IZRADA APLIKACIJE ZA ODRŽAVANJE RAKETARSKOG NATJECANJA

Jukić, Mario

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

MARIO JUKIĆ

Z A V R Š N I R A D

IZRADA APLIKACIJE ZA ODRŽAVANJE RAKETARSKOG NATJECANJA

Split, rujan 2019

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Programiranje na internetu

Z A V R Š N I R A D

Kandidat: Mario Jukić

Naslov rada: Izrada aplikacije za održavanje raketarskog natjecanja

Mentor: Marina Rodić, predavač

Split, rujan 2019

SADRŽAJ

Sažetak 1
Summary 2
1. Uvod 3
2. Tehnologije 4
2.1 AngularJS 4
2.2 MongoDB 6
2.2.1 MLab 8
2.3 Node.js 8
2.3.1 Express.js 10
3 Praktični dio 12
3.1 Zahtjevi 12
3.2 Implementacija 13
3.2.1 Dijagram stanja 13
3.2.2 Autentifikacija 14
3.2.3 Spajanje na bazu i poslužitelj 20
3.2.4 Putanje 21
3.2.5 Kreiranje modela u MongoDB 23
3.2.6 Korisničko sučelje 25
4 Zaključak 38
5 Literatura 39

Sažetak

Naziv ovog završnog rada je „Izrada aplikacije za održavanje raketarskog natjecanja“. Stranica čija se izrada opisuje u ovom radu namijenjena je održavanju raketarskog natjecanja u padobranu i traci koji održava Hrvatski astronautički i raketarski savez. Zadatak rada sastoji se od osmišljavanja same arhitekture i realizacije internet stranice koja omogućuje održavanje natjecanja u kojemu će glavni sudac imati mogućnost kreiranja natjecanja, natjecatelja i sudaca. Unutar tih opcija glavni sudac ima popis svih kreiranih natjecatelja, natjecanja i sudaca.

Tehnologija koja je primarno korištena u izradi poslužiteljskog dijela aplikacije je Node.js, uz korištenje AngularJS za izradu klijentskog dijela aplikacije. Za pohranu podataka je korišten sustav za upravljanjem bazom podataka MongoDB.

Summary

ROCKET CONTEST WEB PAGE

The name of this final thesis is „**Rocket Contest web page“.** The website, this work is about, is made for rocket competition that is being held by Croatian astronautic and rocketship alliance.

The goal of this work contains of design and realisation of the web application architecture that helps the main judge to create competition, competitors and judges. Also main judge have options to review and edit all created competitors, competitions and judge profiles.

Technology that is primarily used in the website server site is Node.js, with some usage of AngularJS for user interface. For persistence data storage Mongo database is used.

1. Uvod

U ovom radu biti će detaljno opisana izrada internet stranice za održavanje raketarskog natjecanja u padobranu i traci. Stranica omogućuje glavnom sudcu (administratoru) prijavu na upravljačku ploču (engl. *administrator panel*) gdje ima više opcija za kreiranje natjecanja. Prva opcija je kreiranje natjecatelja gdje se unosi ime i prezime pojedinog natjecatelja. Također tu se nalazi i popis svih kreiranih natjecatelja. Druga opcija je kreiranje sudca gdje se unosi ime, prezime, klub sudca koji zastupa te lozinku s kojom će se registrirati na stranicu gdje ima popis natjecatelja i štopericu za mjerenje vremena rezultata. Tu se nalazi i popis svih kreiranih sudaca ili sutkinja. Treća opcija je kreiranje tima gdje se u formu unosi ime tima, ime igrača i ime sudca u kako bi se tim kreirao. Svaki tim ima maksimalno tri igrača. Četvrta opcija je kreiranje natjecanja gdje se unosi ime, početak, lokaciju, kratki opis natjecanja te ime glavnog sudca ili sutkinje. Također u aplikaciji nalazi se i popis svih kreiranih natjecanja. Peta opcija je prijava tima na pojedino natjecanje. Tu se unosi ime tima te pretražuje popis timova koji su prethodno kreirani te se označuju timovi. Sljedeći korak je prijava odabranih timova na prethodno kreirano natjecanje. Zadnja opcija je započinjanje natjecanja. Na toj stranici nalazi se popis natjecanja gdje se odabire jedno iz liste. Nakon toga klikom na dugme `Više o natjecanju` dobiva se popis timova s imenima natjecatelja i imenom sudca za svaki tim posebno. Zatim, odabiru se natjecatelji za pojedinog sudca uz napomenu da sudac ne može suditi svom timu. Prije početka natjecanja potrebno je svim natjecateljima dodijeliti sudce. Sudac prije natjecanja dobiva svoju lozinku za prijavu. Prilikom prijave sudac ima opciju odabrati dvije kategorije. Nakon odabira kategorije sudac iz liste natjecatelja odabire jednog natjecatelja te klikom na dugme `Start` započinje mjerenje rezultata te ima sliku štoperice. Kada sudac završi sa mjerenjem rezultata pritisne dugme `Stop` kako bi završio mjerenje vremena (engl. *time tracking*) i unio automatski rezultat natjecatelja u bazu. Ukoliko se raketa ne otvori ili eksplodira u zraku sudac ima dugme `Nula` gdje će odabranom natjecatelju dodijeliti rezultat nula. Svi sudionici natjecanja imaju na početnoj stranici prikaz rezultata koji ide u živo te sve informacije o natjecanju.

Završni rad podijeljen je na dva dijela, teorijski i praktični. U teorijskom dijelu rada navedene su tehnologije korištene pri izradi aplikacije, opisane njihove karakteristike, prednosti i određene specifičnosti tehnologija. Drugi dio rada je praktični, u kojem je detaljno razrađen opis kôda čiji tekst prati i slikovni prikaz.

2. Tehnologije

2.1 AngularJS

AngularJS, napisan je u programskom jeziku JavaScript, te 2010. godine postaje platforma i okvir otvorenog kôda (engl. *open source*), kojeg je razvila kompanija Google. Googleov zaposlenik Miško Hevery razvio je projekt koji je zamišljen kao radni okvir (engl. *framework*) za izradu jedne stranice korisničke strane (engl. *Single page application*) projekta. Ti projekti su naposljetku postali poznati kao AngularJS. Zbog velikog uspjeha i dalje se ažurira i prati trendove razvoja na tržištu.

Značajke AngularJS je koncept MVC (engl. *Model-View-Controller*), vezivanje modela podataka, pisanje manje kôda i spreman radni okvir za testiranje, nazvan Karma.

MVC je dizajn uzorak korišten pri izradi svih modernih internet aplikacija. Upravljač (engl. *Controller*) se brine za primanje svih zahtjeva za aplikaciju, te zatim radi s modelom kako bi napravio spremnim sve podatke za pogled (engl. *View*). Primljene podatke, pregled tada koristi kako bi kreirao završnu prezentaciju krajnjem korisniku. Preglednik prilikom svih akcija podatke se prosljeđuje modelu.

Prednosti AngularJS-a je dvosmjerno vezivanje što bi značilo da se automatski vrši sinkronizacija podataka i sloja pogleda. Korištenje AngularJS je poprilično jednostavno. Dovoljno je preuzeti sa službene stranice zip datoteku te u `index.html` unutar `<head>` oznake uključiti skriptu `angular.min.js` ili kopirati putanju.

```
<div ng-app="myapp">
  <h1 ng-controller="helloWorldCtrl">
    {{message}}
  </h1>
</div>
<script src="https://code.angularjs.org/1.4.0/angular.js"> </script>
<script type="test/javascript">
  angular.module('myapp', ['$scope'] ).controller( 'helloWorldCtrl',
    function($scope) {
    $scope.message= "Hello World"
  })
</script>
```

Ispis 1: *Primjer AngularJS koda*

U ispisu 1., prikazuje se atribut *ng-app* koji označava da se ova aplikacija smatra AngulaJS aplikacijom. Ime ***myapp*** se može zamijeniti sa željenim imenom aplikacije.

Sva poslovna logika se nalazi unutra upravljača. Unutar *<h1>* oznake se pristupa upravljaču koji sadržava logiku za ispisivanje *Hello World*. U ispisu 1. može se reći da unutar oznake *<h1>* želimo pristupiti upravljaču pod imenom *helloWorldCtrl*.

Prva *<script>* oznaka poziva angular.js skriptu kako bi se koristio AngularJS. Dok se u drugoj *<script>* oznaci definira modul *myapp* i upravljač *helloWorldCtrl*. Upravljač sadrži jednu funkciju unutar koje se definira varijabla *massage* kojoj se pridjeljuje vrijednost *HelloWorld*. Objekt *$scope* je posebna vrsta globalnog objekta unutar AngularJS-a. Koristi se komunikaciju između upravljača i pregleda.

$http je usluga koja služi za čitanje podataka i upućivanje zahtjeva s udaljenim poslužiteljem te vraća odgovor.

Postoji više metoda prečaca *$http* usluga kao što su *.put( ), .post( ), .delate( ), .get( ), .head(), .jsonp( )*. Neke od njih korištene su prilikom kreiranja navedenog projekta.

```
$http.get('api/create/findCompetitions').then(function(response){
      $scope.findCompetitions = response.data
})
```

Ispis 2: *Primjer $http.get() metode prečaca*

U ispisu 2. korištena je *get* metoda kako bi se od poslužitelja dobili rezultati koji se nalaze na *backend/api/create* te se ulazi u putanju *findeCompetitions*. On nadalje radi upit (engl. *queries*), a upit poziva model te na taj način dolazi do podataka spremljenih u bazu.

window je usluga koja upućuje na objekt preglednika. U jeziku JavaScript *window* je globalno dostupan pa može stvarati eventualne probleme. AngularJS se poziva na njega putem *$window* usluge, koja ga na taj način može pregaziti (engl. *overridden*) i ukloniti. Preporuka je ovu uslugu upotrijebiti umjesto izravnog globalnog objekta prozora.

```
$window.location.reload()
```

Ispis 3: *Primjer korištenja $window usluge*

U ispisu 3. korišten je `$window` kako bi se stranica osvježila.

2.2 MongoDB

NoSQL (engl. *Not Only Structured Query Language*), je kao što sama riječ kaže ne samo klasična SQL baza. Alternativa je tradicionalnim relacijskim bazama zato što ne zahtjeva predefiniranu shemu podataka i distribuirana je čime omogućuje unos velike količine podataka, dok na drugu stranu ne podržava relacije kao tradicionalne SQL baze. Od tuda i naziv NoSQL. MongoDB je sustav za upravljanje bazama podataka otvorenog kôda te vodeće upravljanje baze podataka NoSQL napisana u C++, Go, JavaScript i Pythonu. Bazira se na više platformi te osigurava jednostavnu skalabilnost, visoke performanse i dostupnost. Radi na konceptu dokumenata i sakupljanja.

Dokument je postavljen kao ključ-vrijednost para koji funkcionira u dinamičkoj shemi, što znači da u istoj zbirci ne može postojati skup polja ili strukturu. Zajednička polja dokumenata zbirke mogu sadržavati različitu vrstu podataka. U MongoDB-u ne postoji koncept odnosa.

Prednosti MongoDB-a nad RDBMS-om (engl. *Relational Database Management System*) su manje shema, struktura pojedinačnog objekta je jasnija. Spojevi nisu složeni, jednostavnije je skaliranje, pretvaranje ili mapiranje objekata u objekte baze nije potrebno i pristup podatcima radi korištenja interne memorije za spajanje je brže i učinkovitije.

Prednosti korištenja MongoDB je spremanje podataka u notaciji JavaScript objekata (engl. *JavaScript Object Notation*, JSON). Ima veliku dostupnost i replikaciju, bogat je upitima, brže se ažurira, izoštrava se automatski i može se indeksirati na bilo koji atribut. U ispisu 4., vidi se kako izgleda JSON format na primjeru objekta u objektu i niza u objektu.

```
{
   "firstName" : "Mario" ,
   "lastName" : "Jukic" ,
   "age" : 27 ,
   "address" : {
      "streetAddress" : "Street 1" ,
      "country" :  "Hrvatska " ,
      "city" :  "Zadar"
   },
```

```
   "children" : []
}
```

Ispis 4: *Primjer JSON formata*

Jedne od značajki MongoDB su te što svaka baza sadrži zbirku podatka koje zauzvrat sadrže dokumente. Dokument se razlikuje s različitim brojem polja. Sadržaj i veličina dokumenta mogu se međusobno razlikovati. Struktura dokumenta je u skladu kreiranja objekata i klasa. Redovi ili dokumenti ne moraju imati definiranu shemu te se kreiraju slijedno. Model podataka omogućuje lakše spremanje nizova i predstavljanje hijerarhije te drugih složenih struktura.

```
{
   "_id" : "<ObjectID>" ,
   "firstName" : "Mario" ,
   "age" : 27 ,
   "address" : {
      "addressId"
      "streetAddress" : "Street 1" ,
      "country" :  "Hrvatska " ,
      "city" :  "Zadar"
   }
}
```

Ispis 5: *Primjer MongoDB*

U ispisu 5. prikazano je polje *_id* koje MongoDB dodaje kako bi se lakše i jednostavnije identificirao dokument u kolekciji. Primjećuje se da se podatci o adresi (addressId, adresa ulice,zemlja i grad) pohranjuju kao umetnuti dokument u zbirci što čini ključnu razliku modeliranja podataka u MongoDB.

Modeliranje podataka u MongoDB ima fleksibilnu shemu za razliku od SQL baze podataka. U SQL bazi prije stavljanja podataka treba postojati deklarirana shema tablica, u MongoDB to nije slučaj što ga ujedno i čini boljim. Neke stvari koje bi bilo dobro imati na umu prije modeliranja podataka su potrebe aplikacije. Vrstu i podatke podataka koje su potrebne aplikaciji treba pripremiti na temelju toga i donijeti odluku koju strukturu podataka je potrebna. Dobro bi bilo razmisliti o upotrebi indeksa na modelu podataka kako bi poboljšali učinkovitost upita.

Instalacija MongoDB na operacijskom sustavu Windows je jednostavna. Potrebno je otići na službenu stranicu MongoDB i kliknuti na dugme *Try free*. Nakon toga odabire

se kartica *Server* te verzija, operacijski sistem i paket. Kad je odabir završio, kliknemo na opciju *Download* preuzima se datoteka sa .msi ekstenzijom. Pri završetku preuzimanja, klasičnom instalacijom, instalira se program.

2.2.1 MLab

MLab je usluga MongoDB baze podataka koja se nalazi u oblaku (engl. *cloud*). Nudi automatizirano skaliranje i pružanje MongoDB baze. Sadrži sigurnosnu kopiju i lako se oporavi u slučaju gubljenja podataka i drugih nepredvidljivih situacija. Internetski je alat za stručnu podršku i upravljanje te ima 24 satno nadgledanje. mLab je platforma idealna za fokusiranje na razvoj proizvoda, a ne na operacije.

Kako bi se povezali na mLab potrebno je postaviti mLab račun na službenoj stranici. Nakon stvaranja vlastitog računa, u polju MongoDB implementacija klikom na opciju *Create new*, dobija se lista baza za kreiranje, a za potrebe vježbe, odabrana je besplatna baza pod imenom „Sandbox“. Nakon izbora baze, odabire se opcija *Continue*. Zatim slijedi odabir regije te imenovanje baze. Kada su sve predradnje odrađene ispravno, klikom na *Submit order*, baza je kreirana. Kako bi se baza unutar napravljene baze osigurala, odabire se kartica *User* te se u formu unosi ime i šifru za administratora.

```
mongodb://<dbuser>:<dbpassword>@ds121965.mlab.com:21965/sportapp
```

Ispis 6: *Kod preko kojeg ćemo se povezati u projektu*

Kako bi se povezalo na bazu potrebna je linija kôda prikazana u ispisu 6., gdje se unose podaci administratora u polje *<dbuser>* i *<dbpassword>*.

2.3 Node.js

Node.js je poslužiteljska platforma izgrađena na engine-u preglednika Chrome radi jednostavnije izvedbe skalabilnih i bržih mrežnih aplikacija. Razvio ga je Ryan Dahl 2009. godine. Koristi ne blokirajući I/O model koji ga čini učinkovitim i laganim, tj. idealnim za aplikaciju u realnom vremenu (engl. *real-time web applications*). Važne značajke Node.js-a su asinkrono izvođenje kôda, vrlo je stabilna i ima veliku bazu paketa koji se lako implementiraju u vlastiti projekt npm (engl. *Node Package Manager*),. Cijelo aplikacijsko programsko sučelje (engl. *application programming interface*, API) sadrži biblioteke koje su asinkrone, što bi značilo da ne blokiraju. To jest , nikad se ne čeka da API vrati neke podatke.

Mehanizam događaja olakšava poslužitelju da napravi odgovor na način koji ne blokira aplikaciju i poslužitelj čini jako skalabilnim za razliku od tradicionalnih poslužitelja koji stvaraju ograničenje za obradu dobivenih zahtjeva.

Instalacija Node.js-a na operativnom sustavu Windows je jednostavna. Potrebno je preuzeti sa službene stranice .msi datoteku. Klikom na tu datoteku pokreće se instalacija. Nakon toga pokreće se čarobnjak za postavljanje Node.js-a. Prihvaćanjem licence i odabirom lokacije za pohranu instalacije dolazi se do liste komponenti. Odabirom zadane komponente i klikom na gumb *Next* dolazi se do instalacije i nakon toga Node.js je spreman za uporabu.

Kada je Node.js instaliran, potrebno je napraviti još neke stvari kako bi se projekt pokrenuo.

```
npm install
npm init
```

Ispis 7: *Iinstaliranje Node menađer paketa*

U ispisu 7., prvom naredbom instalira se npm, u datoteku u kojoj je rad na projektu zamišljen. Samim instaliranjem dobije se *node_modules* folder kako bi se moglo služiti samim paketima.

Druga naredba kreira *package.json* datoteku vidljivu u ispisu 8., gdje se unose podaci tipa ime projekta, verziju, opis i ostale stvari. Najvažniji je točan unos foldera koji će se pokrenuti prilikom naredbe *npm start* i pokretanje poslužitelja.

```
{
  "name": "sportapp",
  "version": "1.0.0",
  "description": "This is Sport App",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jukic",
  "license": "ISC",
  "dependencies": {}
}
```

Ispis 8: *Izgled package.json*

Za glavnu datoteku kod pokretanja postavljen je *server.js* u kojem su instalirani ostali paketi potrebni za rad, putanje, povezivanje na bazu i poslužitelj.

U ispisu 9., opisano je kreiranje servera i ispisivanje poruka u internet pregledniku. Ukoliko se kreira neka datoteka sa *js* ekstenzijom može se napraviti testni program

```
var http = require('http');

http.createServer(function (req,res){
  res.writeHead(200, {
   'Content-Type' : 'text /html'
  });
  res.end ("Hello world");
}).listen(8000);
```

Ispis 9: *Ispis hello world poruke preko Node.js*

Prva linija kôda je osnovna funkcionalnost funkcije *require* koja čita JavaScript datoteku, izvršava je i nastavlja s objektom. Pomoću tog povratnog objekta mogu se koristiti različite funkcije dostupne u modulu koji poziva funkcija *require*. Komanda *http* koristi funkcionalnost prijenosa podatka na internetu (engl. *Hyper Text Transfer Protocol*, HTTP). Druga linija kôda stvara poslužiteljsku aplikaciju. Funkcija se poziva svaki put kada pošaljemo zahtjev na poslužitelj. Funkcija *writeHead* se koristi za slanje podataka u zaglavlje našeg klijenta, dok će funkcija ***end*** zatvoriti vezu. Kada primimo zahtjev, funkcija će vratiti odgovor „Hello world“ od klijenta. Parametar *res* služi kao zahtjev za objektom ili parametrima primljenih sa stranice, a *req* za vraćanje odgovora na stranicu. Funkcija *server.listen* služi kako bi naša poslužiteljska aplikacija slušala zahtjev na portu „8000“. Taj port može biti bilo koji slobodan port.

Kako bi se pokrenuo kod preko Node.js-a potrebno je otvoriti direktorij gdje se nalazi datoteka te utipkati *node filename.js* te će naše računalo raditi kao server. U našem internet pregledniku upišemo adresu „http://localhost:8000“ kako bi se vidjela poruka upisana u programu.

2.3.1 Express.js

Express.js je radni okvir poslužitelja internet aplikacije Node.js, koji je dizajniran za izradu jednostavnih i hibridnih internet aplikacija. Standardni je poslužiteljski okvir za

Node.js-a koji služi kao pomoćni dio MongoDB, Express.js, AngularJS i Node.js poznatiji kao MEAN (engl. *MongoDB, Express.js, Angular, Node.js*) stog. MEAN je besplatni i otvoreni izvor JavaScript. To je softverski paket koji služi za izgradnju dinamičkih internet stranica i internet aplikacija. Olakšava razvoj aplikacije koja se koristi za obradu više zahtjeva, poput: PUT, GET, DELETE i POST.

Express.js se instalira preko NPM-a sa naredbom :

```
npm install express –save
```

Ispis 10: *Instaliranje express.js*

U ispisu 10., naredba traži pakete od upravitelja kako bi preuzela traženi ekspresni modul i na taj ih način instalira.

Uz navedeni modul navesti će se i moduli koji se koriste u aplikaciji i opisati kako se instaliraju i čemu služe.

```
var express = require('express')
var bodyParser = require('body-parser')
var morgan = require('morgan')
var mongoose = require('mongoose')
var app = express()
```

Ispis 11: *Korištenje modula u našoj aplikaciji*

Prve četiri linije kôda u ispisu 11., opisuju kako se koristiti module koje se definira u *server.js* datoteci.

bodyParser se instalira naredbom *npm install body-parser --save* te služi kao srednji softver (engl. *middleware*), node.js-a za rad s podacima obrasca koju su kodiranu u JSON,URL,Text i Raw.

morgan se instalira naredbom *npm install morgan --save* te služi za stvaranje nove funkcije srednjeg softvera kako bi odredio zapisnik koristeći format i opciju. Format je funkcija koja će biti pozvana sa tri argumenta req, res i token.

mongoose se instalira naredbom *npm install mongoose --save*, a on je MongoDB alat koji služi za modeliranje objekta. Dizajniran je za rad u asinkronom okruženju.

Posljednja linija kôda će kreirati objekt naziva *app*, ima metode za konfiguriranje srednjeg softwera, prikaz HTML pregleda, usmjeravanje HTTP zahtjeva i izmjenu postavki aplikacije koje kontroliraju ponašanje aplikacije.

3 Praktični dio

3.1 Zahtjevi

Aplikacija je kreirana na način da administrator upravlja cijelim procesom. Uređena je na način da administrator ima svoju kreiranu lozinku i korisničko ime pri pokretanju aplikacije. Sljedeći korak je kreiranje sudca za pojedini tim na način da korisničko ime bude *ime + prezime*, a lozinku unosi sam administrator. Tu su smještene i opcije uređivanje podataka i brisanje samog sudca. Administrator također ima opciju kreiranja natjecanja gdje unosi sve podatke potrebne da bi se znalo kad i gdje počinje natjecanje. Svi ti podatci će biti prikazani na početnoj stranici. Postavljena je i opcija za kreiranje natjecatelja gdje administrator unosi u formu podatke za pojedinog natjecatelja i ima listu svih kreiranih natjecatelja. Tu se nalaze i opcije za brisanje i uređivanje natjecatelja. Opcija za kreiranje tima omogućuje pretraživanje igrača po imenu te označavanje pojedinog igrača. Svaki tim mora imati minimalno jednog igrača, a maksimalno tri te svaki tim mora imati jednog sudca. Opcija prijave tima na natjecanje nudi tražilicu timova koji su prethodno kreirani i listu svih natjecanja na koje mogu biti prijavljeni. Zadnja opcija u upravljačkoj ploči je započeti natjecanje. Sudac dobiva listu natjecanja i odabirom na jedno od natjecanja dobiva popis svih timova koji su prijavljeni na to natjecanje. Sljedeći je korak generirati sudce za natjecatelje uz to da ne smije suditi svom timu. Kada je svim natjecateljima dodijeljen sudac, natjecanje može započeti. Gore navedene opcije moraju biti spremljene u bazu podataka.

Kada je kreiran, sudac mora se prvo prijaviti kako bi se došlo do stranice gdje je lista njegovih natjecatelja za suđenje. Svaki natjecatelj ima pravo na maksimalno tri suđenja po kategoriji. Sudac ima opciju odabira kategorije *padobran* i *traka*. Nakon odabira kategorije i natjecatelja sudac može započeti mjerenje vremena rezultata postoje dvije opcije. Prva je završiti mjerenje vremena nakon čega se automatski upisuju rezultati u tablicu gdje su natjecatelji i u tablicu na početnoj stranici gdje će svi natjecatelji imati uvid u rezultat. Druga opcija je *nula*, što bi značilo da ispaljivanje rakete nije uspjelo ili je nešto drugo pošlo po zlu. Prilikom tog događaja rezultat će biti postavljen na nulu.

Sudac i administrator imaju i opciju odjave gdje će automatski biti usmjereni na početnu stranicu. Nakon toga se, radi sigurnosnih razloga, ne mogu vratiti na stranicu bez ponovne prijave.

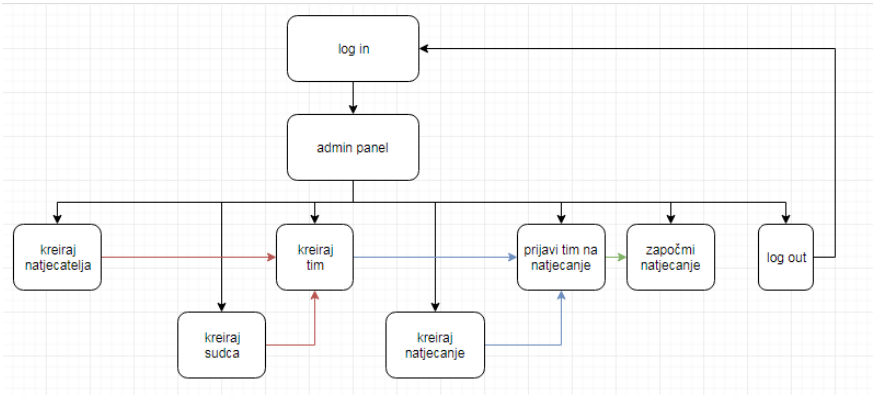
Početna stranica ima klizač (engl. *slider*) sa logotipima timova koji su na natjecanju. Rezultati će se prikazati samo ako je natjecanje aktualno. Ukoliko natjecanje nije aktualno vidljiva su nadolazeća natjecanja.

U podnožju je smještena poveznica (engl. *link*) za povijest svih natjecanja. Poveznica mora biti dostupna svima, ujedno kao i poveznica s pravilima natjecanja.

3.2 Implementacija

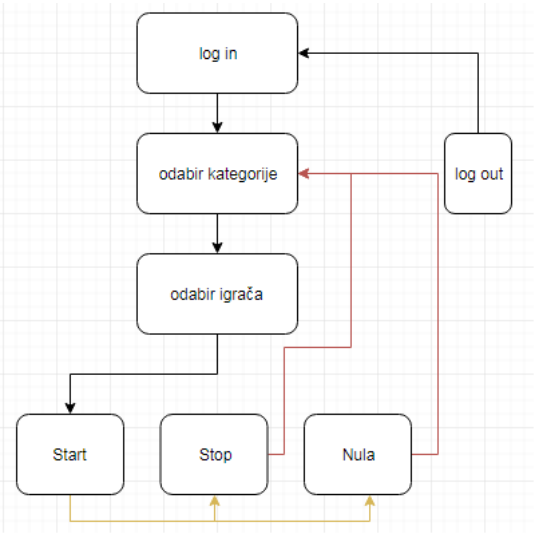
3.2.1 Dijagram stanja

Dijagram stanja (engl. *State Machine*), koristi se za opisivanje ponašanja objekta ovisnog o stanju. Objekt različito reagira na isti događaj, odnosno ovisno o stanju u kojem se nalazi. Slika 1. prikazuje stanje prilikom prijave glavnog sudca. Nakon prijave prikazuje se `admin panel`, gdje ima više stanja. Stanje `kreiraj tim` ovisi o stanjima `kreiraj natjecatelja` i `kreiraj sudca`. `Prijavi tim na natjecanje` ovisi o stanju `kreiraj tim` i `kreiraj natjecanje`. `Započni natjecanje` ovisi o stanju `prijavi tim na natjecanje`. Sva stanja u `admin panel`-u su dostupna ali ako nisu zadovoljena ne mogu aktivirati. `Logout` stanje vraća korisnika na početak tj. na `login`.



Slika 1 : *Dijagram stanja za glavnog sudca*

Slika 2. prikazuje dijagram stanja za sudca kojem se nakon prijave prikazuje polje za odabir kategorije, igrača te opcije za start, stop i nula i odjavu. `Odabir igrača` ovisi o stanju `odabir kategorije`, a `Start` ovisi o `odabiru igrača`. `Stop` i `Nula` ovise o stanju `Start`. Stanja `Start` i `Nula` vraćaju na početak stanje `odabir kategorije`. `Logout` stanje vraća korisnika na početak tj. na stanje `login`.



Slika 2 : *Dijagram stanja za sudca natjecanja*

3.2.2 Autentifikacija

U ispisu 12., prikazano je da se klikom na *Log In* dugme poziva funkcija za prijavu u *loginController*-u.

```
<form class="loginForm">
  <div class="column">
    <p>USERNAME</p>
    <input id="username" type ="text"  ng-model="loginModel.username">
  </div>
  <div class="column">
  <p>PASSWORD</p>
  <input id="password" type ="password"  ng-model="loginModel.password" >
  </div>
  <div class="column">
    <button  type="button"  class="btn"  ng-click="login(loginModel)"  >
LOGIN</button>
  </div>
</form>
```

Ispis 12 : *Forma za login u HTML-u*

Ta funkcija poziva *login* funkciju u *authService* prikazanu u ispisu 13., koja je definirana radi lakšeg snalaženja u kôdu. Funkcija radi ***post*** zahtjev na poslužiteljsku stranu točnije na *api/login/authenticate* te šalje poslužitelju korisničko ime i lozinku koja se nalazi u *user* modelu.

```
var login = function(user) {
```

```
  return $http.post('api/login/authenticate', user).then(function(result)
{
    storeUserCredentials(result.data.token)
    return result
  })
}
```

Ispis 13: *Funkcija u authService.js*

Putanja *api/login* definirana je u *server.js* datoteci.

```
app.use('/api/login', require('./backend/api/authApi'))
```

Ispis 14: *Definiranje putanje u server.js datoteci*

Ispis 14., prikazuje datoteku na lokaciji *backend/api/authApi* koja pripada toj putanji. Sljedeći korak je ući u putanju *authenticate* koja je zapravo jedina u toj datoteci.

```
var User = require('../models/User')
User.findOne({
   username: req.body.username
}).select('_id username password roles').exec(function(err,user){
```

Ispis 15: *Pretraga u bazi User*

U bazi se traži postoji li korisnik sa korisničkim imenom koji je poslan sa klijentske strane u *user* modelu i označava se na njegov id, korisničko ime, lozinku i ulogu koja je prikazana u ispisu 15. Ako ga ne nađe vratit će se status 401 sa greškom i porukom koji je prikazan u ispisu 16.

```
res.status(401).json({
  success: false,
  msg: 'Authentication failed. User not found.'
})
```

Ispis 16: *Ispis poruke i greške*

Ukoliko je pronađen provjera se lozinka onog korisnika koji je pretraživan u bazi pod imenom *username* i ukoliko je ona jednaka *passwordu* koji je poslan sa klijentske strane kreće se na sljedeći korak koji je opisan u ispisu 17.

```
var jwt = require("jwt-simple")
if(user.password.toString() == req.body.password.toString()){
  user = user.toObject()
```

```
  delete user.password
  var token = jwt.encode(user, 'TopSecret')
  if(user.roles.toString() == 'admin'){
    res.json({ success: true, token: 'JWT ' + token , username:
user.username, isAdmin: true })
  }else{
    res.json({ success: true, token: 'JWT ' + token , username:
user.username, isAdmin: false })
  }

}else{
  res.status(401).json({ success: false, msg: 'Authentication failed.
Wrong password.'})
}
```

Ispis 17: *Provjera je li ista lozinka*

Nadalje, u kôdu se briše označena lozinka radi sigurnosti da ne bi bila dostupna na klijentskoj strani. Nakon toga radi se enkodiranje user objekta koji je dohvaćen iz baze pomoću `JWT.encode` sa ključnom riječi `TopSecret`, a `jwt` (engl. *JSON Web Token*), je funkcija paketa. Sljedeći korak je provjera je li korisnik administrator ili sudac, ako je administrator poslat će varijablu `isAdmin: true` na klijentskoj strani, a ako nije bit će `false` vrijednosti. Ako je lozinka kriva vratit će se greška 401 i poruka.

```
function storeUserCredentials(token) {
  window.localStorage.setItem(LOCAL_TOKEN_KEY, token)
  useCredentials(token)
}
```

Ispis 18: *Funkcija storeUserCredentials*

Nakon toga se poziva `authService.js` odakle je putanja i pozvana. Poziva se funkcija `storeUserCredentials` opisana u ispisu 18., koja postavlja u pregledniku lokalne pohrane taj kriptirani `JWT` token koji je došao sa poslužitelja.

Dalje se poziva funkcija `useCredentials` vidljiva u ispisu 19., koja se šalje kao argument koji ima izgleda slučajnog alfanumeričkog niza poput „dasdasdasdasd15684815“ što za ljudsko oko ne znači ništa jer je kriptiran.

```
function useCredentials(token) {
  isAuthenticated = true
  authToken = token
  var currentUser = parseJwt(token)
  userService.setUser(currentUser)
  $http.defaults.headers.common.Authorization = authToken
}
```

Ispis 19: *Funkcija useCredentials*

Postavlja se *isAuthenticated* na *true* i *authToken* dobiva token koji je primljen sa poslužitelja. Nakon toga se odvija akcija kod svih ostalih putanja koje se pozivaju kada je korisnik prijavljen. U funkciji *parseJwt* provjerava se da li je definiran token na poslužitelju i ako je onda se token dekodira. Nakon dekodiranja poziva se *userService* koji je isto tako napravljen radi lakšeg snalaženja u kôdu i moguće ga je pozvati iz svih vrsta upravljača. Funkcija *setUser* vidljiva je u ispisu 20., će postaviti varijablu *currentUser* na korisnika koji se dobio dekodiranjem.

```
angular.module('SportApp').service('userService', [ function(){
      var currentUser = null
      return {
            getUser : function(){
                  return currentUser
            },
            setUser : function(user){
                  currentUser = user
            }
      }
}])
```

Ispis 20: *Funkcije setUser i getUser*

Sljedeća linija kôda postavlja *headers.common. Authorization* koji će služiti kod svakog sljedećeg poziva na poslužitelj kao dokaz da je korisnik prijavljen.

Prilikom odjave poziva se funkcija imena *destroyUserCredentials* gdje se postavlja *authToken* na vrijednost *undefined*, što je isto kao da ga brišemo. Varijabla *isAuthenticated* postavlja se na *false* i *headers.common*. Authorization na *undefined*. Na taj način nitko više neće moći pozvati putanju sa poslužitelja bez da se

iznova prijavi. Preko *userService.setUser({})* postavlja se prijavljeni objekt *usera* na prazan objekt što je vidljivo u ispisu 21.

```
function destroyUserCredentials() {
  authToken = undefined
  isAuthenticated = false
  $http.defaults.headers.common.Authorization = undefined
  window.localStorage.removeItem(LOCAL_TOKEN_KEY)
  userService.setUser({})
}
```

Ispis 21: *Funkcije uništavanja prava*

U opisu ispisa 22., koristi se kreiranje natjecanja. Primjer je dobar jer da bi se kreiralo natjecanje potrebno je biti prijavljen kao administrator.

```
var router = express.Router()
var auth = require('../services/authServices')
var cq = require('../queries/createQueries')
router.post('/competition', auth.ensure, checkIsAdmin, function(req, res,
next){
  cq.createCompetition(req.body)
  .then(data => res.ok(data))
  .catch(err => res.error(err))
})
```

Ispis 22: *Post API metoda u create.js datoteci*

Prije ulaza u putanju postoje dvije provjere središnjeg softvera *auth.ensuer* i *checkIsAdmin*. U prvoj provjeri zahtjeva se da korisnik bude prijavljen, a u drugoj da li je i administrator odnosno da je njegova rola administrator. *var auth* pokazuju putanju gdje je definiran *authServices*, a *var cq* gdje je definiran *createQueries*. Iz *authServisa* poziva se *ensure*.

```
var passport = require('passport')
var passportAuthenticator = passport.authenticate('jwt', { session: false
})
var fakeAlwaysTrueAuthenticator = (req, res, next) => next()
var ensure = true ? passportAuthenticator : fakeAlwaysTrueAuthenticator
```

Ispis 23: *Definiranje ensure u authServices*

Nakon toga poziva se *passportAuthenticator* prikazan u ispisu 23., koji će provjeriti je li token poslan sa klijentske strane stvarno odgovara tokenu tog korisnika tj. provjerava se je li netko lažirao token.

AuthConfig se poziva prije nego se uđe u bilo koju putanju te je pozvan i definiran u *service.js* datoteci.

```
var passport = require('passport')
var passportJWT = require('passport-jwt')
var User = require('../models/User')
var JwtStrategy = require('passport-jwt').Strategy,
    ExtractJwt = require('passport-jwt').ExtractJwt
module.exports = function() {
  var opts = {}
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderWithScheme('jwt')
  opts.secretOrKey = 'TopSecret'
  passport.use(new JwtStrategy(opts, function(jwt_payload, done) {
    User.findById(jwt_payload._id, function(err, user) {
      if (err) { return done(err, false) }
        if (user) {
          return done(null, user)
        } else {
          return done(null, false)
        }
    })
  }))
}
```

Ispis 24: *AuthConfig.js servis*

U ispisu 24. radi se ponovna dekripcija tokena koji je postavljen u *autherizationHeader* i nakon toga u dekriptiranom user objektu nalazimo *ID* trenutnog prijavljenog korisnika i ukoliko je zaista pronađen u bazi, to je stvarno on i uredno je prijavljen. Sljedeća provjera provjerava je li korisnik administrator.

```
var checkIsAdmin = function(req, res, next){
  var roles = req.user.roles
  if(roles == 'admin'){next()}
  else{res.status(400).json({err: 'You dont have permission '})}
}
```

Ispis 25: *CheckIAdmin kreiran u create.js*

Drugi središnji softver provjerava je li administrator te ukoliko je nastavlja dalje sa procesom, a ako nije dobiva se status 400 i poruku greške opisan u ispisu 25.

3.2.3 Spajanje na bazu i poslužitelj

U ovom poglavlju opisano je kako se spaja da bazu podataka i poslužitelj. Prilikom pokretanja aplikacije sa naredbom *npm start*, poziva se *server.js* datoteka. U njoj se pokreće spajanje na bazu i poslužitelj.

```
var mongoose = require('mongoose')
var config = require('./backend/config/appConfig')
mongoose.connect(config.database,function(err){
  if(err){
    console.log(err)
  }else{
    console.log('Connected on database !!!')
  }
})
```

Ispis 26: *Spajanje na bazu podataka*

U ispisu 26. definiran je *mongosse* paket koji je prethodno instaliran. Nakon toga se spaja na bazu pomoću metode *mongoose.connect()*, koja prima parametar *config.database*. Config pokazuje putanju gdje je definiran varijabla *database* u ispisu 27., sa putanjom od mLab-a. Ukoliko sve prođe dobro ispisat će se poruka u terminalu gdje je pokrenuta aplikacija, ako ne ispisat će se greška.

```
module.exports = {
  'database': 'mongodb://admin:admin@ds121965.mlab.com:21965/sportapp',
  'port': process.env.PORT || 3000,
}
```

Ispis 27: *AppConfig sa putanjom mLab-a i portom*

Sljedeća korak je povezivanje na poslužitelj koji je pozvan u *server.js* datoteci.

```
var app = express()
var config = require('./backend/config/appConfig')
app.listen(config.port, function(err){
  if(err){
    console.log(err)
  }else{
    console.log('Listening on port 3000')
```

```
  }
})
```

Ispis 28: *Spajanje na server*

U ispisu 28., prikazana je funkcija *App.listen* s kojom poslužiteljska aplikacija sluša zahtjeve klijenta koja prima parametar *config.port*. Config pokazuje putanju gdje je definiran port. Port ima *process.env.PORT || 3000*, što znači da će koristiti port 3000 osim ako ne postoji.

3.2.4 Putanje

Putanje (engl. *routes*), služe za prelazak na drugu stranicu aplikacije. S obzirom da je korištena aplikacija s jednom stranicom, korišten je *ui-router* modul koji će omogućiti jednostavno stvaranje i korištenje putanja u AngularJS-u.

Da bi se mogao koristiti modul, mora se uključiti datoteka koja se nalazi unutar *vendor* foldera unutar *<script>* oznake u datoteci *index.html* vidljiv u ispisu 29.

```
<script type="text/javascript" src="vendor/angular-ui-router.min.js">
</script>
```

Ispis 29: *Skripta sa ui-router datotekom*

Kreirana je datoteka pod nazivom *aap.routes.js* vidljiva u ispisu 30., gdje su postavljena sva putanje.

```
$urlRouterProvider.otherwise("/")
$locationProvider.html5Mode(true)
$stateProvider
  .state('login', {
    url: "/",
    templateUrl: "login/login.html",
    controller: "loginController"
   })

   .state('home', {
     url: "/create",
     templateUrl: "home/home.html",
     controller: "homeController"
   })
   .state('home.kreirajTim', {
     url: '^/create/kreirajTim',
```

```
        views: {
          'adminPanel@home': {
             templateUrl: 'home/kreirajTim/kreirajTim.html',
             controller: 'kreirajTimController'
           }
        }
    })
```

Ispis 30: *Datoteka aap.routes.js sa primjerima usmjerenja*

Otherwise će sa *$urlRouterProvider* preusmjeriti sve nepostojeće adrese na početnu stranicu. Korištenjem *$locationProvider* pohranjene su veze, *$locationProvider.html5Mode(true)* služi za pronalaženje početnog *index.html* datoteke koja u sebi sadrži *<base href='/'>* i definirana je unutar *<head>* oznake. Taj dio kôda postavljen je prije unosa ostalih datoteka potrebnih za rad na klijentskoj strani.

Stanje (engl. *state*) odgovara „mjestu“ u aplikaciji u smislu navigacije i cjelokupnog korisničkog sučelja. Stanje opisuje kako izgleda korisničko sučelje. *$stateProvider* nudi sučelje za objavu tih stanja u aplikaciji. Prva putanja ima ime *login*, usklađeni lokator sadržaja (engl. *Uniform Resource Locator*, URL), vrijednosti „/“, predložak (engl. *template*) sa putanjom prezentacijskog jezika za izradu internet stranice te naziv upravljača.

State imena *home.kreirajTim* ima u URL polju vrijednost *^/create/kreorajTim*. Tu se koristi poseban simbol **„^“** pomoću kojeg se sprječava aktivacija stanja ako ne dođe do apsolutnog podudaranja s traženim URL-om. Unutar pregleda nalazi se apsolutni cilj pregleda *adminPanel* te njegovo *home* stanje. Sadržaj pregleda treba biti objekt koji ima ciljani prikaz i vrijednosti.

```
<div class="adminNav">
<a class="item"
   ui-sref-active="active"
   ui-sref="home.kreirajNatjecanje">
   Kreiraj natjecanje</a>
</div>
<div class="panel" ui-view="adminPanel">
  No panel selected.
</div>
```

Ispis 31: *Ui-view i ui-sref*

U ispisu 31., direktiva imena *ui-view* se nalazi u * * oznaci, te ima ime *adminPanel*. Koristi se za označavanje mjesta gdje će biti smješteni predlošci.

Unutar oznake *<a>* nalazi se direktiva imena *ui-sref* , koja u sebi sadrži ime stanja koje će odvesti korisnika nakon klika. Pomoću direktive *ui-sref-active* i trenutno aktivne putanje se elementu koji vodi na istu putanju pridoda se klasa (engl. *class*), te na taj način korisniku označujemo aktivan element u aplikaciji.

3.2.5 Kreiranje modela u MongoDB

U sljedećem dijelu opisano je stvaranje sheme korisničkog modela na bazi. Na početku su za korisnički model dodana samo osnovna polja kao na primjer *username*, *password* i *firstName*, a kasnije je model nadograđivan sa svim ostalim poljima. Korisničku shemu koristimo u tri svrhe: administrator, sudac i igrač.

Administratora određuje polje *roles*. Ukoliko je ono definirano kao *admin* takav korisnik će imati pravo kreiranja natjecanje, sudca, tima i sve ostale administratorske funkcije.

Sudac je definiran sa ulogom *user* gdje ima polje funkcija (engl. *function*) i to isto polje postavili smo na *referee*. Kada ima funkciju i rolu (*"function": ["referee"],"roles": ["user"]*), moći će se prijaviti kao sudac u aplikaciju i dodjeljivati rezultate igračima.

U ispisu 32., igrač je definiran sa ulogom *user* ali je bitno polje funkcije gdje mu je dodijeljena funkciju *player*. Funkcija je dodijeljena prilikom spremanja u bazu kod kreiranja igrača. Igrač nema nikakve mogućnosti prijave ili administriranja, već će samo moći pratiti rezultate na početnoj stranici.

Polje *function* može poprimiti jedino vrijednosti *player* i *referee*, što je jasno određeno prilikom definiranja *userSchema*. Dio koda *enum: ['player', 'referee']* omogućuje takvo definiranje u MongoDB.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema
var UserSchema = new Schema({
      firstName: String,
      secondName: String,
      username: String,
      password: String,
      roles: {
            type: [{ type: String, enum: ['user', 'admin']}],
            default: ['user']
      },
      function: {
            type: [{ type: String, enum: ['player', 'referee']}]
      },
      sport: String,
      club: String,
    team: String,
      isDeleted: {type: Boolean, default:false},
},{ timestamps: true })

module.exports = mongoose.model('User', UserSchema)
```

Ispis 32: *User shema na bazi podataka*

Od značajnijih polja u *user* modelu može se spomenuti logičko polje *isDeleted* koji je indikator je li neki igrač ili sudac izbrisan. U koliko je izbrisan neće više biti vidljiv administratoru za pretraživanje u bilo kojem djelu aplikacije.

Valja napomenuti i mogućnost *default* koja služi tome da se ne treba striktno brinuti da li će se definirati polje *roles* unutar kôda i postaviti ga na neku vrijednost jer će to MongoDB sam postaviti za nas. Sljedeći primjer je malo kompleksniji jer se koriste naprednije mogućnosti MongoDB kao što je vidljivo u ispisu 33.

Primjer: polje *players* gdje se vidi da će to biti niz igrača, a referira se na *id* igrača *user*. *type: mongoose.Schema.Types.ObjectId* je indikator da ova shema očekuje dobiti jedino id igrača i ništa drugo.

Još jedno zanimljivo polje je *registeredCompetition* gdje se ovaj put referiram na *id* natjecanja i prilikom spremanja *teamSchema* u bazu na tom polju jedino može biti id nekog prijašnje kreiranog natjecanja.

```
var Schema = mongoose.Schema
var teamSchema = new Schema({
      teamName: String,
      players: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
      sport: String,
      teamOnePlayer: { type: mongoose.Schema.Types.ObjectId, ref: 'User'
},
      teamOnePlayerResults: Object,
      teamTwoPlayer: { type: mongoose.Schema.Types.ObjectId, ref: 'User'
},
      teamTwoPlayerResults: Object,
      teamThreePlayer:  {  type:  mongoose.Schema.Types.ObjectId,  ref:
'User' },
      teamThreePlayerResults: Object,
      refereeForThisTeam:  {  type:  mongoose.Schema.Types.ObjectId,  ref:
'User' },
      registeredCompetition: { type: mongoose.Schema.Types.ObjectId, ref:
'Competition'}
},{ timestamps: true })

module.exports = mongoose.model('Team', teamSchema)
```

Ispis 33: *Team shema na bazi podataka*

3.2.6 Korisničko sučelje

Korisničko sučelje napravljeno je na AngularJS platformi za rad s komponentama koje se opisuju te vrše manipulaciju izgleda i prikaza nekog dijela aplikacije. Komponente se sastoje od tri glavne datoteke: datoteka jezika za prezentiranje internet stranice, u kojoj se nalazi predložak koji definira izgled komponente, JavaScript datoteka u kojoj je definiran upravljač za primanje upita, mapiranje podataka i obradu podataka te datoteke za rad sa stilom dokumenta (engl. *Cascading Style Sheets*, CSS). Za komunikaciju između

klijentske strane i internet API-a koristi se `$http` metoda koja sluša poslužitelj. Nakon što poslužitelj obradi podatke, ti isti podaci biti će mapirani u tip podataka kojeg klijentska aplikacija može pročitati i koristiti.

Osnovni dijelovi aplikacije na klijentskoj strani koji se koriste kroz aplikaciju su raspored prikaza (engl. *layout*) koji uključuje zaglavlje, glavni sadržaj i podnožje te servisi. Svaka akcija koja se dogodi u aplikaciji pozvati će funkciju kreiranu u upravljaču te se na taj način poziva određena metoda na API putanju unutar poziva šalje ili prima sve informacije. Poslužitelj sluša zahtjeve klijentske strane i kad se pozove putanja poslužitelja automatski zna gdje treba otići. Sve putanje su kreirane u `server.js` datoteci. Putanje odrađuju provjere ako su potrebne i poziva se funkcija koja radi upit na bazu. Svi upiti u aplikaciji kreirani su unutar mape `backend/queries`. Unutar upita, kreira se ili dohvaća određeni model koji je prethodno kreiran u mapi `backend/models`. Na taj način funkcionira korisničko sučelje i poslužitelj.

Slika 3. prikazuje početnu stranicu gdje je prikazana navigacijska traka sa logotipom saveza i forma za registraciju. U glavnom sadržaju prikazuje se klizač koji u sebi ima slike logotipa klubova koji se natječu na natjecanju. Ispod njega nalazi se popis natjecanja koji još nisu započeli, dok su u podnožju smještene informacije od predstavnika Hrvatskog raketarskog saveza te dvije poveznice `O nama` i `Povijest natjecanja`.

Slika 3 : *Prikaz početne stranice ukoliko nema započetog natjecanja*

U ispisu 34. opisan je način na koji je kreiran klizač, u predlošku je kreirana * * oznaka u kojoj je postavljena AngularJS direktiva *ng-switch* imena *slideshow* s kojom se prikazuje ili skriva element ovisno o izrazu. Unutar * * oznake smješteno je više * * oznaka koje imaju AngularJS direktivu *ng-switch-when* sa atributom kako bi se znalo koja će se slika klizača prva prikazati.

```
<div  class="imageslide" ng-switch='slideshow'>
  <div class="slider-content" ng-switch-when="1">
    <img src="../img/aadz_zadar_logo.gif">
  </div>
  <div class="slider-content" ng-switch-when="2">
    <img src="../img/fanix.jpg">
  </div>
  <div class="slider-content faust" ng-switch-when="3">
    <img src="../img/faust.png">
  </div>
</div>
```

Ispis 34: *Kreiranje klizača u predlošku*

Unutar upravljača vidljiv na ispisu 35., su dvije varijable koje predstavljanju broj klizanja i u kojem će se vremenskom periodu promijeniti slika klizača. Mijenjanje slika u klizaču odrađuje se pomoću usluge *$timeout* koja služi za pozivanje funkcije na određeno vrijeme, a unutar funkcije uvećava se broj klizanja i na taj način mijenja prikaz slika klizača.

```
var slidesInSlideshow = 3;
var slidesTimeIntervalInMs = 3000;
$scope.slideshow = 1;
var slideTimer = $timeout(function interval() {
    $scope.slideshow = ($scope.slideshow % slidesInSlideshow) + 1;
    slideTimer = $timeout(interval, slidesTimeIntervalInMs);
}, slidesTimeIntervalInMs);
```

Ispis 35: *Mijenjanje slike klizaca unutar upravljača*

Ispis natjecanja koja nisu započeta se dobivaju pozivom na metodu *get* na API putanju *api/create/findCompetitions* opisanoj u ispisu 36. Server sluša zahtjev na putanju definiranu u *server.js*.

```
.get('/findCompetitions', function(req, res, next){
  cq.findCompetitions()
  .then(data => res.ok(data))
  .catch(err => res.error(err))
})
```

Ispis 36: *Metoda get na API putanji findCompetitions*

Unutar *findCompetitions* poziva se funkcija traženja natjecanja *findCompetitions* koja se nalazi u *createQueries.js* datoteci.

```
var findCompetitions=function(){
  return Competition.find({$and:[
    {competitionStart:{$gt:Date.now()}},
    {isStarted: false}
  ]}).lean()
}
```

Ispis 37: *Query findCompetitions*

U ispisu 37., *findCompetitions* vraća pretražene podatke iz modela *Competition* na način da preko operatora logičkih upita *$and* obavlja logičku AND operaciju na niz u kojem *competitionStart* mora imati vrijednost većeg datuma od današnjeg što odrađuje *$gt:Date.now()* i *isStarted* mora imati vrijednost *false*. Na taj način dobivamo sva natjecanja koja nisu započeta. Ako postoje takva natjecanja podatci će se ispisati pomoću direktive *ngRepeat* unutar HTML-a. Primjerom je naveden u ispisu 38.

```
<table>
  <tr>
    <th>Redni broj</th>
    <th>Ime natjecanja</th>
    <th>Početak natjecanja</th>
    <th>Lokacija natjecanja</th>
    <th>Opis natjecanja</th>
    <th>Glavni sudac/sutkinja</th>
  </tr>
  <tr ng-repeat="player in findCompetitions" >
    <td>{{ $index + 1 }}</td>
    <td>{{ player.competitionName }}</td>
    <td>{{ player.competitionStart }}</td>
    <td>{{ player.competitionLocation }}</td>
```

```
    <td>{{ player.description }}</td>
    <td>{{ player.chiefReferee }}</td>
  </tr>
</table>
```

Ispis 38: *Primjer ngReapet direktive*

Unutar `<tr>` oznake kreira se AngularJS direktiva `ng-repeat` da prođe kroz stavke niza. Podatci dobiveni od metode `$http.get` vidljivi su u ispisu 39., te se pohranjenu u `$scope` objektu kojem je dodijeljena varijabla člana imena `findCompetitions`. Preko `{{ player.competitionName }}` za svaku stavku polja dobivamo vrijednost imena i prikazujemo ga.

```
$http.get('api/create/findCompetitions').then(function(response){
   $scope.findCompetitions = response.data
})
```

Ispis 39: *Primjer metode „$http.get“ u kotroleru*

Prozor s opisom greške i uspjeha pokaže se korisniku ako se dogodi neka greška tokom prijave prikazanoj na slici 4. Poruke se ispisuju jer je došlo do greške odnosno uspješne komunikacije sa poslužiteljem.

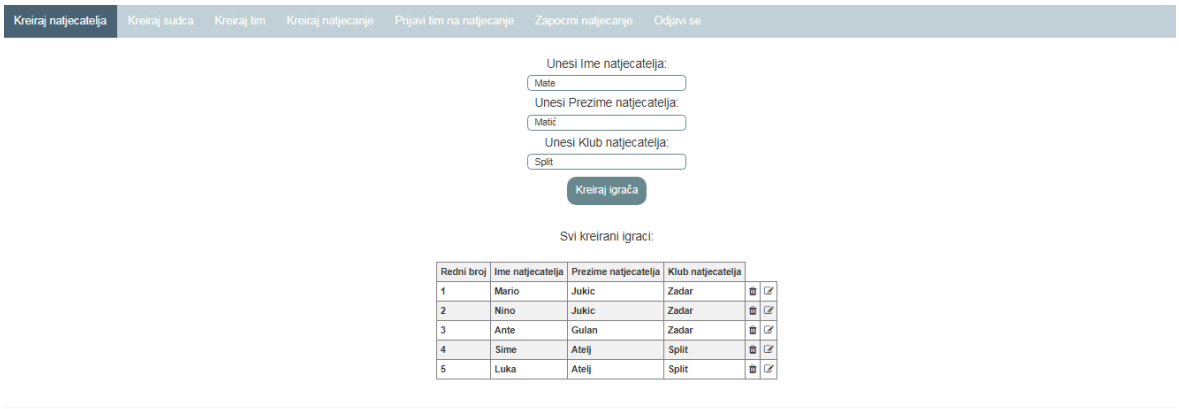
Slika 4: *Primjer prikaza greške i uspjeha u aplikaciji*

Poruka se ispisuje preko `toastr` biblioteke koju je potrebno uključiti u index.html datoteci da bi je uspješno koristili prikazanoj u ispisu 40.

```
toastr.success('You are logged in as ' + user.username)
toastr.error(err.data.msg)
```

Ispis 40: *Korištenje toastr za ispis poruke*

Nakon uspješne prijave kao administrator, aplikacije prikazuje upravljačku ploču sa brojnim opcijama vidljivim na slici 5. Prva opcija upravljačke ploče je kreiranje natjecatelja.



Slika 5: *Prikaz upravljačke ploče sa opcijom kreiranja natjecatelja*

Kako bi se kreirao natjecatelj potrebno je zadovoljiti formu. Forma sadrži ime, prezime i klub natjecatelja. Kada je forma zadovoljena, pritiskom na gumb *`Kreiraj igrača`* poziva se funkcija *`createPlayer`* unutar upravljača. Funkciji *`createPlayer`* prosljeđuju se svi podatci uneseni u formi. Funkcija poziva metodu *`post`* na API putanje *`api/create/player`* i unutar poziva šalje sve informacije koje su unesene u formi. Poslužitelj prepoznaje zahtjev i kada se pozove putanja, automatski se usmjerava na nju. U ovom slučaju putanja koja se poziva definirana je u *`server.js`* datoteci vidljivoj u ispisu 41.

```
app.use('/api/create', require('./backend/api/create'))
```

Ispis 41: *Putanja unutar server.js datoteke*

Unutar putanje isto tako postoji provjera je li forma koja je poslana u nekom slučaju prazan objekt *`!_.isEmpty(req.body)`*, ukoliko nije, nastavlja se daljnja provjera prikazana u ispisu 42. Provjerava se je li je uneseno polje *`firstName`* , *`secondName`* i *`team`*.

```
.post('/player', auth.ensure, checkIsAdmin, function(req, res, next){
      if(!_.isEmpty(req.body)){
        if(req.body.firstName != undefined && req.body.secondName != 
undefined && req.body.team != undefined){
            cq.createPlayer(req.body)
              .then(data => res.ok(data))
              .catch(err => res.error(err))
          }else{
            res.error('input firstName or secondName')
          }
```

```
        }else{
          res.error('input firstName and secondName')
        }
})
```

Ispis 42: *Team shema na bazi podataka*

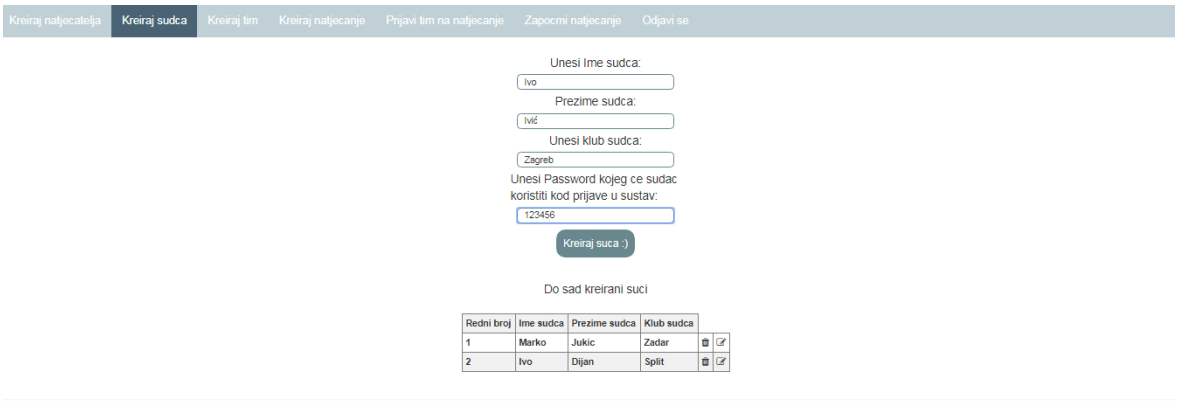
Ako ova provjera prođe kontrolu poziva se funkcija kreiranja igrača *createPlayer* koja se nalazi u *createQueries.js* datoteci.

```
var createPlayer = function(player){
  var player = new User({
    firstName: player.firstName,
    secondName: player.secondName,
    team :player.team,
    function: 'player'
  })
  return player.save()
}
```

Ispis 43: *Team shema na bazi podataka*

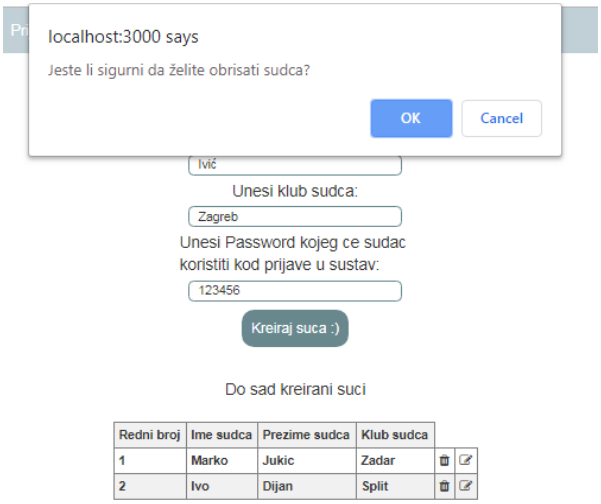
U ispisu 43. vidljiva je funkcija koja kreira *User* u ovom slučaju igrača. Kao što je opisano u dijelu kreiranje modela na bazu podataka, igraču se dodjeljuje funkcija *player*, a on će po defaultu dobiti rolu *user* Funkcija *player.save( )* sprema igrače u bazu. Nakon što se igrač spremi u bazu podataka i ukoliko je sve prošlo u redu vraćamo status 200 natrag u upravljač iz kojeg je pozvana metoda *$http.post*. Nakon primljenog statusa 200 radi se osvježavanje stranice i automatski dohvat svih igrača kreiranih u bazi podataka.

Opcija kreiranja sudca prikazana na slici 6., ima sličnu formu kao i kreiranje natjecatelja. Jedina razlika je u tome što se poziva API putanja *api/create/referee*.



Slika 6 : *Prikaz upravljačke ploče sa opcijom kreiranja sudca*

Unutar liste sudaca može se izbrisati ili urediti sudac kao što je prikazano na slici 7.



Slika 7: *Akcija brisanja sudca*

Unutar tablice kreiranih sudaca je opcija brisanja sudca. Klikom na ikonu `smeće` poziva se direktivu `confirm.js` koja prikazuje poruku u prozoru i dva gumba sa opcijama `OK` i `Cancel`. Ako se klikne na `Cancel` prozor se zatvara i akcija se prekida. Ukoliko se klikne na `OK` pozvat će se funkcija `delateReferee` u upravljaču koja prima `refereeId` kako bi se lakše prepoznao sudac kojeg se želi izbrisati, primjer je vidljiv u ispisu 44. Poziv metode `delete` na putanju `'api/referee/' + refereeId`.

```
$scope.deleteReferee=function(refereeId){
    $http.delete('api/referee/' + refereeId).then((response)=>{
       $scope.allPlayers = response.data
    })
}
```

Ispis 44: *Funkcija deleteReferee u upravljaču*

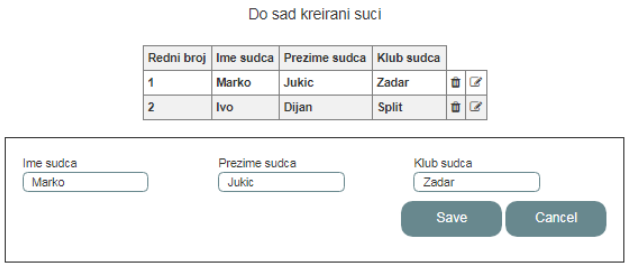
Usluga sluša zahtjev te se poziva putanja. Unutar putanje poziva se *refereeQueries* upit koji ima *deleteReferee* funkciju i prima *id*, što je opisano u ispisu 45.

```
var deleteReferee = function(refereeId){
        return User.findByIdAndUpdate(refereeId, {$set : { isDeleted : true }},{new:true}).exec()
}
```

Ispis 45: *Funkcija deleteReferee u referee-queries.js*

User model napravit će napraviti pretragu po *id*-u i osvježiti ga *findByIdAndUpdate*. Tamo će se postaviti *isDelete : true* i *new : true* kako bi vratilo dokument nakon promjene ažuriranja i na kraju se izvršava kako bi se sudac izbrisao.

Druga opcija u kreiranju sudca je uređivanje (engl. *edit*) kao što je vidljivo na slici 8. Na taj način se mijenjaju podatci u koliko su krivo uneseni ili je došlo do promjena.



Slika 8 : *Uređivanje sudca*

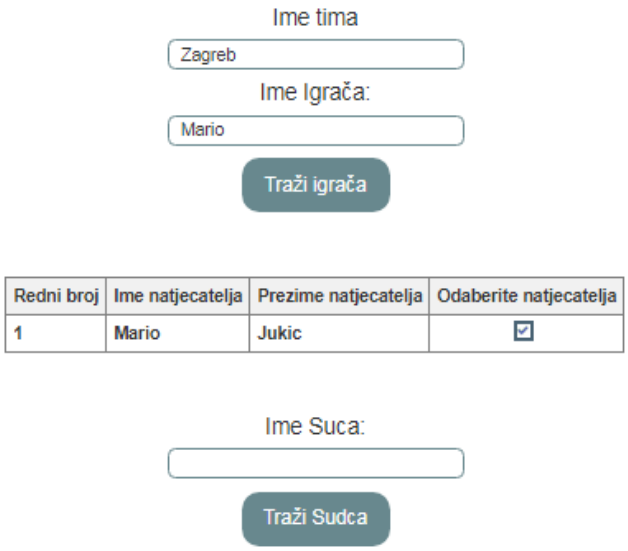
Nakon klika na ikonu uređivanja poziva se funkcija *editReferee* koja šalje objekt *player* gdje se nalaze svi podatci potrebni za održavanje ove forme i prikaže se blok sa podatcima sudca kojeg želimo urediti. Unutar bloka nalaze se tri polja za unos podataka i dva gumba: *Save* za spremanje podataka i *Cancel* za prekid akcije.

Ukoliko se promjene vrijednosti polja i klikne se na *Save* poziva se funkcija *saveEditReferee* koja prima objekt unesenih vrijednosti i id uređenog sudca vidljiv na ispisu 46 .

```
$scope.saveEditReferee=function(referee_edit){.log(referee_edit)
     $http.post("api/referee/edit/" + referee_edit.id , referee_edit)
          .then (function(response){
              $scope.message=response.data
              $window.location.reload()
          })
          .catch(err => toastr.error(err.data))
 }
```

Ispis 46: *Funkcija saveEditReferee u upravljaču*

Nakon pozivanja funkcije poziva se metoda *api/referee/edit* sa *id*-em sudca i objektom izmijenjenih podataka *referee_edit*. Unutar putanje poziva se *refereeQueries* gdje je definirana funkcija *editReferee* koja prima objekt izmijenjenih podataka. Unutar upita poziva se *User* model koji će tražiti sudca po id i promijeniti ime, prezime i klub ukoliko postoji. Kreiranje tima nudi unos ime tima i pretragu imena igrača i sudaca prikazanog na slici 9.



Slika 9: *Kreiranje tima*

Kad se unese ime u polje za unos podataka i kline na gumb *Traži*, prikaže se tablica sa rezultatom pretrage. Način pretrage se odvija tako što prilikom klika na gumb

poziva funkcija u upravljaču, a ona komunicira sa poslužiteljem kao što je vidljivo u ispisu 47.

```
<p class="otherFields">Ime Igrača:</p>
<input class="stopWatch-input" type ="text"  ng-model="player.firstName"
>
<button type="button" class="stopWatch-btn"
        ng-click="findPlayers(player.firstName)">Traži igrača</button>
<div class="tableContainer">
<table ng-show="natjecatelji">
  <tr>
    <th>Redni broj</th>
    <th>Ime natjecatelja</th>
    <th>Prezime natjecatelja</th>
    <th>Odaberite natjecatelja</th>
  </tr>
  <tr ng-repeat="player in allPlayers" >
    <td>{{ $index + 1 }}</td>
    <td>{{ player.firstName }}</td>
    <td>{{ player.secondName }}</td>
  </tr>
</table>
</div>
```

Ispis 47: *Izgleda predloška za pretragu igrača*

Unosom imena u polje pohranjuje se podatak koji će AngularJS direktiva *ng-model* sačuvati pod imenom *player.firstName*. Ng-model direktiva koja predstavlja modele, služi za pohranu podataka i osigurava da se podaci u „pregled i model“ čuvaju u sinkronizaciji cijelo vrijeme

Klikom korisnika na gumb poziva se funkcija pomoću AngularJS direktive *ng-click*. Koristi se obično za gumb kada treba izvršiti neki događaj.

Za prikazivanje i sakrivanje elemenata dobro je koristi AngularJS direktivu *ng-show*. Koristi se na način da joj dodijelimo ime i onda sa vrijednostima *true* ili *false* prikazujemo ili skrivamo elemente. Vrijednosti se definiraju u upravljaču.

Ukoliko je potrebno prikazati neki popis stavki to nam olakšava AngularJS direktiva *ng-repeat* koja koristi vrijednosti definirane unutar našeg upravljača te ih ispisuje.

Ispisuje se unutar {{ }} zagrada gdje je potrebno upisati varijablu koja se koristi za upućivanje na stavku. Na primjer *{{ player.firstName }}*. Gdje je *player* varijabla.

Za pozicioniranje elemenata korisno je koristi Flexbox za centriranje elemenata, postavljanja elemenata okomito ili vodoravno, ravnomjerno raspoređivanje u retku i mnoge druge stvari. Flexbox se u CSS-u definira kao spremnik sa linijom *display: flex;*

flex-direction ima opcije: *row*- koristi se vodoravni poredak sa lijeva na desno,

row-reverse- koristi se vodoravni poredak sa desna na lijevo,

column - okomiti poredak odozgo prema dnu ,

column-reverse- okomiti poredak odozgo prema gore.

justify-content ima opcije: *flex-start*- stavlja stavke prema početku,

flex-end – stavlja stavku prema kraju,

center – centrira stavku,

space-between - ravnomjerno raspoređene u retku, prva stavka je na početku , a posljednja na kraju,

space-around – ravnomjerno raspoređuje u liniji i razmak je jednak oko njih.

Kad neki elementi ne upadaju u rezoluciju ekrana te se stil stranice slomi dobro je koristiti upite medija (engl. *mediaQuery*), kako bi se definirale različite rezolucije ekrana i u njima mijenjali stilovi stranice. Primjer toga vidljiv je u ispisu 48.

```
@media screen and (max-width: 600px) { }
```

Ispis 48: *Primjer media Query-a*

Kad je CSS potrebno mijenjati samo za širinu maksimalno od 600px korisno je upotrijebiti upite medija.

4 Zaključak

U izradi aplikacije za održavanje raketarskog natjecanja korištene su tehnologije bazirane na programskom jeziku JavaScript. Kod izrade klijentskog dijela aplikacije korišten je AngularJS sa podrškom za HTML i CSS.

Poslužiteljski dio kao takav napisan je uz pomoć Node.js platforme, dok se za sustav koji upravlja bazom podataka koristi MongoDB kao NoSql koja je spojen na mLab.

Prilikom kreiranja aplikacije pojavio se problem u dijelu gdje se automatski dodjeljuju sudci natjecateljima zbog Node.js kao asinkrono izvođen JavaScript kôd. Zbog toga kôd je postavljen na način da se sudci ručno dodjeljuju. Također problem se javio i kod kreiranja strukture baze podataka i povezivanje svih potrebnih modela odnosno shema unutar baze podataka. Referenciranje na polja koja su tipa objektID izuzetno su komplicirana.

Pozitivne strane aplikacije su trenutno praćenje rezultata na natjecanju koji se trenutno održava. To omogućuje spretno dizajniran model u kojem unesen rezultat od strane sudca automatski postaje vidljiv korisnicima na početnoj stranici aplikacije. Sve prednosti rada sa Node.js i MongoDB već su prethodno opisane.

Izazov izrade klijentske strane aplikacije olakšala je upotreba AngularJS-a radi njegovih komponenti i raznih funkcionalnosti. Dobra strana AngularJS-a je lagana manipulacija podacima koji se dobiju sa poslužiteljske strane. Datoteke za rad sa stilom dokumenta jednostavne su radi korištenja `flexbox`-a, koji je olakšao pozicioniranje elemenata na stranici i činio ih responzivnim. Upotreba `JWT` tokena također nije jednostavna i za pohranjivanje istog bio je potreban `localStorage` preglednik. Taj token kasnije se provlačio kroz skoro sve moguće putanje na kojima se zahtijevalo da korisnik bude prijavljen.

5 Literatura

Više autora: „What is AngularJS? Arhitecture & Features“ https://www.guru99.com/angularjs-introduction.html (14.08.2019)

Google: „Tutorial“ https://docs.angularjs.org/tutorial (08.09.2018)

MongoDB, Inc: „Welcome to the MongoDB Docs“ https://docs.mongodb.com/ (08.09.2018)

Više autora: „What is MongoDB? Introduction, Architecture, Features & Example“ https://www.guru99.com/what-is-mongodb.html (15.07.2019)

ObjectLabs Corporation: „Quick-Start Guide to mLab“ https://docs.mlab.com/ (08.09.2018)

Više autora: „Node.js Tutorial: Learn in 3 Days“ https://www.guru99.com/node-js-tutorial.html (21.08.2019)

Više autora : „Node.js Express FrameWork Tutorial – Learn in 10 Minutes“ https://www.guru99.com/node-js-express.html (21.08.2019)