Matematički fakultet, Univerzitet u Beogradu

MASTER RAD

Efikasan i kvalitetan razvoj veb aplikacije za raspoređivanje i prikaz geo-prostornih tačaka u ravni

Mentor:
Prof. Dr. Vladimir Filipović

Kandidat:
Ljiljana Lazić

Beograd, 2017

Sadržaj

1. Uvod

U poslednje vreme, u praksi se sve češće zahteva, da se u što kraćem vremenskom roku, proizvede softver, koji bi odgovarao iskazanim zahtevima i potrebama. Softversko rešenje je potrebno osmisliti, dizajnirati i razviti tako da se omogući: lako dodavanje novih funkcionalnosti, jednostavno uklanjanje onih funkcionalnosti koje više nisu potrebne, unapređenje ili zamena tehnologija, jednostavna instalacija i održavanje projekta. Istovremeno, programski kôd treba da bude čitljiv, razumljiv i proširiv.

Zbog toga je veoma značajan izbor odgovarajućih tehnologija, koje omogućavaju korišćenje okvira i alata, a čiji razvoj se stalno usavršava. One pomažu da se postavi potrebno okruženje i konfiguracija, kako bi glavni fokus daljeg razvoja aplikacije, bio na implementaciji specifičnih zahteva.

U radu se razmatra kako se korišćenjem modernih tehnologija, alata i razvojnih okvira, može postići lakši i efikasniji razvoj složene veb aplikacije. U razmatranju se stavlja akcenat na korišćenje uzoraka dizajna i na omogućavanje razvoja „čistog“ programskog kôda, koji je čitljiv, lak za razmevanje i za održavanje, kao i za buduća proširenja.

Predstaviće se izrađena aplikacija shodno temi rada (u daljem tekstu „prototipska aplikacija“). Aplikacija je prototip kompleksne industrijske aplikacije, koja bi mogla da ima višenamensku upotrebu u raznim oblastima gde postoji potreba da se dobiju određene informacije o objektima, koji se mogu opisati tačkama u prostoru i pripadaju određenim prostornim segmentima. Tačke će biti predstavljene u 2D prostoru i raspoređene u grupe po odgovarajućim kriterijumima, korišćenjem kvadratnog stabla. Kompletirana veb aplikacija bi se mogla koristiti, na primer, da na zahtev korisnika, za odgovarajuću lokaciju, prikaže dostupne informacije o određenim apotekama na teritoriji zadate lokacije.

Prototipska server-klijent veb aplikacija je napisana u programskom jeziku Java (serverski deo) i programskom jeziku JavaScript (klijentski deo).

Osnovne tehnologije koje su korišćene: Spring (Boot, Security, MVC, Data), AngularJS, MongDB, Maven uz korišćenje JHipster (ili „Java Hipster“) generatora aplikacije. Pomenute tehnologije su detaljnije predstavljene u poglavlju 3.1.

2. Faze u razvoju veb aplikacije

U industriji, veb aplikacija nastaje iz potrebe da se softverski proizvod, koji obezbeđuje određene funkcionalnosti, plasira na tržište i proda. Imajući u vidu konkurentnost tržišta i zahteve kupaca, softverskim inženjerima se postavlja zadatak, da u što kraćem vremenskom roku, implementiraju softversko rešenje, koje će da odgovori zahtevima. Softverski inženjeri se tada susreću sa mnogim pitanjima, na koja treba da odgovore, pre nego što pristupe samom razvoju aplikacije:

1. Funkcionalni zahtevi koje treba implementirati
2. Izbor detaljnosti arhitekture: studiozni ili jednostavniji, kako bi se aplikacija završila na vreme
3. Suočavanje sa posledicama jednostavnijeg arhitektonskog pristupa: proširivost aplikacije kako bi se zadovoljili budući očekivani zahtevi
4. "Čistoća" programskog kôda (engl. Clean Code) i primena uzoraka dizajna (engl. Design Patterns)
5. Izbor tehnologija
6. Istraživanje softverskih alata i okvira koji već podržavaju deo zahteva i čija će primena olakšati razvoj aplikacije
7. Monolitna arhitektura, arhitektura zasnovana na mikroservisima ili neka treća
8. Izbor sistema za upravljanje bazom podataka
9. Prikaz izveštaja o podacima koji se koriste u aplikaciji
10. Sigurnost aplikacije za upotrebu u različitim okruženjima (engl. Web Application Security)
11. Podrška za višejezičnost u aplikaciji (engl. Internationalization)
12. Testiranje proizvoda
13. Održavanje programskog kôda
14. Dokumentacija projekta
15. Procedura isporučivanja aplikacije u druga potrebna okruženja (engl. Application Deployment)
16. Nadziranje pokrenute aplikacije u raznim okruženjima (engl. Application Monitoring)

2.1. Analiza zahteva

Pre pristupa izradi aplikacije, neophodno je dobro razumeti zahteve. Oni obično stižu do inženjerskog tîma kao isuviše opšti ili apstraktni, tzv. zahtevi visokog nivoa (engl. High Level requirements). Potrebna je veština rešavanja problema i poznavanje domena problema (engl. Domain knowledge), kako bi se zahtevi: razumeli, analizirali, povezali, po potrebi izmenili/dopunili i raščlanili u konkretnije zahteve niskog nivoa (engl. Low Level Requirements).

Domen problema može da bude vrlo kompleksan i da zahteva znanje iz drugih oblasti ljudske delatnosti (kao, na primer, bankarski sistemi, sistemi za upravljanje raznim elektronskim i mobilnim uređajima i mnogi drugi). Osim primene matematičkog i računarskog znanja, znanje iz pomenutih oblasti može da bude presudno za uspešnost projekta. Zbog toga je često praksa da se na početku definiše model oblasti [1], tako da će softversko rešenje predstavljati njegovo ogledalo. Time je olakšano razumevanje problematike, kao i komunikacija među ljudima koji učestvuju na projektu, u raznim ulogama (uprava, inženjeri za razvoj, inženjeri

za testiranje, krajnji korisnici). U tom slučaju, radi se o Dizajnu vođenim domenom (engl. Domain Driven Design ili DDD).
U ovoj fazi, nastaje i prva dokumentacija projekta, koja se koristi kao osnova za implementaciju veb aplikacije, ali i za pisanje slučajeva korišćenja (engl. Use Cases).

2.2. Pravljenje arhitekture veb aplikacije

Veb aplikacije se projektuju kao server-klijent aplikacije.
Izbor tehnologija koje se mogu koristiti za implementaciju oba dela aplikacije je veliki i zavisi od namene server, kao i vrste klijenta (veb pregledač ili neka druga aplikacija).
Serverski i klijentski deo komuniciraju preko odgovarajućeg protokola, čiji izbor, takođe, zavisi od prethodno navedenog. Takođe je potrebno osigurati odgovarajuću bezbednost pristupa i razmene podataka.

Da bi se kvalitetno razmotrili problemi i alternative, pristupilo se kreiranju prototipa veb aplikacije.
Serverski deo prototipske aplikacije napisan je u Java (sekcija 3.1.1.) programskom jeziku, uz korišćenje pretežno Spring okvira (sekcija 3.1.3.).
Klijentski deo je napisan korišćenjem AngularJS okvira (sekcija 3.1.2.) i njegovih priključaka (engl. Plugins).
Za komunikaciju serverskog i klijentskog dela se koristi HTTP/HTTPS [2] protocol, uz upotrebu REST [3] pristupa. Podaci koji se razmenjuju između servera i klijenta su u JSON [4] formatu.

Serverska aplikacija razvijena po obrascu monolitne arhitekture [5] je slojevita. Glavni slojevi koji se mogu identifikovati su:

- Entiteti i skladišta (engl. Repository) – zaduženi za upravljanje objektima u bazi podataka
- Servisi – sadrže implementaciju poslovne logike i zahteva
- Sloj zadužen za integraciju i komunikaciju komponenti sistema i slične funkcionalnosti (na primer, konfiguracije, sistemi za poruke, logove, sigurnost aplikacije i sl.)
- Prezentacioni sloj – kontroleri koji predstavljaju interfejs ka klijentskoj aplikaciji

Prema Ričardsonu, uočeni glavni problemi ovog pristupa bi trebalo da se reše korišćenjem obrasca mikro-servis arhitekture [6], koji treba da obezbedi:

- Jednostavniji razvoj aplikacije
- Olakšan proces instalacije
- Proširivost

U mikro-servis arhitekturi, aplikacija se dekomponuje na više servisa. Svaki od njih se razvija i upotrebljava nezavisno od ostalih. Svaki od njih može da ima i svoju bazu podataka. Oni između sebe mogu da komuniciraju preko odabranog protokola. Svaki mikroservis se, takođe, može instalirati nezavisno od ostalih, što omogućava da se menja ili zameni novim, bez uticaja na ostatak sistema.

Moderan pristup u razvoju klijentske komponente je Aplikacija jedne strane (engl. Single Page Application) [7]. Kod ovog pristupa, dovlači se jedna HTML strana i dinamički ažurira tokom interakcije sa korisnikom. Time se izbegavaju čekanja prilikom dovlačenja sadržaja sa server, koja se vizuelno primećuju.

Aplikacija bi trebalo da bude skalabilna [8], tj. proširiva u smislu da se mogu dodavati resursi, a da se sama aplikacija ne menja. Horizontalna skalabilnost podrazumeva dodavanje još jedne instance aplikacije ili baze podataka, u distribuiranom sistemu. Vertikalna skalabilnost podrazumeva dodavanje resursa jednoj aplikaciji, kao što su memorija ili procesor.

Popularan pristup koji se koristi u razvoju veb aplikacije je Dizajn vođen domenom, opisan u tački 2.1.

U tradicionalnom pristupu implementacije, prvo se piše programski kôd aplikacije, a onda se za njega pišu testovi. Metoda Razvoj vođen testovima (engl. Test Driven Development ili TDD) [9] forsira obrnuti pristup. Testovi se pišu pre aplikacije. Proces se izvodi u iteracijama. U svakoj se dodaju testovi koji opisuju novi funkcionalni zahtev. Piše se onoliko programskog kôda aplikacije, koliko je potrebno da testovi uspešno prođu.

Metoda Razvoj vođen ponašanjem (Behavior-driven Development ili BDD) [10] predstavlja proširenje metode TDD, uz korišćenje metode DDD. Ova metoda olakšava razvoj i saradnju između tîmova: inženjera za razvoj, inženjera za testiranje i uprave.

2.3. Testiranje veb aplikacije

Testiranje veb aplikacije treba da obuhvati sve njene nivoe. Popularan vodič za testiranje veb aplikacija [51] razlikuje šest većih kategorija:

1) Funkcionalno testiranje (engl. Functionality Testing)
   Testiranje funkcionalnosti veb aplikacije preko veb pregledača.
2) Testiranje korišćenja (engl. Usability testing)
   Aplikacija se testira preko veb pregledača nezavisno od funkcionalnosti.
3) Testiranje interfejsa (engl. Interface testing)
   Testira se interakcija sa veb serverom koji je pokreće i bazom podataka.
4) Testiranje kompatibilnosti (engl. Compatibility testing)
   Testira se kompatibilnost sa okruženjem: veb pregledačem, operativnim sistemom, različitim uređajima sa kojih se pokreće, itd.
5) Testiranje performansi (engl. Performance testing)
   Testiranje performansi aplikacije i iskorišćenosti resursa sistema (memorija, procesor, prostor na disku …) prilikom opterećenja aplikacije velikim brojem korisničkih zahteva ili podataka.
6) Testiranje sigurnosti (engl. Security testing)
   Testira se sigurnost aplikacije – prava pristupa, prava korišćenja, sigurnost podataka koji se čuvaju i/ili razmenjuju sa korisnikom i/ili drugim sistemom.

Postoji više vrsta testova koji se mogu pisati korišćenjem Java/JavaScript jezika. Takođe, postoji veliki broj okvira i alata koji se upotrebljavaju za efikasnije pisanje i izvršavanje.

U ovom radu će biti predstavljeni neki od njih.

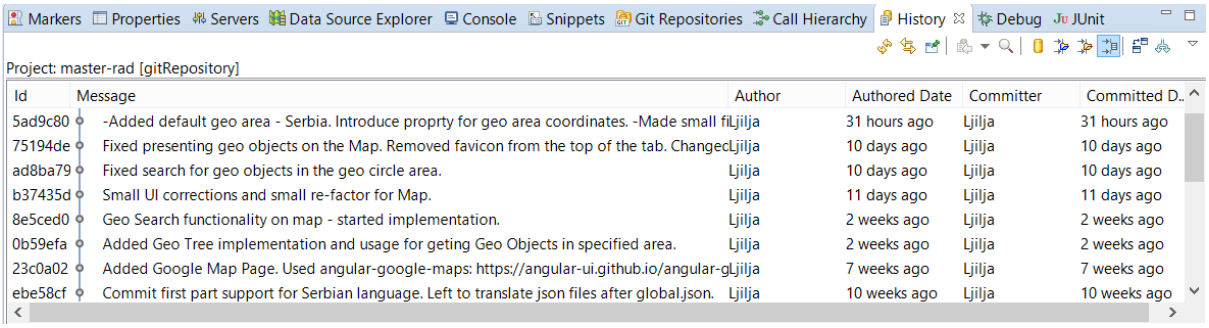
- Testovi za manje jedinice kôda (engl. Unit) - za metode, klase i module. To su testovi koji se izvršavaju u memoriji, nezavisno od drugih komponenti sistema. Ako je potrebno da programski kôd koristi druge komponente, one se simuliraju definišući lažne objekte, sa očekivanim ponašanjem (engl. Mocks).
  Primer okvira i alata:
  - Junit [52] je okvir za pisanje testova u Javi.
  - Jasmine je samostalni okvir, "vođen ponašanjem" (engl. behavior-driven), za pisanje JavaScript testova [53].
  - Karma je alat za pokretanje Java Script jediničnih testova [54].
  - PhantomJS je skript veb pregledač. Koristi se za pokretanje JavaScript jediničnih testova u veb pregledaču [55].
- Integracioni (engl. Integration) testovi, za razliku od jediničnih, koriste druge komponente sistema, kao što su: drugi objekti i niti, baza podataka, datoteke itd.
  Primer okvira:
  - Spring TestContext [56] okvir za integracione testove u Javi
  - Protractor je okvir za end-to-end testiranje AngularJS aplikacija [57].
- Funkcionalni (engl. Functional) testovi se koriste za testiranje funkcionalnosti opisanih u specifikaciji problematike, kojom se bavi aplikacija. Ovim testovima se proverava da li je dobijen očekivani rezultat, zanemarujući sporedne provere stanja i drugih objekata. Mogu se pisati kao integracioni.
- Testovi opterećenja (engl. Load Tests)
  Primer okvira:
  - Gatling je okvir namenjen za Load testiranje [58].

Neki principi za pisanje "čistih" testova opisani su u sekciji 3.1.1.1., tačka 7.
Popularna metodologija pisanja testova je Ponašanjem Vođen Razvoj (engl. Behavior-driven development illi BDD), objašnjen u stavci 2.2. Pomenuti Jasmine okvir upravo koristi ovaj pristup.

2.4. Održavanje programskog kôda

Za programski kôd je potrebno obezbediti centralno skladište za njegovo upravljanje (engl. version control ili revision control ili source control). Na taj način se omogućava jednostavan pristup i sinhronizacija u radu više korisnika. To je i sistem verzionisanja, koji za svaku izmenu, beleži propratne informacije - zašto, kada i ko je napravio izmenu u skladištu (slika 2.4). U svakom trenutku treba da bude moguće preuzeti bilo koju verziju kôda. Postoji veliki broj dostupnih alata za održavanje i korišćenje ovakvog skladišta. U prototipskoj aplikaciji je korišćen Git (sekcija 4.1.)

Slika 2.4 Prikaz dnevnika aktivnosti čuvanja programskog kôda iz razvojnog okruženja Eclipse (više o ovom okruženju u sekciji 4.1.)



Postoje alati koji mogu da se integrišu sa skladištem i omoguće kontinuirano generisanje krajnjeg proizvoda aplikacije (engl. build), pokretanje testova, kao i instalaciju aplikacije u zadato okruženje (engl. deploy). Neki od popularnih su Jenkins [59] i Bamboo [60].

Takođe, postoje alati koji se integrišu sa ovakvim sistemima i mogu da prate kvalitet kôda koji se smešta u skladište (engl. commit), ukazuju na propuste, generišu izveštaje i predlažu rešenja. Jedan od njih je SonarQube [61].

2.5. Održavanje dokumentacije

Prikupljanje, kreiranje i korišćenje dokumentacije su vrlo bitni elementi tokom čitavog životnog ciklusa aplikacije, koji obuhvata: postavljanje zahteva, razvoj, testiranje, održavanje i upotrebu.

Prva dokumentacija nastaje u početnoj fazi razvoja, prilikom analize zahteva (sekcija 2.1.).

Dokumentacija treba da bude dostupna za korišćenje i ažuriranje, po usvojenim procedurama i standardima, članovima svih tîmova koji učestvuju u životnom ciklusu aplikacije. Kao tehnike izrade, osim tekstualne, koriste se i vizuelne, u vidu raznih dijagrama, tabela, slika … Format dokumenta može da bude u različitim oblicima. Postoje različiti alati koji pomažu u njenom kreiranju, čuvanju i deljenju.

Dokumentacija olakšava razumevanje problematike i dobijanje smernica za dalji razvoj. Takođe, olakšava komunikaciju svih učesnika projekta, kao i efikasno uvođenje novih članova.

Postoji više tipova dokumentacije istog proizvoda, u različitim oblicima, gde je proizvod viđen iz različitih uglova, sa različitim nivoima detalja, prilagođena određenoj grupi korisnika …

- Tîmu inženjera za razvoj je potrebna početna dokumentacija, koja opisuje funkcionalnosti koje treba da budu implementirane. Potom, tîm proizvodi sopstvenu dokumentaciju vezanu za samu implementaciju. Ona obuhvata specifikacije dizajna, implementacije i testiranja. Dokumentacija se često razvija iterativno, počev u fazi pre razvoja, menja se i upotpunjuje tokom samog razvoja i konačnu formu dobija nakon završene implementacije. Sam programski kôd, kao i testovi, ako su napisani “čisto” (sekcija 3.1.1.1.), predstavljaju jednu vrstu dokumentacije. Takođe, postoje alati koji se koriste u svrhu proizvodnje dokumentacije za programski interfejs aplikacije (engl. Application Programming Interface ili API). Jedan takav alat je, na primer, Swagger, korišćen u prototipskoj aplikaciji (sekcije 5.5.7, 6.5.7).

- Tîmu zaduženom za testiranje, takođe je potrebna dokumentacija o funkcionalnostima, da bi, na osnovu nje, mogli da naprave svoju dokumentaciju sa koracima za testiranje (engl. test use cases). Da bi validirali proizvod, potrebna im je i određena dokumentacija u vezi sa korišćenjem aplikacije, koja najčešće dolazi od tîma zaduženog za razvoj. Testiranje različitih nivoa aplikacije i njenog okruženja, (sekcija 2.3.) proizvodi različitu dokumentaciju.
- Za instalaciju (sekcija 2.6), održavanje i nadgledanje (sekcija 2.7.), neophodna je druga vrsta dokumentacije, kako bi se aplikacija uspešno i sigurno (sekcija 3.2.) instalirala i omogućila se upotreba krajnjim korisnicima. Tîm zadužen za to, treba da bude dobro upoznat sa zahtevima aplikacije u smislu resursa i njenim potencijalnim slabim tačkama, koji bi mogli da dovedu do problema.
- Upravnicima projekta je potrebna dokumentacija o proizvodu i njegovoj upotrebi na višem nivou, izuzeta od konkretnih implementacionih detalja, orijentisana ka predstavljanju na tržištu i korišćenju od strane krajnih korisnika.
- Kompleksne aplikacije mogu da imaju segmentirane funkcionalnosti po korisničkim grupama, pa je za svaku od njih potrebno obezbediti odgovarajuću dokumentaciju. Takođe, upotreba iste aplikacije može biti prilagođena različitim grupama korisnika, u zavisnosti od jezika i regiona u kom se nalaze (sekcija 3.3.).
  Primer dokumentacije za korišćenje aplikacije od strane krajnjeg korisnika je prikazan na prototipskoj aplikaciji u odeljku 6.

2.6. Isporučivanje veb aplikacije

Računarska infrastruktura, za instalaciju i upotrebu aplikacije, može biti u oblaku (engl. cloud computing) ili u okviru kompanije, tj. privatna (engl. data center). Prema jednom od davalaca usluga obezbeđivanja računarske infrastrukture u oblaku "Amazon Web Services", računarstvo u oblaku se odnosi na isporuku IT resursa i aplikacija putem interneta, sa naplatom iste dok se koristi [62]. Neke navedene prednosti ovakve infrastrukture su:

- Oslobađanje korisnika od kupovine i održavanja hardvera, kao i procene potrebnih fizičkih resursa. Takođe, ova infrastruktura je skalabilna shodno potrebama, što je teško postići sa privatnom infrastrukturom.
- Brže se dobije potrebno okruženje za razvoj aplikacije.
- Zbog velikog broja klijenata, manji su troškovi hardvera za davaoca usluga, a samim tim je i manja cena usluga klijentima.
- Aplikacija može da se instalira u regionima širom sveta, što obezbeđuje bolju internet vezu, a samim tim i bolje iskustvo korišćenja aplikacije krajnjim korisnicima (engl. user experience) u tim regionima.

Privatna infrastruktura se preporučuje kompanijama koje imaju više zahtevnijih aplikacija, koje iziskuju različite resurse. Takođe, kada je u pitanje sigurnost aplikacije (sekcija 3.2.), prednost se daje privatnim.

Obično se izdvajaju sledeća okruženja: razvojno (engl. development environment), test (engl. test environment), pripremno (engl. staging environment) i produkciono (engl. production environment), koje će biti korišćeno od strane krajnjih korisnika.

Aplikcija za instalaciju treba da sadrži svu potrebnu konfiguraciju [63]. Potrebno je da bude optimizovana, kao, na primer, optimizacija HTML, CSS i JavaScript resursa (enlg. minification) [64]. Održavanje programskog kôda „čistim" (sekcija 3.1.1.1.) je, takođe, vid optimizacije, u smislu smanjenja obima kôda.

Proces instalacije treba da bude što jednostavniji i što više automatizovan. Poželjno je da krajnji proizvod aplikacije bude jedna zapakovana datoteka, sa svom potrebnom konfiguracijom. One mogu da imaju ugrađen veb server koji ih pokreće. Na primer, proizvod prototipske aplikacije je "war" datoteka, koja ima ugrađen Tomcat (sekcija 4.2).

Najčešće se koriste alati (od drugog proizvođača ili napravljeni za sopstvene potrebe), koji automatski mogu da izvrše ceo process instalacije aplikacije, uključujući i bazu podataka i druge pridružene resurse i servise.

Za veće grupe korisnika, poželjno je horizontalno skalirati (sekcija 2.2) aplikaciju (engl. clustering), kao i bazu podataka (engl. replication or sharding, sekcija 3.1.4.), radi boljih performansi.

Instalacija zavisi i od arhitekture same aplikacije (monolitna ili zasnovana na mikroservisima, sekcija 2.2.).

Posebna pažnja, prilikom instaliranja aplikacije, posvećuje se oblasti bezbednosti (sekcija 3.2.).

2.7. Nadziranje veb aplikacije

Okruženje, u kome je instalirana aplikacija, potrebno je stalno nadgledati i pratiti (engl. monitoring), kako bi se na vreme i uz minimalan napor, otklonile greške i problemi. Za produkciono okruženje (engl. production environment) to je neophodno, ali se takva praksa preporučuje i za ostala okruženja (sekcija 2.6.).

Uzrok problema može da bude hardverske prirode - nedostatak procesora, RAM memorije, prostora na disku, mrežni ili neki drugi problem, zbog kojih aplikacija ne može pravilno da se izvršava ili koristi.

Uzroci softverske prirode mogu da budu različiti: greška u implementaciji (engl. bug) ili nedostaci u dizajnu kao što su: loše performanse (spora obrada zahteva, neefikasan pristup podacima) ili neskalabilnost (nemogućnost obrade velikog broja zahteva u određenom vremenskom intervalu, nedovoljno prostora za skladištenje podataka).

Jednu vrstu praćenja svojih aktivnosti (engl. logging), aplikacija može da obezbedi sama. Svaka aplikacija treba da generiše izveštaje, tj. dnevnike o izvršenim akcijama (engl. logs), koji mogu da sadrže različite nivoe informacija (engl. log levels) – greške, upozorenja, opšte informacije ili pak detalje. Poželjno je da nivoi budu podesivi i da se mogu menjati tokom izvršavanja aplikacije, bez njenog zaustavljanja. Generisanje izveštaja i menjanje nivoa u prototipskoj aplikaciji, opisano je u sekcijama 5.5.6. i 6.5.6. Postoje alati, koji mogu da procesiraju ovako generisane izveštaje prema definisanim šablonima i generišu odgovarajuća obaveštenja ili upozorenja.

Aplikacija, takođe, može sama da implementira i sakupljanje odgovarajuće statistike o resursima koje koristi, kao što su: servisi, baza podataka, JVM i fizički resursi sistema (memorija, procesor i dr.). Za prototipsku aplikaciju, to je opisano u sekcijama: 5.5.2., 5.5.3., 6.5.2., 6.5.3.

Osim oslanjanja na samu aplikaciju, poželjno je odabrati i odgovarajuće alate, koji bi nezavisno od aplikacije, pratili njene aktivnosti, interakciju sa okruženjem u kojem se izvršava, samo okruženje i interakciju sa korisnicima. Dostupan je veliki broj alata u ovu svrhu, koji mogu da obezbede grafički prikaz stanja, kao i slanje upozorenja ukoliko dođe do nekog problema, putem raznih servisa za poruke (na primer, elektronskom poštom), ciljnoj grupi.

3. Izdvojeni aspekti veb aplikacije

3.1. Izbor tehnologija

Neke kompanije imaju standarde kada su u pitanju programski jezici i prateće tehnologije koje će se koristiti. Uvođenje korišćenja novih tehnologija je uslovljeno određenim procedurama i kriterijumima. Postoje kompanije koje dozvoljavaju slobodu izbora tehnologija tîmu softverskih inženjera, koji će raditi na projektu. Izbor je nekada uslovljen znanjem i veštinama koje inženjeri već poseduju. Praksa pokazuje da se obično najbolji izbor napravi ukoliko se najpre uradi istraživanje aktuelnih tehnologija. Neki od kriterijuma za odabir bi trebalo da budu: mogućnosti tehnologije; poželjno je da je softver otvorenog kôda; kompatibilnost sa drugim tehnologijama koje će se koristiti; dokumentovanost, jasnoća i lakoća upotrebe; podrška od strane autora ili dostupnost adekvatne pomoćne literature i članaka; aktuelnost, obim primenjenosti u industriji, što potencijalno može uticati na njenu sudbinu opstanka na tržištu.

U narednim odeljcima, predstaviće se neke od tehnologija koje su korišćene i u prototipskoj aplikaciji.

3.1.1. Java

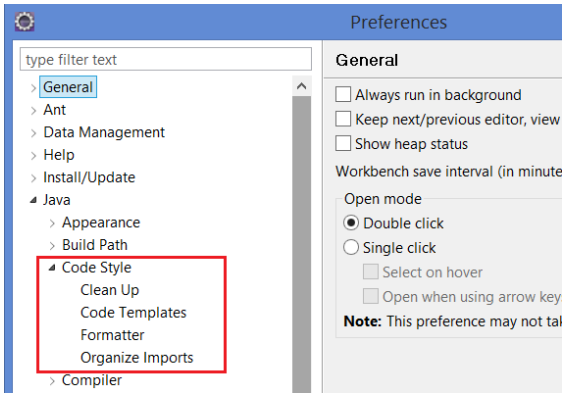
Java [11] je jedan od najaktuelnijih programskih jezika današnjice. Na pojedinim rang listama zauzima prvo mesto. Jedna od njih je rang lista holandske kompanije TIOBE [12], na kojoj je više puta od 2006. zauzimala prvo mesto. Kriterijum merenja za tu rang listu je broj rezultata pretrage na vebu, za zadati izraz koji sadrži ime programskog jezika.

Java je nastala devedesetih godina prošlog veka. Poslednja dostupna verzija Jave, u trenutku pisanja ovog teksta, je Java 8, nastala 2014. godine.

3.1.1.1. Pisanje "čistog" kôda

Pisanje "čistog" kôda i primena dizajn uzoraka čine programski kôd čitljivijim, razumljivijim, lakše proširivim i manje podložnim greškama. Sve to može značajno da olakša svakodnevni rad na projektu inženjerima softvera. Tako napisan kôd lakše pregledaju drugi članova tîma. Takođe je, na ovaj način, novim inženjerima mnogo lakše da se upoznaju sa projektom i za kraće vreme postanu produktivni.
U podizanju produktivnosti, može da pomogne i korišćenje šablona za stil kôda. U Eclipse razvojnom okruženju ih je jednostavno definisati (slika 3.1.1.1). Primer Javinih šablona se nalazi u prototipskoj aplikaciji, u direktorijumu *master-rad/config/eclipse*.

Slika 3.1.1.1



U razmatranju čistoće kôda, predstaviće se neki od principa pisanja čistog kôda, opisanih u knjizi: "Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship" [13].
Posebna pažnja se obraća na one principe pisanja čistog kôda koji su primenjeni i u prototipskoj aplikaciji:

1. Korišćenje smislenih imena (Poglavlje 2 knjige [13]).
   Klase, promenljive, konstante i metode treba da imaju smislena imena. Treba izbegavati ona koja uzrokuju pogrešno razumevanje značenja i upotrebe. Ovo ne mora da se primeni na lokalne promenljive čiji opseg važenja je mali (kao što su, na primer, imena promenljivih u petljama metoda.)
   Primer:
   service/geospatial paket koristi jednoznačnu notaciju za opis elemenata i operacija u geografskom koordinatnom sistemu. Jasno su naznačene klase, metode i svrha njihove upotrebe.
   Nazivi klasa: `GeoNode`, `GeoPoint`, `GeoCircle`,`GeoRect` ...
   Nazivi metoda: `GeoNode#getGeoRectCentroid()` ...
   Nazivi promenljivih: `private GeoRectArea nodeArea;`

2. Funkcije (Poglavlje 3 knjige [13])
   Metode (funkcije) bi trebalo da budu kratke i izvršavaju jednu akciju. Poželjno je napisati je tako da može da se čita kao priča, kako bi se razumela namena. Broj argumenata treba da bude što manji. Ako ih je više, trebalo bi ih predstaviti zajedničkim objektom. Treba izbegavati ponavljanje kôda.
   Primer:
   Brojni primeri se mogu naći u *service/geospatial* paketu.
   `GeoLibrary#getGeoRectCentroid(final GeoRectArea area)`
   Argument metode je objekat koji sadrži četiri geo-koordinate. Nakon formiranja tačaka od zadatih koordinata, funkcionalnost pronalaženja centroida za dve tačke je izdvojena u posebnu metodu `GeoLibrary#getCentroidOfGeoPoints(final List<GeoPoint> points)`, koja je generička u smislu da može odrediti centroid proizvoljnog broja tačaka. Konvertovanje geo-tačke u Dekartov sistem i obratno je izdvojeno u posebne metode, radi preglednosti kôda i kako bi bile upotrebljive za druge svrhe. To su sledeće metode: `geoPointToCartesian3Dpoint()`, `cartesian3DpointToGeoPoint()`.

3. Komentari (Poglavlje 4 knjige [13])
   Komentare ne bi trebalo upotrebljavati da se objasni loše napisan kôd. Oni bi trebalo da imaju informativnu svrhu samo na mestima gde je neophodno da objasne nameru ili upozore. Ne treba ih stavljati tamo gde je razumevanje kôda očigledno. Ovo važi i za komentare koji se generišu za potrebe dokumentacije (Javadoc [14]).
   Komentare ne treba upotrebljavati da zamene deo kôda koji nije implementiran, na primer, za izuzetak koji nije obrađen. Ovo ne važi za "uraditi" (engl. TODO) komentare. Takođe, komentare ne treba upotrebljavati da označe deo kôda koji može biti izmešten u posebnu metodu ili promenljivu.
   Primer:
   Promenljive i metode klase `Cartesian3DPoint` ne sadrže nijedan komentar, jer je iz njihovih naziva očigledna svrha.
   Komentar uz promenljivu klase `GeoCircle` služi samo da naznači da je vrednost u stepenima, nije potrebno reći i da je u pitanju geografska širina.

```
/** decimal degree **/
private Double latitude;
```

   Kratke metode koje jasno označavaju šta rade, ne moraju imati komentare, kao što je slučaj sa brojnim metodama klase `GeoTreeStorage`.

4. Objekti i strukture podataka (Poglavlje 6 knjige [13])
   Objekti sakrivaju podatke, a izlažu funkcije koje operišu nad tim podacima. Strukture podataka izlažu svoje podatke i nemaju drugih funkcija. Nije poželjno kombinovati ih u istoj klasi. Objekti bi trebalo da imaju mali skup promenljivih i svaka metoda bi trebalo u svojoj implementaciji da koristi što više promenljivih iz tog skupa. Oni bi trebalo da budu "zatvoreni" za izmene promenljivih, a "otvoreni" za dodavanje novih funkcija.
   Primer:
   U prototipskoj aplikaciji, primer objekta bi bio `GeoNode`, a primer strukture podataka `GeoPoint`.

5. Upravljanaje greškama (Poglavlje 7 knjige [13])
   Poželjno je koristiti izuzetke umesto raditi sa kodovima grešaka.
   Pri obradi izuzetka, trebalo bi obezbediti dovoljno informacija o njegovom uzroku. Nekada su neproveravani (engl. unchecked exceptions) izuzeci poželjniji od proveravanih (engl. checked exceptions), u slučaju da se dogodila greška. Tada ima smisla da se proces prekine, sa ispisom odgovarajuće poruke i uzroka.
   Važno je odrediti gde Proveravani izuzeci treba da budu obrađeni. Prosleđivanje proveravanih izuzetaka ima za posledicu promenu potpisa svih metoda "iznad" u hijerarhiji poziva. Važno je voditi računa da se ne povredi enkapsulacija komponenti u hijerarhiji, da one u višem nivou nemaju znanje o izuzecima definisanim u nižem sloju.
   U tome mogu da pomognu Proveravani izuzeci definisani programskim kôdom. Oni, takođe, mogu da sakriju izuzetke definisane bibliotekama, što doprinosi lakšem održavanju kôda.
   Ako u slučaju izuzetka nije potrebno prekinuti proces, umesto izuzetka, poželjno je koristiti objekat, kao povratnu vrednost metode koja se poziva.
   Izuzetke bi trebalo generisati i u slučaju da su metode dobile nevalidne vrednosti za argumente, kao, na primer, prazne vrednosti (engl. null).
   Primer:

U konstruktoru klase `GeoNode` se proverava da li je geo-prostor, prosleđen kao argument, prazna vrednost. U tu svrhu se koristi Assert biblioteka Spring okvira:
`notNull(area, "area should not be null.");`

6. Klase (Poglavlje 10 knjige [13])
   Klase bi trebalo da počnu listom javnih statičkih konstanti. Zatim, slede privatne statičke promenljive, a onda privatne dinamičke promenljive. Retko kada postoji potreba za javnim promenljivama. Potom slede javne metode. Posle javne metode, mogu da se stave privatne metode, koje ta javna metoda poziva. Nekada postoji potreba da se za metode smanji nivo pristupa na zaštićeni (dostupne u paketu i klasama koje je nasleđuju) ili samo na nivo paketa.
   Primer1:
   Veliki broj metoda klase `GeoTreeStorage` je dostupna samo na nivou paketa.

   Trebalo bi izbegavati pisanje velikih klasa. Kada klasa postane velika i teško razumljiva, znači da se može podeliti na više klasa. Skup promenljivih u klasi ne bi trebalo da bude veliki i svaka definisna metoda bi trebalo da koristi što više njih iz datog skupa. Kao i metode, i klase bi trebalo da poštuju princip jedinstvene odgovornosti.
   Primer2:
   Klasa `GeoNode` je zadužena samo za kreiranje i operacije nad geo-čvorom. Neka druga klasa će inicirati kreiranje korena čvora i pozivati potrebne operacije za kreiranje drveta. Klasa `GeoTreeStorage` je zadužena samo za upravljanje skladištima podataka, koji pripadaju drvetu. Operacije nad skladištima pozivaju druge klase, uglavnom pomenuta `GeoNode`.

7. Pisanje jediničnih testova (Poglavlje 9 knjige [13])
   Jedinični testovi su važni za verifikaciju programskog kôda i pronalaženje slučajeva u kojima kôd ne radi očekivano, a što je bilo teško primetiti tokom njegovog pisanja. Poželjno je da pokrivenost kôda testovima bude što veća. Oni takođe treba da pokriju što više slučajeva: situacije u kojima kôd radi i situacije u kojima će da podbaci – tzv. pozitivni i negativni testovi.
   Testovi su takođe jedan od mehanizama za verifikaciju refaktorisanog programskog kôda (engl. Code refactoring) – testovi koji su se uspešno izvršavali pre prepravljanja kôda, treba da se uspešno izvrše i nakon njegove refaktorizacije. Ako testovi nisu napisani "čisto", teško ih je održavati, naročito kada se programski kôd proširuje. Pod "čistim" se misli da su: čitljivi, jednostavni, da je jasna namera.
   Test bi trebalo da ima jednu tvrdnju (engl. assert). Za inicijalizaciju promenljivih i provere, trebalo bi izbegavati dupliranja i ponavljanja izdvojiti u posebne metode i promenljive.
   Svaki test bi trebalo da se sastoji od tri dela:
   a) "dato" – postavka, pripremaju se promenljive za koje će se pozvati akcija
   b) "kada" – akcija, poziv metode koja se testira
   c) "onda" – tvrdnja, provera očekivanih rezultata
   Testovi bi trebalo:
   - da se izvršavaju brzo kako bi se mogli često pokretati
   - da budu nezavisni jedan od drugog
   - da budu ponovljivi, tj. da mogu da se izvrše više puta, u raznim okruženjima, sa istim rezultatima.

   Primer:

Primeri JUnit [15] testova se mogu naći u `GeoLibraryTest` klasi.

3.1.1.2. Uzorci dizajna

Uzorci dizajna su, kao i algoritmi, svuda oko nas. Najteže ih je prepoznati, kako bi se primenili u konkretnom slučaju. Najbolji uspeh se postiže vežbom i refaktorisanjem početno napisanog kôda. Nekada se najbolje rešenje ne može odmah primetiti, zato je prepravljanja potrebno raditi u iteracijama. Nakon svake iteracije, postaje jasnije i očiglednije koji je sledeći korak u prepravci.
Uzoraka dizajna ima mnogo, kao i knjiga i tekstova koji se njima bave. Neki od njih, opisani u knjizi: "`Joshua Kerievsky. Refactoring to Patterns`" [16], su primenjeni i u prototipskoj aplikaciji.

1. Unikat (engl. Singleton)
   Moguće je napraviti samo jedan objekat klase.
   Primer:
   Potrebno je da postoji samo jedno drvo na nivou prototipske aplikacije, tako da je objekat, koji predstavlja koren drveta, napravljen po ovom uzorku. Klasa: `GeoTreeRoot`.

2. Fabrika (engl. Factory)
   Kreira se više objekata, pri čemu, za kreiranje svakog od njih, može biti zadužena druga nasleđena klasa.
   Primer:
   Prototipska aplikacija bi, korišćenjem ovog uzorka, mogla da se proširi da implementira više različitih drveta, za razne tipove geo-objekata, kao što je navedeno u sekciji 5.2.4.

3. Zastupnik (engl. Proxy)
   Objekat se koristi za pristup nekom drugom objektu, čije kreiranje može biti odloženo do trenutka njegove upotrebe.
   Primer:
   Service `GeoTreeServiceImpl` sadrži objekat koji predstavlja koren drveta, ali je njegova inicijalizacija, a samim tim i kreiranje drveta, odloženo do momenta pozivanja metode: `createAndInitializeGeoTree()`.

4. Korišćenje enumerisanog tipa podataka
   Korišćenje ovog uzorka omogućava više fleksibilnosti u radu sa konstantama, kao i definisanje dodatnih atributa, koji im se mogu pridružiti.
   Primer:
   Korišćenje `GeoAreaEnum` za identifikaciju dece čvorova, povećava čitljivost kôda i pojednostavljuje implementaciju identifikacije deteta čvora.

5. Posmatrač (engl. Observer)
   Često korišćen uzorak u obradi događaja. Implementira ga i Spring okvir.
   Primer:
   Poglavlje 5.5.5. opisuje upotrebu ovog uzorka za prikupljanje podataka o svim prijavljivanjima korisnika Sistema, u određenom vremenskom periodu.

6. Šablon metoda (engl. Template method)
   Određeni koraci algoritma se mogu promeniti u nasleđenoj klasi.
   Primer:
   Jedan od primera primene bi bio u zameni implementacije metode `ifDivisionCriteriaMet()` algoritma za raspoređivanje geo-prostornih tačaka, kao što je opisano u sekciji 5.2.4.

7. Graditelj (engl. Builder)
   Uzorak za kreiranje objekata umesto korišćenja konstruktora sa više parametara. Objektu se pridružuje drugi objekat, koji se poziva za postavljanje svakog od parametara. Rezultat je željeni objekat.
   Primer:
   Klasa `Cartesian3DPoint` ima pridruženog graditelja, unutrašnju klasu: `Cartesian3DPointBuilder`.

8. Sakupljajući parametar (engl. Collecting Parameter)
   Ovaj uzorak podrazumeva prosleđivanje objekata u više metoda da upišu svoj rezultat, umesto pozivanja svake od njih i sakupljanja rezultata.
   Primer:
   Sledeća metoda prosleđuje objekat map u rekurziji, koji se puni u svakom izvršavanju metode.
   `GeoNode# getAllLeavesLocationsAsMap(final Map<Integer, GeoRectArea> map)`.

3.1.2. AngularJS

AngularJS [17] je jedan od popularnijih JavaScript okvira otvorenog kôda, koji se koristi za razvoj klijentskog dela veb aplikacije. Za održavanje ovog razvojnog okvira je inicijalno bio zadužen Google tîm. Njegovom razvoju danas doprinose i pojedinci, koji žele da unaprede softver, kako bi bolje odgovorio izazovima modernog dizajna klijentske aplikacije (sekcija 2.2.).
Postoje dve glavne verzije: AngularJS 1 i Angular JS 2, koje i sintaksno i konceptualno imaju dosta razlika.

U prototipskoj aplikaciji, korišćen je AngularJS 1 (verzija 1.5.5.).
Glavne karakteristike AngularJS 1 okvira su [18]:

- Implementira varijantu Model-Izgled-Kontroler uzorka dizajna (engl. Model View Controller ili MVC), u kojem je kontroler izostavljen, jer korisnik definiše samo izgled i model, a AngularJS ih povezuje.
  Podaci sa izgleda automatski se sinhronizuju sa modelom, u oba smera (engl. two-way data binding).
- Koristi HTML za definisanje izgleda i omogućava korisniku da definiše nove HTML elemente ili AngularJS direktive. Na taj način se podržava uzorak dizajna neponavljanja (engl. DRY, Don't Repeat Yourself). Definisane direktive se mogu koristiti u celom projektu.
- Pomaže da se aplikacija postavi i dobro struktuira, što olakšava održavanje i testiranje.

- Podržava uzorak „ubrizgavanja zavisnosti“ (engl. dependency injection), tako što okvir poziva implementacije iz aplikacije, pri čemu se one prosleđuju kao objekti.

Programerima je na raspolaganju obimna dokumentacija i veliki broj vodiča za implementaciju, kao i za pisanje “čistog kôda”. Jedan od njih je korišćen i u implementaciji prototipske aplikacije [19].

3.1.3. Spring okvir

Spring [20] je okvir za Java platform, koji obezbeđuje infrastrukturu za razvoj aplikacije. Zahvaljujući tome, softverski inženjeri mogu da se fokusiraju na razvoj biznis zahteva aplikacije.

Spring je po dizajnu modularan. Nudi više od dvadeset projekata, koji obezbeđuju različite infrastrukture: osnovni Spring okvir (engl. Spring framework), okvir koji postavlja i podiže aplikaciju (engl. Spring Boot), okvir za rad sa bazama podataka (engl. Spring Data), okvir koji obezbeđuje alate za rad u oblaku (engl. Spring Cloud), okviri za sigurnost aplikacije (engl. Spring Security, Spring LDAP), okviri za upravljanje porukama (engl. Spring Integration, Spring AMQP) i drugi.

Osnovni Spring okvir (engl. Spring Core) [21] omogućava kreiranje skladišta objekata, korišćenjem dizajn uzorka Inverzija kontrole (engl. Inversion of control container).

Sastoji se od oko dvadesetak modula, grupisanih u: Osnovno skladište (engl. Core Container), Pristup/integracija podataka (engl. Data Access/Integration), Veb (engl. Web), Aspekt orjentisano programiranje (engl. Aspect Oriented Programming ili AOP), Instrumentalizacija (engl. Instrumentation), Upravljanje porukama (engl. Messaging) i Test (engl. Test).

Spring minimizuje svoje zavisnosti ka drugim bibliotekama i okvirima. Jedina zavisnost osnovnog Spring okvira je ka biblioteci za upis u dnevnik aktivnosti (engl. logging).

Integracija Springa u aplikaciju je jednostavna uz korišćenje alata, kao što je, na primer, Maven (što je korišćeno i u prototipskoj aplikaciji).

Spring Boot [22] je okvir koji omogućava kreiranje infrastrukture za samostalne aplikacije, spremne za produkciju. Korišćen je u prototipskoj aplikaciji.

Ima ugrađen veb server (Tomcat, Jetty ili Undertow), pa je aplikaciju dovoljno pokrenuti kao Java aplikaciju. Klasa prototipske aplikacije, koja sadrži metodu `main()`, je `MasterRadApp`.

Spring Boot, takođe, postavlja inicijalni *pom.xml* fajl, za konfiguraciju sa Maven alatom.

Neke od funkcionalnosti koje obezbeđuje su [23]:

- Osnovni Spring okvir
- Eksternu konfiguraciju
- Profile
- Upis u dnevnik aktivnosti
- Podršku za veb
- Podršku za neke baze podataka
- Upravljanje sistemima poruka
- Podrška za pisanje i pokretanje testova
- Protokole: HTTP, JMX, SSH
- Nadgledanje aplikacije: metrika, statistika, niti, procesi

Spring Security [24] je okvir koji nudi rešenje za autentifikaciju i autorizaciju aplikacije. Spring Boot okvir, kao podrazumevanu autentifikaciju, implementira osnovni tip Spring Security okvira. Takođe podržava OAuth2. (sekcija 3.2.).

Za implementaciju REST kontrolera, u prototipskoj aplikaciji, korišćen je Spring Boot MVC okvir [25]. REST kontroleri komuniciraju sa klijentskom stranom aplikacije preko HTTP/HTTPS protokola. Podaci su u JSON formatu. Za konfiguraciju se koriste zabeleške (engl. annotation). Na primer, zabeleška *@RestController* označava da je klasa REST kontroler; zabeleška *@RequestMapping* vezuje kontroler za URL preko koga je dostupan. Ista zabeleška sa odgovarajućim atributima se koristi i na nivou metoda kontrolera da označi URL do metode, kao i tip zahteva: DOHVATI (engl. GET), POSTAVI (engl. POST), OBRIŠI (engl. DELETE) ….

Za rad sa bazom podataka, u prototipskoj aplikaciji, korišćen je Spring Boot MongoDB [26]. Entiteti su predstavljeni u POJO modelu. Za upravljanje podacima koriste se skladišta. Skladišta nasleđuju klasu *MongoRepository*, čime sadrže već implementirane, podrazumevane metode za: čuvanje, brisanje i osnovno dohvatanje entiteta po imenu jednog ili više atributa. Na primer, metoda `findOneByActivationKey()` interfejsa `UserRepository` dohvata jedan dokument po aktivacionom ključu i nije potrebno da je programer implementira.

Konfiguracija se postavlja korišćenjem zabeleški. Klasa *MongoTemplate* je pomoćna klasa za implementaciju operacija nad podacima.

3.1.4. MongoDB

MongoDB [27] je nerelaciona baza podataka otvorenog kôda. Postoji i komercijalna varijanta koja pruža više mogućnosti, kao što su dodatni servisi sa sertifikatima i podrška MongoDB tîma inženjera u razvoju i rešavanju problema. Umesto tradicionalne strukture tabela relacionih baza podataka, MongoDB je orjentisana ka dokumentima.

Popularnost nerelacionih baza porasla je sa ekspanzijom veba i potrebe da se uskladište različiti multimedijalni sadržaji, koje je teško smestiti u relacione baze.

Zašto odabrati ovu bazu obrazloženo je u [28]. Glavni argumenti su:

- MongoDB može da uskladišti sadržaj bilo kog tipa.
- Ima ugrađenu skalabilnost.
- Procenjuju se manji troškovi u produkciji.

Arhitekturalne karakteristike MongoDB baze [29] su:

- Dinamička šema, za razliku od statičke kod relacionih baza podataka. Nije je potrebno predefinisati, kako bi se sačuvali podaci. Šema može da se menja bez zaustavljanja aplikacije. Kolekcija prihvata podatke različite strukture. Promena u strukturi novih podataka ne stvara nužnost promene starih.
- Šema se sastoji od kolekcija, a kolekcija od dokumenata. Dokument se sastoji od polja. Dokumenti se čuvaju u binarnoj reprezentaciji (engl. Binary JSON ili BSON).
- Pri dizajnu kolekcije, nastoji se da dokument sadrži sve potrebne podatke koji opisuju objekat, za razliku od relacionih, gde bi to bilo podeljeno između više tabela. To smanjuje potrebu za operacijom spajanja kolekcija, čime se upiti brže izvršavaju. Od

verzije 3.2, uveden je $lookup operator za levo spoljno spajanje, inspirisano potrebama u pravljenju izveštaja sa analitičkim prikazom.

- Podržana su svojstva: atomičnosti, konzistencije, izolacije i trajnosti (engl. ACID - Atomicity, Consistency, Isolation, Durability) na nivou dokumenta.
- MongoDB podržava 4 tipa skladišta podataka (engl. storage engine). Od verzije 3.2. tip Žičani tigar (engl. WiredTiger) je podrazumevano skladište.
    - Kontroliše istovremeni upis na nivou dokumenta, što omogućava istovremeno pisanje različitih dokumenata u okviru iste kolekcije.
    - Koristi tehniku verzionisanja kontrolnih tačaka, da označi podatke u memoriji koji još uvek nisu upisani na disk (engl. MultiVersion Concurrency Control - MVCC).
    - Vode se evidencije dešavanja između kontrolnih tačaka dve verzije (engl. Journal), kako bi se baza mogla oporaviti ukoliko prekine sa radom u nekom trenutku.
    - Postoji mogućnost kompresije kolekcija i indeksa, čime se štedi prostor na disku, ali troši više procesorskog vremena.
- Što se izražajnosti upitnog jezika tiče, podržano je više vrsta upita. Neki od njih su:
    - upiti ključ-vrednost (engl. key-value queries)
    - upiti opsega (engl. range queries)
    - geo-prostorni upiti (engl. geospatial queries)
    - upiti kojima se izvršavaju operacije nad grupom vrednosti (engl. aggregation queries).
- Koriste se sekundarni indeksi za optimizaciju upita.
- Ima ugrađeno „horizontalno“ skaliranje kojim se podaci distribuiraju na više fizički odvojenih particija (engl. sharding). Podržano je više tipova:
    - Raspodela dokumenata je po vrednosti ključa skaliranja, dokumenti koji su blizu najverovatnije će pripasti istoj particiji (engl. range-based sharding).
    - Skaliranje po MD5 disperziji vrednosti ključa skaliranja (engl. hash-based sharding).
    - Korisnički definisana veza vrednosti ključa skaliranja sa particijom (location-based sharding).
- Podržana je replikacija. Ukoliko jedna kopija prestane da radi, preostale u sistemu će preuzeti i njenu funkciju.
- Postoji varijanta MongoDB baze smeštene u memoriju (engl. in-memory MongoDB), koja može da se koristi i u kombinaciji sa fizičkim bazama u replikaciji.
- Postoji veliki broj alata za vizuelizaciju i manipulisanje podacima.
- Podržana je integracija sa velikim brojem programskih jezika: Java, .NET, Ruby, PHP, JavaScript, node.js, Python, Perl, Scala …
- MongoDB podržava sigurno korišćenje:
    - Autentifikaciju - podržava integraciju i sa drugim sigurnosnim mehanizmima
    - Autorizaciju kao prava pristupa podacima
    - Vodi se evidentiranje i pregled (engl. auditing) pristupa bazi i njenog korišćenja
    - Podaci se mogu šifrovati. Jedno od skladišta podataka koje je podržano je kreirano upravo u ovu svrhu (engl. Encrypted storage engine).
- Kreirani su alati koji omogućavaju lakšu instalaciju, nadgledanje i održavanje.

3.1.5. JHipster

JHipster [30] je alat otvorenog kôda, koji se koristi za generisanje veb aplikacija. Korišćen je od strane velikog broja poznatih kompanija. Za implementaciju prototipske aplikacije korišćena je verzija: 3.4.0, poslednja dostupna u trenutku započinjanja rada na projektu.

JHipster podržava veći broj popularnih alata i okvira [31] za generisanje serverske i klijentske aplikacije.
Neki od njih, koji su korišćeni na klijentskoj strani, su:

- HTML5 Boilerplate [32]
  Koristi se za postavljanje početne aplikacije sa HTML 5 šablonima, koja se dalje može nadograđivati, shodno potrebama korisnika.
- Bootstrap [33]
  Okvir otvorenog kôda koji definiše HTML i CSS šablone, kao i JavaScript proširenja.
- AngularJS
  Opisan u sekciji 3.1.2.
- Yeoman [34]
  Postavlja šablon klijentske strane aplikacije, korišćenjem najbolje prakse za razvoj, alate i okvire.
- Bower [35]
  Upravlja paketima koji sadrže: HTML, CSS, JavaScript, znakove i slike. Omogućava instalaciju potrebnih verzija i njihovih zavisnosti.
- Gulp [36]
  Alat za automatizaciju izvršavanja zadataka u razvojnom okruženju.
- Karma i PhantomJS (kratko opisani u sekciji 2.3.)

Na serverskoj strani aplikacije, korišćeni i u prototipskoj aplikaciji, podržani su:

- Spring (Boot, Security, REST, Data) (sekcija 3.1.3.)
- MongoDB (sekcija 3.1.4).
- Maven

Maven [37] je alat koji se koristi za upravljanje projektom: njegovu izgradnju, kao i obezbeđivanje izveštaja i dokumentacije iz centralizovanog izvora informacija.
Neke od njegovih mogućnosti su [38]:

- Jednostavno postavljanje projekta, jednog ili više modula
- Jednostavno upravljanje zavisnostima (engl. dependency management), čime se omogućuje da u izgradnji uvek učestvuju prave verzije biblioteka. Koristi centralno skladište. Zahtev za dovlačenje/korišćenje date biblioteke u projektu, predstavlja jednostavnu konfiguraciju u *pom.xml* datoteci.
- Podržava veliki broj priključaka (engl. plugins)
- Pravljenje krajnjeg proizvoda projekta kao što je datoteka JAR ili WAR formata
- Generisanje informacija i izveštaja vezanih za projekat
- Upravljanje krajnjim proizvodom i njegovo prilagođavanje za različita okruženja, u kome će aplikacija biti pokrenuta. Lako se integriše sa sistemom za verzionisanje kôda (enlg. control versioning system) i sistemom za kontinuirano pokretanje testova i pravljenje krajnjih proizvoda projekta (engl. Continuos integration system).

Primena JHipstera u prototipskoj aplikaciji objašnjena je kroz poglavlje 5.

3.2. Sigurnost veb aplikacije

Obezbeđivanje veb aplikacije podrazumeva sigurnost same aplikacije, kao i njenog okruženja, tj. obezbeđenje različitih nivoa sigurnosti (enlg. "layers of security") [39]. Svaki od njih treba da ima svoj sigurnosni mehanizam. Nivoi se mogu posmatrati hijerarhijski. Svaki sledeći je dodatni nivo sigurnosti. Na dnu se nalazi nivo koji podrazumeva sigurnost transporta podataka i sistemsku identifikaciju. Zatim je potrebno osigurati pristup operativnom sistemu i sistemu datoteka samo za privilegovane korisnike. Može se osigurati i pristup Java Virtuelnoj Mašini (za Java veb aplikacije). U te svrhe se koriste softveri zaduženi za nadgledanje sistema i sprečavanje nedozvoljenih radnji.

Najviši nivo sigurnosti je osiguranje koje implementira sama veb aplikacija, a što se najčešće odnosi na autentifikaciju i autorizaciju.

OWASP (Open Web Application Security Project) [40] je svetska, neprofitna, dobrotvorna organizacija koja se bavi pitanjima sigurnosti softvera.

U dokumentaciji ove organizacije, može se pronaći opis sledećih kategorija, koje opisuju postupke i metode, koje ugrožavaju bezbednost aplikacije:

- Napadi (engl. Attacks) su tehnike koje napadači koriste kako bi iskoristili „ranjivosti" ili „slabe tačke" aplikacije [41].
- Ranjivost (engl. Vulnerability) je slaba tačka aplikacije, nastala kao propust u dizajnu ili greška u implementaciji, a koja omogućava napadaču da nanese štetu akterima nad aplikacijom: vlasniku ili korisnicima [42].
- Dodavanje/izmena delova kôda u aplikaciji koji mogu da ugroze sigurnost [43].
- Pretnja (engl. Threats) kao zlonameran softver [44].

Ova organizacija, takođe, izdvaja deset najčešćih pretnji sigurnosti aplikacije [45]. Predstaviće se prvih pet:

1. Ubrizgavanje (engl. Injection) – Primer je slanje neosiguranih SQL, OS ili LDAP podataka kao deo komande ili upita koji mogu biti prošireni neprijateljskim podacima a potom i izvršeni u neznanju, bez odobrenja.
2. Propusti u mehanizmima autentifikacije i upravljanja sesijama (engl. Broken Authentication and Session Management) – usled nedostataka u dizajnu i implementaciji ovih mehanizama, napadač može da ugrozi šifre, ključeve i oznake sesija kako bi preuzeo identitet korisnika.
3. Izvršavanje skripte kroz sajtove (engl. Cross Site Scripting) – ukoliko aplikacija šalje neproverene podatke veb pregledaču, napadač može da izvrši skriptu u pregledaču žrtve koji preuzme sesiju, obriše veb stranice ili preusmeri korisnika na zlonameran sajt.
4. Nezaštićeno direktno referisanje na objekte (engl. Insecure Direct Object References) – ukoliko programski kôd izlaže reference na važne objekte bez zaštite, napadač može manipulisati njima, kako bi došao do važnih podataka.
5. Nedostaci u sigurnosnoj konfiguraciji (engl. Security Misconfiguration) – sve komponente aplikacije i okruženja treba da budu obezbeđeni: okviri, aplikativni server, veb server, baza podataka, platforma. Podrazumevana sigurnost ovih

komponenti obično nedostaje ili je slaba, pa ju je potrebno dodatno konfigurisati.

Najpoznatiji tipovi HTTP autentifikacije korisnika, koje i Java podržava, su [46]:

- Osnovna autentifikacija (engl. basic) – korisničko ime i šifra se šalju kao Base64 kodiran tekst. Ukoliko se ne koristi SSL ili VPN uz to, nije mnogo siguran mehanizam, jer se server ne autentifikuje.
- Osnovna autentifikacija zasnovana na formi (engl. formed-based) – korisničko ime i šifra se šalju kao običan tekst. Sigurnost je, kao i za osnovnu autentifikaciju, opisana u prethodnoj stavci. U ovom pristupu, može da se menja izgled forme za autentifikaciju.
- Kod sistematizovane (engl. digest) autentifikacije, umesto šifre šalje se šifrovana vrednost lozinke i vrednosti za raspršivanje (engl. hash value).

Popularan standard, za davanje prava pristupa drugoj aplikaciji (engl. third party), je OAuth standard. Umesto korisničkog imena i šifre, koristi se žeton (engl. token), koji generiše server za davanje prava pristupa. Taj žeton se potom koristi za pristup resursima aplikacije. Poslednja verzija ovog standarda je OAuth 2.0. [47]. Ova metoda autorizacije se koristi uz autentifikaciju po OpenId [48] standardu, gde korisnik može da pristupi nezavisnim veb sajtovima, bez ponovne potvrde identiteta.

3.3. Višejezičnost u veb aplikaciji

Često je potrebno prilagoditi sadržaj i izgled veb aplikacije korisnicima, koji pripadaju različitim kulturama ili regionima i govore različitim jezicima. Dizajn veb aplikacije, koji bi takav koncept trebalo da podrži, je poznat pod engl. terminom Internationalization [49]. Koristi se skraćenica “i18n”, koja označava da postoji 18 znakova između znaka ‘i’ i znaka ‘n’ u nazivu na engleskom jeziku. Aplikacije se dizajniraju tako, da nude opcije korisniku za izbor odgovarajućeg jezika.

Prilagođavanje tako dizajnirane veb aplikacije konkretnoj kulturi ili regionu, sa određenim jezikom, predstavlja lokalizaciju (engl. Localization) [50]. Skraćenica koja se koristi je “l10n” i nastala je po sličnom principu kao “i18n”.

Kada je u pitanju podrška za jezik, za veb aplikaciju to znači prikazivanje sadržaja odgovarajućim pismom.

Osim pisma, različite kulture i regioni mogu da imaju različitu podršku za: datum/vreme, valutu, poštanske kodove, predstavljanje imena i adresa, itd.

3.4. Generisanje izveštaja

Često postoji potreba da aplikacija poseduje mogućnost generisanja izveštaja o podacima koje je sakupila ili proizvela. Takav zadatak može da bude obiman i kompleksan, pa zahteva posebnu komponentu ili novu aplikaciju, koja će se samo time baviti.

Osnovne faze su:

- Prikupljanje podataka

  Ulazni podaci mogu biti iz same aplikacije ili iz nekog drugog izvora.

Količina podataka može biti velika, tako da posebnu pažnju treba obratiti na njihov prijem i skladištenje.
Nekada su mogućnosti aplikacije ograničene za učestali prijem podataka, pa se javlja potreba za privremeni smeštaj u redovima (engl. queue), dok aplikacija ne bude mogla da ih prihvati.
Postoji dostupan veliki broj alata razvijenih za ovu namenu.

- Procesiranje podataka

Često se javlja potreba da se prikupljeni podaci obrade i pripreme u odgovarajući format, pre nego što budu dostupni za prikaz.

U zavisnosti od vrste podataka i namene, procesiranja mogu da budu dosta kompleksna i zahtevaju primenu raznih algoritama, od kojih su sve popularniji i algoritmi mašinskog učenja.
Postoji dosta alata razvijenih i u ovu svrhu. Najveći izazov u njihovom korišćenju, predstavlja prepoznavanje i definisanje ulaznog i izlaznog skupa podataka u/iz algoritma u konkretnom slučaju.

- Prikazivanje podataka

Prikazivanje ili vizuelizacija podataka je namenjena krajnjem korisniku i treba da mu pruži potrebne informacije, u čitljivom i jasno vidljivom obliku.

Postoji veliki broj alata koji prihvataju ulazne podatke u formi dokumenta ili poseduju mogućnost pristupa skladištu, kao što je baza podataka. Izlazni podaci mogu biti predstavljeni u formi: teksta, tabela, raznih dijagrama ili na mapama. Mnogi od njih nude mogućnost krajnjem korisniku da sam dizajnira izveštaje, definisanjem obrazaca.

4. Zahtevi za prototipsku aplikaciju

U narednom delu biće predstavljena server-klijent aplikacija, koja predstavlja samo prototip kompleksne industrijske aplikacije. Industrijska aplikacija bi mogla da ima višenamensku upotrebu u raznim oblastima, gde postoji potreba da se dobiju određene informacije o objektima, koji se mogu opisati tačkama u prostoru i pripadaju određenim prostornim segmentima.

Prototipska aplikacija se može koristiti da se za zadatu lokaciju, pronađu dostupne informacije o određenim apotekama, na teritoriji oko zadate lokacije. Lokacije su zapravo geo-tačke. Geo-tačke su predstavljene u 2D prostoru i raspoređene u grupe po odgovarajućim kriterijumima, korišćenjem kvadratnog stabla.

Aplikacija je napisana u Java (serverski deo) i JavaScript (klijentski deo) programskom jeziku. Osnovne tehnologije koje su korišćene u izradi su: Spring (Boot, Security, MVC, Data), AngularJS, MongDB i Maven, uz korišćenje JHipster (ili „Java Hipster“) generatora aplikacije. Detaljniji opis implementacije i upotrebe aplikacije nalazi se u narednim poglavljima.

4.1. Razvojno okruženje aplikacije

Aplikacija je razvijena korišćenjem JHipster generatora aplikacije, verzije 3.4.0.
Potrebno je instalirati implementacije sledećih tehnologija:

- Javu 8 [94]
- Git [95]
- Node.js [96]
- Yeoman, Bower, Gulp i JHipster se instaliraju pokretanjem odgovarajućih komandi, praćenjem uputstva [97].

Prototipska aplikacija je razvijena u Windows 8 okruženju, primenom prethodno navedenog uputstva JHipstera za lokalnu instalaciju. Korišćene su sledeće verzije implementacija tehnologija i alata:

- jdk-8u77-windows-x64
- Git-2.6.3-64-bit
- node-v5.2.0-x64
- mongodb-win32-x86_64-2008plus-ssl-3.0.4-signed

Kôd je razvijen u Eclipse Luna okruženju, koje ima podršku za Javu 8: eclipse-jee-luna-SR2-win32-x86_64 [98].
Za rad sa bazom podataka, korišćen je grafički alat: mongochef-x64 [99].
Za održavanje kôda, korišćen je alat Git sa lokalnim skladištem. Upravljanje alatom je rađeno iz Eclipse okruženja, korišćenjem odgovarajućeg priključka za rad sa Git skladištem: *Eclipse* Egit 3.4.2 [100].
Java kôd je stilizovan, korišćenjem šablona definisanih u direktorijumu: *config/eclipse* projekta.

Nakon što se projekat učita u Eclipse okruženje, potrebno je pratiti sledeće instrukcije:

- Sistemska Java biblioteka treba da bude Java 8 JDK
- Nivo kompajliranja projekta treba da bude 1.8
- Podići MongoDB bazu sledećim komandama (iz Command Prompt programa):
  > cd $PUTANJA_DO_MONGODB_INSTALACIONOG_DIREKTORIJUMA\bin
  > mongod --storageEngine mmapv1 –dbpath PUTANJA_DO_MONGODB_DATA_DIREKTORIJUMA
- Odraditi sledeće Maven komande:
  mvn clean
  mvn install
- Po završetku kompajliranja, aplikacija se može pokrenuti tako što se klasa `MasterRadApp` pokrene kao Java aplikacija
- Klijentski deo aplikacije je dostupan u veb pregledaču na adresi:
  http://localhost:8080

4.2. Tehnološki zahtevi u okruženju korisnika

Proizvod prototipske aplikacije je "war" datoteka.

Neće biti primenjena složena procedura instalacije, kao što se radi za produkcijska okruženja (sekcija 2.6.). Stoga, okruženje za korišćenje prototipske aplikacije nije zahtevno u smislu potrebnih tehnologija.

Prilikom kompajliranja aplikacije, kreiraju se dve "war" datoteke [101]. Jedna od njih sadrži ugrađen Tomcat 8 [102], što joj omogućava da bude izvršiva, bez potrebe da se dodatno instalira veb server.

Da bi se aplikacija pokrenula, potrebno je:

- Pokrenuti bazu (kao što je već opisano u 4.1.)
- Pokrenuti izvršnu datoteku master-rad-0.0.1-SNAPSHOT.war sledećim komandama:
  >cd $PUTANJA_DO_DATOTEKE
  >java -jar master-rad-0.0.1-SNAPSHOT.war --spring.profiles.active=dev

5. Tehnički aspekti prototipske aplikacije

Aplikacija je implementirana korišćenjem JHipster generatora aplikacije. Serverski deo je napisan u Javi, korišćenjem uglavnom Spring okvira i njegovih proširenja. Klijentska strana koristi AngularJS sa priključcima, HTML 5 i Bootstrap CSS.

Neki od ostalih korišćenih alata i okvira su:

- Kao alat za upravljanje serverskom stranom aplikacije, korišćen je Maven (sekcija 3.1.5.).
  Prilikom generisanja aplikacije, kreirana su i dva Maven profila: "dev" i "prod", kako bi se različita podešavanja koristila u razvoju i produkciji. Aplikacija je razvijana u "dev" profilu. Profili su podržani od strane Spring Boot okvira [65].
- Na klijentskoj strani aplikacije koriste se Bower i Gulp (sekcija 3.1.5.)
- Za postavljanje vrednosti parametara koji se upotrebljavaju u aplikaciji i mogu da se menjaju nezavisno od programskog kôda, korišćen je Spring Boot okvir [66]. Parametri su definisani u YAML [67] datoteci. Za svaki od profila, postoji posebna datoteka: *application-dev.yml* i *appllication-prod.yml*. Takođe, posebna datoteka postoji i za testove: *application.yml*.
- Baza podataka je MongoDB (sekcija 3.1.4). Za rad sa bazom korišćen je okvir Spring Data MongoDB (sekcija 3.1.3.).
- Zapisi u dnevniku (engl. logs), kao i njihova metrika, opisani su u sekciji 5.5.6.
- Upravljanje greškama na serverskoj strani je podržano od strane Spring okvira. Za greške nastale tokom obrade zahteva od klijenta, upisuje se odgovarajući kôd greške u zaglavlje odgovora. Opis kôda se čuva u odgovarajućim **.json* datotekama na klijentskoj strani aplikacije. Za njihov prikaz korisniku, zadužen je *AlertService*. Ovo važi ne samo za greške, nego i za poruke o izvršenoj akciji dobijene od servera.
- Metrika sistema i dokumentacija opisani su u sekciji 5.5.
- Alati i okviri korišćeni za testove navedeni su u sekciji 5.7.

Aplikacija je prototipska i zbog jednostavnosti napravljen je izbor monolitne arhitekture (sekcija 2.2.).

Serverski i kilijentski deo su nezavisni, ali smešteni pod isti projekat čiji je krajnji proizvod za instalaciju, "war" datoteka (više o instalaciji u sekciji 4).

Serverski kôd je podeljen u pakete po funkcionalnostima: entiteti, skladišta, domenski servisi, konfiguracije, sigurnost podataka, upravljanje greškama, REST kontroleri.

Inicijalno generisana serverska aplikacija od strane JHipster-a, poširena je sledećim:

- Generisan je geo-objekat entitet (sekcija 5.1.)
- Implementirana je funkcionalnost prostorne raspodele i pronalaženja geo-objekata (sekcija 5.2.)
- Registrovan je servis za slanje elektronske pošte (sekcija 5.4.)

U napisanom kôdu se koriste „Uzorci dizajna" i principi pisanja" čistog kôda", opisani u sekcijama 3.1.1.1. i 3.1.1.2, redom.

Klijentski kôd je organizovan po principima pisanja "čistog kôda" za AngularJS, kao što je spomenuto u sekciji 3.1.2.

Inicijalno generisana klijentska aplikacija od strane JHipster-a, poširena je sledećim:

- Izmenjen je izgled početne strane. Slika je preuzeta iz kolekcije sajta: `https://www.dreamstime.com/`.
- Uklonjen je JHipster logo sa jezička (engl. Tab) veb pregledača (engl. favicon).
- Promenjen je logo geo-objekta da označava apoteku.
- Dodata je nova strana Maps (biće opisano u sekciji 5.3.).
- Dodata je podrška za srpski jezik (opisaće se u sekciji 5.6.).

Proširivost aplikacije može da se ogleda u sledećem:

- Moguće je nadograditi verziju JHipstera [68] i tako nadograditi funkcionalnosti novim, ili nadograditi verzije postojećih biblioteka i okvira.
- Takođe je moguće nadograđivati biblioteke i okvire manuelno. Njihove verzije su registrovane u *pom.xml* datoteci za serverski deo i *bower.json*, odnosno *package.json*, za klijentski deo aplikacije.
- Umesto MongoDB baze, moguće je koristiti neku drugu bazu podataka. Aplikacija je napisana u slojevima, pa bi se zamenio sloj Entiteta i skladišta.
- Proširivost definicije entiteta i dodavanje novih, opisana je u sekciji 5.1.
- Proširivost funkcionalnosti geo-prostorne raspodele i pronalaska objekata, opisana je u sekciji 5.2.

5.1. Geo-objekti

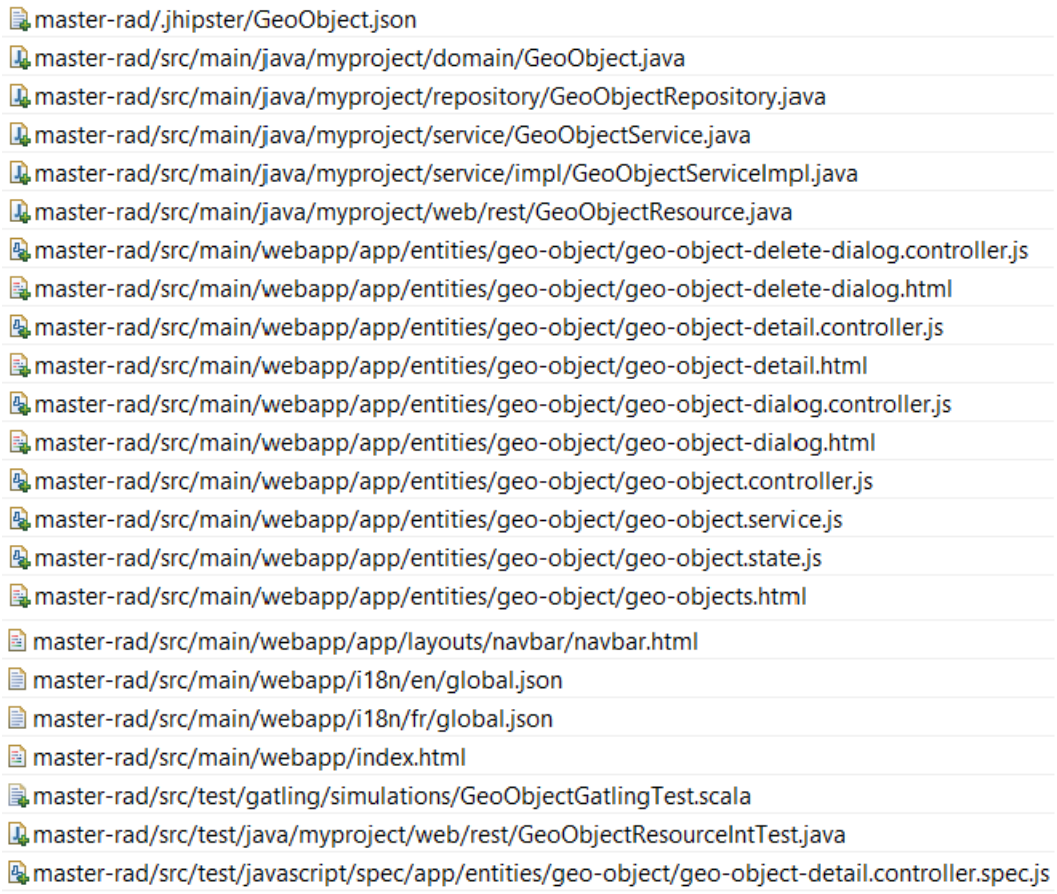
Geo-objekat je entitet koji predstavlja tačku u 2D prostoru.
U konkretnoj primeni aplikacije, ovim objektom je predstavljen entitet apoteka.

Geo-objekat je definisan uz pomoć JHipster generatora aplikacije primenjivanjem sledeće komande:

```
>cd PUTANJA_DO_PROJEKTA
>yo jhipster:entity GeoObject
```

Proces kreiranja je interaktivan. Korisnik definiše: kolekciju, atribute, tip atributa, validaciju i druge karakteristike, prema mogućnostima opisanim u uputstvu [69].

Tom prilikom su generisane sledeće datoteke:



Na serverskoj strani, kreirane su sledeće klase:

- `GeoObject`
  Predstavlja tačku u geografskom koordinatnom sistemu, opisanu odgovarajućim atributima:
  - ime
  - adresa
  - koordinate u geografskom koordinatnom sistemu (u daljem tekstu geo-koordinate): latituda (geografska širina) i longituda (geografska dužina)
- `GeoObjectRepository`
  Implementirana je korišćenjem Spring Data MongoDB okvira za rad sa bazom podataka.
  Između ostalog, podržava i paginaciju, što je korišćeno za dohvatanje svih objekata iz baze i njihov prikaz u tabeli na klijentskoj strani. Paginacija je lepo objašnjena u dokumentaciji Spring okvira [70].
- `GeoObjectService`
  Servis za upravljanje geo-objektima – čuvanje, ažuriranje, brisanje i dohvatanje iz baze.
- `GeoObjectServiceImpl`
  Implementacija servisa iz prethodne tačke.
  Inicijalna implementacija od strane JHipstera je izmenjena, tako da se:
  - Prilikom čuvanja, ažuriranja i brisanja geo-objekta, ažurira i odgovarajuće skladište implementiranog geo-stabla.
  - Prilikom traženja određenog geo-objekta, najpre se potraži u skladištu geo-stabla. Ukoliko se iz nekog razloga ne nađe, potražiće se u bazi podataka.

- `GeoObjectResource`
  Predstavlja REST kontroler, implementiran korišćenjem Spring okvira. Korišćen je od strane klijentskog dela aplikacije, za upravljanje geo-objektima. Podržava metode: kreiranja, brisanja, ažuriranja geo-objekta i dohvatanje konkretnog ili svih dostupnih geo-objekata. Za implementacije svojih metoda poziva servis `GeoObjectService`.

Osim test klasa, ostale datoteke su: JavaScript, Html i Json tipa, na klijentskoj strani, kreirane ili ažurirane u svrhu nastanka nove stranice za upravljanje geo-objektima.

Nakon kreiranja inicijalnih datoteka, moguće je ponovo generisati isti entitet, pozivanjem iste komande sa istim ili izmenjenim podešavanjima. Takođe je moguće manuelno menjati generisane datoteke.

Kreiranje objekta zahteva postavljanje vrednosti za sve atribute.
Osiguranje od greške, kako se ne bi dozvolilo kreiranje objekta bez nekog od definisanih polja, postignuto je:

- U serverskom delu aplikacije:
  U klasi `GeoObject`, postavljanjem zabeleške *@NotNull*, *javax* paketa, na svaki od atributa.
- U klijentskom delu aplikacije:
  AngularJS *required* atributom, *ngModel*-a [71], u *geo-object-dialog.html* datoteci.

Takođe je prisutna validacija atributa: nije moguće zadati više od 100 karaktera za ime i adresu; latituda je u opsegu [41, 47]; longituda je u opsegu: [18, 23].
Validacija je postignuta postavljanjem odgovarajućih zabeleški na atribute u serverskom delu klase `GeoObject`. U klijentskom delu, postavljanjem odgovarajućih AngularJS *ngModel* atributa.

Prednost ovakvog pristupa, u kreiranju objekata, je ušteda vremena za kreiranje osnovnih datoteka. Za kompleksnije zahteve, ili po potrebi, može se koristiti tradicionalni pristup manuelnog kodiranja.

5.2. Raspodela i pronalaženje geo-objekata

5.2.1. Osnovni pojmovi

Osnovni pojmovi, korišćeni u implementaciji raspodele i pronalaženja geo-objekata, su:

- Geo-tačka
  Predstavlja tačku u geografskom koordinatnom sistemu.
  Opisana je sa dva atributa: latituda i longituda. Visina (engl. elevation) nije razmatrana, radi pojednostavljenja algoritma.
  U kôdu je predstavljena klasom: `GeoPoint`.

- 3D tačka u Dekartovom koordinatnom sistemu
  Opisana je sa tri koordinate: x, y i z.
  Koristi se za konverziju iz geografskog koordinatnog sistema u Dekartov i obratno, prilikom računanja srednje tačke.
  U kôdu je predstavljena klasom: `Cartesian3Dpoint`.
- Geo-krug
  Opisan je sa: latitudom i longitudom, koje predstavljaju centar kruga i radijusom, koji predstavlja poluprečnik.
  U kôdu je predstavljena klasom: `GeoCircle`.
- Geo-prostor
  Predstavlja pravougaonik, opisan sa četiri geo-koordinate: južna latituda, severna latituda, zapadna longituda i istočna longituda.
  U kôdu je predstavljen klasom: `GeoRectArea`.
  Aplikacija postavlja teritoriju Srbije kao podrazumevani geo-prostor, na kojem će se moći koristiti funkcionalnost aplikacije – pronalaženje dostupnih apoteka, u određenoj oblasti.

Geo-tačke (objekti) se unose u sistem, kao što je opisano u poglavlju 5.1.

5.2.2. Raspoređivanje geo-objekata

Za raspoređivanje geo-tačaka koristi se kvadratno stablo.
Napomena:
U okviru rada se ne razmatra skalabilnost i optimalnost rešenja u primeni u velikim sistemima. U praksi se često koriste druge vrste stabala za slične primene, kao što je R stablo. Takođe, postoje baze podataka koje podržavaju prostorne pretrage. Jedna od njih je i MongoDB. Za inspiraciju u implementaciji poslužili su članci [72], [73].

Osnovne klase korišćene u implementaciji stabla su:

- `GeoNode` – čvor stabla.
  Sadrži sledeće informacije:
  - Geo-prostor čvora - objekat klase `GeoRectArea`.
  - Težište čvora - objekat klase `GeoPoint`, središnja tačka geo-prostora.
  - Niz od četiri objekta klase `GeoNode`, koji predstavljaju pokazivače na čvorove potomke.
- `GeoTreeStorage` – skladište objekata raspoređenih u drvetu.
  Samo lišće drveta može da sadrži geo-objekte.
  Jedan geo-objekat može da pripada samo jednom listu.
  List može da nema objekata, a najveći broj objekata, koji može da sadrži, je podesiv preko konfiguracionog fajla. U pitanju je parametar `objNumInNode`, čija se vrednost može postaviti u datoteci *application-dev.yml*. Inicijalna vrednost je postavljena na 50. Ukoliko se izostavi postavka ovog parametra, aplikacija definiše podrazumevanu vrednost konstantom `MAX_GEO_OBJECTS_NUM_PER_LEAF`, koja je postavljena na vrednost 100.
  Zbog potrebe pristupa objektima prema određenom kriterijumu, održavaju se dva skladišta:

- `geoObjectsInLeaves` – sadrži informacije o geo-objektu i pridruženom čvoru listu, kojima se pristupa preko identifikatora geo-objekta.
- `leavesWithGeoObjects` – za zadati identifikator čvora lista, može se dobiti lista identifikatora pridruženih geo-objekata.

- `GeoTreeService` – servis za kreiranje i korišćenje drveta.
- `GeoTreeServiceImpl` - implementacija servisa iz prethodne tačke.
  Implementira metodu `createAndInitializeGeoTree()`, za kreiranje i inicijalizaciju drveta.
  Implementira i metode za ažuriranje skladišta drveta, nakon što se doda, izmeni ili obriše geo-objekat. (`addGeoObject()`, `removeGeoObject()`).
  Takođe, implementira metodu `findGeoObjectsFromCircle()`, za pronalaženje geo-objekata u zadatom prostoru, koji je definisan krugom.
- `GeoTreeResource` – REST Controller, koji implementira POST metodu, za dohvatanje geo-objekata u zadatom prostoru. Metoda koristi poziv prethodno opisanog servisa. Ovaj REST Controller se poziva od strane serivisa `GeoMapService`, definisanog na klijentskoj strani aplikacije.
- `GeoLibrary` – pomoćna klasa koja definiše sve potrebne metode za operacije u geo-koordinatnom sistemu.

Kreiranje drveta je potrebno obaviti samo jednom, prilikom podizanja aplikacije. To je postignuto pozivanjem metode `createAndInitializeGeoTre()`, servisa `GeoTreeService`, u `main()` metodi klase `MasterRadApp`, kreirane od strane Spring Boot-a. Klasa je opisana u poglavlju 3.1.3.

Koren stabla se inicijalizuje na podrazumevani geo-prostor, teritoriju Republike Srbije. Geo-koordinate podrazumevanog prostora su podesive i mogu se postaviti u konfiguracionu datoteku *application-dev.yml*.
Napomena: Zbog korišćene implementacije određivanja težišta, podržan je geo-prostor samo sa pozitivnim koordinatama.

Nakon kreiranja korena stabla, čitaju se postojeći geo-objekti iz baze podataka. Za svaki od njih, pokušava se pronaći odgovarajuće mesto u stablu, na sledeći način:

- Za svaki geo-objekat se polazi od korena stabla.
- Ukoliko čvor ima lišće, proverava se kom listu geo-tačka pripada. U proveri se koristi metoda `isGeoPointInGeoRectArea()` klase `GeoLibrary`.
- Ako ne pripada ni jednom listu, geo-objekat se neće smestiti u drvo i ispisaće se odgovarajuća poruka o grešci.
- Ako se pronađe odgovarajući list, postupak se rekurzivno poziva za taj list i geo-objekat.
- Geo-objekat se pridruži čvoru. Pridruživanje se sastoji u dodavanju informacija u pomenuta skladišta: `geoObjectsInLeaves` i `leavesWithGeoObjects`.
- Proverava se da li je ispunjen kriterijum za podelu čvora na decu čvorove. Kriterijum podele je najveći broj objekata koji list može da sadrži, kako je ranije opisano.
- Ako kriterijum nije ispunjen, smeštanje se završava.
- Ako je kriterijum podele ispunjen, čvor se deli na četiri nova čvora, koja predstavljaju decu čvorove. Podela se radi u odnosu na težište čvora. Težište se određuje pozivom metode `getGeoRectCentroid()`, klase `GeoLibrary`.

- Nakon podele se radi razdruživanje geo-objekata od roditelja čvora i njihovo pridruživanje odgovarajućem detetu čvoru.

Klasa `GeoLibrary` implementira sledeće javne metode:

- Metode za validaciju latituda, longituda, radijusa i kruga (`validLatitude()`, `validLongitude()`, `validRadius()`, `validGeoCircle()`)
- Metodu za pronalaženje težišta geo-prostora (`getGeoRectCentroid()`) [74]
- Metodu koja ispituje da li je tačka u zadatom prostoru (`isGeoPointInGeoRectArea()`)
- Metodu koja ispitije da li je tačka u zadatom krugu (`isGeoPointInCricle()`)
- Metodu koja ispituje da li je krug u zadatom prostoru (`isGeoCircleInGeoRectArea()`)
- Metodu koja ispituje da li geo-prostor seče krug (`isGeoRectIntersectGeoCircle()`)
- Metodu koja određuje rastojanje između dve geo-tačke u prostoru, izraženo u kilometrima (`getGeoDistance()`) [75].

5.2.3. Pronalaženje geo-objekata

Za pronalaženje geo-objekata u zadatoj oblasti, koristi se metoda `findGeoObjectsFromCircle()`, klase `GeoTreeServiceImpl` .
Oblast u kojoj se traže geo-objekti, predstavljena je ranije opisanim geo-krugom (klasa `GeoCircle`).
Algoritam pronalaženja geo-objekta je sledeći:

- Pronađe se najmanji čvor drveta oko zadatog kruga, ili, ukoliko takav ne postoji, najveći čvor koji ga seče ili je u njemu.
- Ako ne postoji takav čvor, vrati se prazna lista geo-objekata.
- Ako takav čvor postoji, pronađu se svi listovi datog čvora, pošto su geo-objekti smešteni u njima.
- Ako listovi postoje, filtriraju se na skup onih koji se nalaze u krugu ili ga seku.
- Ako nema listova, čvor je list.
- Rezultat predstavljaju geo-objekti lista/listova koji pripadaju krugu.

5.2.4. Proširivost funkcionalnosti

Proširivost implementirane funkcionalnosti se ogleda u:

- U konkretnom slučaju, radi se raspodela i pronalaženje objekata koji su apoteke, ali je vrlo lako zameniti objekat sa bilo kojim druge vrste. `GeoObject` bi mogao da postane interfejs ili apstraktna klasa, koju bi nasledio objekat apoteka i drugi potrebni objekti. S obzirom da je implementacija za raspodelu i pronalaženje objekata nezavisna od konkretne vrste objekta, ona bi ostala neizmenjena.
- Aplikacija implementira jedno drvo za apoteke (engl. Singleton, sekcija 3.1.1.2.). Vrlo lako bi se moglo dodati kreiranje više drveta po potrebi, za različite objekte i/ili kriterijume raspodele. To bi bio “fabrika uzorak”. (engl. Factory pattern, sekcija

3.1.1.2.). Implementacija pojedinačnog drveta bi ostala ista, uz izmenu specifičnih delova koji se tiču vrste objekta i kriterijuma raspodele.

- Raspodela po drugom kriterijumu, podrazumevala bi minimalne izmene – izmenu implementacije metode `ifDivisionCriteriaMet()`, klase `GeoNode`.
- Operacije nad geo-objektima u geografskom koordinatnom sistemu, u zavisnosti od potrebe, mogle bi da imaju drugačiju implementaciju. Dovoljno je zameniti implementaciju odgovarajuće metode drugom, u klasi `GeoLibrary`. Takođe bi se cela klasa mogla zameniti nekom drugom bibliotekom, koja bi mogla da bude i biblioteka otvorenog kôda.
- Sistemu za raspodelu i pronalaženje geo-objekata se pristupa preko `GeoTreeService` interfejsa, čiju implementaciju predstavlja opisana `GeoTreeServiceImpl` klasa. Implementaciju je moguće zameniti i bilo kojom drugom. Moguće je koristiti druge vrste stabla ili bazu podataka direktno, kao što je objašnjeno u ranije navedenoj fusnoti. Klijentski deo aplikacije i REST kontroler bi ostali nepromenjeni.
- U arhitekturi zasnovanoj na mikroservisima, ovaj sistem bi mogao lako da se izdvoji u zasebnu komponentu, jer je ceo smešten u paketu *service/geospatial*.

5.3. Mapa

Iako JHipster navodi [76] Google map biblioteku [77] u listi podržanih modula [78], ona se ne može koristiti sa verzijom 3.4.0 JHipstera, korišćenom u aplikaciji, zbog poznatih problema [79].

Za kreiranje stranice „Mapa“, korišćen je priključak AngularJS okvira: angular-google-map [80].

Za instalaciju biblioteke, korišćen je alat Bower (sekcija 3.1.5.).

Potrebno je pokrenuti sledeću komandu iz direktorijuma u kome se nalazi *bower.json* datoteka projekta, kako bi se instalirala poslednja verzija biblioteke:

bower install angular-google-maps --save

Da bi se biblioteka koristila, potreban je ključ, koji se generiše za registrovanog korisnika na Google sajtu, praćenjem uputstva zvanične stranice [81]. Ključ je potom potrebno registrovati u JavaScript kôdu, što je učinjeno u datoteci *app.module.js*, pri registraciji *masterRadApp* modula.

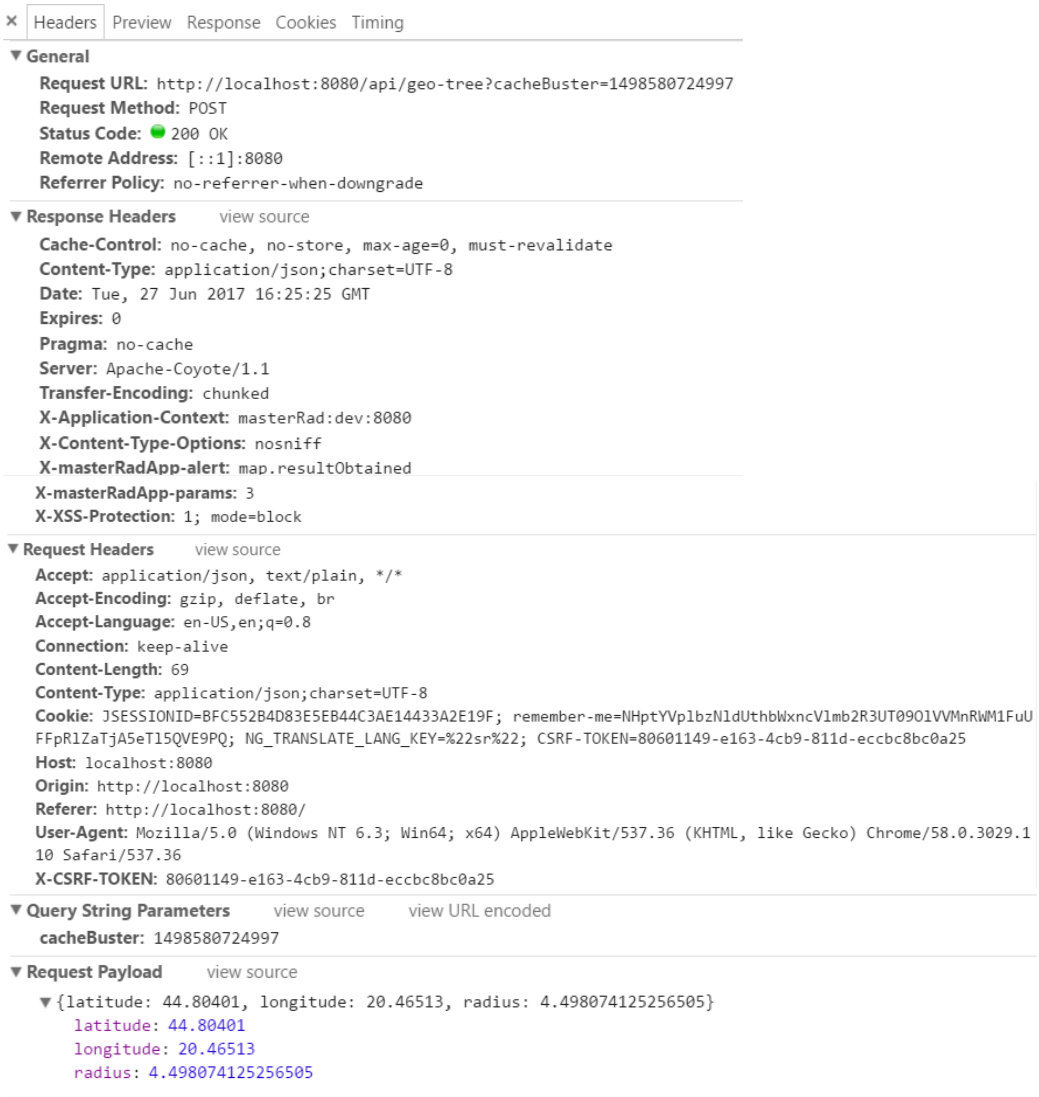
Implementacija je rađena uz korišćenje zvanične dokumentacije biblioteke [82].

Prilikom razvoja, kreirane su sledeće datoteke:

- *map.html*
  Sadrži html kôd stranice.
  Koristi AngularJS direktive za prikaz same mape, kao i njenih elemenata: markera lokacije, kruga i markera pronađenih geo-objekata.
- *map.controller.js*
  Definiše `MapController` zadužen za inicijalizaciju i upravljanje elemenatima na stranici.
  Kontroler poziva servis opisan u tački ispod, kako bi dobio zahtevane geo-objekte. Ukoliko su objekti uspešno dohvaćeni, prikazuje ih na mapi, uz odgovarajuću poruku o uspehu. Ako dođe do greške, prikazuje se odgovarajuća poruka o grešci.

- *map.service.js*
  Definiše `GeoMapService` koji implementira http POST zahtev ka `GeoTreeResource` REST kontroleru, na serverskoj strani aplikacije.
  Objekat se šalje u formatu:
  Content-Type:application/json;charset=UTF-8
  Primer: {latitude: 44.80401, longitude: 20.46513, radius: 11.439987428186967}
  Slika 5.3 prikazuje ceo zahtev, koji može da se vidi, na primer, u Chrome pregledaču, korišćenjem alata za razvoj.
- *map.state.js*
  Predstavlja ruter za Mapa stranicu. Obezbeđuje „izgled“ u zavisnosti od stanja.
- *map.json*
  Za tekstualne prikaze i poruke, postoje dve datoteke, za svaki od podržanih jezika: *en/map.json, sr/map.json*

Slika 5.3



5.4. Sigurnost aplikacije

Za sigurnost aplikacije, JHipster koristi unapređen „zapamti me“ (engl. remember me) mehanizam [83], Spring Security okvira.
Unapređenje se ogleda u sledećem:

- Moguće je da korisnik vidi listu trenutno otvorenih sesija i da ih proglasi nevažećim.
- Kada se korisnik odjavi, samo trenutna sesija se proglasi nevažećom.
- Skladište se dodatne informacije za potrebe vođenja evidencije, kao što su: IP adresa i korisnički agent (engl. user agent).

Podržane su sledeće mogućnosti za upravljanje korisničkim nalogom:

- kreiranje novog korisnika
- aktivacija naloga
- prikazivanje naloga
- izmena naloga
- promena šifre
- prikaz liste aktivnih sesija
- proglašavanje sesije navažećom

Gore pomenute mogućnosti su dostupne klijentskom delu aplikacije preko `AccountResource` REST kontrolera.

Podaci su smešteni u bazi podataka, u sledeće kolekcije:

- `jhi_user` - odgovara entitetu `User`; za svakog registrovanog korisnika postoji dokument.
- `jhi_persistent_token` - odgovara entitetu `PersistentToken` i sadrži podatke o tokenu. Tokeni se koriste od strane Spring Security okvira, kako bi se automatski prijavili korisnici.
- `jhi_persistent_audit_event` – odgovara entitetu `PersistenceAuditEvent`. Opisan je u sekciji 5.5.5.
- `jhi_authority` – odgovara entitety `Authority`. Sadrži po jedan dokument za svaku podržanu grupu korisnika sa određenim pravima korišćenja, odnosno autorizacije (engl. security role).
  a) ROLE_USER
  b) ROLE_ADMIN

Slede primeri dokumenata nekih od kolekcija:

5.4.1. Primer izgleda dokumenta koji opisuje „admin“ korisnika iz kolekcije `jhi_user`:

```
{
    "_id" : "user-2",
    "login" : "admin",
    "password" : "$2a$10$gSAhZrxMllrbgj/kkK9UceBPpChGWJA7SYIb1Mqo.n5aNLq1/oRrC",
    "first_name" : "admin",
    "last_name" : "Administrator",
    "email" : "admin@localhost",
    "activated" : "true",
    "lang_key" : "en",
    "created_by" : "system",
```

```
    "created_date" : ISODate("2016-05-28T14:33:19.271+0000"),
    "authorities" : [
        {
            "_id" : "ROLE_USER"
        },
        {
            "_id" : "ROLE_ADMIN"
        }
    ]
}
```

5.4.2. Dokument iz kolekcije `jhi_persistent_token` koji predstavlja zabeleženo prijavljivanje „admin“ korisnika u sistem:

```
{
    "_id" : ObjectId("5749b701f2aaf5706de67362"),
    "_class" : "myproject.domain.PersistentAuditEvent",
    "principal" : "admin",
    "auditEventDate" : ISODate("2016-05-28T15:19:29.789+0000"),
    "event_type" : "AUTHENTICATION_SUCCESS",
    "data" : {
        "sessionId" : "9DDB55DEA863C82CD2FA804E9FBF7369",
        "remoteAddress" : "127.0.0.1"
    }
}
```

Za potrebe aktivacije naloga, implementiran je servis `MailService` za slanje elektronske pošte, korišćenjem Spring okvira [84].
U prototipskoj aplikaciji, korišćen je Gmail servis, za šta je bilo potrebno podesiti sledeće:

- Za željeni Gmail nalog, potrebno je dozvoliti njegovo korišćenje za slanje elektronske pošte, od strane servisa koji nije Gmail, u ovom slučaju gore pomenutog `MailService` servisa. To se može uraditi praćenjem uputstva: https://support.google.com/accounts/answer/6010255?hl=en.
- Zatim je potrebno izvršiti podešavanja u *application-dev.yml* datoteci, prema JHipster uputstvu: https://jhipster.github.io/tips/011_tip_configuring_email_in_jhipster.html.

5.5. Administriranje

Dobijanje i prikazivanje statistike aplikacije i okruženja, sastavni je deo veb aplikacije generisane od strane JHipstera. Sledi opis aktivnosti na serverskoj i klijentskoj strani.

5.5.1. Prikazivanje liste svih registrovanih korisnika sistema

Na serverskoj strani, REST kontroler `UserResource` obezbeđuje listu korisnika sistema, pozivajući skladište `UserRepository`, koje koristi kolekciju `jhi_user` baze podataka.

Na klijentskoj strani, za vizuelizaciju podataka, koriste se: AngularJS kontroleri, servis i HTML kôd iz /user-management direktorijuma.

5.5.2. Metrika

Za prikupljanje: statistike aktivnosti JVM, HTTP poziva, kao i poziva REST kontrolera, serverska strana koristi Dropwizard biblioteku i integraciju sa Spring okvirom [85].
Prikazivanje statistike je podesivo postavljanjem parametara pod sekcijom `metrics`, u *application-dev.yml* datoteci, na odgovarajuću vrednost.
Prikupljanje statistike je implementirano u klasi `MetricsConfiguration`, koja nasleđuje klasu za integraciju sa Spring okvirom, `MetricsConfigurerAdapter`.
Nadgledanje REST kontrolera je postignuto korišćenjem zabeleške *@Timed*.
Inicijalizacija metrike se obavlja u klasi `WebConfigurer`, koja nasleđuje Spring Boot klase: *ServletContextInitializer* i *EmbeddedServletContainerCustomizer*. Tu se postavlja i URL adresa, koju će koristiti klijentska strana aplikacije, za pristup statističkim podacima.

Za vizuelizaciju statistike, na klijentskoj strani, koriste se: AngularJS kontroleri, servis i HTML kôd iz /metrics direktorijuma.

5.5.3. Dijagnostika

Aplikacija nadgleda i prikazuje dijagnostiku: baze podataka, servisa za elektronsku poštu i prostora na disku.
Na serverskoj strani je to postignuto na isti način kao što je opisano za metriku u sekciji 5.5.2.
Za vizuelizaciju podataka, na klijentskoj strani, koriste se: AngularJS kontroleri, servis i HTML kôd iz /health direktorijuma.

5.5.4. Podešavanja

Na aplikaciji su dostupna podešavanja vrednosti različitih promenljivih u vezi sa aplikacijom i njenim okruženjem.
Na serverskoj strani je to implementirano na način opisan u sekciji 5.5.2.
Za vizuelizaciju podešavanja, na klijentskoj strani, koriste se: AngularJS kontroler, servis i HTML kôd iz /configuration direktorijuma.

5.5.5. Prijavljivanja korisnika

Beleže se sva prijavljivanja, svih korisnika i vizuelizuje se lista prijavljivanja u određenom vremenskom periodu.
Da bi se to postiglo, na serverskoj strani se koristi klasa `SpringSecurityAuditorAware`, koja predstavlja implementaciju Spring Data Audit okvira [86].
Registrovani događaji Springove klase `AuditEvent` se čuvaju u bazi podataka, u kolekciji: `jhi_persistent_audit_event`, kojoj odgovara entitet `PersistenceAuditEvent`.

Klijentski deo aplikacije poziva REST kontroler `AuditResource`, radi dohvatanja uskladištenih podataka. Pomenuti kontroler koristi metode servisa `AuditEventService`, a on skladišta `PersistenceAuditEventRepository`.
Za vizuelizaciju dobijenih podataka, klijentska strana koristi: AngularJS kontroler, servis i HTML kôd iz /audit direktorijuma.

5.5.6. Procesi za upis u dnevnik aktivnosti

Prikazuje se lista svih procesa (engl. loggers) u sistemu, zaduženih za upis u dnevnik aktivnosti (engl. logs). Postoji mogućnost promene nivoa upisa (engl. log level) tokom rada aplikacije.

Na serverskoj strani se koristi *net.logstash.logback* biblioteka [87], verzija 4.6. REST kontroler `LogsResource` obezbeđuje izveštaje o procesima i promene nivoa izveštavanja. Njega poziva klijentska strana aplikacije radi dobijanja podataka. Za definisanje procesa koji izveštavaju, koristi se biblioteka *sl4j* [88] ugrađenja od strane Spring Boot okvira.
Klijentska strana aplikacije, za vizuelizaciju podataka, koristi: AngularJS kontroler, servis i HTML kôd iz */logs* direktorijuma.

5.5.7. Programski interfejs aplikacije

Aplikacija ima mogućnost vizuelnog prikaza implementiranog programskog interfejsa (engl. API). Funkcionalnost je moguće uključiti/isključiti postavljanjem parametra *jhipster.swagger*, u *application-dev.yml* datoteci.

Serverska strana je implementirana korišćenjem Springfox biblioteke [89].

Na klijentskoj strani je korišćena swagger-ui biblioteka [90], verzija 2.1.4.
Integracija biblioteke u aplikaciju je omogućena u datoteci *index.html*, iz direktorijuma */webapp/swagger-ui*. Stranica se pokreće iz *docs.html* fajla istoimenog direktorijuma.

5.6. Višejezičnost

Za internacionalizaciju (poglavlje 3.3.) su korišćene ekstenzije AngularJS okvira, a sama funkcionalnost je implementirana korišćenjem JHipster generatora.
JHipster ima predefinisani set jezika koji su podržani. Izbor željenijh jezika, za aplikaciju, se postiže izvršavanjem sledeće komande:
yo jhipster:languages
Prototipska aplikacija podržava dva jezika: srpski i engleski.

Srpski jezik nije u predefinisanoj listi. Njegova podrška je dodata manuelno, praćenjem uputstva JHipster dokumentacije [91]. Prevod teksta u odgovarajućim datotekama na srpski jezik je rađen iz Eclipse okruženja (sekcija 4.1.), za šta je bilo potrebno promeniti metodu

šifrovanja na UTF-8 sa podrazumevanog ISO-8859-1. Promenu je potrebno uraditi u stavci: *Window -> Preferences -> General -> Content Types*.

5.7. Testiranje

Različite vrste i pristupi u testiranju veb aplikacije, opisani su u sekciji 2.3.
JHipster generiše podrazumevane testove za serverski i klijentski deo aplikacije [92].
Osnovni testovi, koji se generišu za funkcionalnosti koje dolaze uz aplikaciju, su :

- Jedinični testovi
  U njihovoj implementaciji se koristi okvir Spring Boot JUnit.
- Integracioni testovi za REST kontrolere
  Napisani su korišćenjem Spring Boot okvira za integraciono testiranje.
  Za sve testove podiže se samo jedan Spring kontekst (engl. Spring context), sa svim objektima u njemu (engl. Spring beans).
  Podrazumevana baza podataka je ugrađena MongoDB (engl. embeded MongoDB), što je postignuto korišćenjem biblioteke *de.flapdoodle.embed.mongo* [93]. Korišćenje ove baze je opciono, tj. može se koristiti i MongoDB baza aplikacije, isključivanjem ove biblioteke u *pom.xml* datoteci i odgovarajućom konfiguracijom u *application.yml* datoteci.
- Testovi performansi
  Generišu se za svaki entitet. Napisani su korišćenjem okvira Gatling. Za Geo-objekat generisan je test *GeoObjectGatlingTest.scala*.
- AngularJS jedinični testovi
  Napisani su korišćenjem okvira Jasmine i pokreću se korišćenjem alata Karma, u pregledaču PhantomJS.
- AngularJS integracioni testovi
  Koristi se okvir Protractor.
  To su tzv. testovi s kraja na kraj (engl. end-to-end).
  Testovi će pokrenuti veb pregledač i pokušati da koriste aplikaciju kao korisnik, stoga je potrebno da se aplikacija prethodno pokrene.

Dodati su sledeći testovi za funkcionalnost geo-prostorne raspodele tačaka i njihove upotrebe:

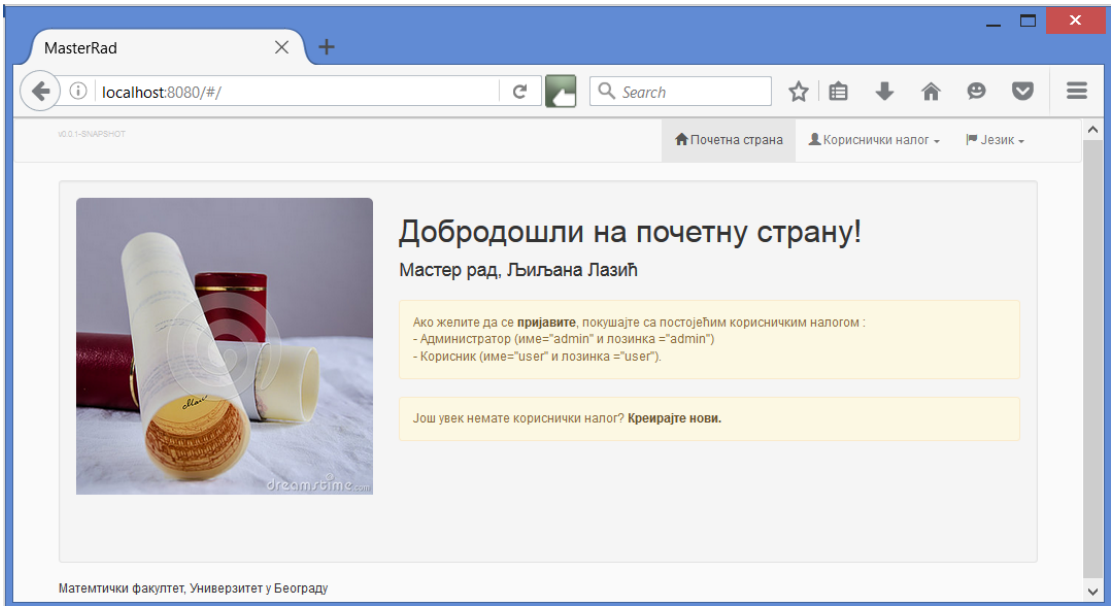
- `GeoLibraryTest`
  Sadrži jedinične testove za klasu `GeoLibrary`.
  Testovi su napisani praćenjem principa pisanja „čistih“ testova, opisanih u odeljku 3.1.1.1.
- `GeoTreeServiceTest`
  Ovo je integracioni test, kojim se proveravaju metode upotrebe geo-prostornih objekata klase `GeoTreeServiceImpl`.
- `GeoTreeResourceTest`
  Predstavlja integracioni test za REST kontroler `GeoTreeResource`.
- `GeoObjectLoaderTest`
  U pitanju je integracioni test koji kreira geo-objekte na zadatom prostoru, metodom proizvoljne raspodele. Prostor i broj tačaka su konfigurabilni. Test se koristi za učitavanje objekata u prezentacione svrhe. Unapređen, mogao bi da se koristi i za testiranje limita sistema (engl. Stress testing).

6. Korišćenje prototipske aplikacije

6.1. Početna strana i prijava na sistem

Nakon pokretanja aplikacije, pojaviće se početna strana.
Korisnik može da se prijavi sa nekim od predefinisanih korisničkih naloga ili da, praćenjem uputstva, kreira sopstveni nalog (slika 6.1.a).

Slika 6.1.a

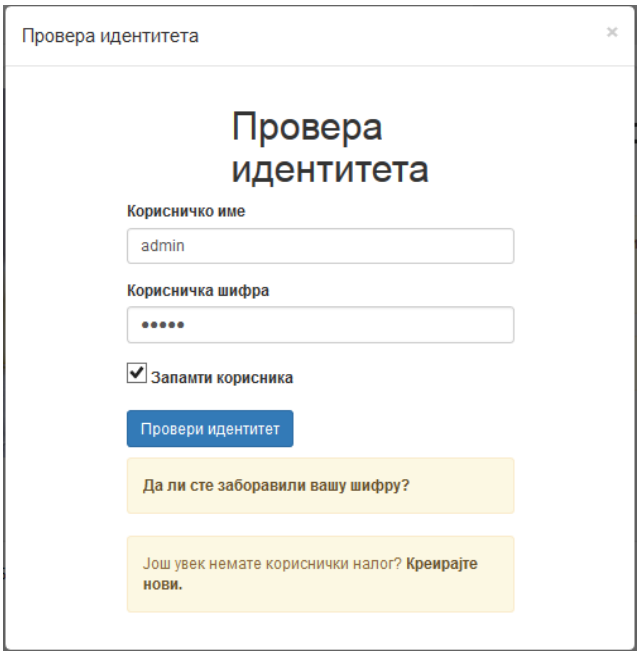


Aplikacijom se definišu dve grupe privilegija, opisane u poglavlju 6.1.

1. ROLE_USER – korisniku sa ovim privilegijama stavka „Administriranje“ neće biti dostupna.
2. ROLE_ADMIN – korisniku sa ovim privilegijama dostupne su sve stavke.

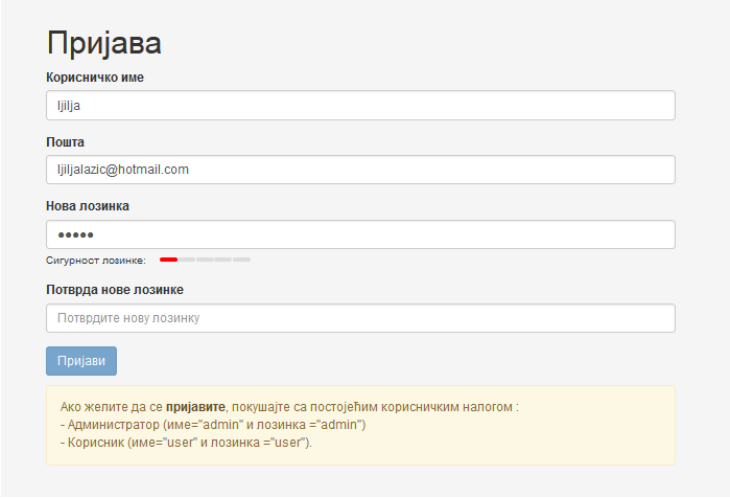
U sistemu postoje predefinisani korisnički nalozi kojima se dodeljuje neka, ili obe od ovih grupa privilegija.
1. admin / admin (slika 6.1.b)
Ovaj korisnik ima privilegije : ROLE_USER i ROLE_ADMIN.
2. user / user
Ovaj korisnik ima privilegije ROLE_USER.

Slika 6.1.b



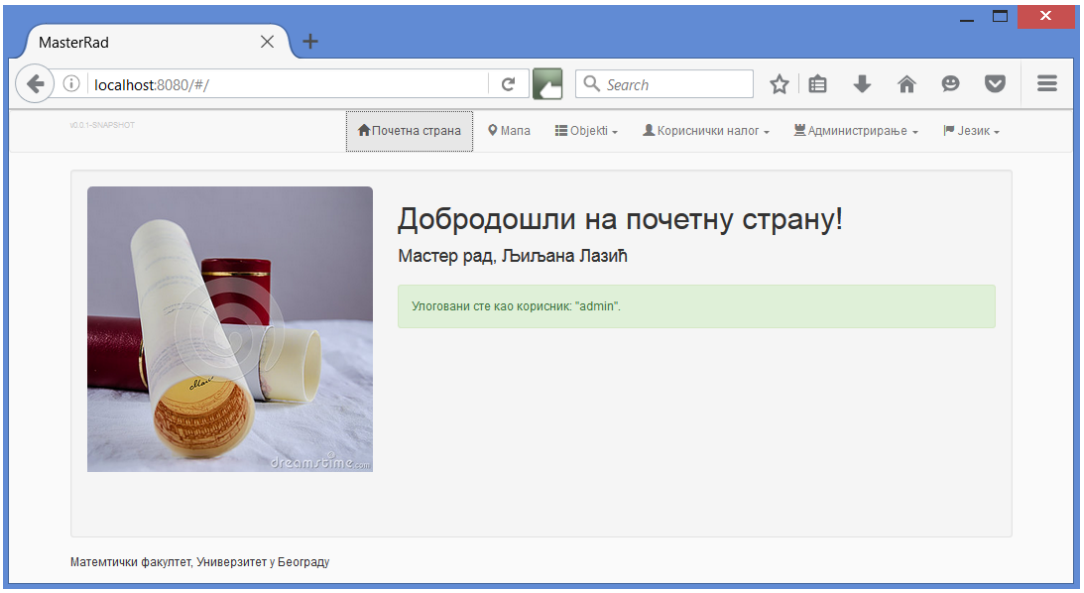
Ukoliko korisnik odabere da kreira svoj nalog, biće mu poslata e-pošta sa veb adresom za aktivaciju (slika 6.1.c).
Nalog će biti automatski aktiviran nakon pristupa poslatoj veb adresi.
Korisnik koji na ovaj način kreira svoj nalog, može da dobije najviše ROLE_USER privilegije.
Korisnika sa ROLE_ADMIN privilegijama može da kreira samo drugi korisnik sa istim privilegijama.
Ovde je poštovan sledeći princip dodeljivanja privilegija:
Korisnik sa odgovarajućim privilegijama može da kreira druge korisničke naloge sa istim ili manjim privilegijama, a ne može da kreira korisničke naloge sa većim privilegijama.

Slika 6.1.c



Nakon prijavljivanja na sistem, korisniku će biti dostupne sledeće stranice (slika 6.1.d): Objekti (Apoteka), Mapa, Korisnički nalog (Podešavanja, Šifra, Sesije, Odjava), Administriranje (Korisničko podešavanje, Statistički pokazatelji, Opšte stanje, Konfiguracija, Nadgledanje aktivnosti, Zapisi, API), Jezik (Srpski, Engleski).

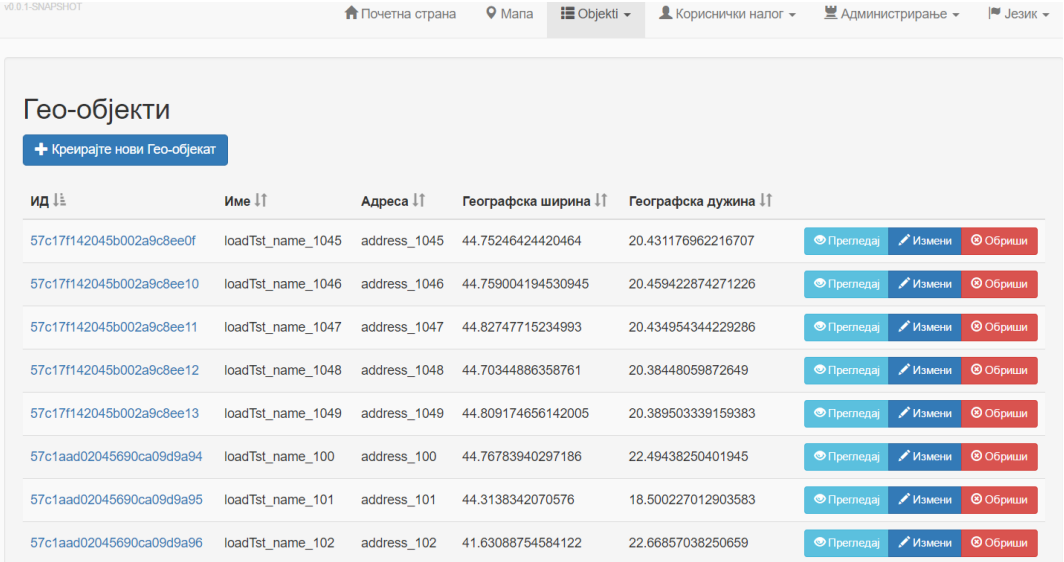
Slika 6.1.d



6.2. Objekti

Pod ovom stavkom (slika 6.2.a) se nalazi stranica “Apoteke”, sa koje je moguće upravljanje Geo-objektima, kojima su predstavljene apoteke u sistemu.

Slika 6.2.a



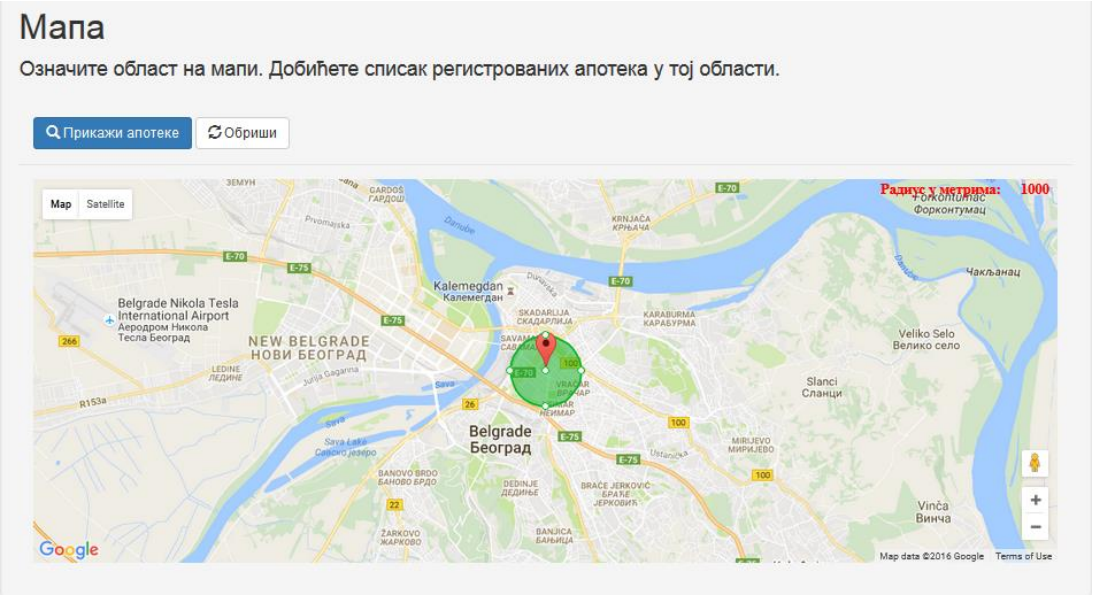
Omogućeno je kreiranje novih, ažuriranje i brisanje postojećih apoteka.
Prilikom kreiranja, obavezno je uneti sva polja: ime, adresu i lokaciju, predstavljenu geo-koordinatama (slika 6.2.b).

Slika 6.2.b

6.3. Мапа

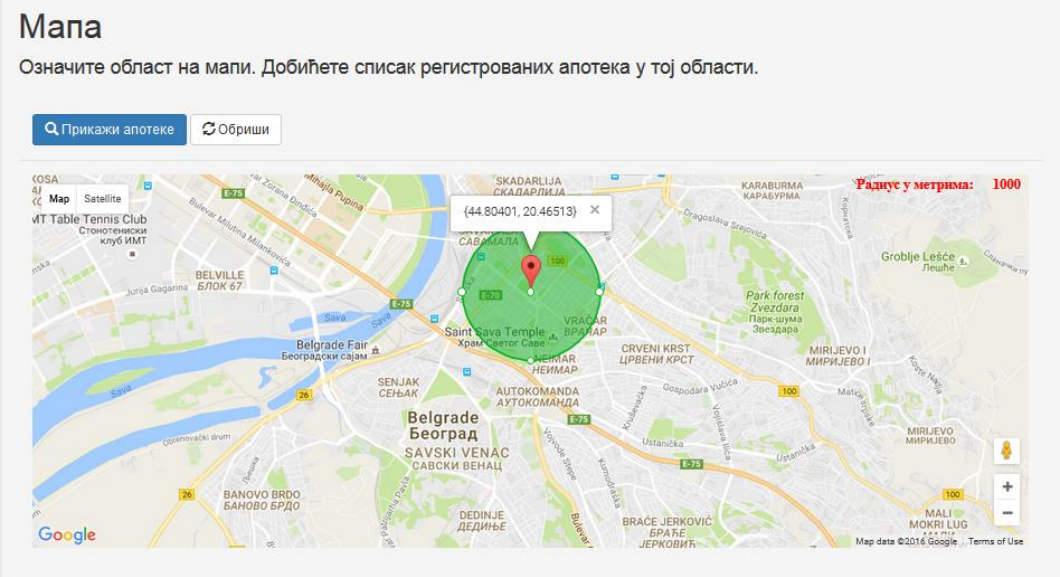
Stranica prikazuje Google mapu, sa inicijalno postavljenim crvenim markerom na teritoriji Beograda. Oko markera se nalazi zeleni krug, sa početnim radijusom od 1000m, što se može videti u gornjem, levom uglu mape (slika 6.3.a).

Slika 6.3.a



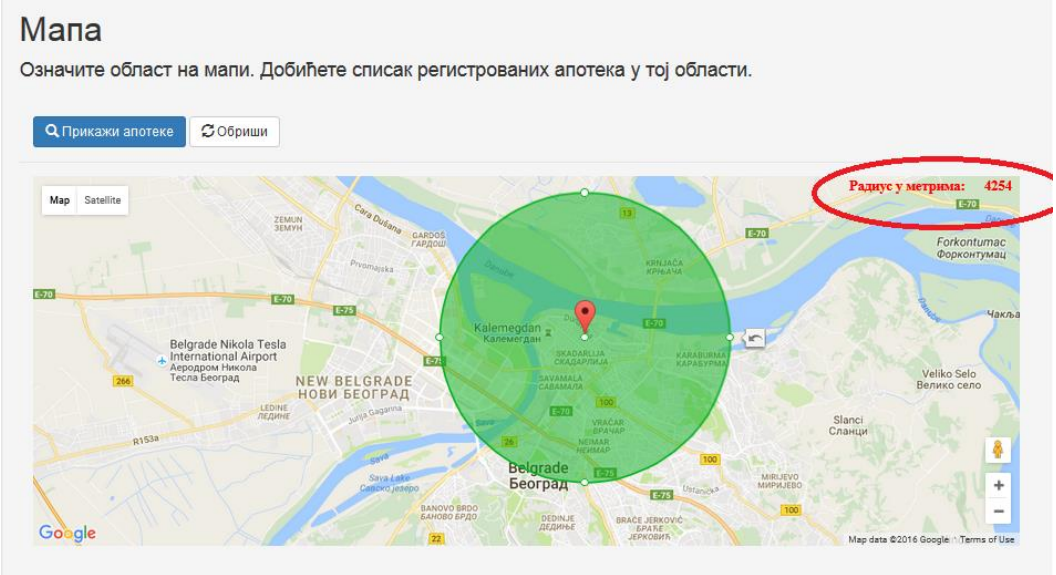
Klikom miša na crveni marker, prikazaće se njegova geo-lokacija (slika 6.3.b).

Slika 6.3.b



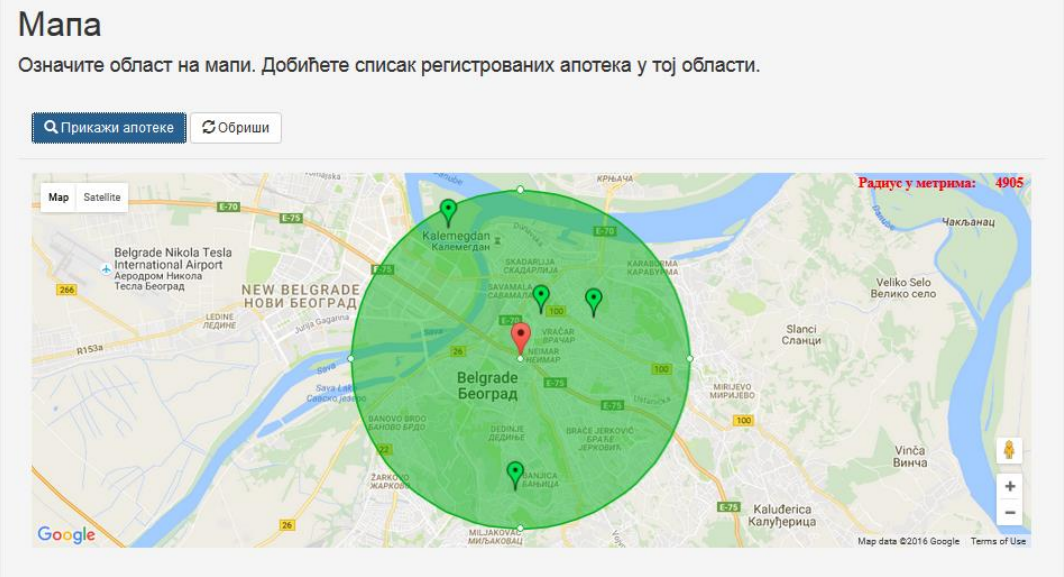
Marker sa krugom definiše oblast interesovanja za korisnika. Marker je moguće pomerati na mapi, tako što se mišem klikne na njega i zadrži, dok se ne odabere željena lokacija. Takođе je moguće menjati veličinu kruga, tako što se mišem klikne na jednu od označenih tačaka na njegovom obodu i zadrži dok se krug razvlači ili skuplja. Oznaka u gornjem desnom uglu mape će automatski biti ažurirana novim poluprečnikom kruga (slika 6.3.c).

Slika 6.3.c



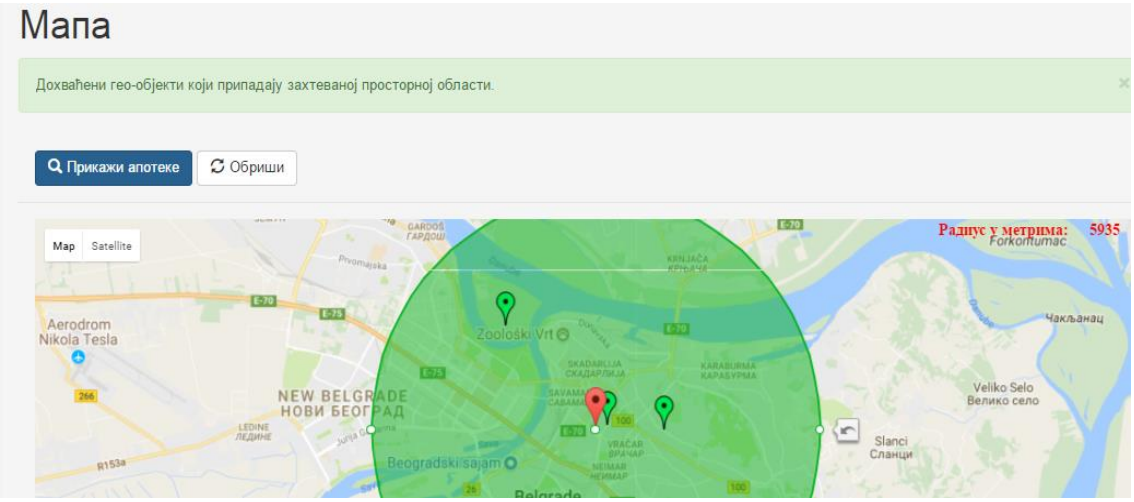
Klikom na dugme “Prikaži Apoteke”, zelenim markerima će biti prikazane sve apoteke registrovane u sistemu, koje pripadaju oblasti označenoj zelenim krugom (slika 6.3.d).

Slika 6.3.d



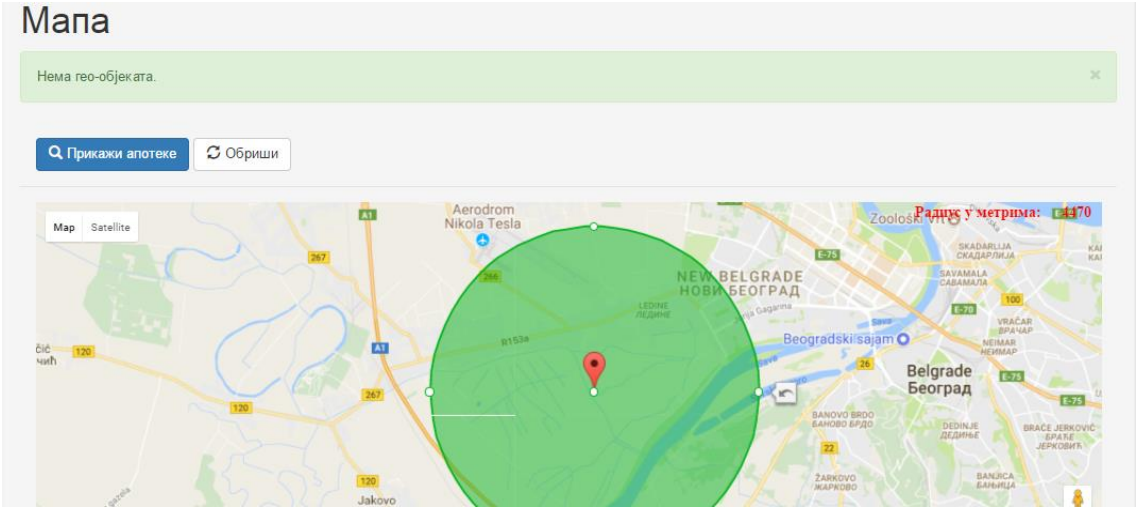
Ako je dohvatanje i prikazivanje registrovanih apoteka prošlo bez greške, korisniku će biti ispisana kratkotrajna poruka: "Dohvaćeni geo-objekti koji pripadaju zahtevanoj prostornoj oblasti." (slika 6.3.e).

Slika 6.3.e



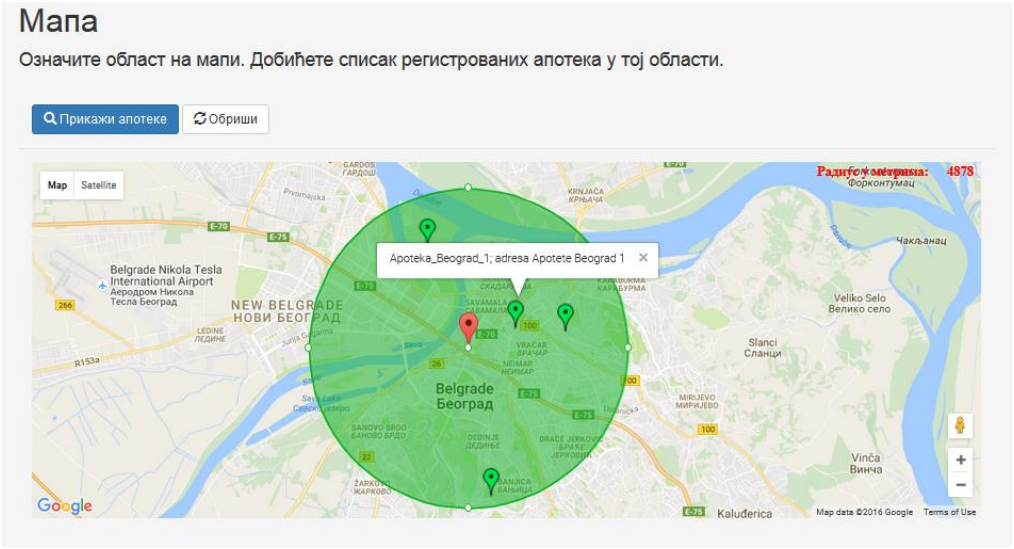
Ukoliko nema objekata, ta informacija će biti naznačena u poruci (slika 6.3.f).

Slika 6.3.f



Такође, ukoliko je došlo do greške, odgovarajuća poruka biće prikazana.
Klikom miša na neku od registrovanih apoteka, dobiće se informacija o toj apoteci – ime i adresa (slika 6.3.g).

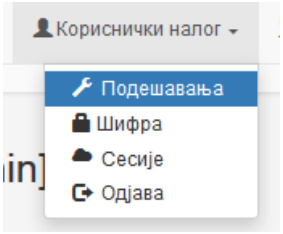
Slika 6.3.g



6.4. Korisnički nalog

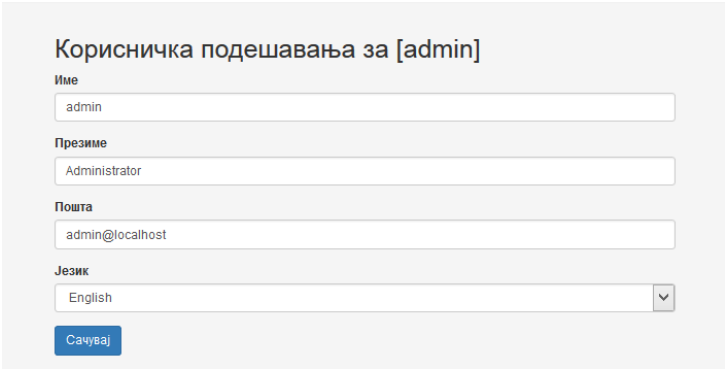
Moguće je odabrati jednu od ponuđenih stavki (slika 6.4.a) za upravljanje korisničkim nalogom.

Slika 6.4.a



Stavka “Podešavanja” omogućava postavljanje ili ažuriranje nekog od postojećih atributa prijavljenog korisnika (slika 6.4.b).

Slika 6.4.b



Stavka “Šifra” omogućava izmenu tekuće šifre prijavljenog korisnika (slika 6.4.c).

Slika 6.4.c

Stavka “Sesije” prikazuje listu aktivnih sesija prijavljenog korisnika. Klikom miša na dugme “Proglasi nevažećom” moguće je poništiti sesiju (slika 6.4.d).

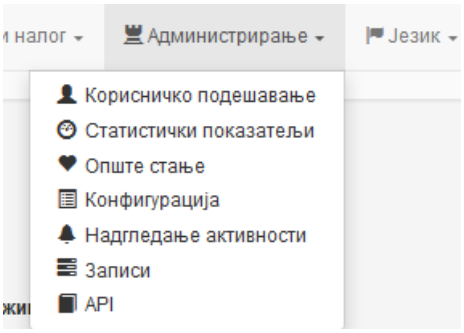
Slika 6.4.d

Klikom miša na stavku "Odjava", prijavljeni korisnik se odjavljuje iz sistema.

6.5. Administriranje

Izborom jedne od ponuđenih opcija, moguće je dobiti uvid u stanje aplikacije i sistema (slika 6.5).

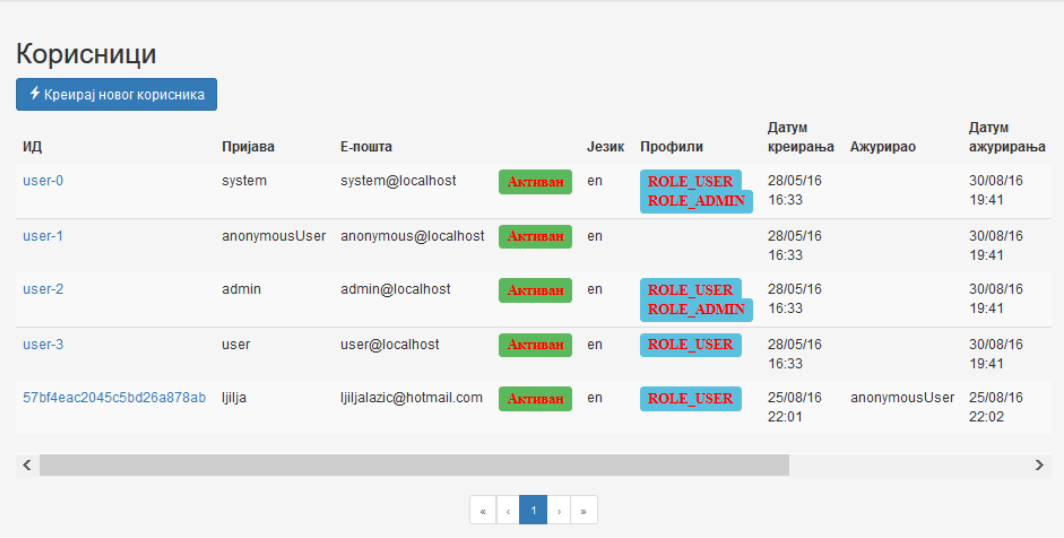
Slika 6.5



6.5.1. Korisničko podešavanje

Izborom stavke Korisničko podešavanje, prikazuje se lista postojećih korisnika u sistemu (slika 6.5.1). Takođe je moguće kreirati novog korisnika, sa istim ili nižim pravima pristupa i korišćenja aplikacije od prijavljenog korisnika, kao što je opisano u poglavlju 6.1.

Slika 6.5.1

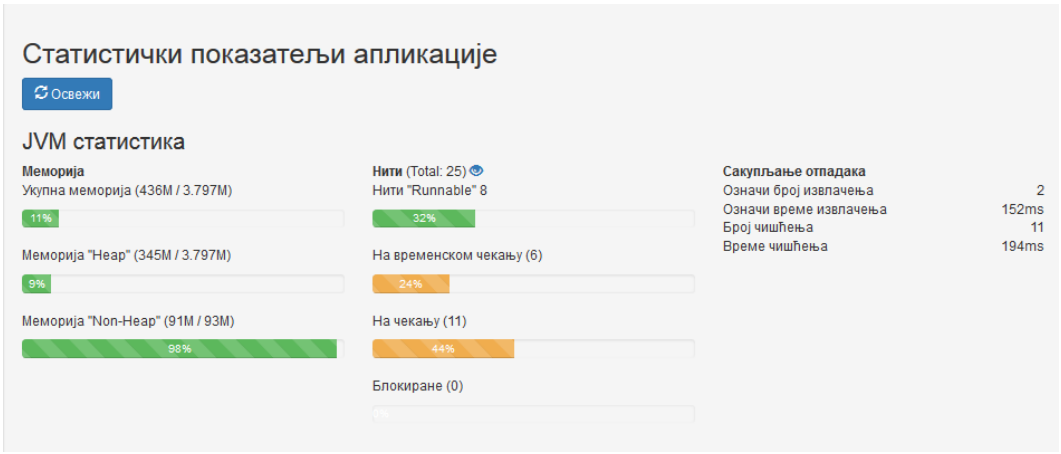


6.5.2. Statistički pokazatelji

Odabirom ove stavke, mogu se videti statistički podaci o aplikaciji i okruženju u kome se ona izvršava:

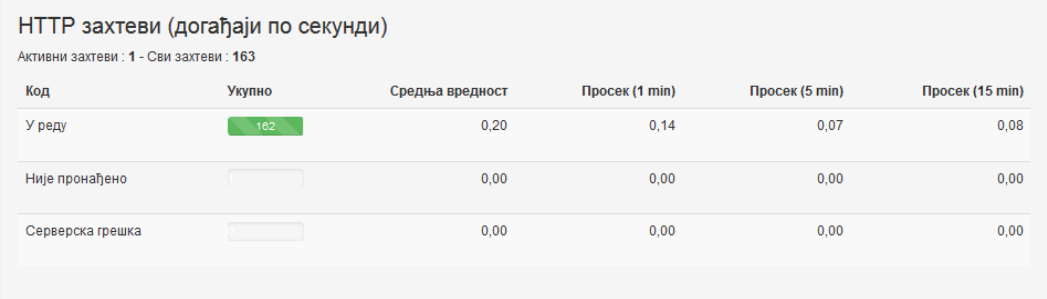
a) Podaci o Java virtuelnoj mašini – o memoriji, nitima i sakupljanju otpadaka (engl. garbage collections) (slika 6.5.2.a).

Slika 6.5.2.a



b) Podaci o HTTP zahtevima prema aplikaciji (slika 6.5.2.b).

Slika 6.5.2.b



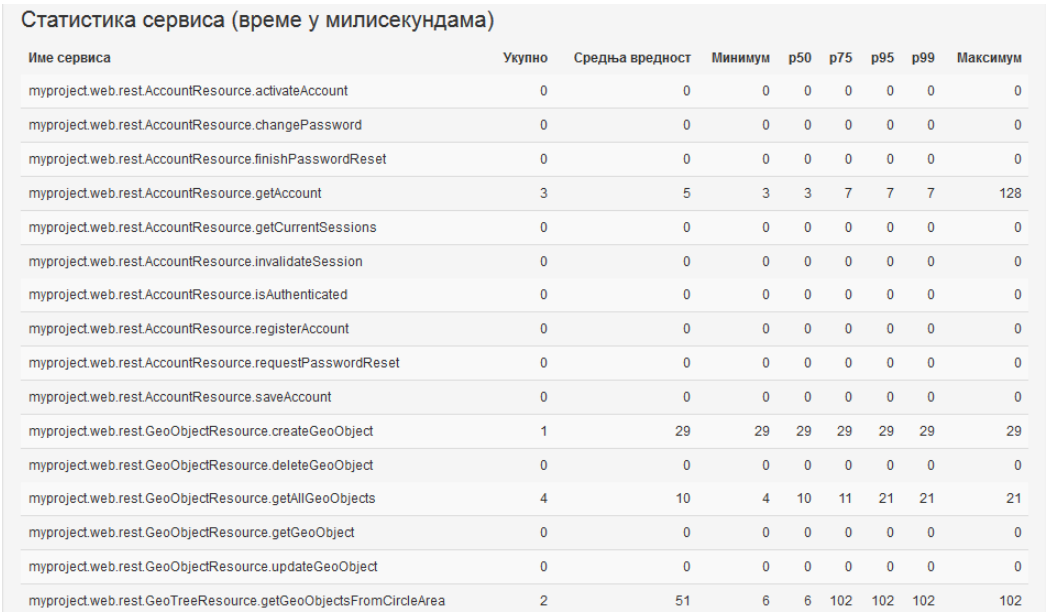
HTTP захтеви (догађаји по секунди)

Активни захтеви : **1** - Сви захтеви : **163**

Код	Укупно	Средња вредност	Просек (1 min)	Просек (5 min)	Просек (15 min)
У реду	162	0,20	0,14	0,07	0,08
Није пронађено		0,00	0,00	0,00	0,00
Серверска грешка		0,00	0,00	0,00	0,00

c) Podaci o pozivima metoda REST kontrolera (slika 6.5.2.c).

Slika 6.5.2.c



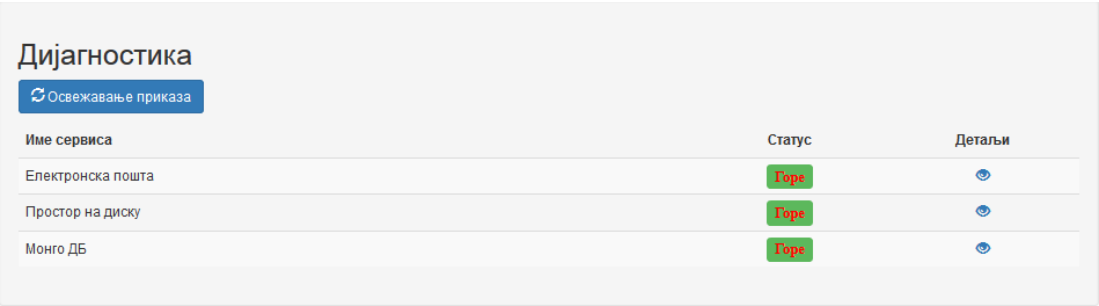
Статистика сервиса (време у милисекундама)

Име сервиса	Укупно	Средња вредност	Минимум	p50	p75	p95	p99	Максимум
myproject.web.rest.AccountResource.activateAccount	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.changePassword	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.finishPasswordReset	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.getAccount	3	5	3	3	7	7	7	128
myproject.web.rest.AccountResource.getCurrentSessions	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.invalidateSession	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.isAuthenticated	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.registerAccount	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.requestPasswordReset	0	0	0	0	0	0	0	0
myproject.web.rest.AccountResource.saveAccount	0	0	0	0	0	0	0	0
myproject.web.rest.GeoObjectResource.createGeoObject	1	29	29	29	29	29	29	29
myproject.web.rest.GeoObjectResource.deleteGeoObject	0	0	0	0	0	0	0	0
myproject.web.rest.GeoObjectResource.getAllGeoObjects	4	10	4	10	11	21	21	21
myproject.web.rest.GeoObjectResource.getGeoObject	0	0	0	0	0	0	0	0
myproject.web.rest.GeoObjectResource.updateGeoObject	0	0	0	0	0	0	0	0
myproject.web.rest.GeoTreeResource.getGeoObjectsFromCircleArea	2	51	6	6	102	102	102	102

6.5.3. Opšte stanje

Odabirom ove stavke, prikazuje se dijagnostika: baze podataka, servisa za elektronsku poštu i prostora na disku (slika 6.5.3).

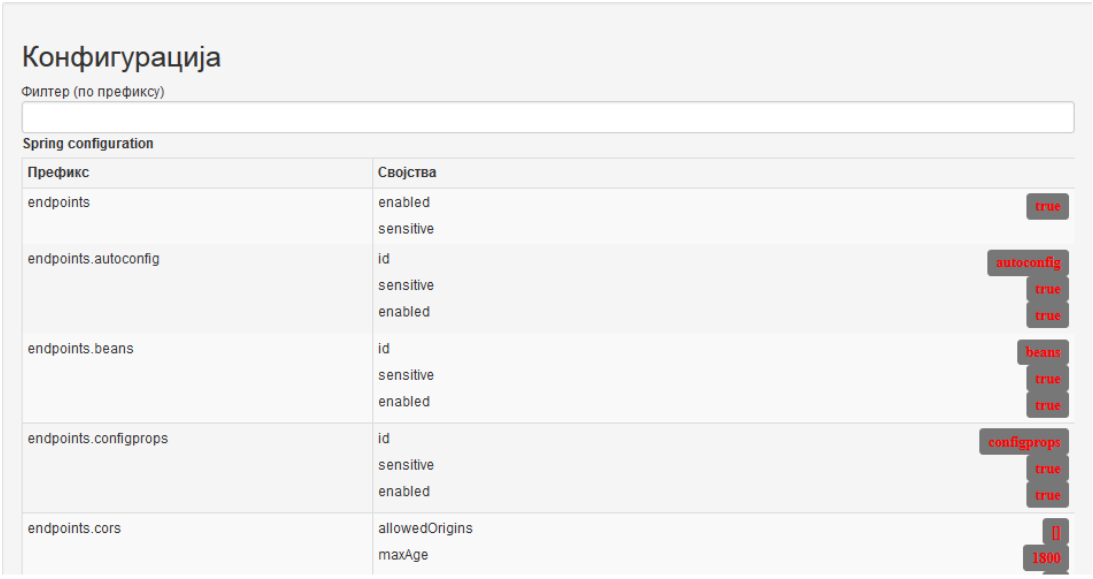
Slika 6.5.3



6.5.4. Konfiguracija

Pod ovom stavkom, mogu se videti vrednosti raznih podešavanja u vezi sa aplikacijom i okruženjem (slika 6.5.4).

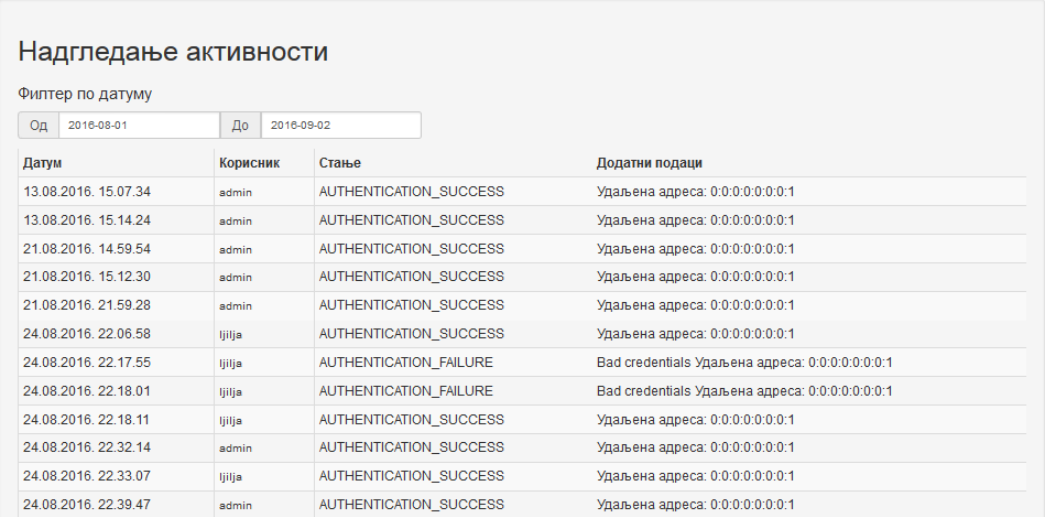
Slika 6.5.4



6.5.5. Nadgledanje aktivnosti

Odabirom ove stavke, može se videti lista svih prijavljivanja korisnika, u određenom vremenskom periodu (slika 6.5.5).

Slika 6.5.5



6.5.6. Dnevnik aktivnosti

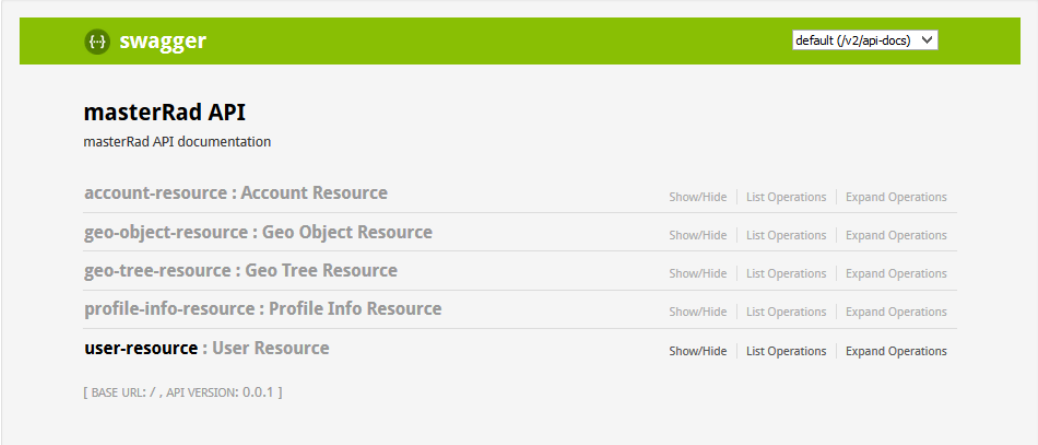
Ovde se mogu videti zapisane aktivnosti svih procesa u sistemu. Takođe je moguće menjati nivo zapisivanja dok je aplikacija pokrenuta (slika 6.5.6).

Slika 6.5.6

6.5.7. API

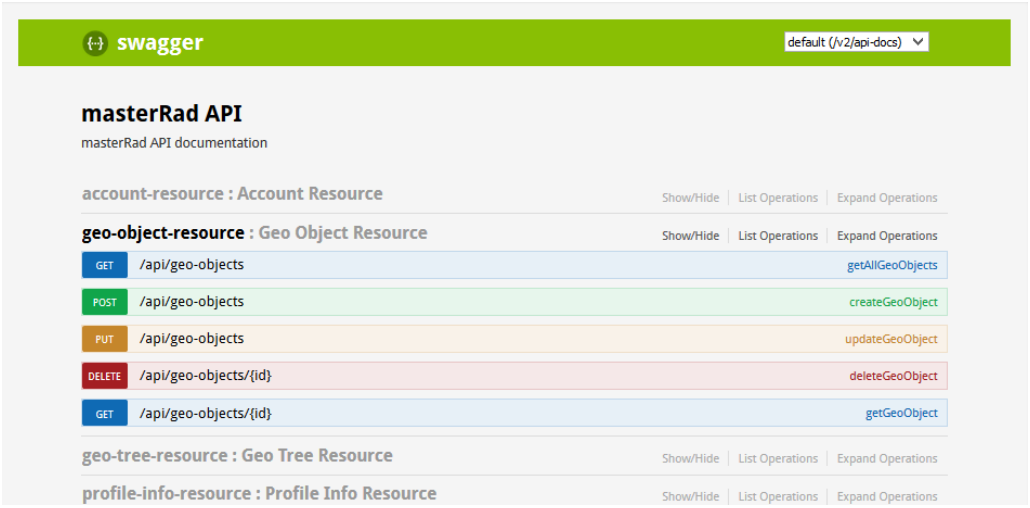
Pod stavkom Programski interfejs aplikacije, može se videti dokumentacija postojećih REST resursa (slika 6.5.7.a).

Slika 6.5.7.a



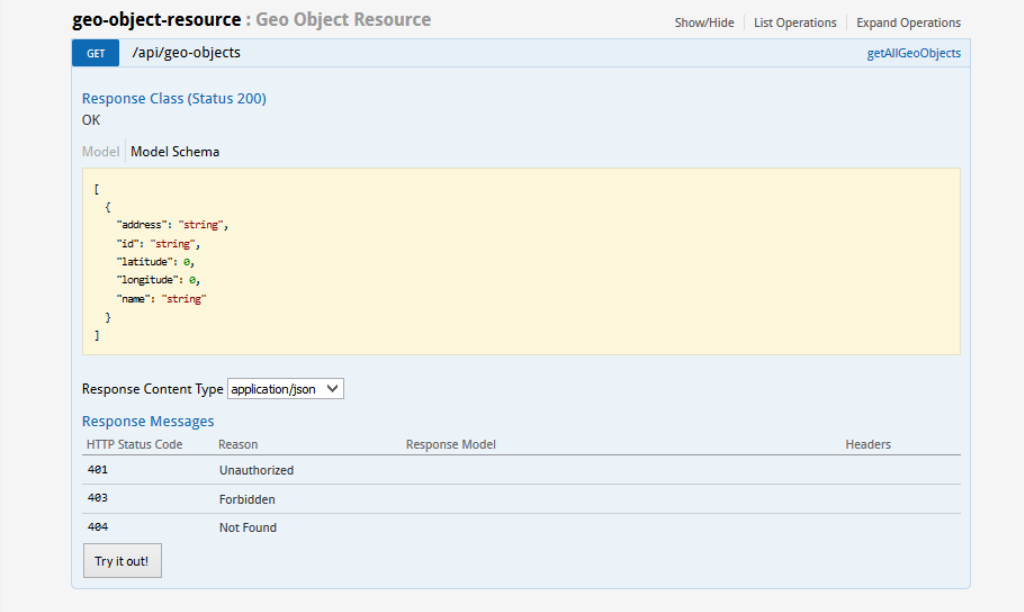
Takođe se mogu videti detalji koji prikazuju, za svaki REST resurs: njegove metode, HTTP putanje do njih i vrstu zahteva (GET, POST, DELETE) (slika 6.5.7.b).

Slika 6.5.7.b



Za svaku od metoda, na raspolaganju je više detalja kao što su: opis ulaznih i izlaznih parametara, zaglavlja HTTP zahteva, tela HTTP zahteva i podržani HTTP status kodovi (slika 6.5.7.c).

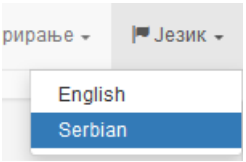
Slika 6.5.7.c



6.6. Višejezičnost

Korisnik može da izabere jedan od dva podržana jezika: Engleski ili Srpski (slika 6.6). Tekst i poruke aplikacije će se prikazivati na odabranom jeziku.

Slika 6.6



7. Zaključak

Razvoju aplikacije treba pristupiti studiozno, počev od postavljanja zahteva, dizajna i izbora tehnologija, do završne faze, procesa održavanja aplikacije. Svaku od ovih faza treba prilagoditi potrebama i nameni konkretne aplikacije.

Održavanje programskog kôda „čistim“, uz upotrebu uzoraka dizajna, doprinosi ne samo efikasnijem i kvalitetnijem razvoju aplikacije, već i održavanju i proširivosti rešenja, kao i optimizaciji. Takođe, tako napisan programski kôd, predstavlja oblik svojevrsne dokumentacije projekta.

Programski jezici i tehnologije se svakodnevno usavršavaju. Veliki broj slobodnih softvera i softvera otvorenog kôda je dostupno na tržištu. Sve više su popularne konferencije, koje za cilj imaju da predstave nove tehnologije, promovišu aktuelne i buduće standarde, povežu ljude iz srodnih oblasti i podstaknu razmenu informacija i znanja.

Korišćenje dostupnih alata i okvira, može pomoći u postavljanju aplikacije i implementiranju opštih funkcionalnosti, nezavisnih od namene same aplikacije, kao što je pokazano u radu (bezbednost aplikacije, definisanje i upravljanje entitetima, višejezičnost, nadgledanje aplikacije i resursa, generisanje izveštaja i mnoge druge). Tehnologije treba pažljivo odabrati, shodno potrebama konkretne aplikacije.

Sami alati i okviri, pojedinačno, često ne pružaju kompletno rešenje, pa ih je potrebno proširiti i prilagoditi samoj aplikaciji. Alate i okvire treba nadograđivati, čime se obezbeđuje održavanje aplikacije u skladu sa aktuelnim standardima, kao i mogućnost proširivosti za nove funkcionalnosti.

Arhitektura aplikacije treba da omogući: efikasan i kvalitetan razvoj, jednostavnu instalaciju, održavanje, upotrebu i proširenja. Proizvod treba da odgovori postavljenim zahtevima. Nadogradnja ili zamena alata i okvira, na nekoj komponenti, treba da bude moguća, bez većeg uticaja na ostatak sistema.

Literatura

[1] A. Abel / F. Marinescu, „Domain-Driven Design Quickly, A summary of Eric Evan's Domain-Driven Design," InfoQ, 8 December 2006. [Na mreži]. Dostupno: https://www.infoq.com/minibooks/domain-driven-design-quickly. [Poslednji pristup 4 September 2016].

[2] R. Fielding / J. Reschke, „Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," HTTP workinig group, June 2014. [Na mreži]. Dostupno: http://httpwg.org/specs/rfc7230.html. [Poslednji pristup 7 September 2016].

[3] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures. CHAPTER 5 Representational State Transfer (REST)," 2000. [Na mreži]. Dostupno: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Poslednji pristup 7 September 2016].

[4] T. Bray / E. , „The JavaScript Object Notation (JSON) Data Interchange Format," Google, Inc, March 2014. [Na mreži]. Dostupno: https://tools.ietf.org/html/rfc7159. [Poslednji pristup 7 September 2016].

[5] C. Richardson, „Pattern: Monolithic Architecture," 2014. [Na mreži]. Dostupno: http://microservices.io/patterns/monolithic.html. [Poslednji pristup 10 September 2016].

[6] C. Richardson, „Pattern: Microservices Architecture," 2014. [Na mreži]. Dostupno: http://microservices.io/patterns/monolithic.html. [Poslednji pristup 10 September 2016].

[7] M. Takada, „Single page apps in depth, a.k.a Mixu's single page app book," 2013. [Na mreži]. Dostupno: http://singlepageappbook.com/goal.html. [Poslednji pristup 5 September 2016].

[8] „Scalability," Wikimedia Foundation, Inc, 31 March 2016. [Na mreži]. Dostupno: https://en.wikipedia.org/wiki/Scalability. [Poslednji pristup 10 September 2016].

[9] „Test-driven development," Wikimedia Foundation, Inc, 9 September 2016. [Na mreži]. Dostupno: https://en.wikipedia.org/wiki/Test-driven_development. [Poslednji pristup 10 September 2016].

[10] „Behaviour Driven Development," BDDWiki, 29 September 2014. [Na mreži]. Dostupno: http://behaviourdriven.org/. [Poslednji pristup 10 September 2016].

[11] „Java," Oracle, 2016. [Na mreži]. Dostupno: http://www.java.com. [Poslednji pristup 10 September 2016].

[12] „Tiobe index," TIOBE software BV, 2016. [Na mreži]. Dostupno: http://www.tiobe.com/tiobe-index/. [Poslednji pristup 4 September 2016].

[13] R. C. Martin, „Clean Code: A Handbook of Agile Software Craftsmanship," 13 ur., Prentice Hall, 2008.

[14] „Javadoc," Oracle and/or its affiliates, 2016. [Na mreži]. Dostupno: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html. [Poslednji pristup 4 September 2016].

[15] „JUnit," JUnit, 2016. [Na mreži]. Dostupno: http://junit.org/junit4/. [Poslednji pristup 4 September 2016].

[16] J. Kerievsky, „Refactoring to Patterns,“ Addison Wesley, 2004.

[17] „AngularJS,“ Google, 2016. [Na mreži]. Dostupno: https://angularjs.org/. [Poslednji pristup 11 September 2016].

[18] „AngularJS,“ Google, 2016. [Na mreži]. Dostupno: https://github.com/angular/angular.js. [Poslednji pristup 11 September 2016].

[19] „Using AngularJS,“ JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/using-angularjs. [Poslednji pristup 10 September 2016].

[20] „Spring,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: https://spring.io/. [Poslednji pristup 11 September 2016].

[21] Spring Framework Team, „Spring Framework Reference Documentation. Part I. Overview of Spring Framework. 2. Introduction to the Spring Framework,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html. [Poslednji pristup 12 September 2016].

[22] „Spring Boot,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://projects.spring.io/spring-boot/. [Poslednji pristup 12 September 2016].

[23] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis / S. Deleuze, „Spring Boot Reference Guide,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/. [Poslednji pristup 12 September 2016].

[24] B. Alex, L. Taylor, R. Winch / G. Hillert, „Spring Security Reference. 17. Remember-Me Authentication,“ Pivotal Software, Inc, 2015. [Na mreži]. Dostupno: http://docs.spring.io/spring-security/site/docs/4.1.3.RELEASE/reference/htmlsingle/#remember-me. [Poslednji pristup 12 September 2016].

[25] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch / A. Wilkinson, „Spring Boot Reference Guide. Part IV. Spring Boot features. 27.1 The ‘Spring Web MVC framework’,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/1.4.0.RELEASE/reference/htmlsingle/#boot-features-spring-mvc. [Poslednji pristup 12 September 2016].

[26] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch / A. Wilkinson, „Spring Boot Reference Guide. Part IV. Spring Boot features. 30.2 MongoDB,“ Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/1.4.0.RELEASE/reference/htmlsingle/#boot-features-mongodb. [Poslednji pristup 12 September 2016].

[27] „MongoDB,“ MongoDB, Inc, 2016. [Na mreži]. Dostupno: https://www.mongodb.com/. [Poslednji pristup 14 September 2016].

[28] „Relational Vs Non Relational Database,“ MongoDB, Inc, 2016. [Na mreži]. Dostupno: https://www.mongodb.com/scale/relational-vs-non-relational-database. [Poslednji pristup 14 September 2016].

[29] „MongoDB Architecture,“ MongoDB, Inc, 2016. [Na mreži]. Dostupno: https://www.mongodb.com/mongodb-architecture. [Poslednji pristup 14 September 2016].

[30] „JHipster,“ JHipster, 2016. [Na mreži]. Dostupno: https://jhipster.github.io. [Poslednji pristup 4 September 2016].

[31] „Technology stack,“ JHipster, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/tech-stack. [Poslednji pristup 4 September 2016].

[32] M. Bynens, „HTML5 Boilerplate,“ 2016. [Na mreži]. Dostupno: https://html5boilerplate.com. [Poslednji pristup 4 September 2016].

[33] M. Otto, J. / T. a. c. , „Bootstrap,“ 2016. [Na mreži]. Dostupno: http://getbootstrap.com. [Poslednji pristup 4 September 2016].

[34] The Yeoman Team, „Yeoman,“ 2016. [Na mreži]. Dostupno: http://yeoman.io/. [Poslednji pristup 4 September 2016].

[35] Bower Team, „Bower,“ 2016. [Na mreži]. Dostupno: https://bower.io/. [Poslednji pristup 4 September 2016].

[36] Gulp team and contributors, „Gulp,“ 2016. [Na mreži]. Dostupno: http://gulpjs.com. [Poslednji pristup 4 September 2016].

[37] „Maven,“ Apache, 2016. [Na mreži]. Dostupno: https://maven.apache.org/. [Poslednji pristup 11 September 2016].

[38] „Feature Summary,“ Apache, 2016. [Na mreži]. Dostupno: https://maven.apache.org/maven-features.html. [Poslednji pristup 11 September 2016].

[39] B. Alex, L. Taylor, R. Winch / G. Hillert, „Spring Security Reference. Part I. Preface,“ 2015. [Na mreži]. Dostupno: http://docs.spring.io/spring-security/site/docs/4.1.1.RELEASE/reference/htmlsingle/#preface. [Poslednji pristup 12 September 2016].

[40] „Welcome to OWASP,“ OWASP, 2016. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Main_Page. [Poslednji pristup 12 September 2016].

[41] „Category: Attack,“ OWASP, 2016. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Category:Attack. [Poslednji pristup 12 September 2016].

[42] „Category:Vulnerability,“ OWASP, 2016. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Category:Vulnerability. [Poslednji pristup 12 September 2016].

[43] „Category:Code Snippet,“ OWASP, 2011. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Category:Code_Snippet. [Poslednji pristup 12 September 2016].

[44] „Category:Threat,“ OWASP, 2007. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Category:Threat. [Poslednji pristup 12 September 2016].

[45] „Top 10 2013-Top 10,“ OWASP, 2015. [Na mreži]. Dostupno: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Poslednji pristup 12 September 2016].

[46] „The Java EE 6 Tutorial. Securing Web Applications. Specifying Authentication Mechanisms,“ Oracle, 2013. [Na mreži]. Dostupno: http://docs.oracle.com/javaee/6/tutorial/doc/gkbaa.html#bncbo. [Poslednji pristup 12 September 2016].

[47] B. Cook, P. Saint-Andre, H. Tschofenig, B. Leiba / D. Atkins, „The OAuth 2.0 Authorization Framework,“ Oracle, 2012. [Na mreži]. Dostupno: https://tools.ietf.org/html/rfc6749. [Poslednji pristup 12 September 2016].

[48] „OpenID Connect Core 1.0 incorporating errata set 1,“ OpenID Foundation, 8 November 2014. [Na mreži]. Dostupno: http://openid.net/specs/openid-connect-core-1_0.html. [Poslednji pristup 12 September 2016].

[49] „Internationalization,“ W3C, 2016. [Na mreži]. Dostupno: https://www.w3.org/standards/webdesign/i18n. [Poslednji pristup 10 September 2016].

[50] R. Ishida, S. K. Miller / Boeing, „Localization vs. Internationalization,“ W3C, 2015. [Na mreži]. Dostupno: https://www.w3.org/International/questions/qa-i18n. [Poslednji pristup 10 September 2016].

[51] „Web Testing: A Complete guide about testing web applications,“ Software Testing Help, 25 August 2015. [Na mreži]. Dostupno: http://www.softwaretestinghelp.com/web-application-testing. [Poslednji pristup 10 September 2016].

[52] „JUnit 4,“ JUnit, 2016. [Na mreži]. Dostupno: http://junit.org/junit4/. [Poslednji pristup 10 September 2016].

[53] „Jasmine,“ Pivotal Labs, 2016. [Na mreži]. Dostupno: https://github.com/jasmine/jasmine. [Poslednji pristup 10 September 2016].

[54] F. Ziegelmayer, „Karma,“ 2016. [Na mreži]. Dostupno: http://karma-runner.github.io/1.0/index.html. [Poslednji pristup 10 September 2016].

[55] A. Hidayat, M. Svay / J. Mason, „PhantomJS,“ 2016. [Na mreži]. Dostupno: http://phantomjs.org/. [Poslednji pristup 4 September 2016].

[56] Spring Framework Team, „Spring Framework Reference Documentation. Part IV. Testing. 15. Integration Testing,“ Pivotal. Spring, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/integration-testing.html. [Poslednji pristup 10 September 2016].

[57] „Protractor. Code,“ Protractor team, 2016. [Na mreži]. Dostupno: https://github.com/angular/protractor. [Poslednji pristup 10 September 2016].

[58] „Async Scala-Akka-Netty based Load Test Tool,“ Gatling Corp, 2016. [Na mreži]. Dostupno: https://github.com/gatling/gatling. [Poslednji pristup 10 September 2016].

[59] „Jenkins,“ Jenkins team and contribuotrs, 2016. [Na mreži]. Dostupno: https://jenkins.io. [Poslednji pristup 16 September 2016].

[60] „Bamboo Server,“ Atlassian, 2016. [Na mreži]. Dostupno: https://www.atlassian.com/software/bamboo. [Poslednji pristup 16 September 2016].

[61] „SonarQube,“ SonarSource S.A, 2016. [Na mreži]. Dostupno: http://www.sonarqube.org. [Poslednji pristup 16 September 2016].

[62] „What is Cloud Computing?,“ Amazon Web Services, Inc, 2016. [Na mreži]. Dostupno: https://aws.amazon.com/what-is-cloud-computing/. [Poslednji pristup 14 September 2016].

[63] A. Wiggins, „The Twelve-Factor App. III. Config. Store config in the environment,“ 2012. [Na mreži]. Dostupno: https://12factor.net/config. [Poslednji pristup 14 September 2016].

[64] „Minify Resources (HTML, CSS, and JavaScript),“ Google, 2016. [Na mreži]. Dostupno: https://developers.google.com/speed/docs/insights/MinifyResources. [Poslednji pristup 14 September 2016].

[65] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch / A. Wilkinson, „Spring Boot Reference

Guide. Part IV. Spring Boot features. 24. Externalized Configuration," Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html. [Poslednji pristup 12 September 2016].

[66] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch / A. Wilkinson, „Spring Boot Reference Guide. Part IV. Spring Boot features. 24. Externalized Configuration. 24.6 Using YAML instead of Properties," Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html. [Poslednji pristup 12 September 2016].

[67] „Yaml," YAML.org., 2006. [Na mreži]. Dostupno: http://www.yaml.org/start.html. [Poslednji pristup 11 September 2016].

[68] „Upgrading an application," JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/upgrading-an-application. [Poslednji pristup 11 September 2016].

[69] „Creating an entity," JHipster team, 2016. [Na mreži]. Dostupno: http://jhipster.github.io/creating-an-entity. [Poslednji pristup 4 September 2016].

[70] M. Pollack, T. Risberg, O. Gierke, C. Leau, J. Brisbin, T. Darimont / C. Strobl, „Spring Data MongoDB - Reference Documentation," Pivotal Software, Inc, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/. [Poslednji pristup 4 September 2016].

[71] „AngularJS. API Reference. ng. input components in ng. input," Super-powered by Google, 2016. [Na mreži]. Dostupno: https://docs.angularjs.org/api/ng/input/input%5Bnumber%5D. [Poslednji pristup 4 September 2016].

[72] S. Lambert, „Quick Tip: Use Quadtrees to Detect Likely Collisions in 2D Space," 3 September 2012. [Na mreži]. Dostupno: http://gamedevelopment.tutsplus.com/tutorials/quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374. [Poslednji pristup 8 September 2016].

[73] R. Sedgewick / K. Wayne, „QuadTree.java," 2016. [Na mreži]. Dostupno: http://algs4.cs.princeton.edu/92search/QuadTree.java.html. [Poslednji pristup 8 September 2016].

[74] „Calculation Methods," GeoMidpoint, 2016. [Na mreži]. Dostupno: http://www.geomidpoint.com/calculation.html. [Poslednji pristup 4 September 2016].

[75] A. Hedges, „Finding distances based on Latitude and Longitude," 2016. [Na mreži]. Dostupno: http://andrew.hedges.name/experiments/haversine. [Poslednji pristup 4 September 2016].

[76] T. Mottet, „Install Google maps on your JHipster application," 2015. [Na mreži]. Dostupno: https://github.com/moifort/generator-jhipster-google-maps. [Poslednji pristup 4 September 2016].

[77] „Google Maps APIs. Web. Maps JavaScript API," Google team, 2016. [Na mreži]. Dostupno: https://developers.google.com/maps/documentation/javascript/tutorial. [Poslednji pristup 4 September 2016].

[78] „Marketplace," Jhipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/modules/marketplace/#/list. [Poslednji pristup 4 September 2016].

[79] T. Mottet, „generator-jhipster-google-maps. Issues,“ 2015. [Na mreži]. Dostupno: https://github.com/moifort/generator-jhipster-google-maps/issues. [Poslednji pristup 4 September 2016].

[80] „Angular Google Maps,“ Angular Google map team with contributors, 2016. [Na mreži]. Dostupno: https://angular-ui.github.io/angular-google-maps/#!/. [Poslednji pristup 4 September 2016].

[81] „Google Maps APIs. Web. Maps JavaScript API. Get a Key/Authentication,“ Google team, 2016. [Na mreži]. Dostupno: https://developers.google.com/maps/documentation/javascript/get-api-key. [Poslednji pristup 4 September 2016].

[82] „API Documentation,“ Angular Google map team with contributors, 2016. [Na mreži]. Dostupno: https://angular-ui.github.io/angular-google-maps/#!/api/GoogleMapApi. [Poslednji pristup 4 September 2016].

[83] „Securing your application,“ JHipster team, 2016. [Na mreži]. Dostupno: http://jhipster.github.io/security. [Poslednji pristup 11 September 2016].

[84] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis / S. Deleuze, „Spring Boot Reference Guide. Part IV. Spring Boot features. 34. Sending email,“ Pivotal. Spring, 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-email.html. [Poslednji pristup 4 September 2016].

[85] „Metrics,“ Coda Hale, Yammer Inc, 2014. [Na mreži]. Dostupno: http://metrics.dropwizard.io. [Poslednji pristup 4 September 2016].

[86] M. Pollack, T. Risberg, O. Gierke, C. Leau, J. Brisbin, T. Darimont, C. Strobl / M. Paluch, „Spring Data MongoDB - Reference Documentation. 11. Auditing,“ 2016. [Na mreži]. Dostupno: http://docs.spring.io/spring-data/data-mongodb/docs/current/reference/html/#auditing. [Poslednji pristup 4 September 2016].

[87] „logstash-logback-encoder,“ Elastic.co., 2016. [Na mreži]. Dostupno: https://github.com/logstash/logstash-logback-encoder. [Poslednji pristup 4 September 2016].

[88] „Simple Logging Facade for Java (SLF4J),“ slf4j.org., 2016. [Na mreži]. Dostupno: http://www.slf4j.org/. [Poslednji pristup 4 September 2016].

[89] D. Krishnan / A. Kelly, „Springfox Reference Documentation,“ 2016. [Na mreži]. Dostupno: https://springfox.github.io/springfox/docs/current. [Poslednji pristup 4 September 2016].

[90] „swagger-ui,“ SmartBear Software, 2016. [Na mreži]. Dostupno: https://github.com/swagger-api/swagger-ui. [Poslednji pristup 4 September 2016].

[91] „Internationalization. How to add a new language that is not supported?,“ JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/installing-new-languages. [Poslednji pristup 10 September 2016].

[92] „Running tests,“ JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/running-tests. [Poslednji pristup 10 September 2016].

[93] „Embedded MongoDB,“ Organization Flapdoodle OSS, 2016. [Na mreži]. Dostupno: https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo. [Poslednji pristup 10 September 2016].

[94] „Java SE Downloads,“ Oracle, 2016. [Na mreži]. Dostupno: http://www.oracle.com/technetwork/java/javase/downloads/index.html. [Poslednji pristup 10 September 2016].

[95] „Git,“ Git team, 2016. [Na mreži]. Dostupno: https://git-scm.com. [Poslednji pristup 4 September 2016].

[96] „NodeJS,“ Node.js Foundation, 2016. [Na mreži]. Dostupno: https://nodejs.org/en/. [Poslednji pristup 4 September 2016].

[97] „Installing JHipster,“ JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/installation. [Poslednji pristup 4 September 2016].

[98] „Eclipse Luna,“ The Eclipse Foundation, 2016. [Na mreži]. Dostupno: https://eclipse.org/luna. [Poslednji pristup 4 September 2016].

[99] „3T MongoChef – The GUI for MongoDB,“ 3T Software Labs GmbH, 2016. [Na mreži]. Dostupno: http://3t.io/mongochef. [Poslednji pristup 4 September 2016].

[100] „EGit/User Guide,“ The Eclipse Foundation, 2016. [Na mreži]. Dostupno: https://wiki.eclipse.org/EGit/User_Guide. [Poslednji pristup 4 September 2016].

[101] „Using JHipster in production,“ JHipster team, 2016. [Na mreži]. Dostupno: https://jhipster.github.io/production/. [Poslednji pristup 4 September 2016].

[102] „Apache Tomcat 8,“ The Apache Software Foundation, 2016. [Na mreži]. Dostupno: http://tomcat.apache.org/tomcat-8.0-doc. [Poslednji pristup 4 September 2016].