Fakultet elektrotehnike i računarstva

Raspodijeljeni sustavi

**Microservices - Spring Cloud**

Ana Božić, Helena Tamburić, Brigita Vrbanec, Vladimir Olujić

siječanj, 2017.

Uvod 3

1. Mikroservisi 4

Mikroservisi u našoj aplikaciji 9

2. Spring Tool Suite i Spring Boot 13

3. Spring Cloud 19

4. Analiza besplatnih cloudova koji se mogu koristiti za isporuku mikrousluga 29

5. Naš projekt 38

6. Upute za instalaciju i korištenje 43

7. Zaključak 51

8. Literatura 51

# Uvod

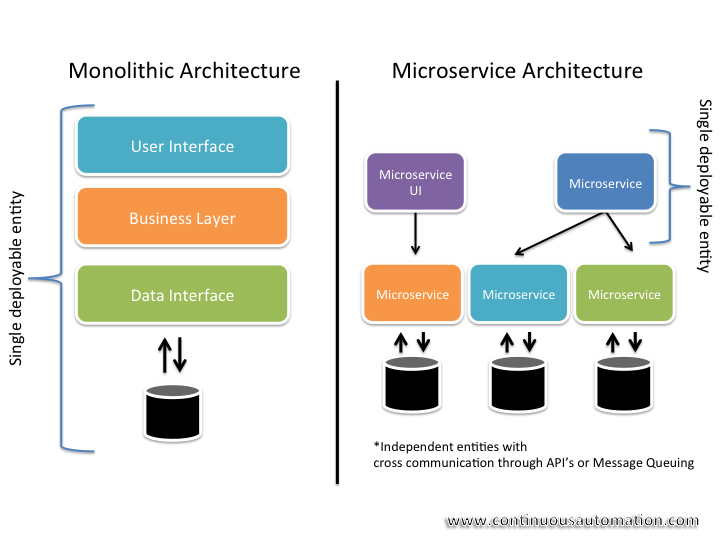
Naš zadatak je bio istražiti mikrousluge te implementaciju pomoću Spring Clouda. Potom na studijskom primjeru mikrousluga napraviti web-aplikaciju i mobilnu aplikaciju te ju isporučiti na besplatni cloud po izboru. Zadatak je bio podijeljen u nekoliko faza. Prva faza se odnosila na istraživanje mikrousluga čiji su produkt prva četiri poglavlja ove dokumentacije. U drugoj fazi projekta proučavali smo izradu jednostavnih API-ja korištenjem Spring frameworka i izvedbu mikrousluga u razvojom okruženju Spring Tools Suit (STS). Zasebno su svi članovi tima kreirali svoju mikrouslugu. U trećoj fazi projekta mikrousluge su povezane korištenjem Eureke kako bi mogle međusobno komunicirati. U četvrtoj fazi projekta bavili smo se ostvarivanjem neovisnog web servisa koji će komunicirati sa svim mikroservisima. Potom je nad web servisom izgrađen međuposrednički i prezentacijski sloj koji su zajedno sačinjavali našu aplikaciju. U petoj fazi kreiran je mobilni servis koji će komunicirati sa mikroservisima. Za razliku od template-ova on vraća sirove podatke koje će mobilna aplikacija koristiti. Mobilni servis kao i mobilna aplikacija nažalost nisu napravljeni zbog većih problema i utroška vremena prilikom susretanja sa arhitekturom mikroservisa i prevladavanja prepreka iste.

# 1. Mikroservisi

Iako sama ideja razvoja aplikacija nad arhitekturom mikroservisa nije toliko inovativna, njena popularnost konstantno raste od početka 2014. godine. Arhitektura mikroservisa potpuno je suprotna dosada najčešće primjenjivanoj monolitnoj arhitekturi i interes za nju jasno pokazuje kako velike tvrtke uočavaju brojne nedostatke monolitne arhitekture i okreću se modelima koji bolje i prirodnije opisuju poslovni model, a isto tako odgovara i manjim startup-ovima.

Arhitektura mikroservisa i monolitna arhitektura

Posebnosti arhitekture mikroservisa najlakše je uočiti u usporedbi sa, svima dobro poznatom, monolitnom arhitekturom. Usporedba ove dvije arhitekture dana je na Slici 1.



*Slika 1. Usporedba monolitne i arhitekture mikroservisa*

Monolitna arhitektura podrazumijeva izradu čitave aplikacije kao jedne cjeline. Aplikacije se uglavnom dizajniraju u tri dijela; prezentacijeski sloj na klijentskoj strani šalje zahtjeve na server koji se bavi svom pozadinskom logikom i komunicira s bazom podataka. MVC aplikacije su tipičan primjer primjene takve arhitekture. Programski kod se dijeli u klase i namespace-ove koji prate strukturu jezika domene primjene. Sva logika se izvršava u jednom procesu i jedini način za horizontalno skaliranje je pokretanje više instanci čitave aplikacije iako nisu svi dijelovi aplikacije pod jednakim opterećenjem. Drugi problem kod monolitnih aplikacija je objavljivanje novih verzija. Svaka promjena u kodu znači ponovno objavljivanje čitave aplikacije iako najčešće promjena obuhvaća samo mali dio koda. Funkcije često pozivaju funkcije drugih klasa pa tako promjena ponašanja jedne klase može negativno utjecati na druge i zahtijevati izmjene koda na više mjesta. Sve te probleme rješava arhitektura mikroservisa.

Aplikacije se dizajniraju kao skup servisa od kojih svaki obavlja sve zadatke iz svoje poddomene. Tako jedan servis ima vlastitu bazu podataka u koju sprema one podatke koji se tiču njegove poddomene, logika koju izvršava je također vezana samo uz tu poddomenu, a rezultate svog rada otvara prema drugim servisim najčešće preko API-a. Svaki servis se izvršava u svom prcesu i skaliranje je moguće postići pokretanjem više instanci istog servisa tako da se ukupni resursi sustava puno bolje raspoređuju i odgovaraju stvarnim potrebama klijenata. Objavljivanje nove verzije servisa obavlja se nezavisno o drugima, a takav pristup je također pogodan za continuous deployment. Servisi međusobno komuniciraju preko definiranih sučelja pa tako promjena logike unutar jednog servisa ne mijenja ništa u njegovom odnosu prema drugima, podaci koje šalje prema van ostaju u istom formatu i servisi ne znaju kada se koji od njih promijenio. Komunikacija preko mreže je sporija u odnosu na komunikaciju unutar jednog procesa, no jasno definirano sučelje pozitivno utječe na koheziju i spregu koda.

Servisi kao komponenete

Glavno obilježje arhitekture mikroservisa je podjela sustava na komponente pri čemu svaka komponenta bude realizirana kao jedan servis. Komponente uglavnom reflektiraju strukturu tvrtke, odnosno tima koji je izgrađuje. Nad jednom komponentom rade zajedno dizajneri, developeri i product manageri. Najčešće jedna osoba pripada samo jednom timu pa je i takva podjela sustava smislena. Granice komponenete su jasno definirane poddomenom, kao i ulogama ljudi koji na njoj rade. Veličina tima se određuje po pravilu dvije pizze; čitav tim se mora moći nahraniti sa dvije pizze.

Komunikacija među servisima

Drugo važno svojstvo arhitekture mikroservisa je komunikacija. Dok se u monolitnim aplikacijama komunikacija obavlja pozivanjem unutarnjih metoda, kod arhitekture mikroservisa koristi se drugačiji pristup. Servisi nalikuju na Unix filtere. Na ulaz servisa dolaze parametri, servis obavlja logiku nad njima i na izlaz daje podatke u definiranom formatu. Najčešće korišteni protokoli za komunikaciju su HTTP u kombinaciji s API-em i komunikacija porukama preko sabirnice. Kod prelaska s monolitne na arhitekturu mikroservisa, najveći problem leži upravo u prilagodbi komunikacije. Programeri se u tom slučaju ponekad odlučuju na pozivanje udaljenih procedura, no takvo rješenje nije u skladu s arhitekturom mikroservisa.

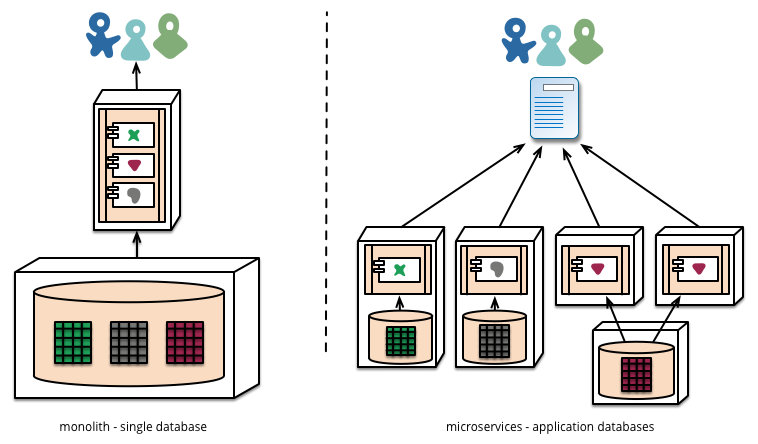
Ostaje pitanje kako prilagoditi aplikaciju da servisi funkcioniraju i u slučaju ispada jednog od njih. U tom slučaju se najčešće koriste defaultni podaci ili se dio aplikacije koji bi ih trebao koristiti ne prikazuje korisniku.

Decentralizacija

Mikroservisi implementiraju ideju domain driven designa zvanu ograničeni kontekst. Domena problema podijeljena je na više poddomena uz definirane odnose među njima. Takav decentralizirani koncept može se primijeniti kod monolitnih aplikacija, no prirodno je povezan s arhitekturom mikroservisa.

Prednost mikroservisa nad monolitnom arhitekturom leži i u činjenici da svaki servis može koristiti druge tehnologije. Tako servis za generiranje izvješća može biti implementiran u Node.js-u, a neki servis, kojem su vremenske performance prioritet, u C++-u. Ista metodologija primjenjuje se i kod upravljanja podatcima. Suprotno od monolitne arhitekture, gdje sve komponente pristupaju istoj bazi podataka, servisi imaju svaki svoju bazu podataka, a kako joj drugi neće prispupati, ona može biti drugačijeg tipa za svaki servis. Nažalost, decenralizirana baza podataka dolazi po cijeni konzistentnosti podataka. Kod monolitnih aplikacija konzistentnost se postiže transakcijama, no distribuirane transakcije se teško ostvaruju pa arhitektura mikroservisa naglašava koordinaciju servisa baz transakcija. Komunikacija se obavlja asinkrono te se ne može uvijek garanitirati da će uspjeti. Ovakav pristup je koristan kod sustava kojima je bitna responzivnost. Amazon primjerice cijeli proces narudžbe obavlja asinkrono, a ako nešto pođe krivo i narudžba ne uspije, kupac dobije kompenzaciju. Gubitci kod neuspjelih narudžbi su manji od gubitaka zbog slabe responzivnosti stranice.

Takav pristup upravljanja podacima prikazan je na Slici 2.



Slika 2. Decentralizirani pristup podatcima

Načini podjele sustava

Dobra podjela domene u kojoj sustav radi ključna je za uspješan rad mikroservisa. Dva glavna načina podjele su po glagolima ili scenarijima korištenja i podjela po imenicama ili resusrsima.Kod podjele po glagolima particije se definiraju prema radnjama koje će servisi obavljati. Tako, na primjer, jedan servis može sužiti za login, drugi za kontakt... Podjela po resursima znači da je servis obavlja sve operacije na nekim resursom sustava. Primjer takvog servisa bio bi servis za inventar trgovine koji provjerava koji su proizvodi dostupni.

Idealno, svaki servis ima mali set odgovornosti, u skladu sa single responsibility principom koji nalaže da bi klasa trebala imati samo jedan razlog za promjenu. Arhitektura mikroservisa također nalikuje na Unix koji ima set alata poput grep, find, cut od kojih svaki obavlja samo jednu funkciju i mogu se međusobno kombinirati i povezivati.

Problemi pri razvoju

Uz sve već opisane prednosti, razvoj aplikacija nad arhitekturom mikroservisa ima i određene nedostatke.

* Izdavanje novih verzija je složenije. Održavanje i upravljanje produkcijskim okruženjem sasavljenim od mnogo različitih servisa je kompliciranije nego kod monolitnih aplikacija.
* Povećana potreba za memorijom. Kako bi se instance servisa izolirale, svaki servis pokreće svoj JVM, a često se i svaki servis pokreće u svom virtualnom stroju. Takav način pokretanja i skaliranja sustava zahtijeva znatno više memorije od monolitne aplikacije.
* Razvoj je složeniji. Alati i IDE-ovi koje programeri koriste namijenjeni su razvoju monolitnih apikacija i ne prilagođavaju se dobro raspodijeljenim. Testiranje cijele aplikacije je također složenije. Komunikacija među servisima zahtijeva implementaciju komunikacijskih mehanizama, a slučajevi korištenja koji obuhvaćaju više servisa zahtijevaju kooordinaciju timova.

Poznate tvrtke koje koriste arhitekturu mikroservisa

Iako je arhitekura mikroservisa relativno nova, brojne tvrtke je već koriste i postupno prelaze s monolitne. Pioniri ove arhitekture su Amazon, Netflix, eBay, The Guardian, UK Government Digital Service, realestate.com.au, Forward i comparethemarket.com. U Hrvatskoj je najbolji primjer ovakve arhitekture Farmeron.

Netflix je popularan servis za gledanje video sadržaja te odgovoran za gotovo 30% internet prometa. Dnevno prima preko milijardu zahtjeva na svoj API s više od 800 vrsta uređaja. Svaki poziv prema API-u za prijenos sadržaja raspoređuje se na šest poziva prema backend servisima u prosjeku.

Amazon je originalno koristio dvodjelnu arhitekturu, no zbog potrebe za skaliranjem prešli su na arhitekturu mikroservisa sa nekoliko stotina servisa u backendu. Amazon.com web stranica poziva 100-150 servisa za dobivanje svih podataka potrbnih za renderiranje stranice.

eBay je prešao s monolitne na servisnu arhitekturu. Aplikacijski sloj sastoji se od više nezavisnih aplikacija. Svaka od njih implementira poslovnu logiku za određeno područje, npr. prodaju ili kupovinu. Kombinaciju x, y i z skaliranja koriste i na sloju baze podataka.

## Mikroservisi u našoj aplikaciji

Aplikacija koju smo radili sastoji se od četiri mikroservisa. Svaki od njih zadužen je za jedan dio aplikacije i radi sasvim neovisno o drugim mikroservisima. Zaduženje svakog mikroservisa je u obliku modela preko API-a vratiti tražene podatke.

1. Mikroservis za registraciju korisnika

Preko ovog mikroservisa se izvršava registracija i prijava korisnika u aplikaciju, kreiranje novih korisnika i ispis svih korisnika u bazi podataka. Tako, na primjer, upisom “registrations/all”, mikroservis vrati sve registrirane korisnike s njihovim korisničkim imenom, lozinkom i email adresom u JSON formatu.

Model registracije korisnika:

|  |
| --- |
| @Entity  public class Registration {    @Id  @GeneratedValue  private long id;  private String email;  private String username;  private String password;  private long personId; |

1. Mikroservis za pse

Mikroservis za pse brine se oko registracije pasa, dohvaćanja podataka o njima i omogućavanje daljnjeg korištenja istih. Mikroservis uz pomoć kontrolera u JSON obliku daje podatke o psima ovisno o njihovom identifikacijskom broju u bazi upisom “dogs/{id}”, ili listu pasa ovisno o tome tko im je vlasnik sa “dogs/owner/{ownerId}”, a ima i opciju dohvaćanja svih registriranih pasa sa “dogs/all”.

Model psa:

|  |
| --- |
| @Entity  public class Dog {  @Id  @GeneratedValue  private Long id;  private String name;  private Long ownerId;  private String breed;  private Double age;  private String gender;  private String health;  private String hairColour;  private String eyeColour;  private Double weight;  private Double height;  private String hairLength; |

1. Mikroservis za korisnike

Svaki korisnik nakon prijave u aplikaciju može ispuniti obrazac o sebi i tako napraviti svoj profil kao šetač ili vlasnik psa. S “users/{id}” dohvaćamo korisnika i njegove podatke u JSON formatu, s “users/username/{username}” dohvaćamo podatke o korisniku na temelju njegovog korisničkog imena, a pomoću “users/walkers” dohvaćaju se svi podaci o šetačima.

Model korisnika:

|  |
| --- |
| @Entity  public class User {  @Id  @GeneratedValue(strategy=GenerationType.AUTO)  private Long id;  private String username;  private String firstName;  private String lastName;  @Column(unique = true, nullable = false)  private String email;  private boolean active = true;  private boolean walker = true; |

1. Mikroservis za šetnje pasa

Brine se za dohvaćanje podataka o prihvaćenim šetnjama (“walks/accept”), ponuđenim šetnjama (“walks/offer”), šetnjama po šetačima (“walks/{walkerId}”) odnosno po vlasnicima šetanih pasa (“walks/{ownerId}”) te po id šetnje (“walks/{id}”).

Model šetnje:

|  |
| --- |
| @Entity  public class Walk {  @Id  @GeneratedValue  private Long id;  private Long dogId;  private Long walkerId;  private Long ownerId;  private Integer duration;  private Integer distance;  private Integer price; |

5. Mikroservis za evaluaciju pasa

Brine se za dohvaćanje podataka o evaluaciji pasa (“evaluations/dog/{dogId}”), i stvaranju novih evaluacija(“evaluations/save”).

Model evaluacije tj.ocjenjivanja i komentiranja pasa:

|  |
| --- |
| @Entity  public class Evaluation {  @Id  @GeneratedValue  private Long evalId;  private Long walkerId;  private Long ownerId;  private Long dogId;  private String walkerComment;  private int walkerRating;  private String dogComment;  private int dogRating; |

# 2. Spring Tool Suite i Spring Boot

Spring Tool Suite

Spring Tool Suite je razvojno okruženje koje se bazira na Eclipseu, razvojno okruženje najčešće korišteno za programiranje u Javi. Spring se orijentira na programiranje takozvanih Spring aplikacija o kojima ćemo više u idućem poglavlju. Ovaj alat ima široku pripremu jer je između ostalog *open source*, što znaći da je dostupan i besplatan za širu javnost i njegove dokumentacije i samo okrištenje su dopuštene za korištenje svima.

Korisnicima je omogućeno programiranje, debugiranje, razvoj i pokretanje aplikacija što omogućuju Pivotal tc Server, Pivotal Cloud Foundry, Git, Maven i AspectJ. Pivotal tc Server je optimizirana zamjena za Apache Tomcat server koji se inače koristi. Pivotal Cloud Foundry je sladište za aplikacije koje su uvijek spremne za pokretanje u kontroliranim uvjetima „oblaka“. Vrlo je jednostavno kada sami radite aplikaciju voditi računa o tome što ste i kada napravili, no to u stvarnom životu nije slučaj. Na jednom projektu zna raditi više ljudi koji istodobno rade promjene na istom kodu. Kako bi se to sve moglo uskladiti i kako bi svi imali vidljive promjene bez problema, koristi se Git. Na Git se stavljaju kodovi s porukama o promjenama ili se pak uzimaju zadnje promjene na kojima se nastavlja raditi, a kako bi sama aplikacija mogla raditi potrebno ju je izgraditi i formirati u lijepu cjelinu koja se može pokretati. Tu ulogu ima alat za izgradnju zvan Maven. AspectJ je ekstenzija koja služi za formiranje ponašanja samog koda pritom ga ne mjenjajući.

Sa svojom konzolom, Spring korisnicima omogućuje praćenje ponašanja aplikacija u stvarnom vremenu ispisima poruka što programerima olakšava pronalaženje grešaka u kodu i njihovo ispravljanje. Ovaj alat svoju raznolikost prikazuje i u načinu pokretanja aplikacija. To je moguće lokalno, virtualno i na serverima koji se baziraju na oblaku.

Karakteristike Springa

* 1)  Spring Tool Suite razumije vaše Spring projekte. On parsira datoteke s konfiguracijskim postavkama i prikazuje detaljne informacije o definiranim metodama u Spring spremistu s anotacijom *Bean.* Prikazuje i koje *dependencie* koristimo te napisane servise, zahtjeve za kontrolerima i slično.
* 2)  Validacija se vrši za vrijeme pisanja koda pa tako korisnik i prije pokretanja same aplikacije može vidjeti greške koje su se pojavile.
* 3)  Ovaj alat je mijenjanje koda znano kao refactoring doveo na skroz novu razinu. Naime, Spring osim klasičnih *Java refactora* nudi i promjene u beansima i preimenovanje dubljih komponenti na vrlo jednostavan način.
* 4)  Dva velika pomagača pri pisanju koda definitivno su *content-assistant* i *quick-fixes*. *Content- assistant* pripomaže tako da pri kodiranju, dok pišemo neku metodu ili naredbu, izbacuje klizni izbornik s ponuđenim komponentama koje bih nam mogle poslužiti. To je vrlo korisno kod pisanja varijabli, pogotovo ako su dugačka imena, te na primjer pisanje *System.out.println()* naredbe za ispis koju puno brže možemo napisati kombinacijom upisa syso pa Ctrl i Space.

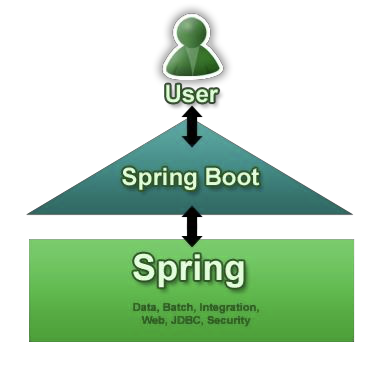
*Quick-fixes* radi na principu ponude ispravke nekih najčešće pojavljivanih grešaka. Radi na principu da kod crveno podcrtanih dijelova koda, koji izbacuju grešku, pritiskom na *Enter* i F12, detaljnije napiše uzrok problema i ponudi moguća rješenja koja se sama implementiraju u slučaju odabira.

* 5)  Kako bi dobili pregledniji uvid u komponente samog Spring projekta, te na koji način je sve povezano u cjelinu, Spring Tool Suite nudi mogućnost izrade UML dijagrama razreda koji sadrži sve klase s atributima i implementiranim metodama, zatim sučelja te načine na koje je sve to povezano.
* 6)  Spring ima integriranu takozvanu NoSQL bazu Mongo DB koja se od SQL bazi razlikuje po tome što nema relacije već se podaci spremaju u kolekcije. Način spremanja sličan je JSON obliku s dinamičnim shemama zbog čega ga još često nazivaju BSON.
* 7)  Još neke bitne stavke su podrška za alate za izgradnju programa kao što su Maven i Gradle, zatim podrška za Javu 8 te integracija Gita.

Spring Boot

Spring Boot se koristi za olakšanu izradu Spring aplikacija i servisa uz minimalne muke. Uz njegovu pomoć mogu se izraditi aplikacije s neovisnim pokretanjem komandom java -jar ili one tradicionalnije u obliku war. Također, osiguran je i alat za upis komandi i pokretanje Spring skripti.

Na slici 1. prikazana je struktura postignuta korištenjem Spring Tool Suitea i Spring Boota. Kao što je vidljivo, Spring Tool Suite je radni okvir koji pruža radnu okolinu developeru te samo pisanje koda koji onda Spring Boot konfigurira, pripremi za pokretanje i povezuje sa serverom aplikaciju te ju tako čini dostupnom korisniku koji pokretanjem vidi što aplikacija nudi i može ju nesmetano koristiti.



*Slika 1: Arhitektura Spring Projekta*

Primarni ciljevi Spring Boota:

* 1)  Šira dostupnost i veća brzina pri korištenju, a pogotovo za početnike koji se upoznaju sa Spring alatom.
* 2)  Nudi takozvano „razmišljanje izvan kutije“ gdje se korisnicima omogućuje raznolikost opcija i mogućnosti.
* 3)  Veliki raspon zadovoljenih nefunkcionalnih zahtjeva koji omogućuju širu primjenu zajedničku velikom broju projekata i klasa (zaštita, matrice, serveri...)
* 4)  Nema generiranja koda i te ručnog upisivanja XML konfiguracija ako to eksplicitno nije potrebno.

Politika kojom se Spring Boot vodi je Convention over Configuration. To je softverska dizajnerska paradigma koju u današnje vrijeme sve više koriste radni okviri kojoj je cilj povećati broj odluka koje developer može donjeti, bez nužnog gubljenja fleksibilnosti pri samom programiranju.

Izradu aplikacija koje su lagano pokretljive i neovisne uvelike olakšava Maven plugin. Pomoću Mavena se aplikacija izgrađuje i pakira u file tipa jar koji je neovisno pokretljiv.

Ono što olakšava korištenje je automatska konfiguracija. Tako, na primjer, spring-boot-starter-web dodaje sve librarije i konfigurira najčešće korištene beanse kao na primjer DispatcherServlet, ResourceHandlers i MessageSource. Osim toga, on pri pokretanju aplikacije automatski poziva spring- boot-starter-tomcat koji namješta i pokreće server za korištenje. Još jedan primjer može biti spring- boot-starter-Thymeleaf koji dodaje Thzmeleaf library dependencie i uz to konfigurira ThymeleafViewResolver bean.

Struktura Spring Boot Aplikacije za šetnju pasa:

|  |
| --- |
| @SpringBootApplication  @EnableAutoConfiguration  @EnableDiscoveryClient  public class WalksApplication {  public static void main(String[] args) {    SpringApplication.run(WalksApplication.class, args);  }} |

Anotacija @SpringBootAplication označava da se radi o Spring Boot Aplikaciji i omogućuje automatsko pokretanje svih navedeneh dependencia i konfiguraciju beansa. Pokretanjem ove aplikacije, rezultat vrlo jednostavno možete pogledati u svom web pregledniku na adresi <http://localhost:9888/>. Port za servis za pse je 8081, za registraciju korisnika 7778, za podatke o korisnicima 8888, za web server 9876 i za eureka server 8761.

Anotacija @EnableAutoConfiguration daje Spring Bootu doznanja da je omogućena automatska konfiguracija dependencia. Kroz sve konfiguracije, spoznaje se da se radi o web aplikaciji.

|  |
| --- |
| @RestController  @RequestMapping("/dogs/\*")  public class DogController {  protected DogsRepository dogsRepository;  @Autowired  public DogController(DogsRepository dogsRepository) {  super();  this.dogsRepository = dogsRepository;  }  @RequestMapping("/{id}")  public Dog byId(@PathVariable("id") Long Id) {  return dogsRepository.findById(Id);  }  } |

@RestController anotacija daje doznanja da se koriste HTTP metode s posebnim načinom zadavanja URL-a koje se vrši pisanjem anotacije @RequestMapping i željenog nastavka putanje za određenu metodu. U gornjem primjeru će se nakon pokretanja aplikacije, unosom u web preglednik http://localhost:8081/dogs/1 ispisati će se model psa s id-em 1 u JSON formatu. Još neke anotacije koje se pritom znaju koristiti za dohvaćanje su @PathVariable i @RequestBody za dohvaćanje podataka.

Zašto baš Spring?

Kritičari Spring Tool Suitea i Spring Boota izjavili su, „If Spring is the cake, Spring Boot is the icing.“ Spring Tool Suite i Spring Boot zajedno čine odličan spoj koji svakom developeru pruža mogućnost izrade kvalitetnih aplikacija bez prevelike muke oko konfiguracija. Uvelike olakšavaju izradu mikroservisa koji su zapravo dijelovi aplikacije razdvojeni u zasebne cjeline koje zajedno funkcioniraju i pružaju traženu uslugu. Ovi alati korisniku nude puno više fleksibilnosti u radu, a da pritom ne mora ručno mijenjati konfiguracijske datoteke koje se automatski osvježavaju svakom promjenom u kodu. Spring Boot sve potrebne informacije dobiva preko anotacija koje ga upućuju o kakvoj aplikaciji se radi te koje dependencije i kakvu konfiguraciju zahtjeva.

Ovakvim načinom razvoja aplikacija napravljen je odmak od standardnih Java aplikacija te je pruženo više slobode developeru u izradi promjena, prenamjenjivanju i dodavanju dijelova koda. Također, zahvaljujući Mavenu i Gradleu, omogućeno je nezavisno pokretanje jar datoteke samo jednom naredbom java -jar.

Spring se pobrinuo da svojim korisnicima pruži sve u jednom, od razvojne okoline, sučelja, konzola za upis naredbi i praćenje ponašanja programa, strukturiran prikaz sadržaja projeta, povezanost sa serverom, jednostavno pokretanje aplikacija te automatske konfiguracije. Baš iz tog razloga broj korisnika sve više raste te ovaj alat postaje vrlo popularan među Java developerima.

# 3. Spring Cloud

Uvod u Spring Cloud

Porast popularnosti cloud-native arhitektura i pomak prema implementaciji aplikacija kao niz usmjerenih mikrousluga razvijenih oko vezanih konteksta je dovela do otkrića novih i ponovnog korištenja postojećih uzoraka korisnih u izradi distribuiranih sustava. Tehnike poput otkrivanja usluga, centraliziranog pristupa konfiguraciji putem okruženja implementacije i jednostavna promjena ponašanja aplikacije putem prekidača uobičajena su rješenja za ovaj stil arhitekture i može biti ugrađen u alate za primjenu ove tehnike u raznim aplikacijama.

Spring cloud je biblioteka otvorenog koda temeljena na Spring Boot-u koja programerima olakšava razvoj Java Virtual Machine aplikacija u oblaku. On donosi developerima alate za brzi razvoj nekih od često korištenih uzoraka u raspodijeljenim sustavima (npr. upravljanje konfiguracijom, otkrivanje usluga, osigurača, pametnog usmjeravanja, mikro-proxy, kontrolna sabirnica, jednokratni tokeni, globalni satovi, odabir vođe, raspodijeljene sesije, stanje klastera) te pruža mnoštvo opcija za spajanje ovisnosti usluga u mikrouslužnim sustavima. Pomoću Spring Cloud-a, aplikacije se mogu lakše spajati sa uslugama i otkrivati informacije o okolini u Cloud Foundry i Heroku oblacima. Također, to je moguće proširiti i na druge platforme oblaka i nove usluge.

Jedna od mnogih prednosti pokretanja aplikacije u oblaku je laka dostupnost širokog raspona usluga. Umjesto konfiguracije hardware-a, instalacije, održavanja, kreiranja backupa i ostalih nužnih operacija, potrebno je samo kreirati i vezati servise klikom miša ili putem naredbe u konzoli. Na primjer, ako imamo relacijsku bazu podataka povezanu s našom aplikacijom, potrebno je kreirati DataSource objekt temeljen na toj usluzi. To je nešto oko čega nam Spring Cloud izlazi u susret, uklanjajući sav posao potreban za pristup i konfiguraciju konektora za uslugu i omogućava nam fokus na korištenje tih servise. Uz to, otkriva i i informacije o instanci aplikacije (adresa host-a, port, naziv, itd…).

Spring Cloud čini sve ovo na način neovisan o samom oblaku putem koncepta zvanog Cloud Connector. Budući da on pruža implementaciju za Cloud Foundry i Heroku, moguće je proširiti istu na druge oblake implementirajući sučelje i koristeći prednosti ostatka biblioteke. Zatim je potrebno samo dodati biblioteku sa ekstenzijama do classpath-a (put do klasa) aplikacije.

Značajke Spring Cloud-a

Spring Cloud fokusira se na pružanje dobrog iskustva već za samo «out of the box» rješenje za tipične primjere upotrebe i mehanizam nadograđivanja za pokriće drugih upotreba.

* Distribuirana / verzionirana konfiguracija
* Registracija i otkrivanje servisa
* Usmjeravanje
* Usluga – usluga pozivi
* Balansiranje opterećenja
* Prekidači
* Globalni brave
* Izbor vođe i klaster stanja
* Distribuirane poruke

Ovo je primjer jednostavne aplikacije temeljene na Spring Boot tehnologiji:

|  |
| --- |
| @Controller  public class HomeController {      @Autowired(required = false) DataSource dataSource;      @Autowired(required = false) RedisConnectionFactory redisConnectionFactory;      @Autowired(required = false) MongoDbFactory mongoDbFactory;      @Autowired(required = false) ConnectionFactory rabbitConnectionFactory;      @Autowired ApplicationInstanceInfo instanceInfo;      @RequestMapping("/")      public String home(Model model) {          Map<Class<?>, String> services = new LinkedHashMap<Class<?>, String>();          services.put(dataSource.getClass(), toString(dataSource));          services.put(mongoDbFactory.getClass(), toString(mongoDbFactory));          services.put(redisConnectionFactory.getClass(), toString(redisConnectionFactory));          services.put(rabbitConnectionFactory.getClass(), toString(rabbitConnectionFactory));          model.addAttribute("services", services.entrySet());            model.addAttribute("instanceInfo", instanceInfo);            return "home";      }        // ... various toString() methods to create a string representation for each service    } |

Sastoji se od kontrolera sa beans-ima koji predstavljaju vezane usluge te početne stranice koja ispisuje informacije o uslugama vezanim za aplikaciju.

HomeController ima 4 injektirane ovisnosti koje predstavljaju usluge koje mogu biti vezane za aplikaciju i još jednu za ApplicationInstanceInfo. Ruta «/» dodaje string reprezentaciju svake usluge skupa sa svojom klasom.

Za konfiguraciju, potrebno je dodati:

|  |
| --- |
| @Configuration  @ServiceScan  @Profile("cloud")  public class CloudConfig extends AbstractCloudConfig {      @Bean      public ApplicationInstanceInfo applicationInfo() {          return cloud().getApplicationInstanceInfo();      }  } |

Klasa proširuje AbstractCloudConfig, što je način korištenja Java config pristupa sa Spring Cloud-om. Postavljamo @Profile(«cloud») kako bismo osigurali da se ta konfiguracija učita samo u cloud okolini. @ServiceScan anotacija skenira sve vezane usluge i kreira bean za svaku.

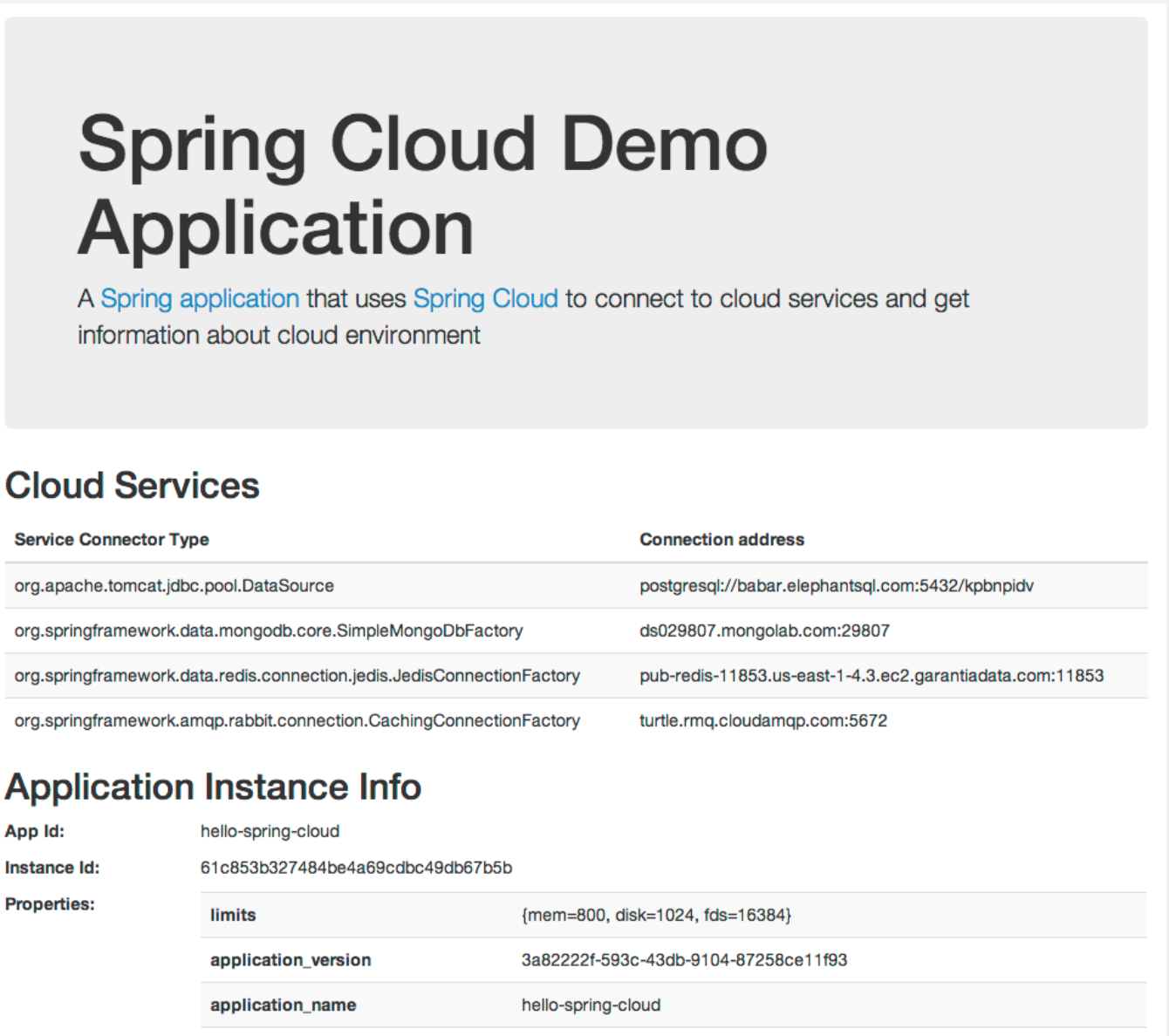
Uključujemo sljedeći manifest.yml koji povezuje sve 4 potrebne usluge.

|  |
| --- |
| applications:  - name: hello-spring-cloud    memory: 512M    instances: 1    host: hello-spring-cloud-${random-word}    domain: cfapps.io    path: target/hello-spring-cloud-0.0.1-SNAPSHOT.jar    services:  - postgres-service  - amqp-service  - mongodb-service  - redis-service    env:      SPRING\_PROFILES\_DEFAULT: cloud |

Sve što je sada potrebno je build i push:

sh $ mvn package $ cf push

Kada posjetimo stranicu, dobivamo sljedeće:



Korištenje @ServiceScan olakšava dohvat i početak korištenja svih usluga, ali u praksi je često potrebno više kontrola oko kreiranja konektora servisa poput postavljanja njihovih parametara. U tom slučaju, može se koristiti Spring Cloud-ov Java Config ili XML config podrška.

|  |
| --- |
| @Configuration  @Profile("cloud")  public class CloudConfig extends AbstractCloudConfig {      @Bean      public ConnectionFactory rabbitConnectionFactory() {      return connectionFactory().rabbitConnectionFactory();      }        @Bean      public DataSource dataSource() {      return connectionFactory().dataSource();      }        @Bean      public RedisConnectionFactory redisConnectionFactory() {      return connectionFactory().redisConnectionFactory();      }      @Bean      public ApplicationInstanceInfo applicationInfo() {          return cloud().getApplicationInstanceInfo();      }  } |

U odnosu na prvu verziju, uklonili smo @ServiceScan anotaciju. Umjesto toga, koristi se API izložen od strane AbstractCloudConfig kako bi se kreirao bean za svaku uslugu. Za sada, bean-ovi stvoren na ovaj način su identični onima stvorim od @ServiceScan, ali sada imamo mogućnost dodatne konfiguracije. Na primjer, ako smo htjeli vezati DataSource bean za određene usluge i inicijalizirati ga sa određenom konfiguracijom, možemo napraviti sljedeće izmjene:

|  |
| --- |
| @Bean  public DataSource dataSource() {      PoolConfig poolConfig = new PoolConfig(20, 200);      ConnectionConfig connectionConfig =          new ConnectionConfig("sessionVariables=sql\_mode='ANSI';characterEncoding=UTF-8");      DataSourceConfig serviceConfig =          new DataSourceConfig(poolConfig, connectionConfig);       return connectionFactory().dataSource("my-service", serviceConfig);  } |

Spring Cloud koristi vrlo deklarativan pristup, zahvaljujući kojem će se uz promjenu samo classpath-a ili anotacije moći koristiti velik broj značajki Spring Cloud tehnologije. Primjer aplikacije koja je discovery klijent:

|  |
| --- |
| @SpringBootApplication  @EnableDiscoveryClient  public class Application {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  } |

Spring Cloud projekti

**Spring Cloud Config**

Centralizirano vanjsko upravljanje konfiguracijom podržano od strane Git repozitorija. Konfiguracija resursa pokazuje direktno na Spring «Environment», ali može se koristiti od strane ne-Spring aplikacija po želji.

**Spring Cloud Netflix**

Integracija sa raznim Netflix OSS komponentama (Eureka, Hystrix, Zuul, Archaius, itd...).

**Spring Cloud Bus**

Event sabirnica za povezivanje usluga i instanci usluga zajedno s distribuiranim porukama. Korisno za propagiranje promjene stanja u cijelom klaster (npr. konfiguriranje promjena događaja).

**Spring Cloud for Cloud Foundry**

Integrira Vašu aplikaciju sa Pivotal Cloudfoundry. Omogućava otkrivanje usluga te olakšava implementaciju SSO i OAuth2 zaštićenih resursa, i Cloudfoundry usluga brokera.

**Spring Cloud Cloud Foundry Service Broker**

Pruža početnu točku za gradnju servisa brokera koja upravlja Cloud Foundry managed servisom.

**Spring Cloud Cluster**

Odabir vođe i česti stateful obrasci sa apstrakcijom i implementacijom za Zookeeper, Redis, Hazelcast, Consul.

**Spring Cloud Consul**

Otkrivanje usluga i upravljanje konfiguracijom sa Hashicorp Consul.

**Spring Cloud Security**

Pruža podršku za load-balanced (omptimizacija tereta) OAuth2 rest klijent i relej za autentikacijsko zaglavlje u Zuul posredniku.

**Spring Cloud Sleuth**

Distribuirano traženje za Spring Cloud aplikacije, kompatibilna sa Zipkin, HTrace i log-bazirano (npr. ELK) traženje.

**Spring Cloud Data Flow**

Cloud native orkestracijski servis za aplikacije sastavljene od mikroservisa. Jednostavan DSL, drag-and-drop grafičko sučelje i REST-API zajedno pojednostavnjuju cjelokupnu orkestraciju puta podataka temeljenog na mikroservisima.

**Spring Cloud Stream**

Lagani event-driven mikroservis framework za brzo izgradnju aplikacija koje se mogu povezati s vanjskim sustavima. Jednostavan deklarativan model za slanje i primanje poruka putem Apache Kafke ili RabbitMQ između Spring Boot aplikacija.

**Spring Cloud Stream App Starters**

Spring Integracijske aplikacije temeljene na Spring Boot-u koje omogućuju integraciju s vanjskim sustavima.

**Spring Cloud Task**

Kratkotrajan microservis framework za brzu izgradnju aplikacija koje izvode konačnu količinu obrade podataka. Jednostavno deklarativno dodavanje i funkcionalnih i nefunkcionalih značajki za Spring Boot aplikacije.

**Spring Cloud Task App Starters**

Spring Boot aplikacije koje mogu biti bilo koji proces, uključujući Spring Batch poslove koji se ne vrte zauvijek, već završavaju ili stopiraju nakon ograničenog razdoblje obrade podataka.

**Spring Cloud Zookeeper**

Otkrivanje usluga i upravljanje konfiguracijom Apache Zookeper.

**Spring Cloud for Amazon Web Services**

Jednostavna integracija sa hostingom Amazon Web Services. Ona nudi prikladan način za interakciju sa AWS pruženim uslugama pomoću dobro poznatih Spring idioma i API-ja, poput komunikacije porukama ili caching API šučelja. Programeri mogu graditi svoju aplikaciju oko hostanih usluga bez obzira na infrastrukturu ili održavanje.

**Spring Cloud Connectors**

Olakšava PaaS (Platforma kao usluga) aplikacijama spajanje sa backend servisima poput baza podataka ili brokerima poruka (projekt je bio poznat pod imenom “Spring Cloud”)

**Spring Cloud Starters**

Projekt koji olakšava upravljanje ovisnostima korisnika Spring Cloud-a.

**Spring Cloud CLI**

Spring Boot CLI plugin za brzo kreiranje Spring Cloud komponenti aplikacija u Groovy-ju.

Izdanja Spring Cloud-a

Spring Cloud je kišobran projekt koji se sastoji od neovisnih projekata koji imaju raličit tempo izdavanja. Kako bi se uskladilo upravljanje portfeljem izdan je troškovnik (Bill of Materials) je objavljen sa skupom ovisnosti na pojedinom projektu. Izlasci imaju imena, a ne verzije, kako bi se izbjegla zamjena s potprojektima.

Verzije poredane abecednim slijedom nazvane po imenima stanica londonske podzemne željeznice:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Angel.SR6 | Brixton.SR7 | Camden.SR3 | Camden.BUILD-SNAPSHOT |
| spring-cloud-aws | 1.0.4.RELEASE | 1.1.3.RELEASE | 1.1.3.RELEASE | 1.1.4.BUILD-SNAPSHOT |
| spring-cloud-bus | 1.0.3.RELEASE | 1.1.2.RELEASE | 1.2.1.RELEASE | 1.2.2.BUILD-SNAPSHOT |
| spring-cloud-cli | 1.0.6.RELEASE | 1.1.6.RELEASE | 1.2.0.RC1 | 1.2.0.BUILD-SNAPSHOT |
| spring-cloud-commons | 1.0.5.RELEASE | 1.1.3.RELEASE | 1.1.6.RELEASE | 1.1.7.BUILD-SNAPSHOT |
| spring-cloud-contract |  |  | 1.0.2.RELEASE | 1.2.3.BUILD-SNAPSHOT |
| spring-cloud-config | 1.0.4.RELEASE | 1.1.3.RELEASE | 1.2.1.RELEASE | 1.2.4.BUILD-SNAPSHOT |
| spring-cloud-netflix | 1.0.7.RELEASE | 1.1.7.RELEASE | 1.2.3.RELEASE | 1.1.4.BUILD-SNAPSHOT |
| spring-cloud-security | 1.0.3.RELEASE | 1.1.3.RELEASE | 1.1.3.RELEASE | 1.1.3.BUILD-SNAPSHOT |
| spring-cloud-starters | 1.0.6.RELEASE |  |  |  |
| spring-cloud-cloudfoundry |  | 1.0.1.RELEASE | 1.0.1.RELEASE | 1.0.2.BUILD-SNAPSHOT |
| spring-cloud-cluster |  | 1.0.1.RELEASE |  |  |
| spring-cloud-consul |  | 1.0.2.RELEASE | 1.1.2.RELEASE | 1.1.3.BUILD-SNAPSHOT |
| spring-cloud-sleuth |  | 1.0.11.RELEASE | 1.1.0.RELEASE | 1.1.1.BUILD-SNAPSHOT |
| spring-cloud-stream |  | 1.0.2.RELEASE | Brooklyn.RC1 | Brooklyn.BUILD-SNAPSHOT |
| spring-cloud-zookeeper |  | 1.0.3.RELEASE | 1.0.3.RELEASE | 1.0.4.BUILD-SNAPSHOT |
| spring-boot | 1.2.8.RELEASE | 1.3.8.RELEASE | 1.4.2.RELEASE | 1.4.2.BUILD-SNAPSHOT |
| spring-cloud-task |  | 1.0.3.RELEASE | 1.0.3.RELEASE | 1.0.4.BUILD-SNAPSHOT |

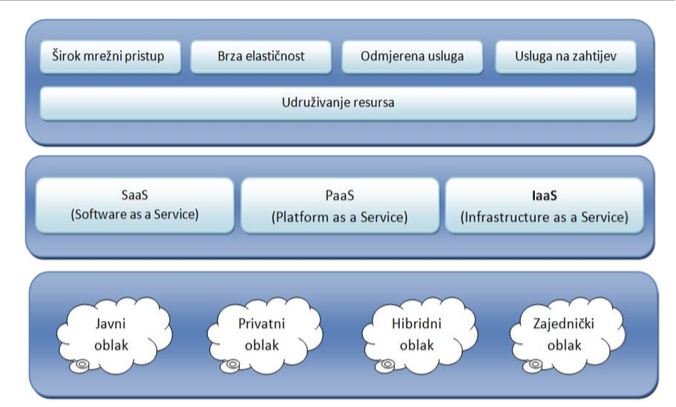
Zašto Spring Cloud?

Spring Cloud nativno podržava sve alate potrebne za izradu aplikacije u oblaku temeljene na mikrouslugama, a njegovo korištenje olakšava izradu robusnih cloud aplikacija građenih od mikrousluga. Također, važno je naglasiti da je istu aplikaciju moguće implementirati na više računalnih oblaka uz relativno malo dodatnog truda, što ovu tehnologiju čini vrlo atraktivnom za sve koji se bave izradom aplikacija u oblaku s obzirom da je put od nacrta do implementacije u oblaku, čak i za nekoga bez iskustva u ovoj tehnologiji, vrlo kratak. Uz sve to, postoji i velik broj projekata koji Spring Cloud čine još zanimljivijim i razvijaju njegove funkcionalnosti i proširuju samu platformu, poput Spring Cloud Netflix projekta, koji donosi integraciju sa raznim Netflix komponentama od kojih je najpoznatija Eureka (pronalaženje usluga).

# 4. Analiza besplatnih cloudova koji se mogu koristiti za isporuku mikrousluga

**Cloud (Oblak)**

Prema NIST-u (National Institute of Standards and Technology) Cloud computing (računarstvo u oblaku) je model za omogućavanje sveprisutnog, odgovarajućeg, mrežnog pristupa na zahtjev (on-demand) za dijeljenje konfigurabilnih računalnih resursa (npr., mreže, servera, spremišta podataka, aplikacija i servisa/usluga) koji se mogu brzo omogućiti i dodijeliti uz minimalan napor i interakciju sa davateljem usluge. Ovakav model oblaka se sastoji od pet osnovnih karakteristika, tri modela usluga, te četiri implementacijska modela.



*Slika 1. Pet osnovnih karakteristika, tri modela usluga, te četiri implementacijska modela*

Što znači biti u “oblaku”?

Cloud computing je revolucionarni koncept koji nudi novi način pristupa osobnim podacima i aplikacijama, koji više nisu smješteni na računalu već u “oblaku” – što znači da programu, evidencijama i dokumentaciji možete pristupiti s većeg broja uređaja, u bilo koje vrijeme i s različitih lokacija. Sve što je potrebno je internet veza. Kao rezultat toga korisnici usluga u “oblaku” mogu bolje, brže i jednostavnije koristiti i mijenjati podatke.

***3 modela usluge:***

**1. Infrastructure as a Service (IaaS)**, daje samo baznu infrastrukturu ostavljajući krajnjeg korisnika odgovornog za platformu i konfiguraciju potrebnu za isporuku aplikacija. Primjerice Amazonov AWS i Microsoftov Azure.

**2. Software as a Service (SaaS**) primjerice Gmail ili Salesforce.com.

**3. Platform as a Service (PaaS)**, which helps to reduce the development overhead (environment configuration) by providing a ready-to-use platform. PaaS services can be hosted on top of infrastructure provided by an IaaS.

***Koje su prednosti korištenja usluga u "oblaku"?***

* Dostupnost aplikacija
* Skalabilnost aplikacija (mogućnost opsluživanja velikog broja korisnika)
* Fleksibilnost u mijenjanju i prilagodbi aplikacija
* Stalno praćenje rada i održavanje infrastructure

***Nedostaci koje treba uzeti u obzir kada se počne razmišljati o razvoju aplikacija koje će se smjestiti u "oblak"?***

* Mora se usvojiti nov način razvoja aplikacija:

a. Platforme koje danas nude različiti proizvođači ili ponuditelji poput Microsofta i Amazona vrlo su različite i zahtijevaju poseban način projektiranja arhitekture takvih aplikacija.

b. Nije moguće (jednostavno) premještati postojeće aplikacije.

* Ne postoje standardi za:

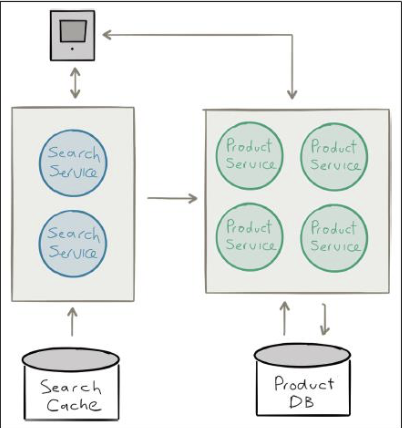
a. povezivanje aplikacija (servisa) u "oblacima" različitih proizvođača (ili čak istog proizvođača),

b. razmjenu podatka,

c. premještanje podataka i programske podrške iz "oblaka" jednog proizvođača u drugi.

Mikroservisi (Mikrousluge)

Mikroservisi su novi (ili redizajnirani) način razvoja raspodijeljenih sustava. Sustav se sastoji od skupine malih dobro-definiranih servisa (izgrađenih kao komponente) koji su neovisni i upravljaju vlastitim podacima. Servisi međusobno komuniciraju asinkrono. Raspodijeljeni sustav nastaje upravo interakcijom servisa.

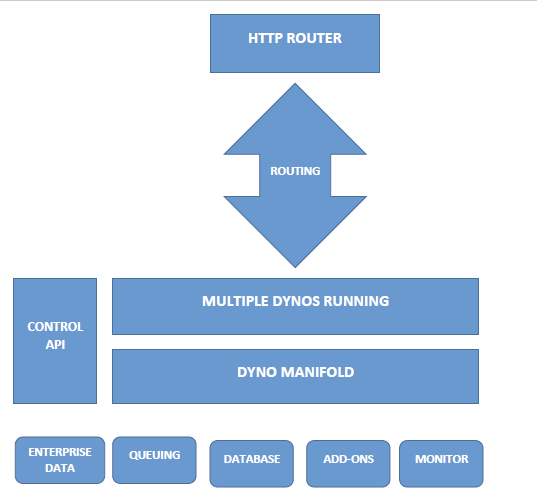


*Slika 1. Mikroservisi*

Besplatni cloudovi

**Heroku**

Heroku je Cloud platform-as-a-Service (PaaS) koji podržava nekoliko programskih jezika koji se koriste kao model implementacije web aplikacija. Kao jedna od prvih cloud platforma u razvoju je još od lipnja 2007. godine, kada je bio podržan samo na Ruby programski jezik. Heroku trenutno podržava i ostale programske jezike, Java, Node.js, Scala, Clojure, Python, PHP, i Go. Heroku je poliglot platforma jer omogućava developerima gradnju, pokretanje aplikacija na sličan način u svim jezicima.



*Slika 2. Arhitektura HEROKU-a*

Aplikacije koje se pokreću na Heroku poslužitelju koristite Heroku DNS Server za preusmjeravanje na aplikacijsku domenu ("applicationname.herokuapp.com"). Svaki od kontejnera aplikacije(ili dynos) se širi preko "Dyno mreže", koji se sastoji od nekoliko poslužitelja. Herokov Git poslužitelj rukuje repozitorijima tako da omogućava push samo ovlaštenim korisnicima.

Definiranje aplikacije: Definicija aplikacije tj. izvorni kod i opis temelje se na Heroku-ovom frameworku. Mehanizmi ovisnosti razlikuju po jezicima, jer za Ruby programer koristi Gemfile, za Python requirements.txt, u Node.js package.json, u Java pom.xml, i tako dalje.

Znanje o izvršavanju: Programeri ne trebaju napraviti mnoge promjene u aplikaciji kako bi se one izvodile na Heroku-u. Jedan zahtjev je informiranje platforme o tome koji dijelovi aplikacije su tipa runnable. To je učinjeno u Procfile, tekstualnoj datoteci koja prati izvorni kod. Svaki redak Procfile deklarira tip procesa.

Objavljivanje aplikacije: Razvoj aplikacija na Heroku-u primarno se obavlja kroz GIT. Aplikacija dobiva novi GIT upravljač koji se obično zove Heroku zajedno sa svojim lokalnim Git spremištem gdje je napravljena aplikacija. Stoga je deployment na HEROKU sličan korištenju gitove push naredbe. Aplikacija može biti i objavljena putem GitHub, Dropbox ili API- ja.

Implementacija aplikacije: Mehanizam za implementaciju obično se razlikuje za različite jezike, ali slijedi konzistentan obrazac preuzimanja određene ovisnosti i stvaranje potrebnih atributa.

***Razina besplatnosti***

Heroku nudi mogućnost besplatnog korištenja njihovih usluga pod određenim uvjetima. Radi se o ekperimentalnom načinu rada koji omogućava jezgrene platformske mogućnosti, u isto vrijeme samo 1 web/worker može pristupati, dopušta prilagođene domene, 550 sati besplatnih dyno sati. Aplikacije se mogu vrtiti sve dok god je ostalo dyno sati. Za demonstrativne svrhe ovog projekta ti dostupni resursi bi bili dovoljni.

***Opći dojam korištenja Heroku za isporuku mikrousluga***

Heroku nudi detaljnu dokumentaciju za isporuku Spring Boot aplikacija. Dokumentacija opisuje sve od stvaranja Spring Boot aplikacije, pripreme aplikacije za Heroku preko git repozitorija ili Maven-a, spajanja na bazu podataka, same isporuke i skaliranja aplikacije. Nažalost ne navodi da se isti postupak može provesti i za mikrousluge bez ikakve izmjene. Obzirom na specifičnost arhitekture moguće je da će proizvesti neke neočekivane probleme. To nećemo znati dok ne isprobamo. Internet ne obiluje ni videozapisima i uputama o tome kako isporučiti mikrousluge.

Drugo rješenje navodi se u članku na Heroku blogu. Netflix za svoja rješenja koristi upravo mikrousluge. On je pritom i razvio niz odličnih open-source komponenti koje sačinjavaju njegovu platformu. Oni uključuju Eureku za otkrivanje usluga, Hystrix za baratnje greškama usluga i Ribbon za bolji balans učitavanja na klijentskoj strani i mnoge druge. Kao takve IBM, Yelp, Hotels.com i mnogi drugi su prihvatili te tehnologije u svoje sustave. Iako korištenje takvih komponenti zahtjeva mnogo kompliciranih konfiguracija zahvaljujući Spring Framework-u lako je isporučiti Netflix

OSS na Heroku. U članku je objašnjeno kako na jednostavan način prvo isporučiti Netflixov Eureka server na Heroku i zatim povezati svoje vlastite mikrousluge na njega. Potrebno je kreirati Eureka server, zatim Eureka klijent i replike instanca Eureka servera obzirom sa Eureka server ne podržava perzistenciju podataka.

**Cloud Foundry**

Cloud Foundry je open source cloud computing platforma izvorno razvijena u VMware-u. To je sada u vlasništvu Pivotal softvera, što je zajednički pothvat sastavljen od VMware-a, EMC, i General Electrica. Za razliku od većine Cloud Computing platform usluga koje su vezane uz specifičnog cloud davatelja usluga, Cloud Foundry je dostupan kao samostalan programski paket. Može se naravno isporučivat na Amazonov AWS, ali također se može i samostalno isporučivat na vlastiti OpenStack server ili HP-ov Helion ili VMware-ov vSphere.

Iako Cloud Foundry podržava mnoštvo jezika i frameworka, uključujući Javu, JavaScript, Go, PHP, Python, and Ruby,ali neće sve uporabe nužno dobro funkcionirati. Kao i sa svim modernim aplikacijskim primjenama projekt bi trebao pratiti Twelve-Factor App standard.

Ključne prednosti Cloud Foundry: - Prenosivost aplikacija - Primjena auto-skaliranja - Centralizirana platform administracija - Centralizirano loggiranje.

- Dinamičko usmjeravanje - Upravljanje zdravljem aplikacije - Integracija s vanjskim komponentama za loggiranje kao što su Elasticsearch i Logstash - Pristup temeljen na ulogama za razmještene aplikacije - Podrška za vertikalno i horizontalno skaliranje - Sigurnosna infrastruktura - Podrška za razne IaaS usluge

***Razina besplatnosti***

Radi se zapravo o softveru čije je korištenje besplatno. Cijena ove opcije je ona koju nam zada odabrani davatelj usluga kojeg moramo uz Cloud Foundry odabrati.

***Opći dojam korištenja Cloud Foundry za isporuku mikrousluga***

Za korištenje Cloud Foundry potrebno je napraviti račun i podesiti Cloud Foundry Command Line Interface (cf). Installer za cf postoji za Mac, Windows i Linux. Iako cf omogućava niz opcija zapravo je cf push jedina dovoljna. Obzirom da je Cloud Foundry samo softver, potrebno je zapravo obratiti se jednom od certificiranih providera usluga. Cloud Foundry je podržan kod:

AppFog from CenturyLink

Pruža potporu za Javu, omogućava izradu besplatnog računa sa $500 u upotrebi. Nakon toga plaćanje se izvršava svaki sat sa $0.04/GB

Atos Cloud Foundry

Izuzetno pogodan za mikrousluge. Ne pruža besplatno korištenje bez razgovora sa zastupnikom.

GE Predix

Moguć besplatan račun sa 10 instanci servisa ,100 ruta i 4 GB memorije. Nadogradnja na platni model je uvijek moguća.

IBM BlueMix

Bluemix je besplatan prvih 30 dana. Probni rok ne zahtijeva unos kreditne kartice. 2 GB runtime i container memorije su dostupni kroz 30 dana.

SAP HANA Cloud Platform

Omogućava također besplatno testiranje sa maksimalno 10 posjeta na dan, 1GB pohrane, 1 krajnjeg korisnika.

Pivotal Web Services

Besplatni period omogućava $87 kredita. Nije potrebno unositi kreditnu karticu. Inače cijena iznosi $0.03/GB po satu. Obzirom da radi u suradnji sa Cloud Foundry-jem, ponovo se sve rješava samo jednom naredbom $ cf push.



*Slika 3. Proces koji prolazi svaka aplikacija*

Pivotal Web Services (PWS) nudi podršku za Ruby, js, Javu i Go. Također nudi alate za operacije i motrenje postavljenih aplikacija. Dokumentacija je dostupna za nis frameworka tako i za Spring. U dokumentaciji je navedeno sve od deklariranja ovisnosti, alociranja potrebne memorije, konfiguriranja veza sa bazama, manifesta, do same isporuke aplikacije. Poveći je i broj vodiča te videozapisa dostupan koji objašnjavaju isporuku konkretno mikrousluga na PWS.

**Ostali cloudovi**

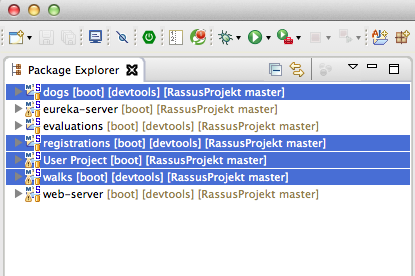
Internet obiluje nizom tvrtki koje nude usluge clouda . Većinom rade na iste principe kao i već navedeni cloudovi. Svaki od njih ponovo nudi određeno besplatno razdoblje , pa i čak do 3 aplikacije ali nakon toga usluge se naplaćuju. Primjeri takvih cloudova: Microsoft Azure, Docker, OpenShift, Boxfuse, Amazon Web Services, Vagrant, Jelastic, Jenkins, VMware vSphere.

**Zaključak**

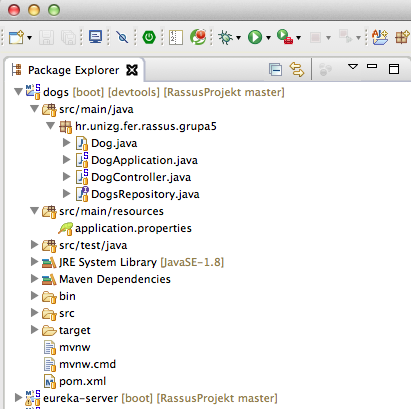
Velik je broj cloudova na koje je moguće isporučiti aplikacije, usluge i mikrousluge. Nažalost malen je broj onih koji značajnu količinu resursa daju besplatno. Većinom omogućavaju besplatni probni rok , a potom rade na principu plati koliko resursa trošiš, koliko memorije i kolika ti brzina treba. Za velike kompanije ovo je povoljan i siguran način objavljivanja i pohranjivanja aplikacija. Za potrebe našeg studenskog rada dovoljne su i probne verzije koje nude primjerice Pivotal i IBM. Uzeli smo u obzir Heroku, Microsoft Azure, IBM te Docker. Zbog detaljnije dokumentacije, većeg broja videozapisa za mikrousuge te dobre podrške za Javu tj. Spring odlučili smo se za Pivotal za kojeg se nadamo da će poslužiti svrsi demonstracije izgradnje mikrousluge. Njega ćemo koristiti u suradnji sa Cloud Foundry Command Line Interface-om.

# 5. Naš projekt

U drugoj fazi projekta započeli smo rad na našim mikroservisima. Svi članovi su kreirali vlastiti mikroservis sa određenom zadaćom.



*Slika 1. Mikroservisi*



*Slika 2. Struktura mikroservisa*

Mikroservisi kao takvi imaju strukturu običnih API-ja koji se svaki spaja na zasebnu bazu. Pomoću kontrolera za određene rute dobivamo podatke koji nas zanimaju. Naknadno je dodan I treći mikroservis evaluations za ocjenjivanje pasa. U trećoj fazi projekta mikrousluge su povezane korištenjem Eureke kako bi mogle međusobno komunicirati.

Eureka server se sastoji od EurekaServerApplication.java i application.yml datoteka. Ovo je primjer našeg Eureka servera, koji služi za registraciju i traženje mikroservisa. Sam kod je jednostavan, te je preuzet sa Spring Cloud projekta, a u application.yml smo postavili parametre Eureka servera:

|  |
| --- |
| @SpringBootApplication  @EnableEurekaServer  **public** **class** EurekaServerApplication {  **public** **static** **void** main(String[] args) {  **new** SpringApplicationBuilder(EurekaServerApplication.**class**)  .web(**true**).run(args);  }} |

Iz ovoga možemo vidjeti da je port na kojem se Eureka vrti 8761 te se naši mikroservisi na istom tom portu trebaju registrirati što također postavljamo u njihovim application.yml datotekama.

|  |
| --- |
| server:  port: ${PORT:8761}  eureka:  instance:  hostname: localhost  client:  registerWithEureka: **false**  fetchRegistry: **false**  serviceUrl:  defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/  server:  waitTimeInMsWhenSyncEmpty: 0 |

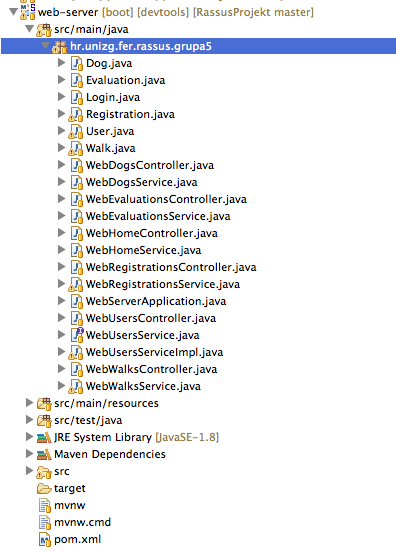
Registracija pojedinog servisa na Eureku je rješena korištenjem annotationa @EnableDiscoveryClient. Također za potrebe korištenja Eureke potrebno je pravilno podesiti ovisnosti u pom.xml file-u.

|  |
| --- |
| @SpringBootApplication  @EnableAutoConfiguration  @EnableDiscoveryClient  **public** **class** DogApplication{  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(DogApplication.**class**, args);  }  } |

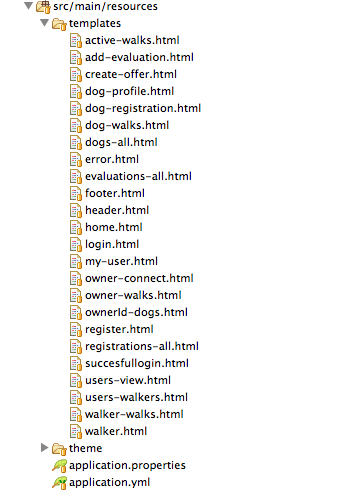
Servisi su lokalno morali bit postavljeni na različitim portovima kako bi se pravilno izvršavali. U četvrtoj fazi projekta bavili smo se ostvarivanjem neovisnog web servisa koji će komunicirati sa svim mikroservisima.

U strukturi nalazimo modele slične onima u mikroservisima, prilagođenim na potrebe frontenda tj. same web aplikacije. Servisi funkcioniraju kao poveznica između pozadinskih mikroservisa I web kontrolera. Web kontroler mapira rute frontenda, preko servisa dohvaća relevantne podatke I poslužuje View-ove.

View-ovi se nalaze u folderu templates čija je struktura prikazana na slici 4.



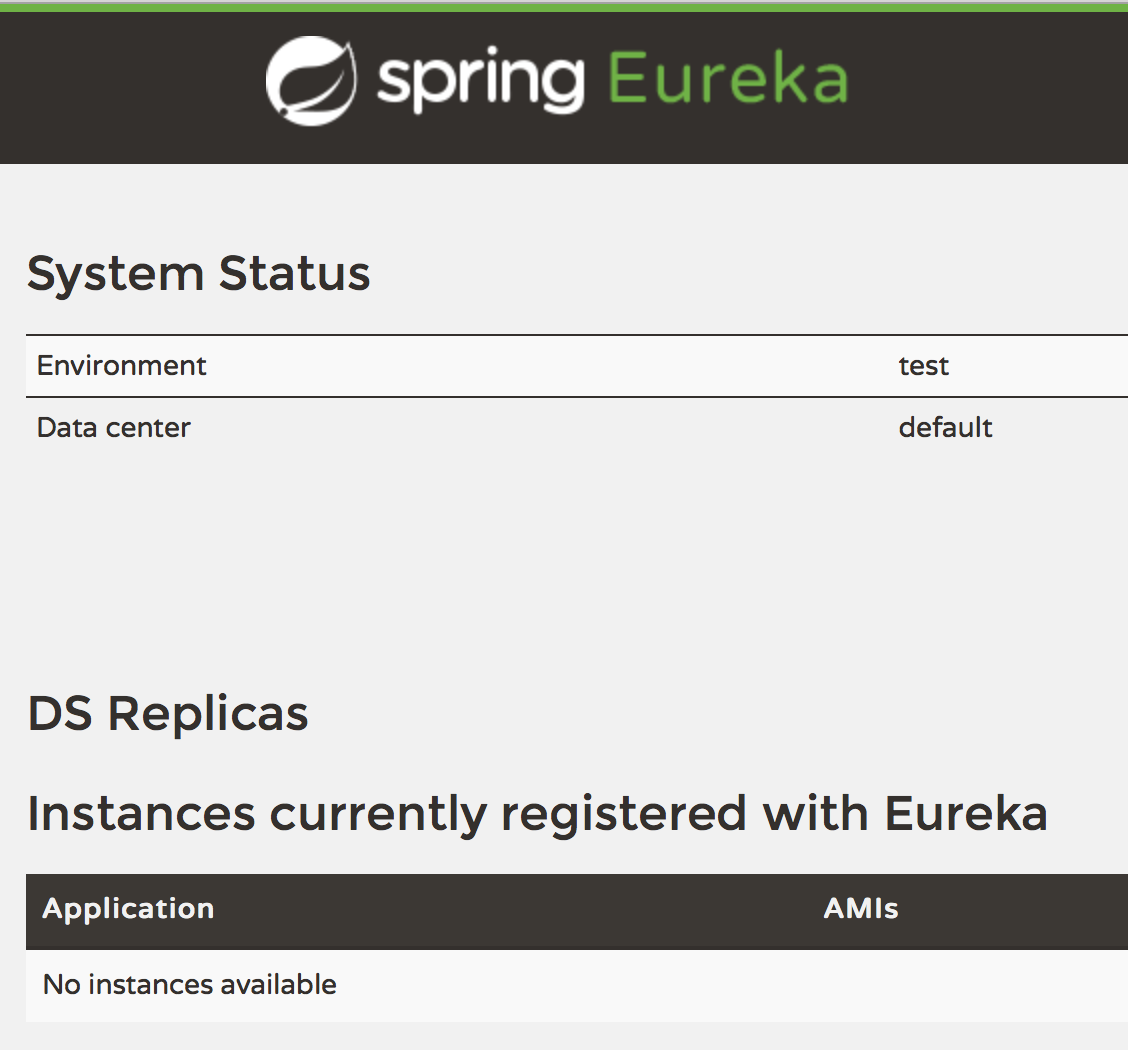
*Slika 3. Struktura web servisa*



*Slika 4. Template datoteke*

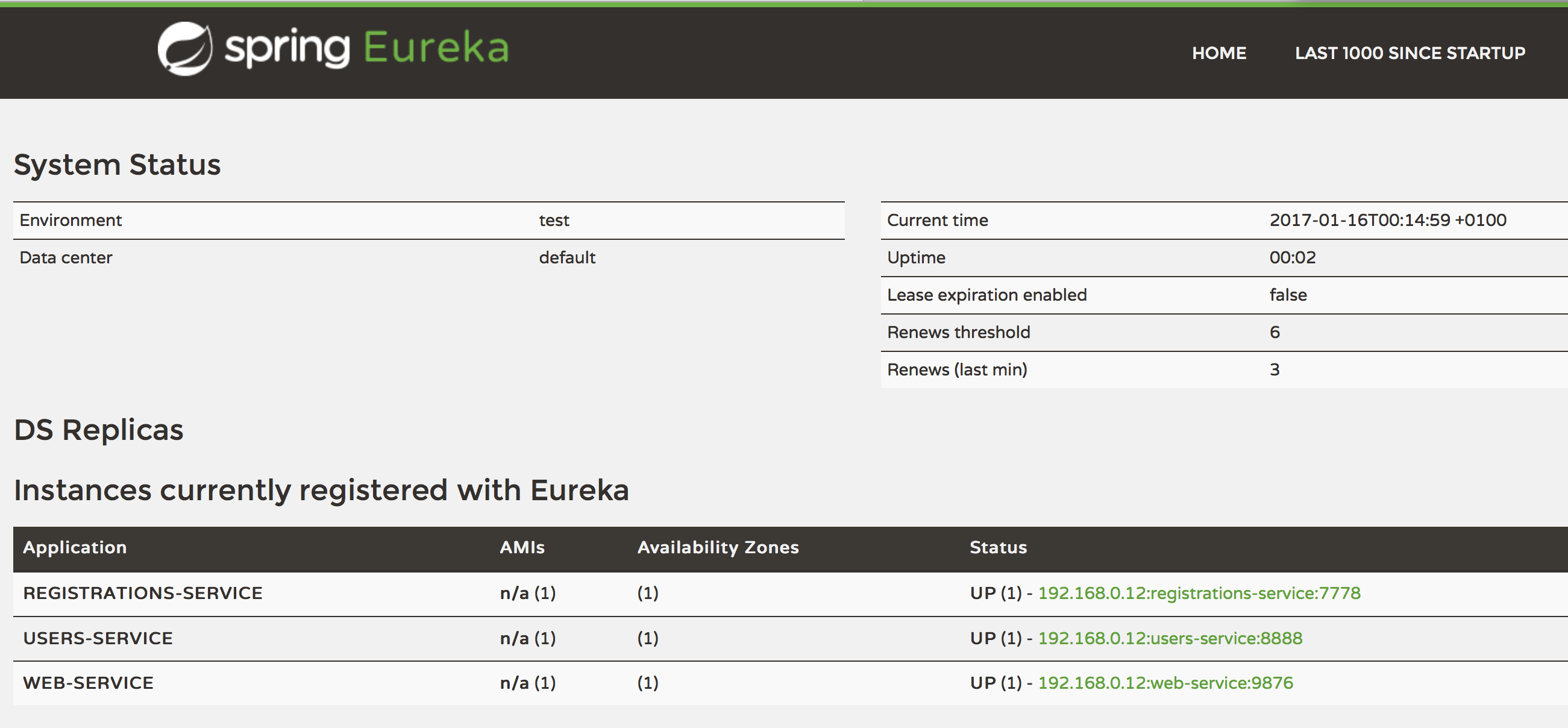
# 6. Upute za instalaciju i korištenje

Za pokretanje aplikacije potrebno je instalirati neki IDE za pokretanje Java projekata. Preporučeni IDE je Spring Tool Suite. Dovoljno je importati projekte u workspace i pokrenuti ih. Pokretanje same aplikacije izvodi se na način da se u STS-u prvo pokrene servis eureka-server putem Run as -> Spring Boot App, a zatim na isti način i ostali mikroservisi, zatim web-server. Nakon što se pokrene eureka-server, ukoliko se u browser upiše <http://localhost:8761>, dolazi se na korisničko sučelje Eureka servera.



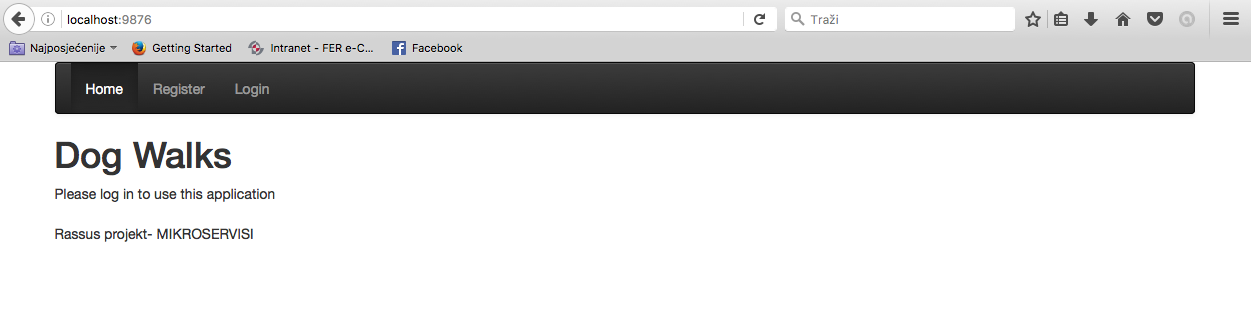
*Slika.1: Isječak korisničkog sučelja Eureka servera prije registracije mikroservisa*

Nakon što se na isti (Run as -> Spring Boot App ) način pokrenu i ostali (mikro)servisi, na korisničkom sučelju Eureka servera pojaviti će se informacije o registriranim servisima.



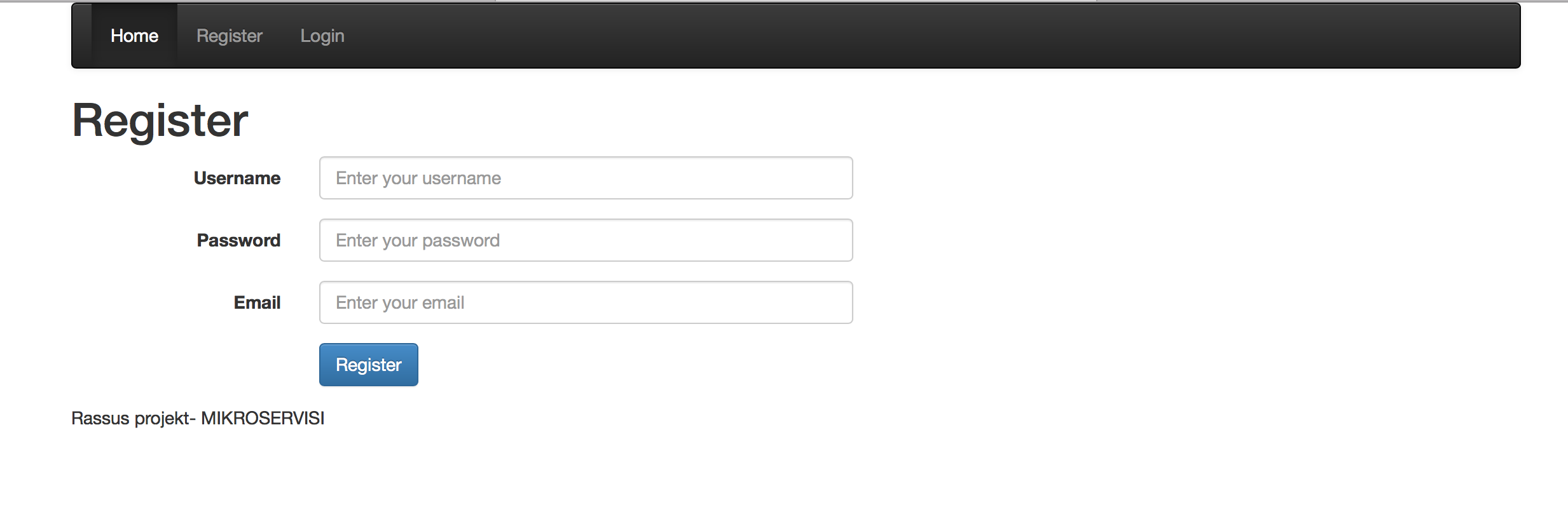
*Slika 2: Isječak korisničkog sučelja Eureka servera nakon registracije mikroservisa (u ovom slučaju web-servera te mikroservisa za korisnike i registraciju)*

Nakon što pokrenemo sve mikroservise koji se pritom registriraju na Eureka server, upisivanjem adrese <http://localhost:9876> u browser dolazimo na početni ekran korisničkog sučelja aplikacije.



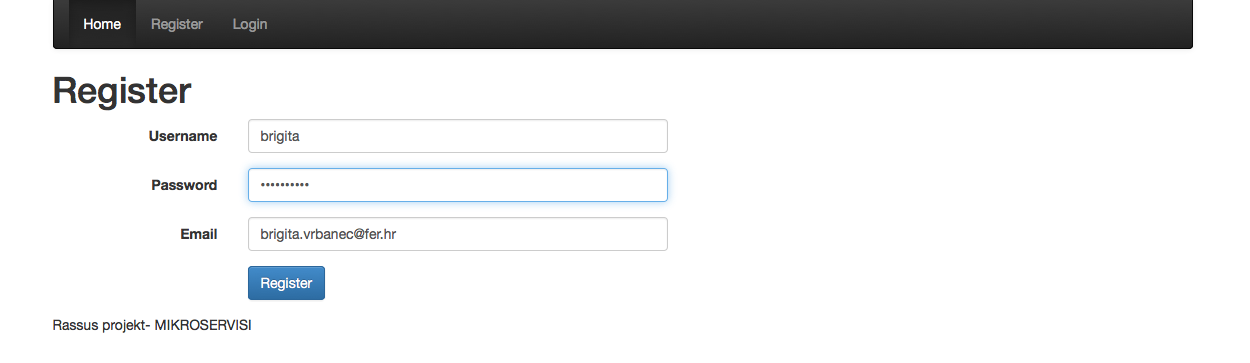
*Slika 3: Početna stranica korisničkog sučelja aplikacije*

Obzirom da kod prvog pokretanja aplikacije ne postoje registrirani korisnici koji bi se mogli ulogirati u aplikaciju, potrebno je klikom na link Register otići na stranicu za registraciju novog korisnika.



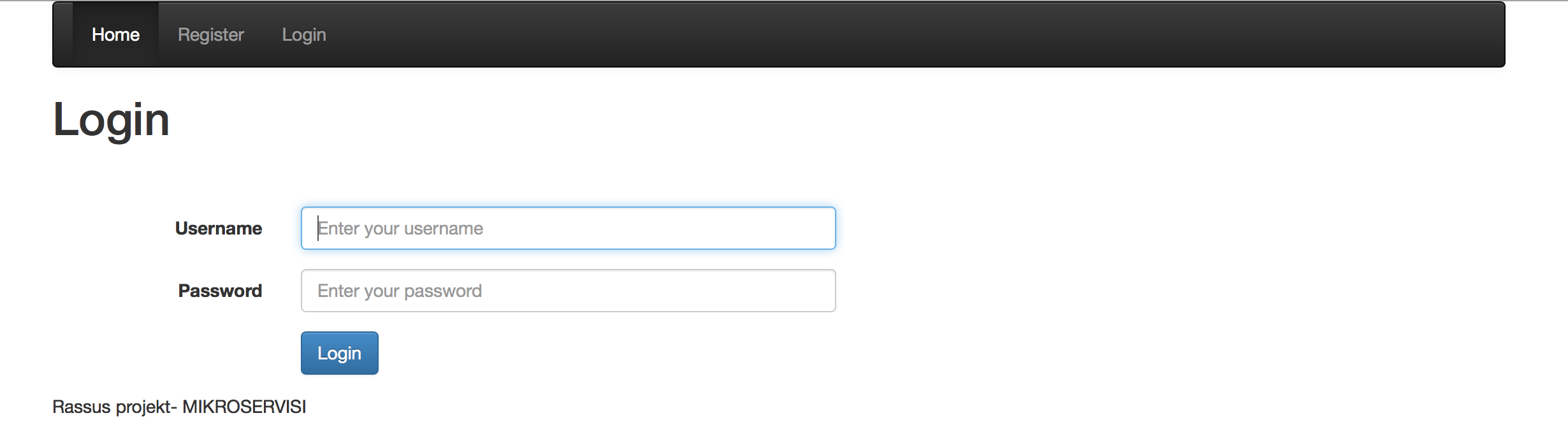
*Slika 4: Stranica za registraciju novog korisnika*

U formu za registraciju korisnika potrebno je upisati korisničko ime, lozinku te e-mail novog korisnika.



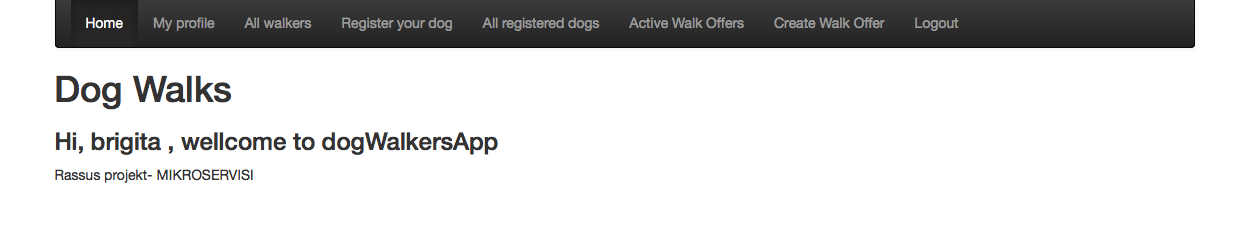
*Slika 5: Primjer ispunjavanja registracije*

Nakon uspješne registracije aplikacija nas šalje na link za login korisnika, pri čemu je potrebno upisati korisničko ime i lozinku, u našem slučaju kao na slici.



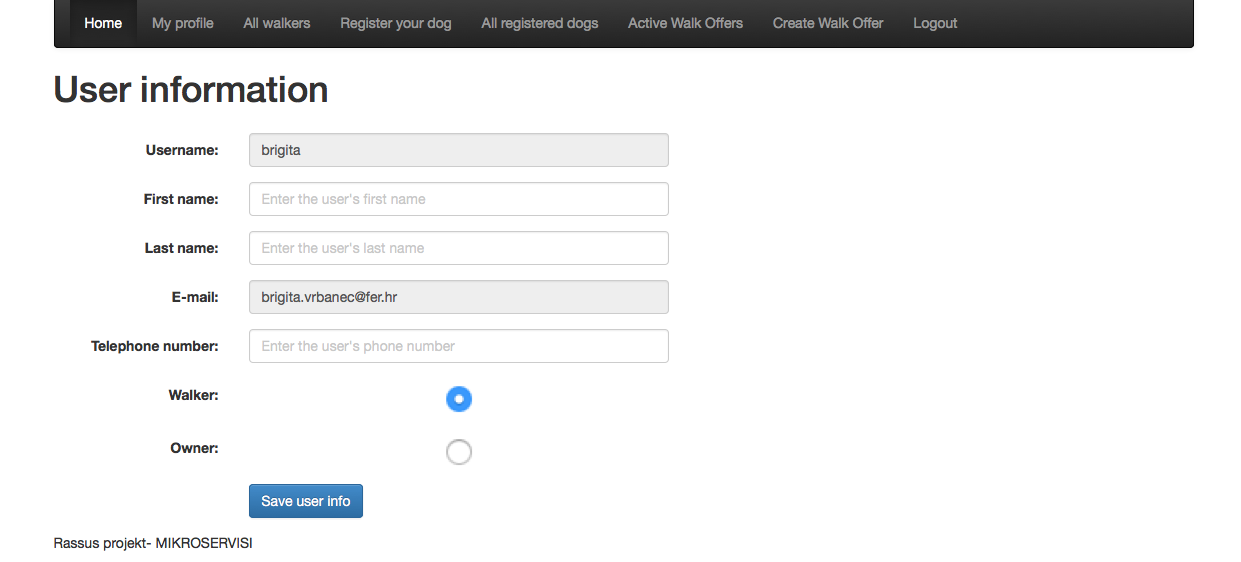
*Slika 6: Forma za prijavu korisnika*

Nakon uspješne prijave putem korisničkog imena i lozinke upisane prilikom registracije, aplikacija nam vraća poruku o uspješnoj prijavi, te sada možemo vidjeti linkove za kretanje kroz samu aplikaciju.



*Slika 6: Početna stranica aplikacije, prikazuje se nakon uspješnog logina*

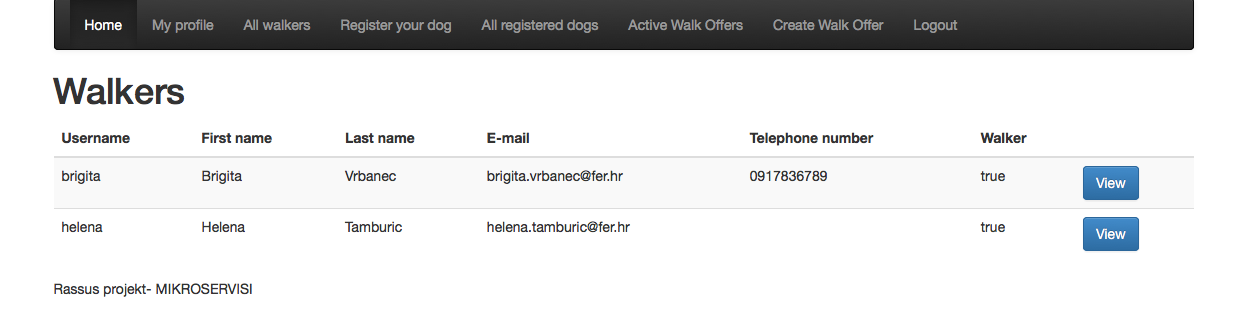
Aplikacija nam sada kroz link prikazan u zaglavlju pod nazivom ''My profile'' nudi prikaz našeg profila, pri čemu možemo mijenjati neke od atributa samog korisnika. Atributi koje ne možemo mijenjati su oni koje je korisnik upisao prilikom registracije, poput korsničkog imena i adrese elektroničke pošte (E-mail).



*Slika 7: Forma za promjenu podataka o prijavljenom korisniku*

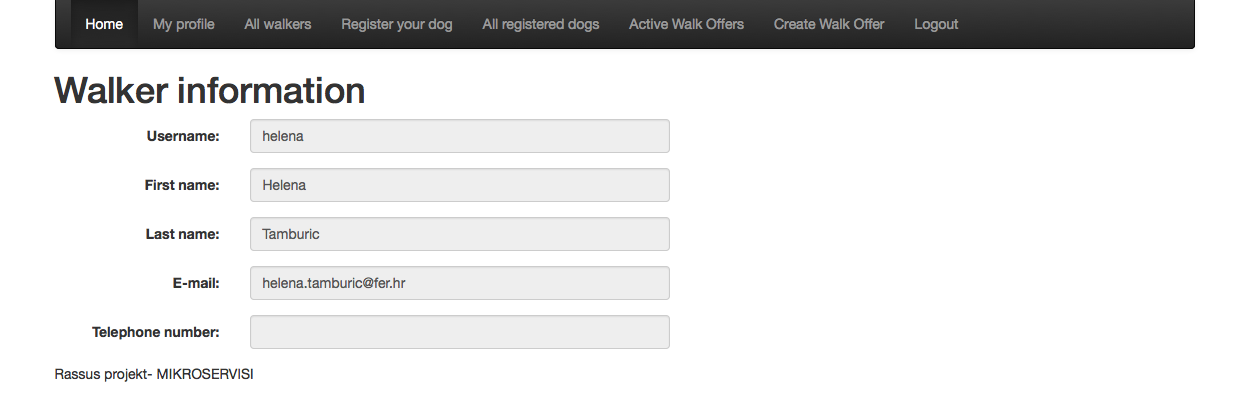
Atributi koje je dozvoljeno mijenjati su ime, prezime, broj telefona te tip korisnika, šetač ili vlasnika psa/pasa.

Nakon toga, klikom na link ''All walkers'', aplikacija nam nudi prikaz svih korisnika šetača.



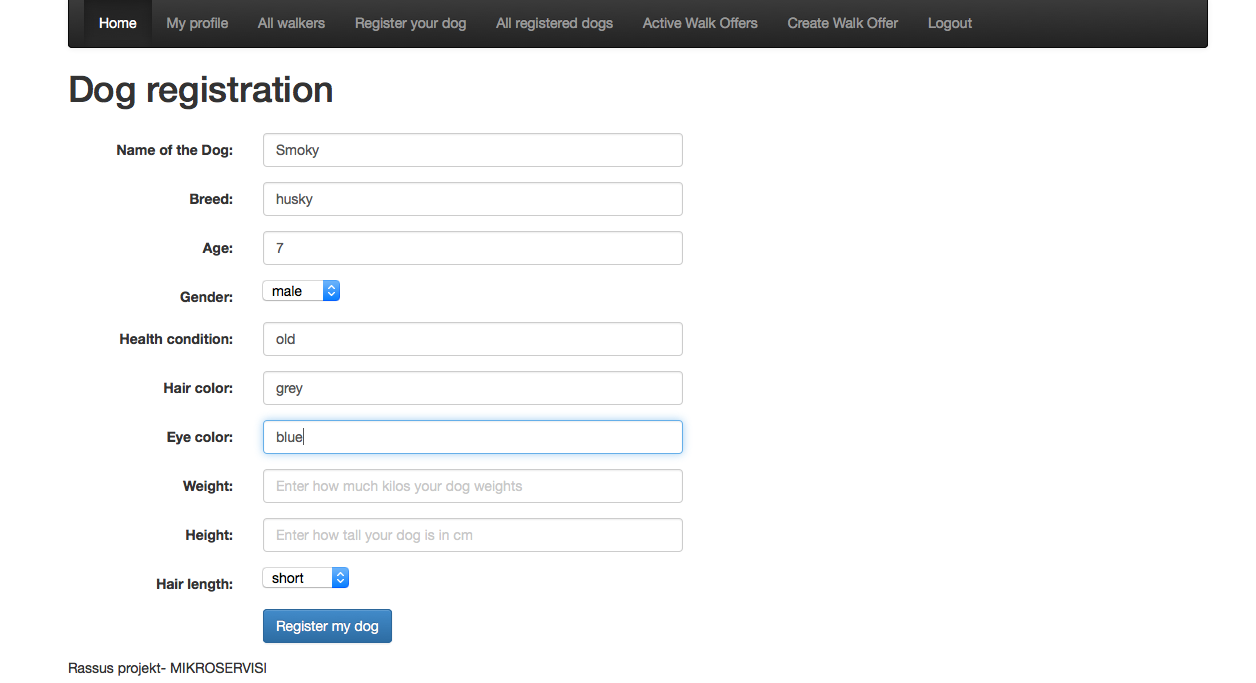
*Slika 8: Lista svih šetača, dostupna na linku ''All walkers''*

Također, putem gumbiju označenih sa ''View'' u zadnjoj desnoj koloni svakog šetača moguće je pristupiti formi s detaljima o istom. Ta je forma slična onoj za promjenu informacija korisnika, međutim putem ove forme informacije o korisnicima ne mogu se mijenjati.



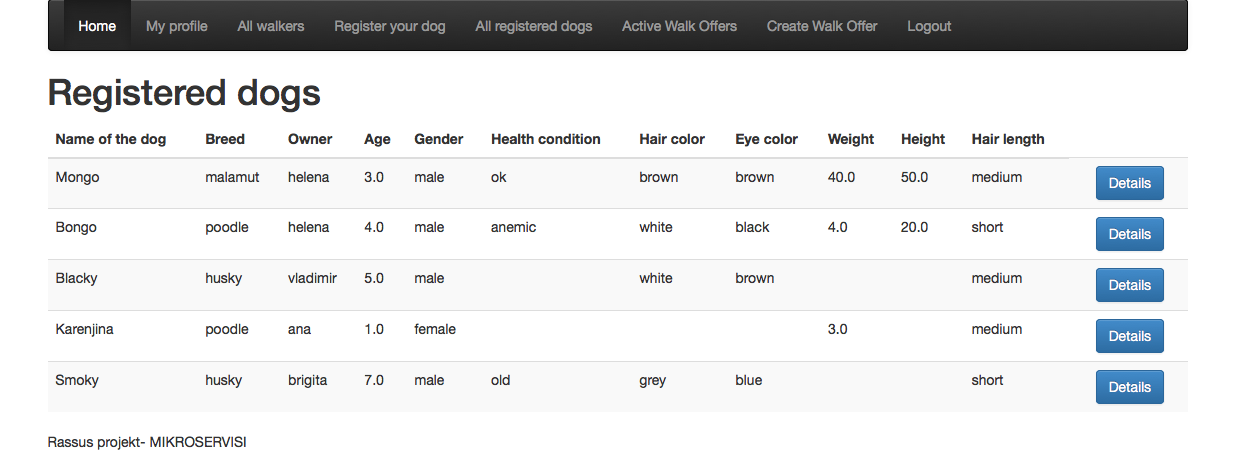
*Slika 9: Forma koja prikazuje informacije o pojedinom šetaču*

Sljedeći link vodi nas na formu na kojoj je korisnik u mogućnosti registrirati psa.



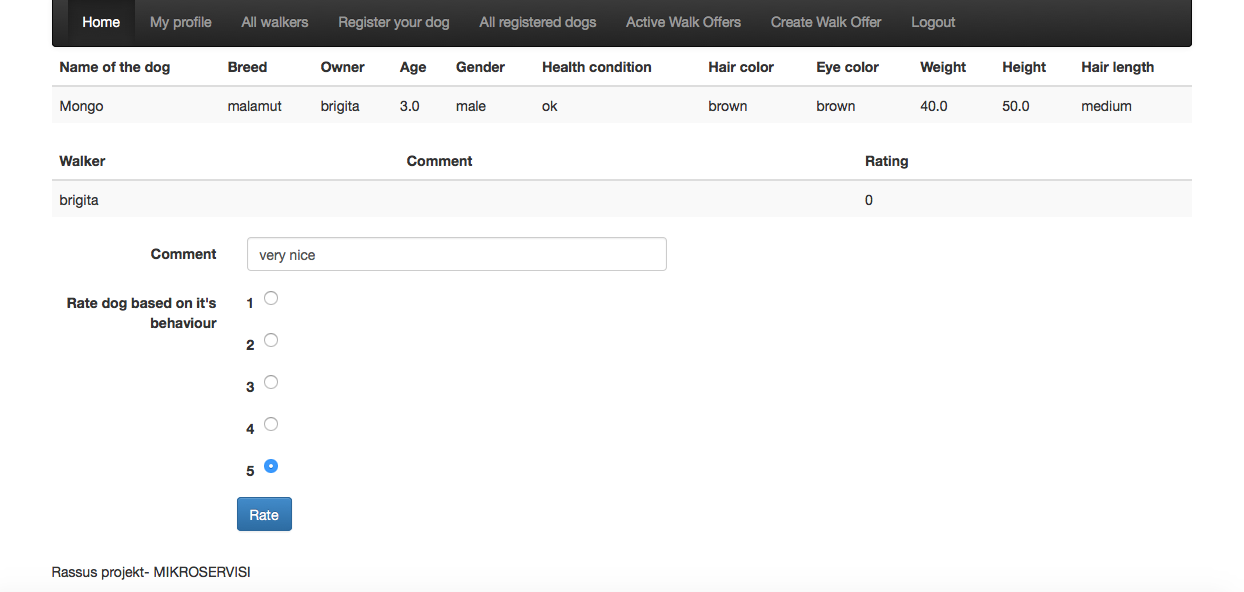
*Slika 9: Forma za registraciju pasa*

Nakon registracije psa, aplikacija nas vodi na listu koja sadrži popis svih registriranih pasa u aplikaciji, inače dostupnu i putem linka ''All registered dogs''.



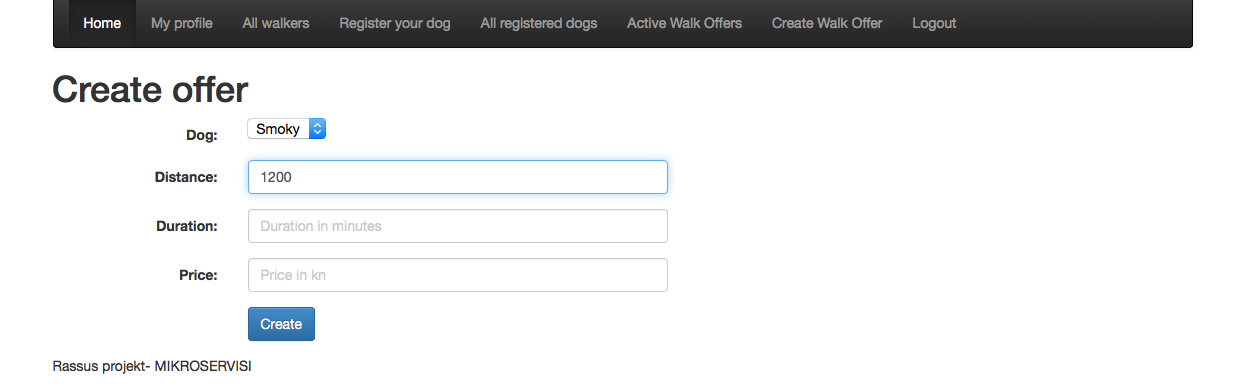
*Slika 10: Lista svih registriranih pasa*

Nakon klika na link ''Details'' prikazuju se detaljne informacije o registriranom psu. Ukoliko je trenutno prijavljani korisnik već šetao tog psa, otvori mu se forma za unos ocjene i komentara psa, u suprotnom se samo izlistaju dostupne ocjene.



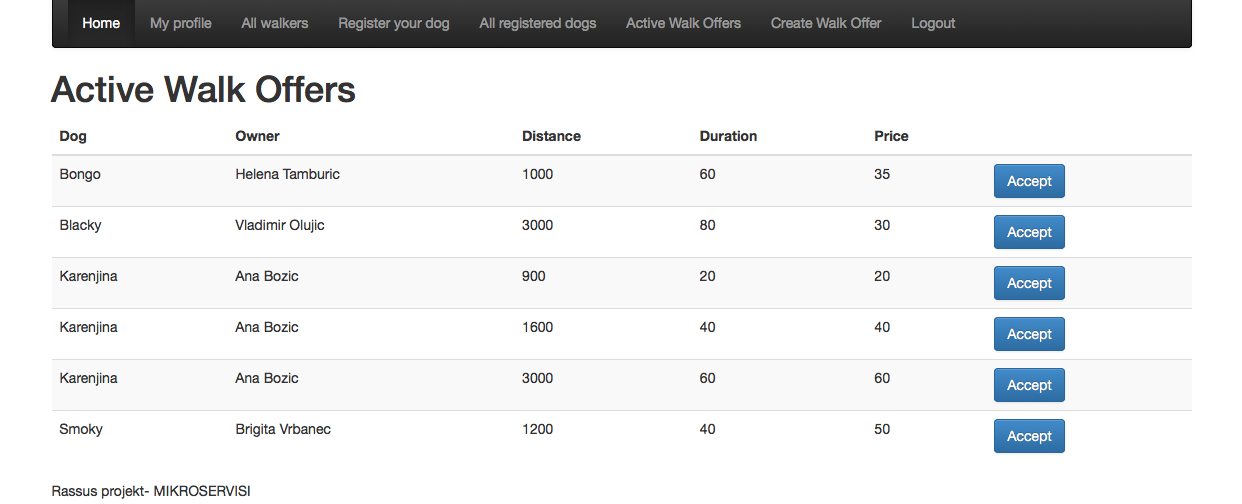
*Slika 10: Detalji o registriranom psu*

Nakon što registriramo psa, sljedeće što možemo u aplikaciji je kreirati ponudu za šetnju psa, pri čemu nam aplikacija sama nudi psa iz liste registriranih.

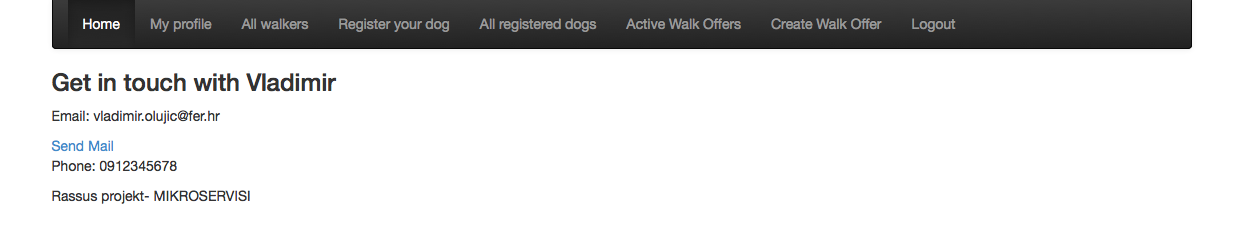


*Slika 10: Forma za kreiranje ponude za šetnju psa*

Nakon što se ponuda kreira, u listi aktivnih ponuda na linku ''Active dog offers'' moguće je klikom na link Accept (prihvati) pristati na oglašenu ponudu za šetnju psa pri čemu se prikazuje ekran sa porukom i kontakt podacima o korisniku koji je tu ponudu oglasio.



*Slika 11: Prikaz aktivnih ponuda šetnji*

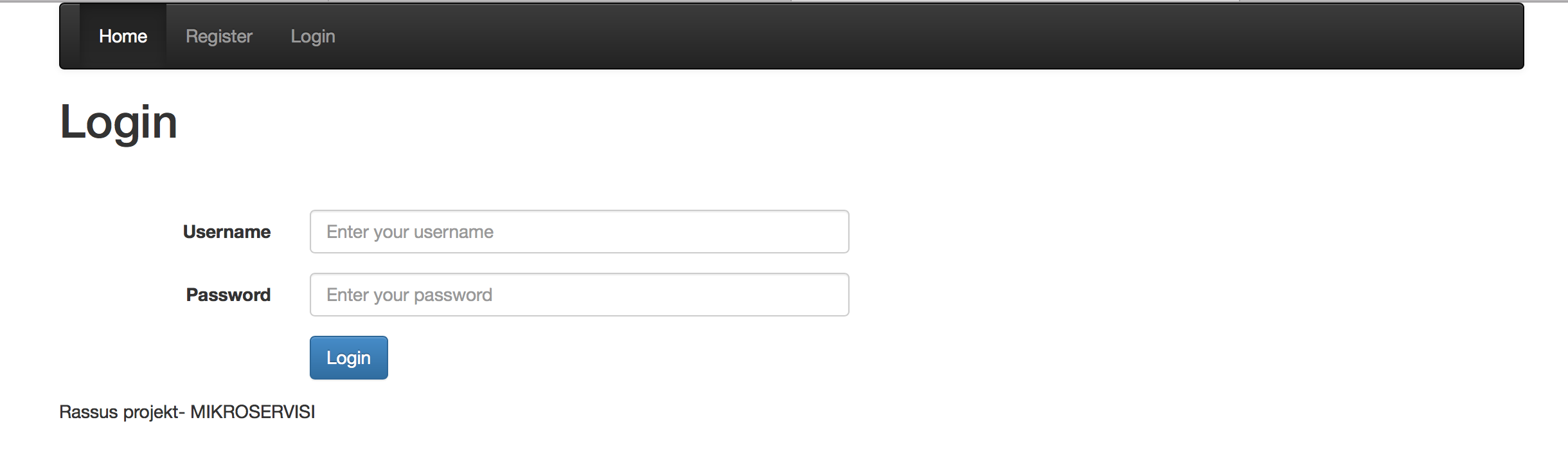


*Slika 12: Prikaz ekrana nakon što korisnik prihvati oglas za šetnju psa*

Klikom na Send Mail otvara se nova poruka e-pošte.

Obzirom da smo ovime pokrili cijeli put kroz navigaciju, ostalo je još jedino odjaviti se iz aplikacije klikom na link Logout.

Nakon klika na taj link, aplikacija nas vraća na ekran za login.

*Slika 13: Prikaz ekrana nakon što korisnik izađe iz aplikacije*

# 7. Zaključak

# Rad na ovom projektu se pokazao iznimno zahtjevnim. Od programiranja prvih aplikacija sa Springom do uklapanja koncepta mikroservisa. Iako je na internetu dostupna određena dokumentacija i vodiči kako postaviti određene komponente i dalje smo teško prihvatili novu arhitekturu mikroservisa. Ona zaista ima neke pozitivne strane npr. da smo u početku neovisno jedni o drugima mogli razvijati svoje servise. Kad je došlo do stvaranja prave web aplikacije uočili smo mane u izradi mikroservisa, te ih je neprestano trebalo mijenjati ovisno o novih zahtjevima koji su se pojavljivali na sustav. Također pri izradi aplikacije bilo je potrebno povezati se međusobno što je opet zahtijevalo naporne konfiguracije sustava kako bi sve rute uspješno radile.

# 8. Literatura

**Mikroservisi**

http://microservices.io/patterns/microservices.html

http://martinfowler.com/articles/microservices.html

https://en.wikipedia.org/wiki/Microservices

https://dzone.com/articles/microservices-architecture-what-when-how

https://dzone.com/articles/scalable-cloud-computing-with-microservices

https://www.nginx.com/blog/introduction-to-microservices/

<http://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html>

http://www.sinarm.net/sto-je-cloud-computing-ili-usluga-u-oblaku/

https://pogledkrozprozor.wordpress.com/2009/08/29/„cloud-computing“-iliprogramska-

rjesenja-u-oblacima/

http://www.cert.hr/sites/default/files/NCERT-PUBDOC-2010-03-293.pdf

<http://www.fer.unizg.hr/_download/repository/RS-2016_08.pdf>

**Heroku**

https://en.wikipedia.org/wiki/Heroku

https://blog.heroku.com/managing\_your\_microservices\_on\_heroku\_with\_netflix\_s\_

eureka

https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku

https://blog.codeship.com/exploring-microservices-architecture-on-heroku/

Isporuka Spring aplikacija na cloud

http://docs.spring.io/spring-boot/docs/current/reference/html/clouddeployment.

Html

**Cloud Foundry, Pivotal, Bluemix**

http://cloudacademy.com/blog/cloud-foundry-benefits/

https://www.cloudfoundry.org/how-can-i-try-out-cloud-foundry-2016/

https://www.youtube.com/watch?v=RC1HX\_7aGa8

https://run.pivotal.io

https://console.ng.bluemix.net

<http://docs.run.pivotal.io/buildpacks/java/gsg-spring.html>

<https://spring.io/tools/sts>

**Spring Boot**

http://projects.spring.io/spring-boot/

https://en.wikipedia.org/wiki/Spring\_Framework#Spring\_Boot

http://alvinalexander.com/blog/post/java/load-spring-application-context-file-java-swing-application

https://spring.io/guides/gs/spring-boot/

https://github.com/spring-projects/spring-boot

https://en.wikipedia.org/wiki/Spring\_Framework

https://www.infoq.com/articles/microframeworks1-spring-boot

https://dzone.com/articles/why-springboot

https://spring.io/guides/tutorials/bookmarks/

https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone