UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Ana Vuksić

PRIKAZ SISTEMA BLOOMREACH I SERVISA ELASTICSEARCH NA PRIMERU RAZVOJA APLIKACIJE ZA PRETRAGU I RAZMENU IMDB RECENZIJA

master rad

Beograd, 2022.

Mentor:

dr Milan Banković, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Vladimir Filipović, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Aleksandar Kartelj, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Prikaz sistema Bloomreach i servisa Elasticsearch na primeru razvoja aplikacije za pretragu i razmenu IMDB recenzija

Rezime: U ovom radu će biti predstavljen razvoj veb aplikacije za pretragu i razmenu recenzija o filmovima iz IMDB baze pomoću sistema za upravljanje sadržajem kompanije Bloomreach, koja se naziva Bloomreach Experience Manager i servisa Elasticsearch. Pored ovih tehnologija, koristi se i Kotlin programski jezik za razvoj logike aplikacije i komunikaciju između Bloomreach Expereience Manager sistema i Elasticsearch servisa. Elasticsearch će biti korišćen kao baza podataka i servis za pretragu. Za razvoj prednjeg dela aplikacije se koristi Freemarker šablon zajedno za Bootstrap bibliotekom. Korišćeni su podaci o filmovima iz IMDB skupa podataka.

Ključne reči: veb tehnologije, Bloomreach, Elasticsearch, sistemi za upravljanje sadržajem, sistemi za pretragu podataka

Sadržaj

Glava 1

Uvod

1.1 Motivacija

Početak veba kao javno dostupnog servisa se vezuje za avgust 1991. godine kada je Tim Berners-Li objavio prvi sajt ikada. Ideja je proistekla iz želje da se napravi mreža za lako deljenje naučnih radova iz fizike i da ti radovi budu povezani u fusnotama. Godinama pre toga Tim Berners-Li je radio na razvitku tehnologija potrebnih za funkcionisanje veba. Ovo uključuje jezik za označavanje hipertekstova (engl. *Hypertext Markup Language - HTML*), protokol za prenos hiperteksta (engl. *Hypertext Transfer Protocol - HTTP*), kao i veb adrese (engl. *Uniform Resource Locators - URLs*)), kako bi bilo moguće pristupiti dokumentima i stranicama. Izgled tadašnjih veb stranica i veb stranica kakve viđamo danas se znatno razlikuju, kao i tehnologije koje su korišćene za njihovu izradu. Ipak, suština je ostala ista, a to je potreba za deljenjem informacija.

Zbog velike konkurencije i borbe za svakog novog posetioca, veb stranice postaju tehnološki sve naprednije i sadržajnije. Korisnici zahtevaju sveže, transparentne i ažurne informacije. Moderna, pristupačna i sadržajna veb prezentacija postaje sve značajniji aspekt poslovanja i delovanja svake kompanije, institucije ili pojedinca. Veb sistemi za upravljanje sadržajem (engl. *Web Content Management System - WCMS*) pomažu da se kreira, postavi i održava veb stranica koja ispunjava zahteve korisnika. Ovi sistemi se mogu koristiti i za ažuriranje podataka od strane tehnički neobučenih lica, čime se postiže značajna ušteda vremena i novca.

U današnje vreme smo svedoci sve veće količine informacija koje se generišu na dnevnom nivou. Kako je veb postajao sve pristupačniji, tako su počele da se pojavljuju i dele sve veće količine podataka na mreži. Samo u 2020. je generisano oko

2.5 bilijardi[1] podataka dnevno na mreži. S obzirom na ovakav stepen prezasićenosti informacijama, nemoguće je setiti se svega. Zato i postoji veliki broj alata koji pomažu da se efikasno zapamte i pretraže sve te informacije. U današnje vreme se samo u jednom danu na Google[2] pretraživaču pokrene preko 3.5 milijardi pretraga[3]. Korisnici su sve naviknutiji na oblik pretrage koje nudi ovaj poznati pretraživač. Više nije dovoljno da samo postoje relevantne i ažurne informacije, potrebno je da se omogući korisnicima i efikasna pretraga tih informacija na veb sajtu. Informacije su većinski u tekstualnom obliku ili se mogu opisati tekstom, odnosno približno preslikati u tekstualnu formu. Otuda se možemo skoncentrisati na problem pretrage velike količine tekstualnih podataka.

Za poslednjih 30 godina, nastalo je mnogo novih veb tehnologija koje se bave ovim problemima. U ovom radu predstavljamo *Bloomreach Experience Manager* (u daljem tekstu BrXM) kao sistem za upravljanje sadržajem, kao i *Elasticsearch* kao servis za skladištenje i pretragu podataka. U cilju demonstracije ovih tehnologija, prikazano je i idejno rešenje za veb aplikaciju zasnovanu na ovim tehnologijama, čije su glavne karakteristike lakoća održavanja i od strane tehnički neobučenih lica, jednostavna proširivost i jednostavnost poveziavnja sa drugim aplikacijama.

1.2 Sistemi za upravljanje sadržajem

Sa pojavom računara nastaje i potreba da se u njima skladište dokumenti u elektronskoj formi. Kako su računari postajali sve pristupačniji, i sama količina sadržaja je postajala sve veća i komplikovanija za održavanje. Nastankom ogromne količine digitalnog sadržaja, nastaje i potreba da se tim sadržajem upravlja. Šire posmatrano, pod pojmom sistema za upravljanje sadržajem mogu se podrazumevati svaki informacioni sistem koji služi za rad sa bilo kojom vrstom sadržaja.

Svaki sistem za upravljanje sadržajem (engl. *Content Management System - CMS*) se sastoji od niza procedura koje se koriste za upravljanje procesom rada u zajedničkom okruženju. Neke od poželjnih osobina koje bi sistemi za upravljanje sadrzajem trebalo da imaju su [1]:

- uključuju intuitivni korisnički interfejs za osobe koje uređuju sadržaj,

[1] bilijarda - hiljadu biliona (1 000 000 000 000 000)

[2] Google - `www.google.com`

[3] Informacija je preuzeta sa sajta `https://www.internetlivestats.com/google-search-statistics/`

- podržavaju istovremeni rad više ljudi,
- podržavaju upravljanje nalozima članova i njihovim pravima,
- podržavaju višejezični način rada,
- veoma su modularni,
- permanentno su dostupni,
- sadrže korisnički interfejs za prikaz sadržaja,
- sadrže veb interfejs za administraciju,
- sadržaj je odvojen od interfejsa,
- sadržaj se uglavnom čuva u bazi podataka,
- interfejs se čuva u posebnim fajlovima,
- sadrže sistem šablona za lakše menjanje izgleda i rasporeda, korisničkog i administratorskog interfejsa,
- podržavaju uređivanje teksta preko mreže promoću WYSIWYG (engl. *What You See Is What You Get*) uređivača teksta.

Tradicionalni pristup izradi veb sajtova ima određene mane. Prethodno navedene karakteristike sistema za upravljanje sadržajem imaju za cilj da se te mane otklone, a proces postavljanja i održavanja sajta unapredi. Jedan od nedostataka klasičnog pristupa izradi i održavanju veb sajtova je ta što svaka izmena koja se napravi mora da se opet postaviti na server. Samim tim je potrebno da uvek bude prisutan neko sa određenim tehničkim znanjem čak i za najmanju izmenu. Sistemi za upravljanje sadržajem, između ostalog, olakšavaju postavljanje sadržaja na veb, i to direktno iz veb pregledača. Sama procedura je najčešće jednostavna – potrebno je pristupiti adresi na kojoj se nalazi administratorski ulaz u aplikaciju i sve promene obaviti preko korisničkog interfejsa. Po čuvanju promena, one se istog trenutka prikazuju na stranicama. Jednostavnost postavljanja i ažuriranja sadržaja je jedna od jako bitnih osobina, jer je u današnjem svetu sem izgleda i samog sadržaja, veoma bitna i ažurnost podataka na sajtu, koju ovim dobijamo. Dodatno, na ovaj način dobijamo i bolju podelu rada, jer je svako skoncentrisan samo na posao kojim se bavi. Lakoći

korišćenja svedoči i sve veći broj blogova koji se pojavljuju u današnje vreme. U pozadini svakog ovakvog sajta je jedan sistem za upravljanje sadržajem.

Sistemi za upravljanjem sadržajem najčešće podržavaju mogućnost održavanja sadržaja podsredstvom WYSIWYG uređivača teksta. Korišćenjem WYSIWYG uređivača teksta eliminiše se potreba da se dodatno stilizuje tekst od strane dizajnera pre njegovog postavljanja na sajt. Dovoljno je da urednici uz pomoć WYSIWYG uređivača teksta formatiraju tekst kako žele i upravo tako će se sadržaj i prikazati na sajtu (podebljan tekst, podvučen tekst, ubacivanje linkova u tekst itd.).

Značajna karakteristika ovih sistema je i da podržavaju istovremeni rad više ljudi na sadržaju sajta, što omogućava bržu dostavu veće količine sadržaja. Intuitivni korisnički interfejs omogućava brzu obuku novih lica koja uređuju sadržaj sajta. Otuda, ako ima previše posla za trenutni tim, ovaj tim se može brzo proširiti novim urednicima.

Jedna od veoma značajnih osobina sistema za upravljanje sadržajem je i modularnost. Modularnost znači da se aplikacija sastoji iz više delova, pri čemu svaki od delova predstavlja zasebnu celinu koja samostalno može funkcionisati. Moduli se mogu jednostavno uklapati i kombinovati sa drugim projektima.

Tipovi sistema za upravljanje sadražajem

S obzirom na svrhu korišćenja, postoji više tipova sistema za upravljanje sadržajem. Neke od najznačajnijih tipova sistema za upravljanje sadržajem su navedeni u nastavku[4]. Pritom, klasifikacija koja sledi nije striktna, s obzirom da se pojedini sistemi mogu koristiti u različite svrhe.

Sistemi za upravljanje sadržajem za kompanije – ECMS

ECMS (engl. *Enterprise Content Managment System*) se odnosi na tehnologije, metode, strategije i alate koji se koriste za pribavljanje, obradu, čuvanje, pripremu i dostavu podataka i dokumenata u nekoj organizaciji[5]. ECMS je kombinacija velikog broja tehnologija i komponenti, od kojih se neke mogu koristiti i samostalno.

[4]Više informacija o ovome može se naći na sajtu: `http://www.cms.co.uk/types/`

[5]Definicija ECMS-a je preuzeta sa veb prezentacije AIIM organizacije (engl. *Intellignet Information Management Community*): `https://info.aiim.org/what-is-ecm`

Sistemi za upravljanje dokumentima – DMS

DMS[6] (engl. *Document Managment System*) je sistem jedne ili više aplikacija koje se koriste za praćenje i čuvanje elektronskih dokumenata, fotografija i skeniranih papirnih dokumenata. Često dolazi kao komponenta ECMS-a, tako da se mogu smatrati i kategorijom za sebe, a i kao deo ECMS-a.

Sistemi za upravljanje sadržajem za mobilne telefone – MCMS

MCMS (engl. *Mobile Content Managment System*) je tip sistema za upravljanje sadržajem koji se koristi kod mobilnih uređaja. Ovaj tip CMS-a je u stanju da sačuva, obradi i dostavi podatke u obliku kakav podržavaju mobilni uređaji. Prilikom ispunjavanja svih zahteva mora se misliti i na kanal kojim će se podaci prenositi (npr. ograničena brzina Interneta). Takođe, prikaz podataka mora biti prilagođen prikazivanju na mobilnim uređajima. Jedna od bitnih stavki je i kompatibilnost sa što većim brojem uređaja.

Sistemi za upravljanje sadržajem na nivou komponenti – CCMS

CCMS (engl. *Component Content Management Systems*) vrši kontrolu dokumenata na nivou komponenata, za razliku od ostalih CMS-ova koji kontrolu vrše nad samim dokumentima. Svaka komponenta je celina za sebe. Komponenta može biti tema, koncept ili resurs (npr. slika) i čuva se na samo jednom mestu u memoriji. Svaka komponenta ima životni vek, vlasnika, verziju, ograničenja pristupa i korišćenja, te se može pratiti individualno ili kao deo veće celine. CCMS sistem može biti odvojeni sistem, a može i biti funkcionalnost nekog drugog sistema (kao npr. WCMS-a).

Veb sistemi za upravljanje sadržajem – WCMS

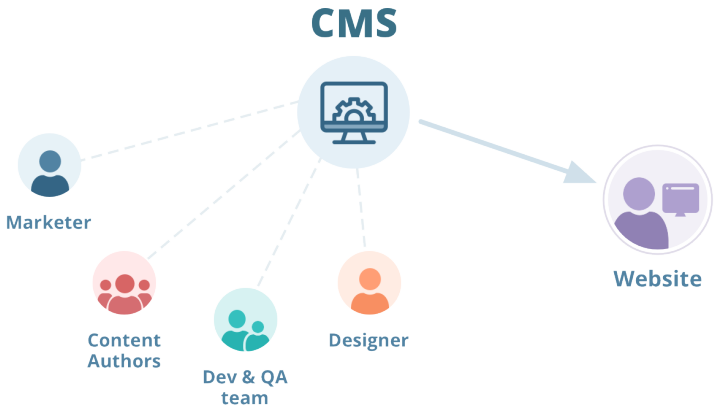
WCMS (engl. *Web Content Managment System*) je, sa stanovništva ovog rada, najznačajniji tip sistema za upravljanje sadržajem, s obzirom da je u nastavku rada upravo opisan jedan konkretan sistem ovog tipa. WCMS je implementiran kao veb aplikacija koja služi kreiranju i održavanju HTML sadržaja. Namenjen je održavanju velikih i dinamičnih HTML sadržaja. WCMS je po pravilu i deo ECMS-a. Osnovne funkcije koje WCMS treba da podržava su kreiranje, brisanje i održavanje

[6]Više informacija o DMS-u se može naći na sajtu: `https://www.aiim.org/what-is-document-imaging`

veb sadržaja. Može se smatrati da je WCMS podeljen na dva dela, s obzirom na logiku podele rada i uloga. To su administratorski i korisnički deo [14].

Administratorski deo podrazumeva mogućnost rada na sadržaju, tj. mogućnost izmene stranica promenom dokumenata koji se nalaze u sistemu, a koji se kasnije prikazuju korisniku. Osoba koja uređuje sadržaj se obično mora prijaviti na sistem pomoću veb pregledača kako bi dodavala ili menjala sadržaj. Nakon uređivanja sadržaja, potrebno je sačuvati promene, koje se po pravilu zapisuju u centralizovanu bazu podataka.

Korisnički deo je zadužen za čitanje sačuvanog sadržaja i njegov prikaz posetiocima sajta. Za korisnički deo je poželjno da se lakom izmenom koda i šablona u sistemu može promeniti izgled sajta. Ova dva dela funkcionišu potpuno nezavisno jedan od drugog, a zajedno čine jednu moćnu platformu za upravljanje sadržajem na internetu. Za čuvanje podataka WCMS uglavnom koristi baze podataka (npr. MySQL) ili XML datoteke.



Slika 1.1: WCMS

Na slici 1.1 se može videti klasična podela rada u veb sistemima za upravljanje sadržajem.[7] Vidi se da na sistemu za upravljanje sadržajem radi tim ljudi podeljenih u manje timove po tipu posla kojim se bave. Imamo tim osoba koje uređuju sadržaj, tim programera, dizajnere i tim za marketing. Sistemi za upravljanje sadržajem omogućavaju jasnu podelu rada, kako bi se svako skoncentrisao isključivo na svoj deo posla.

[7]Slika 1.1 je preuzeta sa sajta `https://www.logicify.com/en/blog/why-your-website-needs-a-content-management-system/`

Veb sisteme za upravljanje sadržajem se može podeliti na nekoliko vrsta, pri čemu ova podela na neki način oslikava i samu evoluciju ovih sistema [11].

Tradicionalni sistemi za upravljanje sadržajem. Ovo je najstariji tip veb sistema za upravljanje sadržajem. To su sistemi za upravljanje sadržajem koji su vezani za jedan sajt. Jedan od najpopularnijih tradicionalnih sistema je WordPress[8]. Kod tradicionalnih sistema, sajt i baza su tesno povezani sa CMS-om. [9] Glavni benefit ovih sistema je lakoća podizanja sajta, kao i količina dostupnih dodataka. Sa druge strane, glavne mane ovih sistema su to što sajt vremenom pri dodavanju sadržaja i dodataka postaje sve sporiji, to što nisu dovoljno bezbedni, kao i to što često i za najmanju promenu izgleda mora sve opet kompajlirati, s obzirom da je sajt tesno povezan sa CMS-om.

DXP sistemi. Platforme za digitalno iskustvo (engl. *Digital Experience Platform*) su jako kompleksni sistemi koji teže tome da ponude sve alate koji mogu da zatrebaju (online prodavnice, analize, pretrage, itd.), uz sam WCMS. Ovakvi sistemi su uglavnom namenjeni kompanijama. Glavni benefit jeste upravo veliki broj dodatnih funkcionalnosti koje ovi sistemi nude. Mane su kompleksnost, otežano korišćenje bez obuke, kao i cena. Jedan od predstavnika platformi za digitalno iskustvo je *SiteGlide*[10].

Bezglavi sistemi (engl. *Headless systems*) Ovo je najnoviji tip veb CMS-a koji se pojavio na tržištu. Kod ovih sistema je fokus na sadržaju koji je odvojen od sajta. Ovi sistemi čine sadržaj dostupnim putem aplikativnih programskih interfejsa (engl. *Application Programming Interface (API)*) za prikaz na bilo kom uređaju. Uglavnom je u pitanju rešenje u oblaku (engl. cloud) i podržava SaaS opciju[11]. Prednosti ovog pristupa su da su sami sistemi uglavnom prostiji od ostalih, da je izlaz moguće poslati na bilo koju drugu platformu, kao i da se aplikacija brzo pokreće. Mana ovih sistema je da je za pokretanje okruženja neophodan programer. Dobra strana ovog pristupa je i to što je jednostavno u budućnosti promeniti neku

[8] Wordpress: `https://wordpress.org/`

[9] U poslednje vreme nastaju i hibridni tradicionalni sistemi koji imaju dodatak (*Hybrid Headless APIs*) kako bi se omogućio pristup sadržaju drugih aplikacija.

[10] Više informacija o ovome se može naći na sajtu: `https://www.siteglide.com/`

[11] SaaS - Software as a Service. Više informacija se može naći na lokaciji: `https://www.techradar.com/news/what-is-saas`

od tehnologija koja se koristi za izgled sajta, kao i izvršiti integraciju sa nekom drugom aplikacijom. Jedan od predstavnika bezglavih sistema je *Agility CMS*[12].

Od sredine 2000-ih godina, uz komercijalne sisteme, na tržištu se nudi i veliki broj besplatnih CMS alata. Oni obično nude određen broj unapred definisanih šablona koji omogućuju brzu i jednostavnu izradu veb sajtova. Njihova prednost su dostupnost i cena, a nedostatak veliki broj gotovo identičnih veb-stranica nastalih primenom šablona i nedostatak mnogih naprednih funkcija koje nude komercijalni proizvodi. Najpoznatiji besplatni sistemi za upravljanje sadržajem su *Wordpress*, *Joomla*[13] i *Drupal*[14].

Sistem za upravljanje sadržajem koji se opisuje u ovom radu je *Bloomreach Experience Manager* (BrXM), bezglavi veb sistem za upravljanje sadržajem. BrXM je rešenje otvorenog koda koje nudi kompanija *Bloomreach* i koji je deo digitalne platforme namenjene kompanijama koja se naziva *Bloomreach Experience Platform (brX)*.

1.3 Sistemi za pretragu podataka

Može se reći da su sistemi za pretragu podataka stari koliko i svetska komunikaciona mreža (engl. World Wide Web). Veb pretraživači su internet servisi koji služe za pretragu veba. Ovi sistemi koriste skup naprednih algoritama za prolazak kroz veb i izvlačenje relevantnih stranica. Veb pretraživači funkcionišu tako što korisnik unese ključnu reč za pretragu u određeno polje i zatim se rezultat pretrage prikazuje u uređenoj listi. Klikom na neki od rezultata može se otvoriti odgovarajuća stranica.

Prvi veb sistem za pretragu podataka je bio *Aliweb* (engl. *Archie Like Indexing for the Web*, slika 1.2[15]), koji se pojavio u novembru 1993.[16] Aliweb je dozvoljavao administratorima da dodaju svoje stranice i odgovarajuće ključne reči u sistem. Tokom godina sami sistemi za pretragu su dosta uznapredovali i postali deo svakodnevnice. Sa pojavom Google pretraživača popularnost ovih sistema je naglo porasla. Novina koju je Google uveo i ono što ga je izdvojilo od drugih je bilo rangiranje rezultata. Smatralo se da ako ima puno hipertekstualnih linkova koji vode ka nekoj

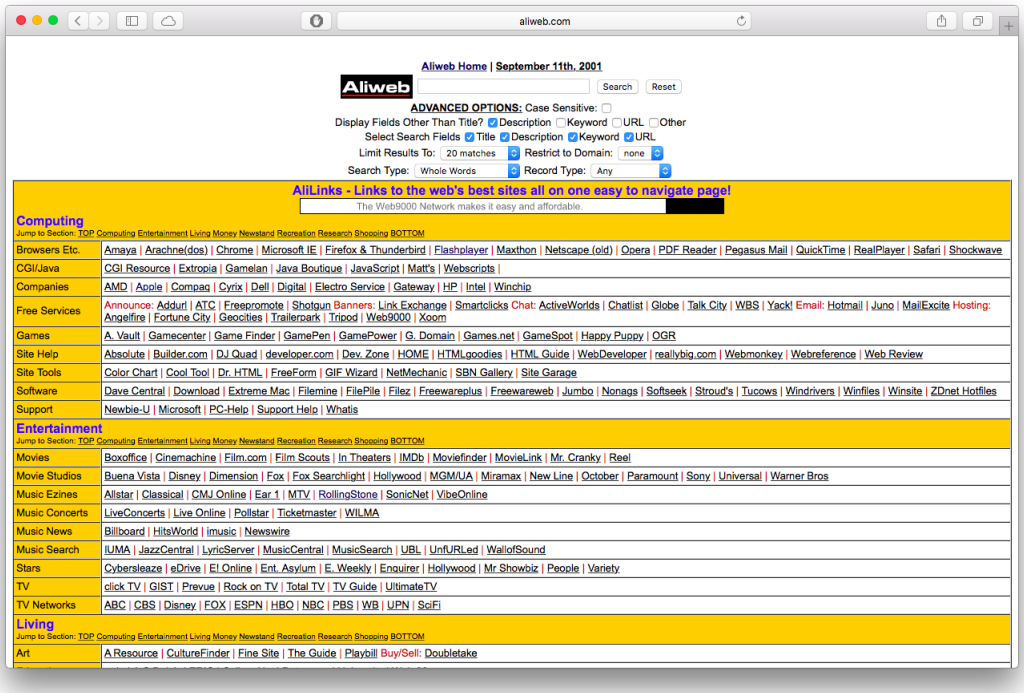
[12]Više informacija može se naći na sajtu: `https://agilitycms.com/`

[13]Joomla - `https://www.joomla.org/`

[14]Drupal - `https://www.drupal.org/`

[15]Slika 1.2 je preuzeta sa sajta `http://www.aliweb.com/`

[16]Više o istoriji veb sistema za pretragu podataka: `https://carlhendy.com/history-of-search-engines/#archie`



Slika 1.2: AliWeb

stranici, tada je ona relevantnija od drugih, pa je zato treba postaviti na vrh prikazanih rezultata pretrage. Od 1990. godine do danas nastalo je mnogo različitih sistema, ali njihova struktura i način rada su slični [13, 16].

Polazna tačka je *veb indekser* (ili pauk) za preuzimanje svih veb stranica. On obilazi čitav veb ili određeni njegov podskup da bi preuzeo stranice i sačuvao za upotrebu drugih komponenti pretraživača.

Zatim *parser* uzima sve preuzete sirove rezultate, analizira ih i na kraju pokušava da ih skladišti u određenu strukturu i izvuče relevantne informacije iz njih. U slučaju pretraživača teksta, to se radi izdvajanjem ključnih reči i proverom njihovih frekvencija. Osim samog prebrojavanja pojavljivanja ključa, koriste se i dodatni sistemi bodovanja kako bi dobili pravu ocenu relevantnosti samog rezultata.

Pretraživač mora da ima način da utvrdi koje stranice su važnije od ostalih kako bi ih predstavio korisnicima odgovarajućim redosledom. Ovo se naziva *sistem rangiranja*. Ovaj sistem koristi informacije prikupljene u ranijoj fazi od strane parsera (poput frekvencija pojavljivanja ključnih reči i drugih relevantnih informacija). Najpoznatiji algoritam za rangiranje je *PageRank* koji je objavljen od strane kompanije

Google[17].

Pouzdan *sistem za čuvanje* je od velikog značaja za svaku aplikaciju. Potrebno je da se sve uskladišti na najefikasniji način kako bi se osigurale dobre performanse. Izbor baze podataka i dizajn šeme mogu dovesti do velikih razlika u performansama. Najizazovniji deo je ogroman obim preuzetih datoteka koje treba sačuvati pre nego što ih preuzmu drugi moduli.

Najzad, sam *interfejs* preko koga korisnici vrše pretragu treba da bude intuitivan i lak za korišćenje. Kada korisnik upiše reč za pretragu, očekivano je da se rezultati ispišu u formi lakoj za čitanje, obično u vidu proste liste sa rezultatima sortiranim po relevantnosti.

U današnje vreme ljudi su navikli na pretraživače kao što je Google, gde možemo i sa greškama pri kucanju da dobijemo šta smo inicijalno tražili i gde brzo dobijamo relevantne rezultate. Korisnici očekuju slične performanse i na većini drugih današnjih sajtova. Jedna od tehnologija koja omogućava brzu i efikasnu implementaciju veb pretraživača u okviru nekog veb sajta je Elasticsearch koji je opisan u ovom radu. Elasticsearch pruža mogućnost da se nad podacima lako implementira efikasna pretraga, kakvu imaju veb pretraživači.

Na priloženoj slici 1.3[18] se vidi da je Elasticsearch još od 2016. najpopularniji sistem za pretragu podataka. Parametri koji su korišćeni za merenje popularnosti sistema su broj pominjanja sistema na vebu, učestalost pretrage ovog pojma u veb pretraživačima kao što je Google ili Bing, broj ponuda za posao koje pominju određeni sistem itd.

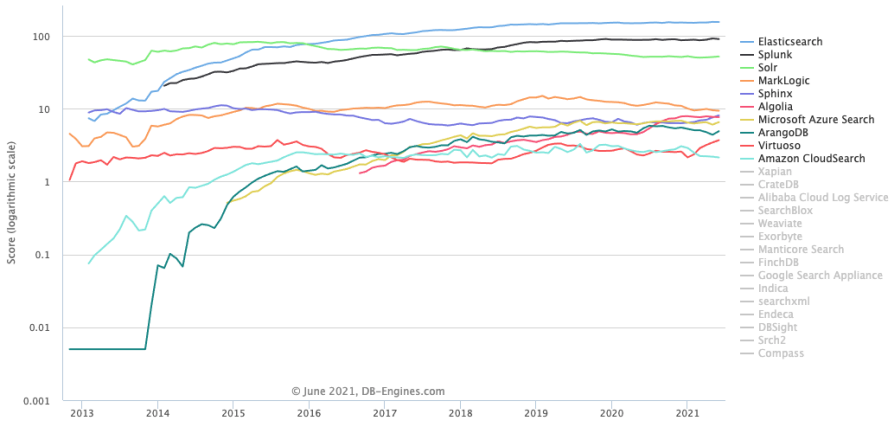
1.4 Organizacija teze

Ostatak ove teze je organizovan na sledeći način. U drugoj glavi ovog rada je detaljno razmotren sistem za upravljanje sadržajem BloomReach Experience Manager, objašnjeno je na kojim konceptima počiva i za šta se sve može koristiti. Prikazane su i njegove prednosti nad drugim sistemima.

U trećoj glavi je opisan sistem za pretragu podataka i njihovo skladištenje Elasticsearch. Prikazani su osnovni koncepti na kojima je zasnovan, kako se podešava i na šta treba obratiti pažnju pri skladištenju podataka. Pomenuti su i dodatni proizvodi koji su usko vezani sa servisom Elasticsearch, kao što su Logstash i Kibana.

[17]Više informacija o PageRank algoritmu se može naći na sajtu: `https://towardsdatascience.com/pagerank-algorithm-fully-explained-dc794184b4af`

[18]Slika je preuzeta sa sajta `https://db-engines.com/`



Slika 1.3: Elasticsearch popularnost

U četvrtoj glavi je prikazana veb aplikaciju koja je razvijena tokom rada na ovoj tezi, a u cilju demonstracije navedenih tehnologija. Prikazano je na koji način se može odvijati komunikacija između BloomReach Experience Manager sistema i Elasticsearch servisa. S obzirom da je kao implementacioni jezik korišćen Kotlin, u ovoj glavi je dat i kratak uvod u Kotlin jezik. Takođe, date su i ideje za dalji razvoj aplikacije i integraciju dodatnih funkcionalnosti.

Glava 2

Bloomreach

U ovom poglavlju je razmatran sistem za upravljanje sadržajem Bloomreach Experience Manager (BrXM) koji je deo platforme za digitalno iskustvo Bloomreach Experience (BrX) i proizvod je kompanije Bloomreach. Najpre je prikazan kratak istorijat razvoja kompanije Bloomreach, kao i samog sistema BrXM. Nakon toga je prikazana arhitekturu i koncepte na kojima je ovaj sistem zasnovan. Opisan je način korišćenja jednog BrXM sistema, kako se vrši pokretanje ovog sistema, na kojim tehnologijama se zasniva, a prikazan je i primere dodavanja novih funkcionalnosti.

2.1 Istorijat

Kompanija Bloomreach nastala je 2009. godine, a osnovana je od strane Raž De Date, preduzetnika iz Silikonska doline i Ašutoš Garga, istraživača iz kompaniji Google. Ideja na kojoj je kompanija Bloomreach zasnovana je da svako online iskustvo može biti drastično poboljšano ako bi bilo pravljeno za svakog korisnika individualno [2]. Neke od bitnih godina za kompaniju Bloomreach, pored godine osnivanja (2009. godine) su:

- 2012. godine - predstavljanje javnosti prvog proizvoda Bloomreach kompanije *Intelligent Search*, koji je deo njihovog *Bloomreach Discovery*[1] proizvoda, koristi se za unapređivanje pretrage na sajtu korisnika.
- 2014. godine - lansirana su dva nova proizvoda u oblasti online trgovine, koji su takođe deo *Bloomreach Discovery* proizvoda i zovu se *Strategic Merchandising*

[1]Više informacija o Bloomreach Discovery proizvodu: `https://www.bloomreach.com/en/products/discovery`

i *Automated SEO*.

- 2016. godine - sklopljen je ugovor sa kompanijom Hippo, čiji je proizvod sistem za upravljanje veb sadržajem Hippo CMS.
- 2018. godine - Bloomreach je prepoznat kao vizionar od strane kompanije Gartner[2] u polju digitalnih internet platformi.
- 2021. godine - kupuje kompaniju Exponea[3] koja se bavi automatizacijom marketinga.

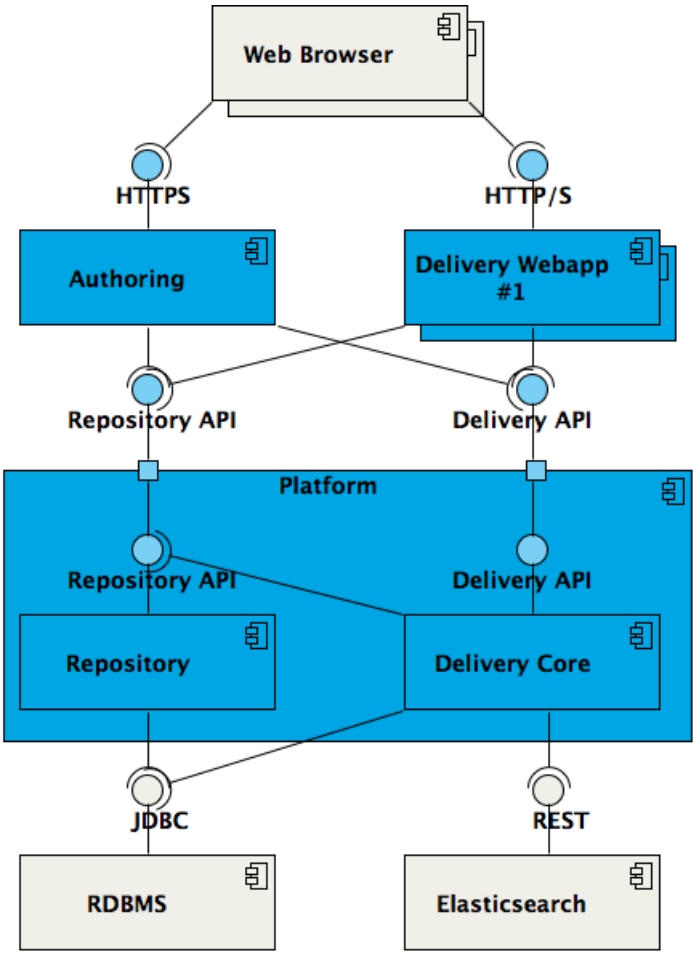
Sam sistem BrXM zapravo nije nastao u kompaniji Bloomreach. Pre nego što je kopanija Bloomreach otkupila ovaj sistem za upravljanje sadržajem, sistem se zvao Hippo CMS, a originalno je razvijen od strane kompanije Hippo 1999. godine. Hippo CMS je imao dve verzije: verziju otvorenog koda, koja je besplatna i verziju za kompanije, za koju klijenti plaćaju naknadu na godišnjem nivou. Osnovni deo Hippo CMS-a je bio isti u obe verzije, samo je verzija za kompanije sadržala više integrisanih proizvoda (npr. alat za analizu pristupa sajtu). Izdanje otvorenog koda je bilo pod licencom ASL 2.0[4].

Hippo CMS je napisan u Java programskom jeziku. Svi delovi Hippo CMS sistema poseduju interfejs koji omogućava lakšu integraciju sa drugim aplikacijama. Takođe, sistema sadrži i veoma intuitivan i moćan korisnički interfejs za upravljanje sadržajem sajtova. Sve ove karakteristike koje je imao Hippo CMS se takođe prenose i na BrXM. Kompanija Bloomreach je nastavila da održava sistem i da ga unapređuje. Jedno od najznačajnijih unapređenja je transformacija BrXM sistema u bezglavi sistem za upravljanje sadržajem. Bloomreach kompanija ga smatra samostalnim proizvodom, kao i delom digitalne platforme BrX.

2.2 Arhitektura BrXM

BrXM je zasnovan na modernim principima za dizajn arhitekture softvera, kao što su razdvajanje odgovornosti, jedinstvena odgovornost, princip najmanjeg znanja, itd. BrXM se oslanja na mnoštvo softvera otvorenog koda i na otvorene standarde. Takođe, kako bi se lako integrisale druge aplikacije, na svim nivoima arhitekture

[2]Gartner - `https://www.gartner.com/en`
[3]Exponea - `https://exponea.com/`
[4]Više informacija se može naći na sajtu: `https://www.apache.org/licenses/LICENSE-2.0`



Slika 2.1: BrXM arhitektura

dostupan je odgovarajući API. BrXM je takođe i veoma modularan sistem, tj. veliki deo koda podeljen je u manje celine koje imaju određenu funkcionalnost. Ovakva organizacija koda obezbeđuje brojne prednosti sa tehničke strane, ona čini ovu aplikaciju skalabilnom, lakšom za održavanje i daje mogućnost ponovnog korišćenja delova koda [5].

Na slici 2.1[5] se vidi arhitektura BrXM sistema. Kada ovu arhitekturu posmatramo na visokom niovu, može se uočiti podelu na tri celine. Te celine su platforma (engl. *The platform*), aplikaciju za uređivanje (engl. *The authoring webapp*) i veb aplikacije za isporuku (engl. *The delivery webapp(s)*). U nastavku rada je detaljnije prikazan svaki od ovih delova sistema.

[5]Slika je preuzeta sa sajta `https://documentation.bloomreach.com/14`

Platforma

Ovaj deo BrXM sistema podeljen je na dva dela – repozitorijum i jezgro aplikacije. *Repozitorijum* (engl. *repository*) je zadužen za skladištenje metapodataka, podešavanja i samog sadržaja. Zasnovan je na repozitorijumu sadržaja (engl. *content repository*) otvorenog koda namenjenog Java jeziku, pod nazivom *Apache JackRabbit*[6]. Apache JackRabbit implementira JSR-170[7] i JSR-283[8] specifikacije Java programskog interfejsa za pristup repozitorijumu (engl. *Java Content Repository - JCR*), a oslanja se na *Apache Lucene* tehnologiju za indeksiranje i pretragu. Ove dve specifikacije govore kako repozitorijum sadržaja treba da radi i kako treba da izgleda kôd kako bi bio u skladu sa njima. Repozitorijum je, u suštini, pokrenuta instanca Apache JackRabbit sistema i predstavlja ceo sadržaj. Apache JackRabbit nudi mogućnost postojanja više radnih prostora (engl. *workspace*), koji služe za podelu sadržaja. Repozitorijum koristi jedan radni prostor koji se dalje sastoji iz čvorova (engl. *nodes*) i osobina (engl. *properties*), koji u BrXM sistemu predstavljaju dokumenta i vrednosti njihovih polja. Iako nije baza u klasičnom smislu, repozitorijum se može smatrati jednom vrstom baze podataka.

Kompletan sadržaj je sada dostupan putem interfejsa za pristup prethodno opisanoj strukturi. Ovo omogućava laku integraciju BrXM sistema sa drugim aplikacijama i da se lako manipuliše sadržajem (pravljenje kopija iz sigurnosnih razloga, kreiranje sadržaj, da se uveze sadržaj iz drugih izvora itd.). Postoji i veb aplikacija koja predstavlja prost intuitivan korisnički interfejs za pristup svim JCR čvorovima aplikacije (slika 2.2). Pretraga ovih čvorova se može vršiti na razne načine, a preporučuje se korišćenje XPath[9] tehnologije.

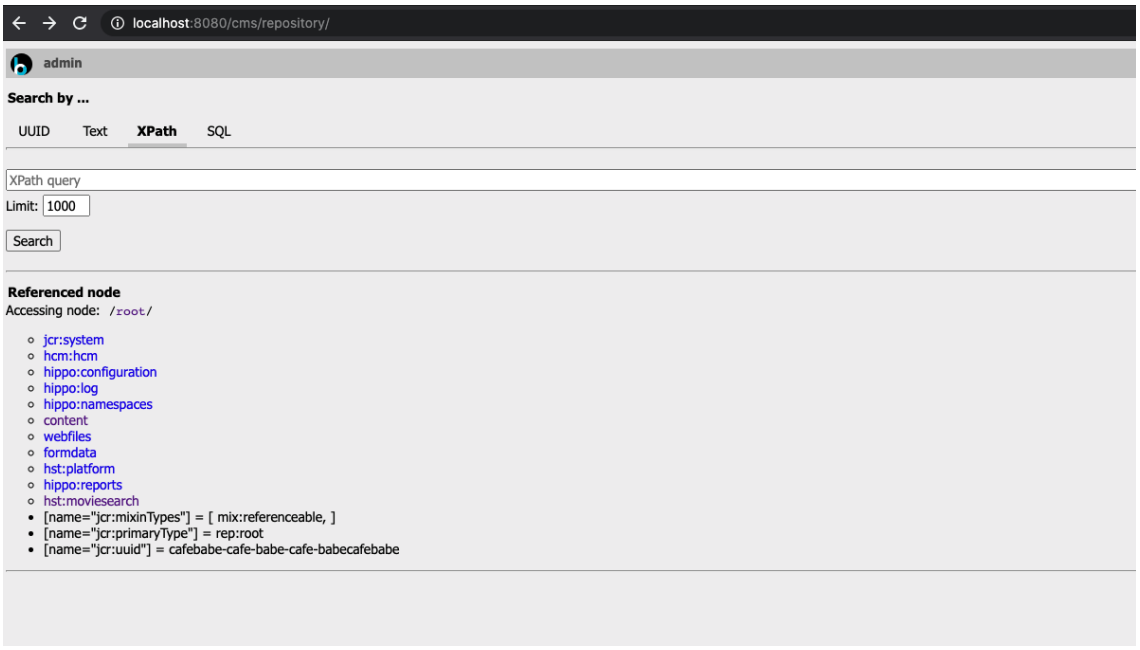
Drugi deo platforme je *jezgro za isporuku* (engl. *delivery core*) veb aplikacije, tj. podešavanja aplikacije. U njoj se nalaze podešavanja za sve kanale, ta podešavanja govore kako se preslikavaju dolazeći zahtevi sa odgovarajućom stranicom, kao i sadržajem i komponentama koje ta stranica ima. Kasnije u radu je detaljno objašnjeno kako se može podesiti nova komponenta u BrXm sistemu, kao i kako se za nju pišu podešavanja i način povezivanja sa nekom stranicom. BrXM daje još jedan intuitivan korisnički interfejs preko kog se mogu lako praviti izmene i na ovom delu

[6] `https://jackrabbit.apache.org/jcr/getting-started-with-apache-jackrabbit.html`

[7] Više na sajtu: `https://jcp.org/aboutJava/communityprocess/final/jsr170/index.html`

[8] JSR-283 je samo novija verzija istih specifikacija.

[9] XPath - `https://www.w3schools.com/xml/xpath_intro.asp`



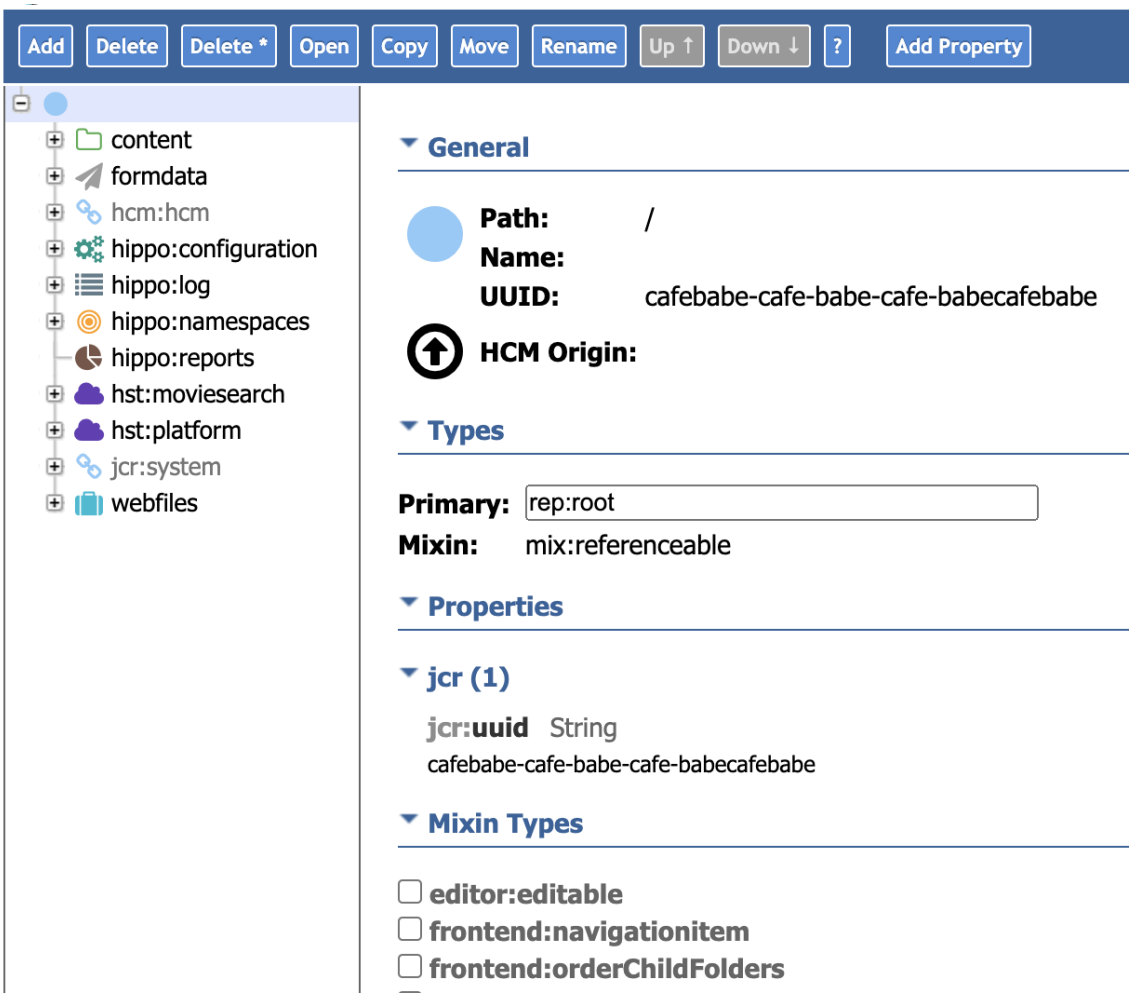
Slika 2.2: Repozitorijum

sistema (slika 2.3).

Autorski interfejs

Ovaj sloj arhitekture se odnosi na korisnički interfejs preko kog timovi koji održavaju sadržaj mogu da sarađuju (slika 2.4). Urednici sajta preko njega mogu da prave podsajtove, menjaju stranice, menjaju strukturu stranica, kao i da menjaju meta tagove samih stranica. Prava pristupa se mogu ograničiti za pojedine autore ili grupe autora. U novijim verzijama ovog sistema, kako ne bi došlo do oštećenja podataka, samo jedna osoba može da radi na nekom dokumentu i da menja njegov sadržaj, pri čemu je taj dokument za to vreme zaključan za sve ostale autore.

Autorski interfejs takođe ima mogućnost integracije eksternih aplikacija i može se izmeniti na nivou Java koda preko *GUI* dodatka čija je arhitehtura zasnovana na *Apache Wicket* [4] veb okviru (engl. *web framework*). Wicket je okvir orijentisan ka veb komponentama na Java server strani koji ima za cilj da pojednostavi izgradnju veb interfejsa uvođenjem poznatih šablona za razvoj korisničkog interfejsa. Korišćenjem Wicket okvira je moguće napraviti veb aplikaciju koristeći samo Java kôd i HTML stranice kompatibilne sa XHTML-om. Ovim se gubi potreba za upotrebom Javascript jezika. Wicket okvir štiti programere od rada na niskom nivou i



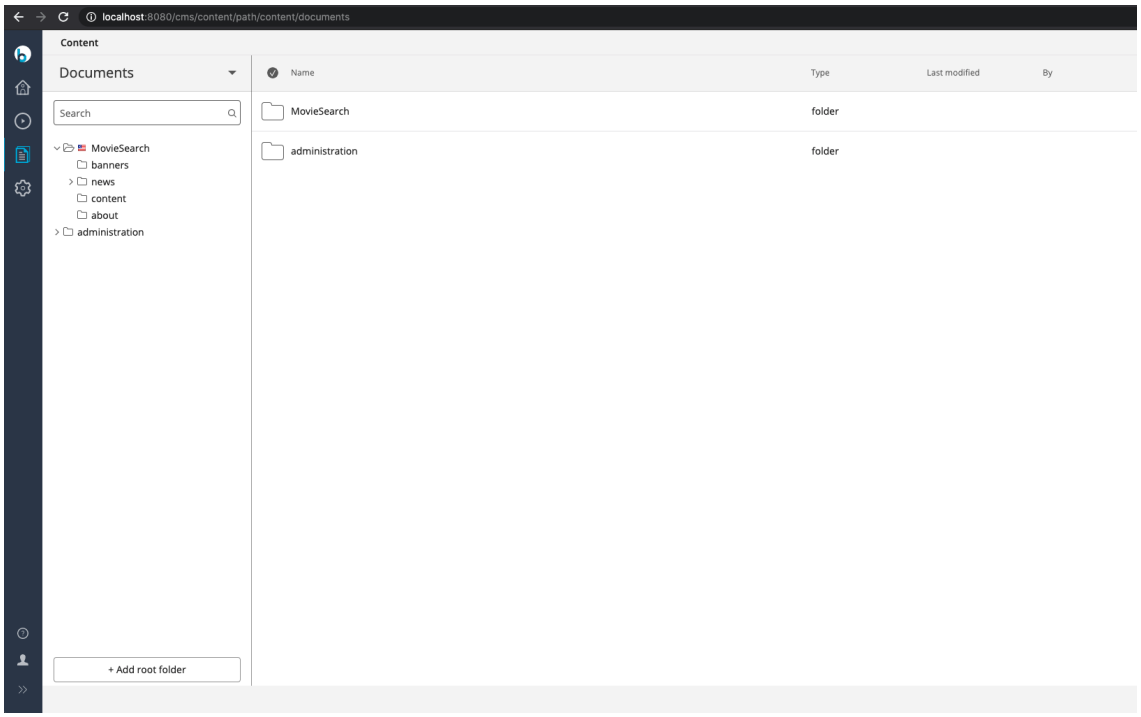
Slika 2.3: Konzola za podešavanja

omogućava programerima da se usresrede na poslovnu logiku.

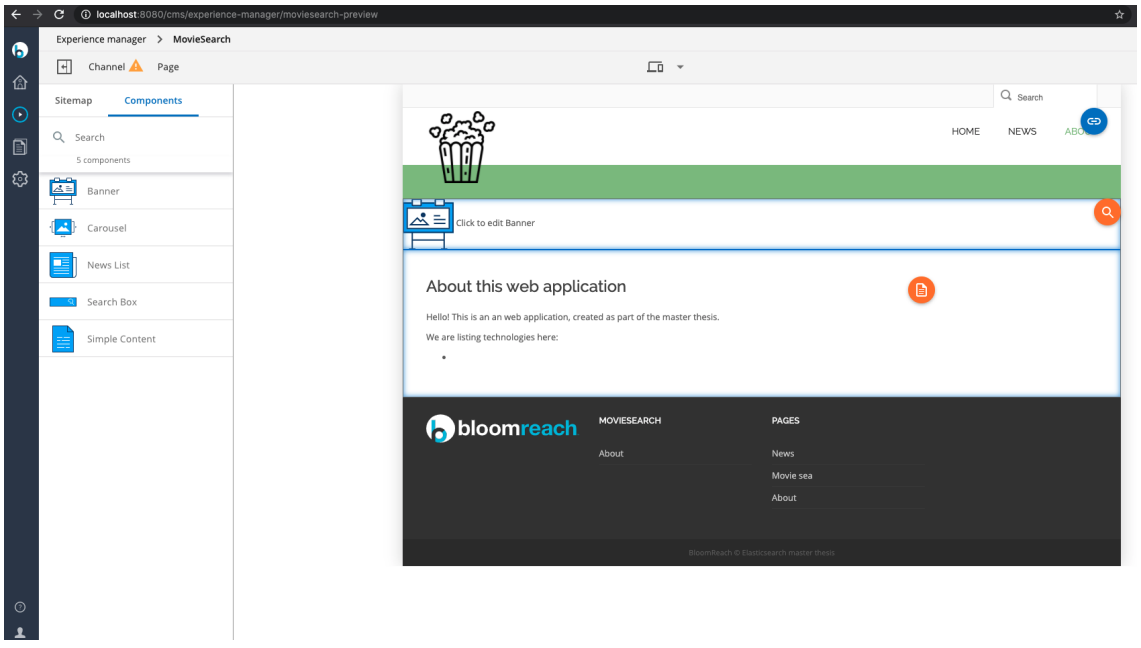
Na slici 2.5 se vidi da se u autorskoj aplikaciji nalazi deo koji nosi naziv koji je sadržan i u nazivu samog sistema za upravljanje sadržajem *Experience Manager*. Kad se otvori taj deo aplikacije i odabere kanal koji treba da se modifikuje, otvoriće se početna strana samog sajta. Nakon toga, postoji mogućnost jednostavnog dodavanja komponenti i izmene sadražaja sa trenutnim prikazom izmena.

Veb aplikacije za isporuku

Veb aplikacije za isporuku predstavljaju sam korisnički interfejs koji vide krajnji korisnici. Ove aplikacije koriste podešavanja sajta da preslikaju dolazeće zahteve ka stranicama i da prikažu odgovarajući sadržaj. Programeri, kao što se može videti



Slika 2.4: Veb aplikacija autorskog interfejsa



Slika 2.5: Experience Manager deo

kasnije, mogu lako da razviju nove funkcionalnosti dodavanjem novih podešavanja i poslovne logike. Zahvaljujući razdvojenosti šablona i sadržaja, veb aplikacija za

isporuku se može napraviti i van samog sistema. U slučaju implementacije u kojoj je aplikacija za isporuku odvojena, sistem zadržava mogućnost lakog pristupa sadržaju, kao i održavanju sadržaja preko autorskog interfejsa.

Bezglavi koncept

Opisana arhitektura omogućava jednostavnu implementaciju koncepta bezglavog[10] sistema. Naime, ovakvi sistemi za upravljenje sadržajem se mogu posmatrati i kao pozadinski sistemi koji sadrže repozitorijum sa podacima kojima se može pomoću *RESTful API-ja*[11] pristupiti. Prednost ovakvih sistema je što omogućavaju da se lako nad istim podacima grade razne aplikacije, tj. nad istom pozadinskom aplikacijom moguće je napraviti mobilnu aplikaciju ili veb aplikaciju. Ipak, u cilju bržeg razvoja, u ovom radu će biti korišćeni *Freemarker šabloni*, umesto da se razvija potpuno odvojena korisnička aplikacija. Detaljniji prikaz Freemarker [3] šablona biće dat kasnije, na primeru izrade jedne komponente.

Na slici 2.6[12] se vidi razlika između tradicionalnog i bezglavog koncepta.

Osobine BrXM arhitekture

Prethodno je prikazano kako izgleda ahitektura BrXM sistema na visokom nivou. U nastavku su navedene neke od osobina koje ovaj sistem ima i zašto su one korisne.

Jednostavnost integracije

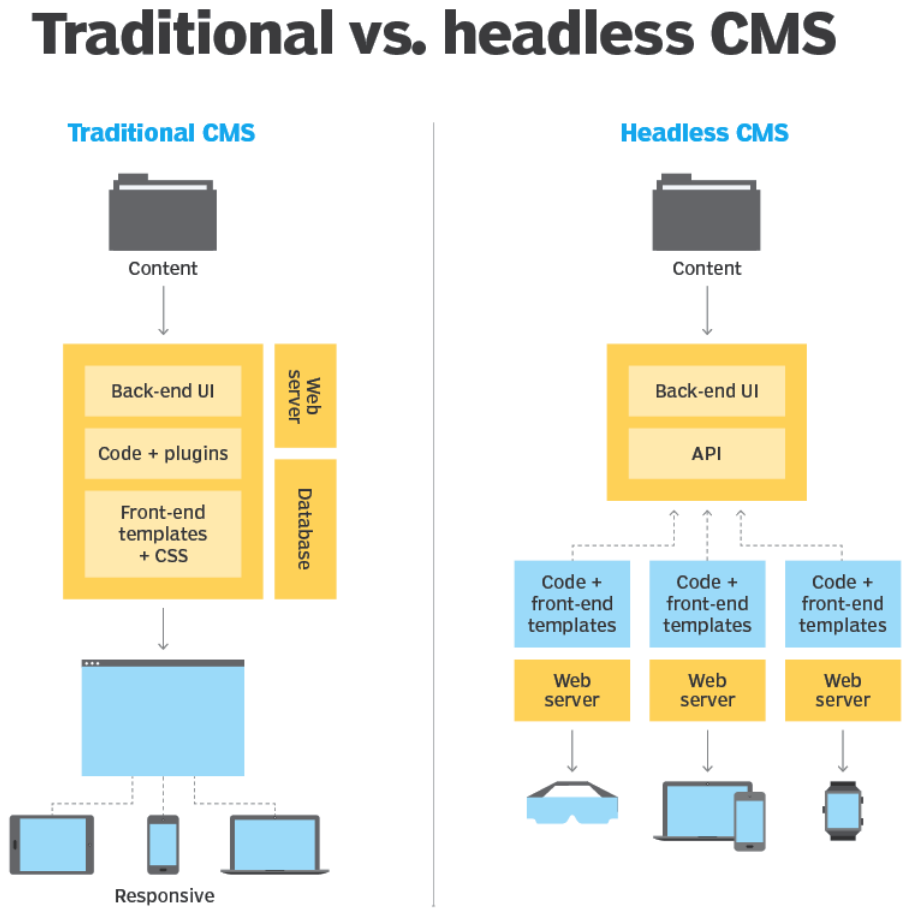
BrXM je dizajniran sa namerom da bude deo nekog većeg rešenja, zato i nije čudno da postoje tačke integracije na svim nivoima, kako bi ovaj sistem mogao da se uklopi u već postojeće aplikacije. S obzirom da su podaci veoma bitan resurs svake kompanije, BrXM nudi mogućnost pristupa sadržaju na više načina, poput podrške za JAX-RS (Java API for RESTful Services)[13], podrške za slanje događaja drugim aplikacijama (npr. mejl obaveštenja da je neko objavio novi dokument na sajtu), podrške za pristup na niskom nivou sadržaju preko JCR 2.0 API-ja, itd.

[10] Naziv „bezglavi" potiče otuda što je korisnički interfejs posmatran kao glava, a pozadinski deo kao telo aplikacije.

[11] `https://www.techtarget.com/searchapparchitecture/definition/RESTful-API`

[12] Slika je preuzeta sa sajta `https://searchcontentmanagement.techtarget.com/feature/Headless-CMS-powers-personalized-omnichannel-e-commerce`

[13] Više informacija na lokaciji: `https://documentation.bloomreach.com/14/library/concepts/rest/restful-jax-rs-component-support-in-hst-2.html`



Slika 2.6: Razlika između tradicionalnog i bezglavog CMS-a

U nastavku je naveden spisak svih dostupnih programskih interfejsa za Bloomreach Experience Manager 14.5[14]:

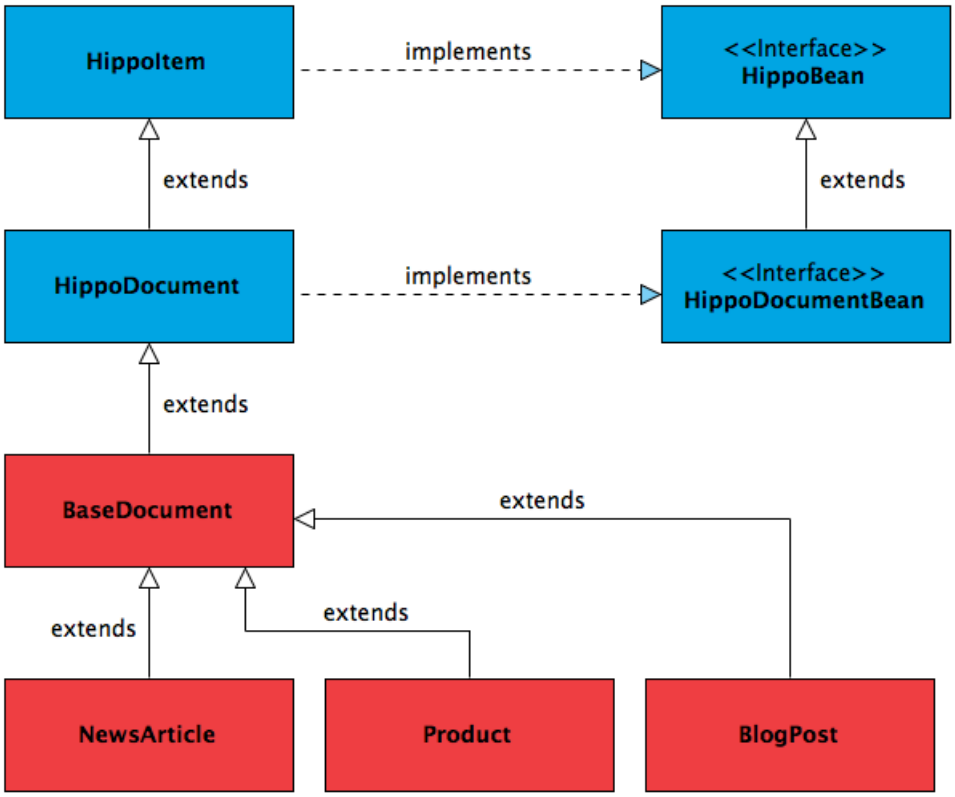
- CMS API 14.5
- Commons API 14.5
- Repository API 14.5
- Services API 14.5
- Site Toolkit (HST) API 14.5
- CRISP API 14.5

[14]Više informacija na lokaciji: `https://documentation.bloomreach.com/14/library/about/javadoc.html`

- JSP/Freemarker Tag Library : HST Tag Library 14.5
- JavaScript TypeDoc : OpenUI Extension Library 14.5
- JavaScript TypeDoc : Commerce React Components 14.5

Proširivost

BrXM ne nudi odmah rešenje za sve moguće potrebe jedne veb aplikacije, ali zato omogućava da se postojeće komponente lako prošire, kao i da se lako dodaju nove komponente u sistem. BrXM je baziran na MVC šablonu. Na slici 2.7 se vidi kako izgleda jedan primer nasleđivanja uz korišćenje *HippoBean* interfejsa.



Slika 2.7: Proširivost

Skalabilnost

U današnje vreme, veoma je bitno da se sistem može prilagoditi broju poseta sajta. U sistemu BrXM postoji opcija i horizontalne i vertikalne skalabilnosti. Svaki od prethodno navedenih delova arhitekture moze da se skalira.

Vertikalna skalabilnost je kada je aplikacija smeštena na jednom serveru, a na povećan protok se reaguje tako što se serveru dodaje memorija, jači procesor, nova jezgra ili dodatni hard disk.

Horizontalna skalabilnost (smanjenje) povezuje više stavki kako bi radile kao jedna logička jedinica. Dodavanjem novih servera sistem nastavlja da radi kao do sada, samo sa novim serverom u timu.

Dobre performanse

Perofrmanse sistema se mogu značajno poboljšati upotrebom keširanja. U tu svrhu, BrXM ima nekoliko različitih sistema za keširanje. Jedan od njih je *Bundle-Cache* koji se odnosi na sam sadržaj. Repozitorijum kešira sadržaj u određeni tip dokumenta, koje nazivamo snop (engl. *bundle*). Ovakvim keširanjem sadržaja smanjuje se potreba za pristupom bazi pri svakom čitanju sadržaja i time se značajno poboljšavaju preformanse.

Veb aplikacija za isporuku ima četiri različita keša: *binarni keš* (engl. *Binaries Cache* – koristi se za statičke podatke, PDF dokumenta ili slike), *keš veb dokumenata* (engl. *WebFiles Cache* – statički sadržaj u vidu veb dokumenata), *keš čvora* (engl. *Node Cache* – čuva određene JCR čvorove) i *keš stranice* (engl. *Page Cache* – mogućnost keširanja celih stranica). Na slici 2.8[15] se vidi diagram sa pomenutim vrstama keša.

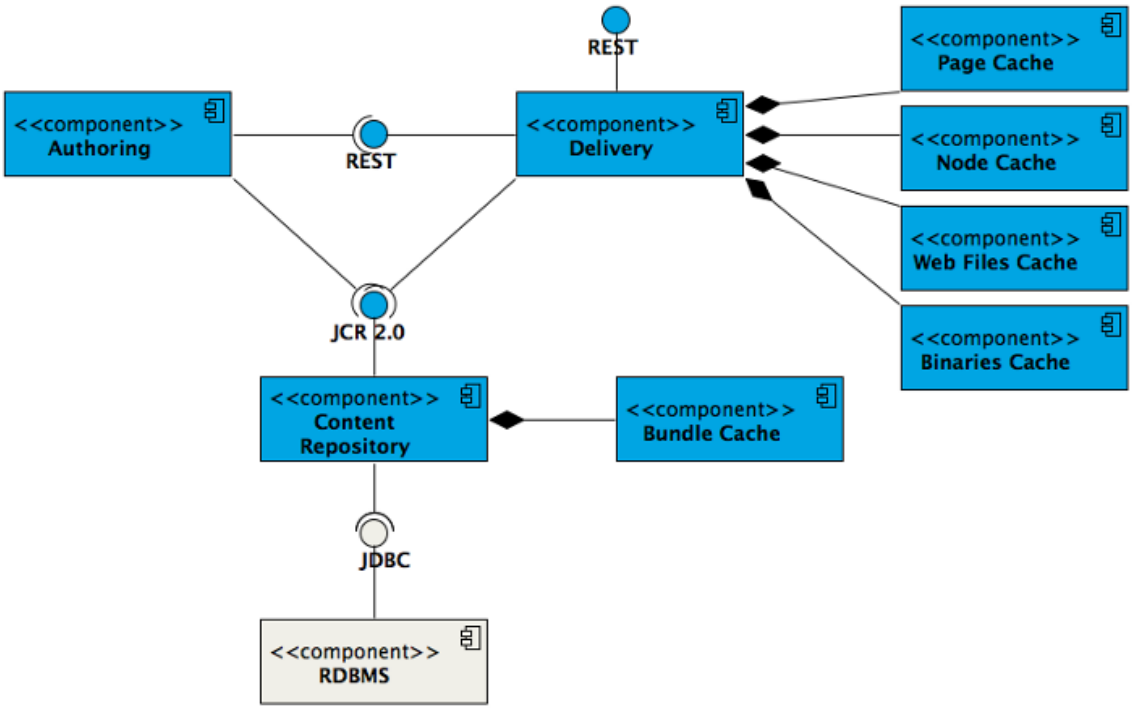
Sigurnost

BrXM kontinualno prati i održava sistem zaštićenim od strane svih poznatih napada. U samoj dokumentaciji se može pronaći i spisak napada od kojih je sistem zaštićen, kao i kako da napravite aplikaciju sa najvećim nivoom zaštite. U samoj autorskoj aplikaciji postoji sistemski zapis (engl. *log*) koji dopušta da se vidi svaka izmena u sistemu, ko je napravio izmene, kao i da se naprave različite grupe u sistemu za različitim dozvolama. Ovim se smanjuje mogućnost da autori naprave greške. Takođe, ovaj sistem pruža više načina za rukovanje nalozima autora. Mogu se koristiti LDAP[16], dvostepenu autentifikaciju, SSO[17] itd.

[15] Slika je preuzeta sa `https://xmdocumentation.bloomreach.com/library/about/why-brxm/performance.html` stranice

[16] `https://ldap.com/`

[17] `https://www.techtarget.com/searchsecurity/definition/single-sign-on`



Slika 2.8: Keširanje u BrXM sistemu

2.3 Koncepti BrXM sistema

U nastavku su prikazani koncepti na kojima je BrXM izgrađen, pri čemu je akcenat na onim konceptima koji su korišćeni u ovom radu.

Tip dokumenta. Tip dokumenta predstavlja strukturu podataka, šablon za uređivanje nekog dokumenta od strane urednika. On se gradi od polja koja mogu biti prosta (primitivna) i složena (druge kompnente).

Složeni tip. Složeni tip definiše strukturu podataka i obrazac za uređivanje više polja za višekratnu upotrebu. Složeni tip može sadržati i primitivna i složena polja.

Tip skupa slika. Skup slika je posebna vrsta dokumenta koji se koristi za čuvanje različitih veličina slika.

Prostor imena. Prostor imena grupiše više tipova sadržaja kako bi se sprečila kolizija imena u slučaju dupliranih imena tipova sadržaja.

Prototip. Prototip je obrazac iz kojeg se kreira bilo koja nova instanca tipa sadržaja.

Čvor. Čvorovi čine strukturu baze sadržaja. Čvor može imati nula ili više podčvorova.

Osobina. Osobina je deo sadržaja uskladištenog u čvoru. Osobina ima primitivni tip kao što je *String* ili *Double*.

Varijanta dokumenta. Varijanta dokumenta je čvor koji predstavlja dokument u određenom stanju toka posla, poput nacrta, neobjavljenog ili objavljenog. Istovremeno mogu postojati različite varijante dokumenata.

Podešavanje isporuke. Podešavanje isporuke je skup stavki podešavanja potrebnih da se po zahtevu određene stranice ona i isporuči korisniku. Stavke podešavanja uključuju: mapu sajta, modele stranica, komponente itd.

Sajt. Sajt povezuje podešavanja za isporuku sa korenim sadržajem u bazi. Moguće je imati više veb sajtova koji mogu da koriste ista podešavanja sa različitim korenima sadržaja. Na primer, u slučaju prevedenih veb sajtova, ovakve sajtove nazivamo i kanalima.

Model stranice. Model stranice je struktura komponenata u sistemu.

Komponenta. Komponenta je jedna jedinica u hijerarhiji koja čini model stranice. Tipično ima jednu svrhu, kao što je preuzimanje jedne ili više stavki sadržaja, izvršavanje upita za pretragu ili delegiranje na podređene komponente u hijerarhiji stranice. Po želji, komponenta može imati šablon za prikazivanje zasnovan na Javi na serveru.

Katalog. Katalog sadrži komponente koje urednik može koristiti.

Šablon. Šablon se koristi za prikazivanje komponente zasnovane na Javi na strani servera, obično koristeći Freemarker. Šabloni nisu potrebni kada se koristi API modela stranice, tj. u slučaju kada se koristi sistem kao bezglavi, tada je sam izgled stranica je definisan u drugoj aplikaciji.

Mapa sajta. Mapa sajta definiše URL prostor veb sajta kao hijerarhiju stavki. Ona se sastoji iz stavki mape sajta koje vezuju URL adrese za model stranice.

2.4 Pokretanje i struktura

BrXM je *Maven*[18] projekat. Maven je alat koji se koristi za izgradnju i upravljanje projektima koji su zasnovani na Java programskom jeziku. Maven koristi POM[19] (engl. *Project Object Model*) datoteke prilikom izgradnje projekta. Ove datoteke su u XML formatu i sadrže informacije o projektu, kao i detalje podešavanja potrebne da se projekat izgradi. Jedna od bitnijih karakteristika alata Maven je to da poseduje mehanizam pomoću koga se automatski preuzima bilo koji JAR[20] (engl. *Java ARchive*) koji je potreban u izgradnji projekta. Maven prolazi kroz POM datoteku projekta i kada pronađe zavisnost od nekog projekta on prvo pretraži lokalni repozitorijum – direktorijum na samom računaru i proveri da li tražena JAR datoteka postoji u njemu. Ukoliko ne postoji, nastavlja potragu u globalnom repozitorijumu (npr. `search.maven.org`) i preuzima traženu JAR datoteku. Kako bi pokrenuli BrXM sistem potrebno je da imamo na uređaju instalirane *Maven 3.3.9+* i *Java 8+*.

BrXM dobijen prilikom preuzimanja ima pet podprojekata, a to su:

- *cms* - koji kreira BrXM veb aplikaciju, tj. samu platformu sa autorskim interfejsom.
- *cms-dependencies* - mesto sa projektima od kojih zavisi ovaj sistem, definisanih u POM-u.
- *repository-data* - koji pravi četiri JAR datoteke koje sadrže početni sadržaj reozitorijuma.
- *site* - pravi JAR datoteku sa komponentama veb aplikacije i WAR (engl. Web application ARchive)[21] koji sadrži BrXM veb aplikaciju za isporuku.
- *essentials* - kreira veb aplikaciju koja ima već gotove komponente koje nudi Bloomreach, a koje se mogu dodati u projekat.

[18] `https://maven.apache.org/`

[19] `https://maven.apache.org/guides/introduction/introduction-to-the-pom.html`

[20] `https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html`

[21] WAR je format datoteke koji sadrži veb aplikaciju spremnu za dalju distribuciju. Više se može pročitati na: `https://docs.oracle.com/javaee/6/tutorial/doc/bnadu.html`

Komanda sa kojom se gradi BrXM projekat je:

```
mvn clean verify
```

Kada se uspešno izgradi projekat, može se preći na njegovo pokretanje. Kao što je pomenuto ranije, u ovom sistemu se nalazi više od jedne veb aplikacije, ali to ne znači da se svaka od tih veb aplikacija pokreće odvojeno. Pomoću Maven dodatka koji se zove *Cargo* [6], komanda za pokretanje projekta pokreće sve tri aplikacije odjednom, tj. instancu sistema za upravljanje sadržajem sa JCR repozitorijumom, platformu i veb aplikaciju za isporuku. Cargo pravi omotač oko Java programskog interfejsa za podešavanje, pokretanje, zaustavljanje i isporuku aplikacija. Podešavanje za Cargo se nalazi u BrXM POM datoteci projekta. Podešen je tako da koristi *Tomcat 9*[22] za pokretanje aplikacija. Pri prvom pokretanju on će preuzeti Tomcat 9 distribuciju i smestiti u `target-tomcat9x` direktorijum. Sistemski zapisi aplikacije se mogu naći na lokacijama `target/tomcat9x/logs/site.log` i `target/tomcat9x/logs/cms.log`.

Komanda za pokretanje projekta je:

```
mvn -P cargo.run
```

Po izvršavanju ove komande, BrXM je pokrenut i može se pristupiti svim veb aplikacijama. Početnim podešavanjem je aplikacija postavljenja na port 8080. Sada se može autorskom interfejsu pristupiti na adresi `http://localhost:8080/cms`.

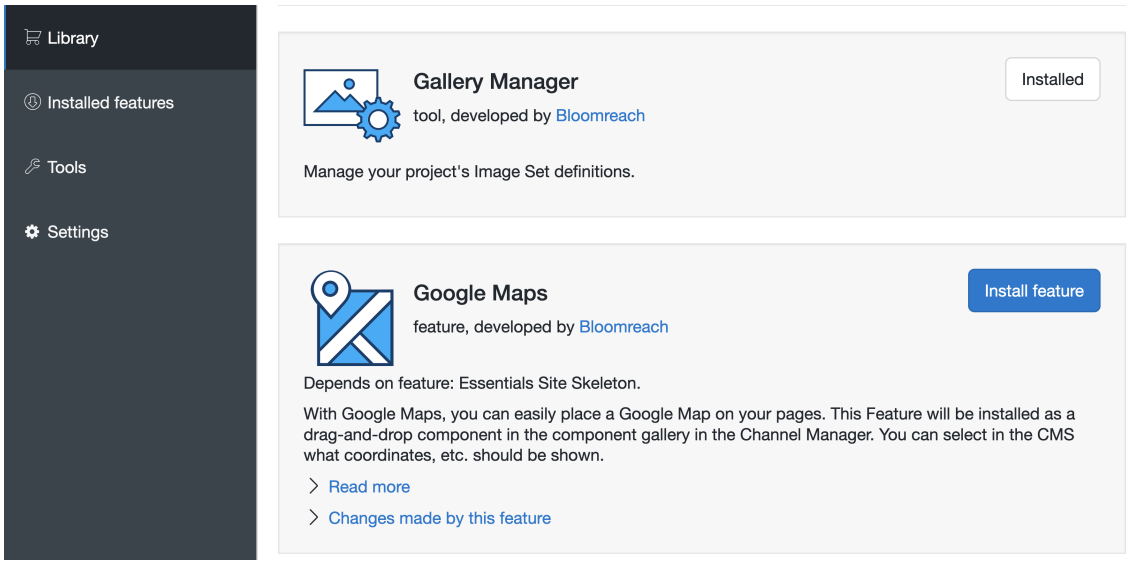
Na lokaciji `http://localhost:8080/cms/console` se pristupa korisničkom interfejsu za JCR repozitorijum koji koristi CMS. Često se ova veb aplikacija koristi od strane programera za podešavanje komponenti i sajta, kada su u pitanju oni aspekti podešavanja koji nisu dostupni preko CMS-a, a ova veb aplikacija ih čini preglednijim od direktnih izmena u kodu. Početna adresa sajta se nalazi na `http://localhost:8080/site`.

2.5 Dodavanje funkcionalnosti

Dodavanje postojećih funkcionalnosti

BrXM, ima mogućnost dodavanja već postojećih dodataka, ponuđenih od strane kompanije Bloomreach. Naravno, to je najlakši način da dodamo novu funkcionalnost u projekat. Ponuđeni dodaci mogu se videti putem aplikacije *The Essentials* koja se nalazi na adresi `http://localhost:8080/essentials/` (slika 2.9).

[22] Apache Tomcat - `http://tomcat.apache.org/`



Slika 2.9: The Essentials

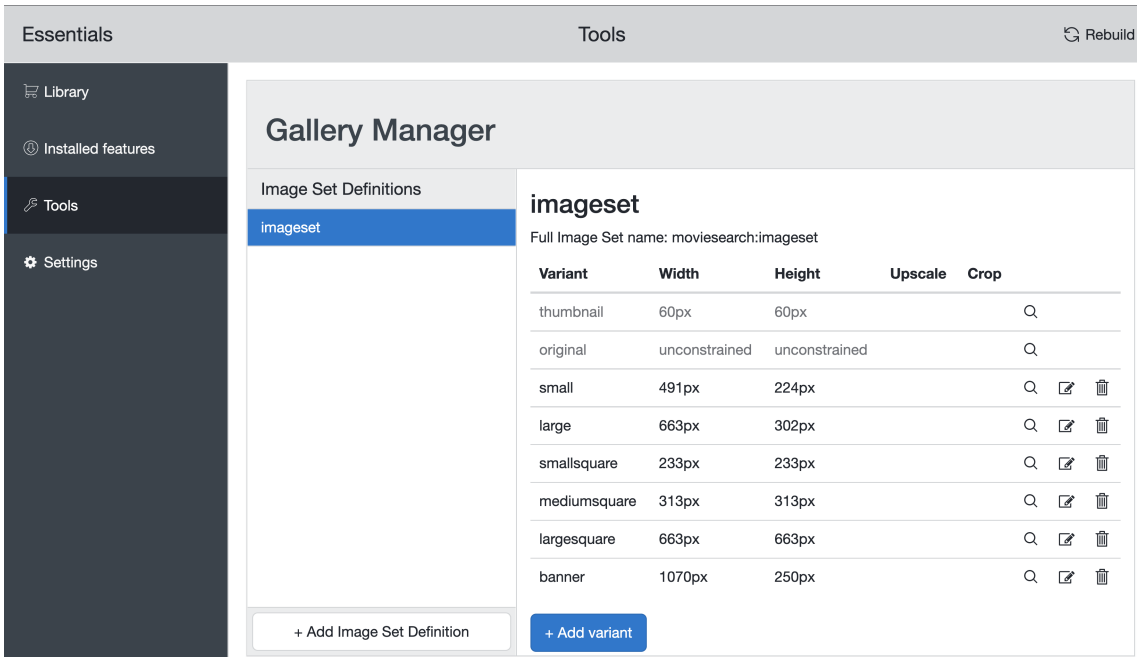
U ponudi se nalazi određeni broj alata koji mogu da pomognu programerima tako što će da ubrzaju ili čak da automatizuju neke od čestih zadataka pri razvijanju novih funkcionalnosti u sistemu.

Jedan primer je *Gallery Manager* dodatak, koji služi sa podešavanje raznih dimenzija slika. Mogu se dodati vrednosti za sve rezolucije koje će biti potrebne sajtu. Prilikom ubacivanja slika u CMS, ovaj dodatak će automatski omogućiti skaliranje na sve ove dimenzije, uz mogućnost isecanja (engl. *cropping*) ako je to potrebno. Na slici 2.10 se vidi kako ovaj dodatak izgleda.

Još jedan od veoma korisnih dodataka je *Beanwriter* koji služi da automatski napravi Java klase za prikaz modela pri pravljenju novih tipova dokumenata u sistemu. Ovaj dodatak pravi uštedu vremena programerima i smanjuje mogućnost greške.

Razvoj novih funkcionalnosti

Kao i kod svakog sistema za upravljanje sadržajem, i kod BrXM sistema je fokus na podacima koje kreiraju i unose urednici sajta. Kada je u pitanju razvoj novih funkcionalnosti u ovom sistemu, po pravilu se misli na kreiranje novih komponenti koje će koristiti urednici. BrXM je zasnovan na principima MVC arhitekture, te se pri razvoju svake komponente koristi upravo MVC obrazac. MVC (engl. *Model-view-controller*) arhitektura je projektni obrazac (engl. *design pattern*) koji omogućava

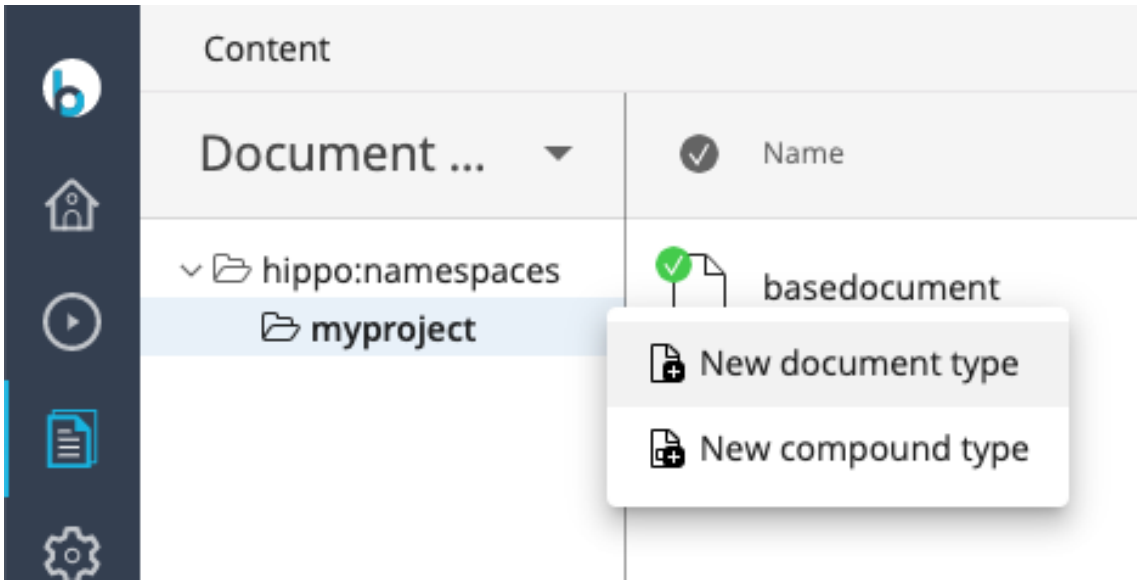


Slika 2.10: Gallery Manager

bolju podelu koda što ga čini lakšim za održavanje. Zasniva se na ideji o ponovnoj upotrebi već postojećeg softverskog koda. MVC projektni šablon se sastoji od tri zasebne, ali međusobno povezane komponente: model podataka (engl. *model*), prikaz podataka (engl. *view*) i kontroler (engl. *controller*). MVC je široko prihvaćen obrazac, koji se koristi u mnogim poznatim programskim jezicima [15].

Ipak, pre nego što se započne razvoj MVC strukture nove komponente, najpre se definiše tip dokumenta koji se vezuje za komponentu koju programer razvija. Ovaj tip dokumenta je i krajnji proizvod koji urednik može da dodaje u sistem i popunjava ga sadržajem koji će biti prikazan na samom sajtu.

Razvoj nove komponente najčešće i počinje upravo tu, definisanjem novog tipa dokumenta. Novi tip dokumenta se najčešće kreira preko korisničkog interfejsa u samom BrXM CMS sistemu. U CMS-u se treba pozicionirati u deo za upravljanje svim tipovima dokumenata (i komponenata), tako što se odabere iz padajuće liste u delu sa sadržajem opcija *Document types*. Na slici 2.11 se vidi kako taj deo sistema izgleda. Klikom na postojeći direktorijum se dobija mogućnost pravljenja novog tipa dokumenta (opcija *New document type*). Odabirom ove akcije otvara se deo za uređivanje novih tipova dokumenata. Na slici 2.11 se vidi da postoji i mogućnost održavanja većeg broja imenskih prostora (engl. *namespaces*), kako bi moglo da postoji isto imenovanje za više komponenti koje imaju različitu funkcionalnost.



Slika 2.11: Pravljenje novog tipa dokumenta

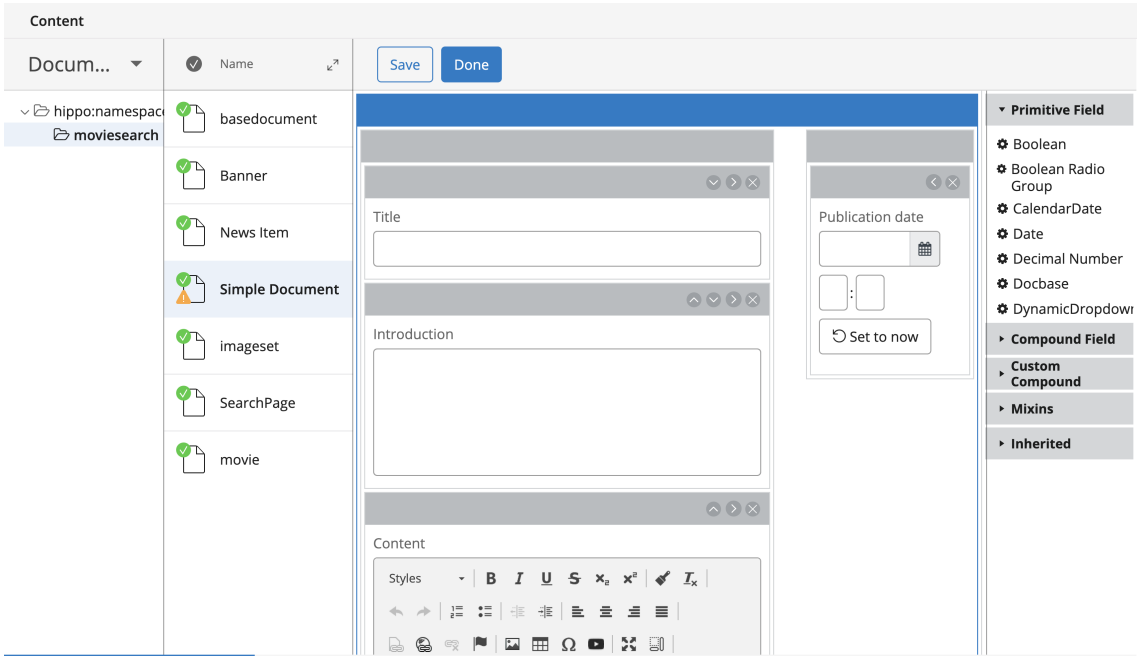
Potrebno je da budu definisana sva polja koja bi mogla da budu potrebna urednіku da prikaže željeni sadržaj na sajtu. Treba razmisliti o vrsti sadržaja koji će biti predstavljen ovim dokumentom (na primer, da li se koristiti za postavljanje vesti na sajt, ili je u pitanju uputstvo za neki samostalni projekat) i da se shodno tome grade delovi dokumenta.

Pomoću uređivača tipa dokumenta (slika 2.12), se dodaju potrebna polja. Postoji mogućnost odabira između primitivnih polja i komponenti. Primitivna polja imaju osnovne tipove polja kao što su: tekstualno polje, polje za datum, grupa radio dugmića itd. Druga mogućnost je dodavanje drugih komponenti, koje već dolaze sa sistemom, kao i komponenti koje su novo razvijene.

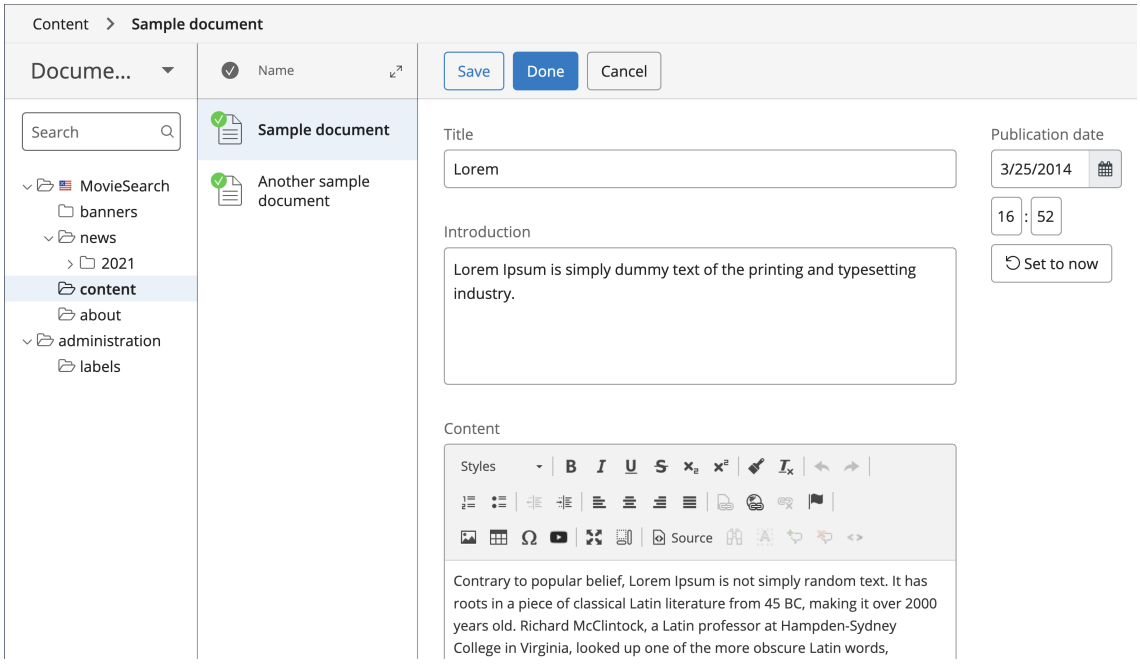
Nakon što se odaberu željena polja i završi sa pravljenjem novog tipa dokumenta, izborom opcije *Type Actions* se taj tip dokumenta objavljuje u sistemu. Sada je ovaj tip dokumenta spreman za korišćenje.

Posle ovih koraka, kada se pristupi delu sadržaja za dokumente (engl. *Documents*) u sistemu i pokrene pravljenje novog dokumenta, naćiće se i ovaj tip dokumenta u ponudi. Na slici 2.13 se vidi kako izgleda kada se odabere dodavanje ovog tipa dokumenta u sistem – otvoriće se prozor za uređivanje sa svim prethodno definisanim poljima. Vrednosti koje se tu upišu će biti prikazane na sajtu. Ovo je deo sistema u kome rade isključivo urednici.

Nakon što smo definisali tip dokumenta, potrebno je definisati delove MVC obra-



Slika 2.12: Pravljenje polja u novom tipu dokumenta



Slika 2.13: Izmena novog dokumenta

sca koji odgovaraju ovom tipu dokumenta – model, kontroler i prikaz (tj. šablon koji definiše kako će izgledati prikaz ovog dokumenta na sajtu). Sve ovo je neophodno da bi se sadržaj uspešno prikazao na sajtu.

Prvo se pravi model za novi tip dokumenta. Modeli su implementirani kao takozvani POJO (engl. *Plain old Java Object*) objekti, tj. obični Java objekti koji obmotavaju stvarni sadržaj koji je uskladišten u JCR čvorovima. Svaki dokument mora da ima odgovarajuću klasu koja je njegov model i koja proširuje *HippoBean* klasu, koju smo mogli videti na slici 2.7. U najnovijoj verziji BrXM sistema, za većinu tipova dokumenata ova klasa se automatski generiše. Naravno, uvek se može napraviri eksplicitno model ako ima potrebe za nestandarnim ponašanjem.

U BrXM sistemu, kontroleri najčešće služe da pokupe podatke iz dokumenta i da ih pripreme i učine dostupnim za prikaz na sajtu. Oni se definišu pomoću Java klasa, koje se nalaze u *site-components* delu koda. Sve klase koje su kontrolori proširuju *BaseHstComponent*.

```
1  package org.example.components;
2
3  imports ...
4
5  public class SimpleComponent extends BaseHstComponent {
6
7      @Override
8      public void doBeforeRender(final HstRequest request, final
       HstResponse response) throws HstComponentException {
9          super.doBeforeRender(request, response);
10         final HstRequestContext ctx = request.getRequestContext();
11
12         // Retrieve the document based on the URL
13         HippoBean document = ctx.getContentBean();
14
15         if (document != null) {
16             // Put the document on the request
17             request.setAttribute("document", document);
18         }
19     }
20 }
```

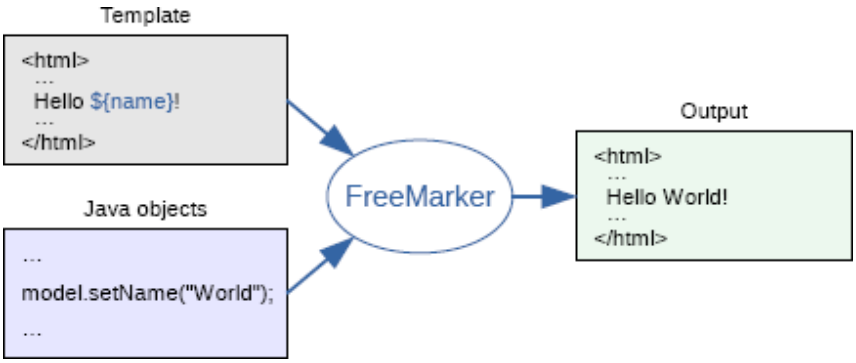
Glavna metoda koja mora da se nalazi u ovoj klasi je *doBeforeRender()* metoda. U toj metodi se rade sve željene modifikacije podataka i spremamo ih za prikaz na sajtu korisnicima.

U ovom jednostavnom primeru, model klase se izvlači iz *ctx* promenljive koja je tipa *HippoBean*, pomoću preslikavanja koje će kasnije biti objašnjeno. Zatim se model sa vrednostima upisuje u atribut zahteva i time ga stavljamo na raspolaganje

šablonu koji se koristi za prikaz.

Kao što je ranije pomenuto, BrXM ima mogućnost da funkcioniše kao bezglavi sistem, u kom slučaju nije potrebno razvijati deo MVC obrazca koji je zadužen za prikaz. Ukoliko ipak ima ptrebe da aplikacija uključuje i prikaz sadržaja, može se koristiti *Freemarker šablon*. U pitanju je procesor šablona (engl. template engine) zasnovan na Java programskom jeziku koji podržava upotrebu MVC obrasca. Procesor šablona predstavlja softversku komponentu koja generiše tekst (HTML sadržaj, SQL skripte, Java izvorni kôd, itd.).

Iako može da generiše različite teskstualne izlaze, Freemarker se najčešće koristi za generisanje HTML stranica. Ovakvom podelom je omogućeno dizajnerima da menjanju izgled stranica bez potrebe da programeri menjaju druge delove koda aplikacije.



Slika 2.14: Freemarker

Na slici 2.14[23] je prikazana osnovna šema transformacije koju vrši Freemarker procesor kod veb aplikacija zasnovanih na MVC-u. Sa jedne strane se nalaze već napravljene Freemarker šablonske stranice, a sa druge Java kôd. Java kôd predstavlja model podataka koji sadrži atribute i njihove vrednosti. Freemarker procesor koristi pripremljene promenljive prosleđene iz Java klasa čije vrednost umeće u šablonske stranice i kao rezultat daje statičke HTML stranice.

Kao dodatak regularnoj Freemarker sintaksi i izrazima, imamo na raspolaganju i BrXM specifične tagove [24]. Freemarker šabloni se u kodu uglavnom čuvaju u `repository-data/webfiles/src/main/resources/site/freemarker/` datoteci.

Primer izgleda jednog freemarker šablona je prikazan u nastavku. Izgled ovog šablona odgovara komponenti sa slike 2.13. Prvo se vrši provera da li dokument

[23]Slika je preuzeta sa sajta `https://freemarker.apache.org/`
[24]Spisak se može naći na sajtu `https://javadoc.onehippo.org/13.0/tlddocs/`

postoji, a zatim, ukoliko postoji, se prikazuje njegov sadržaj.

```
<html>
<head>
</head>
<body>
  <#if document??>
    <h1>${document.title?html}</h1>
    <div>
      <@hst.html hippohtml=document.content />
    </div>
  </#if>
</body>
</html>
```

Podešavanje dokumenata i stranica

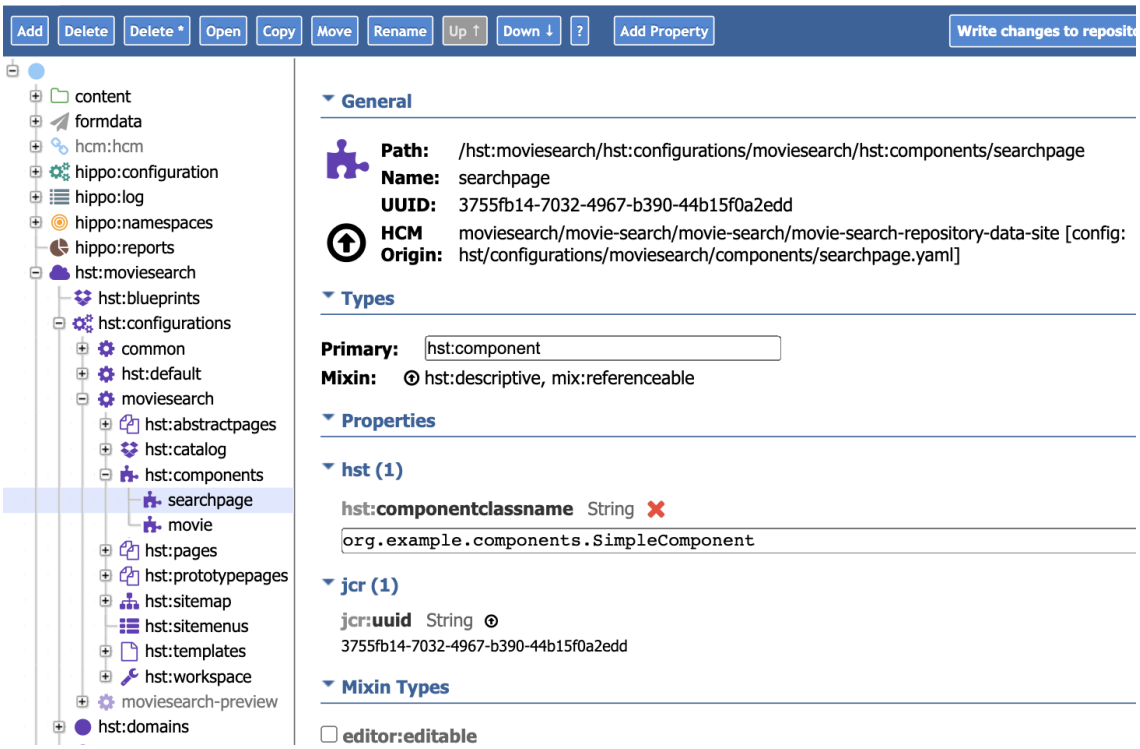
Sada kada su napravljeni podaci, izgled i funkcionalnost za novi tip dokumenta, potrebno je još i da se povežu svi ti delovi, a to se radi preko datoteka za podešavanje. Može im se pristupiti direktno iz koda, ali može i da se koristi konzola, tj, veb aplikacija za upravljanje podešavanjima. Izgled ove veb aplikacije se vidi na slici 2.15.

Datoteka za podešavanje su u *YAML*[25] formatu. YAML je jezik za serializaciju podataka koji se često koristi za stvaranje datoteka za podešavanje, a može se koristiti u bilo kom programskom jeziku. U smislu izražajnosti, YAML je nadskup JSON-a[26], iako je sintaksa nešto drugačija. Jedna od glavnih sintaksnih razlika je u tome što YAML koristi nove redove i uvlačenja za razdvajanje polja, dok JSON u tu svrhu koristi zagrade.

Prvo je pokazano kako dodati kontroler za novi tip dokumenta. To se radi dodavanjem novog čvora u *hst:components*, koji u navedenom primeru nosi naziv *simplecomponent*. Zatim se dodaje osobina *hst:componentclassname* i podešava putanja ka novoj klasi. Kako ovo izgleda u YAML datoteci se vidi u primeru ispod.

```
/hst:project/hst:configurations/myproject/hst:components:
  /simplecomponent:
    jcr:primaryType: hst:component
    hst:componentclassname: org.example.components.SimpleComponent
```

[25] https://yaml.org/
[26] https://www.json.org/json-en.html



Slika 2.15: Podešavanje u konzoli

Pored dodavanja podešavanja za kontroler, potrebno je dodati i podešavanje za šablon. U slučaju šablona se dodaje novi čvor, koji će imati naziv *simple*, u *hst:templates* čvor, a zatim mu se dodaje osobina *hst:renderpath* sa putanjom do šablona.

```
/hst:myproject/hst:configurations/myproject/hst:templates:
  /simple:
    jcr:primaryType: hst:template
    hst:renderpath: webfile:/freemarker/simple.ftl
```

Sada kada je u sistemu registrovana komponenta sa kontrolerom, kao i njen šablon, ostaje još da se omogući da se prilikom pristupa određenoj putanji u okviru sajta prikaže baš ta komponenta. Najpre treba dodati novu stranicu, tako što se čvoru *hst:pages* doda novi dete čvor (u ovom primeru se taj čvor naziva *home*). Zatim se njemu dodaje osobina *hst:referencecomponent* i poziva malopre definisana komponenta (tip dokumenta). Takođe, se dodaje i osobina u kojoj se referiše na novo definisanu komponentu *hst:components/home*. Primer podešavanja se nalazi u narednom kodu.

```
/hst:myhippoprojet/hst:configurations/myproject/hst:pages/home:
```

```
  hst:referencecomponent: hst:components/simplecomponent
  hst:template: simple
```

Ostaje još da se podesi URL putanja stranice sa sadržajem. Ovo se može uraditi preko konzole. Sve što je potrebno je da se doda novi čvor u *hst:sitemap*. Zatim tom čvoru da se doda polje *hst:relativecontentpath* sa vrednošću putanje novog dokumenta.

```
/hst:myproject/hst:configurations/myproject/hst:sitemap:
 /root:
  jcr:primaryType: hst:sitemapitem
  hst:componentconfigurationid: hst:pages/home
  hst:relativecontentpath: home
```

Novi tip dokumenta je sada u potpunosti na raspolaganju urednicima za korišćenje. Prilikom pristupa određenoj stranici, njen sadržaj će se prikazati korisniku.

Primer koji je ovde prikazan predstavlja osnovni način dodavanja novih funkcionalnosti u BrXM sistem, koji brzo dovodi do većih rezultata i odgovara potrebama aplikacije koja je razvijena uz ovaj rad. Ovo nije jedini način za pravljenje novih komponenti i dokumenata. Naime, nije uvek potrebno da se definiše i nova strana, može postojati i više komponenti koje su deo jedne strane.

Glava 3

Elasticsearch

Elasticsearch je distibuirani server otvorenog koda koji služi kao sistem za pretragu i analizu podataka. Napisan je u Java programskom jeziku, što omogućava pokretanje na svim platformama. Baziran je na *Apache Lucene*[1] biblioteci za indeksiranje. Elasticsearch pruža JSON zasnovan REST API kao omot oko Apache Lucene funkcija. Elasticsearch omogućava korisnicima da pretražuju veliku količinu podataka vrlo brzo. Elasticsearch se koristiti i za čuvanje podataka ali je njegova primarna uloga indeksiranje i pretraga podataka u realnom vremenu.

3.1 Istorija i osnovne karakteristike

Elasticsearch je proizvod kompanije *Elastic* čiji je jedan od osnivača Šej Banon. Početak rada na projektu koji će dovesti do Elasticsearch servisa kakav danas postoji vezuje se za 2000. godinu. Zanimljivo je da je motivaciju za taj projekat Šej dobio radeći na aplikaciji za kuvanje. Ideja je bila da napravi jednostavnu aplikaciju, sa poljem za pretragu podataka, a podaci su bili pretežno tekstualnog tipa. Dok je radio na ovoj aplikaciji on je došao prvi put u kontakt sa Apache Lucene bibliotekom.

Apache Lucene je biblioteka napisana u Java programskom jeziku koja omogućava pretragu velike količine tekstualnih podataka sa visokim performansama. Besplatna je i pogodna za skoro svaku primenu. Međutim, kada je Šej pokušao da je doda u svoj projekat uvideo je da to nije tako lako. Primetio je da može lako da mapira podatke sa relacionom bazom podataka, koristeci objektno relaciono mapiranje (engl. *object relational mapping*), ali ne i sa bibliotekom Lucene. U želji da sebi olakša ovakvo povezivanje nastala je biblioteka koja se zvala *Kompas*. Biblioteka je

[1] `https://lucene.apache.org/`

pomagala pri mapiranju podataka u obliku pogodnom za Apache Lucene. Biblioteka je bila otvorenog koda i ubrzo je postala široko korišćena.

Vremenom je priliv podataka postao višestruko veći ulaskom u internet doba, doba pretraživača i društvenih mreža. Često se događalo da na raspolaganju imamo ogromnu količinu podataka, pa se pojavilo pitanje kako da se koristi ova biblioteku ako postoji više podataka nego što može da stane na jednu mašinu. Tada je Šej počeo da radi na tome da napravi distribuirano rešenje. Kao rezultat ovog rada, nastaje Elasticsearch servis.

ELK Stek

U vreme kada je Elasticsearch nastao, postojala su još dva projekta koja su brzo rasla i koja su usko vezana sa njim. Oba projekta su takođe otvorenog koda. Džordan Sisel je radio na *Logstash* projektu. Logstash se brine o tome da unos podataka bude transformisan u jedan od podržanih formata koji će se poslati nekoj drugoj aplikaciji, poput Elasticsearch-a, na dalju analizu i korišćenje. Pokušao je i da napravi korisnički interfejs za Elasticsearch, ali nije bio baš uspešan u tome. Međutim, na istom problemu je radio i Rašid Kan, koji je uspeo da naparavi dobar korisnički interfejs koji se zove *Kibana*. Danas je teško zamisliti da se radi samo sa Elasticesearch serverom bez korišćenja ovih dodatnih tehnologija.

Njih trojica su vremenom odlučili da se udruže i kao rezultat nastaje kompanija Elastic, osnovana 2012. godine i takozvani ELK stek — Elasticsearch, Logstash i Kibana. Kasnije te godine su napravili i par komercijalnih rešenja, većinom vezanih za sigurnost podataka, kako bi mogli da lakše održe svoju kompaniju u životu, a glavni proizvodi su ostali otvorenog koda.

Osnovne karakteristike

Osnovne karakteristike Elasticsearch servera su:

- omogućava rad sa velikom količinom podataka,
- koristi no-SQL baze za skladištenje podataka,
- koristi denormalizaciju kako bi ubrzao pretragu,
- omogućava rad u realnom vremenu,
- može da se distribuira,

- podatke predstavlja pomoću JSON objekata,
- besplatan je i dostupan pod Apache licencom,
- omogućava jednostavnu analizu i vizuelizaciju podataka.

3.2 Osnovni koncepti

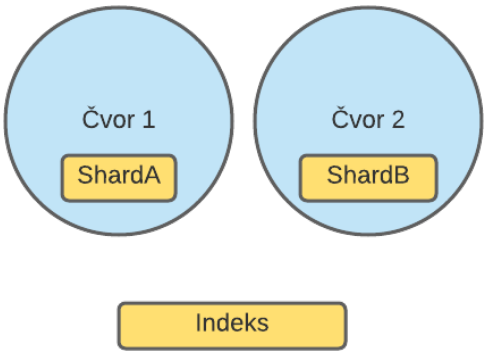
Najpre je prikazana strukturu podataka sa kojom radi Elastisearch. Osnovna jedinica od koje se polazi je *polje*. Polje predstavlja jedan podatak. Skup polja predstavlja jedan JSON objekat. Skup JSON objekata se naziva *dokument*. U ranijim verzijama je kolekcija dokumenata koji imaju ista polja nazivana *tip*. Tipovi su se dalje organizovali u strukturu koja se nazivala *indeks*. Na primer, indeks je mogao da sadrži podatke o društvenoj mreži, gde su tipovi profili, komentari, poruke i sl. Tip je uklonjen u novijim verzijama Elasticsearch servera, od verzije 7+. Preporuka je da sve što je bio tip u prethodnim verzijama, bude samostalni indeks. U prethodno navedenom primeru sa indeksom o društvenom mrežom bi to značilo da se preporučuje korišćenje odvojenih indeksa za profile, komentare, poruke itd.

Čvor predstavlja jednu pokrenutu instancu Elastcsearch-a. Dakle, moguće je pokrenuti više instanci Elastisearch-a i rasporediti ih na različite servere nekog distribuiranog sistema. Ono što ubrzava pretragu je baš ova mogućnost distribuiranja, jer na taj način pretraga može da se paralelizuje. Više čvorova koji su povezani predstavljaju *klaster*. Kad se pokrene istanca Elasticsearch servera, klaster se automatski pravi, čak i kad ima samo jedan čvor. Klaster omogućava da se čitav sistem posmatra kao celina. Grafički prikaz klastera se vidi na slici 3.1.

Indeksi, koji predstavljaju skupove dokumenata, se horizontalno mogu podeliti u strukture koje nazivamo *šard*-ovi (engl. *shard*). Ovo omogućava da se jedan indeks rasporedi na više različitih servera. Osim toga, podrazumevano je postojanje jednog duplikata, ali se mogu kreirati i dodatni duplikati indeksa i njegovih delova. *Duplikati* takođe mogu da se rasporede na više servera i pružaju veću sigurnost podataka kao i bržu pretragu. Nije obavezno da instance budu na različitim serverima – moguće je da se sve pokrenu na jednom serveru.

Pokretanje Elasticsearch servera

Postoji više načina za preuzimanje i pokretanje Elasticsearch servera. Pokazan je postupak preuzimanja koda Elasticsearch projekta na lokalnoj mašini i direktnog



Slika 3.1: Prikaz jednog klastera

pokretanja na njoj. Komanda za preuzimanje projekta iz terminala na Linux sistemu je:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/
   elasticsearch-7.12.1-linux-x86_64.tar.gz.sha512
shasum -a 512 -c elasticsearch-7.12.1-linux-x86_64.tar.gz.sha512
tar -xzf elasticsearch-7.12.1-linux-x86_64.tar.gz
cd elasticsearch-7.12.1/
```

Istanca se može pokrenuti preko komandne linije pomoću sledeće komande:

```
./bin/elasticsearch
```

Kominikaciju sa Elasticsearch serverom moguće je vršiti pomoću *curl*[2] komande. Provera da li je server dobro pokrenut se može izvršiti zadavanjem odgovarajuće curl komande u komandnoj liniji, ili ako se pomoću pregledača pristupiti `localhost:9200/?pretty` adresi. U nastavku se nalazi primer poruke koju servis vraća kod dobro pokrenute Elasticsearch instance.

```
BG-C-00045:~ van1bg$ curl -X GET "localhost:9200/?pretty"
{
  "name" : "BG-C-00045",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "EYGvElNrS3-XFCKrYKTHmA",
  "version" : {
    "number" : "7.12.0",
    "build_flavor" : "default",
    "build_type" : "tar",
```

[2] `https://curl.se/`

```
    "build_hash" : "78722783c38caa25a70982b5b042074cde5d3b3a",
    "build_date" : "2021-03-18T06:17:15.410153305Z",
    "build_snapshot" : false,
    "lucene_version" : "8.8.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Poređenje sa relacionim bazama podataka

U relacionim sistemima, bazu podataka predstavlja skup tabela, dok se u slučaju Elasticserch-a, baza sastoji iz skupa indeksa koji se nalaze u klasteru. Tabela, odnosno indeks, su jedinice nad kojima se vrše upiti, pa se mogu posmatrati kao analogone. Na najnižem nivou su polja i kolone, koje predstavljaju same podatke koji će se čuvati u bazi. Primetna je razlika da Elasticsearch polje može da ima i listu kao tip podataka, dok se u SQL-u može čuvati samo jedan podatak deklarisanog tipa.

Od polja se dalje grade redovi (u relacionim bazama), odnosno dokumenti (u Elasticsearch-u). Glavna razlika između reda i dokumenta je u tome što je red striktnije definisan. Iako dokument ima strukturu, ona nije tako striktna, te omogućava lakše izmene i prilagođavanje novim podacima. Elasticsearch može prihvatiti i novo polje bez ikakve dodatne potrebe za izmenom mapiranja dokumenta (šema u relacionim bazama).

Kao što se može videti, korespondencija između pojmova u relacionim sistemima i Elasticsearch-u nije u potpunosti „jedan na jedan", a i semantika se malo razlikuje. Ipak, više je sličnosti nego razlika.

Iako Elasticsearch može da se koristi kao primarna baza, tako nešto u većini slučajeva nije preporučljivo. Pojedine operacije poput dodavanja ili izmena podataka se izvršavaju znatno sporije nego u relacionim bazama. Primarna upotreba Elasticsearch-a je u primenama koje podrazumevaju pretragu velike količine podataka, naročito u slučaju kada je potreban uvid u podatke u realnom vremenu (engl. *real-time*). Način na koji se grade dokumenti podrazumeva denormalizovan pristup, tako da nemamo potrebu za spajanjem tabela, što može biti veoma skupa operacija u relacionim bazama podataka.

3.3 Unos podataka

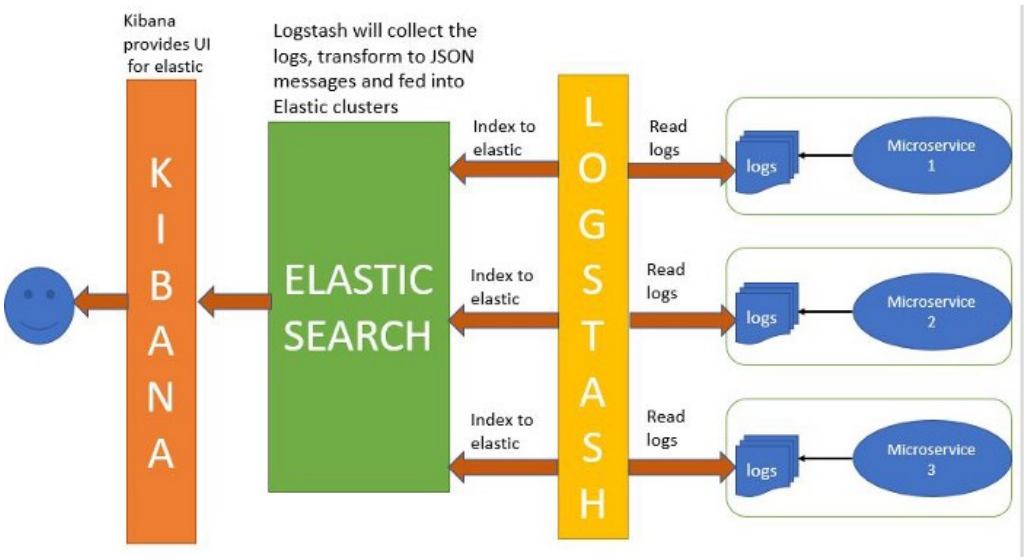
Da bi Elasticsearch mogao da se koristi za pretragu i analizu podataka, najpre je potrebno uneti podatke u Elasticsearch. Podaci koji se uvoze u Elasticsearch sistem mogu biti dostupni u raznim formatima. Neki od načina za unos podataka u Elasticsarch su unos podataka iz komandne linije, uz pomoć curl komande, kao i putem korisničkog interfejsa – Kibane. U nastavku je opisan pristup zasnovan na korišćenju Logstash aplikacije.

Logstash

Logstash je alat koji značajno olakšava manipulaciju podacima i njihovo dovođenje u oblik koji odgovara Elasticsearch serveru. Razvijen je u JRuby[3] programskom jeziku, tako da se pokreće pomoću Java virtuelne mašine (JVM). Logstash će se pobrinuti da ulazni podaci budu transformisani u jedan od podržanih formata koji će se kasnije predati Elasticsearch serveru. Logstash se nalazi između podataka i aplikacije u koju treba da se pošalju. Moguće je istovremeno raditi sa više izvora podataka i slati transformisane podatke na više različitih lokacija. Logstash ima mogućnost interakcije sa podacima i dodatne transformacije, ukoliko je potrebno. Takođe, postoji mogućnost anonimizacije podataka, poput isključivanja svih ličnih podataka. Isključivanje svih ličnih podataka može biti od velike koristi pri radu na zaštiti podataka. Na slici 3.2[4] se vidi prikaz komunikacije između svih delova ELK steka.

[3] `https://www.jruby.org/`

[4] Slika je preuzeta sa stranice: `https://www.javainuse.com/spring/springboot-microservice-elk`



Slika 3.2: ELK stek komunikacija

Dokument za podešavanje unosa podataka pomoću Logstash-a ima tri dela: *ulaz*, *transformaciju* (koja uključuje filtritanje) i *izlaz*. Prvi deo je *ulaz*, koji pored običnih dokumenata mogu da predstavljaju i sistemski zapisi. *Transformacija i filtriranje* se odvijaju prema skupu prethodno definisanih pravila.

Zahvaljujući dostupnim dodacima, *izlaz* može biti u velikom broj različitih formata, a u sledećem primeru se koristi format koji odgovara Elasticsearch-u. U nastavku se nalazi primer jednog dokumenta za podešavanje:

```
input {
    file {
        path => "/home/Documents/esearch/example.csv"
        start_position => "beginning"
        }
    }
filter {
    csv {
        columns => [ "id", "name", "height" ]
    }
    mutate{
        convert => [ "height" , "integer" ]
    }
    }
output {
    elasticsearch {
    hosts => ["localhost:9200"]
```

```
18      stdout {}
19      }
20  }
```

U dokumentu za podešavanje prvo je definisano odakle dolaze podaci, a u tu svrhu je korišćen ključ *input*. U ovom primeru se očekuje da ulaz bude datoteka koja se nalazi na putanji definisanoj poljem *path*. Poljem *start_position* se definiše da čitanje podataka počinje od početka dokumenta. Zatim, se definiše tip dokumenta, kao i kolone koje će posle transformacije postati polja na Elasticsearch serveru. Logstash sve vidi kao tekstualni tip, tako da je neophodno eksplicitno definisati sve što treba da bude drugog tipa i to se radi u *mutate* delu dokumenta.

U delu dokumentu za podešavanje za izlaz nema mnogo dodatnih opcija, samo lokaciju servera i naziv indeksa koji se popunjava podacima. Ukoliko indeks ne postoji, automatski će se napraviti novi indeks sa datim imenom.

Pokretanje Logstash aplikacije i unosa podataka se vrši pomoću sledeće komande:

```
1 bin/logstash -f logstash-filter.conf
```

Postavljanjem *stdout* u izlaznom delu dokumenta, omogućeno je praćenje unosa podataka, tako što će se na standardnom izlazu prikazati ispis. Primer kako ispis može izgledati se nalazi u nastavku. Format u kom se ispisuje podseća na JSON.

```
1  {
2                  "@timestamp" => 2021-05-04T20:10:51.580Z,
3                    "@version" => "1",
4                        "host" => "BG-C-00045",
5                        "path" => "/home/Documents/esearch/example.csv
       ",
6                          "id" => "00001",
7                        "name" => "John Smith",
8                      "height" => "183",
9                     "message" => "00001,John Smith,183"
10 }
```

Kibana

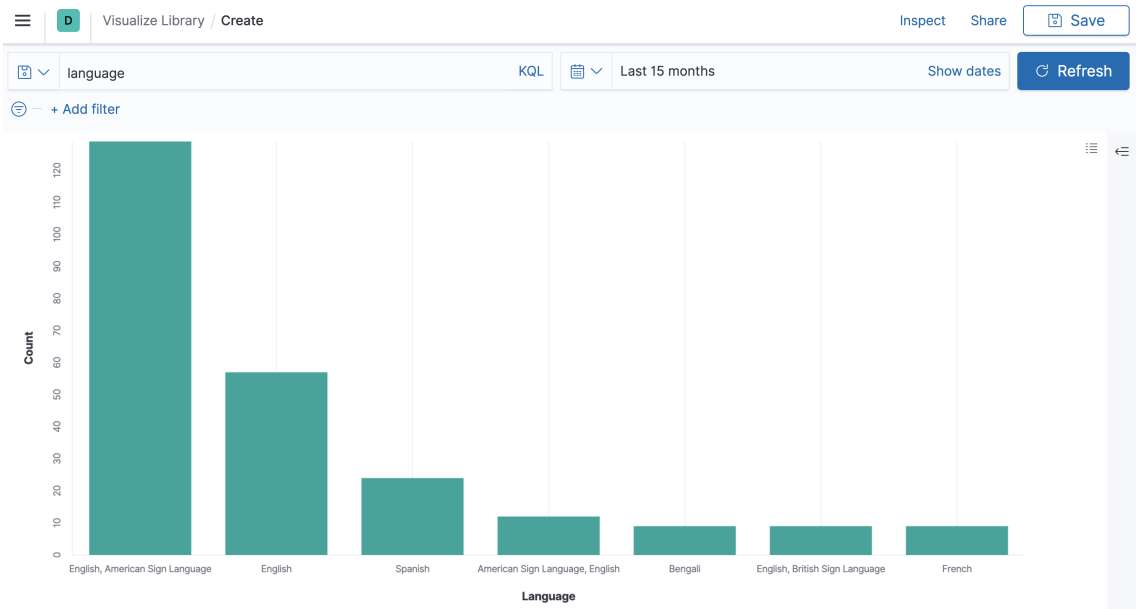
Kibana je korisnički interfejs za Elasticsearch koji pruža mogućnost vizualizacije sadržaja indeksiranog na Elasticsearch serveru. Korisnici mogu kreirati trakaste, linijske i raštrkane dijagrame kao i „pita” grafikone velikih količina podataka. Kroz Kibana aplikaciju se mogu lako vršiti upiti nad Elasticsearch indeksima, mogu se kreirati novi indeksi, menjati mapiranja, unositi podaci, vršiti pretrage itd. Kibana

takođe pruža alat za prezentaciju, nazvan *Canvas*, koji omogućava korisnicima da kreiraju prezentacije koje povlače podatke direktno iz Elasticsearch-a.

Kibana se pokreće tako što se pristupi direktorujimu gde je prethodno preuzeti izvorni kôd Kibane sa komandnom linijom i zatim izvrši naredna komanda:

```
./bin/kibana
```

Kibana je nakon toga dostupna na adresi `http://localhost:5601`. Na slici 3.3 se vidi primer upotrebe aplikacije Kibana da se vizuelno prikaže koliko filmova na kom jeziku postoji u datoj bazi. Vidi se da ima najviše filmova na engleskom jeziku.



Slika 3.3: Kibana

3.4 Mapiranje podataka

Bitan deo izrade svake baze podataka je definisanje strukture baze i tipova podataka koji će se čuvati u njoj. Mapiranje je proces u Elasticsearch serveru koji podrazumeva definisanje dokumenata, kao i tipova polja koja se nalaze u njima. Ovo je ekvivalentno procesu pravljenja šeme u relacionim bazama.

Svaki dokument je skup polja, a svako polje je određenog tipa. Elasticsearch nudi preko 20 različitih prostih i složenih tipova koji se mogu odabrati. Za tekstualna polja postoji više mogućih tipova, zavisno od toga kako je potrebno da se podatak procesuira. Na primer, mogu se odabrati *text* ili *keyword* kao tip polja. Razlika je

u tome što se kod *text* tipa polja njegova vrednost analizira (pomoću analizatora), dok u slučaju *keyword* tipa, polje ostaje nepromenjeno i očekuje se da se koristi za sortiranje ili filtriranje.

Elasticsearch podržava dva tipa mapiranja. To su dinamičko mapiranje i eksplicitno mapiranje. *Dinamičko mapiranje* pomaže da se brže započne sa radom nad podacima, tako što Elasticsearch sam odradi mapiranje za nas. Kako bismo aktivirali dinamičko mapiranje, potrebno je postaviti polje *dynamic* na vrednost *true*. Kada je dinamičko mapiranje aktivirano, Elasticsearch sam prepoznaje logičke vrednosti (engl. *boolean*), brojeve sa pokretnim zarezom, cele brojeve, objekte, nizove (zavisno od tipa prvog elementa) i tekstualna polja. Ovi podaci se redom prevode u tipove *boolean*, *float*, *long*, *object*, dok se tekstualna polja, u zavisnosti od tipa teksta, mogu prevesti u *date*, *float*,*long*, *text* ili *keyword*.

Eksplicitno mapiranje omogućava da programer sam odredi koji je željeni tip polja. Moguće je da se definiše koja polja će služiti za tekstualnu pretragu, kakav je format datuma, da li polje sadrži brojeve, datum ili geolokaciju, i sl. Uvek se može definisati i kako je željeno mapiranje dinamički dodatih polja. Postoji i mogućnost korišćenja izvršnih (engl. *runtime*) polja, kako bi mogli da menjamo mapiranje, ali bez potrebe za reindeksiranjem, čime se dobija na preformansama. Kada se pošalje upit Elasticsearch serveru, skripta koja je vezana za izvršna polja će se pokrenuti i izračunati vrednost tih polja.

S obzirom na jednostavnost mapiranja, postoji opasnost od dodavanja prevelikog broja polja u dokument, na šta treba obratiti pažnju. Postoji i podešavanje kojim se ograničava broj polja u dokumentu, a podrazumevana vrednost ovog podešavanja je 1000 polja.

Primer jednog mapiranja je dat u nastavku:

```
1
2 PUT /movies/
3 {
4   "mappings": {
5     "properties" : {
6     "@timestamp" : {
7           "type" : "date"
8         },
9         "description" : {
10          "type" : "text",
11          "analyzer" : "english"
12        },
```

```
        ...
        ...
        ...,
        "genre" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "title" : {
          "type" : "text",
          "analyzer": "autocomplete"
        }
      }
    }
}
```

Primera radi, posmatrajmo mapiranje za polje *description*:

```
        "description" : {
          "type" : "text",
          "analyzer" : "english"
        },
```

U primeru se vidi da je ovo polje tipa *text*. Takođe, postoji i dodatno polje *analyzer*, čije je značenje detaljnije objašnjeno u nastavku.

3.5 Analizatori i invertovani indeksi

Analiza podataka u Elasticsearch serveru se koristi u cilju unapređenja tekstualne pretrage. Naime u zavisnosti od toga da li je definisan *analizator* (engl. *analyzer*), kao i koji tip analizatora je u pitanju, vrednosti se mogu dodatno procesuirati na različite načine. Postoji veliki broj unapred dostupnih različitih analizatora. *Standardni analizator* (engl. *Standard Analyzer*) deli tekst na reči (tokene). Ukoliko se eksplicitno ne definiše analizator, primeniće se standardni analizator. Pored toga, uklanja većinu interpunkcijskih simbola, transformiše velika u mala slova, a podržava i uklanjanje zaustavnih reči. Podešavanje uklanjanja zaustavnih reči vrši se pomoću polja *stopwords*, čija je podrazumevana vrednost _*none*_. Često korišćena

vrednost je _*english*_ (zaustavne reči u ovom slučaju su: a, an, and, are, as, itd.). Moguće je definisati i listu zaustavnih reči kao vrednost polja *stopwords*.

Analizator ključnih reči (engl. *keyword*) je analizator koji prihvata bilo koji tekst koji mu je dat i daje potpuno isti tekst nazad. Pošto je podrazumevani analizator standardni analizator, ukoliko tekst ne treba da se analizira moguće je koristiti analizator ključnih reči, čime će se izbeći analiza teksta.

Pored navedenih analizatora, Elasticsearch nudi i mnoge analizatore specifične za prirodne jezike, poput engleskog i francuskog.

U nastavku je naveden primer koji će pokazati šta se tačno dešava sa rečenicom prilikom njene analize. Rečenica nad kojom je pokrenut analizator je „42. A LONG time ago in a galaxy far, far.... ".

```
1
2 POST /_analyze
3 {
4   "text": "42. A LONG time ago in a galaxy far, far....",
5   "analyzer": "standard"
6 }
```

Rezultat ove analize teksta je niz tokena. Može se primetiti da će analiza preskočiti znakove interpukcije, velika slova transformisati u mala i vidi se da uklanjanje zaustavnih reči nije uključeno, tako de će i one biti deo tokena. Izlaz i spisak dela tokena posle izvršene analize je:

```
1 {
2   "tokens" : [
3     {
4       "token" : "42",
5       "start_offset" : 0,
6       "end_offset" : 2,
7       "type" : "<NUM>",
8       "position" : 0
9     },
10    {
11      "token" : "a",
12      "start_offset" : 4,
13      "end_offset" : 5,
14      "type" : "<ALPHANUM>",
15      "position" : 1
16    },
17    {
18      "token" : "long",
```

```
19        "start_offset" : 32,
20        "end_offset" : 35,
21        "type" : "<ALPHANUM>",
22        "position" : 8
23      },
24      ...
25      ...
26      {
27        "token" : "far",
28        "start_offset" : 37,
29        "end_offset" : 40,
30        "type" : "<ALPHANUM>",
31        "position" : 9
32      }
33      ...
34    ]
35  }
```

Invertovani indeks

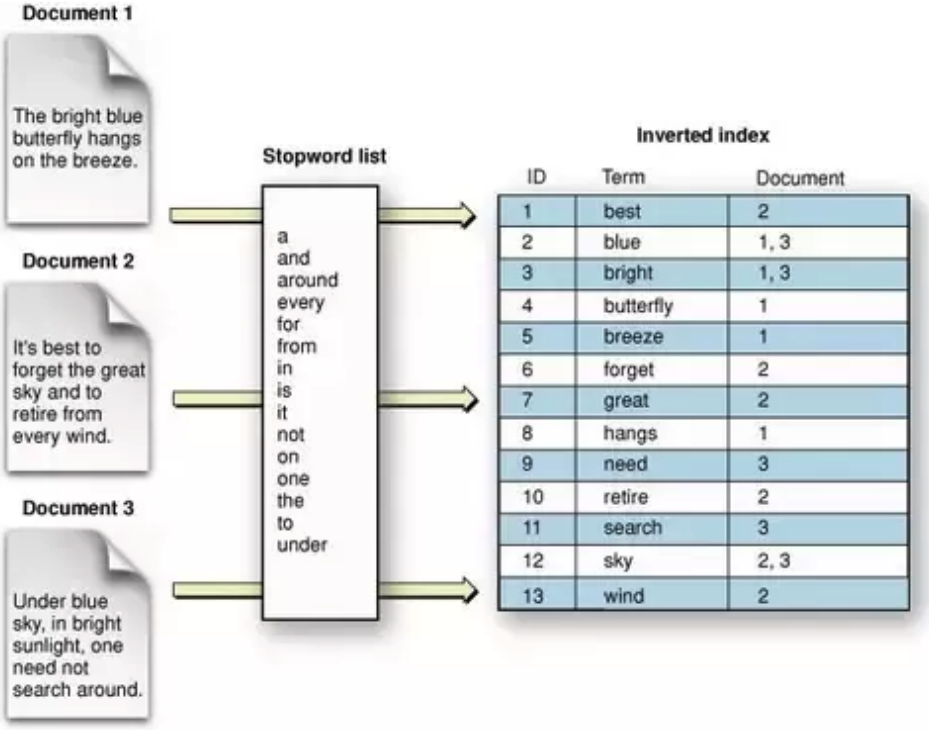
Nekoliko različitih struktura podataka koristi se za čuvanje vrednosti polja u Elasticsearch-u. Sama struktura koja će se koristiti zavisi od tipa podataka odgovarajućeg polja. Razlog za to je da bi se osigurala efikasnost izvršavanja upita. Svim ovim strukturama podataka na nižem nivou upravlja Apache Lucene, a ne Elasticsearch. Invertovani indeks je jedna od glavnih struktura podataka na koju se oslanja Apache Lucene.

Invertovani indeks [12] je struktura koja omogućava brzu pretragu po rečima u velikim kolekcijama teksta. Intuitivno, možemo zamišljati kao da Elasticsearch pravi fasciklu za svaku reč na koju naiđe i zatim u njoj zapisuje dokumenta u kojima se ta reč pojavila. Leksikografski poređane fascikle bi bilo lako pretražiti za određenu reč i zatim vratiti kao rezultat sva dokumenta u kojima je ta reč pronađena. U matematičkom smislu, invertovani indeks je funkcija koja svakom terminu pridružuje skup dokumenata. Ovaj skup se obično predstavlja kao rastući ili opadajući niz celobrojnih vrednosti koje predstavljaju serijske brojeve dokumenata. Na slici 3.4[5] se vidi primer pravljenja invertovanog indeksa nad tri dokumenta. Može se primetiti da između invertovanog indeksa i dokumenata stoji lista zaustavnih reči koje anali-

[5]Slika je preuzeta sa sajta `https://community.hitachivantara.com/s/article/search-the-inverted-index`

zator izbacuje, dok se u samom invertovanom indeksu mogu videti preostale reči u leksikografskom poredku, kao i u kojim dokumentima se pojavljuju.

Invertovani indeksi koji se čuvaju u Apache Lucene mogu sadržati u sebi dodatne podatke, poput podataka koji se koriste za ocenu relevantnosti. Ova ocena omogućava da dobijeni rezultati pretrage budu sortirani prema stepenu podudarnosti sa upitom pretrage. Invertovani indeks se pravi za svako tekstualno polje, dok se polja ostalih tipova, poput numeričkih polja, datuma i podataka geolokacije čuvaju u BKD (engl. *Block of K-Dimensional trees - BKD trees*) drvetima[6]. Razlog za čuvanje ovih podatak u BKD drvetima je efikasnija pretraga ovog tipa podataka.



Slika 3.4: Primer invertovanog indeksa

[6]Više o BKD drvetima u Elastisearch serveru može se naći na sajtu: `https://medium.com/swlh/bkd-trees-used-in-elasticsearch-40e8afd2a1a4`

Glava 4

Prikaz veb aplikacije

U cilju demonstracije upotrebe tehnologija opisanih u prethodnim glavama, razvijena je veb aplikacija koja omogućava pretragu nad podacima o filmovima. Posetioci sajta će moći da vrše pretragu filmova, a kada pronađu željeni film, moći će da vide više detalja o njemu, kao i da pogledaju da li za traženi film postoje recenzije od strane drugih posetilaca sajta. Sa druge strane, urednici mogu dodavati i vesti, poput tekstova o novim filmovima, omiljenom filmu, najboljih 10 filmova određenog žanra itd.

Osim BrXM sistema i Elasticsearch servera, u razvoju ove aplikacije je korišćen i Kotlin programski jezik, kao i Bootstrap[1] i JavaScript[2]. S obzirom da jezik Kotlin spada u relativno nove tehnologije i da kao takav, za razliku od Javascript-a ili Bootstrap-a, nije toliko poznat široj stručnoj javnosti, u nastavku su ukratko predstavljene osnovne karakteristike ovog programskog jezika. Zatim je bliže predstavljena čitaocu sama aplikacija, obljašnjena arhitektura aplikacije na višem nivou, kao i predstavljena baza i bitniji delovi koda aplikacije.

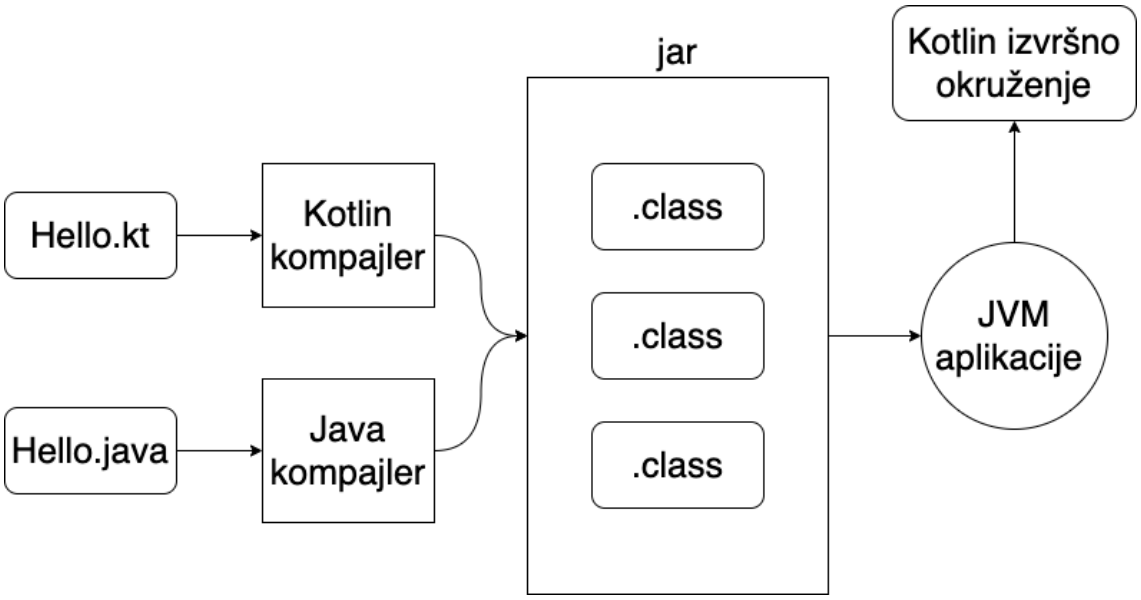
4.1 Kotlin

U ovom poglavlju je ukratko predstavljen programski jezik Kotlin. Dat je kratak osvrt na njegovu istoriju, navedeni su njegove osnovne prednosti u odnosu na druge jezike, a prikazana je i njegova osnovna sintaksa. U projektu se Kotlin koristiti za pisanje modela i kontrolera u BrXM delu projekta.

[1] `https://getbootstrap.com/`
[2] `https://www.javascript.com/`

Kotlin je jedan od novijih programskih jezika, koji se veoma brzo pokazao kao vredan pažnje. Kreiran je od strane kompanije JetBrains. Prva verzija dostupna javnosti je objavljena 2012. godine, dok je prva stabilna verzija objavljena 2016. godine. Ubrzo, već 2017. godine, Kotlin programski jezik je odabran kao preporučeni jezik za pisanje aplikacija za Android uređaje[3].

Osnovna svrha programskog jezika Kotlin je da obezbedi koncizniju i bezbedniju alternativu Java programskom jeziku. Najčešće primene Kotlina su pisanje aplikacija za servere i pisanje aplikacija za Android uređaje. Jedna od najprimetnijih razlika između Java i Kotlin programskih jezika je vidno kompaktnija i jednostavnija sintaksa Kotlin jezika. Dokazano je i da je u nekim slučajevima vreme izvršavanja Kotlin koda kraće od vremena izvršavanja Java koda, jer Kotlin podržava i umetanje (engl. *inline*) čitavog tela funkcije. Što se tiče zauzeća memorije, slično je kao i kod Jave, dok je prevođenje programa malo sporije.



Slika 4.1: Postupak prevođenja izvornog koda u izvršni program

Na slici 4.1 se vidi kako izgleda postupak prevođenja Kotlin izvornog koda, kao i Java izvornog koda u izvršni program. Vide se datoteke *Hello.kt* i *Hello.java*, koje se dalje pomoću kompilatora prevode u datoteke tipa *class*. *Class* datoteke sadrže bajt kôd i biće spakovane u datoteku tipa *.jar*, koje se pokreću na Java virtualnoj mašini. Zato je ovaj jezik u potpunosti interoperabilan sa Java programskim jezikom.

[3]Više informacija može se naći na sajtu `https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/`

Pored već navedenih osobina Kotlin je i statički tipiziran jezik, objektno-orijentisan i funkcionalan (nije čisto funkcionalan, ali pruža veliku podršku funkcionalnom programiranju) [17].

Koncepti

Kao što je ranije pomenuto, Kotlin je sličan Javi, pa je samim tim veliki broj koncepata preuzet iz Jave. U ovom radu se ne prolazi kroz sve koncepte koje Kotlin nudi, već samo kroz one najznačajnije. Puna lista se može naći na zvaničnom sajtu Kotlin programskog jezika [4].

Struktura Kotlin datoteka

Struktura Kotlin datoteka sa izvornim kodom se ne razlikuje puno od strukture datoteka pisanih u Java programskom jeziku. Pomoću ključne reči *package*, definiše se u kom paketu ova klasa pripada, dok se pomoću ključne reči *import* uključuju dinamičke biblioteke. Kako bi se znala početna tačka izvršavanja programa, svaki program pisan u Kotlin programskom jeziku mora da ima funkciju *main*. Ključna reč za definiciju funkcije je *fun*. Funkcija main može i ne mora da ima argumente.

```
1 package my.test
2
3 import kotlin.text.*
4
5 fun main(args: Array<String>) {
6     println("Hello world!")
7     println(42)
8 }
```

Promenljive

U Kotlin programskom jeziku postoje dve vrste promenljivih. Promenljiva koja je definisana pomoću ključne reči *var* kojoj se može dodeliti vrednost više puta, dok promenljiva koja je definisana pomoću ključne reči *val* vrednost može biti dodeljena samo jednom.

Kotlin je statički tipiziran jezik, to znači da je tip svake promenljive poznat u vreme kompajliranja. Ovo pomaže u sprečavanju i otkrivanju mnogih grešaka.

[4]Zanična dokumentacija se može naći na sajtu: `https://kotlinlang.org/docs/home.html`

Iako je statički tipiziran jezik, Kotlin ne zahteva da se eksplicitno navede tip svake promenljive koja se deklariše. Najčešće Kotlin može da zaključi tip promenljive iz izraza kojim je inicijalizovana ili okolnog konteksta.

```
val PI = 3.14
var x = 0

fun incrementX() {
    x += 1
}
```

Nula vrednosti

Kotlin se smatra bezbednijim od Jave, zbog načina na koji vrši dodatnu proveru nula (engl. *null*) vrednosti. Sistem tipova ovog jezika podržava svojstvo *nullability*, tj. mogućnost da se naglasi da li vrednost promenljive može biti nula ili ne. Ovim se pomaže u izbegavnju nastajanja poznate *NullPointerException* greške [5]. Kako bismo dozvolili da promenljiva sadrži nula vrednost, moramo to izričito naglasiti, tj. promenljivu definisati kao *nullable*. Ovo se postiže dodavanjem upitnika pri definisanju tipa promenljive, kao što se može videti u narednom primeru.

```
var a: String = "abc" # Ne-anulirajuca promenljiva
a = null              # problem pri kompilaciji

var b: String? = "abc" # b je anulirajuca promenljiva
b = null               # ok

val l = a.length
val l = b?length       # b je anulirajuca promenljiva, mora
                       # da se koristiti upitnik pri pozivu
```

Petlje

Osnovne petlje u Kotlinu su *for*, *while* i *do-while* petlje. Petlje u Kotlinu funkcionišu isto kao i u Javi. U narednom primeru je prikazana sintaksa Kotlin programskom jeziku za navedene petlje, kao i primer korišćenja naredbi skoka i rad sa labelama kod *for* petlje.

```

```

[5] Više o ovom tipu greške može se pročitati ovde: `https://docs.oracle.com/javase/7/docs/api/java/lang/NullPointerException.html`

```
outerLoop@ for (i in 1..3) {
    innerLoop@ for (j in 1..3) {
        println("i = $i and j = $j")
        if (i == 2){
            #skok na labelu 'outerLoop'
            break@outerLoop
        }
    }
}

while (x > 0) {
    x--
}

do {
    val y = getNextData()
} while (y != null)
```

Naredbe grananja

Naredbe grananja koje postoje u Kotlinu su *if* i *when* naredbe. Novina u Kotlinu je da se if ponaša kao izraz, tj. ima povratnu vrednost. Iz ovog razgloga if se može koristiti i kao zamena za standardni ternarni operator (:?).

```
if (obj is String) {
    print(obj.length)
}

when (x) {
    is Int -> print(x + 1)
    is String -> print(x.length + 1)
    is IntArray -> print(x.sum())
}

val a = 3
val b = 4
val max = if (a > b) a else b
```

Funkcija proširenja

Još jedan interesantan koncept kod Kotlina je mogućnost pravljenja funkcija proširenja (engl. *extension function*). Funkcije proširenja omogućavaju da se proširi

klasa ili interfejs sa novom funkcijom na lak način, dodavanjem funkcija bilo kojoj klasi bez formalnosti stvaranja izvedene klase. U narednom primeru se može videti kako funkcija proširenja izgleda[6].

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] # 'this' se odnosi na listu
    this[index1] = this[index2]
    this[index2] = tmp
}

val list = mutableListOf(1, 2, 3)
list.swap(0, 2) # vrsi promenu podataka na zadatim indeksima
```

Klasa podataka

Veoma česta upotreba klasa u programskim jezicima jeste kako bi se definisao određeni tip podataka. Iz ovog razloga Kotlin nudi posebnu vrstu klase nazvanu *klasa podataka* (engl. *data class*). Kako bi klasa podataka bila dobro definisana, potrebno je da primarni konstruktor ima bar jedan argument, kao i da argumenti primarnog konstruktora budu označeni sa val ili var. Klase podataka ne mogu biti apstraktne, otvorene, zaključane (engl. *sealed*) ili unutrašnje (engl. *inner*) klase.

Za sve deklarisane argumente u primarnom konstruktoru u klasi podataka kompajler će generisati funkcije za proveru jednakosti *equals()*, izračunavanje heš koda *hashCode()*, funkciju za ispis polja kao niske *toString()*, funkciju za kopiranje *copy()*, kao i funkciju za pristup svakom parametru *componentN()* gde N predstavlja redni broj argumenta. U primeru koji sledi, ilustrovana je razliku između sintakse Kotlin klase podataka i sintakse iste klase u Java programskom jeziku. Prvo je prikazan primer klase u Kotlin programskom jeziku.

```
data class Movie(var name: String, var rating: Float)
```

U nastavku se vidi primer iste klase u Java programskom jeziku.

```
public class Movie {

    private String name;
    private float rating;

    public Movie(String name, float rating) {
        this.name = name;
```

[6]Primer je preuzet sa sajta `https://kotlinlang.org/docs/extensions.html`

```
8         this.studio = studio;
9         this.rating = rating;
10    }
11
12    public String getName() {
13        return name;
14    }
15
16    public void setName(String name) {
17        this.name = name;
18    }
19
20    public float getRating() {
21        return rating;
22    }
23
24    public void setRating(float rating) {
25        this.rating = rating;
26    }
27
28    @Override
29    public int hashCode() {
30        final int prime = 31;
31        int result = 1;
32
33        result = prime * result + ((name == null) ? 0 : name.
    hashCode());
34        result = prime * result + Float.floatToIntBits(rating);
35
36        return result;
37    }
38
39    @Override
40    public boolean equals(Object obj) {
41        if (this == obj)
42            return true;
43
44        if (obj == null)
45            return false;
46        ....
47
48        return true;
```

```
49      }
50
51      @Override
52      public String toString() {
53          return "Movie [name=" + name + ", rating=" + rating + "]";
54      }
55  }
```

Može se primetiti koliko je konciznija sintaksa, dok, naravno, funkcionalnost ostaje ista.

Uključivanje jezika Kotlin u projekat

Kotlin programski jezik je uključen u projekat preko *maven* artifakta u glavnoj POM datoteci[7].

```
1  <properties>
2      <kotlin.version>1.7.0</kotlin.version>
3  </properties>
```

4.2 Aplikacija za razmenu recenzija

U ovom poglavju je detaljno predstavljena veb aplikaciju koja je razvijena u cilju ilustracije opisanih tehnologija, a rezultat je rada na ovoj tezi. Pokazana je arhitektura aplikacije na visokom nivou, razvojno okruženje koje je korišćeno, funkcionalnosti koje aplikacija nudi, upoznati se sa podacima, načinom unosa podataka, a obljašnjena je i implementacija određenih konkretnih delova aplikacije.

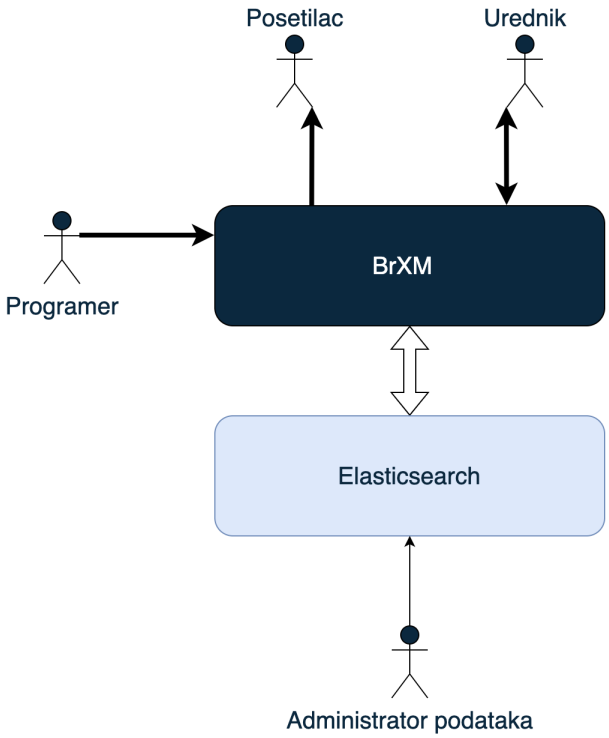
Razvojno okruženje

Veb aplikacije je razvijena na *macOS Catallina* operativnom sistemu[8]. Integrisano razvojno okruženje koje je korišćeno za izradu aplikacije je IntelliJ IDEA[9]. Kako bi aplikacija mogla da se pokrene potrebno je da su na lokalnoj mašini instalirani *Maven 3+* i *Java 8+*.

[7] Više informacija se može naći na sajtu: `https://kotlinlang.org/docs/maven.html`
[8] `https://apps.apple.com/us/app/macos-catalina/id1466841314?mt=12`
[9] Oficijalna stranica za IntelliJ IDEA: `https://www.jetbrains.com/idea/`

Opis aplikacije

Na slici 4.2 se vidi da se aplikacija sastoji od BrXM sistema kojem direktno pristupaju posetilac, urednik i programer. Sa druge strane se nalazi Elasticsearch koji će direktno komunicirati sa BrXM sistemom i biti održavan od strane administratora podataka. Posmatrajući podelu na prethodno navedene korisnike, prikazane su funkcionalnosti veb aplikacije. *Programer* se bavi razvijanjem novih komponenti i održavanjem sistema na mreži. *Urednik* je odgovoran za sadržaj na sajtu, može da kreira nove vesti, bira vesti koje će se prikazati na sajtu, pravi nove stranice i kontroliše pristup samom sistemu. S obzirom na način korišćenja baze podataka, potreban je podrška tehničkog lica pri unosu podataka i time se bavi *administrator podataka*. *Posetilac* će moći da vrši pretragu filmova, da pročita više detalja o odabranom filmu, da čita vesti koje održavaju urednici i da ostavi recenziju.



Slika 4.2: Prikaz aplikacije na visokom nivou

Podaci

Trenutno najpopularnija baza filmova je IMDb[10] baza, koja sadrži više od 7.5 miliona naslova. Neki od podataka koje IMDb baza sadrži su: opis radnje filma, ocena filma, žanr, spisak glumaca i još dosta drugih podataka.

Za prikaz tehnologija nisu potrebni svi podaci koje baza sadrži, kao ni svih 7.5 miliona naslova. Korišćena IMDb bazu koja se nalazi na Kaggle[11] sajtu koja sadrži naslove sa najvišim ocenama. Ova baza, iako manja od originala, ipak daje dovoljno veliki broj (oko 90.000) naslova iz IMDb baze. Baza je dostupna u *csv* (engl. *comma-separated values*) formatu.

Tabela 4.1: Polja u bazi filmova

Naziv polja u bazi	Opis polja
imdb_title_id	ID iz IMDb baze filmova
title	Ime filma
original_title	Originalno ime filma
description	Opis dešavanja
avg_vote	Srednja ocena
reviews	Recenzije
year	Godina obljavljivanja filma
date published	Datum objavljivanja filma
genre	Filmski žanr kome film pripada
duration	Trajanje films (u minutima)
country	Zemlja porekla filma
language	Jezik filma
director	Ime režisera
writer	Ime pisca
production company	Produkcijska kuća
actors	Glumci
votes	Broj ocena
budget	Budžet
usa gross incom	Novčana zarada u Americi
worlwide gross income	Novčana zarada u svetu
metascore	Meta ocena
reviews from users	Broj ocena od korisnika
reviews from critics	Broj ocena od strane kritičara
reviews	Recenzije

[10] `https://www.imdb.com/?ref_=nv_home`
[11] `https://www.kaggle.com/`

Većina polja koja se nalaze u bazi (tabela 4.2) će biti korišćena na stranici za detaljniji prikaz filma, dok je za pretragu najbitnije polje *original_title*. Pored polja koje dolaze iz ove baze, dodaje se i polje *reviews* koje služi za upis recenzija od strane posetilaca.

Unos podataka

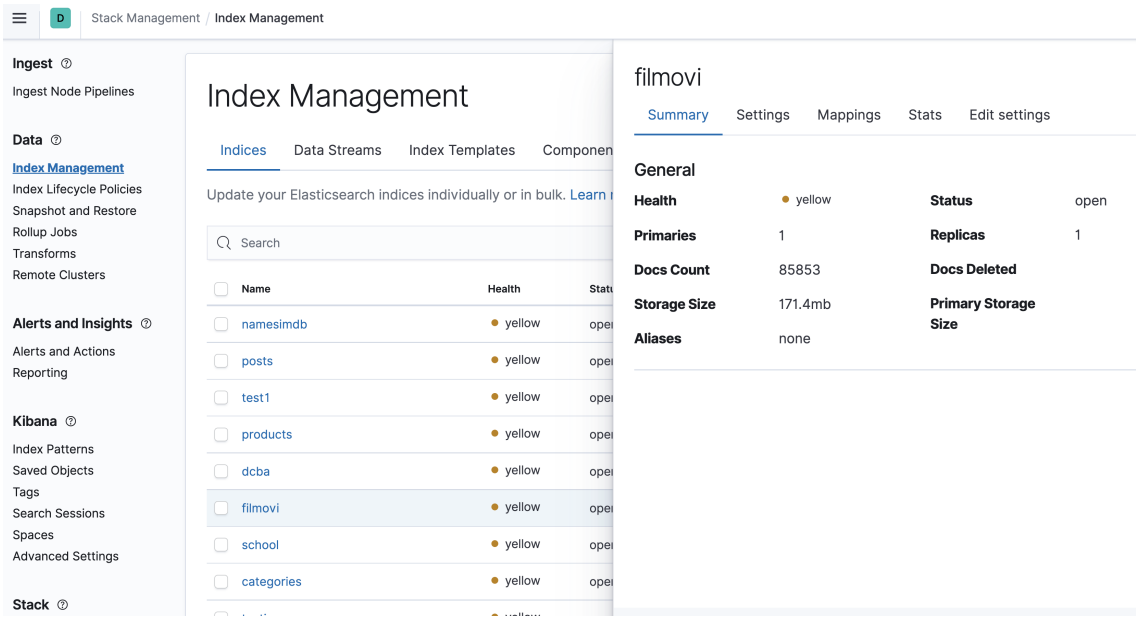
Odabrano je da se unos podataka u Elasticsearch deo aplikacije radi pomoću Logstash aplikacije. U nastavku je prikazan deo Logstash dokumenta za podešavanje koji je korišćen za unos podataka.

```
input {
  file {
    path => "/Users/van1bg/Desktop/ES/archive/IMDBmovies.csv"
    start_position => "beginning"
    sincedb_path => "/tmp/mysincedbfile"
  }
}

filter {
  csv {
    separator => ","
    skip_header => "true"
    columns => ["imdb_title_id",
      "title",
      "original_title",
        ...
      "reviews_from_users",
      "reviews_from_critics"
    ]
  }

  mutate {convert => {"year" => "integer"
                      "avg_vote" => "float"
                      "votes" => "integer"}
}
}
}

output {
  elasticsearch {
    hosts => "http://localhost:9200"
```

```
32      index => "filmovi"
33    }
34
35    stdout {}
36  }
```

U delu za podešavanje ulaza postavljena je putanja dokumenta sa podacima, da čitanje podataka ide od početka dokumenta, kao i polje *sincedb_path*. Polje *sincedb_path* definiše putanju do dokumenta u kom se pamti dokle je unos podataka stigao, u slučaju naglog prekida izvršavanja Logstash aplikacije, zapamćeno je odakle treba da se nastavi unos podataka. Zatim je definisan tip dokumenta, naveden separator, kao i niz kolona koje će posle transformacije postati polja na Elasticsearch serveru. Eksplicitno se pomoću *mutate* dela dokumenta definiše tip za polja *year*, *avg_vote* i *votes*. Definisana je i lokacija servera na koji se šalju podaci i naziv indeksa koji treba da se popuni podacima.

Pošto su pomoću Logstash aplikacije uneti podatci u Elasticsearch server, kao način za potvrdu uspešnosti unosa podataka može se koristiti aplikacija Kibana. Provera se može izvršiti tako što se otvori deo Kibana aplikacije koji izlistava sve indekse koje instanca Elasticsearch servera ima i proveri se da li se u toj listi nalazi željeni indeks, kao i da li ima podataka (slika 4.3).



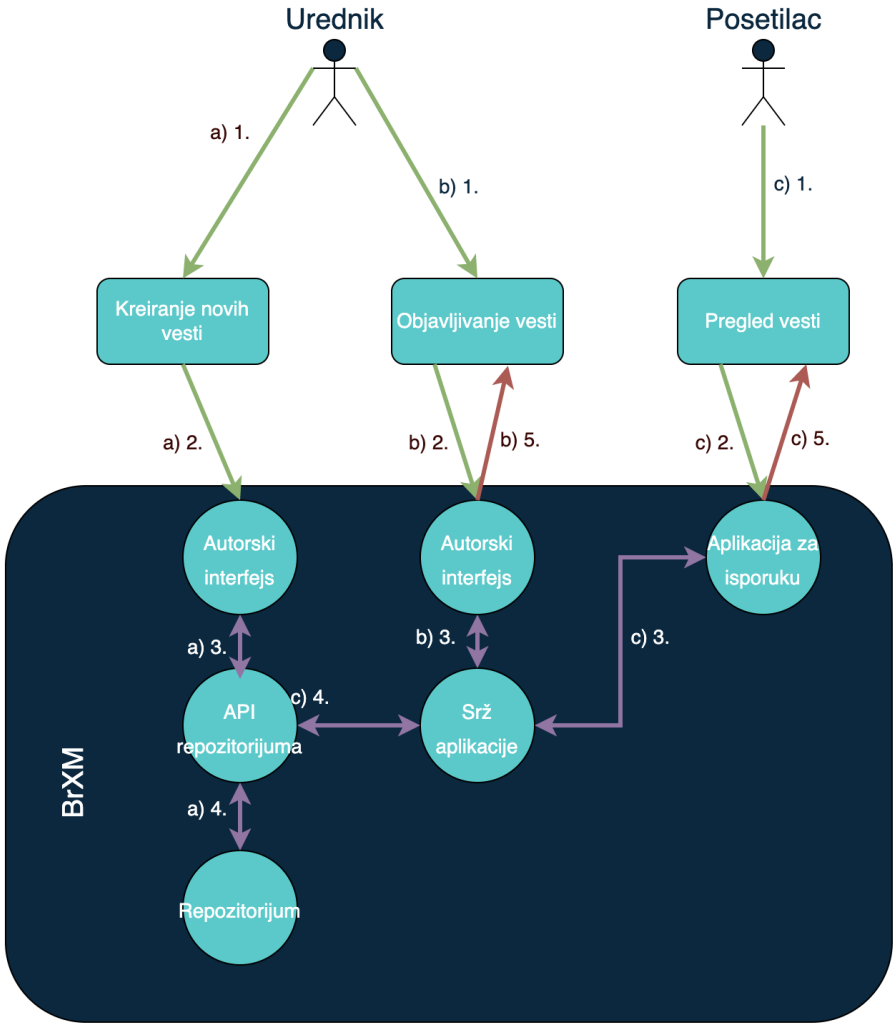
Slika 4.3: Provera podataka preko Kibane

Na slici 4.3 se vidi da se indeks *filmovi* uspešno kreirao i da ima preko 85000

dokumenata.

Održavanje sadržaja od strane autora

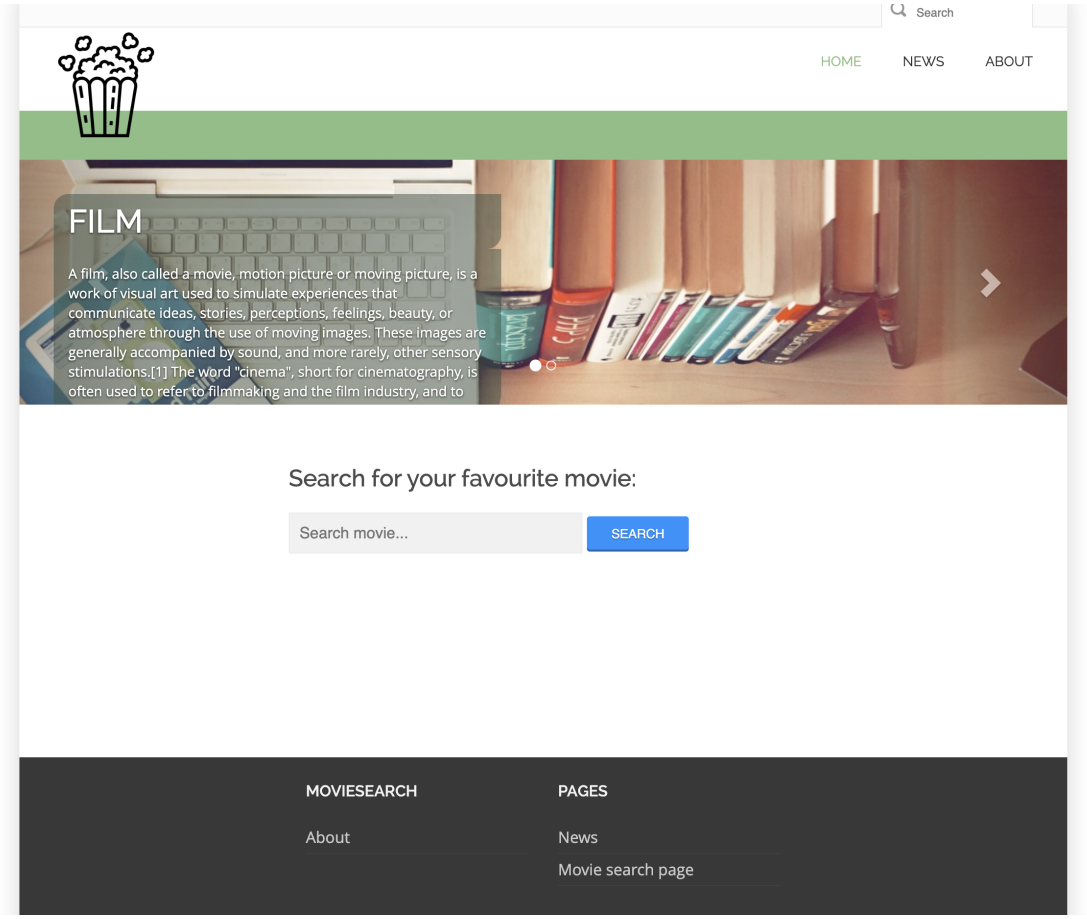
Jedna od prednosti korišćenja sistema za održavanje sadržaja je mogućnost da tehnički neobučena lica lako menjaju sadržaj. Na slici 4.4 se vidi sa kojim delom sistema urednik komunicira pri pravljenju novih vesti (obeleženo sa a), kao i objavljivanje vesti na sajtu (obeleženo sa b). Primetno je da urednik komunicira isključivo sa BrXm sistemom, nema potrebe za pristupom Elasticsearch serveru.



Slika 4.4: Prikaz aplikacije na visokom nivou

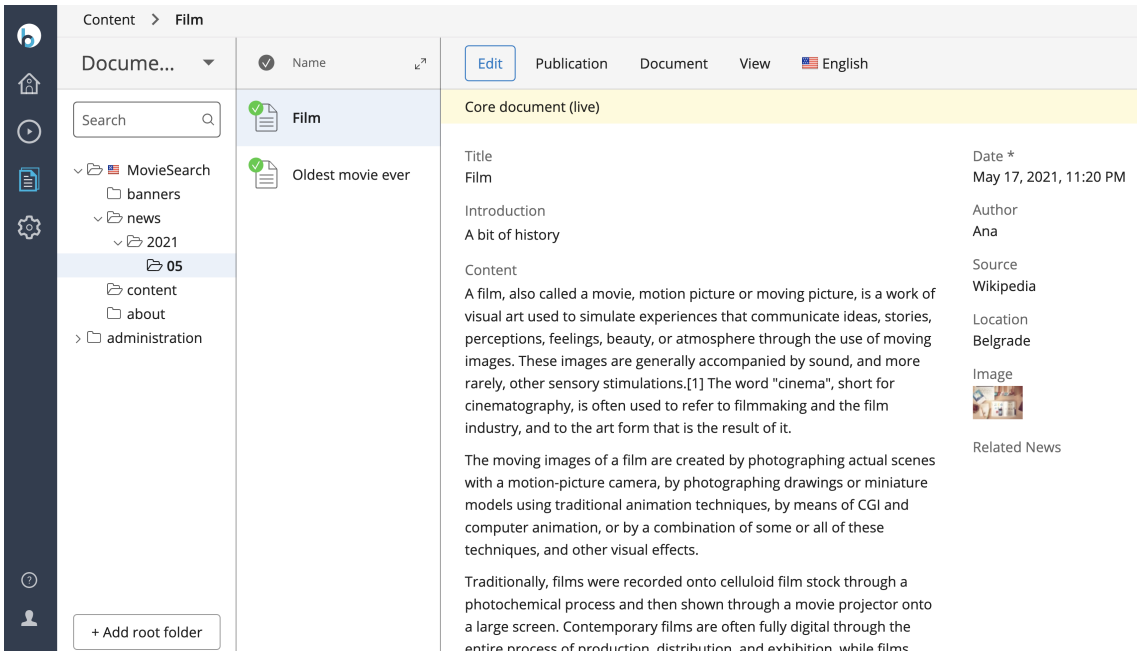
Izgled početne strane i same komponente se može videti na slici 4.5. Početna strana veb aplikacije u centralnom delu stranice ima prikaz komponente koja prikazuje

vesti na sajtu, čiji je sadržaj održavan od strane urednika. Podaci koji se prikazuju su naslov, deo teksta i slika iz odabranog tipa dokumenta za vesti. Komponenta koja je iskorišćena za ovu funkcionalnost je među već ponuđenim komponentama od strane BrXM, njen naziv je *Carousel*, a preuzeta je iz *Essential* aplikacije.



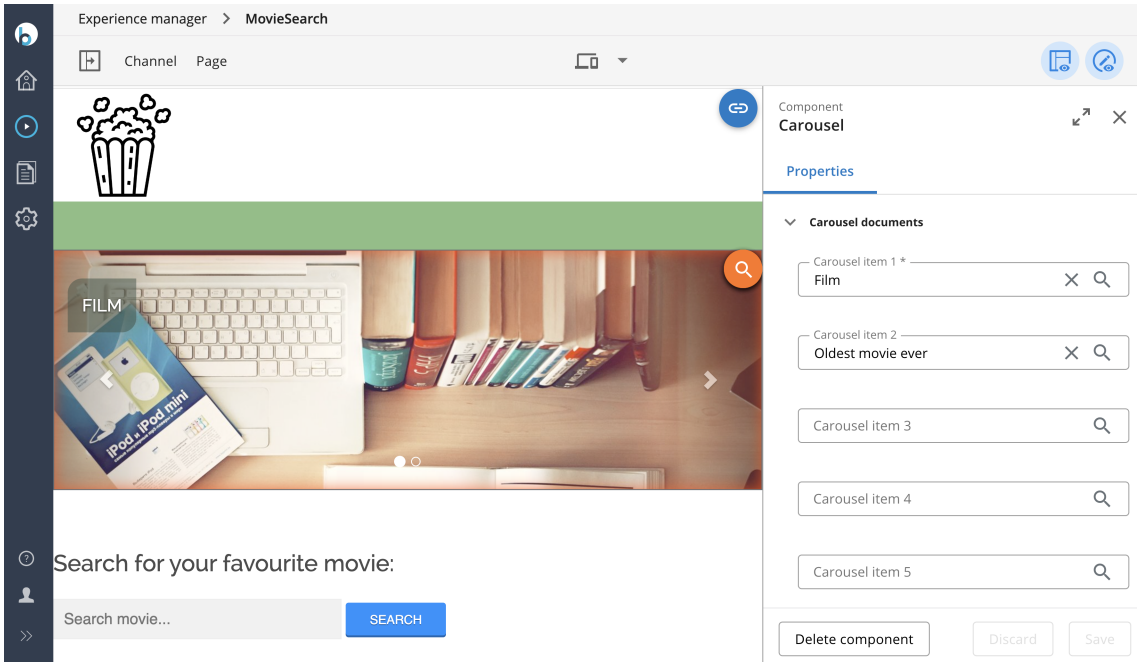
Slika 4.5: Početna strana veb aplikacije

Kako bi se dokument našao na početnoj stranici potrebno je da urednik napravi dokument određenog tipa, kao i da odabere da se dokument prikazuje u komponenti pomoću veb aplikacije autorskog interfejsa. Na slici 4.6 se vidi gde se u veb aplikaciji autorskog interfejsa nalazi direktorijum koji sadrži sve vesti. Ovaj direktorijum ima naziv *News* i u njemu urednik pravi novi dokument tipa *News item* koji je potrebno popuniti željenim tekstom.



Slika 4.6: Direktorijum sa vestima

Potrebno je da se urednik pozicionira u *Experience Manager* delu sistema, otvori početnu stranu sajta i izmeniti trenutni deo sa pregledom odabranih vesti, tj. *Carousel* dodatak.

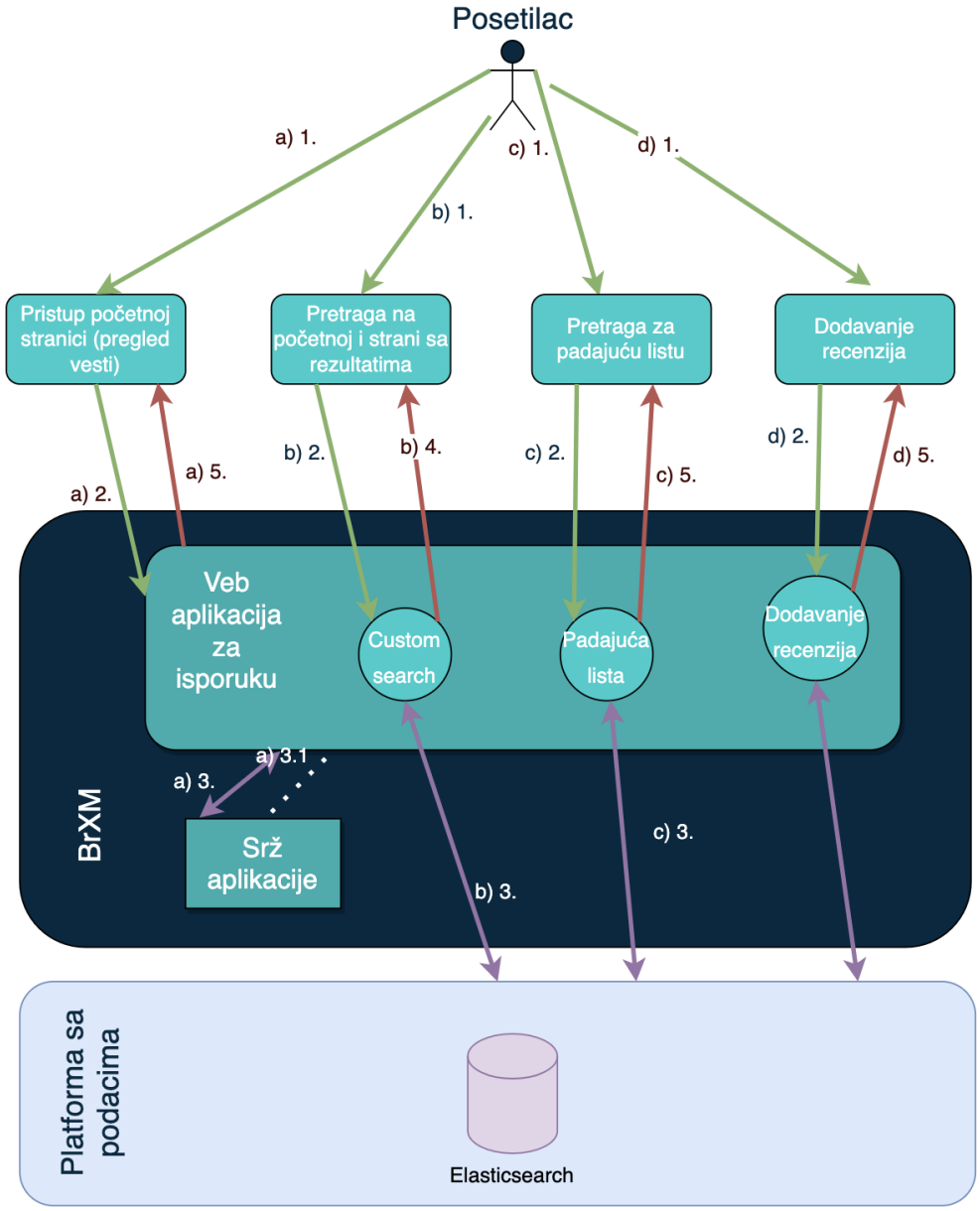


Slika 4.7: Izmena početne strane

Preko desnog panela, koji se vidi na slici 4.7, mogu se pretražiti sva dokumenta u sistemu i mogu da se odaberu ona koje će se prikazivati na početnoj strani sajta. Takođe postoji i mogućnost dodatnih podešavanja (na primer dužina prikaza slike). Sve vesti koje se nalaze u sistemu se mogu videti kada se klikne u zaglavlju sajta u navigacionom meniju na polje *News*.

Implementacija pretraga i recenzija

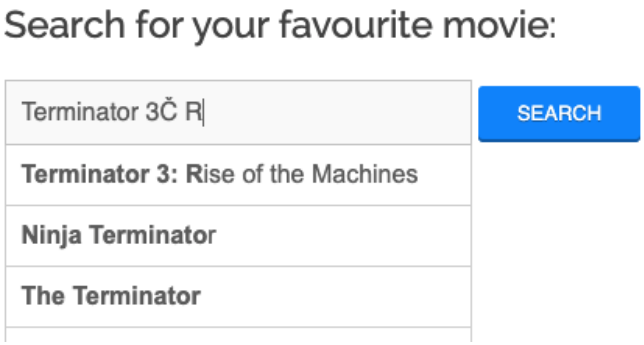
Na dijagramu 4.8 se vidi da posetilac sajta preko BrXM sistema komunicira sa Elasticsearch serverom. Slučajevi upotrebe koji se vide su: čitanja vesti (obeleženo sa *a*), pretrage filmova (obeleženo sa *b*), automatsko dobijanje sugestija u padajućoj listi (obeleženo sa *c*), dodavanje recenzija (obeleženo sa *d*). Kasnije je detaljnije obljašnjena implementacija svih ovih funkcionalnosti.



Slika 4.8: Prikaz interakcije posetioca na visokom nivou

Prikaz rezultata u padajućoj listi

Na početnoj strani (slika 4.5), u centru stranice nalazi se polje za pretragu. Kada korisnik započne unos u polju za pretragu, automatski će se prikazati predlozi za pretragu (slika 4.9).



Slika 4.9: Prikaz rezultata u padajućoj listi

Iz liste sa ponuđenim nazivima, posetilac nađe željeni tekst, odabere tekst klikom i zatim pritiskom na dugme *Search* se izvršava pretraga naslova filmova. Elasticsearch dozvoljava da se jednostavno pristupi podacima korišćenjem programskog interfejsa, kao i da se lako izvrše određeni upit. Funkcionalnost polja za pretragu unapređena je tako da vraća sugestije u realnom vremenu uz pomoć *fetch()* funkcije u JavaScript-u. Fetch API[12] nudi JavaScript interfejs za pristup i manipulisanje delovima HTTP zahteva i odgovora. Primer se nalazi u nastavku.

```
1  var input = $("#myInput").val();
2              input = input.concat("*0.8")
3              var url = 'http://localhost:9200/filmovi/_search?q=
       original_title:';
4              url = url+input
5              var arr = new Array ();
6
7              fetch(url)
8                  .then((response) => {
9                      return response.json();
10                 }).then((myJson) => {
11                 //Kod za formatiranje JSON odgovora
12                 ...
13     });
```

U datom isečku koda, se vidi da se najpre pravi URL adresa sa upitom nad određenim poljem u bazi i zatim procesuira odgovor. Prvo se pristupa samom serveru *localhost:9200*, zatim indeksu *filmovi*, nad indeksom se izvršava pretraga koja je tipa upit (engl. *query*), a zatim se dodaje kao parametar vrednost koju je korisnik uneo

[12] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

u polje za pretagu. Zahtev sa upitom se serveru prosleđuje pomoću fetch() funkcije, koja nakon što prihvati rezultat upita, preda ga JavaScript kodu koji formatira dobijeni JSON odgovor i prikazuje ga korisniku.

Pretraga se bazira na Lucene sintaksi za upite [9].

Zanimljivi parametri, koji su korišćeni u ovoj aplikaciji su *wildcard* i *fuzzy*. Oni se koriste kada u rezultatu pretrage treba da budu vraćena i polja sa sličnim vrednostima, a ne samo polja koja se u potpunosti poklapaju sa prosleđenim tekstom za pretragu.

Wildcard parametar se koristi dodavanjem simbola *?* (ili ***) na nisku i daje mogućnost da se na toj poziciji nađe bilo koji karakter, dok simbol *** dodat na nisku daje mogućnost da se na toj poziciji nađe više proizvoljnih karaktera.

Fuzzy parametar vraća rezultate koji sadrže reč koja se od unete reči razlikuje najviše za neko određeno rastojanje. U računarstvu, *Levenštajnovo rastojanje* je metrika za niske, koja je jedan od načina da se odredi rastojanje uređivanja (engl. *edit distance*) dve niske. Levenštajnovo rastojanje dve niske je određeno minimalnim brojem operacija neophodnim da se jedna niska transformiše u drugu, a operacije su umetanje, brisanje ili zamena jednog karaktera drugim [8]. Dodavanjem broja na nisku upita definiše se *fuzzy* parametar. Ako se ne definiše dodatno taj broj, podrazumevani broj je 0.5.

Veb aplikacija koristi mešavinu ova dva parametra, što se može videti u liniji *input = input.concat(„*0.8”)*, u kojoj se na unos posetioca dodaje **0.8*. Na slici 4.9 se može videti kako se ponaša i koje rezultate daje pretraga – iako imamo slovnu grešku dobijaju se slični naslove kao odgovor.

Glavna pretraga

Na početnoj strani sajta, kada se upiše željeni pojam za pretragu i klikne na dugme, izvršiće se upit nad bazom, a zatim će se učitati nova strana sa spiskom rezultata pretrage. Kôd koji se poziva prilikom ove pretrage za komunikaciju sa Elasticsearch serverom je pisan u Kotlin programskom jeziku uz pomoć programerskog interfejsa za Javu koji nudi Elastic kompanija.

Elastisearch je delom postao popularan među programerima zbog toga što nudi veliki broj gotovih klijentskih biblioteka za rad sa raznim programskim jezicima[13]. U projektu je korišćen zvanični Elasticsearch REST klijent za Javu, koji se naziva *Java*

[13]Više informacija se može naći na sajtu: `https://www.elastic.co/guide/en/elasticsearch/client/index.html`

High Level REST Client. Glavni cilj ove klijentske biblioteke je izlaganje specifičnih metoda programerskog interfejsa koje prihvataju objekte kao argument zahteva i vraćaju objekte kao odgovor. Ovim se izbegava rad na niskom nivou [7].

Java High Level REST Client radi nad *Java Low-Level REST* klijentom. Low-level Java REST client se oslanja na Apache Http Async klijent[14] za slanje HTTP zahteva, a korisnicima ostavlja da sami transformišu zahteve i odgovore u željene objekte (engl. *marshalling*).

U nastavku se može videti primer dela klase za pretragi filmova i kako se ova klijentska biblioteka koristi u razvijenoj aplikaciji.

```
1    fun searchMovie(name:String?): List<MovieDescription>{
2        val client = RestHighLevelClient(
3                RestClient.builder(
4                        HttpHost("localhost", 9200, "http")))
5
6        val searchRequest = SearchRequest("movies")
7        val searchSourceBuilder = SearchSourceBuilder()
8        searchSourceBuilder.size(10)
9        searchSourceBuilder.query(QueryBuilders.matchQuery("
   original_title", name+"*").fuzziness(0.8))
10       searchRequest.source(searchSourceBuilder)
11
12       val result = mutableListOf<MovieDescription>()
13
14       try {
15           val searchResponse = client.search(searchRequest,
   RequestOptions.DEFAULT)
16           val values = searchResponse.hits.hits
17           if (values.size > 0) {
18               for (s in values) {
19                   result.add(mapMovie(s.getSourceAsMap()))
20               }
21           } else {...}
22       } catch (e: IOException) {...}
23       ...
24   }
```

Potrebno je da se prvo uspostavi konekcija sa bazom. Pravi se *RestHighLevelClient* instancu koja se oslanja na metodu koja se nalazi u *RestClient* klasi. *RestClient* instanca se gradi pomoću klase *RestClientBuilder*, korišćenjem *RestCli-*

[14] https://hc.apache.org/httpcomponents-asyncclient-4.1.x/index.html

ent.builder(HttpHost...) metode. Jedini neophodan argument ove metode je jedan ili više adresa servera sa kojim će se vršiti komunikacija, koja se gradi kao instanca klase *HttpHost*[15] klase. *RestClient* instancu treba zatvoriti kada više nije potrebna, kako bi se resursi oslobodili. U protivnom će instanca postojati sve dok se i aplikacija izvršava.

Sledeći korak je konstrukcija klase *SearchRequest* koja se koristi za svaku operaciju vezanu za pretragu dokumenata. Argument koji je prosleđen konstruktoru je naziv indeksa. Ako bi poziv bio bez argumenata, upit bi bio izvršen nad svim indeksima u bazi.

Zatim je potrebno da se kreira instanca *SearchSourceBuilder* klase, koja služi za kreiranje objekta sa potrebnim informacijama za konstrukciju pretrage. Postavljena je vrednosti za željenu količinu podataka u rezultatu na 10, pomoću koda *searchSourceBuilder.size(10)*.

```
searchSourceBuilder.query(QueryBuilders.matchQuery("original_title
    ",name+"*").fuzziness(0.8))
```

Upit se konstruiše pomoću klase *QueryBuilders*[16] i metode *matchQuery(„original_title", name+„*")*. U metodi *matchQuery*, prvi argument predstavlja polje koje se pretražuje, a drugi argument tekst sa kojim se upoređuje. Dodata je i vrednost za *fuzzy* parametar na 0.8 kako bi pretraga bila ista kao i u padajućoj listi.

Zatim se postavlja upit u *searchRequest* objekat pomoću metode *source()* i poziva se izvršavanje upita za pretragu nad *client* objektom.

```
searchRequest.source(searchSourceBuilder)
val searchResponse = client.search(searchRequest, RequestOptions.
    DEFAULT)
```

U nastavku rada se vidi struktura odgovora koju ovakav zahtev vraća.

```
{
   "took":8,
   "timed_out":false,
   "_shards":{
      "total":1,
      "successful":1,
      "skipped":0,
      "failed":0
```

[15]Više na sajtu: https://hc.apache.org/httpclient-legacy/apidocs/org/apache/commons/httpclient/HttpHost.html

[16]Više na sajtu: https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high-query-builders.html

```
9     },
10    "hits":{
11       "total":{
12          "value":1
13     },
14       "max_score":10.954961,
15       "hits":[
16          {
17             "_index":"movies",
18             "_type":"_doc",
19             "_id":"mzetZXkBHr14MnNCsUgg",
20             "_score":10.954961,
21             "_source":{
22                "date_published":"1985-01-11",
23                "reviews_from_critics":"235.0",
24                "imdb_title_id":"tt0088247",
25                 ...
26                "country":"UK, USA"
27             }
28          }
29       ]
30    }
31 }
```

Dobijeni odgovor je u JSON formatu i u početnom delu ima dodatne informacije o dužini izvršavanja upita i stanju servera, dok u *hits* delu odgovora se nalaze rezultati pretrage, tj. dokumenti iz indeksa, takođe u JSON formatu.

Implementacija recenzija

U prethodnom delu rada, u glavi o Elasticsearch serveru je već pomenuto da se on može koristiti i kao glavna baza, iako to nije preporučljivo. Problem koji se javlja prilikom korišćenja Elasticsearch servera, kao glavne baze, je baš u slučaju upisa novih vrednosti (ažuriranja dokumenata). Problem nastaje zbog potrebe za analizom i indeksiranjem novih podataka, tj. reindeksiranjem podataka (engl. *re-indexing*). Zato je operacija ažuriranja skuplja nego u drugim bazama[17]. Iako nije najbolje rešenje, odlučeno je da se u ovoj aplikaciji ne koristi dodatna baza i da se radi direktna izmena nad podacima u Elasticsearch serveru.

[17]Više informacija može se naći na sajtu: `https://mendab1e.github.io/dev/2017/04/22/elastic-search-is-not-db.html`

Java klijentska biblioteka, pored već korišćenih klasa za pretragu, sadrži i klase koje se koriste za izmenu dokumenata na serveru. *UpdateByQueryRequest* klasa se koristi za izmenu nad dokumentima koji odgovaraju postavljenom upitu. Ako nije eksplicitno naglašeno nad kojim dokumentom se vrši izmena, izmena će se izvršiti nad svim dokumentima u indeksu. Ovakvo ponašanje može biti od pomoći u slučaju potrebe za izmenom mapiranja.

Kada Elasticsearch dobije zahtev za izmenu, on prvo pravi snimak indeksa pre izvršavanja upita i pamti verziju dokumenta. Zatim izvršava potrebne izmene i u slučaju da je verzija po završenom procesu još uvek ista, izmena se snima i verzija se povećava. Ukoliko je pak došlo do promene verzije, izmene se ne snimaju, javlja se greška i prekida izvršavanje. Može se izbeći prekid izvršavanja ovog zahteva, tako što se postavi parametar za konflikte na vrednost *proceed*. Ukoliko se postavi ovaj parametar, svi dokumenti koji ne vraćaju grešku će biti izmenjeni.

Kada ima potreba za izvršavanjem nekih naprednijih izmena (ili pretraga) nad dokumentima u Elastricsearch serveru, trebaju da se koriste skripte. Podrazumevan skriptni jezik za Elasticsearch server se zove *Painless* [10]. *Painless* je efikasan (u poređenju sa alternativama se znatno brže izvršava), siguran skriptni jezik dizajniran specijalno za Elasticsearch (time se dibija najviše fleksibilnosti u radu) i sintaksa je slična Java programskom jeziku. Još neki često korišćeni skriptni jezici su: *expression*[18] i *mustache*[19].

Sledeći primer prikazuje kôd funkcije za upis recenzija u bazu iz veb aplikacije, funkcija koristi *Painless* skriptu.

```
1         val client = RestHighLevelClient(
2                 RestClient.builder(
3                         HttpHost("localhost", 9200, "http")))
4
5         val updateByQueryRequest = UpdateByQueryRequest("movies")
6         updateByQueryRequest.setConflicts("proceed")
7
8         updateByQueryRequest.setQuery(TermQueryBuilder("
    imdb_title_id", id))
9
10        val updateMap: MutableMap<String, Any> = HashMap()
11        updateMap["review"] = review
```

[18] https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-scripting-expression.html

[19] https://www.elastic.co/guide/en/elasticsearch/reference/current/search-template.html

```
12          updateMap["user"] = user
13
14          val script = Script(ScriptType.INLINE, "painless", "ctx.
    _source.review.add(params)", updateMap)
15          updateByQueryRequest.script = script
16
17          try {
18              val bulkResponse: BulkByScrollResponse = client.
    updateByQuery(updateByQueryRequest, RequestOptions.DEFAULT)
19              val totalDocs = bulkResponse.total
20              println("updated response id: $totalDocs")
21          } catch (e: IOException) {
22              e.printStackTrace()
23          }
```

Prvo se vrši povezivanje sa bazom. Zatim se pravi instanca klase *UpdateByQueryRequest* koja je takođe deo Java High Level REST klijentske biblioteke i koristi se za izmenu dokumenata. Prosleđuje joj se jedan argument – naziv indeksa čiji dokument treba izmeniti. Takođe se može primetiti i da je vrednost za *conflicts* podešena na *proceed*. Zatim se podešava upit, koji se gradi pomoću *TermQueryBuilder* klase.

```
1 updateByQueryRequest.setQuery(TermQueryBuilder("imdb_title_id", id)
    )
```

Prvi argument govori koje polje se koristi za pretragu dokumenta unutar indeksa, dok je drugo polje sama vrednost koja se pretražuje. Vrednost koja se traži se dobija iz adrese sajta i prosleđuje kao parametar ovoj klasi, pri pozivu funkcije. Zatim je definisan dokument koji treba da se izmeni, pravi se skripta koja će izvršiti željene izmene u bazi. Definisana je heš mapa, sa vrednostima iz forme koju je posetilac popunio. U skripti se referiše na heš mapu i šalje je kao objekat koji će se naći u nizu recenzija u dokumentu.

```
1 client.updateByQuery(request, RequestOptions.DEFAULT);
```

Zatim je pokrenuto izvršavanje. Ovako pokrenut upit je sinhronog tipa, tj. klijent će sačekati da se upit izvrši pre nego se nastavi izvršavanje koda. Skripta pristupa dokumentu i polju *review* dodaje novi objekat. Uzrok prekida ovog upita, sem uspešnog izvršavanja, može biti neuspeh da parsira odgovor, ako upit predugo traje (engl. *timeout*). Kada server vraća grešku *4xx* ili *5xx*, dobijen je generički *ElasticsearchException*.

Glava 5

Zaključak

U ovom radu predstavljeno je idejno rešenje za izradu veb aplikacije za pretragu i razmenu IMDB recenzija koja ispunjava sve potrebe modernih veb aplikacija. Istaknuti su glavni izazovi koji se javljaju pri razvoju veb aplikacija i kako ti problemi mogu biti rešeni korišćenjem BrXM sistema i Elasticsearch servera. U sklopu ovog rada razvijena je veb aplikacija otvorenog koda na programskom jeziku Kotlin koja koristi BrXM i Elasticsearch kao bazu. Veb aplikacija je dostupna na adresi `https://github.com/AnaVuksic/master`.

Može se zaključiti da BrXM nudi dosta pogodnosti. Osim intuitivnog autorskog interfejsa za tehnički neobučena lica, sadrži i odvojene veb aplikacije pomoću kojih je moguće lako dodavati postojeće funkcionalnosti, kao i napraviti nove. Dobijamo veoma dobru i stabilnu osnovu koja se može lako proširiti. Sa druge strane, Elasticsearch server daje efikasnost brze pretrage, kao i mogućnost analize podataka. U malo koraka se dobije okruženje koje daje dosta mogućnosti.

Loša strana ovako razvijene veb aplikacije može biti prevelika kompleksnost sistema. Takođe, problem može biti korišćenje Elasticsearch baze za izmenu podataka, što je jedna od čestih operacija. Kako bi se osigurala i brzu pretragu i dobre preformanse kod operacije ažuriranja, preporučuje se korišćenje i relacione baze uz Elasticsearch.

Iz pomenutih razloga, tehnologije opisane u ovom radu ne moraju predstavljati idealno rešenje za razvoj prostijih veb aplikacija. Međutim, ukoliko imamo potrebe za kompleksnom veb aplikacijom, čiji sadržaj treba da održava veći tim ljudi, a uz to je potrebna i napredna i efikasna pretraga, onda prikazana kombinacija tehnologija može predstavljati veoma dobro rešenje.

Bibliografija

[1] In *Content Management Bible*. John Wiley & Sons, 2005.

[2] Official Bloomreach Documentation. https://documentation.bloomreach.com/14/library/architecture/brxm-architecture.html, 2021.

[3] Apache FreeMarker. https://freemarker.apache.org/, 2022.

[4] Apache Wicket official website. https://wicket.apache.org/, 2022.

[5] Bloomreach Experience Manager Architecture. https://www.bloomreach.com/en/about/our-story, 2022.

[6] Codehause Cargo. https://codehaus-cargo.github.io/cargo/Tomcat+9.x.html, 2022.

[7] Java High-level REST Client Elastic documentation. https://www.elastic.co/guide/en/elasticsearch/client/java-rest/master/java-rest-high.html, 2022.

[8] Levenshtein distance. https://xlinux.nist.gov/dads/HTML/Levenshtein.html, 2022.

[9] Lucene documentation. https://lucene.apache.org/, 2022.

[10] Painless documentation. https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-scripting-painless.html, 2022.

[11] Contentstack. The Ultimate Guide to CMS Vol. 1: Comparing the Architectures and Differences Between Headless CMS vs. Decoupled CMS vs. Traditional CMS. https://www.contentstack.com/blog/all-about-headless/traditional-vs-decoupled-vs-headless-cms/, 2020.

[12] Stefan Büttcher; Charles L. A. Clarke;Gordon V. Cormack. Information Retrieval, Implementing and Evaluating Search Engines. pages 33–45, 2010.

[13] Eric Enge; Stephan Spencer; Rand Fishkin; and Jessie C. Stricchiola. The Art of SEO. 2010.

[14] Kale Suhas B. Study of Web Content Management System. *IOSR Journal of Computer Engineering (IOSR-JCE)*, pages 79–82.

[15] Elizabeta Markuš Mitrinović. Arhitektura MVC aplikacije sa primerima u PHP-u. Niš: Univerzitet u Nišu Prirodno-matematički fakultet Departman za računarske nauke. 2016.

[16] MOZ. The Beginner's Guide to SEO. `https://moz.com/beginners-guide-to-seo`, 2021.

[17] Jakovljević Aleksandar; Serbić Marko; Veljković Marko; Vukadinović Selena. Kotlin kao programski jezik nove generacije. 2019.

Biografija autora

Ana Vuksić je rođena 12. septembra 1994. u Beogradu. Osnovnu školu je završila 2009. u Beogradu. Usled sklonosti ka prirodnonim naukama, upisala je Treću beogradsku gimnaziju, prirodno-matematički smer. U tom periodu postaje zainteresovana za programiranje. Po završetku srednje škole, 2013. godine, upisuje Matematički fakultet u Beogradu, smer Informatika. Osnovne studije završava 2016. godine i odlučuje da nastavi sa master studijama na istom fakultetu, smer Informatika. U tom periodu počinje da volontira u studentskoj organizaciji „Udruženje studenata tehnike Evrope" i ubrzo 2017. godine postaje Koordinator Informacionih tehnologija na nivou Evrope. Zbog rada u timu sa više nacija, u njoj se rađa želja za usavršavanjem u inostransvu i odlučuje da se prijavi za praksu van Srbije. Tako 2018. godine odlazi na praksu od godinu dana u Nemačku. Po završetku prakse vraća se u Beograd i nastavlja svoju karijeru u oblasti razvoja Veb aplikacija u rodnom gradu.