

UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

SEMINARSKI RAD

**Primena XML Tehnologija u .NET-u**

Predmet: XML tehnologije

Nastavnik: prof. dr Dušan Tošić

Student: Miloš Stanković, 2014/2013

Beograd, 2013/2014

SADRŽAJ

[1. Istorija *Javascript* jezika - 2 -](#_Toc390721246)

[2. MVC obrazac dizajniranja - 3 -](#_Toc390721247)

[3. MVC i AngularJS - 4 -](#_Toc390721248)

[3.1. Javascript MVC biblioteke - 4 -](#_Toc390721249)

[3.2. AngularJS biblioteka - 4 -](#_Toc390721250)

[3.3. Osnovni primer AngularJS aplikacije - 6 -](#_Toc390721251)

[3.4. Organizacija realne AngularJS aplikacije - 6 -](#_Toc390721252)

# Istorija *Javascript* jezika

Kada se pojavio WWW (*World Wide Web*) ranih 1990-tih godina, sve veb stranice su bile statične. Internet korisnici su mogli da vide stranice upravo onakve kakve su bile postavljene na serverima i nije bila moguća interakcija između korisnika i veb stranica. Da bi stranice bile sposobne za interakciju sa korisnicima, WWW priči nedostajao je deo koji bi to omogućio. Taj deo je trebalo da ima formu programskog jezika kome bi se zadavale instrukcije za odgovor na akcije korisnika. Da bi odgovor bio momentalan bez ponovnog učitavanja stranice, zahtev koji je taj jezik trebalo da ispuni jeste izvršavanje na istom računaru na kome se nalazi internet pretraživač koji prikazuje posmatranu stranicu.

U to vreme dva pretraživača su imala veliku popularnost – *Netscape Navigator* i *Internet Explorer*. Netscape je bio prvi koji je kreirao programski jezik čija je namena bila 'oživljavanje' veb stranica. Taj jezik je dobio ime *Livescript* i bio je ugrađen u pretraživač. To znači da je pretraživač bio sposoban da izvršava naredbe direktno bez potrebe za njihovim prevođenjem. Svako ko je tada koristio poslednju verziju *Netscape* pretraživača bio je u mogućnosti da interaguje sa stranicama koje su u sebi imale *Livescript* kod.

Drugi programski jezik po imenu *Java* postao je veoma poznat, pa je *Netscape* odlučio da proba da to iskoristi tako što će promeniti ime svog jezika u *Javascript*. Iako *Java* i *Javascript* kodovi izgledaju slično, važno je napomenuti da su to potpuno različiti programski jezici koji imaju različitu namenu.

Da ne bi zaostao, *Internet* *Explorer* ubrzo kreira i integriše dva nova jezika iste namene – *vbscript* koji je bio zasnovan na *BASIC* programskom jeziku i *Jscript* koji je bio veoma sličan *Javascript* jeziku. Sličnost jezika *Jscript* i *Javascript* je bila toliko velika tako da je pažljivim programiranjem bilo moguće napraviti kod koji se izvršavao u *Netscape*-u kao kod *Javascript*-a, a u *Internet Explorer*-u kao kod *Jscript*-a.

Važnost ovog skript jezika je bila prevelika da bi njegov razvoj bio zasnovan na međusobnom takmičenju programera dva internet pretraživača, pa je 1996. godine organizacija ECMA (*European Computer Manufacturers Association*) preuzela odgovornost za *Javascript*. Rezultat toga je standardizacija i preimenovanje *Javascript* jezika u *ECMAScript* ili *ECMA-262*, ali većina ljudi i dalje koristi njegov stari naziv.

Od tada do danas populanost *Javascript* jezika je u stalnom porastu. Svakodnevno nastaju nove tehnologije i biblioteke zasnovane na *Javascript*-u. Među tehnologijama ističe se AJAX (*Asynchronous Javascript And XML*) koja omogućuje asinhronu izmenu internet stranica pomoću razmene male količine podataka između internet pretraživača i servera. Neke od poznatih biblioteka su *jQuery*, *Prototype*, *Backbone*, *AngularJS*. *AngularJS* je popularna MVC *Javascript* biblioteka koja implementira poznati MVC obrazac dizajniranja, i o njoj će biti više reči u nastavku rada. Najpre će u sekciji 2 biti razmatran MVC obrazac dizajniranja. Sledeća sekcija se bavi opštim osobinama *AngularJS* biblioteke, dok će u sekciji biti prikazan postupak razvoja realne aplikacije primenom *AngularJS*-a. Sama aplikacija je sastavni deo ovog rada.

# MVC obrazac dizajniranja

Osnovna ideja obrazaca dizajniranja (*design patterns*) je da izdvoje nacrt rešenja za situacije (probleme) koje se često javljaju i ponavljaju u različitim sferama tokom razvoja softvera. Obrazac dizajna ne predstavlja gotovo rešenje problema koje se može direktno pretočiti u kod. To je opis rešenja problema koje se može primeniti u različitim situacijama.

Obrasci dizajniranje nisu vezani ni za jedan programski jezik, biblioteku (*framework*) i slično, već predstavljaju rešenja koja su se kroz praksu dokazala kao dobra (najčešće i najbolja). Zbog toga, oni mogu da značajno ubrzaju razvoj softvera. Nema potrebe trošiti vreme na pronalaženje rešenja problema, već se može odmah krenuti sa implementacijom obrasca dizajniranja za konkretni problem. Druga veoma bitna prednost obrazaca dizajniranja je ta što poboljšavaju komunikaciju između programera. Programeri mogu da lakše stvore sliku o nekom softverskom proizvodu ukoliko znaju koji od obrazaca dizajniranja je bio korišćen tokom njegovog razvoja.

MVC (*Model-View-Controller*) predstavlja primer jednog obrasca dizajniranja. On nije povezan konkretno sa AngularJS-om, već je uspešno implementiran u različitim programskim jezicima. MVC predstavlja arhitekturu na kojoj se zasniva najveći broj modernih veb, Android, iOS i desktop aplikacija. Na slici Slika 1 – MVC obrazac dizajniranja prikazan je MVC obrazac dizajniranja koji će u nastavku biti detaljno opisan.



Slika 1 – MVC obrazac dizajniranja

MVC se sastoji od tri osnovne logičke komponente: model podataka (*model*), prezentacioni deo ili pogled (view), i kontrolni deo (*controller*).

Model se sastoji od čistih podataka koji se dobijaju iz baze podataka, veb servisa ili iz nekog drugog izvora, i koje je potrebno prikazati korisniku. Zbog toga su modeli najčešće POJO (*Plain Old Java Object*) objekti [x]. POJO klase su jednostavne klase koje ne nasleđuju neke specijalne Java klase. One se sastoje od skupa (privatnih) polja i odgovarajućih *get* i *set* metoda, a njihova uloga je u prenosu podataka.

Pogled je komponenta koju korisnik direktno vidi. Pogled prikazuje podatke iz modela, pri čemu ne zna ništa o semantici tih podataka. U eri mobilnih uređaja i velikog broja različitih ekrana na kojima se pogled prikazuje, česta je pojava da se za određeni model definiše nekoliko pogleda.

Kontroler predstavlja deo MVC-a koji koordinira između modela i različitih pogleda na aplikaciju. On osluškuje događaje koji se dešavaju na pogledu (npr. klik na dugme) i izvršava odgovarajuće akcije pridružene tim događajima. Neke od tih akcija mogu se odnositi na dohvatanje novih podataka iz baze podataka od kojih se formira novi model, a zatim se taj model prosleđuje nekom drugom pogledu koji korisniku prikazuje zahtevane podatke.

Prva očigledna prednost MVC obrasca jeste razdvajanje prezentacionog dela aplikacije od njenog logičkog dela. Podrška različitim vrstama korisnika koji koriste različite vrste uređaja je standardni zahtev današnjih veb aplikacija. Korisnički interfejs mora biti različiti ako zahtev za aplikacijom stiže od desktop računara ili mobilnog uređaja. U oba slučaja model je isti, tj. predstavlja iste podatke koje je potrebno prikazati. Kontroler ima ključnu ulogu u izboru pogleda koji će prikazati zahtevane podatke. Pored izdvajanja pogleda iz poslovne logike aplikacije, MVC smanjuje kompleksnost velikih aplikacija. Kod je struktuiraniji, pa je samim tim jednostavnije upravljati njime, testirati ga i ponovo koristiti.

# MVC i *AngularJS*

## *Javascript* MVC biblioteke

Moderne veb aplikacije se sastoje od skupa kontrolera, modela i pogleda. Obično postoji jedan osnovni kontroler koji prihvata sve zahteve, a zatim ih prosleđuje odgovarajućim specifičnim kontrolerima. Kontoler najpre poziva servisni kod koji izračunava model, a zatim određuje odgovarajući pogled. Postoje čitave biblioteke koje implemeniraju MVC obrazac i omogućuju kreiranje MVC veb aplikacija. Neke od njih su Spring MVC i ASP.NET MVC. Na primer, kod Spring MVC biblioteke kontroleri su posebne Java klase, modeli su (najčešće) POJO Java klase, dok su pogledi JSP (*Java Server Page*) dokumenti koji se prevode u HTML.

Popularnost *Javascript*-a je dovela do toga da se sve više poslovne logike veb aplikacija odvija na klijentskoj strani. Takođe, sa razvojem AJAX (*Asynchronous Javascript And XML*) tehnologije nastala je ideja o kreiranju jednostranih veb aplikacija (*single page web applications*). Jednostrane veb aplikacije se sastoje od jedne osnovne strane. Tokom interakcije korisnika sa aplikacijom, umesto da se ponovo učitavaju kompletne stranice, AJAX tehnologijom se delovi početne strane zamenjuju drugim delovima, tako da korisnik ima osećaj da radi sa klasičnom desktop aplikacijom. Zbog svega navedenog, količina *Javascript* koda kod prosečne veb aplikacije u stalnom je porastu, pa se javila potreba za boljom organizacijom skript koda. Tako nastale čitave *Javascript* biblioteke (frameworks) koje implementiraju različite obrasce dizajniranja, pa i MVC. Najpoznatije biblioteke koje implementiraju MVC obrazac dizajniranja su Backbone.js, Ember.js i *AngularJS*.

## *AngularJS* biblioteka

*AngularJS* je jedna od najpopularnijih *Javascript* MVC biblioteka. Prvu verziju ove biblioteke objavili su Miško Hevery i Adam Abrons iz kompanije Brat Tech LLC 2009. godine. Ubrzo je Adam Abrons napustio projekat, dok je Miško Hevery nastavio da radi na *AngularJS*-u u okviru Google-a.

Od trenutka kada se Google uključio u razvoj *AngularJS* biblioteke, njena popularnost stalno raste, što se može videti na slici Slika 2. Prikazana slika predstavlja grafik interesovanja za termine '*angularjs'*, '*backbone.js*' i '*ember.js*' u periodu od januara 2011. do juna 2014. godine. Interesovanje navedenih termina se zasniva na njihovoj učestalosti u pretraživanjima na Google internet pretraživaču. Grafik je dobijen na sajtu *Google Trends* ([www.google.com/trends](http://www.google.com/trends)), a broj pojavljivanja pomenutih termina skaliran je od 0 do 100.



Slika 2 – Grafik interesovanja za termine '*angularjs*', '*backbone.js*' i '*ember.js*'

Pošto se ovaj rad bavi klijentskim (skript) delom veb aplikacije, zbog jednostavnosti u radu će se za njega koristiti termin 'veb aplikacija' ili samo 'aplikacija'.

Pre svega, *AngularJS* je biblioteka koja definiše različite koncepte za pravilno organizovanje (klijentskog dela) veb aplikacije. Aplikacija se organizuje kao skup modula koji mogu da zavise jedni od drugih, a međusobno referenciranje modula ostvaruje se pomoću tzv. umetanja zavisnosti (*Dependency Injection* - DI). Više o DI može se naći na [x]. Zahvaljujući DI, često korišćeni delovi aplikacije mogu se organizovati kao posebni servisni moduli koji se po potrebi mogu umetati i pozivati bilo gde u aplikaciji.

*Angular* unapređuje HTML posebnim jednolinijskim izrazima (*inline expressions*). U cilju kreiranja efikasnijih pogleda, *Angular* omogućuje izvršavanje jednolinijskih izraza unutar HTML koda koji se zadaju unutar dvostrukih vitičastih zagrada. U tim izrazima dostupan je *Javascript* kod pomoću koga je moguće izvršiti određena izračunavanja i prikazati željeni rezultat. Na slici Slika 3 prikazan je jedan jednolinijski izraz i njegov rezultat.



Slika 3 – Primer jednolinijskog izraza

Takođe, *Angular* proširuje HTML novim elementima i atributima koji se nazivaju direktive (*directives*). Direktive predstavljaju jedno od najmoćnijih funkcionalnosti *Angular*-a. One služe za dodatno struktuiranje HTML koda prilagođavajući ga za podatke u modelu koje je potrebno prikazati. Neke od već definisanih direktiva su *ngApp*, *ngView*, *ngController*, *ngRepeat*, a moguće je i definisati sopstvene direktive. Više o direktivama u nastavku.

Značajna osobina *Angular*-a je i dvosmerno povezivanje podataka (*two way binding*). Dvosmerno povezivanje podrazumeva menjanje modela paralelno sa interakcijom korisnika. Na primer, kada korisnik unosi neki tekst u tekstualno polje HTML forme, paralelno sa unosom menja se i modelni objekat. To znači da nije potrebno osluškivati specifične događaje, reagovati na njih i eventualno ručno menjati HTML, već takve akcije *Angular* sam uspešno obavlja.

Zahvaljujući DI, *Angular* omogućuje jednostavno i efikasno testiranje *Javascript* koda. Postoje posebne biblioteke koje služe za testiranje *AngularJS* aplikacije. U ovom radu testiranje aplikacije neće biti razmatrano, a više o testiranju može se naći na [x].

## Osnovni primer *AngularJS* aplikacije

U ovoj sekciji biće prikazana najjednostavnija *AngularJS* aplikacija. U ovom i u narednim primerima koristiće se verzija 1.2.18 *AngularJS* biblioteke. Na slici Slika 4 prikazana je 'Zdravo svete' aplikacija, gde je na levoj strani slike dat kod aplikacije, dok je na desnoj strani prikazan rezultat izvršavanja aplikacije.



Slika 4 – Primer 'Zdravo svete' programa

Prvo što se može primetiti jeste da se u otvorenom *html* tagu nalazi dodatni atribut *ng-app* (linija 2). Ovaj atribut označava da će HTML dokument predstavljati *Angular* aplikaciju. Navedeni atribut je ugrađena *Angular* direktiva koja ima naziv *ngApp*. Pošto HTML ne razlikuje velika i mala slova, direktive se u HTML dokumentu navode malim slovima sa povlakama na mestima gde bi trebalo da se nađe veliko slovo.

Zatim, linija 8 se sastoji od izmenjenog *h1* taga. Njegov sadržaj je jednolinijski izraz, a ima i atribut *ng-controller*. Ovaj atribut predstavlja ugrađenu direktivu *ngController*, koja služi za navođenje kontrolera koji će upravljati HTML elementom i svim HTML elementima koji se nalaze unutar njega. U ovom slučaju je to *HelloWorldCtrl* koji će kasnije biti definisan.

U liniji 10 se navodi *script* tag kojim se učitava AngulraJS biblioteka, dok se u linijama 11-15 definiše *HelloWorldCtrl*. Ovaj kontroler je obična *Javascript* funkcija sa odgovarajućim imenom. U ovom slučaju kontroler prihvata jedan argument - *$scope*, koji je specijalan *Angular*-ov objekat. Svakom kontroleru se pridružuje jedan *$scope* objekat koji služi da poveže kontroler sa modelom. Važno je napomenuti da *$scope* nije model, već je to objekat koji sadrži model (tj. podatke modela). U navedenom primeru se nad *$scope* objektom definiše dodatno polje (promenljiva) *helloMessage* i postavlja se njena vrednost. Kada se stranica učitava, internet pretraživač najpre učita početak stranice. Kada dođe do kontrolera, postavlja se odgovarajuća vrednost u polje *helloMessage*. Pošto *Angular* podržava dvosmerno povezivanje podataka, ta vrednost će istog trenutka biti vrednost jednolinijskog izraza u liniji 8, pa će se ona i prikazati na stranici. Razlog zbog čega se skript tagovi nalaze na kraju dokumenta je taj da bi se sprečilo izvršavanje *Angular* i *Javascript* koda pre nego što stranica bude učitana.

## Organizacija realne *AngularJS* aplikacije

Iz prethodnog primera može se steći osnovna slika o tome kako *AngularJS* biblioteka funkcioniše. Ipak, može se primetiti da prikazani primer ne razdvaja najbolje pojmove modela, pogleda i kontrolera, jer se cela aplikacija nalazi u jednom dokumentu. Međutim, ovo je osnovni primer koji služi da ostvari prvi kontak programera sa *Angular* bibliotekom, dok se realne aplikacije drugačije organizuju. Kod realnih *Angular* aplikacija sve komponente su potpuno razdvojene i nalaze se u posebnim dokumentima. Generišu se posebni dokumenti za kontrolere, poglede, servise, (korisnički definisane) direktive, itd., i oni se se raspoređuju po folderima koji imaju posebnu strukturu. Da bi početak kreiranja *Angular* aplikacije bio jednostavniji, Google i *Angular* tim su napravili poseban početni projekat, koji se naziva *Angular Seed*.

*Angular Seed* je dobra početna tačka od koje treba krenuti u realizaciju sopstvene aplikacije. On obezbeđuje osnovnu organizaciju dokumenata i datoteka, koja se može modifikovati prema sopstvenoj želji i potrebama. Ovaj projekat je dostupan na sledećoj adresi [x] (<https://github.com/angular/angular-seed>), gde se mogu naći i detaljnije informacije o projektu.

Pored osnovne organizacije datoteka i dokumenata, uz *Angular* Seed dolazi i poseban veb server koji služi za potrebe učenja. To je jednostavni veb server koji može da prihvata i obrađuje HTTP zahteve i da odgovara na njih. Kada se veb server instalira i pokrene, *Angular* Seed aplikacija je dostupna na *http://localhost:8000/app*.

Nakon što se projekat skine i izvrše sva podešavanja opisana na [x], dobija se struktura projekta kao na slici Slika 5. U levom delu slike prikazan je celokupni sadržaj projekta. Datoteka *test* sadrži dokumente vezane za testiranje aplikacije (što nije predmet ovog rada). U datoteci *app* će se nalaziti aplikacija, dok su ostale datoteke i dokumenti konfiguracione prirode (i neće biti razmatrani u nastavku). Prva datoteka unutar *app* datoteke je *bower\_components* u kojoj se nalaze *AngularJS* komponente. Na primer, centralni modul *Angular* biblioteke se nalazi u dokumentima *angular.js* i *angular.min.js* unutar datoteke *angular*. Ipak, centralni deo korisničke aplikacije nalaziće se u datoteci *js* (na slici desno). Pošto se pomoću *AngularJS* biblioteke kreiraju jednostrane aplikacije, dokument *index.html* će biti osnovna strana aplikacije. Njeni delovi će se menjati delovima koji su smešteni u HTML dokumenta unutar datoteka *partials*. Ovako organizovan projekat biće polazna tačka za razvoj realne aplikacije u sekciji 4, koji će biti prikazan kroz korake.



Slika 5 – Struktira *Angular Seed* projekta

# Razvoj realne aplikacije

Najbolji način da se stekne uvid o tome kako se koristi *AngularJS* biblioteka jeste implementacija realne aplikacije. U ovoj sekciji biće predstavljena i u koracima implementirana aplikacija pod nazivom *EventApp*. Ova aplikacija će biti u mogućnosti da kreira neke događaje, da upravlja njima, da vodi evidenciju o sesijama svakog od događaja. Na primer, događaji mogu biti jednodnevna predavanja o *Javascript* bibliotekama, pri čemu svako predavanje ima više sesija koje se bave različitim funkcionalnostima određene biblioteke.

Pošto *AngularJS* omogućuje kreiranje jednostranih veb aplikacija, osnovna strana *EvenApp* aplikacije smeštena je u dokumentu *index.html*. Sadržaj ovog dokumenta predstavljen je na slici Slika 6. Prvo što se može primetiti je postojanje atributa *ng-app* unutar *html* taga čija je vrednost upravo naziv aplikacije. Ovaj atribut je posebna *Angular*-ovadirektiva koja se naziva *ngApp* i služi označi osnovnu strana aplikacije. Vrednost *ng-app* direktive je ujedno i naziv glavnog modula aplikacije koji se definiše u *app.js*. Definicija glavnog modula data je u sekciji 4.1., a sadržaj *app.js* dokumenta prikazan je na slici Slika 7. Na početku *body* elementa nalazi se neuređena lista koja predstavlja navigacionu liniju aplikacije. Zatim u liniji 14 se nalazi *div* element sa atributom *ng-view*, koji je posebna *Angular*-ova direktiva. Ova direktiva označava da će *div* element biti mesto gde će se pogledi učitavati. U liniji 16 se učitava osnovna *AngularJS* biblioteka, a u liniji 17 se učitava posebni modul *AngularJS*-a koji obezbeđuje rutiranje (više o rutiranju u sekciji 4.1.). Nakon toga se učitava centralni dokument *app.js* (linija 18), koji predstavlja početno mesto učitavanja korisničke aplikacije, a zatim se učitavaju i ostali delovi aplikacije (linije 19-22).



Slika 6 – Osnovna stranica jednostrane aplikacije

## Rutiranje

Kod tradicionalnih veb aplikacija URL-ovi ka nekim resursima se mapiraju na fizičke lokacije na serveru čija relativna adresa odgovara URL-u. Na primer, ako korisnik unese u internet pretraživač adresu [*http://www.example.com/resources/page01.html*](http://www.example.com/resources/page01.html), internet pretraživač kreira zahtev za stranicom *page.html*. Ta stranica se fizički nalazi u podfolderu *resources* korenog direktorijuma veb aplikacije.

Moderne MVC veb aplikacije imaju drugačije ponašanje koje se zasniva na rutiranju. Sve MVC biblioteke imaju implementiranu ovu funkcionalnost koja se koristi za mapiranje URL-ova sa odgovarajućim kontrolerima. Svakom URL-u se dodeljuje po jedan kontroler. Kada se unse određeni URL, odgovarajući kontroler se izvršava. Na kraju izvršavanja kontroler određuje koju pogled (stranicu) će prikazati. Na taj način se razdvaja fizička lokacija resursa od URL-ova. Link iz prethodnog primera sada može biti [*http://www.example.com/page01*](http://www.example.com/page01), pri čemu kontroler određuje da će prikazati HTML dokument čija je relativna adresa */resources/page01.html*.

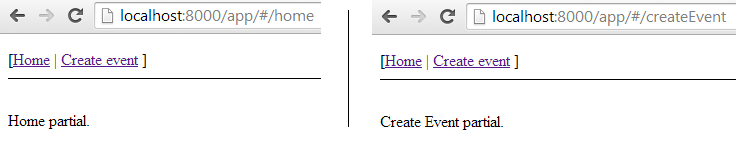
Rutiranje definiše globalno ponašanje aplikacije, pa se njegova implementacija nalazi u dokumentu *app.js* (Slika 7). U ovom dokumentu se najpre kreira glavni modul izvršavanjem funkcije *module* (linije 1-7). Prvi parametar ove funkcije je naziv modula, drugi je niz naziva modula od kojih zavisi modul koji se kreira, a povratna vrednost je objekat koji predstavlja definisani modul. Prva zavisnost *EventApp* modula je modul *ngRoute* i on je definisan u *AngularJS* biblioteci. Da bi bio dostupan, potrebno je učitati odgovarajući skript (Slika 6, linija 17). Ostali moduli u nizu zavisnosti su korisnički moduli i o njima će biti više reči u nastavku.

Slika 7 – Rutiranje aplikacije

U linijama 9-19 konfiguriše se rutiranje. Nad *eventApp* objektom poziva se funkcija *config* koja služi za konfiguraciju aplikacije. Njen argument je funkcija koja će se pozvati da izvrši zahtevanu konfiguraciju. Ona prihvata promenljiv broj parametara, a u ovom slučaju naveden je samo jedan parametar - *$routeProvider*. To je objekat koji predstavlja servis definisan u *ngRoute* modulu. U linijama 10-13 konfiguriše se ruta */home*. Navodi se naziv njoj pridruženog kontrolera (linija 12) i putanja do pogleda (linija 11). Kada korisnik otvori URL [*http://localhost:8000/app/#/home.html*](http://localhost:8000/app/#/home.html), najpre se učitava osnovna stranica aplikacije *index.html*, zatim se izvršava pridruženi kontroler, a *div* element sa atributom *ng-view* zamenjuje se sadržajem pogleda (*home.html*). Slično se definiše i ruta */createEvent*. U liniji 18 navodi se da je podrazumevana ruta */home*, tako da će korisnik biti redirektovan na nju ako greškom unese nepostojeći URL (tj. nepostojeću rutu) u adresnu liniju internet pretraživača.

|  |  |
| --- | --- |
| Slika 8 – Definicija kontrolera | Slika 9 – Definicija pogleda |

Da bi ovako konfigurisana aplikacija mogla da radi, potrebno je definisati kontrolere i poglede. Za početak, definicija kontrolera i pogleda biće osnovna. Kontrolere treba definisati u dokumentu *controllers.js* kao na slici Slika 8. Najpre se definiše poseban modul aplikacije koji će sadržati kontrolere (linija 1), a zatim se definiše svaki kontroler posebno. Definicija kontrolera vrši se pozivom funkcije *controller* nad objektom koji predstavlja modul. Funkcija *controller* prihvata dva argumenta. Prvi argument je naziv kontrolera, dok je drugi argument funkcija koja predstavlja kontroler. *HomeCtrl* je definisan u linijama 3-5,*CreateEventCtrl* u linijama 7-9, i za sada ovi kontroleri su prazne funkcije. Pogledi se definišu kao HTML dokumenti unutar direktorijuma *partials*. Za početak pogledi se sastoje od samo jednog *p* elementa čiji je sadržaj dat na slici Slika 9. Na slici Slika 10 – Prikaz pogleda u internet pretraživaču dat je prikaz oba pogleda u internet pretraživaču. Klikom na link *Home* otvara se (podrazumevani, početni) pogled, dok se klikom na link *Create event* otvara pogled za kreiranje događaja. Implementacija ovih pogleda biće postupno proširavana u narednim sekcijama.



Slika 10 – Prikaz pogleda u internet pretraživaču

## Prikaz resursa i direktiva *ngRepeat*

Jedna od najčešće korišćenih direktiva u *Angular*-uje *ngRepeat*. Ova direktiva služi za ciklično generisanje delova HTML dokumenta. Ona zapravo predstavlja *for* (ili *while*) petlju ugrađenu u HTML.

Početna strana *EventApp* aplikacije služi za prikazivanje osnovnih podataka o događajima iz (imaginarne) baze podataka. U ovoj aplikaciji baza podataka biće predstavljena JSON dokumentom (*Javascript Object Notation*) koji se nalazi na relativnoj putanji */app/data/events.json*. Njegov sadržaj je niz *Javascript* objekta, a deo tog niza je dat na slici Slika 11. Kompletan niz može se videti u dokumentu uz priloženu aplikaciju. Niz se sastoji od 4 objekta koji predstavljaju događaje. Prva dva imaju sesije, dok kod preostala dva sesije nisu definisane. Pored sesija, događaji imaju podatke o nazivu, datumu i vremenu održavanja, adresi... Sesija ima informacije o nazivu, predavaču, trajanju u satima, nivou (početni, srednji, napredni) i apstraktu. Pošto se radi o simulaciji baze podataka, ovi podaci će biti učitani u *Javascript* promenljivu *eventsArray*. Ovu promenljivu je potrebno dodati u dokument *index.html* unutar skript taga. U nastavku biće više reči o dohvatanju podataka iz dokumenta *events.json*.

Osnovni podaci o događajima prikazuju se na početnoj strani, a pored svakog događaja stoji link ka njegovom detaljnijem prikazu. Dakle, kada se otvori početna strana potrebno je učitati podatke iz dokumenta, a za te potrebe definiše se posebna funkcija. Ova funkcija je smeštena u poseban servisni modul aplikacije. Ideja servisnog mogula je da obuhvati koristan i često korišćeni kod. Servisni modul se definiše unutar dokumenta *services.js*. Definicija servisa i funkcije za dohvatanje svih događaja prikazana je na slici Slika 12. Najpre se u liniji 1 definiše servisni modul, a definicija servisa obavlja se pozivanjem funkcije *factory* (linija 3). Slično kao i kod kontrolera, prvi argument ove funkcije je naziv servisa (*EventDataService*), a drugi je funkcija kojom se definiše servis. Za pošetak, servis će zavisiti od dva *Anguler-*ova servisa *$http* i *$q*, pa ih je potrebno navesti kao argumente funkcije. Sam servis predstavljen je objektom koji je povratna vrednost ove funkcije. Taj objekat sastoji se od polja koja su vezana za korisničke funkcije. Više o http i q

U linijama 4-14 definiše se funkcija *getAll* za dohvatanje svih događaja. *$http* servis omogućuje slanje HTTP zahteva serveru kroz metode *get* i *post* na koje je moguće nadovezati pozive funkcija *success* i *error*. U ovom slučaju, koristi se funkcija *get* (linija 7) koja kao argument prihvata adresu resursa. Funkcija *post* je slična i pored adrese, kao drugi argument prihvata podatke koji se šalju na server. U slučaju uspešnog pristupa resursu asinhrono se izvršava funkcija *success*. Suprotno, ako nastane greška asinhrono će se izvršiti funkcija *error*. Obe funcije kao argument prihvataju metode sa parametrom *data*, a ovaj parametar se može proizvoljno nazvati. Parametar *data* služi kao referenca na podatke koje dospeju sa servera, tako da će u slučaju uspešnog GET zahteva podaci sa servera biti smešteni upravo u ovu promenljivu.



Slika 11 – Deo sadržaja dokumenta *events.json*

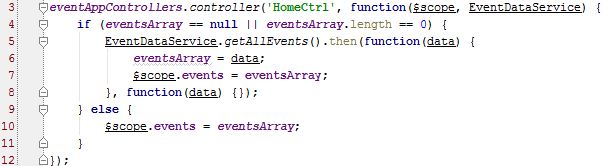
Na prvi pogled deluje da je dovoljno da se u liniji 8 kao povratna vrednost funkcije vrati parametar *data* (tj. da stoji *return data;*). Međutim, metode *get* i *post* su asinhrone. To znači da se ne čeka na povratne podatke sa servera, već kod nastavlja sa izvršavanjem. Ako bi se u liniji 8 našao kod *return data;* , parametar *data* bi imao vrednost *undefined* jer je kod nastavio sa izvršavanjem, a podaci sa servera nisu stigli. Da bi se ova neželjena pojava izbegla koristi se *$q* servis. Najpre je potrebno kreirati objekat *deferred* koji služi za odloženu akciju (linija 5). Ovaj objekat ima u sebi dve metode *resolve* i *reject*, kao i poseban objekat *promise*. Objekat *promise* je povratna vrednost funkcije, a nad njime je moguće pozvati metodu *then.* Metoda *then* prihvata dve funkcije kao argumente (slika 13). Prva funkcija se povezuje sa metodom *resolve* i služi za slučaj uspešnog dohvatanja podataka, dok se druga funkcija povezuje sa metodom *reject* koja se poziva u slučaju greške.

Funkciji *getAll* se izvan definicije servisa pristupa kao *getAllEvents*. Ova metoda biće pozvana unutar kontrolera *HomeCtrl* (slika 13).



Slika 12 – Definicija servisa *EventDataService*

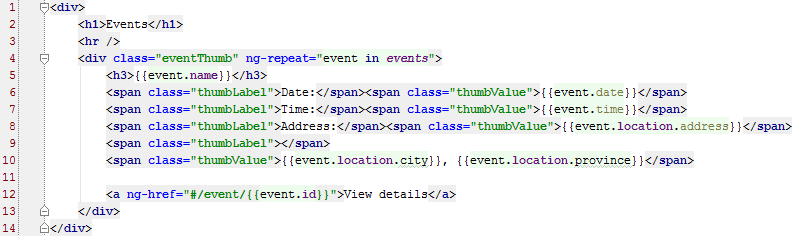
Da bi se pozvala funkcija *getAllEvents* servisa *EventDataService* unutar nekog kontrolera, potrebno je taj servis navesti kao zavisnost željenog kontrolera. Na slici 13 prikazana je definicija *HomeCtrl* kontrolera. On zavisi od servisa *EventDataService*, pa se taj servis prosleđuje kao parametar funkcije (linija 3). Zbog osobine umetanja zavisnosti *Angular* će funkciji proslediti objekat koji predstavlja traženi servis. U liniji 4 se ispituje da li su podaci o događajima već učitani. Ako nisu, u linijama 5-7 vrši se njihovo učitavanje, inače se vrednost polja *events* objekta *$scope* postavlja na vrednost niza *eventsArray*.



Slika 13 – Pozivanje servisa iz kontrolera

U slučaju da je potrebno podatke o događajima učitati iz dokumenta, potrebno je pozvati metodu *getAllEvents* servisa *EventDataService*. Kao što je ranije rečeno, ova metoda vraća specijalni objekat *promise* nad kojim se može pozvati metoda *then*. Upravo je to urađeno u liniji 5. Metoda *then* prihvata dve funkcije. Prva od njih se izvršava u slučaju uspešnog učitavanja dokumenta, a druga služi za obradu greške ukoliko do nje dođe. Obrada greške neće biti razmatrana. Jedan od načina za obradu greške može biti ispisivanje odgovarajuće poruke. U slučaju uspešnog učitavanja podataka, oni će biti smešteni u parametar *data*. Ova vrednost se postavlja za vrednost niza *eventsArray* (linija 6), a zatim se polje *events* objekta *$scope* inicijalizuje na istu tu vrednost.

Objekat *$scope* predstavlja vezu između pogleda i kontrolera. Definisanjem i inicijalizacijom polja *$scope* objekta, ta polja postaju "vidljiva" u HTML kodu. U situaciji sa slike 13, podaci koji su smešteni u *$scope.event* postaju dostupni u HTML-u. Direktiva *ngRepeat* služi za iteraciju kroz te podatke i generisanje delova HTML-a. Primena ove direktive data je na slici 14. U ovom slučaju, direktiva iterira kroz niz događaja i za svaki element niza generiše se po jedan *div* element. Inicijalizacija petlje navodi se unutar vrednosti *ng-repeat* atributa. Promenljiva za prolaz kroz niz događaja je *event*, dok promenljiva *events* sadrži kolekciju događaja. Promenljiva *events* referiše na polje objekta *$scope* sa istim imenom, odnosno na polje *$scope.events*, koje predstavlja niz svih događaja.



Slika 14 – Primena direktive *ngRepeat* u dokumentu *home.html*

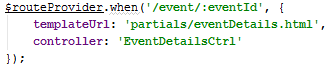


Slika 15 – Početna strana aplikacije nakon izvršavanja *ngRepeat* direktive.

Rezultat izvršavanja *ngRepeat* direktive dat je na slici 15. Kao što se može primetiti, direktiva je za svaki događaj generisala po jedan *div* element, a unutar njega vrednosti jednolinijskih izraza (unutar dvostrukih vitičastih zagrada) su zamenjene odgovarajućim vrednostima. Na kraju svakog *div*-a generisan je i link koji vodi na stranu sa detaljnijim prikazom događaja (slika 14, linija 12). Link se dinamički generiše, različit je za svaki događaj i zavisi od vrednosti *id*-ja svakog događaja. U sledećoj sekciji biće objašnjena primena takozvanih parametrizovanih ruta.

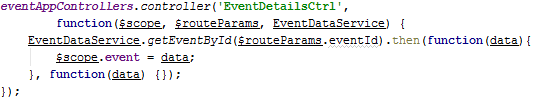
## Parametrizovane rute i servis *$routeParams*

U prethodnoj sekciji je pomoću direktive *ngRepeat* generisan HTML za prikaz svih događaja. Za svaki od događaja prikazani su osnovni podaci, a takođe generisan je link koji vodi do strane sa detaljnijim prikazom događaja. Generisani link zavisi od ID-ja i razlikuje se samo u poslednjem delu (slika 14, linija 12). Ideja MVC-a i rutiranja je da se klikom na svaki od generisanih linkova prikazuje isti pogled sa različitim podacima (modelom). Na osnovu ID-ja (koji se nalazi u linku) dohvataju se i prikazuju podaci za konkretan događaj. Da bi to bilo moguće potrebno je dodati novu rutu u datoteci *app.js*, kao što je prikazano na slici 16.



Slika 16 – Parametrizovana ruta

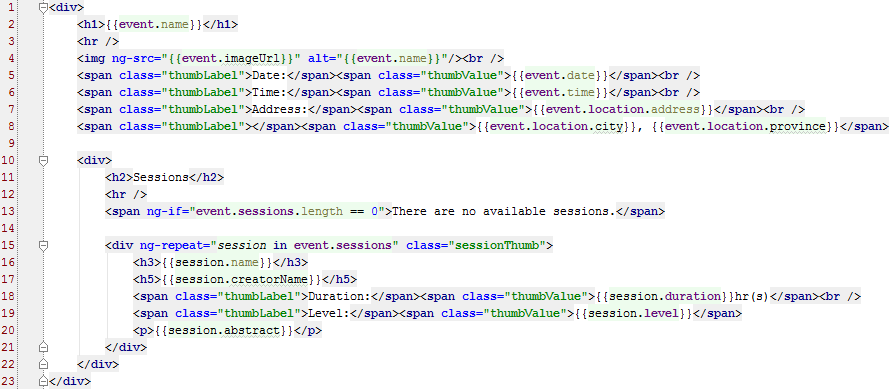
Na slici je prikazan primer parametrizovane rute. Nazivi parametara u ruti počinju dvotačkom. Parametri se mogu shvatiti kao promenljive čija vrednost je dostupna u kontroleru. Sa slike se vidi da je za navedenu rutu potrebno generisati kontroler *EventDetailsCtrl* i pogled *eventDetails.html*.



Slika 17 – Korišćenje *$routeParams* servisa

Definicija kontrolera data je na slici 17 i on zavisi od servisa *EventDataService*, ali i od *AngularJS* servisa *$routeParams*. Servis *$routeParams* je zapravo enkapsulira sve parametre rute u jedan objekat. Parametrima se pristupa tzv. tačka notacijom (kao na slici). Na ovaj način se vrši ekstrakcija parametra iz rute, pa se na osnovu njega može izvršiti odgovarajuća akcija. U ovom slučaju to je dohvatanje događaja koji ima vrednost ID polja jednaku parametru rute, a za to se koristi metoda *getEventById* servisa *EventDataService*. Ova metoda pretražuje sve događaje iz niza *eventsArray* i traži onaj koji ima ID jednak prosleđenom parametru. Zbog jednostavnosti implementacije, ova metoda neće biti dalje razmatrana. Pristigli podaci (model) se prosleđuju *$scope* objektu, pa postaju dostupni u HTML-u.

Pogled za prikazivanje događaja dat je na slici 18. Na početku se prikazuju osnovni podaci (linije 4-8), dok se u linijama 10-23 nalazi sekcija sa sesijama koje se prikazuju pomoću *ngRepeat* direktive. Jedan od detalja koje treba napomenuti jeste upotreba direktive *ngSrc* unutar *img* elementa (linija 4) koja služi da zameni standardni *src* atribut. U slučaju da se navede *src="{{event.imageUrl}}"*, prilikom učitavanja HTML-a može se desiti da internet pretraživač pokuša da dohvati sliku sa navedene adrese pre nego što *Angular* uspe da izračuna jednolinijski izraz. Tada će se javiti greška, jer navedeni izraz nije validna adresa slike. Primenom *ngSrc* direktive izbegava se ova neželjena pojava. Druga zanimljiva direktiva koja se ovde koristi je *ngIf* (linija 13). Ovom direktivom se naglašava da HTML element kome je pridružena treba prikazati samo ako je ispunjen prosleđeni logički izraz. U ovom slučaju će biti ispisana poruka ukoliko događaj nema sesije.



Slika 18 – Pogled detaljnog prikaza događaja

## Filteri

Filteri su posebne komponente *AngularJS*-a koje služe za dodatno formatiranje podataka u modelu pre nego što budu prikazani korisniku. Nad podacima se može primeniti veći broj filtera, tj. moguće je nadovezivanje filtera, a sintaksa je sledeća:

*{{ izraz | filter1:param1:param2 |filter2 | … }}*.

Filter može imati parametre koji se navode nakon imena filtera, a razdvajaju se dvotačkom. Postoji značajan broj već definisanih filtera, a moguće je definisati i sopstvene filtere.

Na primer, filter *currency* služi za formatiranje broja dodajući mu valutu. Tako će rezultat izraza *{{ 25 | currency }}* biti *$25.00*. Filter *number* služi za formatiranje brojeva na određeni broj decimala koji se prosleđuje kao parametar filtera. Rezultat izraza *{{1234 |number:2 }}* će biti *1,234.00*. Detaljnije o filterima u *Angular*-u može se naći u [xxx], a u nastavku će biti prikazano korišćenje filtera pod nazivom *filter*.

Kako mu i samo ime govori, filter *filter* služi za filtriranje niza objekata prema zadatom kriterijumu. U *EventApp* aplikaciji, ovaj filter je iskorišćen za pretragu sesija. Pretpostavka je da jedan događaj može imati veći broj sesija, pa se na ovaj način korisniku omogućuje da se unose kriterijum pretrage. Korisniku će biti prikazane samo one sesije koje zadovoljavaju uneti kriterijum.

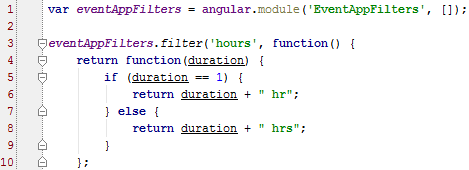
Za unos kriterijuma potrebno je dodati u pogled jedno tekstualno polje. Ono se, na primer, može naći između 13. i 15. linije pogleda (slika 18). Na slici 19 dat je izmenjeni deo pogleda sa slike 18 (linije 14 i 14) kako bi filter *filter* bio primenjen. U tekstualno polje se unosi tekst na koji referiše promenljiva *query* (definisana unutar atributa *ng-model*). Istog trenutka uneti tekst postaje dostupan filteru kao argument. Filter filtrira kolekciju sesija *event.sessions* i menja je prema kriterijumu. Pošto se kolekcija promenila, to inicira ponovno izvršavanje *ngRepeat* direktive i prikazivanje samo onih sesija u kojim se bilo gde pojavljuje uneti tekst.



Slika 19 – Primena filtera *filter*

Može se uvesti restrikcija da se filtriranje vrši po posebnom polju sesije, npr po apstraktu, i to na sledeći način: *filter:{ abstract : query }*. Tada će se prikazati samo one sesije kod kojih se unutar polja *abstract* pojavljuje uneti tekst.

Pored pomenutog filtera, u *EventApp* aplikaciji se koristi i posebno definisani filter koji je nazvan *hours*. To je jednostavan filter koji služi da na trajanje sesije nadoveže *hr* ili *hrs* u zavisnosti od toga da li je trajanje sesije 1 (jedan sat) ili neka druga vrednost. Do sada je nakon broja sati bio fiksiran zapis *hr(s)* (slika 18, linija 18).



Slika 20 – Definisanje filtera

Kreiranje sopstvenih filtera je prilično jednostavno i svodi se na implementaciju jedne funkcije. Na slici 20 prikazan je sadržaj dokumenta *filters.js* koji sadrži korisnički definisane filtere. Najpre se definiše poseban modul aplikacije *EventAppFilters* koji obuhvata korisnički definisane filtere (linija 1). Definisanje filtera vrši se pozivom funkcije *filter* (linija 3). Prvi parametar *filter* funkcije je naziv filtera, a drugi parametar je funkcija koja kreira filter, tj. njena povratna vrednost je filter. Pošto je filter funkcija , povratna vrednost je funkcija. Filter ima ulaznu vrednost, koja mu se prosleđuje kao parametar, a izlaz je obrađena ulazna vrednost. U ovom slučaju, ukoliko je ulaz jednak 1, onda na tu vrednost treba 'nalepiti' string '*hr*'. U suprotnom, na ulaznu vrednost treba nalepiti '*hrs*'. Primena ovog filtera je jednostavna: *{{ session.duration | hours }}*.

## Forma i njena validacija

HTML poseduje nekoliko elemenata koji služe za unos podataka. To su *input*, *select* i *textarea*. HTML forma (*form*) je element koji služi da objedini elemente unosa. *Angular* je posvetio posebnu pažnju formi, s obzirom da je to jedan od najčešćih elemenata za interakciju sa korisnicima.

U *EventApp* aplikaciji forma se upotrebljava za kreiranje događaja, tj. nalazi se u dokumentu *createEvent.html* (slika 21). Kao što se može primetiti, posmatrana forma veoma podseća na klasičnu HTML formu. U nastavku će biti razmatrane razlike u odnosu na klasičnu formu.

Prvo što se primećuje je prisustvo atributa *ng-model* na svim *input* elementima. Ovi atributi imaju ulogu u prikupljanju unetih podataka koji formiraju objekat *event*. Takođe, polja koja su obaveza poseduju validacionu diretivu *ngRequired* (tj. atribut *ng-required*) koja služi za validaciju podataka. Za zadavanje obrasca regularnim izrazom koristi se validaciona direktiva *ngPattern*. U liniji 8 naveden je regularni izraz koji odgovara korektnom datumu.

*Angular* tim se potrudio da rad sa formama dodatno unapredi. Naime, za svaku formu se kreira objekat koji ima naziv jednak *name* atributu i ovaj objekat je sastavni deo *$scope* objekta. U ovom primeru to je objekat *createEventForm*. Njemu se unutar kontrolera pristupa sa *$scope.createEventForm*, a unutar pogleda (HTML-a) sa *createEventForm*. Takođe, za svako polje unosa se kreira objekat sa njegovim nazivom, i taj objekat je polje *createEventObjekta*. Na primer, objektu *time*, koji se kreira za polje za unos vremena (linije 10 i 11), može se pristupiti sa (*$scope.*)*createEventForm.time*.



Slika 21 – Sadržaj dokumenta *createEvent.html*

Postavlja se pitanje čemu sva ta hijerarhija objekata? Ovi objekti poseduju polja u kojima sadrže određene podatke o stanju elementa kome su pridruženi. Neka od tih polja su:

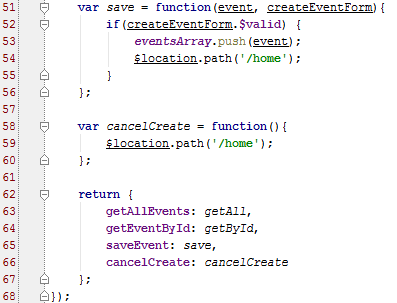
* *$valid*,
* *$invalid*,
* *$pristine*,
* *$dirty*.

Ukoliko je u element unosa unet sadržaj koji ne odgovara navedenoj validaciji (u validacionim direktivama), onda će polje *$invalid* biti postavljeno na *true*, a *$valid* na *false*. Navedena polja imaju suprotne vrednosti ako je unos validan. Polje *$pristine* ima vrednost *true* ako je sadržaj *input* elementa nepromenjen, u suprotnom ima vrednost *false*. Slično, polje *$dirty* ima vrednost *true* ako je sadržaj *input* elementa promenjen, u suprotnom ima vrednost *false*.

Ako je neki od elemenata unosa nevalidan, onda je i cela forma nevalidna. Ova osobina je iskorišćena u primeru sa slike 21. U liniji 28 je navedena direktiva *ngDisabled* koja služi da onesposobi (*disable*) HTML element. Njome se onemogućuje klik na dugme *Save* ukoliko je forma nevalidna.

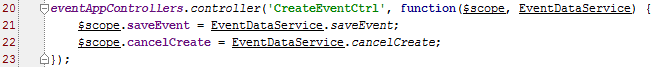
Sada se nameće pitanje na koji način obavestiti korisnika gde je pogrešio. *Angular* postavlja određene CSS klase na element u zavisnosti od njegovog stanja. Tako će nevalidan element imati klasu .*ng-invalid*. Sada je potrebno samo definisati navedenu klasu na željeni način. U radu je ova klasa implementirana tako da postavlja crvenu ivicu (*border*) na nevalidan element. Pored ove klase, *Angular* obezbeđuje dodatne klase kao što su *.ng-valid*, *.ng-invalid*, .*ng-pristine* i *.ng-dirty* koje se postavljaju na HTML elemente unosa u zavisnosti od stanja elementa, pa je moguće dodatno stilizovanje.

Ostalo je još definisati akcije čuvanja i poništavanja unetih podataka. Funkcije za obradu klika se navode kao vrednosti *ngClick* direktive. Ova direktiva služi da zameni atribut *onclick*, a postoje i direktive sa sličnim imenom za zamenu ostalih atributa za obradu akcija korisnika. Neke od njih su *ng-dbl-click*, *ng-mousedown*, *ng-change*, itd. U liniji 29 navedena je funkcija *saveEvent* koja će se izvršiti klikom na dugme *Save*. Ova funkcija prihvata dva parametra – prvi je objekat *event* koji sadrži unete podatke, a drugi je objekat *createEventForm* koji sadrži podatke o formi. U liniji 30, klikom na dugme *Cancel*, izvršava se funkcija *cancelCreate*. Ove funkcije su definisane u servisu *EventDataService*, a veza sa ovim servisom ostvarena je preko kontrolera *CreateEventCtrl*.



Slika 22 - Deo servisa *EventDataService*

Slika 22 prikazuje deo servisa gde su implementirane funkcije *save* i *cancelCreate*. Funkcija *save* prvo ispituje da li je prosleđena forma validna, i ako jeste, dodaje prosleđeni događaj u niz svih događaja. Nakon toga se pomoću *$location* servisa vrši redirekcija na početnu stranu. Funkcija *cancelCreate* je implementirana tako da redirektuje korisnika na početnu stranu sa svim događajima. Navedene funkcije su izvan servisa vidljive kao *saveEvent* i *cancelEvent*.



Slika 23 – Definicija kontrolera *CreateEventCtrl*

Definicija kontrolera *CreateEventCtrl* je data na slici 23. Ovde se vrši povezivanje funkcija iz pogleda sa odgovarajućim funkcijama iz servisa.

## Direktive