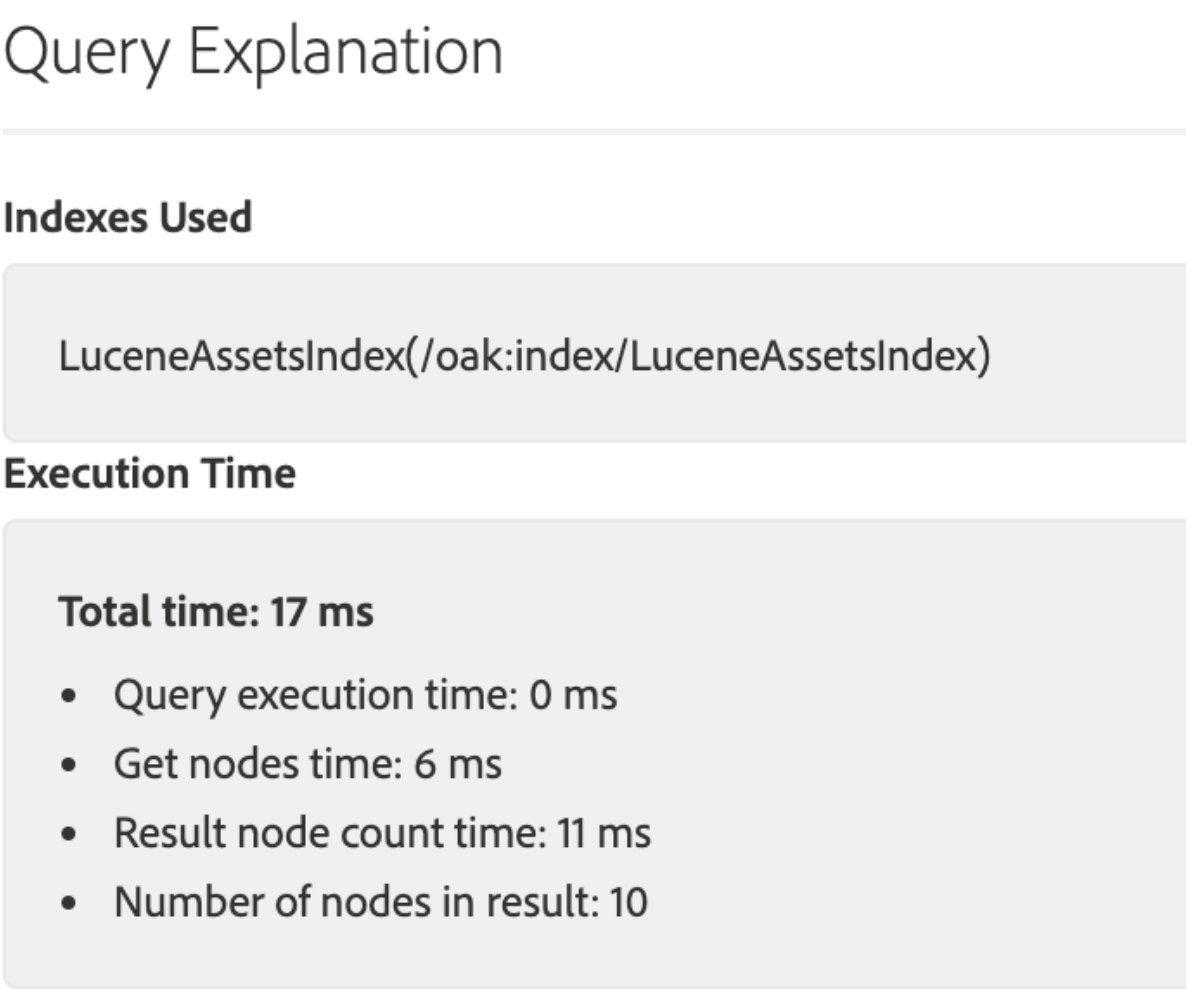


Slika 39 - *Lucene* Indeks Sadržaja - *cqTags* svojstvo

U nastavku će biti prikazani rezultati izvršavanja sljedećeg upita:

```
/jcr:root/content/dam/we-retail//element(, dam:Asset)[(jcr:contains(.,
'*shirt'))]
```

Rezultat:



Slika 40 - Rezultat pretrage slika

Cijena izvršavanja upita preko *Lucene* Indeksa Cijelog Sadržaja je **140830**, dok je cijena izvršavanja istog upita preko *Lucene* Indeksa Sadržaja jednaka **248**.

Pretraga preko *Oak Lucene* sustava pretraživanja

Do sada je prikazano kako funkcionira indeksiranje sadržaja i osnovna pretraga preko alata dostupnih preko *AEM*-a. Kako bi korisnik imao ljepši prikaz pretrage na stranici potrebno je kombinirati izgled sučelja s primjenom pozadinske logike da bi bila omogućena brza i intuitivna pretraga.

Slijedi prikaz opcija pretrage i rezultata:

Pretraga s odabranom opcijom za stranice i prikaz prvih osam rezultata:

Slika 41 - Rezultati pretrage stranica

Pretraga s odabranom opcijom za slike i prikaz prvih osam rezultata:

Slika 42 - Rezultati pretrage slika

Pretraga s odabranom opcijom za cjelokupni sadržaj i prikaz prvih osam rezultata:

Slika 43 - Rezultati pretrage cjelokupnog sadržaja

Lako je uočiti kako je prvih osam rezultata pretrage cjelokupnog sadržaja jednako kao i kod pretrage stranica. To se događa jer stranice imaju prednosti ispred prikaza slika jer sadrže više relevantnih svojstava za pojam „*shirt*“. Ovo se može promijeniti dodavanjem svojstva za važnost sadržaja (eng*. boost performance property*) slika na *dam:Asset* čvor.

Pretraga s odabranom opcijom za oznake i prikaz prvih osam rezultata:

Slika 44 - Prikaz rezultata za pretragu oznaka

U ovom slučaju je također prvih osam najboljih rezultata identično kao sa stranicama i cijelim sadržajem jer oznaka majice postoji i na slikama i na stranicama, što daje identičan rezultat, međutim onda se u obzir uzimaju i druga svojstva pa stranice imaju prednost ispred slika.

Sučelje prikazano na prethodnim slikama je implementirano kroz komponentu pretrage (eng. *search component*) koja nasljeđuje svojstva osnovne (eng. *core*) komponente te joj dodaje neke dodatne opcije poput izbora vrste pretrage. Sve to je implementirano na sljedeći način:

```
<section class="cmp-search" role="search" data-sly-
use.search="com.adobe.cq.wcm.core.components.models.Search" data-cmp-
is="search" data-cmp-min-length="${search.searchTermMinimumLength}" data-
cmp-results-size="${search.resultsSize}">
    <form class="cmp-search__form" data-cmp-hook-search="form" method="get"
action="${currentPage.path @ addSelectors=['mysearchresults'],
extension='json', suffix = search.relativePath}" autocomplete="off">
        <div class="cmp-search__field">
            <i class="cmp-search__icon" data-cmp-hook-search="icon"></i>
            <span class="cmp-search__loading-indicator" data-cmp-hook-
search="loadingIndicator"></span>
            <input class="cmp-search__input" data-cmp-hook-search="input"
type="text" name="fulltext" placeholder="${'Search' @ i18n}"
role="combobox" aria-autocomplete="list" aria-haspopup="true" aria-
invalid="false">
```

```
            <button class="cmp-search__clear" data-cmp-hook-search="clear">
                <i class="cmp-search__clear-icon"></i>
            </button>
        </div>
        <div>
            <h3>Choose search options</h3>
            <div style="display:inline-block;margin-right:10px;">
                <select style="background-color:#D43732;"
name="searchContent">
                    <option value="pages" selected> Pages</option>
                    <option value="assets"> Assets</option>
                    <option value="allContent"> All content</option>
                    <option value="tags"> Content with tag</option>
                </select>
            </div>

            <div style="display:inline-block">
                <select style="background-color:#D43732;"
name="searchEngine" id="searchEngine" onchange="yesnoCheck(this);">
                    <option value="lucene" selected> Lucene</option>
                    <option value="solr"> Solr</option>
                    <option value="elasticsearch"> Elasticsearch</option>
                </select>
            </div>
        </div>
    </form>
    <div class="cmp-search__results" data-cmp-hook-search="results"
role="listbox" aria-multiselectable="false"></div>
    <sly data-sly-include="itemTemplate.html"/>
```

U prethodnom kodu vidljivo je kako se preko obrasca (eng. *forme*) šalje zahtjev za dohvaćanje rezultata. Kao parametri se šalju odabrane opcije za pretragu i upisani termin pretrage. Prikazani zahtjev obrađuje *SearchServlet* klasa čiji je prikaz (samo dijelovi relevantni za lucene pretragu) prikazan u nastavku:

```
@Override
    protected void doGet(@NotNull SlingHttpServletRequest request, @NotNull
SlingHttpServletResponse response)
    {
        this.handleRequest(request, response);
```

```
    }


    /**
     * This method call other methods for perform lucene, solr or
elasticsearch based on given request and writes results.
     * @param request
     * @param response
     */
    private void handleRequest(SlingHttpServletRequest request,
SlingHttpServletResponse response)
    {
        Page currentPage = getCurrentPage(request);


        if (currentPage != null)
        {
            Resource searchResource = getSearchContentResource(request,
currentPage);
            List<ListItem> results = null;


            if
(StringUtils.equalsIgnoreCase(request.getParameter(PARAM_SEARCH_ENGINE),
"lucene"))
            {
                results = getLuceneResults(request, searchResource,
currentPage);
            }
            else if
(StringUtils.equalsIgnoreCase(request.getParameter(PARAM_SEARCH_ENGINE),
"solr"))
            {
                results = getSolrResults(request);
            }
            else if
(StringUtils.equalsIgnoreCase(request.getParameter(PARAM_SEARCH_ENGINE),
"elasticsearch"))
            {
                results = getElasticsearchResults(request);
            }


            writeJson(results, response);
        }
    }
```

Prikazana metoda *handleRequest* obrađuje zahtjev tako da dohvati parametar „*search engine*“ iz zahtjeva te na osnovu vrijednosti parametra – „*lucene*“ poziva metodu za dohvaćanje lucene rezultata te joj kao parametar predaje zahtjev.

Zatim se iz zahtjeva uzima parametar za pretragu kojeg je korisnik upisao te se izgradi objekt *PredicateGroup* koji će biti korišten za slanje upita.

```
Map<String, String> predicatesMap = new HashMap<>();


        String fulltext = request.getParameter(PREDICATE_FULLTEXT);
        if (fulltext == null || fulltext.length() <
searchTermMinimumLength)
        {
            return results;
        }


        long resultsOffset = getResultOffset(request);
        setQueryPredicates(request, predicatesMap, fulltext,
searchRootPagePath);


        PredicateGroup predicates =
PredicateConverter.createPredicates(predicatesMap);
        ResourceResolver resourceResolver =
request.getResource().getResourceResolver();


        Query query = this.queryBuilder.createQuery(predicates,
resourceResolver.adaptTo(Session.class));
        if (resultsSize != 0)
        {
            query.setHitsPerPage(resultsSize);
        }
        if (resultsOffset != 0)
        {
            query.setStart(resultsOffset);
        }
        SearchResult searchResult = query.getResult();


        List<Hit> hits = searchResult.getHits();
        if (hits != null)
        {
            addHitsToResultsListItem(hits, results, request);
        }
```

```
        return results;
```

Metoda dalje dohvaća rezultate slanjem upita i pretvaranjem svakog od rezultata u objekt (stranicu, sliku, stranicu i sliku skupa ili sadržaj na osnovi oznake) pogodan za prikaz na stranici preko *JavaScript* koda. To je ostvareno sljedećom metodom:

```
/**
     * This method adds different type of results as the same object in
result list
     * @param hits
     * @param results
     * @param request
     */
    public static void addHitsToResultsListItem(List<Hit> hits,
List<ListItem> results, SlingHttpServletRequest request)
    {
        for (Hit hit : hits)
        {
            try
            {
                Resource hitRes = hit.getResource();

                if (hitRes.getResourceType().equals(NT_DAM_ASSET))
                {
                    addAssetToResultsList(results, request, hitRes);
                }
                else if (hitRes.getResourceType().equals(NT_PAGE))
                {
                    addPageToResultsList(results, request, hitRes);
                }
            }
            catch (RepositoryException e)
            {
                LOGGER.error("Unable to retrieve search results for
query.", e);
            }
        }
    }
```

Zatim se svaki od rezultata dodaje u listu:

```
/**
     * This method adds page to result list
     * @param results
     * @param request
     * @param hitRes
     */
    public static void addPageToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Page page = getPage(hitRes);

        if (page != null)
        {
            results.add(new PageListItemImpl(request, page));
        }
    }

    /**
     * This method adds asset to result list
     * @param results
     * @param request
     * @param hitRes
     */
    public static void addAssetToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Asset asset = getAsset(hitRes);

        if (asset != null)
        {
            results.add(new AssetListItemImpl(request, asset));
        }
    }
```

PageListItemImpl i *AssetListItemImpl* su klase koje obrađuju primljene rezultate te njihova svojstva prilagođavaju formatu svojstava koje zahtjeva *lucene* indeks.

Lista rezultata se ispisiva preko metode *writeJson*.

```
/**
     * This method takes results to write a response to users on site.
     * @param results
```

```
     * @param response
     */
    private void writeJson(List<ListItem> results, SlingHttpServletResponse
response)
    {
        response.setContentType(APPLICATION_JSON);
        response.setCharacterEncoding(UTF_8);
        ObjectMapper mapper = new ObjectMapper();
        try
        {
            mapper.writeValue(response.getWriter(), results);
        }
        catch (IOException e)
        {
            LOGGER.error(e.getMessage());
        }
    }
```

Indeksiranje sadržaja preko odvojenog *RESTful Solr* poslužitelja

Osnovna razlika rada sa *Solr*-om je u tome što je *Solr* odvojeni poslužitelj koji nije ni na kakav način povezan s bazom podataka. Tako je moguće indeksirati i pretraživati čak i vanjski sadržaj koji se ne mora spremati u bazu podataka. Komunikacija za potrebe indeksiranja i pretrage se odvija preko *REST API* klijenta.

Da bi radili sa *Solr* poslužiteljem, potrebno je dodati *Solr* zavisnosti u *pom.xml* datoteku *parent* modula, te ih referencirati u core modulu.

```
<!-- Solr Dependencies -->
            <dependency>
                <groupId>org.apache.servicemix.bundles</groupId>
                <artifactId>org.apache.servicemix.bundles.solr-
solrj</artifactId>
                <version>5.4.1_1</version>
            </dependency>
            <dependency>
                <groupId>log4j</groupId>
                <artifactId>log4j</artifactId>
                <version>1.2.17</version>
            </dependency>
            <dependency>
```

```
            <groupId>org.noggit</groupId>
            <artifactId>noggit</artifactId>
            <version>0.5</version>
        </dependency>
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.5.6</version>
        </dependency>
```

AEM zahtjeva pakiranje cijelog core modula u *OSGi* paket. U *AEM*-u postoji mnogo servisa koji su *OSGi* paketi i tako je ostvarena komunikacija među njima. *Solr* zavisnost je već zapakirana u *OSGi* paket preko *Apache Servicemix* servisa pa je kao takva uključena u postojeći projekt. Budući da je zahtjevan posao dodati sve tranzitivne zavisnosti *Solr*-a, osnovne su dodane, a ostale su uključene preko *maven-bundle-plugina* koji znatno olakšava manipulaciju vanjskih zavisnosti u *OSGi*. To je ostvareno podebljanim linijama u sljedećem prikazu:

```
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <extensions>true</extensions>
    <configuration>
       <instructions>
            <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
            <Embed-Transitive>true</Embed-Transitive>
            <Embed-Directory>OSGI-INF/lib</Embed-Directory>
            <Export-Package>we.retail.core.model*</Export-Package>
            <Import-Package>*;resolution:=optional</Import-Package>
            <Private-Package>
                    we.retail.core*
            </Private-Package>
            <Sling-Model-Packages>
                    we.retail.core.model
            </Sling-Model-Packages>
        </instructions>
     </configuration>
</plugin>
```

Solr zahtjeva postavljanje konfiguracijskih datoteka u kojima se definira kakve podatke će poslužitelj spremati i na koji način.

Za to je potrebno urediti sljedeće datoteke:

- *solr.xml* – definirane generalne postavke za *Solr* poslužitelja poput vremena isteka obrade zahtjeva, portova i sl. Ovdje su sačuvane postojeće, zadane postavke.
- *protwords.xml* – u ovoj datoteci potrebno je navesti riječi za koje ne želim oda stemming pri analizi obje riječi pretvori u isto, tj. ostavi im isti korijen riječi. Za ovaj projekt nije potrebno raditi izmjene jer nije primijećeno da se neki dio sadržaja ponaša drugačije nego što bi trebao.
- *stopwords.xml* – popis svih riječi koje želimo maknuti iz indeksiranja pri analizi riječi i pretvaranju riječi u tokene.
- *synonyms.xml* – popis svih sinonima, tj. riječi koje u *AEM*-u imaju isto značenje i nema ih potrebe dva puta indeksirati. To bi u ovom slučaju mogle biti riječi „*asse"t*, „*image"* i „*picture"*, jer u projektu ne postoje drugi asseti osim slika. Ove funkcionalnosti su specifične pa nisu mijenjane jer je cilj obratiti pozornost na česte slučajeve primjene u praksi.
- *solrconfig.xml* je datoteka u kojoj se postavljaju konfiguracije *Solr* poslužitelja. Važno je provjeriti na koju je verziju postavljen *Lucene* jer mi pri uključivanju tranzitivnih zavisnosti *Solra* moglo doći do različitih verzija *Lucenea* što može izazvati velike probleme u projektu. Korišteno je zadano ponašanje, a verzija *Lucenea* je 8.2.0 što odgovara verziji u projektu.
- *managed-schema.xml* – ovo je datoteka koju je gotovo uvijek potrebno mijenjati. U njoj se definira svaka vrijednosti koju može primiti indeks i način na koji se ona obrađuje, analizira te sprema. Primjer datoteke koja je korištena za ovaj projekt slijedi u nastavku.

```
<?xml version="1.0" encoding="UTF-8" ?>

<schema name="aem-config" version="1.6">

    <field name="id" type="string" indexed="true" stored="true"
required="true" multiValued="false" />
    <field name="type" type="string" indexed="true" stored="true"
required="true" multiValued="false" />
    <!-- docValues are enabled by default for long type so we don't need to
index the version field  -->
    <field name="version" type="plong" indexed="false" stored="false"/>
```

```
    <!-- If you don't use child/nested documents, then you should remove
the next two fields:  -->

    <!-- for nested documents (minimal; points to root document) -->

    <field name="root" type="string" indexed="true" stored="false"
docValues="false" />

    <!-- for nested documents (relationship tracking) -->

    <field name="nest_path" type="nest_path" /><fieldType name="nest_path"
class="solr.NestPathField" />


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


<!-- PAGES -->

<field name="jcr_title" type="string" indexed="true" stored="true" />

<field name="jcr_created" type="pdate" indexed="true" stored="true" />

<field name="jcr_description" type="string" indexed="true" stored="true" />

<field name="cq_lastModified" type="pdate" indexed="true" stored="true" />

<field name="jcr_name" type="string" indexed="true" stored="true" />

<field name="jcr_primaryType" type="string" indexed="true" stored="true" />

<field name="sling_resourceType" type="string" indexed="true" stored="true"
/>

<field name="cqTags" type="strings" indexed="true" stored="true" />

<field name="searchDescription" type="text_general" indexed="true"
stored="true" omitNorms="true"/>

<field name="pageImportanceRank" type="pint" indexed="true" stored="true"/>

<field name="manualCreationDate" type="pdate" indexed="true" stored="true"
/>



<!-- ASSETS -->

<field name="damAssetId" type="string" indexed="true" stored="true" />

<field name="jcr_mimeType" type="string" indexed="true" stored="true" />

<field name="dam:MIMEtype" type="string" indexed="true" stored="true" />

<field name="cqName" type="string" indexed="true" stored="true" />

<field name="cqParentPath" type="text_general" indexed="true" stored="true"
/>

<field name="damRelativePath" type="text_general" indexed="true"
stored="true" />

<field name="description" type="text_general" indexed="true" stored="true"
omitNorms="true"/>

<field name="sellingPriotiry" type="pint" indexed="true" stored="true" />

<field name="creationDate" type="pdate" indexed="true" stored="true" />

<field name="cqTags_asset" type="strings" indexed="true" stored="true" />
```

U iznad prikazanom su definirana polja u indeksu koja predstavljaju sva svojstva koja mogu imati stranice i slike. Svako od navedenih polja ima svoj tip, međutim *Solr* još uvijek ne poznaje što točno označava *string*, *text_general*, *pint* i ostali korišteni tipovi. U nastavku ove datoteke je i to definirano.

```
<!-- catchall field, containing all other searchable text fields
(implemented via copyField further on in this schema  -->
   <field name="text" type="text_general" indexed="true" stored="false"
multiValued="true"/>
```

Kao što je iznad prikazano, važno je definirati jedno polje koje će definirati ponašanje svakog svojstva koje nije definirano ranije napisanim poljima. U ovom slučaju želimo da to bude tipa text_general, odnosno da se takvo svojstvo spremi kao polje punog teksta (eng. *fulltext field*).

```
<!-- Dynamic field definitions allow using convention over configuration
for fields via the specification of patterns to match field names.
EXAMPLE:  name="*_i" will match any field ending in _i (like myid_i, z_i)
RESTRICTION: the glob-like pattern in the name attribute must have a "*"
only at the start or the end.  -->

<dynamicField name="*_i"  type="pint"    indexed="true"  stored="true"/>
<dynamicField name="*_is" type="pints"    indexed="true"  stored="true"/>
<dynamicField name="*_s"  type="string"  indexed="true"  stored="true" />
<dynamicField name="*_ss" type="strings"  indexed="true"  stored="true"/>
<dynamicField name="*_l"  type="plong"   indexed="true"  stored="true"/>
<dynamicField name="*_ls" type="plongs"   indexed="true"  stored="true"/>
<dynamicField name="*_t" type="text_general" indexed="true" stored="true"
multiValued="false"/>
<dynamicField name="*_txt" type="text_general" indexed="true"
stored="true"/>
<dynamicField name="*_b"  type="boolean" indexed="true" stored="true"/>
<dynamicField name="*_bs" type="booleans" indexed="true" stored="true"/>
<dynamicField name="*_f"  type="pfloat"  indexed="true"  stored="true"/>
<dynamicField name="*_fs" type="pfloats"  indexed="true"  stored="true"/>
<dynamicField name="*_d"  type="pdouble" indexed="true"  stored="true"/>
<dynamicField name="*_ds" type="pdoubles" indexed="true"  stored="true"/>
<!--<dynamicField name="*" type="text_general" indexed="true" stored="true"
/> -->
```

```
<dynamicField name="random_*" type="random"/>
<dynamicField name="ignored_*" type="ignored"/>

<!-- Type used for data-driven schema, to add a string copy for each text
field -->
<dynamicField name="*_str" type="strings" stored="false" docValues="true"
indexed="false" useDocValuesAsStored="false"/>

<dynamicField name="*_dt"  type="pdate"    indexed="true"  stored="true"/>
<dynamicField name="*_dts" type="pdate"    indexed="true"  stored="true"
multiValued="true"/>
<dynamicField name="*_p"  type="location" indexed="true" stored="true"/>
<dynamicField name="*_srpt"  type="location_rpt" indexed="true"
stored="true"/>

<!-- payloaded dynamic fields -->
<dynamicField name="*_dpf" type="delimited_payloads_float" indexed="true"
stored="true"/>
<dynamicField name="*_dpi" type="delimited_payloads_int" indexed="true"
stored="true"/>
<dynamicField name="*_dps" type="delimited_payloads_string" indexed="true"
stored="true"/>

<dynamicField name="attr_*" type="text_general" indexed="true"
stored="true" multiValued="true"/>
```

Dinamička polja definirana u kodu iznad su važna u situacijama kada je određena polja prepoznatljiva po svom nazivu. Posljednja linija označava kako svako polje koje započinje sa „*attr_*“ će uvijek biti spremljeno i indeksirano kao polje punog teksta. Također, može sadržavati više vrijednosti u polju.

```
<!-- Field to use to determine and enforce document uniqueness. Unless this
field is marked with required="false", it will be a required field -->
<uniqueKey>id</uniqueKey>
```

U *managed-schema* datoteci potrebno je definirati i jedinstveni identifikator za svaki dokument u indeksu. U ovom slučaju to je polje „*id*“.

```
<!-- copyField commands copy one field to another at the time a document
```

```
is added to the index.  It's used either to index the same field
differently, or to add multiple fields to the same field for easier/faster
searching. -->
<!-- copyField commands copy one field to another at the time a document is
added to the index.  It's used either to index the same field differently,
or to add multiple fields to the same field for easier/faster searching.-->

<copyField source="jcr_title" dest="text"/>
<copyField source="jcr_description" dest="text"/>
<copyField source="jcr_name" dest="text"/>
<copyField source="jcr_primaryType" dest="text"/>
<copyField source="sling_resourceType" dest="text"/>
<copyField source="cqTags" dest="text"/>
<copyField source="searchDescription" dest="text"/>
<copyField source="pageImportanceRank" dest="text"/>
<copyField source="manualCreationDate" dest="text"/>

<copyField source="cqName" dest="text"/>
<copyField source="cqParentPath" dest="text"/>
<copyField source="damRelativePath" dest="text"/>
<copyField source="description" dest="text"/>
<copyField source="sellingPriotiry" dest="text"/>
<copyField source="creationDate" dest="text"/>
<copyField source="cqTags_asset" dest="text"/>
```

Ako je određeno polje potrebno spremiti na više načina ili je potrebno više polja spremiti u jedno, tada se koristi opcija *copyField*. Primjer toga je polje *creationDate*, koje je moguće pretraživati tako da korisnik upiše točan datum i dobije rezultat, ali i ako upiše djelomičan datum potrebno je moći naći polje kao pretragu punog teksta.

```
<!-- The StrField type is not analyzed, but indexed/stored verbatim. -->
<fieldType name="string" class="solr.StrField" sortMissingLast="true"
docValues="true" />
<fieldType name="strings" class="solr.StrField" sortMissingLast="true"
multiValued="true" docValues="true" />

<!-- boolean type: "true" or "false" -->
<fieldType name="boolean" class="solr.BoolField" sortMissingLast="true"/>
<fieldType name="booleans" class="solr.BoolField" sortMissingLast="true"
multiValued="true"/>
```

```
<!-- Numeric field types that index values using KD-trees. Point fields
don't support FieldCache, so they must have docValues="true" if needed for
sorting, faceting, functions, etc. -->
<fieldType name="pint" class="solr.IntPointField" docValues="true"/>
<fieldType name="pfloat" class="solr.FloatPointField" docValues="true"/>
<fieldType name="plong" class="solr.LongPointField" docValues="true"/>
<fieldType name="pdouble" class="solr.DoublePointField" docValues="true"/>

<fieldType name="pints" class="solr.IntPointField" docValues="true"
multiValued="true"/>
<fieldType name="pfloats" class="solr.FloatPointField" docValues="true"
multiValued="true"/>
<fieldType name="plongs" class="solr.LongPointField" docValues="true"
multiValued="true"/>
<fieldType name="pdoubles" class="solr.DoublePointField" docValues="true"
multiValued="true"/>
<fieldType name="random" class="solr.RandomSortField" indexed="true"/>

<!-- since fields of this type are by default not stored or indexed, any
data added to them will be ignored outright.  -->
<fieldType name="ignored" stored="false" indexed="false" multiValued="true"
class="solr.StrField" />
```

U prethodnom bloku koda definirano je značenje tipova podataka ranije definiranih za polja koja će biti prisutna u indeksu. Npr. podebljani tip *strings* je označen kao polje koje može sadržavati više od jedne vrijednosti.

```
<!-- A general text field that has reasonable, generic cross-language
defaults: it tokenizes with StandardTokenizer, removes stop words from
case-insensitive "stopwords.txt"(empty by default), and down cases.  At
query time only, it also applies synonyms. -->
<fieldType name="text_general" class="solr.TextField"
positionIncrementGap="100" multiValued="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
<!-- in this example, we will only use synonyms at query time <filter
class="solr.SynonymGraphFilterFactory" synonyms="index_synonyms.txt"
ignoreCase="true" expand="false"/>
    <filter class="solr.FlattenGraphFilterFactory"/> -->
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
```

```
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true"
words="stopwords.txt" />
    <filter class="solr.SynonymGraphFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
     <filter class="solr.LowerCaseFilterFactory"/>
   </analyzer>
</fieldType>
```

Za tip *text_general* u prethodnom dijelu koda određeni su analizatori i filteri koje će koristiti polja tog tipa. Slično je napravljeno za sva do sada navedena polja pa sva neće biti prikazana. Na ovaj način je moguće definirati i jezike vrijednosti koje se spremaju u određena polja te na osnovu toga primjenjivati različite vrste analizatora specifične za te jezike.

Kreće rad koji će prikazati konkretne rezultate korisniku, a za to je najprije potrebno na korisničko sučelje dodati opciju sinkronog indeksiranja kako bi se ostvarila mogućnost ručne manipulacije indeksom. Na postojeću komponentu pretrage ranije prikazanu potrebno je dodati sljedeći kod:

```
<form id="solrForm" action="${currentPage.path @
addSelectors=['indexSolrPages'], extension='json', suffix =
search.relativePath}" autocomplete="off" method="get" style="display:
none;">
        <h3>Solr collection management</h3>
        <button name="indexType" type="submit" value="indexpages"
style="background-color:#D43732; border-radius: 8px; display:inline-block;
margin-right:10px">
            Index data
        </button>
        <button name="indexType" type="submit" value="deleteindexdata"
style="background-color:#D43732; border-radius: 8px; display:inline-
block;">
            Delete index
        </button>
    </form>


<script>
    function yesnoCheck(that) {
        if (that.value == "solr") {
            document.getElementById("solrForm").style.display = "block";
        } else {
            document.getElementById("solrForm").style.display = "none";
        }
```

```
    }
</script>
```

Tako postižemo prikaz sljedećeg sučelja pri odabiru opcije „*Solr*“:

Slika 45 - Upravljanje *Solr* indeksom preko korisničkog sučelja

Zahtjev za indeksiranjem ili brisanjem sadržaja na *Solr* serveru obrađuje *SolrIndexingServlet* klasa.

```
@Override
    protected void doGet(@Nonnull SlingHttpServletRequest request, @Nonnull
SlingHttpServletResponse response)
    {
        this.doPost(request, response);
    }


    /**
     * This method calls other methods for indexing content or deleting
indexed content from Solr server.
     * This is based on user's choice provided in given request.
     * This method writes response of successful or unsuccessful
indexing/deleting.
     * @param request
     * @param response
     */
    @Override
    protected void doPost(@Nonnull SlingHttpServletRequest request,
@Nonnull SlingHttpServletResponse response)
    {
        response.setContentType(CT_TEXT_HTML);
```

```
        String indexType = request.getParameter(PARAM_INDEX_TYPE);
        String url = getSolrServerUrl(this.solrConfigurationService);

        if (indexType.equalsIgnoreCase(PROP_INDEX_TYPE_INDEX))
        {
            indexData(request, response, url);
        }
        else if (indexType.equalsIgnoreCase(PROP_INDEX_TYPE_DELETE))
        {
            deleteData(url, response);
        }
    }
```

U metodu doPost se iz zahtjeva preuzimaju relevantni podaci kako bi se odredilo je l isadržaj potrebno indeksirati ili izbrisati iz indeksa. Na osnovu toga se pozivaju metode „*indexData*“ i „*deleteData*“.

```
    /**
     * This method performs indexing.
     * It is done by taking Solr server URL to create client, index data
and return response
     * @param request
     * @param response
     * @param url
     */
    private void indexData(SlingHttpServletRequest request,
SlingHttpServletResponse response, String url)
    {
        Resource resource = request.getResource();
        try (HttpSolrClient server = new HttpSolrClient(url);
ResourceResolver resourceResolver = resource.getResourceResolver())
        {
            Session session = resourceResolver.adaptTo(Session.class);
            List<AbstractSolrItemModel> indexPageData =
this.solrSearchService.crawlContent(session);
            boolean dataIndexed =
this.solrSearchService.indexPagesToSolr(indexPageData, server);

            writeResponse(dataIndexed, response);
        }
        catch (Exception e)
```

```
        {
            LOG.error("Exception due to ", e);
        }
    }
```

U metodi „*indexData*“ potrebno je dohvatiti cjelokupni sadržaj koji će se indeksirati preko metoda „*crawlData*“, a zatim nad vraćenim podacima provesti i indeksiranje preko metode „*indexPagesToSolr*“. Na kraju je potrebno korisniku vratiti odgovor o uspješnom ili neuspješnom indeksiranju sadržaja.

```
    /**
     * This method takes Boolean parameter dataIndex and writes response
based on it's value.
     * @param dataIndexed
     * @param response
     * @throws IOException
     */
    private void writeResponse(boolean dataIndexed,
SlingHttpServletResponse response) throws IOException
    {
        if (dataIndexed)
        {
            response.getWriter().write("<h3>Successfully indexed content
pages to Solr server </h3>");
        }
        else
        {
            response.getWriter().write("<h3>Something went wrong</h3>");
        }
    }
```

Prethodna metoda obavještava korisnika o izvršenim radnjama i njihovim rezultatima.

```
    /**
     * This method takes URL of Solr client to write info response about
index data deletion.
     * @param url
     * @param response
     */
    private void deleteData(String url, SlingHttpServletResponse response)
```

```
    {
        try (HttpSolrClient server = new HttpSolrClient(url))
        {
            server.deleteByQuery(":");
            server.commit();
            response.getWriter().write("<h3>Deleted all the indexes from
solr server </h3>");
        }
        catch (SolrServerException | IOException e)
        {
            LOG.error("Exception due to ", e);
        }
    }
}
```

U slučaju da korisnik želi izbrisati indeksirane podatke, to se ostvaruje preko metode „deleteData“ koja šalje zahtjev na *Solr* Aplikacijsko Programsko Sučelje (u nastavku teksta *API*) za brisanjem podataka te spremi promjene. Važno je naglasiti kako *Solr* funkcionira tako da korisnik može pretraživati podatke sve dok posljednji podatak nije indeksiran. Na *Solr* poslužitelju se podaci dupliciraju (važno je paziti na memorijske resurse), te se briše jedan indeks. Kada je indeks u potpunosti izbrisan, sadržaj postaje nedostupan, a nakon toga se briše i druge indeks s nedostupnim sadržajem. Ako se u bilo kojem trenutnu brisanje želi prekinuti, na ovaj način korisnik ima sačuvane sve podatke. Tako funkcionira ponovno indeksiranje podataka s promjenama, a ovdje je taj proces prikazan manualnim brisanjem i indeksiranjem.

Da bi se podaci uspješno indeksirali, potrebno ih je pretvoriti u *Solr* formate podataka i dodati u *SolrInputDocument* objekt. To je postignuto sljedećim metodama:

```
/**
     * This method connects to the Solr server and indexes page content
using Solrj API. This is used by bulk update handler (servlet)
     * @param indexItemData
     * @param server
     * Takes Json array and iterates over each object and index solr
     * @return boolean true if it indexes successfully to solr server, else
false.
     * @throws SolrServerException
     * @throws IOException
     */
    @Override
```

```
    public boolean indexPagesToSolr(List<AbstractSolrItemModel>
indexItemData, HttpSolrClient server) throws SolrServerException,
IOException
    {
        if (null != indexItemData)
        {
            for (int i = 0; i < indexItemData.size(); i++)
            {
                SolrInputDocument doc = null;
                if (indexItemData.get(i).getType().equals(NT_PAGE))
                {
                    SolrPageModel pageModel = (SolrPageModel)
indexItemData.get(i);
                    doc = createPageSolrDoc(pageModel);
                }
                else if
(indexItemData.get(i).getType().equals(NT_DAM_ASSET))
                {
                    SolrAssetModel assetModel = (SolrAssetModel)
indexItemData.get(i);
                    doc = createAssetSolrDoc(assetModel);
                }
                server.add(doc);
            }
            server.commit();
            return true;
        }

        return false;
    }

    /**
     * This method gets metadata from page and converts it to Solr page
object to create a document
     * @param solrPageModel
     * @return Solr document
     */
    private SolrInputDocument createPageSolrDoc(SolrPageModel
solrPageModel)
    {
        SolrInputDocument doc = new SolrInputDocument();

        doc.addField("id", solrPageModel.getId());
```

```
        doc.addField("type", solrPageModel.getType());


        doc.addField("jcr_title", solrPageModel.getJcrTitle());
        doc.addField("jcr_created", solrPageModel.getJcrCreated());
        doc.addField("jcr_description", solrPageModel.getJcrDescription());
        doc.addField("cq_lastModified", solrPageModel.getCqLastModified());
        doc.addField("jcr_name", solrPageModel.getJcrName());
        doc.addField("jcr_primaryType", solrPageModel.getJcrPrimaryType());
        doc.addField("sling_resourceType",
solrPageModel.getSlingResourceType());
        doc.addField("cqTags", solrPageModel.getCqTags());
        doc.addField("searchDescription",
solrPageModel.getSearchDescription());
        doc.addField("pageImportanceRank",
solrPageModel.getPageImportanceRank());
        doc.addField("manualCreationDate",
solrPageModel.getManualCreationDate());


        return doc;
    }


    /**
     * This method gets metadata from asset and converts it to Solr asset
object to create a document
     * @param solrAssetModel
     * @return
     */
    private SolrInputDocument createAssetSolrDoc(SolrAssetModel
solrAssetModel)
    {
        SolrInputDocument doc = new SolrInputDocument();


        doc.addField("id", solrAssetModel.getId());
        doc.addField("type", solrAssetModel.getType());


        doc.addField("damAssetId", solrAssetModel.getId());
        doc.addField("jcr_mimeType", solrAssetModel.getJcrMimeType());
        doc.addField("cqName", solrAssetModel.getCqName());
        doc.addField("cqParentPath", solrAssetModel.getCqParentPath());
        doc.addField("damRelativePath",
solrAssetModel.getDamRelativePath());
        doc.addField("cqTags_asset", solrAssetModel.getCqTags());
        doc.addField("description", solrAssetModel.getDescription());
```

```
        doc.addField("sellingPriotiry",
solrAssetModel.getSellingPriority());
        doc.addField("creationDate", solrAssetModel.getCreationDate());

        return doc;
    }
```

Svi objekti se prije indeksiranja mapiraju u posebne modele iz kojih se poslije izvlače potrebni podaci za indeksiranje. Za tu svrhu su napravljena tri modela: *AbstractSolrItemModel, SolrPageModel i SolrAssetModel*. Slijedi prikaz navedenih modela:

```
/**
 * Abstract model with all properties that both pages and assets contains
 */
@Model(adaptables = Resource.class, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public abstract class AbstractSolrItemModel
{
    @Self
    protected Resource resource;

    protected String id;
    protected String type;

    public String getId()
    {
        return this.resource.getPath();
    }

    public String getType()
    {
        return this.resource.getResourceType();
    }

}
```

Prethodni model sadrži svojstva tip i *id*, što su jedina zajednička svojstva stranica i slika.

```
/**
 * Extension of AbstractSolrItemModel class with page-specific properties
```

```
 */
@Model(adaptables = Resource.class, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public class SolrPageModel extends AbstractSolrItemModel
{
    @Inject
    @Named("jcr:content/jcr:title")
    private String jcrTitle;

    @Inject
    @Named("jcr:content/jcr:created")
    private Calendar jcrCreated;

    @Inject
    @Named("jcr:content/jcr:description")
    private String jcrDescription;

    @Inject
    @Named("jcr:content/cq:lastModified")
    private Calendar cqLastModified;

    @Inject
    @Named("jcr:content/jcr:name")
    private String jcrName;

    @Inject
    @Named("jcr:content/jcr:primaryType")
    private String jcrPrimaryType;

    @Inject
    @Named("jcr:content/sling:resourceType")
    private String slingResourceType;

    @Inject
    @Named("jcr:content/cq:tags")
    private String[] cqTags;

    @Inject
    @Named("jcr:content/searchDescription")
    private String searchDescription;
```

```
    @Inject
    @Named("jcr:content/pageImportanceRank")
    private String pageImportanceRank;

    @Inject
    @Named("jcr:content/manualCreationDate")
    private Calendar manualCreationDate;

    public String getJcrTitle()
    {
        return this.jcrTitle;
    }

    /**
     * @return jcrCreated property in Solr format (necessary for indexing)
     */
    public String getJcrCreated()
    {
        return SolrUtils.castToSolrDate(this.jcrCreated);
    }

    public String getJcrDescription()
    {
        return this.jcrDescription;
    }

    /**
     * @return cqLastModified property in Solr format (necessary for
indexing)
     */
    public String getCqLastModified()
    {
        return SolrUtils.castToSolrDate(this.cqLastModified);
    }

    public String getJcrName()
    {
        return this.jcrName;
    }

    public String getJcrPrimaryType()
```

```
    {
        return this.jcrPrimaryType;
    }

    public String getSlingResourceType()
    {
        return this.slingResourceType;
    }

    public String[] getCqTags()
    {
        return this.cqTags;
    }

    public String getSearchDescription()
    {
        return this.searchDescription;
    }

    public String getPageImportanceRank()
    {
        return this.pageImportanceRank;
    }

    /**
     * @return manualCreationDate property in Solr format (necessary for
indexing)
     */
    public String getManualCreationDate()
    {
        return SolrUtils.castToSolrDate(this.manualCreationDate);
    }
}
```

Sličan model je napravljen i za slike:

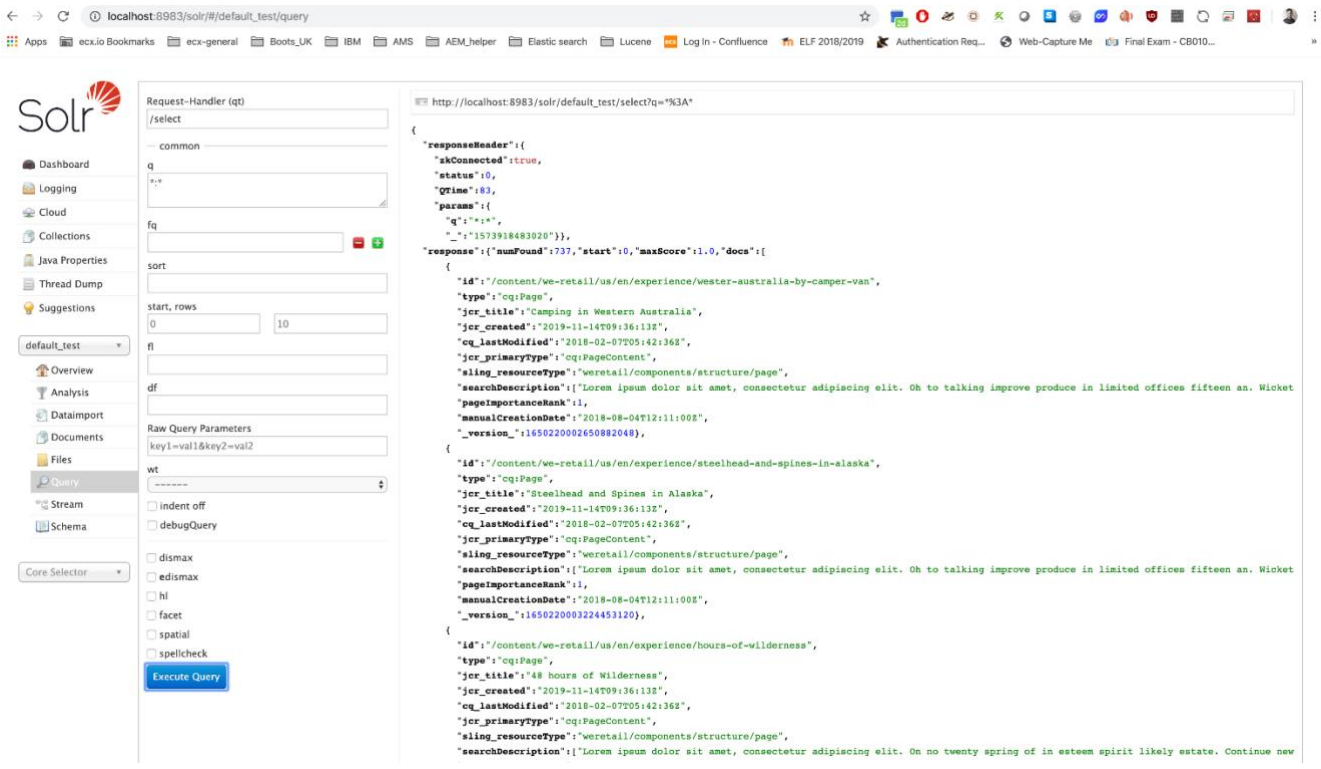
```
/**
 * Extension of AbstractSolrItemModel class with page-specific properties
 */
```

```
@Model(adaptables = Resource.class, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public class SolrAssetModel extends AbstractSolrItemModel
{
    @Inject
    @Named("jcr:content/jcr:mimeType")
    private String jcrMimeType;

    @Inject
    @Named("jcr:content/dam:mimeType")
    private String damMimeType;

    @Inject
    @Named("jcr:content/cq:name")
    private String cqName;

    @Inject
    @Named("jcr:content/cq:parentPath")
    private String cqParentPath;

    @Inject
    @Named("jcr:content/dam:relativePath")
    private String damRelativePath;

    @Inject
    @Named("jcr:content/description")
    private String description;

    @Inject
    @Named("jcr:content/sellingPriority")
    private String sellingPriority;

    @Inject
    @Named("jcr:content/creationDate")
    private Calendar creationDate;

    @Inject
    @Named("jcr:content/cq:tags")
    private String[] cqTags;

    /**
```

```
 * In this case it is not important which mime type to save.
 * In some other cases these could be two different index fields.
 * @return any mime type available
 */
public String getJcrMimeType()
{
    if (this.jcrMimeType == null)
    {
        return this.damMimeType;
    }
    else
    {
        return this.jcrMimeType;
    }
}

public String getCqName()
{
    return this.cqName;
}

public String getCqParentPath()
{
    return this.cqParentPath;
}

public String getDamRelativePath()
{
    return this.damRelativePath;
}

public String getDescription()
{
    return this.description;
}

public String getSellingPriority()
{
    return this.sellingPriority;
}
```

```
    /**
     * @return creation date in solr format
     */
    public String getCreationDate()
    {
        return SolrUtils.castToSolrDate(this.creationDate);
    }

    public String[] getCqTags()
    {
        return this.cqTags;
    }
}
```

Slijedi prikaz indeksiranog sadržaja na *Solr* Administratorskom Sučelju:



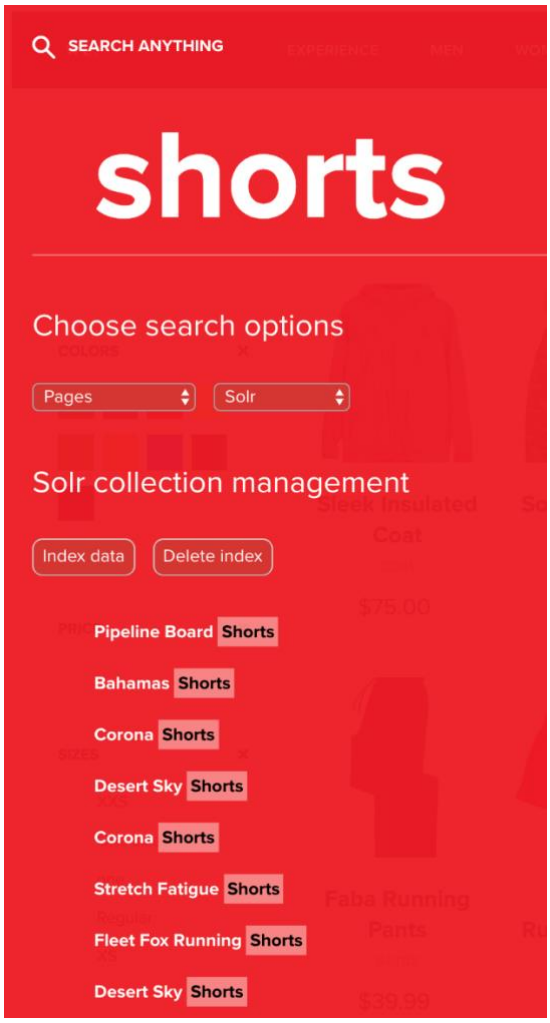
Slika 46 - *Solr* poslužitelj - Administratorsko sučelje

Na prethodnog slici prikazana je struktura podataka kreiranog indeksa.

Pretraga preko *Solr* sustava pretraživanja

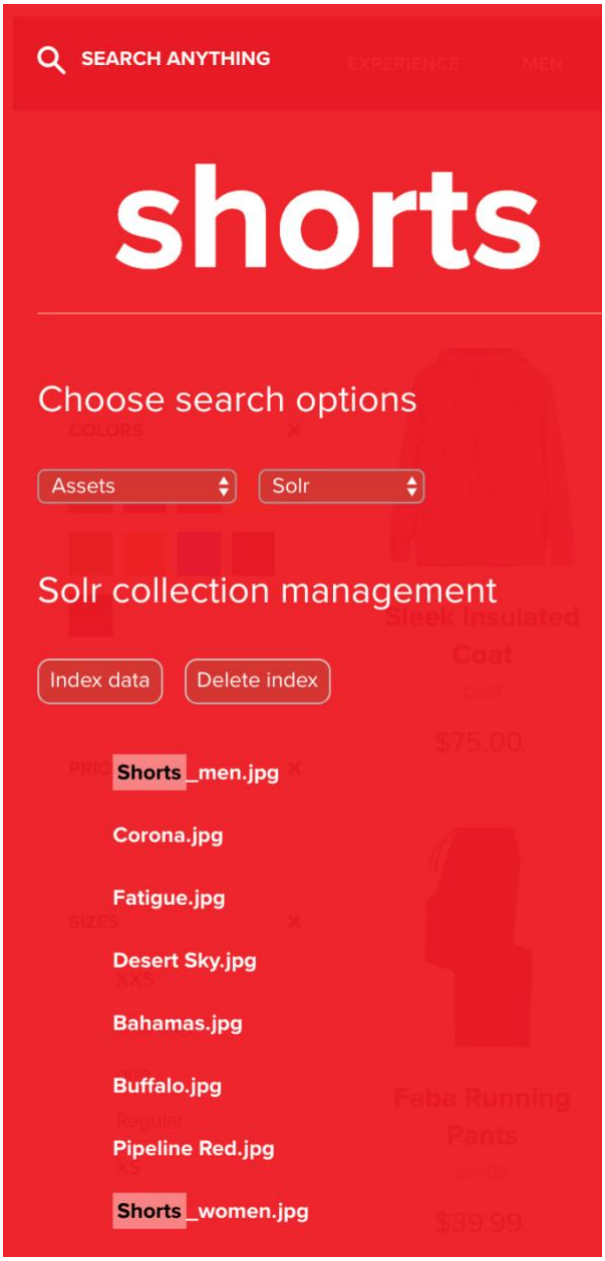
Nakon prikazanog indeksiranja podataka u *Solr* indeks, potrebno je implementirati prikaz rezultata upita koji se šalju iz korisnikovog upisa na *Solr* poslužitelj. Kao i kod *Oak Lucene* pretrage, postoji više opcija za pretragu:

Pretraga stranica i prikaz prvih osam rezultata pretrage:



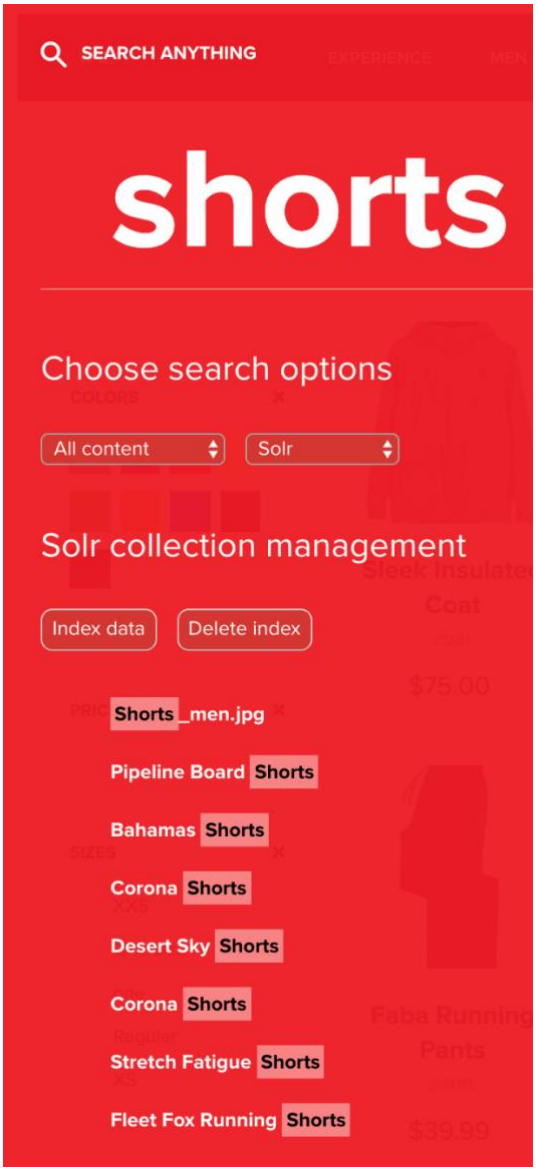
Slika 47 - *Solr* pretraga stranica

Pretraga slika i prikaz prvih osam rezultata pretrage:



Slika 48 - *Solr* pretraga slika

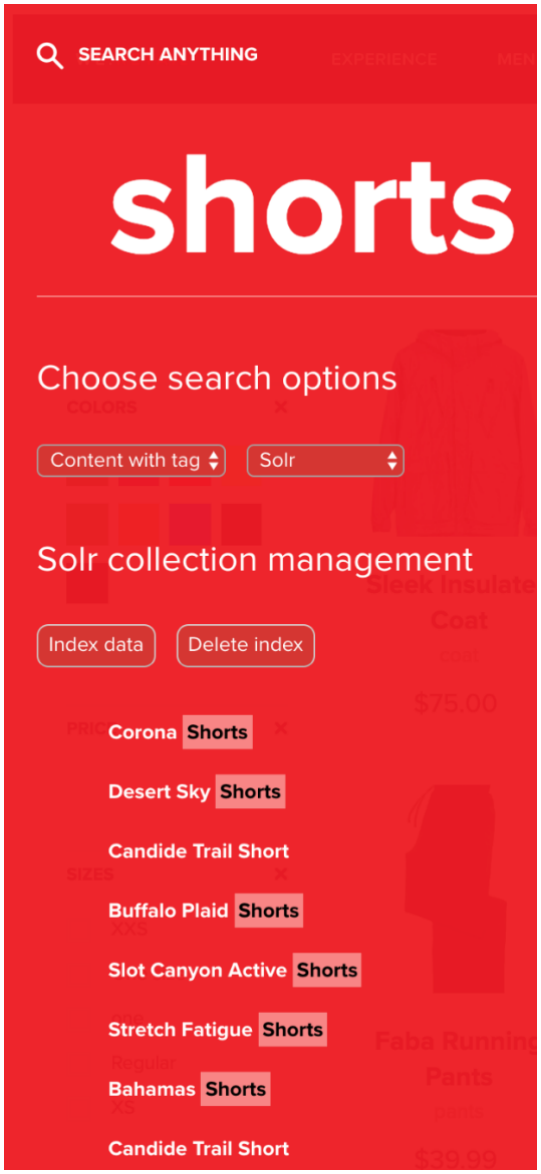
Pretraga cjelokupnog sadržaja i prikaz prvih osam rezultata pretrage:



Slika 49 - *Solr* pretraga cjelokupnog sadržaja

U slučaju s prethodne slike rezultati cjelokupnog sadržaja se razlikuju od rezultata za pretragu stranica za jedan rezultat. Razlika u odnosu na *Lucene* upit je ta da je u *Solr* indeksu (kolekciji) definiran jednak broj polja (svojstava) za svaki dokument bez obzira radi li se o stranici ili slici. U ovom slučaju je prednost dobila slika zbog podudaranja prvog mjesta u imenu.

Pretraga sadržaja po oznakama i prikaz rezultata:



Slika 50 - Rezultat pretrage po oznakama

Slike trenutno nemaju oznaka koje sadrže termin „*shorts*“.

Prikazani rezultati omogućeni su kroz istu klasu (*SearchServlet*) kao i *Oak Lucene* pretraga. Slijedi prikaz dijela implementacije *Solr* pretrage preko *RESTful Solr API*-ja.

```
/**
     * This method call other methods for perform lucene, solr or
elasticsearch based on given request and writes results.
     * @param request
     * @param response
```

```
     */
    private void handleRequest(SlingHttpServletRequest request,
SlingHttpServletResponse response)
    {
            /* prikazano kod Oak Lucene pretrage ... */
            }
            else if
(StringUtils.equalsIgnoreCase(request.getParameter(PARAM_SEARCH_ENGINE),
"solr"))
            {
                results = getSolrResults(request);
            }
            else if
(StringUtils.equalsIgnoreCase(request.getParameter(PARAM_SEARCH_ENGINE),
"elasticsearch"))
            {
                results = getElasticsearchResults(request);
            }

            writeJson(results, response);
        }
    }
```

Prikazana metoda na osnovu parametra iz zahtjeva poziva metodu za pretragu *Solr* rezultata – *getSolrResults* i rezultate sprema u varijablu „*results*“ koju ispisuje na korisničko sučelje.

```
/**
     * This method takes request to return search results by querying index
on remote Solr server
     * @param request
     * @return list of search results
     */
    private List<ListItem> getSolrResults(SlingHttpServletRequest request)
    {
        List<ListItem> results = new ArrayList<>();
        String fulltext = request.getParameter(PREDICATE_FULLTEXT);
        String searchContentType =
request.getParameter(PROP_SEARCH_CONTENT_TYPE);

        SolrQuery query = new SolrQuery();
        addPredicatesToQuery(searchContentType, query, fulltext);
```

```
        try (HttpSolrClient server = new
HttpSolrClient(getSolrServerUrl(this.solrConfigurationService)))
        {
            QueryResponse response = server.query(query);
            SolrDocumentList resultDocs = response.getResults();

            addResultDocsToResultItemList(resultDocs, request, results);
        }
        catch (SolrServerException | IOException e)
        {
            LOGGER.error("Exception due to ", e);
        }

        return results;
    }
```

Metoda *getSolrResults* se spaja na *Solr REST* klijenta te šalje upit. Dobivene rezultate pretvara u odgovarajući format te dodaje u listu rezultata.

```
/**
     * This method takes lucene query results to create Solr documents
ready for indexing
     * @param resultDocs
     * @param request
     * @param results
     */
    public static void addResultDocsToResultItemList(SolrDocumentList
resultDocs, SlingHttpServletRequest request, List<ListItem> results)
    {
        ResourceResolver resourceResolver = request.getResourceResolver();

        for (SolrDocument solrDocument : resultDocs)
        {
            String path = solrDocument.getFieldValue("id").toString();
            Resource res = resourceResolver.getResource(path);

            if (res != null)
            {
                if (res.getResourceType().equals(NT_PAGE))
```

```
                {
                    addPageToResultsList(results, request, res);
                }
                else if (res.getResourceType().equals(NT_DAM_ASSET))
                {
                    addAssetToResultsList(results, request, res);
                }
            }
        }
    }
```

Metoda „*addResultDocsToResultItemList“* dodaje rezultate pretrage u obliku stranice ili slike u listu svih rezultata. Točnije, pozivaju se sljedeće metode za dodavanje rezultata u listu.

```
/**
     * This method adds page to result list
     * @param results
     * @param request
     * @param hitRes
     */
    public static void addPageToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Page page = getPage(hitRes);

        if (page != null)
        {
            results.add(new PageListItemImpl(request, page));
        }
    }

    /**
     * This method adds asset to result list
     * @param results
     * @param request
     * @param hitRes
     */
```

```
    public static void addAssetToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Asset asset = getAsset(hitRes);

        if (asset != null)
        {
            results.add(new AssetListItemImpl(request, asset));
        }
    }
```

Indeksiranje sadržaja preko odvojenog *Elasticsearch* poslužitelja

Implementacija *Elasticsearch* poslužitelja je u osnovi jako slična implementaciji *Solr* poslužitelja. Razlika je u tome što se za rad s *Elasticsearchom* ne moraju postavljati konfiguracije kao u *Solr*-u. Konfiguracijske datoteke za indeks *Elasticsearcha* postoje i to su *.yaml* datoteke, ali se polja definiraju prema prvom polju koje se dobije u indeksu. Ako je prvo polje naziva *manualCreationDate* prepoznato kao datum, tada će svako iduće polje tog naziva biti indeksirano kao datum. Također, indeksiranje nije potrebno obavljati u modificiranim objektima, nego se to može jednostavno učiniti preko *JSON* objekata i polja. Razlika od *Solr*-a je i u tome da administratorsko *Web* sučelje od *Elasticsearch*-a nije besplatno nakon perioda od 14 dana.

Prikaz *Elasticsearch* konfiguracijske datoteke:

```
# ======================== Elasticsearch Configuration
=========================

# ---------------------------------- Cluster -----------------------------
-----
#
# Use a descriptive name for your cluster:
#
cluster.name: elasticsearch_hrzz00hn
#
# ---------------------------------- Paths -------------------------------
-----
#
# Path to directory where to store the data (separate multiple locations by
comma):
#
path.data: /usr/local/var/lib/elasticsearch/
#
# Path to log files:
#
path.logs: /usr/local/var/log/elasticsearch/

indices.query.bool.max_clause_count: 100000
```

U posljednjoj liniji proširen je broj maksimalnih zapisa unutar jednog indeksa što će biti potrebno u svrhu testiranja brzine.

Najveći problem pri implementaciji *Elasticsearch*-a predstavljao je broj tranzitivnih zavisnosti koje je potrebno uključiti u *OSGi* paket koji pri tome gubi modularnost, što je najveća prednost *OSGi* paketa. Rješenje za ovaj problem nalazi se u korištenju starijeg načina dodavanja tranzitivnih zavisnosti preko *apps* modula gdje će sve veće zavisnosti potrebe za rad s *Elasticsearch API*-jem biti dodane kao poseban *OSGi* paket. To je prikazano sljedećim kodom:

```
<!-- ================================================================= -->
<!-- V A U L T   P A C K A G E   P L U G I N                           -->
<!-- ================================================================= -->
<plugin>
   <groupId>com.day.jcr.vault</groupId>
   <artifactId>content-package-maven-plugin</artifactId>
   <extensions>true</extensions>
       <configuration>
          <filterSource>
            src/main/content/META-INF/vault/filter.xml
          </filterSource>
```

```
<verbose>true</verbose>
<failOnError>true</failOnError>
<group>adobe/aem6/sample</group>
<properties>
    <acHandling>merge_preserve</acHandling>
</properties>
<embeddeds>
   <embedded>
       <groupId>com.adobe.cq.sample</groupId>
       <artifactId>we.retail.core</artifactId>
       <target>/apps/weretail/install</target>
   </embedded>
   <embedded>
       <groupId>org.elasticsearch.client</groupId>
       <artifactId>
          elasticsearch-rest-high-level-client
       </artifactId>
       <target>/apps/weretail/install</target>
   </embedded>
   <embedded>
       <groupId>org.elasticsearch</groupId>
       <artifactId>elasticsearch</artifactId>
       <target>/apps/weretail/install</target>
   </embedded>
   <embedded>
        <groupId>org.elasticsearch.plugin</groupId>
        <artifactId>rank-eval-client</artifactId>
        <target>/apps/weretail/install</target>
   </embedded>
</embeddeds>
 ...
```

Pri tome je potrebno ručno preuzeti *.jar* datoteke s *Maven Repozitorija* i pretvoriti ih u *OSGi* pakete. Najlakši način za to napraviti je koristeći *Eclipse IDE* koji budi izradu *OSGi* bundlea i automatski generira manifest datoteku iz zavisnosti potrebnih u vremenu izvršavanja programa.

Kao što je to učinjeno ranije, potrebno je dodati mogućnosti indeksiranja i brisanja indeksa kroz korisničko sučelje.

Slika 51 - *Elasticsearch* korisničko sučelje

Indeksiranje se obavlja preko *ElasticsearchServlet* klase na sljedeći način:

```
/**
 *
 * This servlet acts as a bulk update to index content pages and assets to
the configured Elasticsearch server
 *
 */
@Component(service = Servlet.class, property = {
"sling.servlet.selectors=indexElasticsearchPages",
"sling.servlet.resourceTypes=cq/Page",
  "sling.servlet.extensions=json", "sling.servlet.methods=" +
HttpConstants.METHOD_GET })
public class ElasticsearchServlet extends SlingAllMethodsServlet
{
    private static final Logger LOG =
LoggerFactory.getLogger(ElasticsearchServlet.class);
    private static final String PARAM_INDEX_TYPE = "indexType";
    private static final String PROP_INDEX_TYPE_INDEX = "indexpages";
    private static final String PROP_INDEX_TYPE_DELETE = "deleteindexdata";

    @Reference
    ElasticsearchServerConfiguration esConfigService;

    @Reference
    EsService elasticsearchService;
```

U kodu iznad se uključe svi servisi koji će se koristiti i parametri za uspješno dohvaćanje zahtjeva.

```
    @Override
    protected void doGet(@Nonnull SlingHttpServletRequest request, @Nonnull
SlingHttpServletResponse response)
    {
        this.doPost(request, response);
    }

    /**
     * This method calls other methods for indexing content or deleting
indexed content from Elasticsearch server.
     * This is based on user's choice provided in given request.
     * This method writes response of successful or unsuccessful
indexing/deleting.
     * @param request
     * @param response
     */
    @Override
    protected void doPost(@Nonnull SlingHttpServletRequest request,
@Nonnull SlingHttpServletResponse response)
    {
        response.setContentType(CT_TEXT_HTML);
        String indexType = request.getParameter(PARAM_INDEX_TYPE);

        try (RestHighLevelClient client =
getEsClient(this.esConfigService); ResourceResolver resourceResolver =
request.getResourceResolver())
        {
            if (indexType.equalsIgnoreCase(PROP_INDEX_TYPE_INDEX))
            {
                indexData(response, client, resourceResolver);
            }
            else if (indexType.equalsIgnoreCase(PROP_INDEX_TYPE_DELETE))
            {
                deleteData(client, response);
            }
        }
        catch (ElasticsearchException ex)
        {
            handleElasticsearchException(ex);
```

```
        }
        catch (IOException | RepositoryException ex)
        {
            LOG.error("Elasticsearch exception: {}", ex);
        }
    }
```

S obzirom na vrijednost primljenog parametra iz zahtjeva, pozivaju se metode za indeksiranje sadržaja ili brisanje sadržaja iz indeksa.

```
    /**
     * This method performs indexing.
     * This is done by taking Elasticsearch server client and
resourceResolver to index data and writes indexing response
     * @param response
     * @param client
     * @param resourceResolver
     * @throws IOException
     * @throws RepositoryException
     */
    private void indexData(SlingHttpServletResponse response,
RestHighLevelClient client, ResourceResolver resourceResolver) throws
IOException, RepositoryException
    {
        Session session = resourceResolver.adaptTo(Session.class);
        String indexName =
this.esConfigService.getElasticsearchIndexName();

        GetIndexRequest getIndexRequest = new GetIndexRequest(indexName);
        boolean indesNameExists = client.indices().exists(getIndexRequest,
RequestOptions.DEFAULT);

        if (indesNameExists)
        {
            response.getWriter().write("Index already exists. Choose
different index name.");
        }
        else
        {
            List<XContentBuilder> builders =
this.elasticsearchService.crawlContent(session);
```

```
            CreateIndexRequest createIndexRequest = new
CreateIndexRequest(indexName);
            setIndexRequestOptions(createIndexRequest);
            client.indices().create(createIndexRequest,
RequestOptions.DEFAULT);


            BulkRequest bulkRequest = new BulkRequest();
            addDocsToRequest(bulkRequest, builders, indexName);


            BulkResponse indexResponse = client.bulk(bulkRequest,
RequestOptions.DEFAULT);
            writeResponse(indexResponse, response);
        }
    }
```

Pozivom metode crawlContent dobivamo listu *JSON* objekata koja se šalje u *Elasticsearch* preko *BulkRequest* klase, koja označava indeksiranje više dokumenata paralelno. Rezultat te akcije se ispisiva na korisničko sučelje.

```
    /**
     * This method adds retrieved indexed documents to bulk request needed
for synchronous indexing of many documents
     * @param bulkRequest
     * @param builders
     * @param indexName
     */
    private void addDocsToRequest(BulkRequest bulkRequest,
List<XContentBuilder> builders, String indexName)
    {
        for (XContentBuilder builder : builders)
        {
            String id = Integer.toString(builders.indexOf(builder));
            IndexRequest indexRequest = new
IndexRequest(indexName).id(id).source(builder);
            bulkRequest.add(indexRequest);
        }
    }
```

Prethodna metoda označava dodavanje zahtjeva za indeksiranjem dokumenta u *BulkRequest* zahtjev.

```
/**
     * This method takes search results and converts every result hit to
the JSON page or JSON asset object.
     * @param searchResults
     * @return list of json builders
     * @throws IOException
     * @throws RepositoryException
     */
    private List<XContentBuilder> createResultsList(SearchResult
searchResults) throws IOException, RepositoryException
    {
        ArrayList<XContentBuilder> builders = new ArrayList<>();

        for (Hit hit : searchResults.getHits())
        {
            Resource resource = hit.getResource();
            if (StringUtils.equals(resource.getResourceType(), NT_PAGE))
            {
                builders.add(createPageObject(resource, hit));
            }
            else if (StringUtils.equals(resource.getResourceType(),
NT_DAM_ASSET))
            {
                builders.add(createAssetObject(resource));
            }
        }

        return builders;
    }
```

Prethodna metoda kreira *JSON* objekt stranice ili slike od dobivenih rezultata.

```
    /**
     * This method takes resource and result hit to create a page object
     * that will be indexed as a document on ES server.
     * @param resource
     * @param hit
     * @return json builder
     * @throws RepositoryException
     * @throws IOException
     */
```

```
    private XContentBuilder createPageObject(Resource resource, Hit hit)
throws RepositoryException, IOException
    {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        ValueMap vm = hit.getProperties();

        builder.startObject();
        builder.field(PROP_SEARCH_ID, resource.getPath());
        builder.field(PROP_SEARCH_TYPE, resource.getResourceType());
        builder.field("jcr_title", vm.get(JCR_TITLE));

        addFields(builder, vm, JCR_CREATED, PN_PAGE_LAST_MOD,
"manualCreationDate", JCR_DESCRIPTION, JCR_NAME, JCR_PRIMARYTYPE,
SLING_RESOURCE_TYPE_PROPERTY, TagConstants.PN_TAGS, "searchDescription",
"pageImportanceRank");
        builder.endObject();
        return builder;
    }

    /**
     * This method takes resource of given AEM node and creates an asset
object
     * that will be indexed as a document on ES server
     * @param resource
     * @return json builder
     * @throws IOException
     */
    private XContentBuilder createAssetObject(Resource resource) throws
IOException
    {
        XContentBuilder builder = XContentFactory.jsonBuilder();
        ValueMap vm = getValueMapFromResource(resource);

        builder.startObject();
        builder.field(PROP_SEARCH_ID, resource.getPath());
        builder.field(PROP_SEARCH_TYPE, resource.getResourceType());
        builder.field("damAssetId", resource.getPath());

        addFields(builder, vm, JcrConstants.JCR_MIMETYPE,
DamConstants.PN_NAME, DamConstants.PN_PARENT_PATH,
DamConstants.DAM_ASSET_RELATIVE_PATH, TagConstants.PN_TAGS, "description",
"sellingPriority", "creationDate");
        builder.endObject();
```

```
        return builder;
    }
```

Prethodne dvije metode kreiraju *JSON* objekte stranica i slika spremne za indeksiranje.

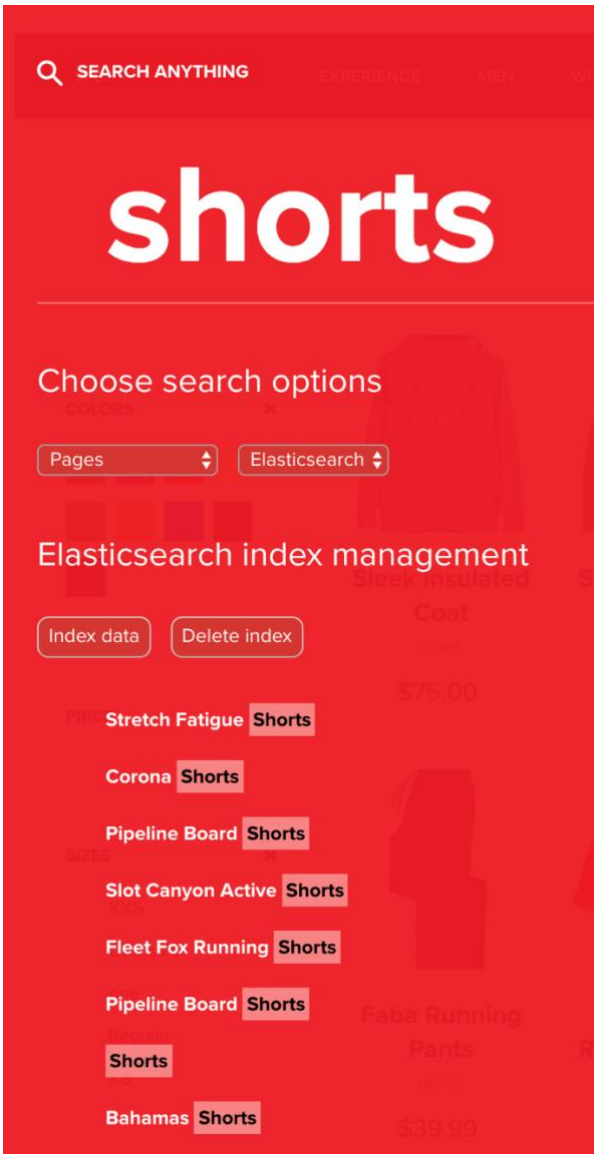
```
    /**
     * This method takes Elasticsearch server client to delete indexed data
and write response
     * @param client
     * @param response
     * @throws IOException
     */
    private void deleteData(RestHighLevelClient client,
SlingHttpServletResponse response) throws IOException
    {
        String indexName =
this.esConfigService.getElasticsearchIndexName();


        DeleteIndexRequest deleteRequest = new
DeleteIndexRequest(indexName);
        AcknowledgedResponse deleteIndexResponse = null;
        boolean acknowledged = false;


        try
        {
            deleteIndexResponse = client.indices().delete(deleteRequest,
RequestOptions.DEFAULT);
            acknowledged = deleteIndexResponse.isAcknowledged();
        }
        catch (ElasticsearchException ex)
        {
            LOG.error("Index can not be found", ex);
        }
        finally
        {
            writeResponse(acknowledged, response, indexName);
        }
    }
}
```

Prethodna metoda šalje zahtjev za brisanjem indeksa na *Elasticsearch* poslužitelj.

Pretraga preko *Elasticsearch* sustava pretraživanja

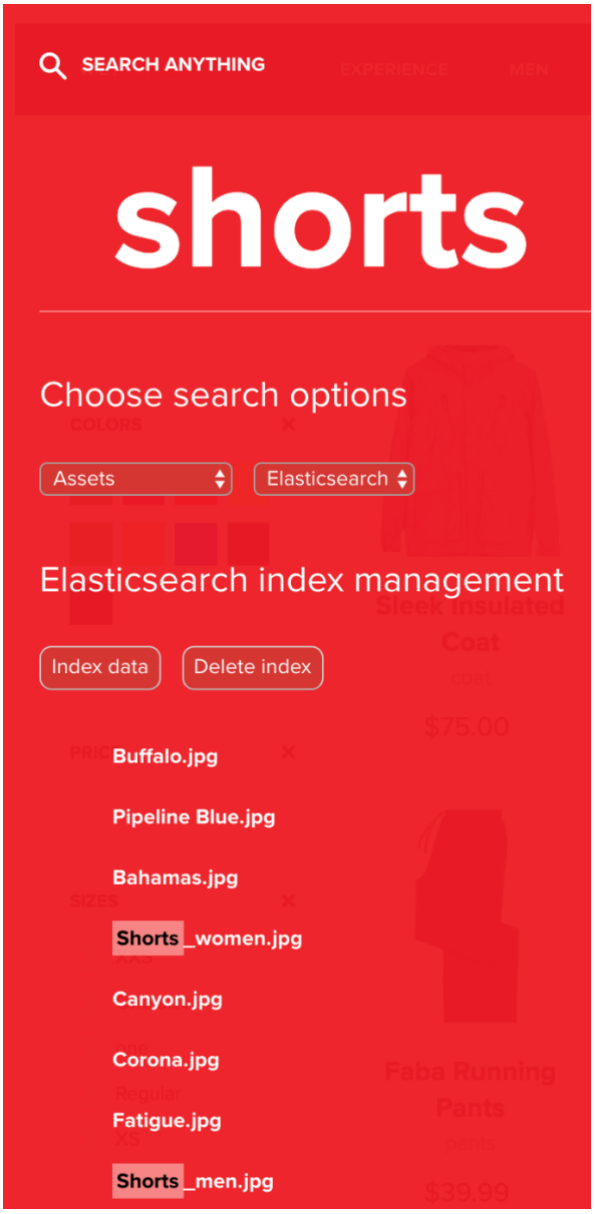
Pretraga preko *Elasticsearch* poslužitelja funkcionira slično kao kod *Solr*-a. Slijedi prikaz pretrage.

Pretraga stranica i prikaz prvih osam rezultata:



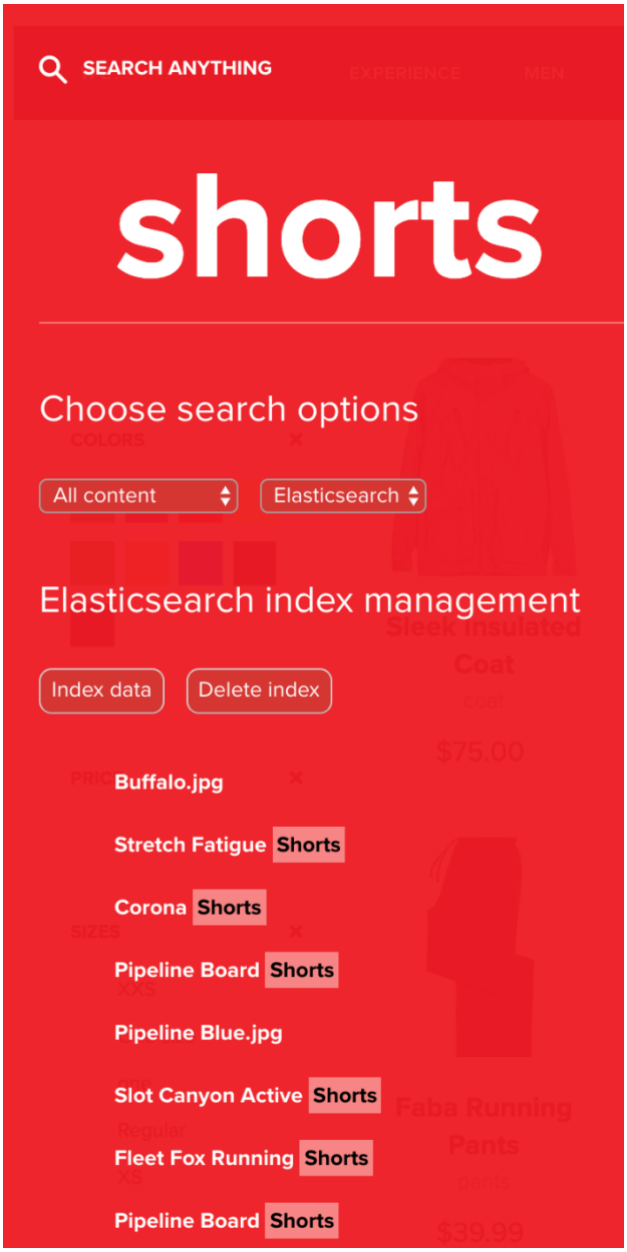
Slika 52 - Pretraga stranica

Pretraga slika i prikaz prvih osam rezultata:



Slika 53 - Pretraga slika

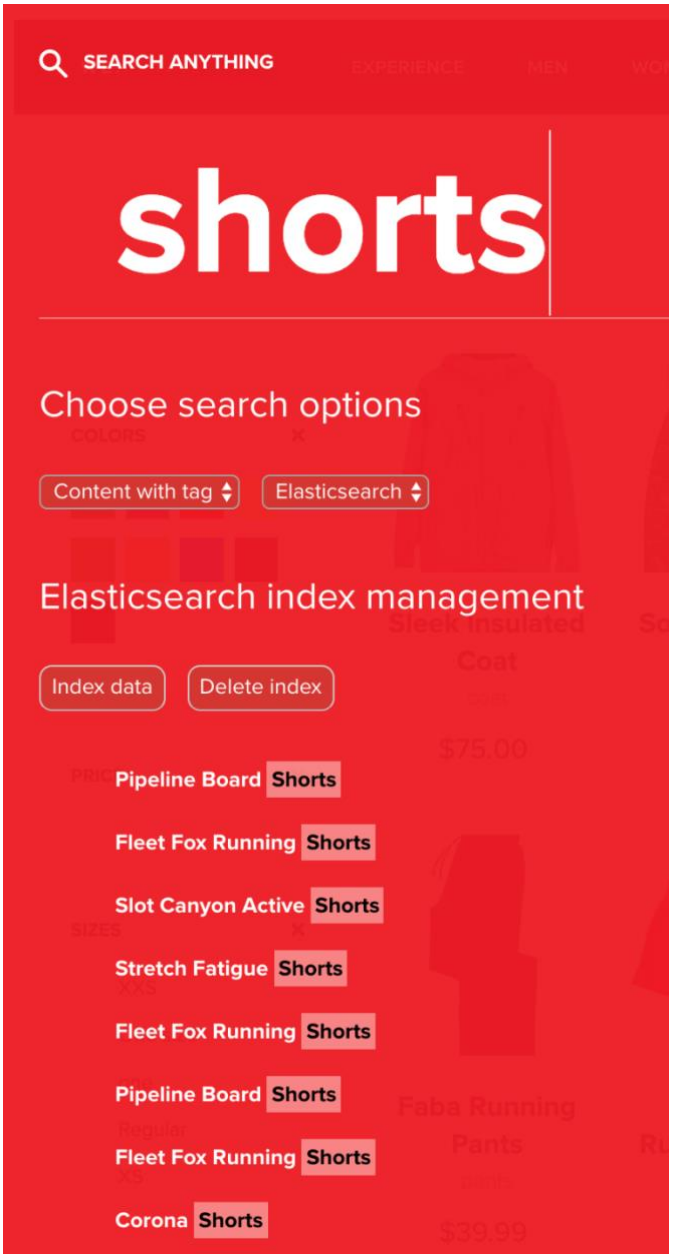
Pretraga cjelokupnog sadržaja i prikaz prvih osam rezultata:



Slika 54 - Pretraga cjelokupnog sadržaja

Pri pretrazi cjelokupnog sadržaja kombinacija rezultata je slična kao kod pretrage pomoću *Solra*.

Pretraga sadržaja po oznakama i prikaz prvih osam rezultata:



Slika 55 - Pretraga sadržaja po oznakama

Sve prikazano ostvareno je preko iste klase kao što je to slučaj bio sa *Luceneom* i *Solrom* – *SearchServlet*.

```
/**
     * This method takes request to return search results by querying index on remote Elasticsearch server
     * @param request
     * @return list of search results
```

```
     */
    private List<ListItem> getElasticsearchResults(SlingHttpServletRequest
request)
    {
        List<ListItem> results = new ArrayList<>();
        String fulltext = request.getParameter(PREDICATE_FULLTEXT);
        String searchContentType =
request.getParameter(PROP_SEARCH_CONTENT_TYPE);
        String indexName =
this.esConfigService.getElasticsearchIndexName();

        try (RestHighLevelClient client =
getEsClient(this.esConfigService))
        {
            BoolQueryBuilder boolQueryBuilder = getBoolQuery(fulltext,
searchContentType);
            SearchSourceBuilder sourceBuilder =
getSourceBuilder(boolQueryBuilder);
            SearchRequest searchRequest = getSearchRequest(sourceBuilder,
indexName);

            SearchResponse searchResponse = client.search(searchRequest,
RequestOptions.DEFAULT);

            addResultsToResultItemList(results, request, searchResponse);
        }
        catch (ElasticsearchException | IOException e)
        {
            LOGGER.error("Exception due to ", e);
        }

        return results;
    }
```

Najprije je potrebno inicijalizirati klijenta za komunikaciju s *Elasticsearch* poslužiteljem. Nakon toga se pripremi upit s parametrima potrebnim ovisno o korisnikovom izboru (stranice, slike, cijeli sadržaj ili sadržaj prema oznakama). Svi rezultati se dodaju u listu rezultata.

Slijedi prikaz pripreme upita.

```
/**
* This method takes user's search input (fulltext) parameter and search
type (searchContentType) parameter
* to return query based on these parameters
* @param fulltext
* @param searchContentType
* @return
*/
public static BoolQueryBuilder getBoolQuery(String fulltext, String
searchContentType)
{
BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();

if (searchContentType.equalsIgnoreCase(PROP_SEARCH_TAGS))
{
boolQueryBuilder.should(QueryBuilders.matchQuery("tags", "" + fulltext +
""));
}
else
{
boolQueryBuilder.must(QueryBuilders.queryStringQuery(fulltext));

if (searchContentType.equalsIgnoreCase(PROP_SEARCH_ASSETS))
{
boolQueryBuilder.must(QueryBuilders.matchPhraseQuery(PROP_SEARCH_TYPE,
NT_DAM_ASSET));
}
else if (searchContentType.equalsIgnoreCase(PROP_SEARCH_PAGES))
{
boolQueryBuilder.must(QueryBuilders.matchPhraseQuery(PROP_SEARCH_TYPE,
NT_PAGE));
}
}
return boolQueryBuilder;
}
```

Slijedi prikaz dodavanja rezultata pretrage u listu.

```
/**
     * This method takes search results and request to convert each result
to data format needed for indexing
     * @param results
     * @param request
     * @param searchResponse
     */
    public static void addResultsToResultItemList(List<ListItem> results,
SlingHttpServletRequest request, SearchResponse searchResponse)
    {
        long numberOfHits = searchResponse.getHits().getTotalHits().value;
        LOG.info("Number of hits recieved from index: {}", numberOfHits);

        ResourceResolver resourceResolver = request.getResourceResolver();
        SearchHits hits = searchResponse.getHits();
```

```
        for (SearchHit hit : hits)
        {
            Map<String, Object> sourceAsMap = hit.getSourceAsMap();

            String path = (String) sourceAsMap.get("id");
            Resource res = resourceResolver.getResource(path);

            if (res != null)
            {
                if (res.getResourceType().equals(NT_PAGE))
                {
                    addPageToResultsList(results, request, res);
                }
                else if (res.getResourceType().equals(NT_DAM_ASSET))
                {
                    addAssetToResultsList(results, request, res);
                }
            }
        }
    }
```

Slijedi prikaz dodavanja rezultata u objekt stranice ili slike koji su potrebni za prikaz na korisničkom sučelju.

```
/**
     * This method adds page to result list
     * @param results
     * @param request
     * @param hitRes
     */
    public static void addPageToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Page page = getPage(hitRes);

        if (page != null)
        {
            results.add(new PageListItemImpl(request, page));
        }
    }
```

```
    /**
     * This method adds asset to result list
     * @param results
     * @param request
     * @param hitRes
     */
    public static void addAssetToResultsList(List<ListItem> results,
SlingHttpServletRequest request, Resource hitRes)
    {
        Asset asset = getAsset(hitRes);

        if (asset != null)
        {
            results.add(new AssetListItemImpl(request, asset));
        }
    }
```

10. Zaključak

Adobe Experience Manager nije i vjerojatno nikad neće biti sustav pretraživanja, međutim da bi funkcionirao koristi se pretragom. U *AEM*-u je pretraga jednako potrebna vanjskim korisnicima kao i administratorima, marketing stručnjacima, autorima sadržaja i pomaže da se *AEM* savršeno uklopi u ostatak skupa Adobeovih tehnologija.

Pretrage na većini *Web* aplikacija su osnova korištenja i znatno utječu na korisničko iskustvo. Osnovna pretraga je nekada dovoljna i *AEM* preko svoje implementacije *Lucene* indeksiranja i pretrage nudi mogućnost jednostavne implementacije sustava pretraživanja. Taj sustav pretraživanja nudi visoku razinu efikasnosti, relevantnosti i brzine korištenja. Preporučuje se upotreba integriranog sustava pretraživanja kada god poslovni slučaj pruža tu mogućnosti razvojnim programerima.

Međutim to nije često slučaj. Integrirani sustav pretraživanja zahtjeva podatke spremljene u bazu podataka što može dovesti do nestabilnosti cijele aplikacije zbog nemogućnosti dovoljne skalabilnosti, brzine i memorijskih resursa. Tada je potrebno uvesti neki od sustava pretraživanja.

U ovom radu uspješno je testirana integracija *Apache Solra* i *Elasticsearcha*. Prema autorovom subjektivnom mišljenju, manju prednost zbog generalnog dojma ima *Elasticsearch*, iako su u osnovi oba sustava pretraživanja bazirana na *Apache Lucene* tehnologiji i između njih je na većini osnovnih poslovnih slučajeva razlika jedna uočljiva.

U provedenoj usporedbi u poglavlju 7. Usporedba sustava pretraživanja *Apache Solr* i *Elasticsearch*, ukupni zbroj ocjena (od ukupno 100) je sljedeći:

- *Apache Solr*: 94
- *Elasticsearch*: 95

Prednost *Elasticsearcha* uglavnom je bazirana na boljoj dokumentaciji za programski jezik *Javu* i točno definiranim ograničenjima u korištenju određenih funkcionalnosti. Negativna strana je ta što se Administratorsko sučelje naplaćuje dok je kod *Solra* besplatno. Za potrebe testiranja integracije, svi sustavi pretraživanja su imali mogućnost implementacije potrebnih funkcionalnosti pretrage.

Popis literature

[1]"Internet Growth Statistics 1995 to 2019 - the Global Village Online", *Internetworldstats.com*, 2019. [Na internetu]. Dostupno: https://www.internetworldstats.com/emarketing.htm. [Pristupljeno: 14- srp- 2019].

[2]"World Internet Users Statistics and 2019 World Population Stats", *Internetworldstats.com*, 2019. [Na internetu]. Dostupno: https://www.internetworldstats.com/stats.htm. [Pristupljeno: 14- srp - 2019].

[3]B. Marr, "How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read", *Forbes.com*, 2019. [Na internetu]. Dostupno: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#7834d86c60ba. [Pristupljeno: 14- srp - 2019].

[4]J. Desjardins, "How Much Data is Generated Each Day?", *Visual Capitalist*, 2019. [Na internetu]. Dostupno: https://www.visualcapitalist.com/how-much-data-is-generated-each-day/. [Pristupljeno: 14- srp - 2019].

[5]R. Kwok, "Best Practice #2: Maintain a crash rate of less than 0.25% in your app's three most critical userflows - Apteligent", *Apteligent*, 2019. [Na internetu]. Dostupno: https://www.apteligent.com/technical-resource/best-practice-2-maintain-a-crash-rate-of-less-than-0-25-in-your-apps-three-most-critical-userflows/. [Pristupljeno: 14- srp - 2019].

[6]G. Linden, "Marissa Mayer at Web 2.0", *Glinden.blogspot.com*, 2019. [Na internetu]. Dostupno: https://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html. [Pristupljeno: 14- srp - 2019].

[7]"Adobe Named a Leader in Gartner Magic Quadrant for Web Content Management for Eighth Consecutive Year", *Adobe Newsroom*, 2019. [Na internetu]. Dostupno: https://news.adobe.com/press-release/experience-cloud/adobe-named-leader-gartner-magic-quadrant-web-content-management-0. [Pristupljeno: 14- srp - 2019].

[8]"Benefits of Using OSGi – OSGi™ Alliance", *Osgi.org*, 2019. [Na internetu]. Dostupno: https://www.osgi.org/developer/benefits-of-using-osgi/. [Pristupljeno: 14- srp - 2019].

[9]"Apache Sling :: The Sling Engine", *Sling.apache.org*, 2019. [Na internetu]. Dostupno: https://sling.apache.org/documentation/the-sling-engine.html. [Pristupljeno: 14- srp - 2019].

[10]"Developing User Guide", *Helpx.adobe.com*, 2019. [Na internetu]. Dostupno: https://helpx.adobe.com/in/experience-manager/6-3/sites/developing/user-guide.html. [Pristupljeno: 14- srp - 2019].

[11]S. Closser, *Adobe Experience Manager Quick-Reference Guide: Web Content Management [formerly CQ]*, 1st ed. United States of America: Adobe Press, 2013.

[12]"Getting Started with HTL", *Docs.adobe.com*, 2019. [Na internetu]. Dostupno: https://docs.adobe.com/content/help/en/experience-manager-htl/using/getting-started/getting-started.html. [Pristupljeno: 14- srp - 2019].

[13]*Extend and Customize Adobe Experience Manager Student Guide*. ADOBE DIGITAL LEARNING SERVICES, 2017.

[14]E. Ng and V. Mohan, *Lucene 4 Cookbook*. Packt Publishing, 2015.

[15]M. McCandless, E. Hatcher and O. Gospodnetić, *Lucene in action*. Stamford, Conn.: Manning Pub., 2010.

[16]A. Gazzarini, *Apache Solr essentials*. Packt Publishing, 2015.

[17]D. Smiley, E. Pugh, K. Parisa and M. Mitchell, *Apache Solr Enterprise Search Server - Third Edition*, 3rd ed. Packt Publishing, 2015.

[18]A. Ahlawat and A. Ahlawat, "Adobe AEM History | AEM CQ5 Tutorials", *AEM CQ5 Tutorials*, 2019. [Online]. Available: http://www.aemcq5tutorials.com/tutorials/adobe-aem-history/. [Accessed: 14- Sep- 2019].

[19]Adobe, "Oak Queries and Indexing", *Helpx.adobe.com*, 2019. [Online]. Available: https://helpx.adobe.com/uk/experience-manager/6-4/sites/deploying/using/queries-and-indexing.html. [Accessed: 14- Sep- 2019].

Popis slika

Popis tablica