



Razvoj web aplikacija

Korišćenjem angular js i play java frameworka

Tema: Servis Za evidenciju prijavljenih ispita

Vitor Tomić | Filip Trifun**o**vić | Internet Tehnologije | 01/07/2017

Sadrzaj

[Uvod 2](#_Toc486719261)

[PODEŠAVANJE RAZVOJNOG OKRUŽENJA 3](#_Toc486719262)

[ANatomija play aplikacije 3](#_Toc486719263)

[Razvoj servisa za upis ocena u play frameworku 4](#_Toc486719264)

[Slucajevi Koriscenja 5](#_Toc486719265)

[Struktura softverskog sistema – Konceptualni (domenski) model 6](#_Toc486719266)

# Uvod

U ovom dokumentu biće objašnjen proces razvoja web aplikacije koja se sastoji od serverskog dela napravljenog u play java frameworku kao I front end angularJS aplikaciji koja komunicira asinhronim pozivima sa bekendom.

Play framework je moderan java I scala framework namenjen za projektovanje skalabilnih web servisa. Odlikuje ga da ima veliki broj asinhronih neblokirajućih poziva, slično NodeJS tehnologiji, što je omogućeno korišćenjem aktor frameworka (aktori su apstrakcija konkurentnosti, praktično predstavljaju nezavisne konkurentne procese sa kojima aplikacija može da komunicira isključivo slanjem I primanjem poruka da bi se izbegli race conditioni I modifikovanje stanja).

Play biblioteke za web pozive kao I čitanje baze itd su većinom asihroni I neblokirajući, što znači da aplikacija ne blokira threadove pri izvršavanju ovih operacija što dalje omogućava da aplikacija servisira mnogo veći broj zahteva sa mnogo manje resursa.

Ovo je pogtovo pogodno za implementiranje mikroservisne arhitekture gde se naša aplikacija sastoji od mnogo manjih koje međusobno komuniciraju preko http protokola.

AngularJS je javascript framework baziran na MVC ili MV-whatever modelu koji omogućava razdvajanje servisnog sloja aplikacije od samog prikaza. Angular omogućava rutiranje na klijentskoj strani, dependency injection singleton servisa u kontrolere za prikaz I komunicira sa bekendom u vidu asinhronih poziva. Ovakva struktura aplikacije nam omogućava da jedan isti bekend servis opslužuje I web aplikaciju I mobilne aplikacije.

AngularJS takođe uključuje korisne feature kao što su diretive I komponente koje predstavljaju celine koje mogu da se reusuju, filtere što su funkcije koje modifikuju neki ulaz bez modifikovanja originalnog objekta(korisno npr za prikaz datuma u određenom formatu na nekom ekranu bez pravljanja novih objekata ili modifkovanja postojećih) kao I veliki broj pomoćnih funkcija I biblioteka.

Moderni javascript podržava lokalni storage, websockete pa cak I web workere kao način da konkuretno izvršavamo procesorski zahtevne procese koji bi inače blokirali izvršavanje standardnog javascripta, to sve znači da je danas moguće u javascriptu uz pomoć odgovarajućih biblioteka I frameworka napraviti bogate I moćne aplikacije.

## PODEŠAVANJE RAZVOJNOG OKRUŽENJA

Da bi se počelo sa razvojem aplikacija na java platformi prvo je potrebno instalirati java development kit ili JDK.

Play aplikacije se kompajliraju I izvrsavaju uz pomoc SBT-a (Scala build tool). SBT je build tool sličan Mavenu koji prvenstveno služi za dovlačenje I instalaciju dependecnya.

SBT je aplikacija komandne linije tako da kada se jednom instalira moze joj se pristupiti pozivanjem komandse sbt iz command prompta.

Najbrži način da se započne nova play aplikacije je da se klonira neka od postojećih seed aplikacija. Seed aplikacije sadrže unapred konfigurisanu I napravljenu osnovnu strukturu koju onda dalje možemo menjati po potrebi. Postoji više različitih seedova zavisno od toga kakvu tačno aplikaciju hoćemo da pravimo I možemo ih naći na sledećoj adresi:

<https://playframework.com/download#examples>

Kada jednom svučemo naš seed projekat dovoljno je da navigujemo u konzoli na root folder (onaj koji sadrži aplikaciju) i pokrenemo je sa SBT run.

Play dolazi sa već prekonfigurisanom h2 in-memory bazom kao i integrisanim http servletom tako da je u ovom trenutku već moguće početi sa razvojem same aplikacije.

Pošto se play pokreće preko komandne linije nije nam potreban IDE, ali se svakako preporučuje radi lakšeg pisanja koda kao i remote debugga. Sbt ima komande za generisanje projekata u formatu poznatih IDE-a kao što su Eclipse i InelliJ IDEA.

H2 bazi je moguće pristupiti kucanjem h2-browser komande u okviru sbt-a.

## ANatomija play aplikacije

Tipična play aplikacija se sastoji od modela, koga čine domenski objekti, servisi sa aplikacionom logikom, repository itd, kontrolera koji zapravo predstavljaju klase sa metodama za hendlovanje http I drugih zahteva, kao I viewa-a koji mogu biti statičke web strane ali mogu sadržati I serverske templejte ili kod koji se izvršava na serveru pre slanja stranice klijentu.

Play sadrži I konfiguracione fajlove gde se specificiraju parametri kao npr konekcija sa bazom I neka osnovna podešavanja, ruting fajl gde se definišu rute kao I akcione metode (u kontrolerima) na koje ove rute pokazuju.

Ova arhitektura je drugačija od standardnih java aplikacija po tome što ne postoji main metoda kao ni glavni entry point same aplikacije, svaka ruta je zaseban entry point, a main metoda ili neka globalna klasa ne postoje, ako neki kod želimo da izvršimo po pokretanju servera ili u neko drugo schedulovano vreme to radimo korišćenjem AKKA (aktor framework) schedulera. Play koristi dependency injection za ubacivanje servisa u kontrolere i u druge servise, a bindovanje interfejsa i implementacija za injection se takodje radi u posebnom modulu aplikacije.

Pored toga moguće je definisati filtere i interceptore za svaku rutu i akciju, kao i specijalne akcije koje pridružujemo kontroler metodama koje će se izvršiti nad ulazom ili izlazom te akcije (ovo je na primer korisno za implementiranje autentikacije).

## Razvoj servisa za upis ocena u play frameworku

Aplikacija pokrivena ovim dokumentom nalazi se hostovana na githubu na sledećem javnom repozitorijumu:

<https://github.com/itehProjektni/seed-application.git>

i može se klonirati izvršenjem komande:

git clone <https://github.com/itehProjektni/seed-application.git>

Aplikacija je servis za unos I izmenu studentskih ocena od strane profesora/administratora. Omogućava registraciju, logovanje i autentikaciju korisnika kao i čuvanje prikaz i modifikovanje podataka vezanih za prijavu ispita od strane studenta.

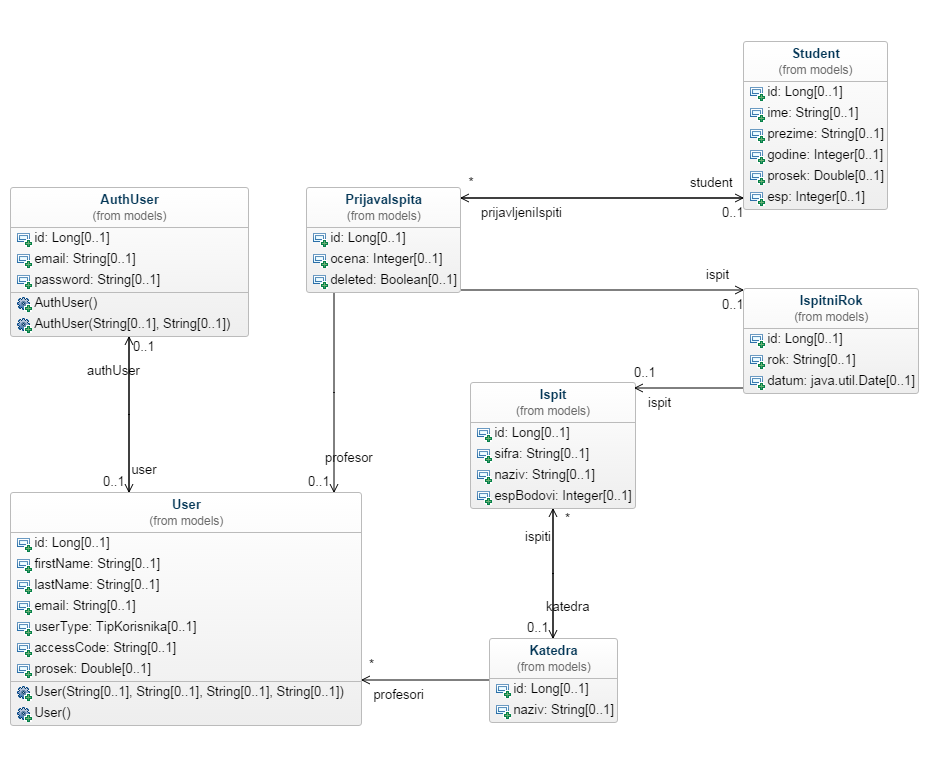
U daljem tekstu biće prikazana arhitektura aplikacije sa dodatni pojašnjenima vezanim za konkreten tehnologije koje su korišćene.

## Slucajevi Koriscenja

U konkretnom slucaju identifikovani su sledeci slucajevi koriscenja :

1. Registracija korisnika
2. Prijava korisnika
3. Pretraga prijava
4. Brisanje prijava
5. Izmena prijava
6. Pretraga prijavljenih korisnika

## Struktura softverskog sistema – uml dijagram klasa



## Domenski modeli – jpa klase

Osnovni element MVC Modela su domenske klase. Za java platforme je specifična JPA (Java persistency api) specifikacija koja definiše mapiranje domenskih klasa u realcione table.

Primer JPA Entiteta

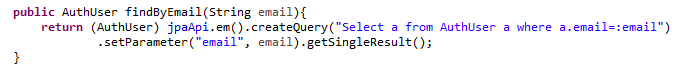


@Entity anotacija označava da je u pitanju klasa koja se persistuje u bazi, @OneToOne kao i @OneToMany i @ManyToOne i @ManyToMany su anotacije koje označavaju vrstu relacije prema propertiju iznad koga su navedene.

Pored toga bitne anotacije su @Transient za polja koja ne treba da imaju odgovarajuće kolone u bazi kao i @Table parametric tabele vezani za unikatna polja tj unique constraintove, nazive kolona I slično.

JPA je specifikacija za persistency layer aplikacije i pored mapiranja objekata u tabele relacionog modela podržava i specijalnu objektnu sintaksu za dovlačenje podataka iz baze. Najpoznatija implementacija ove specifikacije je Hibernate koji je korišćen i u ovoj aplikaciji. Jedan od načina je HQL (Hibernate querry language) upitni jezik koji za razliku od SQL-a podržava objektnu notaciju i uprošćava pisanje upita.

Primer hql upita:



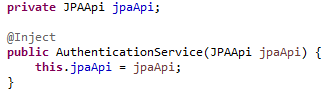
Takođe JPA obezbeđuje jednostavno snimanje i updatovanje postojećih recorda entiteta.



Pored JPA bitan feature enterprise jave koji se koristi u ovom projektu je dependency injection. Depedency injection je način za kreiranje objekata klase koje nam trebaju gde se umesto instanciranja novih objekata u samoj klasi koja treba da ih koristi, ti objekti zapravo instanciraju kao singletoni na nivou cele aplikacije a zatim se po potrebi injectuju u objekte u kojima su potrebni pri instanciranju tih objekata.

Ovo se radi jednostavnim dodavanjem @Inject anotacije koja govori aplikaciji da injectuje property naveden ispod anotacije. U slučaju injectovanja interfejsa potrebno je na nivou aplikacije definisati binding koji govori injectoru koju implementaciju tog interfejsa treba da injectuje.

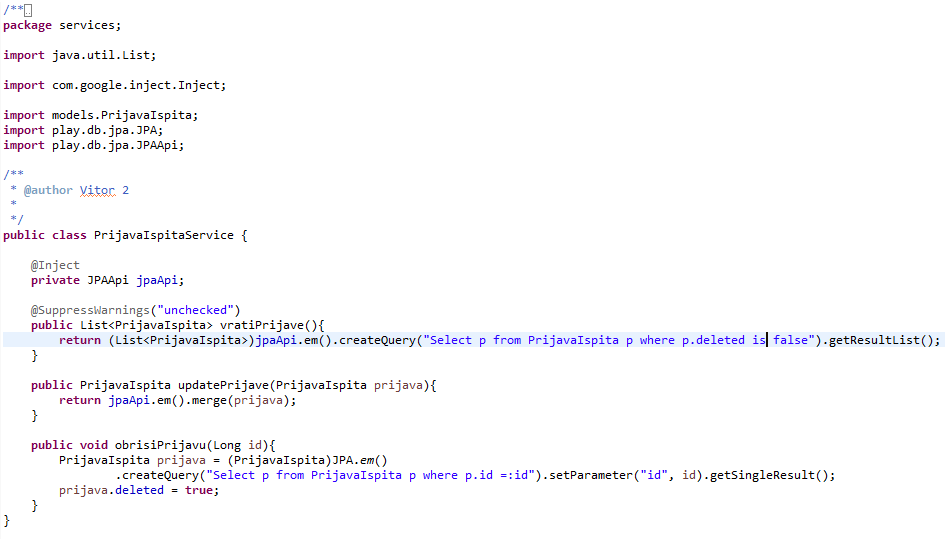
Na slici je prikazano injectovanje jpaApi servisa u AuthenticationService klasu preko konstruktora AuthenticationService-a.



## Servisne klase

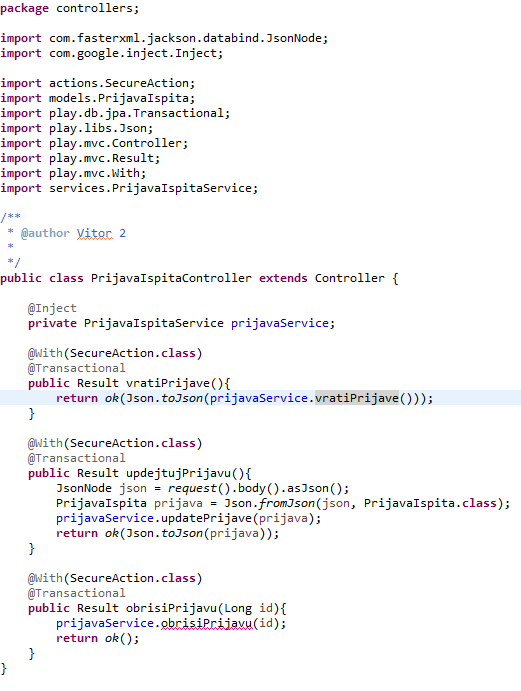
Osnovne gradivne jedinice aplikacione logike su servisne klase. One služe za pravljenje modifikaciju i manipulaciju domenskim objektima uz korišćenje određenih biznis pravila. Servisne klase dalje mogu da se injectuju u druge servisne klase ili u kontrolere i na taj način se koriste.

Primer servisne klase sa injectovanim jpaApijem i CRUD metodama.



## Kontroleri

Kontroleri predstavljaju interfejs aplikacije ka spoljnom svetu. Kontroleri se sastoje od http request handler metoda ili u play terminologiji , takozvanih Akcija. Svaka akcija tj metoda je vezana za određenu rutu kojom se njoj pristupa. Kontroleri koriste injectovanje servise da izverše određeni niz operacija, I ako je potrebno prepakuju response I pošalju ga nazad inicijatoru web zahteva u vidu http response. Pirimer kontrolera:



U routes fajlu su definisane rute koje vode ka metodama ovog kontrolera:



Svi kontroleri nasleđuju Playovu Controller klasu koja nam pruža pojednostavljeno vraćanje željenog responsa pozivanjem ok(), unauthorized(), badRequest() metoda koje setuju odgovarajući status a kao parametar primaju Json objekte ili tekst ili bilo koji content type koji je inače podržan http protokolom.

## Authentication flow – registracija, login i autentikacija

Autentikacija je komponenta aplikacije koja se skoro obavezno pojavljuje u svakom projektu. Opisaćemo osnovni flow za registraciju, login kao I dalju autentikaciju zahteva na servis korišćenjem generisanog tokena.

Metoda za registraciju prima zahtev formata:

POST /register HTTP/1.1

Host: localhost:9000

Content-Type: application/json

{

"email": "vitortomic3@gmail.com",

"password": "1234",

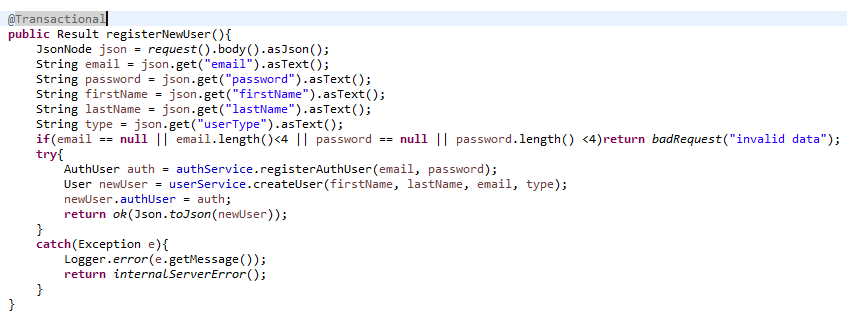
"firstName": "vitor",

"lastName": "tomic",

"userType": "ADMINSTRATOR"

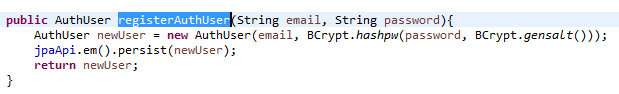
}

Handler metoda za ovu akciju je:



@Transactional anotacija na kontroler metodi označava da će kontroler upisivati i bazu i da će ceo blok kontrolera biti deo jedne database transakcije tako da će ili biti izvršene sve promene u okviru kontroler metode ili nijedna.

Podaci se čitaju iz Json tela zahteva a zatim se poziva servisna metoda za kreiranje usera.



Koristimo Bcrypt biblioteku da bi hashovali password našeg usera i sigurno ga uskladištili u bazi.

Kada se user napravi, response na registration zahtev će biti u vidu Json objekta kreiranog usera



Login ruta se poziva korišćenjem sledećeg zahteva:

POST /login HTTP/1.1

Host: localhost:9000

Content-Type: application/json

{

"email": "vitortomic3@gmail.com",

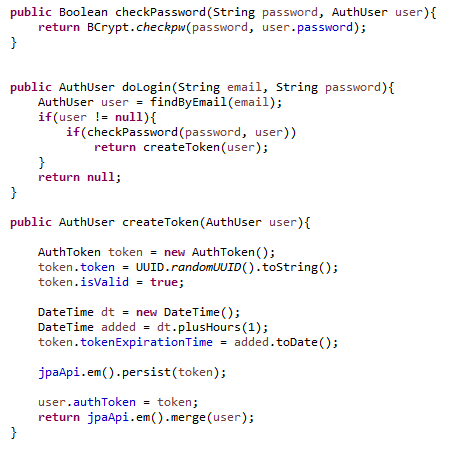
"password": "1234"

}

Kontroler metoda za login:

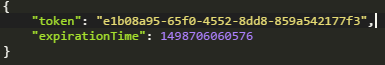


Login metoda takođe poziva autentikacioni servis da validira login podatke dobijene kroz request body.



Opet koristimo Bcrypt biblioteku, ovog puta da bi uporedili dobijenu šifru sa hešovanom vrednošću u bazi. Token generišemo kao random UUID. UUID je univerzalno unikatni identifikator. Metoda koja ga generiše će svaki put vratiti novu univerzalno unikatnu vrednost te je stoga pogodna za tokene za autorizaciju. Podešavamo i vreme isticanja tokena na jedan sat u budućnost.

Ukoliko je autentikacija uspešna, response će sadržati generisani token koji je zatim potrebno prosleđivati kroz Authorization header u svim zahtevima koji nisu deo public apija, da bi se izvršila autentikacija korisnika.



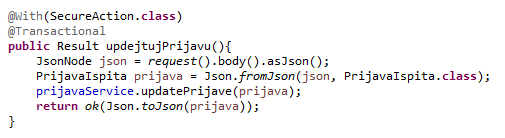
Potrebno je dodati logiku koja će omogućiti da se čita token iz authorization headera od requesta i validira da li je ispravan, u slučaju da jeste nastavlja se sa izvršavanjem requesta u slučaju da nije, vraa se unauthorized response.



SecureAction klasa poziva autentikacioni servis da validira dobijeni token.



Funkcionalnost SecureAction klase možemo jednostavno pridružiti svakom kontroleru dodavanjem anotacije.



Na ovaj način jednostavno razdvajamo public api pozive od secureovanih, secureovani pozivi će imati @With(SecureAction.class) anotaciju dok public pozivi neće.

Poslednja ruta u authorization flow je logout. Ona treba da invalidira trenutni token i tako okonča user sesiju. Logout ruta treba da šalje authorization header tako da će njena hendler metoda imati SecureAction anotaciju.

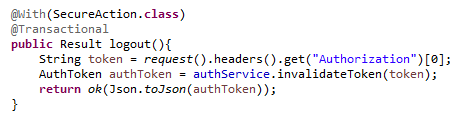
POST /logout HTTP/1.1

Host: localhost:9000

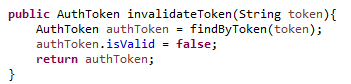
Authorization: 38235fd6-995e-410e-9652-352cc6723ad4

Cache-Control: no-cache

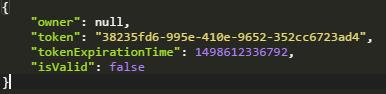
Token koji je prethodno dobijen login rutom šaljemo kroz Authorization header.



Token se dalje invalidira u autentikacionom servisu.



Kontroler metoda će kao response vratiti json objekat invalidiranog tokena

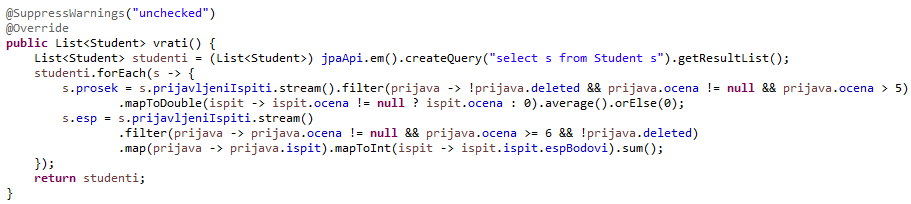


## java 8 stream api

Zanimljiv novitet u javi 8 je stream api. Streamovi su skupovi elemenata koji mogu da se popunjavaju iz postojećih kolekcija I nad kojima je moguće izvršavati funkcionalne operacije bez menjanja ulazne kolekcije.

Ove operacije mogu biti jedna ili više intermediate operacija kao što su filtracija ili mapiranje I jedna terminalna kao što su suma, prosek ili foreach iteracija.

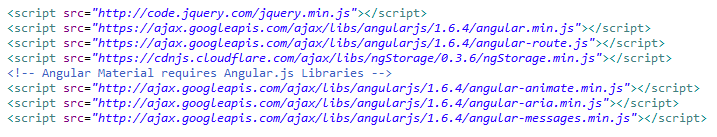
Dat je primer korišćenja stream apija u okviru ove aplikacije.



Prolazi se kroz listu studenta I za svakog studenta se dinamički računa prosek ocena I zbir esp bodova na osnovu njegovih prijavljenih ispita. Prvo se filtriraju prijave od null vrednosti I ocena nižih od 6, zatim se prijave mapiraju u listu ocena kao I listu esp bodova za položene ispite. U slučaju prosečne ocene za ovaj remapirani stream ocena računa se prosek dok u slučaju esp bodova, mapirani stream se sumira.

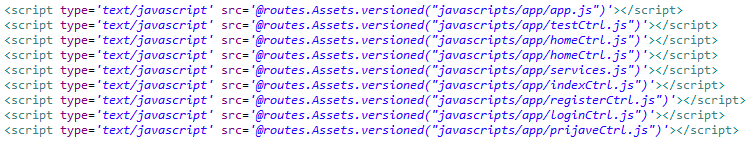
## Struktura angular aplikacije

Angular JS biblioteke se sastoje od niza javascript fajlova koje je potrebno učitati na nekoj html strain da bi se u okviru same stranice pojavila angular aplikacija.



Sama aplikacija je sadrzana u nizu js fajlova koji sadrže glavnu aplikaciju kao I servise I kontrolere vezane za nju. Ovi javascriptovi mogu da budu kombinovani u jedan fajl ali zbog preglednosti je bolje da budu podeljeni po klasama slično kao što je praksa u java programiranju.

I ove javascriptove je potrebno uključiti u glavnu stranicu.



Sama angular aplikacija će biti prikazana u okviru elementa sa ng-view oznakom



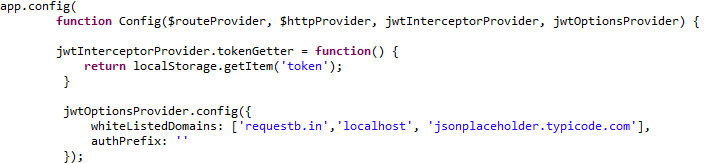
App.js predstavlja glavni element angular aplikacije. U okviru ovog fajla definišemo ime aplikacije i module koje će koristiti. Takođe se podešava konfiguracija koju će aplikacija koristiti kao i angularovo rutiranje koje služi za navigaciju unutar aplikacije.

Svaka rutra u okviru angular aplikacije ima definisan view koji treba da prikaže kao i kontroler za taj view. Dalje se u kontrolere injectuju servisi koji vrše komunikaciju sa serverom kao i izvršavanje biznis logike.

Deklarisanje aplikacije se vrši izrazom:



U navodnicima su navedena imena modula koje aplikacija koristi, npr ngRoute služi za rutiranje, ngMaterial je google implementacija material dizajna za angular dok se ngStorage koristi za pristupanje localStorage browsera preko angulara a angular-jwt je interna bibilioteka za autentikaciju.

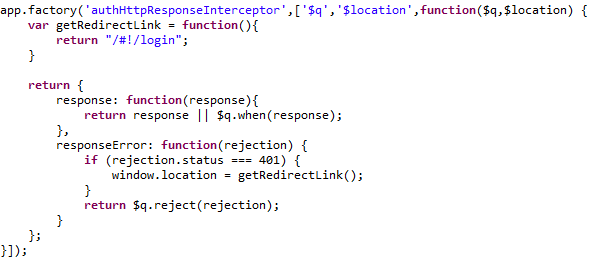


U konfiguraciji aplikacije definisali smo da se token snimljen u localstroage pri svakom pozivu dodaje u Authorization header, na ovaj način omogućavamo lak menadžment user sesije.



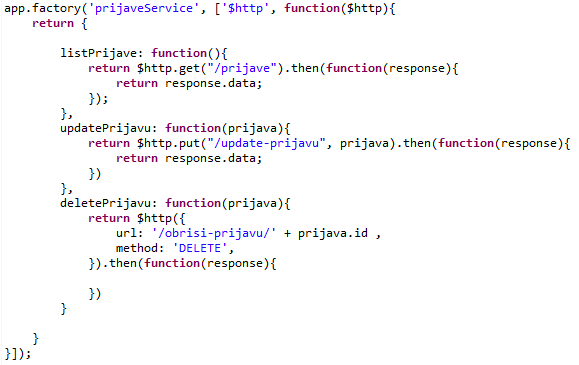
U okviru routeProvider konfiguracije definišemo angular rute kao i njihove view-e i kontroler.

httpProvider konfiguracija služi za pokretanje jwtInterceptora za dodavanje tokena u authorization header svake rute kao i authHttpResponseInterceptor koji će pri dobijanju statusa 401 unauthorized pri nekom requestu, uraditi redirect na login stranu da bi se ponovo ulogovali.



authHttResponseInterceptor predstavlja primer angular servisa koji dalje može da se injectuje po potrebi. Servis registrujemo tako sto pozivamo factory metodu nad osnovnom aplikacijom i prosleđujemo joj ime servisa kao i konstruktor funkciju servisa koja kao parametre može da prima druge servise koji će se pri njenoj inicijalizaciji njoj i proslediti.

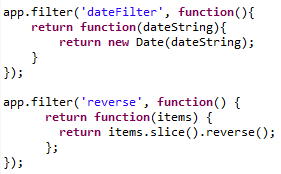
Servisi u okviru return bloka treba da vrate public metode dok će metode van bloka ostati private, ovo je primer javascript closurea.



prijaveService je primer servisa koji koristi angularovu $http biblioteku za asinhrone pozive. Ova bibliteka vraća promise nad kojima kasnije možemo da definižemo then callback funkcije koje će se izvršiti kad se promise resolvuje tj kada se dobije response na http zahtev u ovom slučaju.

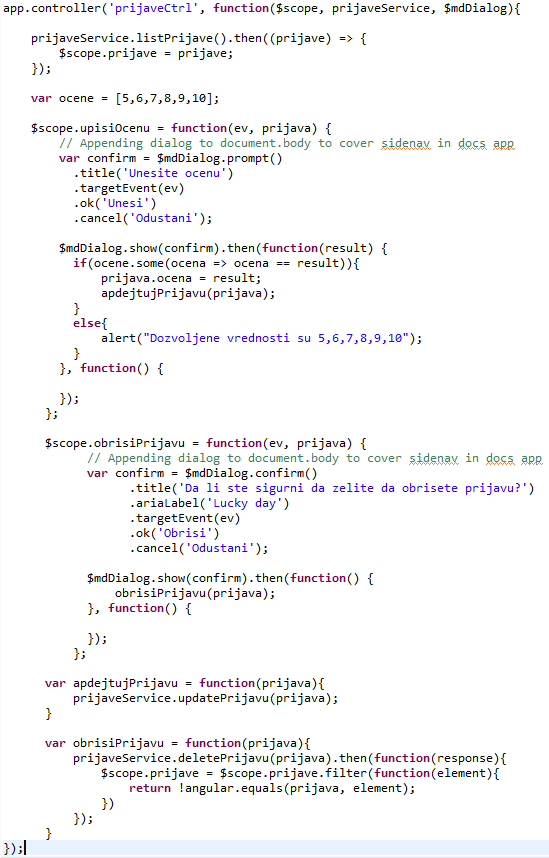


Angular takođe podržava filter funkcije koje su globalno vidljive svim angular templateima i služe za modifkaciju prikaza, a mogu se injectovati kao servisi i u kontrolere i koristiti kao metode.



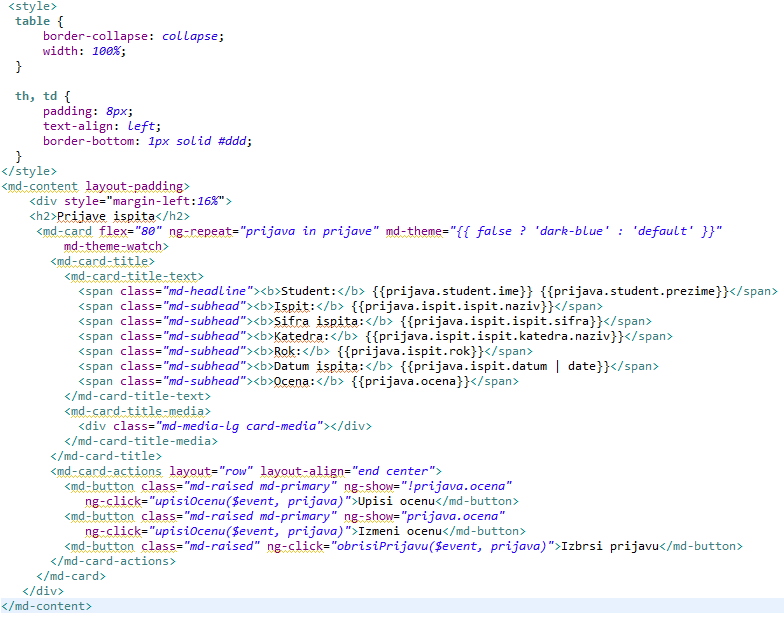
Prikazani filteri služe za prikazivanje string datuma kao javascript datuma kao i obrtanje niza.

Svaki angular view ima odgovarajući kontroler, između kontrolera i samog viewa postoji dvosmerni data binding što znači da se svaka promena u templateu automatski odražava na stanje kontrolera i obrnuto. Objekat koji sadrži promenljive kontrolera i templejta se naziva scope.



U okviru kontrolera definišemo koji se servisi injectuju u njega pri inicijalizaciji. Kod u okviru kontroler funkcije se izvržava pri prikazivanju viewa koji je vezan za kontroler.

Template gore navedeno kontrolera je dat na sledećoj slici.



Dirketive <md-content> I slično predstavljau komponente iz angular material UI biblioteke koja je korišćena pri izradi ove aplikacije. Ove komponente olakšavaju i ubrzavaju izradu interfejsa koji su u skladu sa Google material design jezikom.

## Authentication flow iz angular perspektive

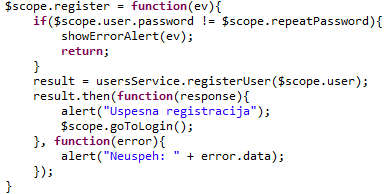
U java bekend delu dokumenta objašnjen je process autentikacije iz perspective servera. U narednoj sekciji isti flow će biti prikazan iz perspective front end aplikacije.

Registracija:

Korisnik popunjava formu za registraciju



Podaci iz forme se dalje pomoču kontrolera prosleđuju servisu za autentikaciju.



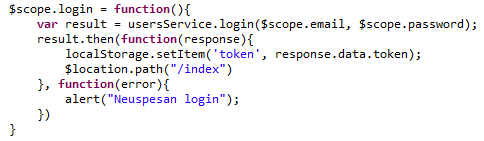


Pozivanjem ovog servisa vrši se registracija usera na bekendu.

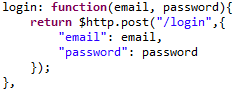
Login:

Kroz login formu korisnik unosi svoj podatke za login:





Login kontroler metoda salje servisni poziv za login, i u slucaju uspesnog logina snima token iz responsea u local storage.



jwtInterceptor će pri svakom pozivu upisvati snimljeni token iz local storage u Authorization header.



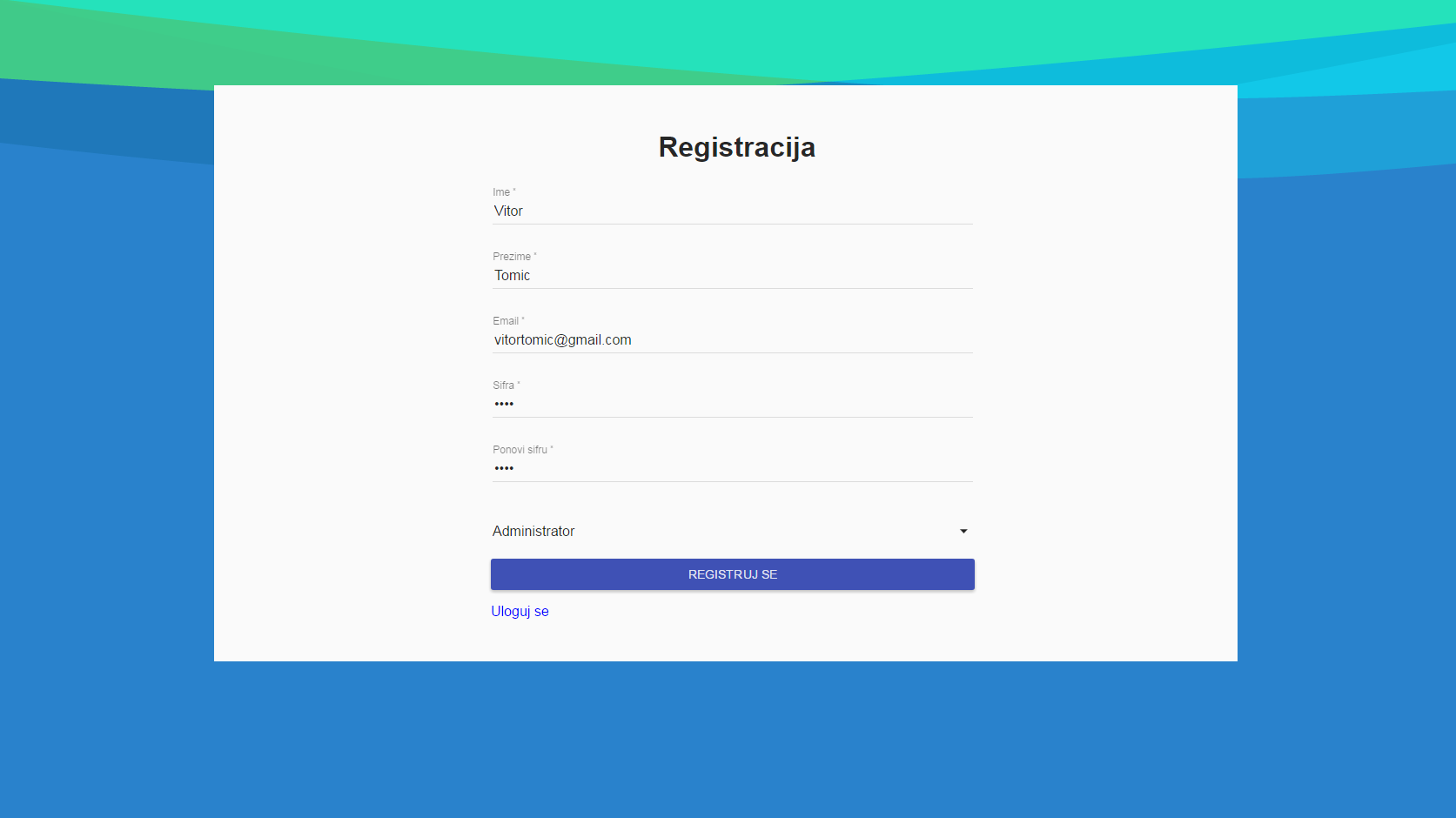
Logout:



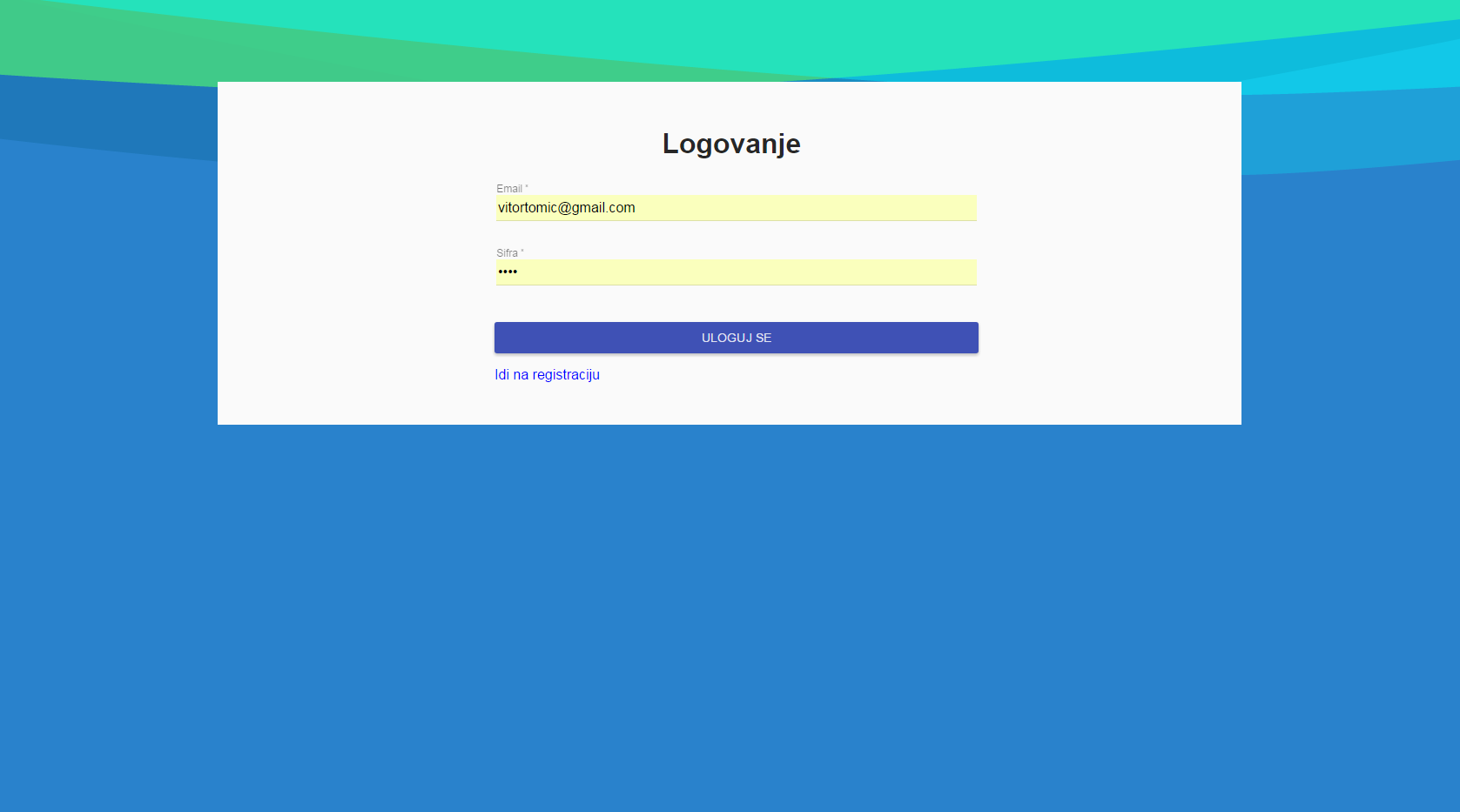
Logout ne šalje parametre osim tokena koji će se automatski dodati u header pomoću jwtInterceptorProvidera.

## application main flow

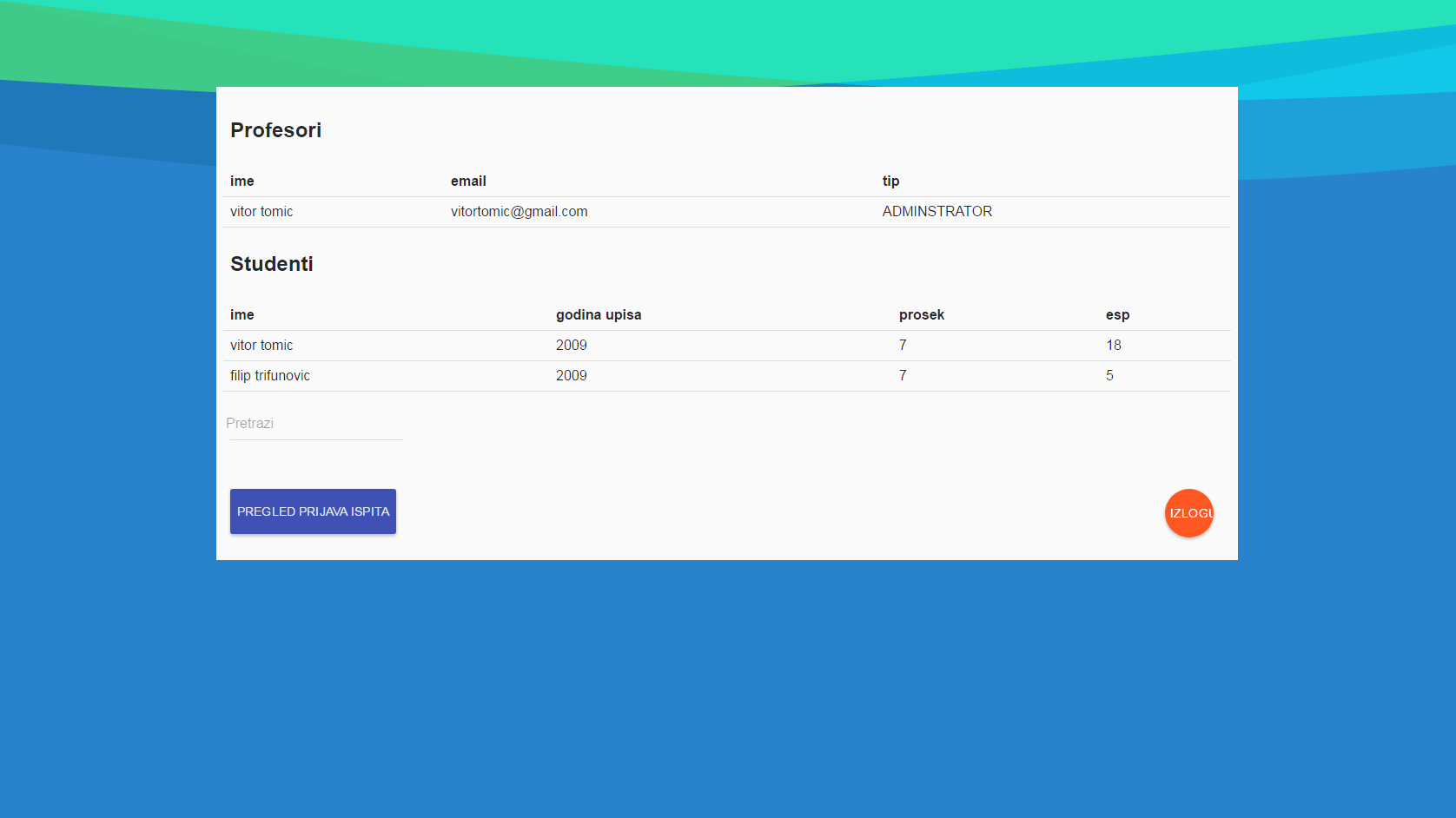
Registracija usera



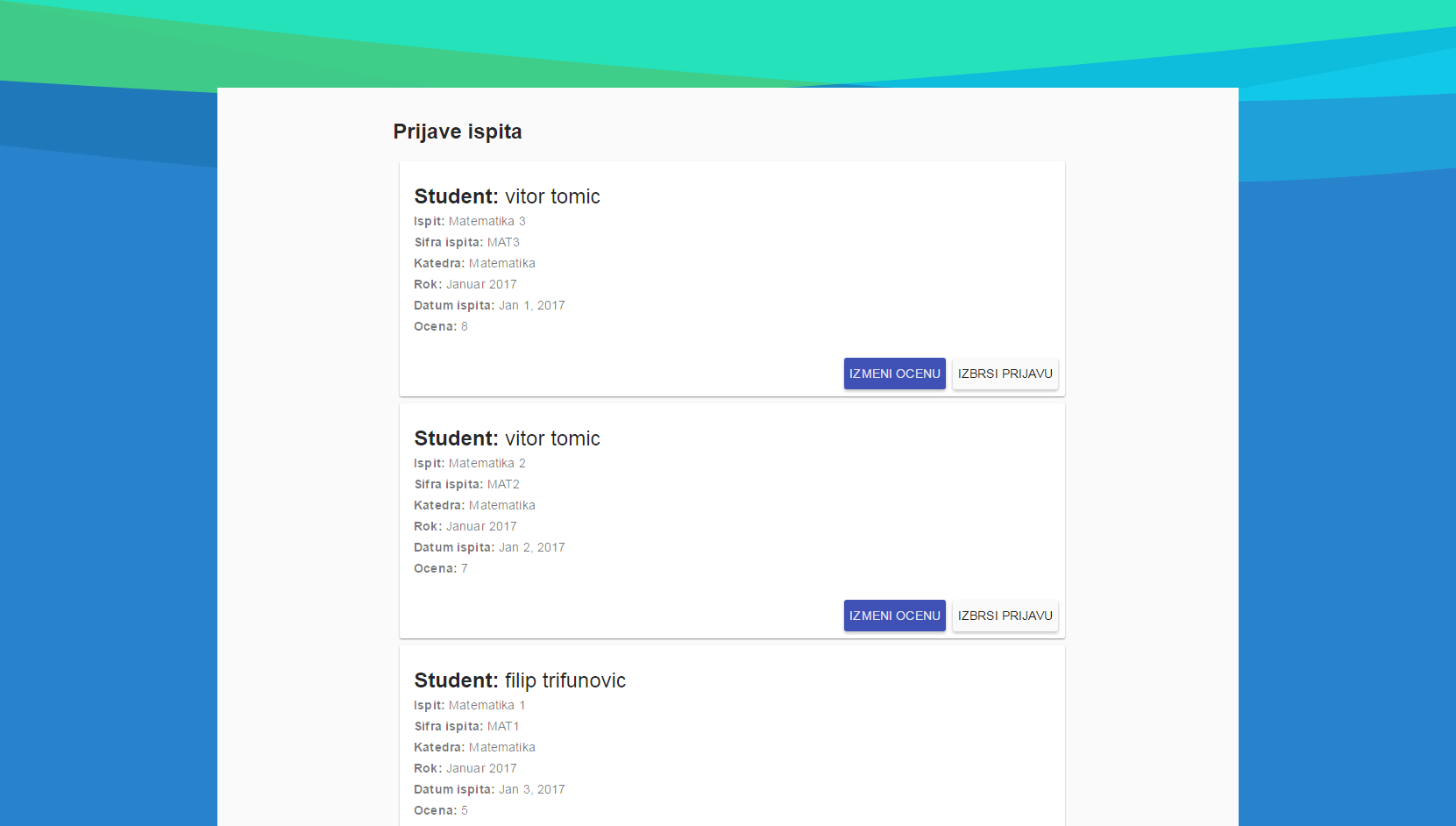
Logovanje:



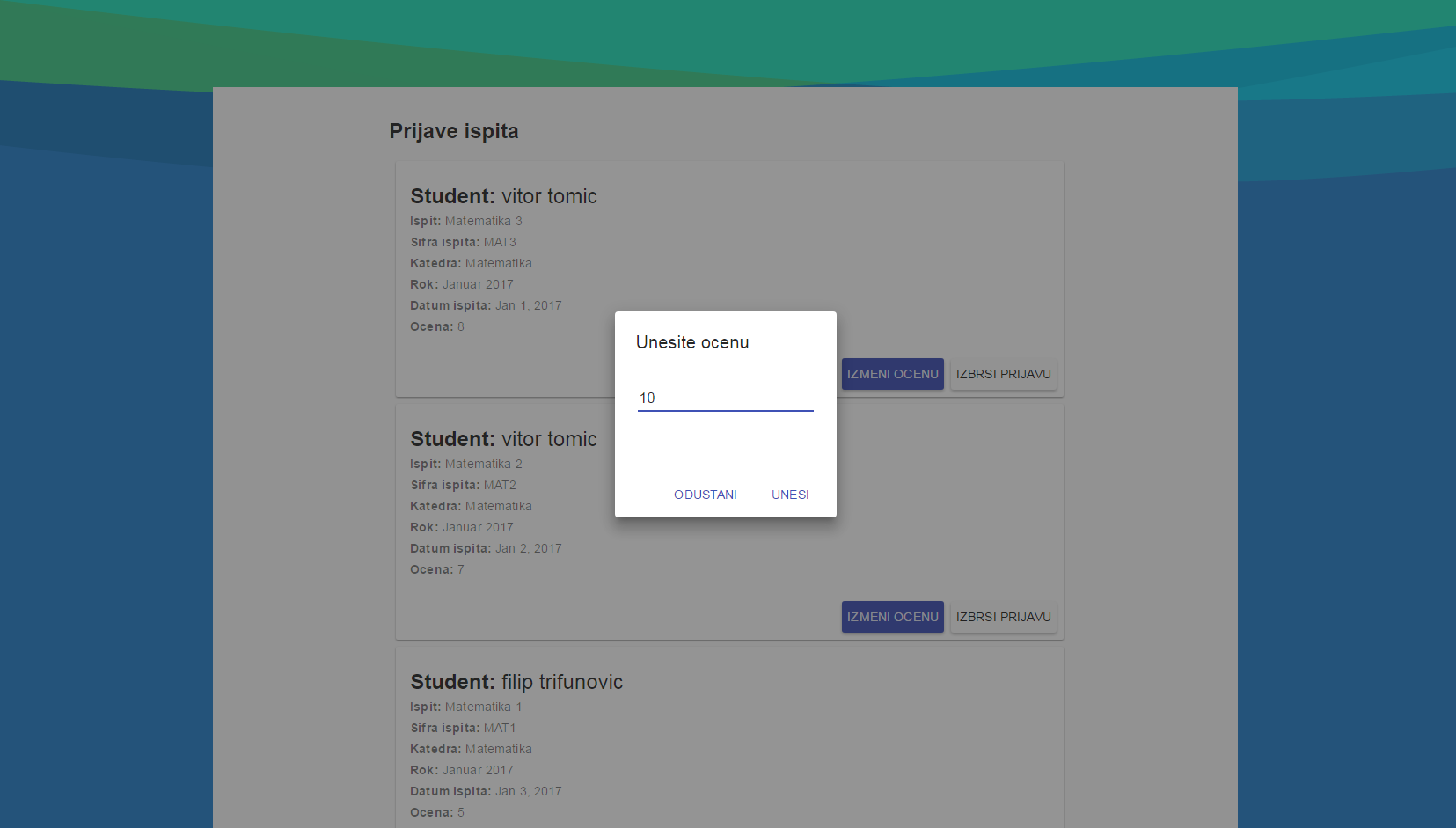
Prikaz podataka o studentima:



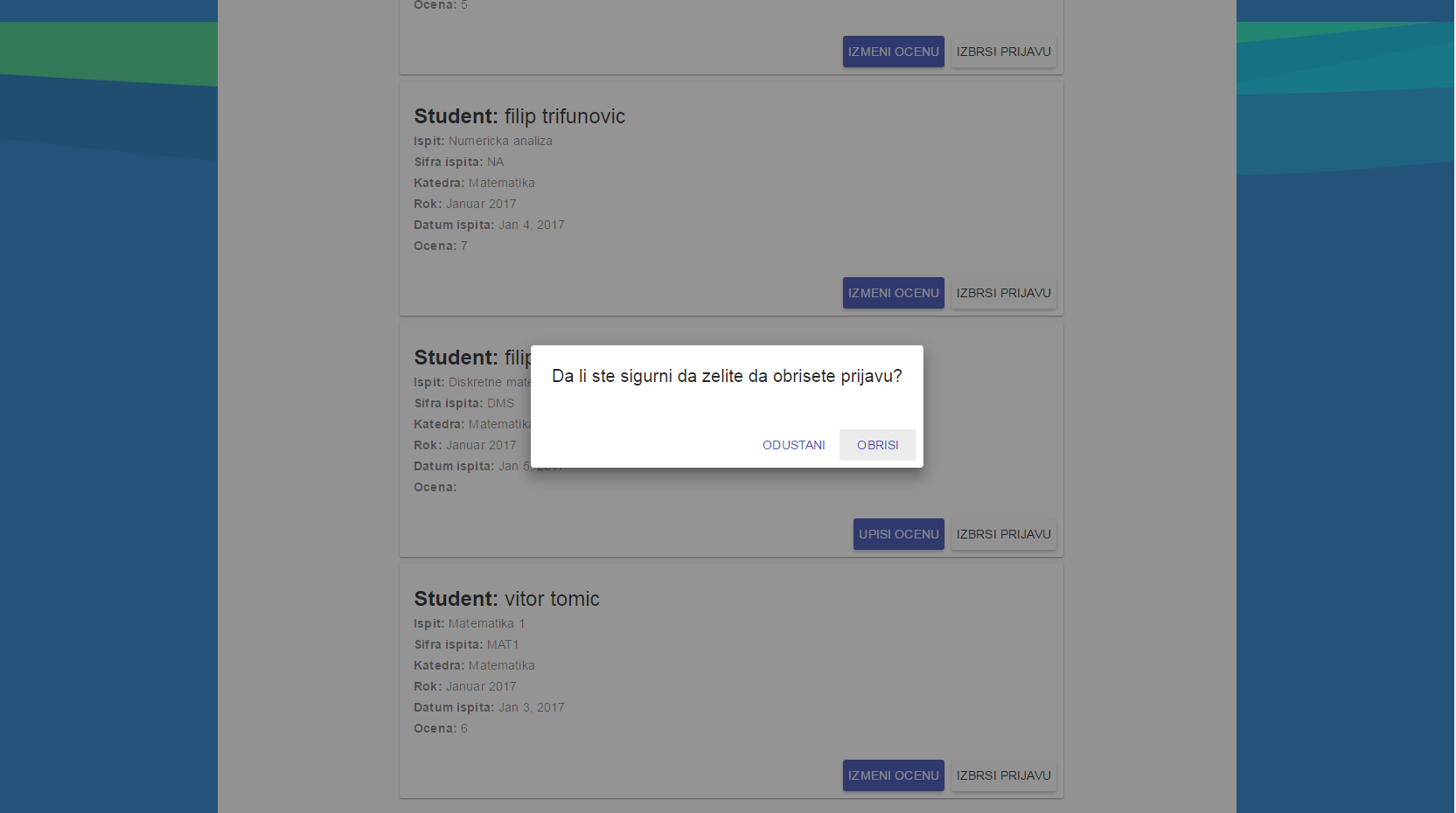
Pregled prijava ispita:



Unos/Izmena ocene:



Brisanje prijave ispita:



Podaci o studentima izmenjeni usled upisa novih ocena:

