****

**Univerzitet Singidunum**

**Tehnički fakultet**

**Razvoj aplikacije pomoću REST servisa**

**- Projektna dokumentacija -**

Predmet: **Internet i softverske arhitekture**

|  |  |  |
| --- | --- | --- |
| **Profesor**: | **Radila**: | |
| Nebojša Bačanin Džakula  **Asistent**:  Uroš Arnaut | | 1. Anđela Jovanović 2017200827 SII |
|  |
|  |

Beograd, 2020. godine

# Uvod

Kroz ovu dokumentaciju dat je kratak opis aplikacije razvijene za ispitni zadatak iz predmeta Internet i softverske arhitekture, kao i tehnologija i platforme na kojoj je razvijena. Kao inspiracija za realizaciju ove aplikacije, poslužila je platforma Roll20, koja omogućava korisnicima virtuelno igranje i organizovanje društvenih igara, poput Dungeons and Dragons-a. Nalik njoj, ovaj projekat predstavlja jedan mali community, gde korisnici nakon uspešnog logovanja ili registracije, mogu da pregledaju sve pre-made partije na kojima su možda učestvovali, ili da naprave svoje custom partije i likove. Aplikacija ima prilično jednostavan i intuitivan interfejs, te je tako pogodna za sve tipove korisnika, koji žele da na virtuelni način organizuju svoje druženje. U nastavku sledi opis tehnologija i platforme koja je korišćenja za realizaciju ovog projekta.

# ****Opis platforme, tehnologija i kratak istorijat****

Celokupna funkcionalnost aplikacije se bazira na klasičnom MVC modelu, sa tom razlikom da je znatno proširena razvojem REST web servisa, koji obrađuju pozadinsku logiku i služe za konekciju sa bazom. Osnovna ideja jeste da se aplikacija razloži na više manjih komponenata, gde će svaka komponenta obavljati osnovne CRUD operacije nad odgovarajućom tabelom u bazi, tako da se praktično osnovni model proširuje idejama mikroservisne arhitekture.

Mikroservisi se koriste u posebnoj tehnici razvoja softvera, koja predstavlja varijantu SOA stila, koji struktuira aplikaciju kao kolekciju labavo povezanih servisa. U mikroservisnoj arhitekturi, servisi su još više granularni, a protokoli koji se koriste prilično jednostavni. Ovime se poboljšava modularnost aplikacije, ona je lakša za razumevanje, razvoj, testiranje I otpornija na greške koje mogu nastati. REST je skraćenica od REpresentional State Transfer I predstavlja skup ograničenja koja, kada se primene na dizajn sistema, predstavljaju još jedan softverski arhitektonski stil. Neka od ograničenja koja su karakteristična za REST:

* Klijent-server: interakcije između klijenta I servera, koji su odvojeni jednoznačnim interfejsom. Zaduženja klijenta I servera su skroz odvojena, pa se mogu nezavisno razvijati I menjati, dok se interfejs između njih ne menja.
* Bez stanja: Svaki klijentski zahtev ka serveru mora da sadrži sve potrebne informacije za razumevanje zahteva I ne može da iskoristi sadržaje koji već postoje na serveru
* Keširanje: podaci u okviru odgovora na klijentski zahtev implicitno ili eksplicitno se dele na one koji se mogu keširati ili se ne mogu keširati itd.

Logika aplikacije razvijena je u okviru Spring radnog okvira. On omogućava programerima da razvijaju enterprise aplikacije korišćenjem POJO klasa. Sam Spring je organizovan kroz module, ali koristi poznate koncepte koji su uvedeni mnogo ranije. Postoji više različitih okvira u okviru iste porodice, kao što je već pomenuti Spring, koji nudi osnovnu podršku za indžektovanje zavisnosti, transakcioni menadžment, web aplikacije (Spring MVC), zatim Spring Security koji pruža podršku za zaštitu aplikacije različitim mehanizmima za autentifikaciju I autorizaciju, Groovi, Reactor I mnogi drugi. Prvu verziju Springa je napisao Rod Johnson, koji je popularizovao okvir objavljivanjem knjige Expert One-on-One J2EE Design and development u oktobru 2002. Okvir je prvi put razvijen pod licencom Apache 2.0 u junu 2003. godine.

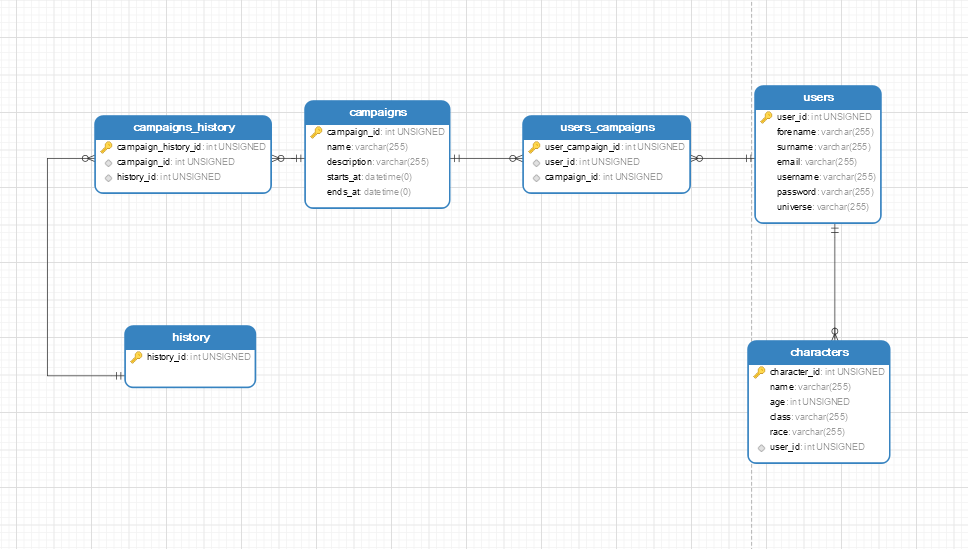
Možemo zaključiti da određene tehnologihe poput Spring-a, Hybernate-a nam olakšavaju posao prilikom pisanja programskog koda, I prinicp labave povezanosti komponenata realizuju u praksi. Međutim, opet I ove komponente se moraju na neki način eksplicitno uključutiti u projekat, kako bi naš IDE pokupio sve informacije neophodne za ispravnu konfiguraciju našeg build-a. Dakle, logično pitanje koje sledi jeste da li postoji neki način/mehanizam kojim bi se I ovo moglo donekle automatizovati? I odgovor leži upravo u Maven-u. Pre nego što se osvrnemo na sam Maven, neophodno je da se vratimo u period pre nego što je napravljen, konkretno na njegovog prethodnika, Apache Ant-a. Ovaj alat se vrlo često koristi za pravljenje Java projekata, I podrazumevan je u NetBeans I Eclipse okruženjima. Međutim, on nije pružao dovoljno “moderan pristup” u razvoju aplikacija, a takođe imao je i dosta mana, koje su developerima dodatno usporavale rad. Glavni nedostak ovog alata jeste činjenica da je potpuno proceduralan i nema podrazumevanih opcija, koji dolaze kao predefinisane sa njime. Svaki korak i svaka destinacija (classpath) se mora definisati ručno. Preciznije rečeno, za svaki projekat se mora ručno iskodirati poseban .xml fajl, koji bi sadržao sve neophodne informacije o projektu, preduzete korake, konfiguracione parametre itd. Kako je ovo preraslo kasnije u lošu praksu, Maven je odlučio da napravi moderniji i lakši pristup prilikom razvoja aplikacija, i nešto kasnije kreiran je Maven, koji se smatra naslednikom Ant-a. Maven je veoma moćan alat za upravljanje projektima koji je zasnovan na POM modelu. Koristi se za izgradnju projekata, uključuvanje zavisnosti, održavanja dokumentacije itd. Značajno pojednostavljuje postupak izrade programskog koda, kao što je Ant radio, te se zato Maven ne smatra nešto specijalno naprednijim u odnosu na svog prethodnika.

Za razvoj klijentske logike, korišćen je Angular, uz dodatak Angular Material komponenata za razvoj user-friendly korisničkog interfejsa. Iako možda ne izgleda tako staro, JavaScript postoji već skoro 20 godina i ima mnogo namena. Programeri ovaj jezik koriste za mnogo različitih veb aplikacija, jer on čini da strana klijenta, to jest strana korisnika izgleda lepo i dobro funkcioniše. Prebacivanjem sve veće logike na klijenta, zbog tendencije kreiranja grafički veoma bogatih Web aplikacija, u JavaScript kod se unosi sve veća složenost.Za redukovanje pomenute složenosti danas postoje brojni softverski okviri, koji programerima olakšavaju rad. Budući da je iz 2009. godine, AngularJS je najstariji od tri okvira. Verovatno baš zbog toga ima i najveću zajednicu korisnika. 2013. godine Angular je bio na četvrtom mestu po broju saradnika i na trećem mestu po broju zvezdica na GitHubu. AngularJS je strukturalni okvir koji omogućava razvoj dinamičkih veb stranica, po ugledu na standardne aplikacije. AngularJS omogućava pretraživaču razumevanje novih HTML konstrukcija, čime se omogućava korišćenje moćnih koncepata programiranja i softverskih šablona, kao što su povezivanje podataka, validacija i MVC.Neke od najpoznatijih kompanija koje koriste AngularJS jesu Gugl i Najki. Od avgusta 2013. godine automobili kompanije Dženeral motors koriste aplikacije koje su kreirane pomoću Angulara.

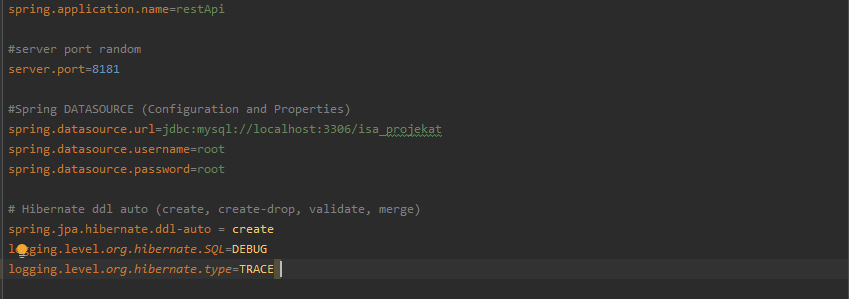
Što se samog radnog okruženja tiče, većina programera programsku logiku razvija u okviru IntelliJ-a ili Eclipse, dok se deo koji se tiče UI-ja razvija u Visual Studio Code-u. Međutim, za potrebe ovog projekta, za gore pomenute stvari korišćen je IntelliJIdea, u kome su generisani odgovarajući folder I napravljena adekvatna struktura, tako da se pozadinska logika ne meša sa fron-endom. IntelliJ IDEA je integrisano razvojno okruženje (IDE) napisano u Javi za razvoj računarskog softvera. Razvio ga je JetBrains (ranije poznat kao IntelliJ), a dostupan je kao izdanje pod licencom Apache-a 2.

# Implementacija baze i rad sa podacima

Sama aplikacija radi sa značajnom količinom podataka, te je neophodno da postoji određeni mehanizam kako bi se ti podaci negde uskladištili I po potrebi dobavili. Kao što je ranije napomenuto, sama aplikacija je koncipirana tako da pratkično svaka Angular komponenta (o čemu će biti reči u nastavku teksta), vrši osnovne CRUD operacije nad odgovarajućom tabelom u bazi. Kada se radi sa REST servisima, većina programera teži da koristi build-in H2 bazu, za koju je potrebno uključiti odgovarajući dependency. H2 predstavlja tzv. in-memory bazu, što znači da ona čuva podatke samo za vreme sesije, tj. rada u aplikaciji. Nakon što se diskonektujemo sa servera, baza se resetuje I kada sledeći put želimo da je koristimo, moramo da ponovo unosimo podatke. H2 zavisnost je pisana u Java programskom jeziku I odnosi se upravo na ovu relacionou bazu podataka, koja se može uključiti u bilo koju Java aplikaciju I da se izvršava u klijent-server modu. Za potrebe ovog projekta, korišćena je MySQL relaciona baza podataka, koja je odrađena u alatu Navicat.

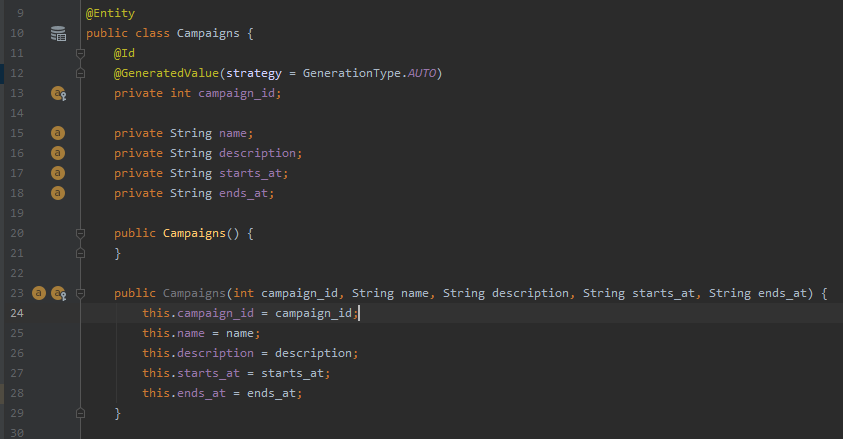


Pošto se ona podrazumevano nalazi na posebnom portu (3306), potrebno je iskoristiti dodatne mehanizme, koji će omogućiti konekciju I rad sa ovakvom bazom. Prvi od njih jeste JPA (Java Persistence API), koji predstavlja Java specifikaciju za pristupanje, čuvanje I upravljanje podacima između Java objekata/klasa I relacione baze podataka. Drugim rečima, JPA se koristi za objektno-relaciono mapiranje (Object Relational Mapping, ORM), koje omogućava da se podaci konvertuju iz objektnog tipa u relacioni tip, I obrnuto. Za potrebe ovog projekta, veza ka eksternoj bazi je uspostavljena preko odgovarajućih konfiguracionih parametara unutar application.properties fajla.

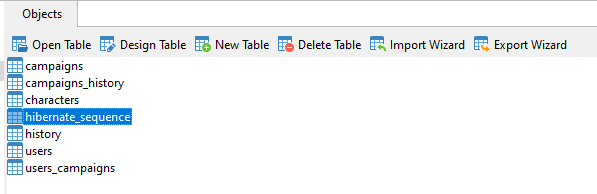


Svojstvo spring.datasource.url definiše MySQL string konekcije, svojstvom spring.datasource.username se zadaje korisničko ime MySQL baze podataka, dok se pomoću vrednosti svojstva spring.datasource.password definiše pristupna lozinka. Takođe u prikazanom kodu je navedeno da se MySQL izvršava na portu 3306. Naravno, od konkretne implementacije zavisiće I vrednosti navednih parametara.

Koristeći tehnologije poput JPA ili Hibernate-a, možemo vrlo lako integrisati našu aplikaciju sa različitim tehnologijama za rad sa bazom podataka, kao što je u ovom slučaju MySQL. Kako bi se sprečilo da aplikacija bude u tesnoj sprezi sa osnovnom tehnologijom perzistencije podataka, Spring nudi mogućnost rada sa tzv. DAO objektima, odnosno repozitorijumima. U ovom projektu, za svaku tabelu postoji odgovarajući repozitorijum, koji svoju funkcionalnost izlaže kroz interfejs. Takođe postoji I kontroler klasa, koja praktično sadrži sve metode neophodne za rad sa bazom podataka, tj. u kojima se izvršavaju osnovne CRUD operacije. Implementirana je I posebna servis klasa, koja praktično služi kao mediator/posrednik između kontrolera I repozitorijuma. Takođe svaka tabela je predstavljena u formi Java klase, anotirane @Entity anotacijom.

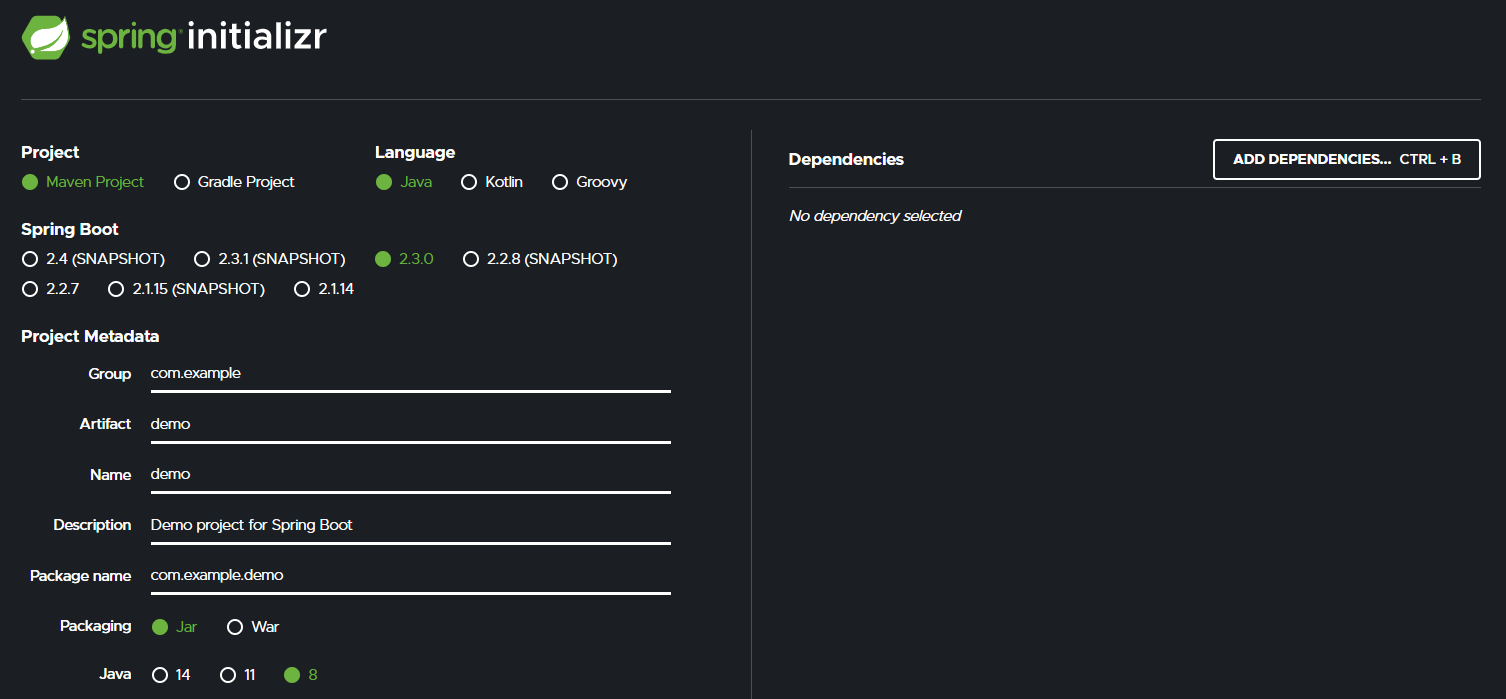


Možemo videti da preko ove anotacije, klasa je prepoznata kao tabela u bazi, a onda su na standardan način implementirana odgovarajuća polja I definisani konstruktori. Pošto je id vrednost koja bi trebalo da se automatski generiše I inkrementira automatski, kada dodajemo nove vrednosti, ovo polje je anotirano preko @Id anotacije, uz dodatak @GeneratedValue. Preko ovog metapodatka, polje se prepoznaje kao id vrednost unutar tabele, i njegova vrednost se inkrementira automatski, zahvaljujući strategy parametru. U suštini u pozadini se kreira dodatna tabela hibernate\_sequence, na osnovu koje se pamti sekvenca id-jeva koji se inkrementiraju.

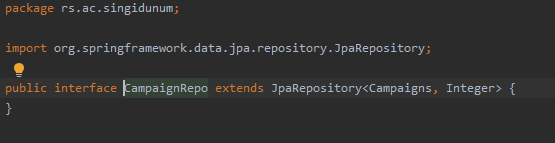


# Struktura projekta i logika

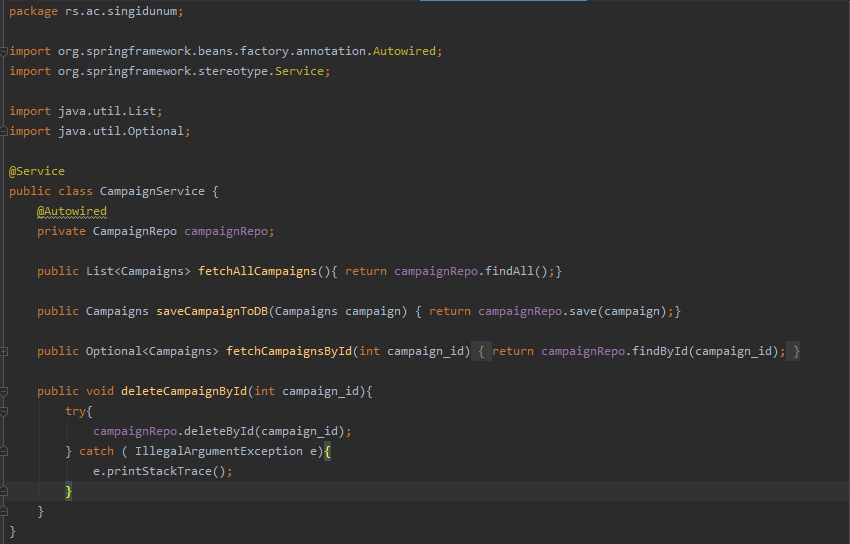
Početna struktura projekta je pripreljena uz pomoć alata ng generate za generisanje Angular komponenata na frontu, I Spring Initilazr-a za automatsko dodavanje početnih zavisnosti i parametara za pozadinsku logiku.



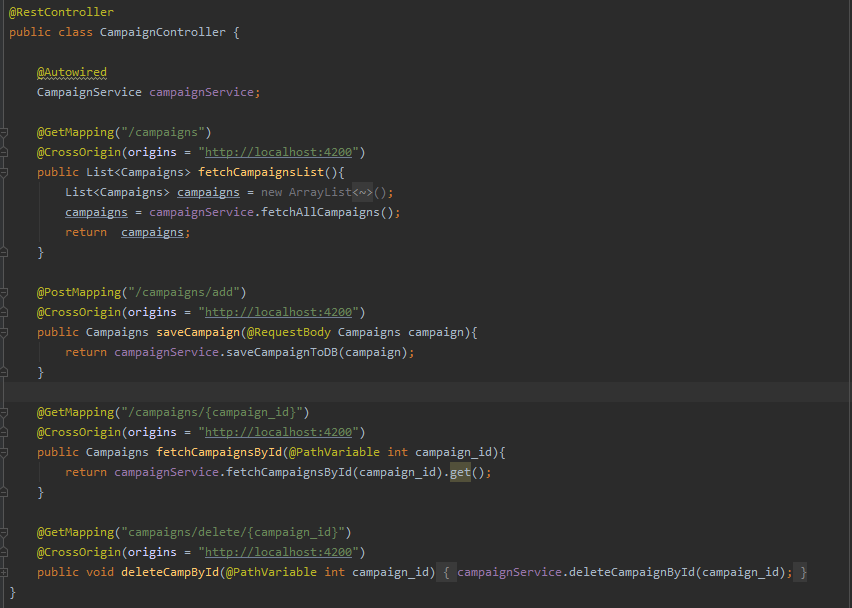
Što se tiče backend dela aplikacije, kao što je napomenuto, korišćene su tehnologije poput REST servisa, MySQL-a, Hibernate-a itd. U prethodnom delu je dat kratak prikaz osnovne implementacije baze I otvaranje konekcije ka njoj. Nakon što su podešeni svi neophodni parametri, napravljene su Java klase anotirane @Entity anotacijom, kako bi bilo jasno da one referenciraju na neki način odgovarajuće tabele u bazi. Sledeći korak jeste pravljenje kontrolera, koji će obraditi poslovnu logiku i spreminiti rezultate, kako bi view (Angular) komponente mogle da ih prikažu korisniku. Osnovna ideja jeste da se koristi mehanizam repozitorijuma, pri čemu se prvo kreira interfejs koji nasleđuje klasu JpaRepository, koja obezbeđuje osnovne funkcionalnosti za rad sa bazom podataka (CRUD).



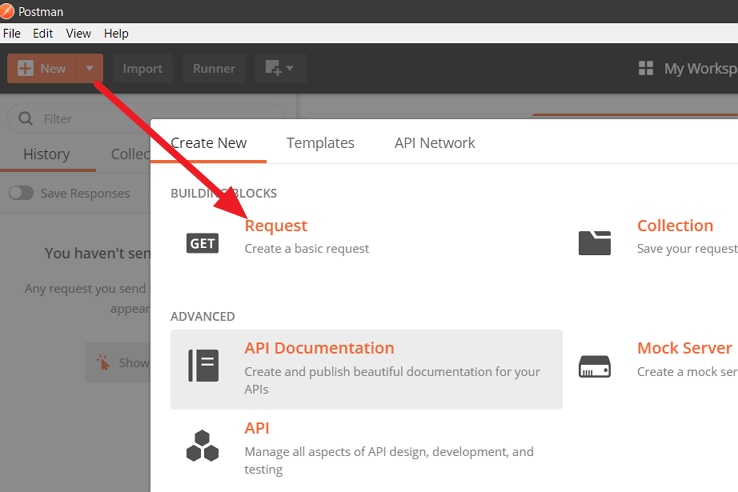
Nakon toga, napravljena je klasa koja će igrati ulogu servisa, koji će biti veza između repozitorijuma, tj. praktično JpaRepository klase I kontrolera, koji treba da obradi CRUD logiku uz pomoć njenih metoda.



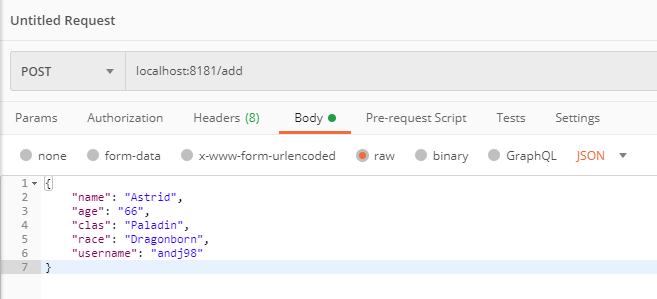
Zatim je potrebno sve ove metode mapirati na odgovarajući URL preko kontrolera, koji predstavlja jedan REST kontroler. Kao što je navedeno u minimalnim zahtevima, u projektu su iskorišćeni GET I POST metodi.



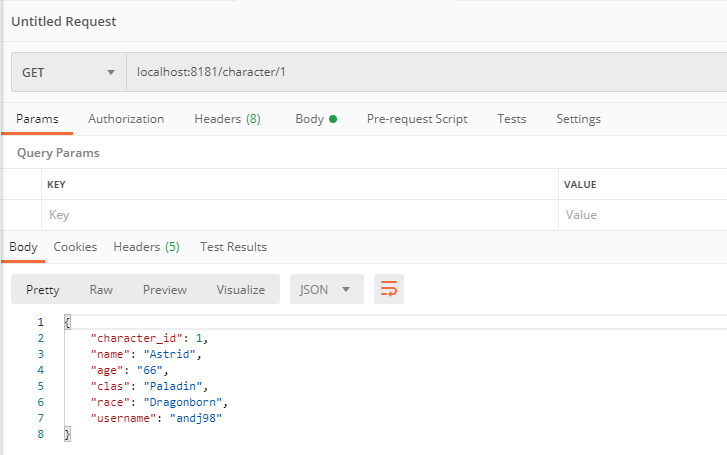
Za potrebe testiranja ovih metoda, korišćena je aplikacija Postman, koja je vrlo korisna prilikom rada sa različitim API-jima ili kada prosto želimo da proverimo da li je naš request zapravo prošao.



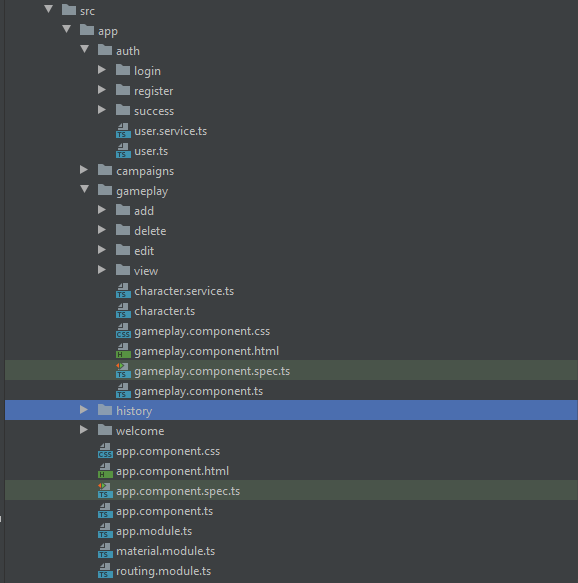
Korišćenje Postman aplikacije je prilično jednostavno I intuitivno. Potrebno je napraviti novi request, odabrati željeni zahtev koji je potrebno testirati, I vrlo se lako može videti da li je uspešno prošao POST zahtev ili recimo da li je rezultat stigao I u kom formatu, ukoliko testiramo GET zahtev. U prilogu je primer testiranih POST I GET zahteva, za rad sa tabelom Characters, gde je potrebno preko POST-a dodati novog lika u bazu, a preko GET-a pročitati informacije o željenom liku.



Potrebno je dakle odabrati željeni metod, a zatim uneti odgovarajući URL na koji je mapirana metoda iz konstroler klase. Isti princip važi I za testiranje GET metoda, samo je poziv drugačiji:

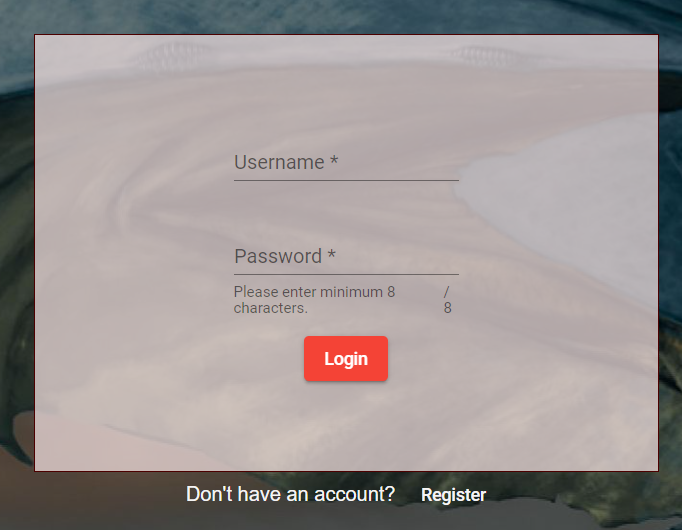


Za rad sa frontend delom, kao što je ranije napomenuto, korišćen je Angular radni okvir. Izuzetno je fleksibilan I omogućava laku integraciju sa različitim backend tehnologijama. Kao što se na prethodnim slikama može videti, pored anotacije za označavanje Rest kontrolera, korišćen je još jedan dodatni indicator a to je @CrossOrigin, gde se kao njegov parametar navodi URL adresa i broj porta na kome se izvršava klijentska aplikacija. U ovom slučaju to je dakle localhost/4200. Ovo je neophodan indikator kako bismo mogli da povežemo klijentsku aplikaciju sa rest servisima koji se podrazumevano izvšravaju na 8181 portu, što se može videti na slici.

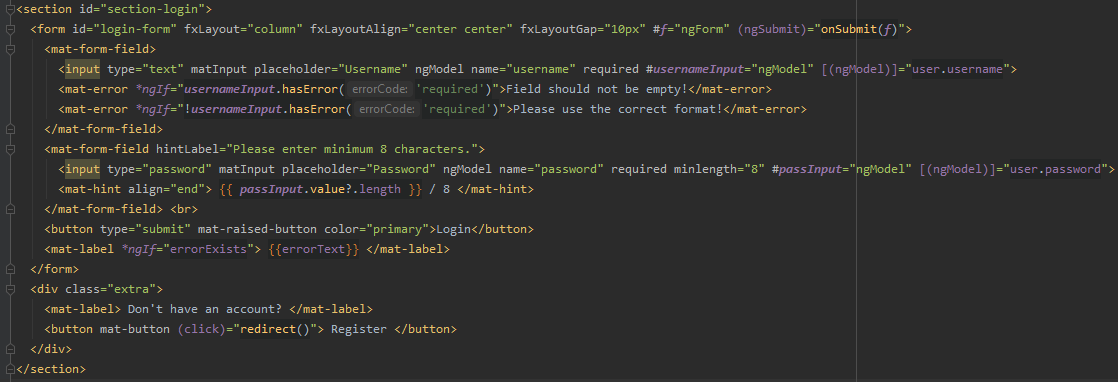


Za generisanje Angular komponenata korišćen je Angular CLI; gde se svaka komponenta generiše preko ng generate naredbe. Angular zatim sam preuzima sve neophodne module i kao rezultat dobijamo strukturu kao što je prikazano na slici. Svaka komponenta sadrži svoj template koji predstavlja mešavinu html tagova i koda, gde tagovi imaju deklarativno značenje – šta prikazati, a kod ima imperativno značenje – kako kreirati prikaz. Na ovaj način je omogućeno dinamičko kreiranje sadržaja, što je ujedno jedna od glavnih prednosti Angulara. Pored toga, moguće je takođe delimično izvšravati pozadinsku logiku specifičnu za određenu komponentu u sklopu njene interne implementacije. Sve ovo omogućava typescript fajl koji se generiše zajedno sa html i css fajlovima koji služe za dizajn i prikaz rezultata neke obrade.

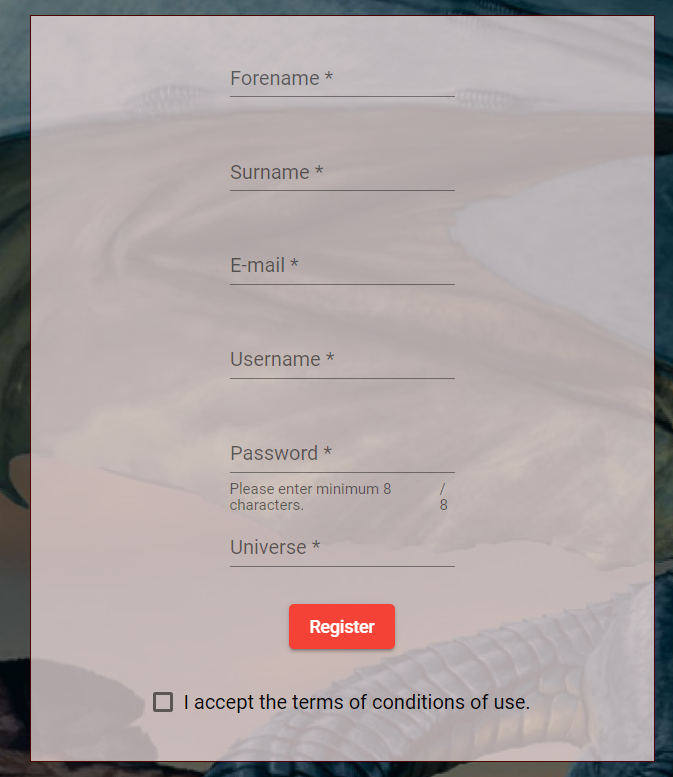
Kada govorimo I pravilima pristupa I privilegijama, kako bi sama implementacija bila jednostavnija, aplikacija radi sa jednom vrstom korisnika, a to su registrovani korisnici, što znači je najpre potrebno da se oni uspešno uloguju Ili registruju.



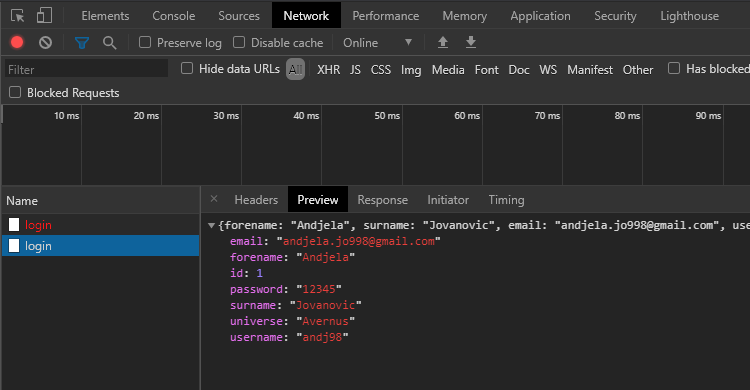
Za rad sa svim formama unutar ove aplikacije, korišćene su Angularove reaktivne forme, koje na samom frontu mogu izvršiti klasičnu validaciju, što je još jedna pogodna stvar jer se značajno uprošćava pozadinska logika u typescript fajlovima.



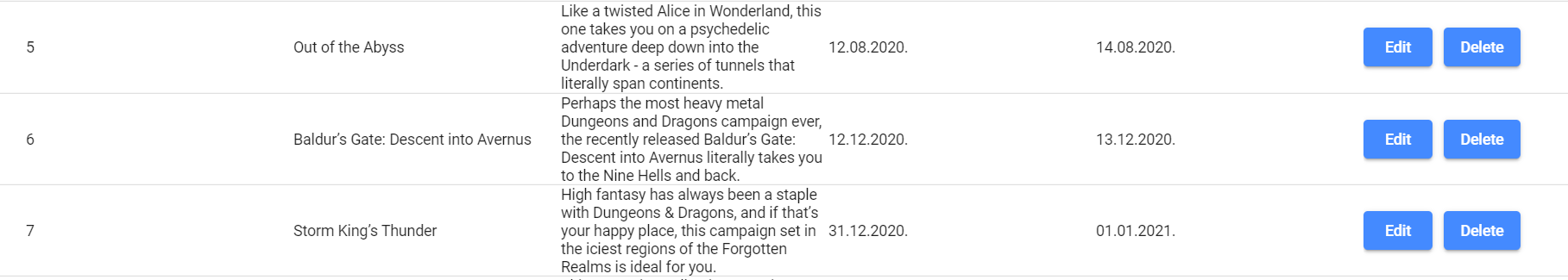
Svaki korisnik mora uneti ispravne kredincijale kako bi se uspešno redirektovao na početnu stranciju. Autentifikacija se vrši pomoću baze, gde se dopremaju podaci kao što su username i password kako bi se proverilo da li uneti parametri ispravni i da li korisnik uopšte postoji u bazi. Ukoliko ne, može se registrovati tako što klikne na dugme Register, gde popunjava tražene podatke, koji moraju biti uneti u ispravnom formatu.



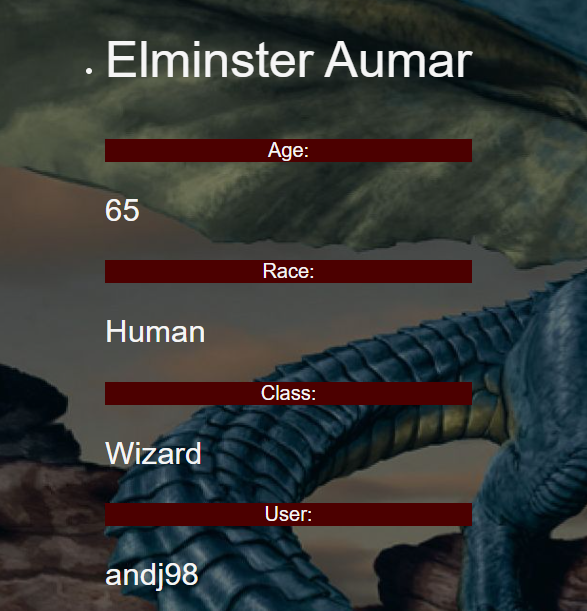
Na slici ispod se može videti izvršena provera registracije jednog korisnika korišćenjem Googl-ovih developer toolsa, gde se u Network sekciji jasno mogu videti informacije o uspostavljenoj konekciji i podacima koji se prenose u JSON formatu.



Nakon ovoga, korisnik se redirektuje na početnu stranu gde može da pravi svoje custom kampanje (interna implementacija prikazana na prethodnim slikama) ili svoje likove. Podaci su mahom predstavljeni tabelarno koristeći Angular Material komponente, konrektno mat-table komponetu, kako bi bilo omogućeno lakše filtriranje, pretraga i paginacija sadržaja.



Korisnicima je takođe data mogućnost da edituju ili obrišu svoje kampanje ili likove, ili da čak vide određene detelje custom made karaktera. Detalji likova su zapravo renderovani JSON podaci, dopremljeni preko GET metoda i prikazani kao na slici ispod.



# Zaključak

Ovaj projekat je rađen na slobodnu temu iz predmeta Internet i softverske arhitekture, tako da je isključivo namenjen za edukativnu upotrebu. Sama aplikacija je napravljena tako da pokrije sve minimalne zahteve koji su navedeni u specifikaciji, ali ima i dodatne funkcionalnosti, koje su obrađene na predavanjima, kako bi i samim korisnicima bilo zanimljivo korišćenje aplikacije, kao što je samom autoru bilo zanimljivo implementirajući je.

