Univerzitet u Sarajevu

Elektrotehnički fakultet

Računarstvo i informatika

**Praktikum - Napredne Web Tehnologije**

**Projektna dokumentacija**

Ragib Smajić

1122/16668

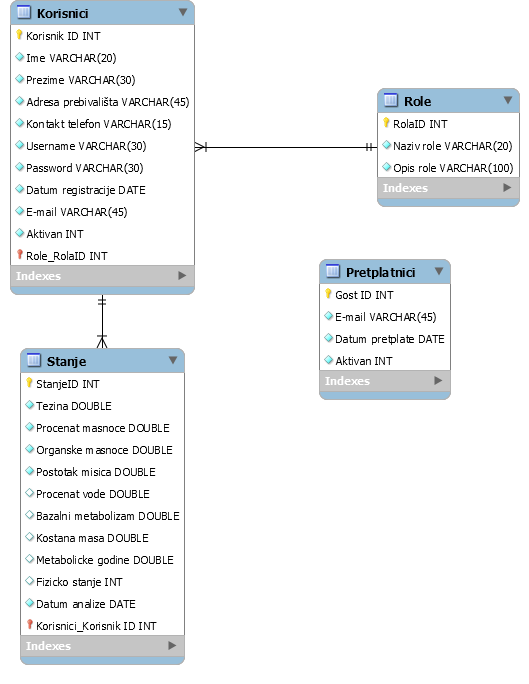
**Zadatak 1**

Jedan od modula koji je sastavni dio aplikacije jeste modul za korisnike. Moj zadatak je bio da u oviru zadatka 1 kreiram isti, odnosno da stavke zadatka 1 realiziram kroz modul za korisnike.

Modul za korisnike je jedan od mikroservisa koji se sastoji od 4 entiteta:

* Korisnik
* Pretplatnik
* Stanje
* Rola

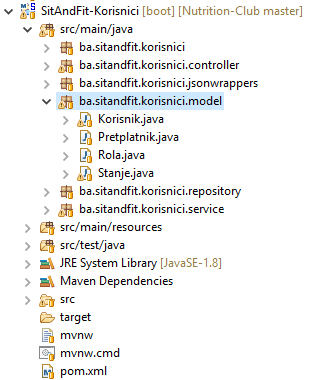
Na sl. 1 je prikazan ER dijagram za pomenuti modul i entitete.



Sl. 1. ERD – Modul za korisnike

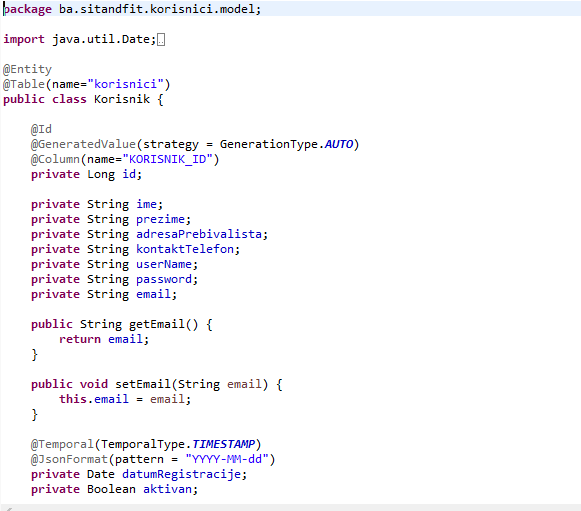
Modul za korisnike komunicira sa modulom za proizvode preko REST servisa koje nudi modul za proizvode.

U Spring Tool Suite (STS) kreirao sam novi Spring Starter Project, koji sam nazvao SitAndFit-Korisnici. U projektu sam kreirao paket model, koji sadži klase koje predstavljaju gore navedene entitete. To je prikazano na slici 1.1.



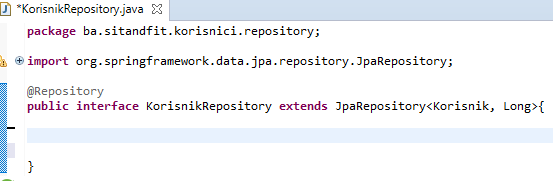
Sl. 1.1. Modeli (Entititeti)

Kako je rađen Code first pristup, odnosno entiteti u bazi podataka se generišu na osnovu kreiranih modela, bilo je potrebno modele označiti odgovarajućim anotacijama. To je urađeno za sve klase model paketa, a na slici 1.2 je prikazana klasa Korisnik.

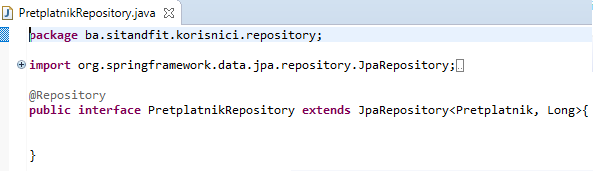


Sl. 1.2 Korisnik klasa sa odgovarajućim anotacijama

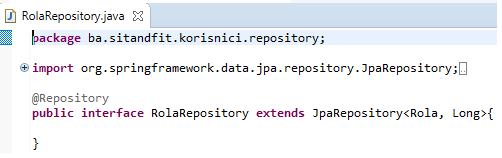
Za svaki od entiteta kreiran je interfejs koji proširuje JpaRepository interfejs, čija implementacija se kreira automatski prilikom builda aplikacije, a koji služi za komunikaciju sa bazom podataka. Na slikama 1.3.1, 1.3.2, 1.3.3 i 1.3.4 su prikazani interfejsi koji predstavljaju specifikaciju za repozitorij koji će biti kreiran automatski.



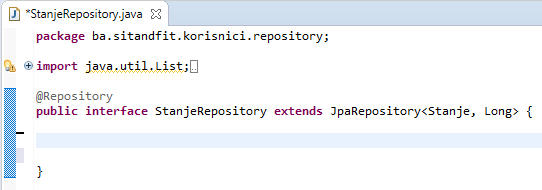
Sl. 1.3.1 KorisnikRepository za korisnike



Sl. 1.3.2 PretplatnikRepository za pretplatnike



Sl. 1.3.3 RolaRepository za role

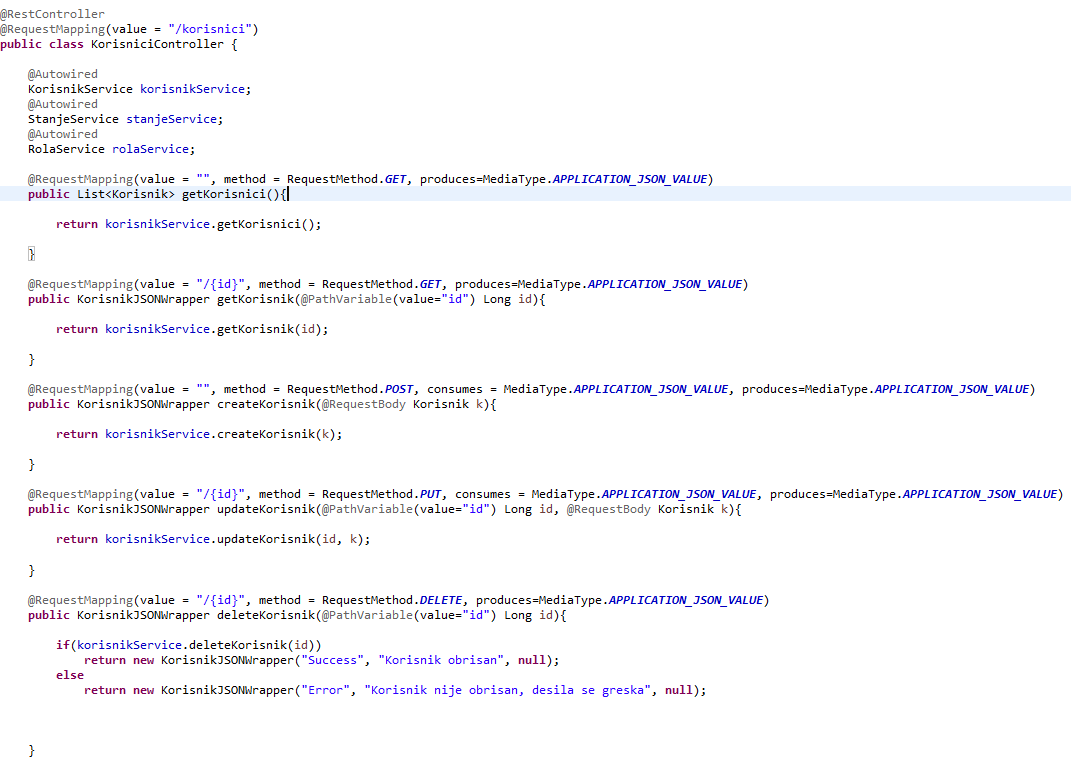


Sl. 1.3.4 StanjeRepository za stanja korisnika

Kako JpaRepository posjeduje osnovne metode za kreiranje, ažuriranje, pretraživanje i brisanje korisnika, čija implementacija će biti kreirana automatski, potrebno je kreirati kontrolere koji će omogućavati osnovne CRUD operacije.

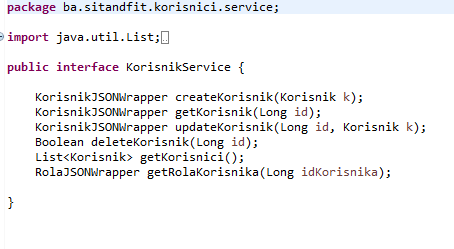
Za svaki od entiteta kreiran je kontroler koji omogućava pristup entitetima na koje se odnosi i za svaki od entiteta kreiran je Service layer koji sadrži svu biznis logiku.

Na slici 1.4 je prikazan kontroler koji na URI „/korisnici“ omogućava pristup resursu korisnicima i pruža osnovne CRUD operacije.



Sl. 1.4 Korisnici kontroler – Osnovne CRUD operacije

Kako kontroler ne bi bio pretrpan kodom koji predstavlja biznis logiku kreiran je layer Service koji enkapsulira istu. Zbog toga se u kontroleru samo poziva odgovarajuća metoda service layer-a, u ovom slučaju klase KorisnikService. KorisnikService (Sl. 1.5) je interfejs koji prestavlja specifikaciju metoda koje konkretna implementacija mora da podrži, te je stoga bilo potrebno kreirati konkretnu implementaciju interfejsa KorisnikService. To je urađeno klasom KorisnikServiceImpl koja je prikazana na slici 1.6.



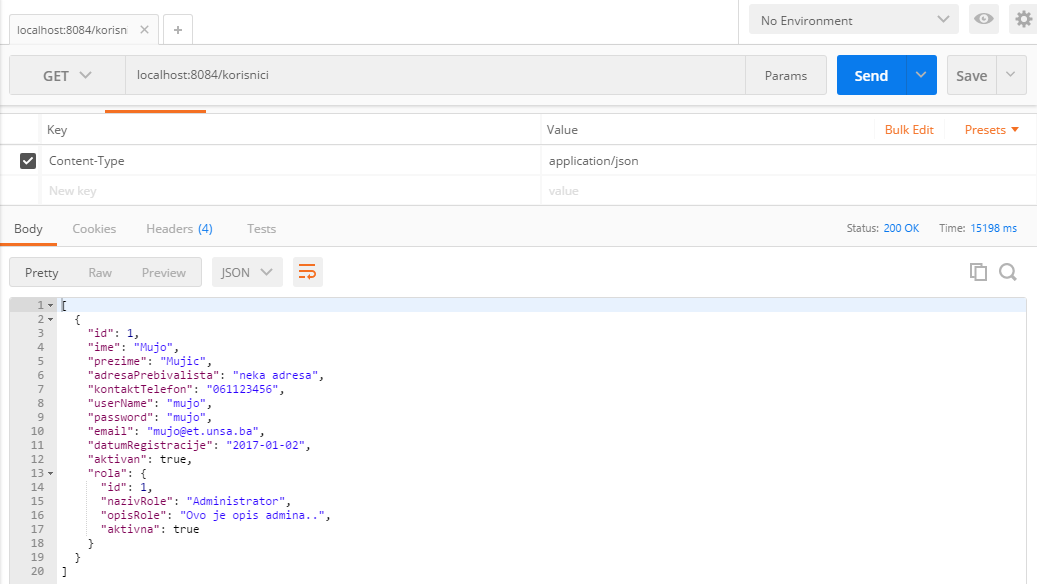
Sl .15 KorisnikService interjefs



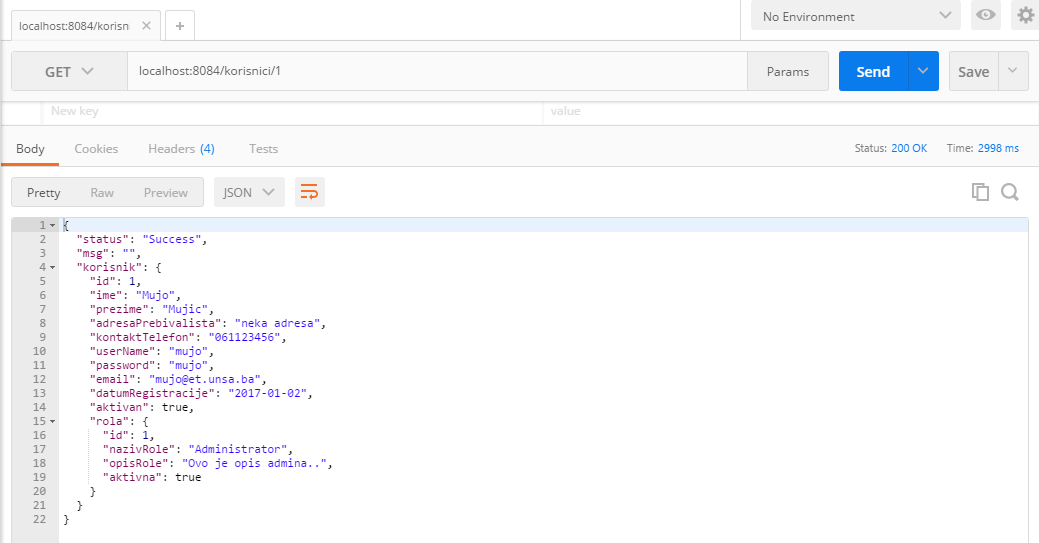
Sl. 1.6 KorisnikServiceImpl

Kreirana klasa (KorisnikServiceImpl) koristi usluge KorisnikRepository-a za dobavljanje podataka iz baze, čija implementacija je automatski kreirana i dodijeljena varijabli korisnikRepository, a pruža usluge klasi KorisnikController. Na taj način se obradi određeni zahtjev korisnika na URI „/korisnici“.

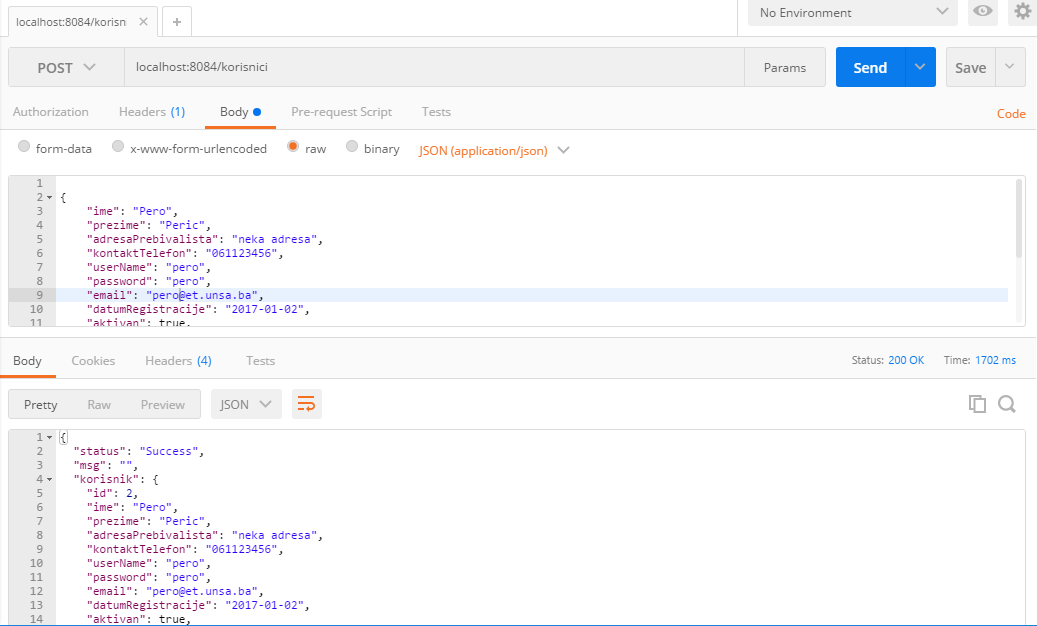
Na slikama 1.7.1, 1.7.2, 1.7.3, 1.7.4 i 1.7.5 su prikazani pozivi upućeni servisu „/korisnici“ u okviru aplikacije Postman.



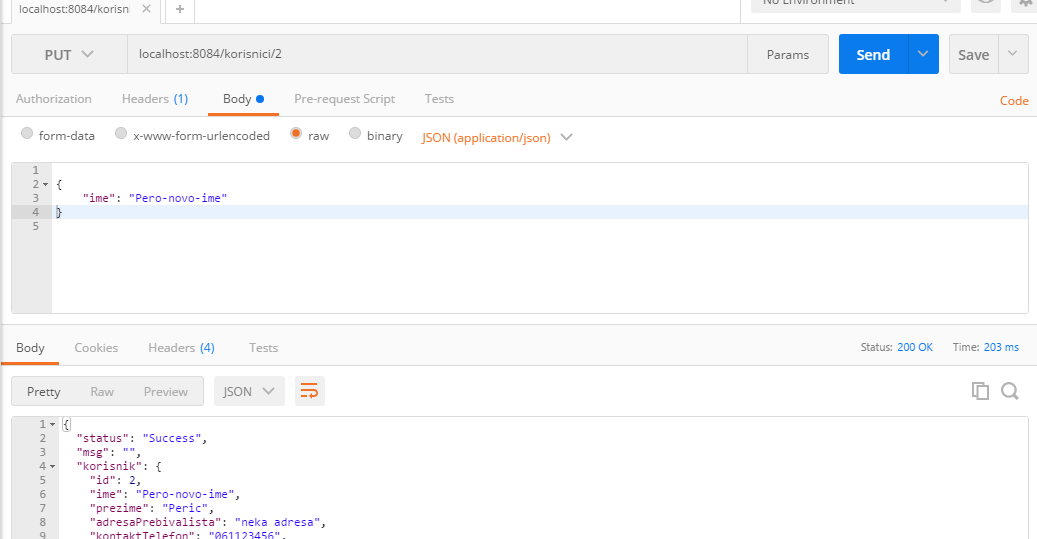
Sl. 1.7.1 GET zahtjev na /korisnici koji vraća sve korisnike iz baze



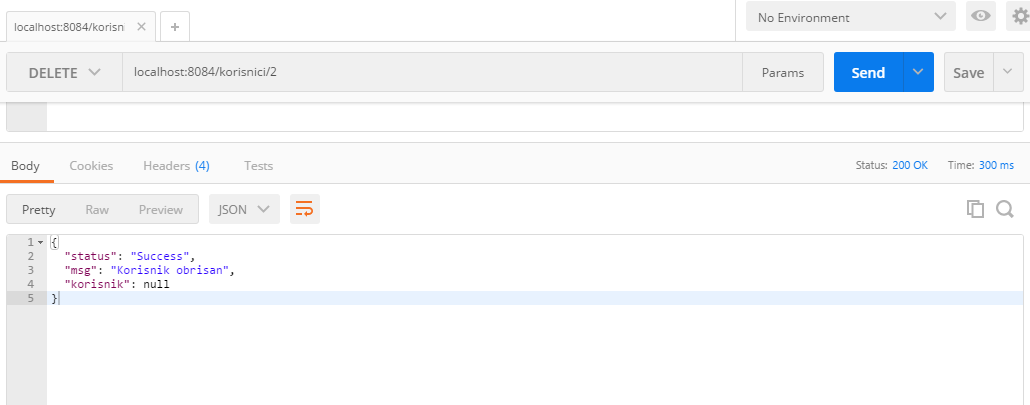
Sl. 1.7.2 GET zahtjev na /korisnici/{id} koji vraća korisnika sa proslijeđenim id-em



Sl. 1.7.3 POST zahtjev na /korisnici za kreiranje novog korisnika



Sl. 1.7.4 PUT zahtjev na /korisnici/{id} za update imena korisnika Pero



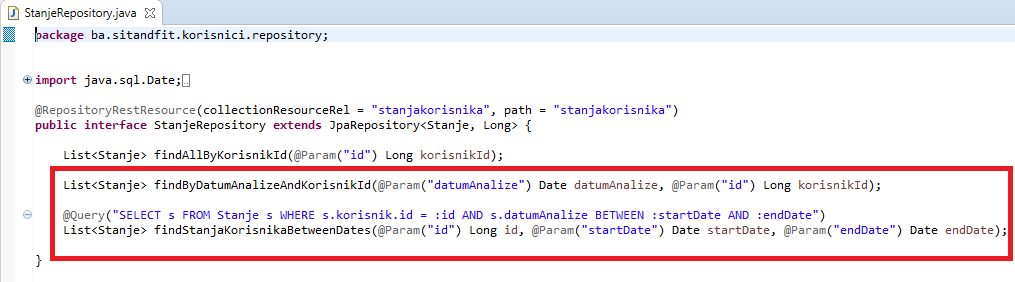
Sl. 1.7.5 DELETE zahtjev na /korisnici/{id} – Brisanje korisnika

Na isti način su realizirani end point servisi i za ostale entitete, odnosno za entitete Rola, Pretplatnik i Stanje.

**Zadatak 2**

Dvije metode koje su izabrane za implementaciju u okviru mikroservisa za korisnike su metoda koja stanja korisnika na određeni datum analize tijela korisnika i metoda koja daje stanja korisnika između proizviljna dva datuma.

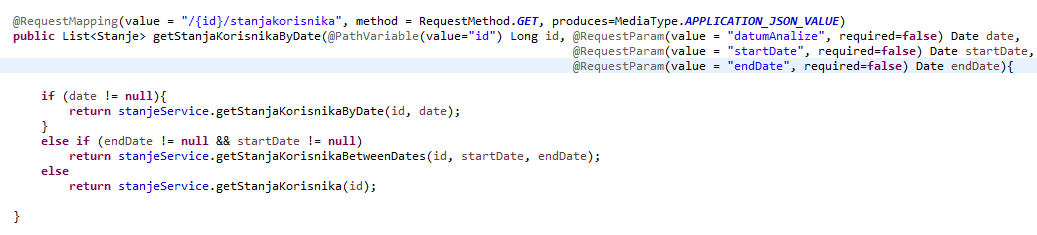
Na slici 2.1 su prikazane navedene metode (metode u crvenom pravougaoniku) u okviru repositorija, odnosno u oviru interfejsa StanjeRepository.



Sl. 2.1 Metode u okviru StanjeRepository interfejsa

Naravno, kako bi se imao pristup metodama preko http-a kreiran je odgovarajući kontroler u KorisnikController klasi, kao i odgovarajuće metode u StanjeService interfejsu.

Kontroler koji je zadužen za ove zahtjeve je prikazan na slici 2.2

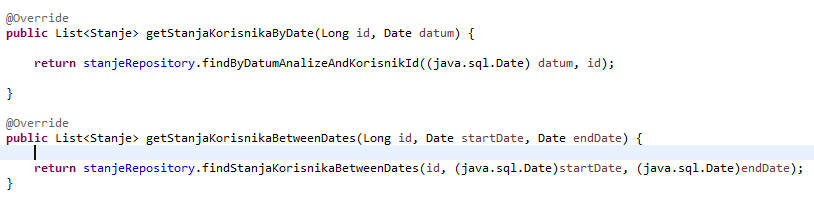


Sl. 2.2 Prikaz kontrolera

Kontroler zahtjeva kao PathVarijablu id korisnika i omogućava da se proslijede 3 dodatna URL parametra na osnovu kojih se odlučuje koja metoda StanjeService-a će se pozvati.

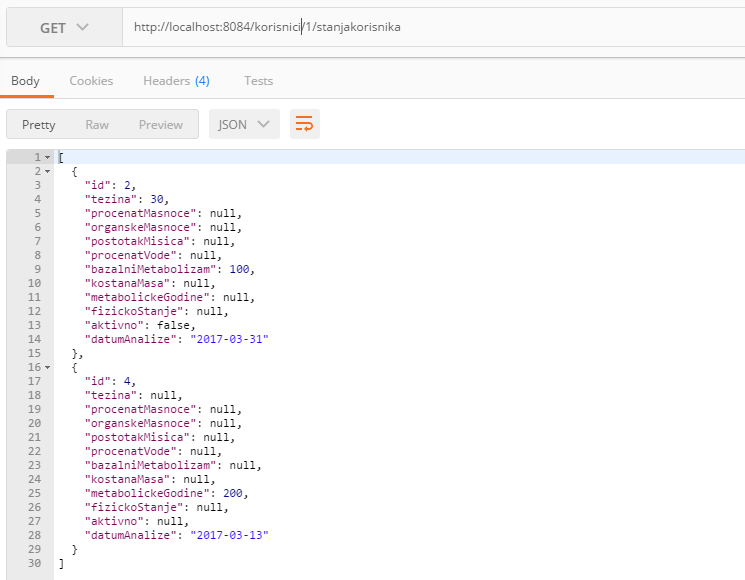
Ako se proslijedi paramterat date, poziva se metoda koja vraća stanja korisnika koja su zabilježena određenog datuma, ako se proslijede 2 parametra startDate i endDate poziva se metoda koja vraća stanja korisnika koja su zabilježena između tih datuma i ako se ne proslije nijedan parametar vraćaju se sva stanja korisnika.

Na slici 2.3 su prikazane metode klase StanjeServiceImpl koje su posrednik izmeđi kontrolera i metoda repozitorija.

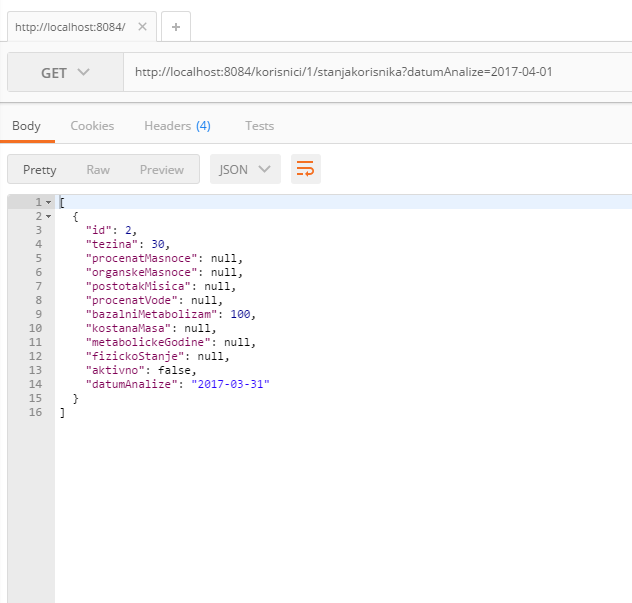


Sl. 2.3 Metode u StanjeServiceImpl – Implementacija StanjeService interfejsa

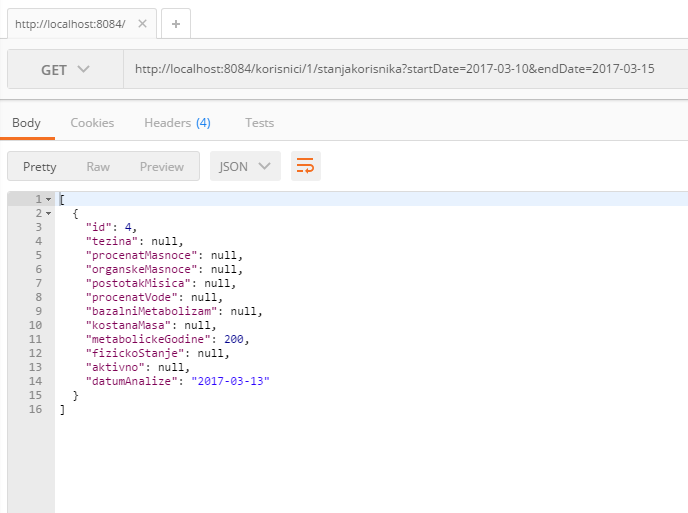
Na slici 2.4.1 je prikazan odziv kada se traže sva stanja korisnika sa id-em 1, na slici 2.4.2 kada se traže stanja na određeni datum i na slici 2.4.3 kada se traže stanja između datuma



Sl.2.4.1 Izlistavanje svih stanja korisnika sa id = 1



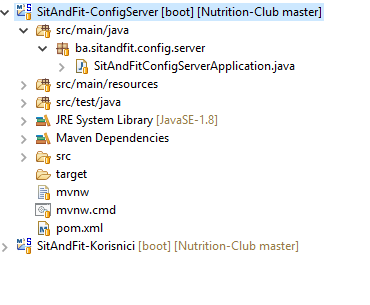
Sl. 2.4.2 Izlistavanje stanja na određeni datum



Sl. 2.4.3 Izlistavanje stanja između datuma

Pored kreiranja dodatne dvije metode u oviru mikroservisa za korisnike, zadatk mi je bio kreirati Spring Cloud Configuration Server, omogućiti osnovnu zaštitu putem autentifikacije i enkripciju konfiguracije.

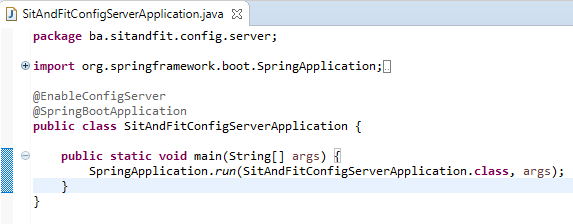
Prije svega bilo je potrebno napraviti novi Spring Boot Starter projekat koji sam nazvao SitAndFit-ConfigServer. To je prikazano na slici 2.5.



Sl. 2.5 Prikaz projekta za Config Server

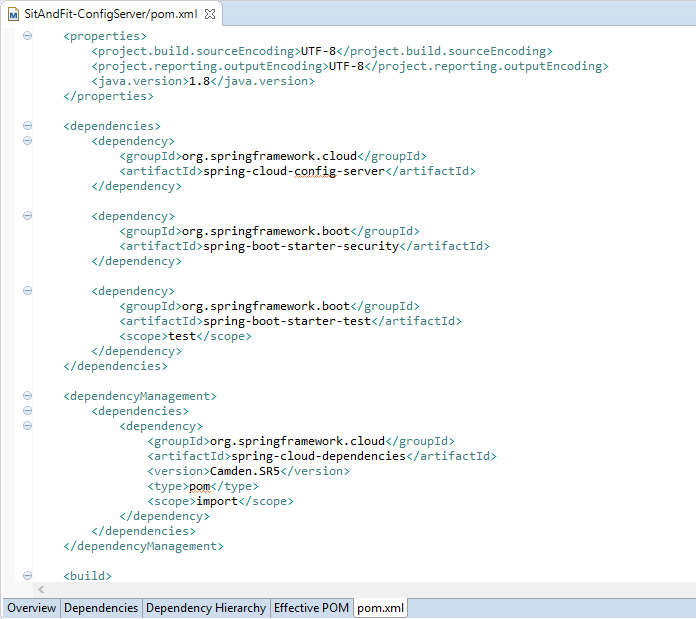
Kako bi se Spring Boot aplikacija ponšala kao konfig server potrebno je u main klasi dodati anotaciju @EnableConfigServer, dodati potrebne dependency-e u pom.xml i specificirati na kojoj lokaciji se nalaze konfiguracijski fajlovi za koje je zaduzen konfiguracijski server.

Na slici 2.6 je prikazana glavna klasa projekta i dodana anotacija.

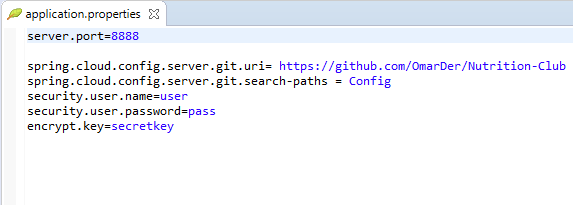


Sl. 2.6 Dodavanje anotacije @EnableConfigServer

Na slici 2.7 je prikazan pom.xml i na slici 2.8 je prikazan application.properties file u kojem je između ostalog specificirana lokacija na kojoj se nalaze konfiguracijski file-ovi za klijente.



Sl. 2.7 Prikaz pom.xml sa dependency-ima



Sl. 2.8 application.properties file koji služi za konfiguraciju samog servera

Prva linija kaže da server radi na portu 8888 što je i uobičajeni port za Spring Config Server.

Druga linija kaže da se konfiguracijski fajlovi za klijenta nalaze na GitHub repozitoriju Nutrition-Club.

Treća linija kaže da je potrebno pretražiti i poddirektorij „Config“ kako bi se našli konfiguracijski fajlovi s obzirom da isti nisu postavljeni direktno u korijen GitHub repozitorija nego se nalaze u folderu Config.

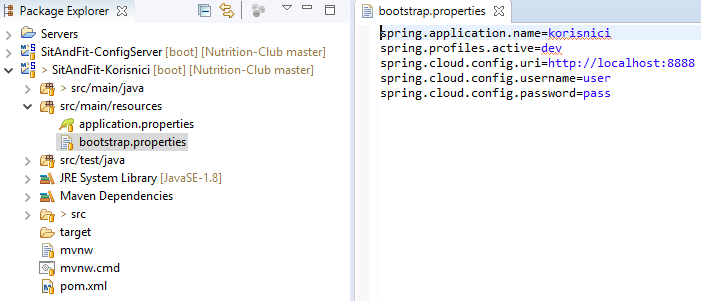
Kako je zadatak bio i omogućiti osnovnu zaštitu putem autentifikacije, u oviru ovog projekta koji se odnosi na Config Server, potrebno je dodati dependency u pom.xml koji se odnosi na Spring Security, koji se može vidjeti na sl. 2.7 ispod dependency-a za Config Server. Kada se doda navedeni dependency omogućena je osnovna zaštita autentifikacijom uz pomoć korisničkog imena i lozinke. Zbog toga četvrta i peta linija u application.properties file-u specificiraju korisničko ime i lozinku koju će koristiti klijenti kada budu od Config servera tražili konfiguraciju.

Pored ovoga, dodatni zahtjev je bio da se može koristiti enkripcija konfiguracije. Kako je po default-u na Config Serveru omogućena enkripcija, u slučaju simetrične enkripcije koja je ovdje izabrana, potrebno je specificirati tajni ključ koji se koristiti prilikom enkripcije i dekripcije.

To specificira šesta linija u application.properties file-u.

Ovim je konfiguracija Config Servera završena.

Kako bi klijenti koristili Config Server i pitali ga za konfiguraciju, potrebno ih je na neki način podesiti da to rade. Konkretno, u slučaju modula za korisnike za koji sam zadužen, napravio sam bootstrap.properties file i u njemu to specificirao. To je prikazano na sl. 2.9



Sl. 2.9 Prikaz boostrap.properties file-a

Ovaj file kaže da je ime aplikacije korisnici, da je trenutni aktivni profil dev, da se config server nalazi na localhost:8888 te da su korisničko ime i lozinka za pristup serveru gore navedeni. Ovo znači da u okviru Gihub repozitorija u folderu Config mora postojati file koji se naziva korisnici-dev.properties koji je konfiguracijski file za ovu aplikaciju. Na osnovu ovoga će modul za korisnike prije svega prilikom svog pokretanja prvo preuzeti konfiguraciju sa Config Servera.

Konfiguracija za mikroservis korisnici koja je kreirana i stavljena na repozitorij u folder Config je prikazana na slici 2.10.

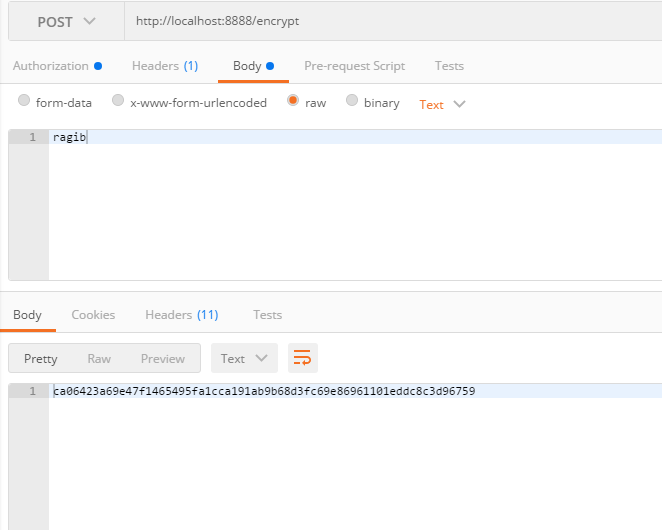


Sl. 2.10 Konfiguracijski file korisnici-dev.properties za mikroservis za korisnike

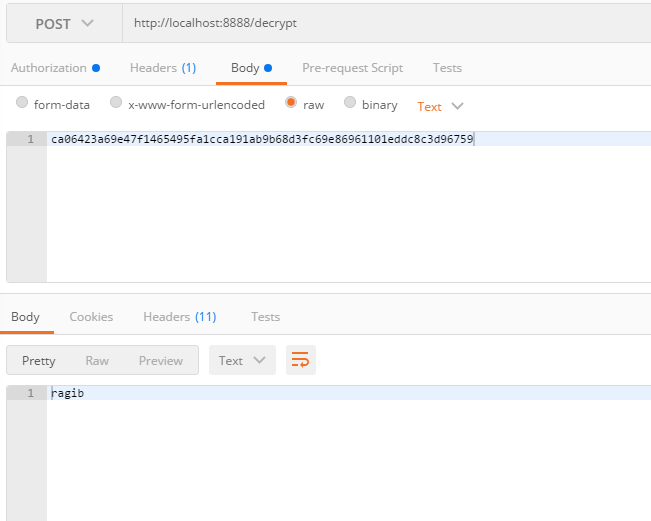
U konfiguraciji vidimo da su polja za username i password za bazu podataka kriptovana obzirom da je to bio jedan od zahtjeva. To je moguće uraditi POST zahtjevom na /encrypt endpoint Config Servera kojem proslijedimo plain text podatak a dobijemo kriptovani podatak i onda takav podatak sa prefiksom {cipher} stavimo u konfiguracijski file.

Prefiks {cipher} je potreban kako bi server znao koja polja je potrebno da dekriptuje prije isporuke konfiguracije nekom od klijenata.

Kriptovanje kao i dekriptovanje podataka upotrebom endpoint-a /encripty i /decrypt je prikazano slikama 2.11 i 2.12



Sl. 2.11 Prikaz enkripcije podataka upotrebom endpointa /encrypt

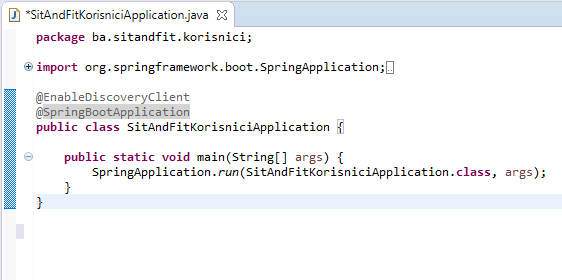


Sl. 2.11 Prikaz dekripcije podataka upotrebom endpointa /decrypt

Da bi ovo bilo moguće autentifikacijsko zaglavlje u okviru POST zahtjeva mora biti postavljeno sa odgovarajućim korisničkim imenom i lozinkom koji su određeni konfiguracijskim fileom Config Servera.

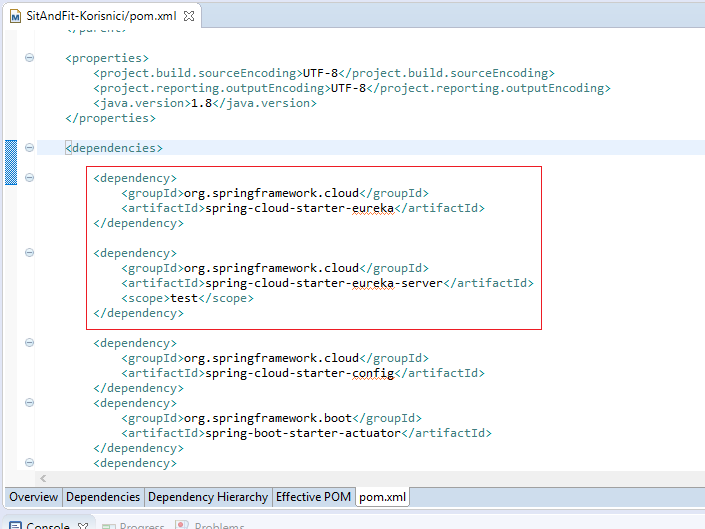
**Zadatak 3**

Kako je kolega imao zadatak da podesi Eureka server, moj zadatak u okviru toga je da podesim mikroservis za korisnike da komunicira sa Eureka serverom. Kako bi to bilo moguće potrebno je iznad glavne klase „SitAndFitKorisniciApplication”, koja zadrži metodu main, staviti anotaciju „@EnableDiscoveryClient”. To je prikazano na slici 3.1.



Sl. 3.1 Anotacija @EnableDiscoveryClient iznad glavne klase

Pored ovoga, potrebno je dodati dependency u pom.xml koji je prikazan na slici 3.2.



Sl. 3.2 Dependency potreban za Eureka client

Drugi dio zadatka 3 je bio da se ostvari komunikacija između servisa. To je urađeno kroz HTTP protokol, korištenjem RestTemplate. Konkretno, modul za korisnike komunicira sa modulom za proizvode i programe i traži od njega listu programa za nekog korisnika. Dakle, to bi se trebalo dobiti na end point-u /korisnici/{idKorisnika}/programi.

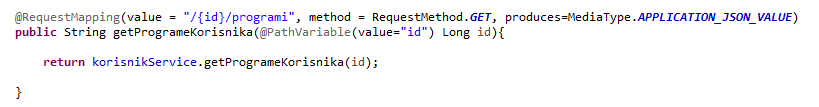
Prije svega napravljena je klasa ProizvodiRESTImpl koja enkapsulira komunikaciju sa modulom za proizvode. Klasa je prikazana na slici 3.3.



Sl. 3.3 ProizvodiRESTImpl klasa za komunikaciju sa mikroservisom „PROIZVODI“

Na sl. 3.3 je prikazana metoda koja je kreirana za gore navedeni servis. To je metoda getProgramiByKorisnikId(Long korisnikId). Ta metoda od Eureka servera saznaje gdje se nalazi mikroservis za proizvode te nakon toga poziva end point /programi/korisnik/{idkorisnika} koji vraća programe za id korisnika.

Na slici 3.4 je prikazan poziv iz kontrolera gdje klasa KorisnikServiceImpl predstavlja posrednika između kontrolera i klase ProizvodiRESTImpl.

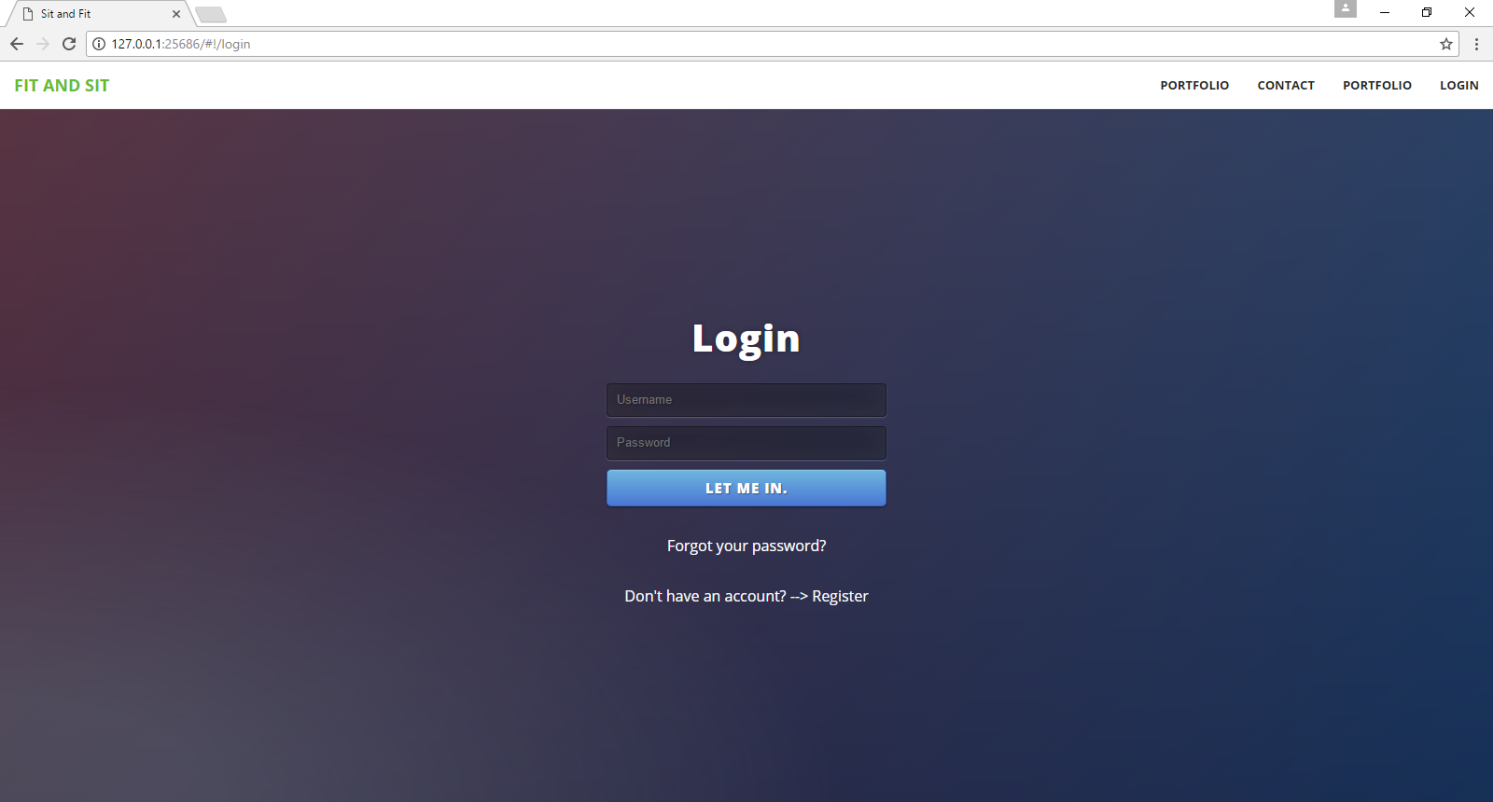


Sl. 3.4 Metoda kreirana u okviru kontrolera za end point /korisnici

**Frontend - AngularJS**

Što se tiče frontenda, moj zadatak je bio da napravim login korisnika nakon implementirane autentifikacije na backend strani.

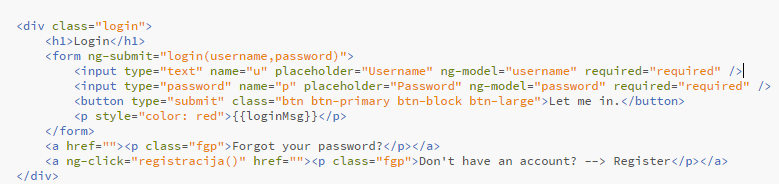
Login korisnika je prikazan na sljedećoj slici:



Login se sastoji od dvije stvari:

* View za login
* Controller za login

Na sljedećoj slici je prikazan View za login.



Pored ovog koda, postoji kod koji je vezan za stil same stranice koji nije toliko važan. Kod se sastoji od 2 input elementa za unos username i lozinke i dugmeta za submit koji su dio forme.

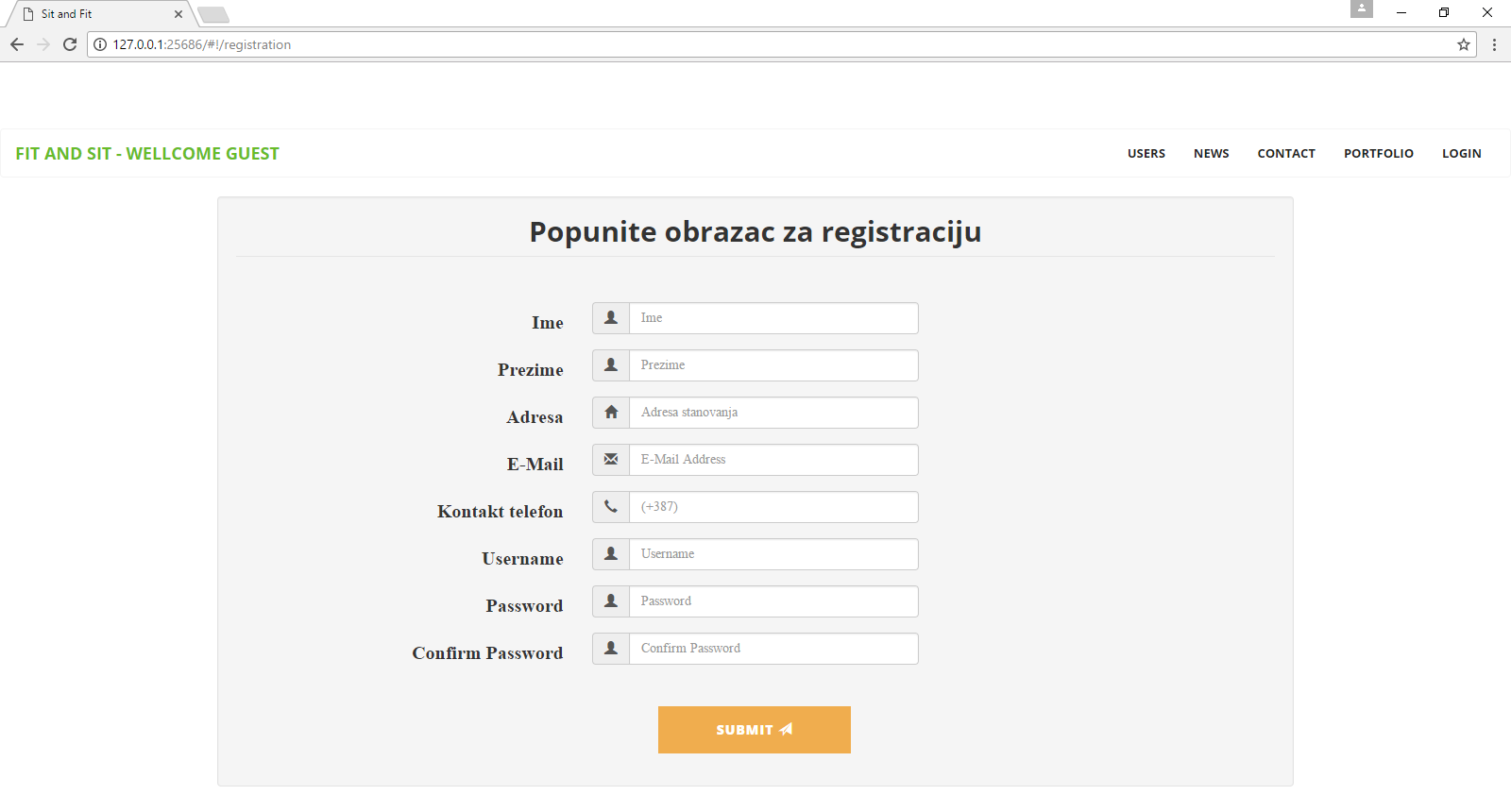
Input elementi imaju na sebi direktivu ng-model na osnovu koje su podaci bind-ani za varijablom username ili password u controlleru. Nakon klika na dugme poziva se funkcija login(usernam, password) sto je određeno direktivom ng-submit na form tagu.

Login controller je prikazan na sljedećoj slici:

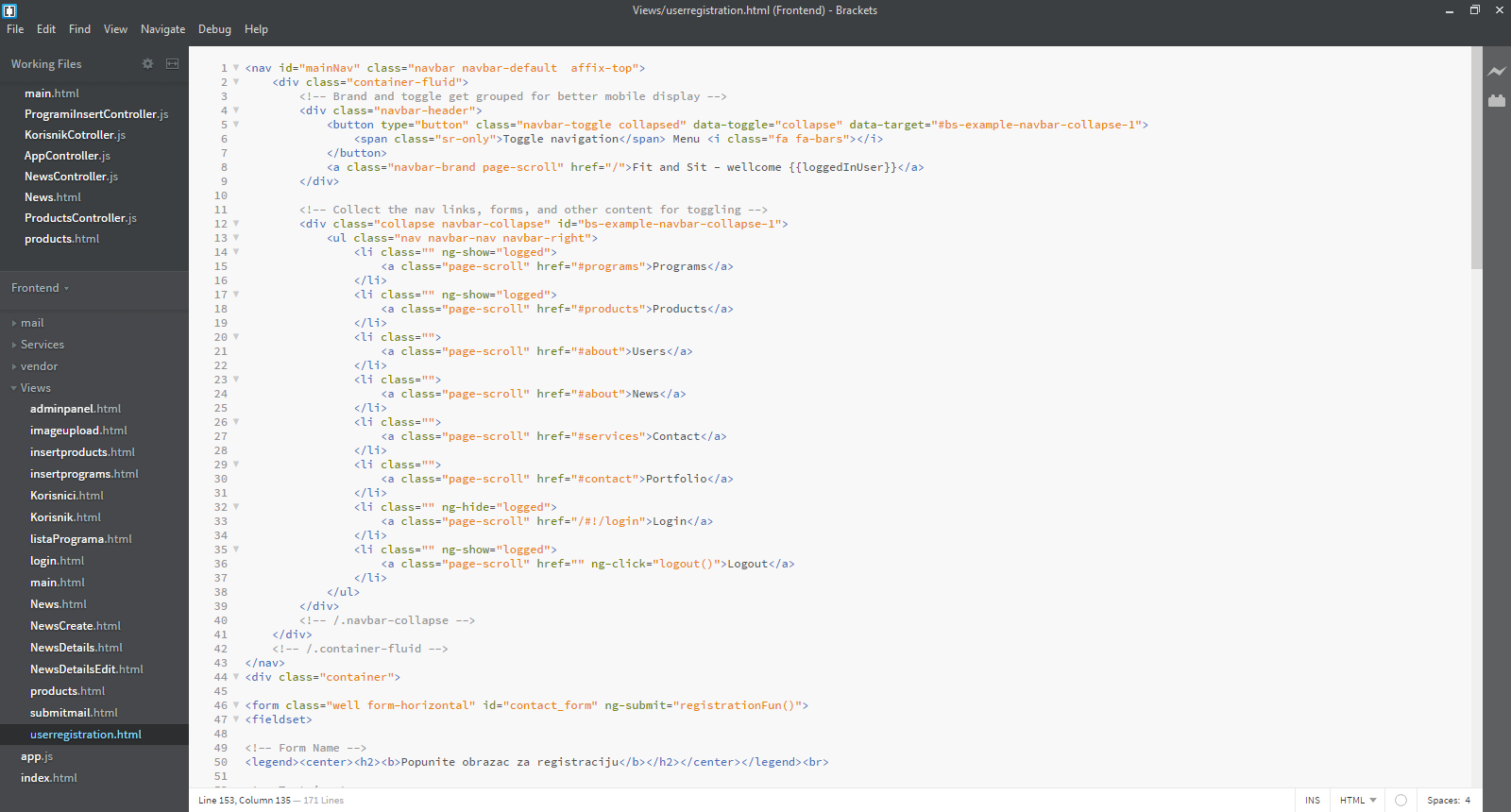


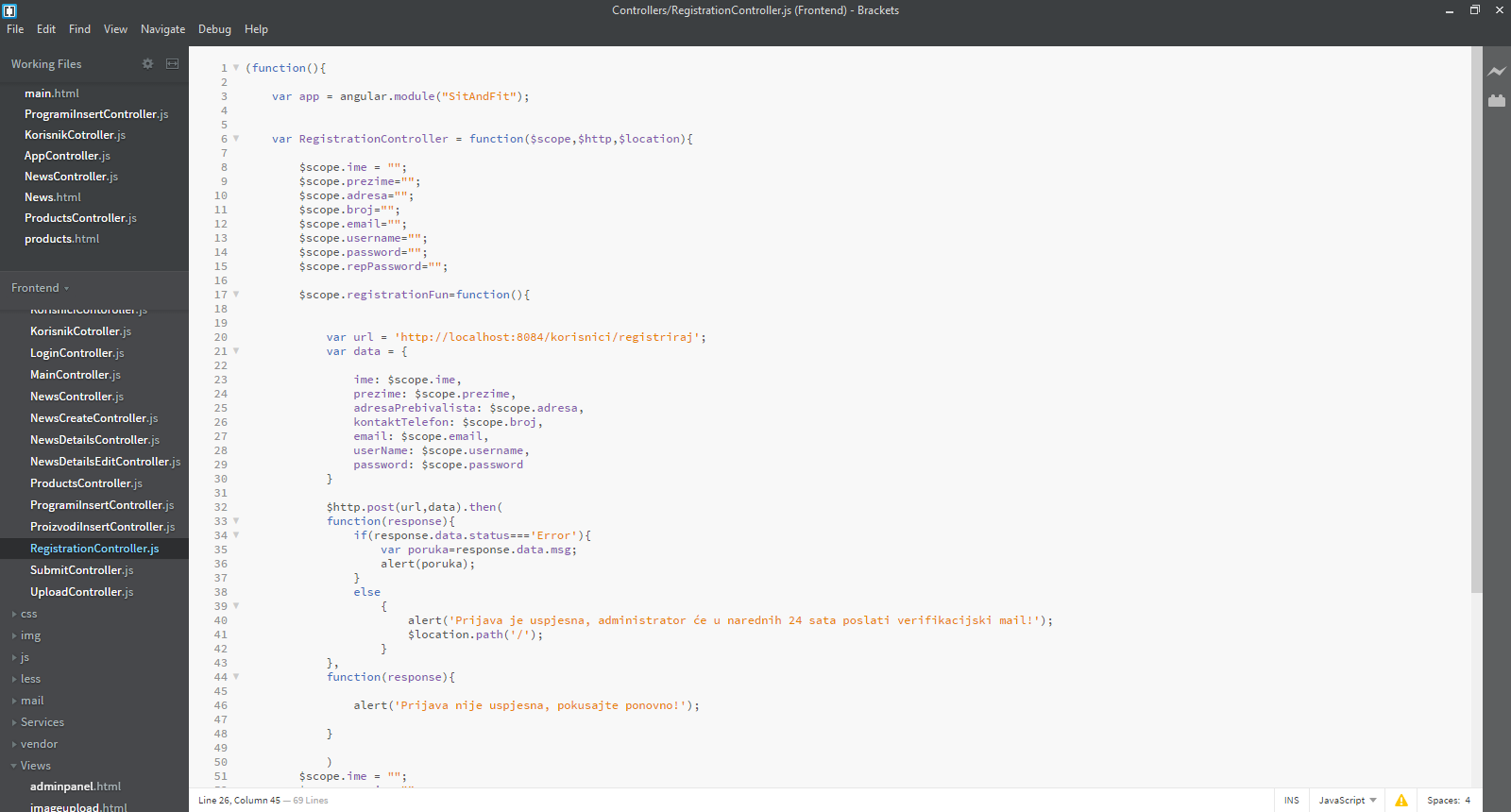
Ovo nije sav kontroler, ali je ovo nazanimljiviji dio. Funkcija koju smo vidjeli u ng-submitu na formi je u kontroleru deklarisana na $scope objektu. Funkcija nakon poziva salje primljene username i password na provjeru servisu za login na backend preko POST zahtjeva. Ukoliko su podaci ispravni kreira se cookie na osnovu sessionStorage-a u kojem se cuvaju podaci o JWT-u i logovanom korisniku. Kasnije se dobijeni JWT šalje uz svaki naredni zahtjev u Authentication headeru.

Ukoliko gost koji je došao na stranicu nema kreiran račun, može ga kreirati klikom na link „Don't have an account?“, nakon čega mu se otvara stranica prikazana na sljedećoj slici:



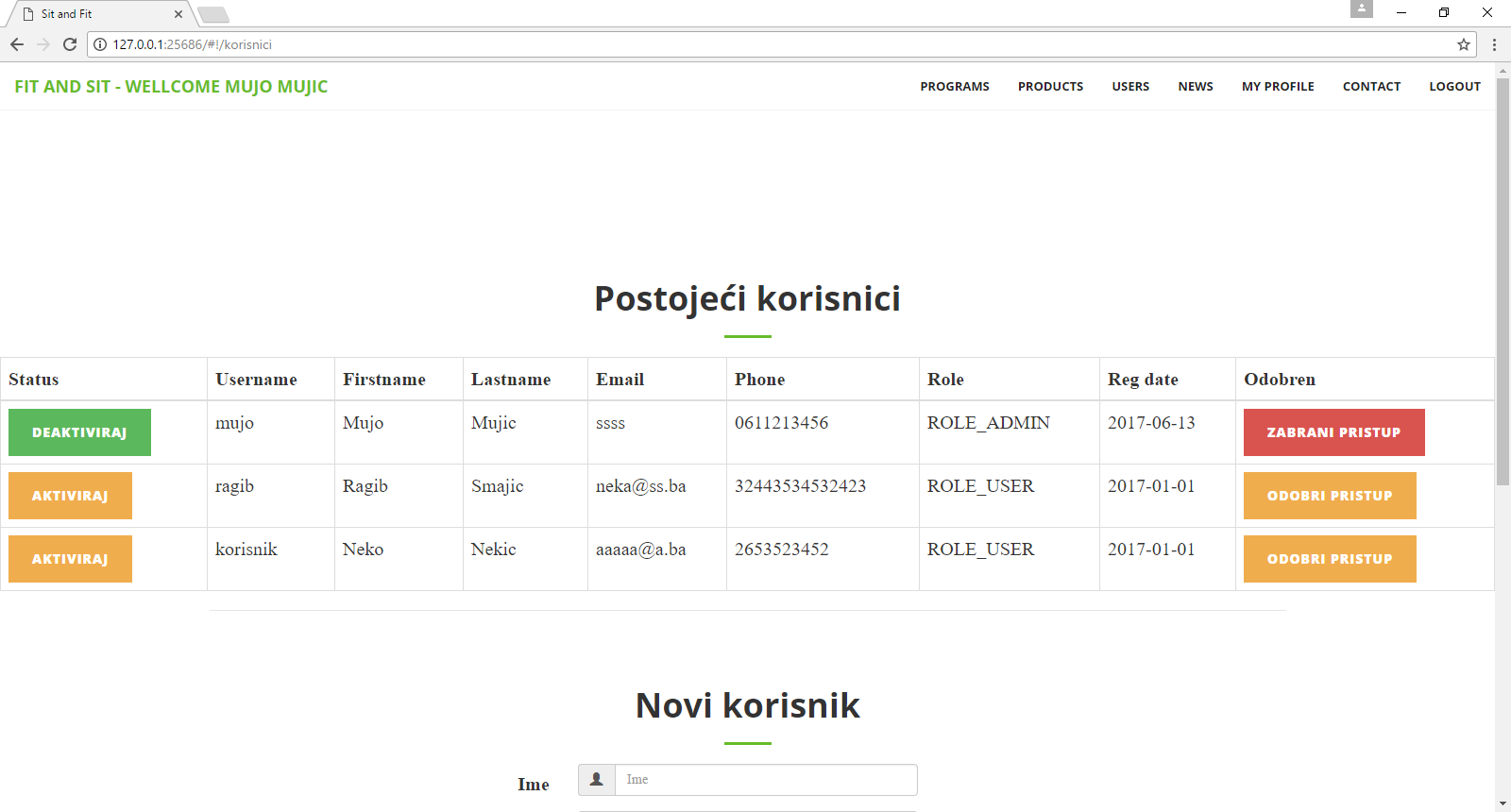
Kod koji predstavlja view i kod koji predstavlja controller za ovu funkcionalnost su prikazani na sljedece dvije slike.





Ovo radi slično kao i login. Dakle, post zahtjevom se salju podaci na backend odakle se spremaju u bazu podataka, nakon čega korisnik koji se zeli registrirati dobija mail da registracija mora biti odobrena od strane administratora.

Prije nego što ima mogućnost da pristupi aplikaciji, potrebno je da administrator odobri zahtjev za registraciju, otvaranjem administratorskog panela koji je prikazan na sljedećoj slici. Korisnik je o svemu tome obaviješten mailom koji se šalje sa backenda kada on submit-a svoju registraciju.

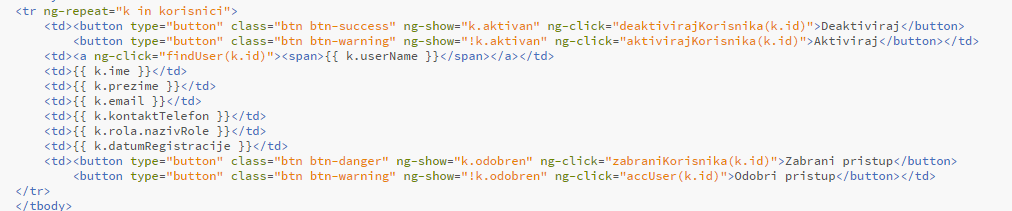


Dakle, vidimo da postoje korisnici koji jos nemaju odobren pristup aplikaciji. Admin odobrava pritup aplikaciji klikom na dugme odobri koje se nalazi u zadnjoj koloni u redu za odgovarajuceg korisnika.

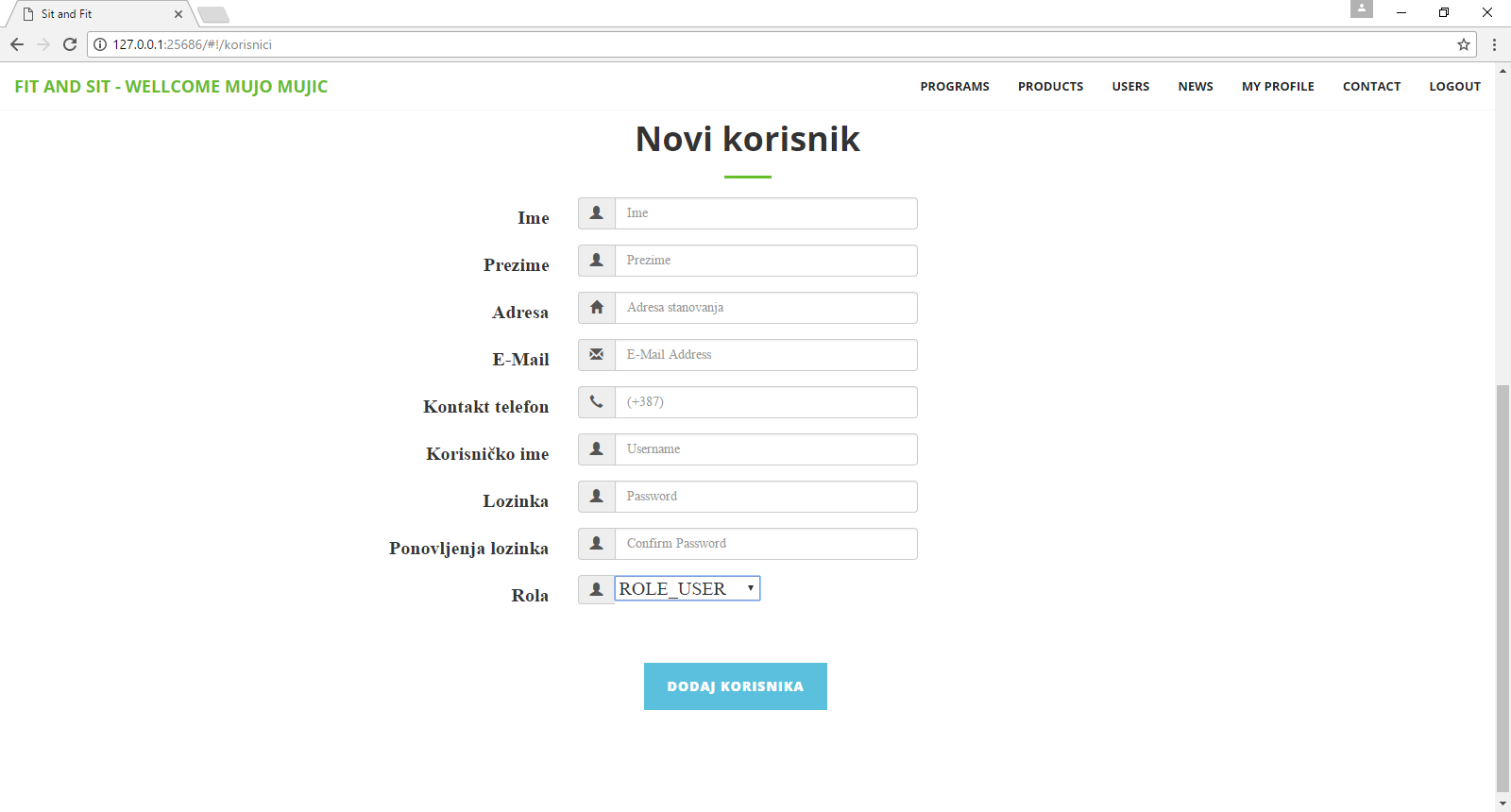
Funkcije u kontroleru za deaktiviranje, aktiviranje i zabranu korisnika su prikazane na sljedećoj slici.



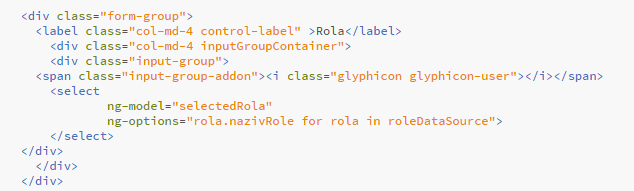
Na sljedećoj slici je prikazano kako se iste pozivaju iz html, tj. View-a.



Na ovom admin panelu moguće je i dodavanje novog korisnika od strane administratora, pri čemu je taj korisnik automatski odobren i ima rolu koju mu postavi administrator. To je prikazano na sljedećoj slici.

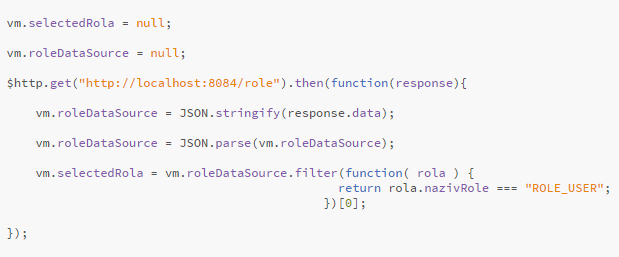


Na dijelu Rola, moguće je izabrati rolu koju ce imati novi korisnik na osnovu koje dobija određen prava sa aspekta pristupa aplikaciji. Kako je taj kod malo zanimljiviji isti je prikazan na sljedećoj slici za view.



Dakle role koje postoje u bazi se dinamički popunjavaju u ovaj combobox ir roleDataSource objekta koji je definisan u kontroleru koji ce biti prikazan kasnije. Na osnovu 2-way bindinga sve sto se desava u kontroleru sa roleDataSource desava se i na view-u. Na select je stavljena ng-model direktiva na osnovu koje će u kontroleru biti selektirani izbor u selectedRola varijabli u okviru $scope objekta.

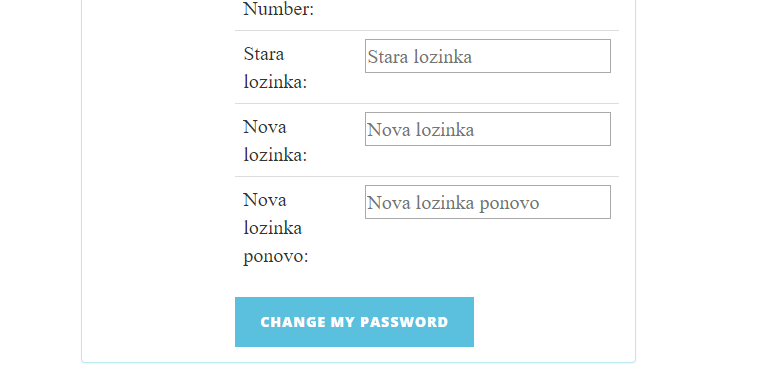
Controller koji stavlja podatke u roleDataSource je prikazan na sljedećoj slici.



U kodu se prije svega deklarisu ranije pomenute varijable. Vm varijabla je u ovom slučaju $scope.

Dakle, pristupa se web servisu na backendu za role, koji vraća sve role i te role se stavljaju u roleDataSource varijablu. Nakon toga se iz tog niza varijabli koji je kreiran uzima rola za obicne korisnike da ona bude defaultn-a. Iz tog razloga je ona prikazana u comboboxu kada se stranica ucita. Dakle ta rola je inicijalno defaultna.

Svaki korisnik moze da otvori svoj profil klikom na My Profile. U tom dijelu ima mogućnost da promijeni svoju lozinku. To je prikazano na sljedećoj slici.



Dakle da bi to uradio, mora da posjeduje staru lozinku i da unese novu. HTML kod kojim je ostvaren navedeni prikaz je prikazan na sljedećoj slici.



Na slici se moze vidjeti da se nakon klika na dugme Change my Password poziva funkcija promijeniPassword koja je definisana u okviru kontrolera. Funkcija je prikazana na sljedećoj slici.



Funkcija provjerava da li je stara lozinka ispravna, da li je nova lozinka uredno potvrdena i ako je sve ispunjeno salje put request pri cemu salje json file koji sadrzi novi password. Problem je ovdje bio sto je inicijalno na backendu bilo podeseno da endpointu /korisnici/\*\* moze pristupati samo admin. To se prilikom implementacije promjene passworda moralo narusiti, ali se provjera prebacila u funkciju.

Kod funkcije, tj. Servisa na backendu koji je zaduzen za ovo je prikazan na sljedecoj slici.



Prilikom uzluzivanja ovog zahtjeva bilo je potrebno provjeriti da li je id koji se nalazi na path-u URL endpointa jednak id-u korisnika u JWT-u. Ako je to slucaj onda se zahtjev moze realizovati. U suprotnom neko drugi korisnik pokusava da promijeni password nekom drugom korisniku. To upravo radi ovaj servis, pri čemu ukoliko je korisnik prijavljen kao administrator on može bilo kome promijeniti šifru.

Da bi se ostvarile ove funkcionalnosti koristena je bibiloteka za parsiranje JWT-ova. Dependency koji je bilo potrebno dodati je prikazan na sljedećoj slici.

