SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3512

APLIKACIJA ZA UPRAVLJANJE TEČAJEVIMA PRILAGOĐENA MOBILNIM UREĐAJIMA

Martin Skec

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3512

APLIKACIJA ZA UPRAVLJANJE TEČAJEVIMA PRILAGOĐENA MOBILNIM UREĐAJIMA

Martin Skec

Zagreb, lipanj 2014.

Izvornik (Skeniraj izvornik)

Prazna stranica ili stranica sa zahvalom

Sadržaj

Uvod

Aplikacija je svaki računalni program koji koristi računalo za izvršavanje korisnih zadataka, a koji ne spadaju pod zadatke vezane uz sam rad računala. Da bi se aplikacija mogla prikazivati na mobilnom uređaju, ona mora biti napisana specifično za operacijski sustav na tom uređaju koristeći programske jezike koje on zna izvoditi. S obzirom na to da gotovo svaki mobilni operacijski sustav koristi druge jezike nije dovoljno napisati jednu aplikaciju. Međutim, gotovo svi moderni mobilni operacijski sustavi imaju internetski preglednik. Preglednici znaju interpretirati sljedeće programske jezike: HTML (*HyperText Markup Language*) [2], CSS (*Cascade Style Sheets)* [2] i skriptni jezik JavaScript [3].

Web aplikacija je bilo koja aplikacija koja koristi internetski preglednik kao klijenta putem kojeg se ona prikazuje krajnjem korisniku. U pozadini je poslužitelj koji odgovara i poslužuje sve korisničke zahtjeve. Poslužitelj se ne nalazi na istom računalu/mobitelu s kojeg korisnik pristupa aplikaciji, već se on nalazi na nekom udaljenom računalu i komunicira putem neke vrste mreže.

Web aplikacije su sve popularnije zbog raširenosti internetskih preglednika, mogućnost dorade i održavanja aplikacije bez potrebe za redistribucijom i instalacijom nove verzije. Još jedan razlog popularnosti je dovršetak HTML5 standarda koji u potpunosti uklanja potrebu za dodacima za preglednik koje je u prošlosti korisnik trebao instalirati kako bi mogao pregledavati video i audio zapise.

Web aplikacije su obično podijeljene u logičke dijelove ili slojeve. Najčešće se koristi troslojna arhitektura koja se sastoji od podatkovnog, poslovnog i prezentacijskog sloja. U podatkovnom sloju se nalazi model podataka aplikacije. On predstavlja najniži sloj u troslojnoj arhitekturi. Iznad njega se nalazi poslovni sloj u kojem se nalazi logika aplikacija, a iznad poslovnog sloja je prezentacijski sloj. Komunikacija se odvija isključivo između susjednih slojeva, tako prezentacijski sloj ne bi trebao komunicirati s podatkovnim slojem. Za trajno pohranjivanje podataka se koristi baza podataka i s njom komunicira podatkovni sloj. Prezentacijski sloj se obično dijeli na tri dijela. Model, pogled i kontroler (engl. *Model-View-Controller*). Kontroler obrađuje korisničke zahtjeve i kontrolira koji pogled i koji podatak će se prikazati korisniku.

Jednostranične web aplikacije (engl. *Single-page* application) su posebne web aplikacije koje na prvi zahtjev učitaju sav potreban HTML, JavaScript i CSS. Postoje slučajevi jednostraničnih aplikacija u kojima se na prvi zahtjev učitava samo potreban kod, a ostatak se učitava u pozadini po potrebi, bez potrebe za osvježavanjem ili prenošenjem kontrole na drugi URL (engl. *Uniform resource locator*). Dinamično dodavanje i brisanje HTML elemenata unutar DOM-a (engl. *Document object model*) ne bi bilo moguće bez JavaScript-a. Manipulacija DOM elementima koristeći isključivo DOM api (engl. *Application programming interface*) zahtjeva puno posla i nije jednostavna. Tu dolaze JavaScript radni okviri (engl. *Frameworks*) koji olakšavaju manipulaciju i ubrzavaju razvoj web aplikacija.

1. Funkcijski zahtjevi

Aplikacija za upravljanje tečajevima treba administratoru pružiti mogućnost unosa novih definicija tečajeva. Osim unosa treba omogućiti uređivanje i brisanje tečajeva. Kada je tečaj definiran treba se omogućiti definiranje termina održavanja.

Korisnici trebaju moći iskazati interes ili napraviti prijavu na tečaj. Sama prijava ne mora značiti da će korisnik doista biti na tečaju. Administrator može pregledavati svaku prijavu i odlučiti hoće li ga prihvatiti na tečaj ili ne. O odluci i razlogu se treba obavijestiti korisnika putem obavijesti unutar aplikacije i putem elektroničke pošte.

Administratoru omogućiti diseminaciju materijala za neki termin tečaja. Svakom prijavljenom korisniku poslati obavijest o dostupnosti novog materijala.

Korisnik u svom profilu treba imati uvid u tečajeve na koje je prijavljen i u tečajeve na koje je prihvaćen. Omogućiti mu uvid u materijale za tečajeve na koje je prihvaćen.

Da bi korisnik mogao imati uvid u svoj profil, primao obavijesti i da bi mu se omogućila prijava na tečajeve potrebno je omogućiti registraciju korisnika. Administrator ne treba posebno aktivirati korisnike.

Radi lakšeg pronalaženja tečajeva potrebno je omogućiti pretraživanje. Pretraživanje se vrši na temelju raspoloživih podataka o tečaju kao što su: ključne riječi, naziv, kategorija i riječi iz opisa tečaja.

Kada je tečaj završen potrebno je omogućiti polaznicima ocjenjivanje tečaja.

2. Pregled korištenih tehnologija

Razvoj web aplikacije koristeći isključivo programske jezike bez dodatnih biblioteka i radnih okvira nije dobar način razvoja aplikacije. Razlog tome je što takav način oduzima previše vremena na rutinske stvari koje nemaju veze sa specifičnim problemom kojeg aplikacija želi riješiti. Postoje dva tipa radnih okvira koji se koriste u razvoju web aplikacija. Prvi je konfiguracija prije konvencije u kojem radni okvir pruža maksimalnu konfigurabilnost i postoje neke postavke koje se moraju konfigurirati. Drugi tip radnih okvira je konvencija prije konfiguracije u kojem se koriste dobre prakse kako bi se većina konfiguracije sakrila od programera. Dobar radni okvir bi trebao biti dovoljno konfigurabilan kako bi se mogao prilagoditi specifičnom problemu kojeg se rješava.

Ovaj rad izrađen je koristeći radni okvir *Grails* [4] opisan u poglavlju 2.1 za poslužiteljsku stranu i radni okvir *AngularJS* [5] koji je opisan u poglavlju 2.2 za klijentsku stranu.

2.1. Grails

Grails je radni okvir otvorenog koda koji slijedi MVC paradigmu, te je stoga specijaliziran za web aplikacije. Također je „full stack“ radni okvir što znači da sadrži sve od baze do web servera, sve potrebno da bi se razvijala aplikacija. Koristi *Groovy* [6] programski jezik koji je baziran na *Java* platformi, odnosno koristi JVM (engl. *Java virtual machine*). Namijenjen je da bude visoko produktivan radni okvir tako da slijedi paradigmu „konvencija prije konfiguracije“ (engl. *Convention over Configuration*) i od programera skriva većinu konfiguracije. U isto vrijeme je dovoljno fleksibilan da ga programer može konfigurirati po svojim potrebama.

Grails je izgrađen nad *Spring MVC* [7] radnim okvirom. *Spring* je također radni okvir otvorenog koda. Pruža mogućnost velike nadogradivosti i slijedi paradigmu „konfiguracija prije konvencije“ (engl. *Configuration over Convention*), te stoga ima puno strmiju liniju učenja i nije prikladan za početnike.

2.1.1. Jednostavna Grails aplikacija

Vrlo je jednostavno započeti s razvojem koristeći *Grails*. Dovoljno je skinuti radni okvir, postaviti *grails* naredbu u sistemske varijable i pomoću jedne linije u konzoli se stvara nova aplikacija.

```
grails create-app MyAppName
```

Ovako stvorena aplikacija ima sve potrebne direktorije i datoteke i spremna je za pokretanje naredbom `grails run-app`. Ova naredba pokreće web server i aplikacija postaje dostupna na adresi `http://localhost:8080/MyAppName`.

Sljedeći korak u razvoju je napraviti domenski entitet. Napravit ćemo entitet za osobu sljedećom naredbom: `grails create-domain-class Person`. Sada u direktoriju `grails-app/domain` postoji naš domenski entitet `Person` kojem možemo početi definirati atribute. U odsječku ispod je primjer sa dva atributa ime i prezime.

```
class Person {
    String firstName
    String lastName
}
```

Jednom kada imamo domenski entitet možemo definirati kontroler naredbom `grails create-controller Person`. Stavit ćemo da ima jednu metodu koja vraća sve osobe u *JSON* (*JavaScript Object Notation*) formatu.

```
class PersonController {
    def list() {
        render Person.all as JSON
    }
}
```

Sada ako odemo na `http://localhost:8080/MyAppName/person/list` dobivamo ispis svih osoba u sustavu u JSON formatu.

2.1.2. GORM (Grails object relational mapping)

Grails object relational mapping ili skraćeno *GORM* je *Grails*-ova implementacija *ORM*-a (Object relational mapping). *GORM* je izgrađen nad vrlo popularnim *Hibernate 3* ORM-u i zahtijeva puno manje konfiguracije prilikom stvaranja domenskih entiteta.

Dodavanje entiteta je vrlo jednostavno i intuitivno. Sve što je potrebno je stvoriti novi primjerak našeg domenskog entiteta i pozvati metodu `save()`.

```
new Person(firstName: 'Pero', lastName: 'Perić').save()
```

Dohvaćanje podataka se obavlja pozivom metode `get(id)` koja prima jedan argument, a to je primarni ključ entiteta.

```
Person person = Person.get(1)
```

Brisanje se obavlja naredbom: `person.delete()`.

Zahvaljujući dinamičkim svojstvima *Groovy* programskog jezika, *GORM* nudi mogućnost stvaranja dinamičkih upita. Takve statičke metode po načinu poziva, a dinamičke po nazivu nisu nigdje definirane unutar naše aplikacije. One se dinamički generiraju za vrijeme izvođenja na temelju atributa našeg domenskog entiteta nad kojim se pozivaju.

Prototip dinamičke metode je prikazan u nastavku.

`Person.findBy([Property][Comparator][Boolean Operator])?[Property][Comparator].`

Atribut (engl. *Property*) je atribut iz našeg domenskog entiteta. Komparator (engl. *Comparator*) je jedan od preddefiniranih komparatora. Definirani komparatori su: `InList`, `LessThan`, `LessThanEquals`, `GreaterThan`, `GreaterThanEquals`, `Like`, `Ilike`, `NotEqual`, `InRange`, `Rlike`, `Between`, `IsNotNull`, `IsNull`. Logički operatori (engl. *Boolean operators*) mogu biti `And` i `Or`. Argumenti metode ovise o komparatoru i mogu biti istog tipa kao atributa, kolekcija objekata istog tipa kao atribut, a može biti i bez argumenta.

Primjer dinamičkog upita nad entitetom `Person` u kojem se traži prvi entitet sa traženim imenom i prezimenom se nalazi u odsječku ispod.

```
Person.findByFirstNameAndLastName('Pero', 'Perić')
```

Osim `findBy` koji traži prvi entitet koji odgovara upitu, može se koristiti i `findAllBy` koji vraća sve entitete koji odgovaraju upitu.

2.1.3. Dodatak Spring Security

Spring Security [8] dodatak (engl. *plugin*) je integracija popularnog *Java* radnog okvira *Spring Security* iz *Spring* radnog okvira u *Grails*. Koristimo ga kako bi se aplikaciji brzo i jednostavno dodala autentifikacija i autorizacija korisnika. Ovaj dodatak pruža veliku konfigurabilnost. Moguće je konfigurirati koji hash algoritam će se koristiti kod spremanja zaporki, koji naš domenski entitet predstavlja korisnika, na koje URL-ove će se mapirati prijava i odjava. Kako će se raditi autentifikacija, hoće li biti obična HTTP autentifikacija, digest autentifikacija ili pak autentifikacija putem certifikata.

Kao i *Grails* ništa od ove konfiguracije ne mora raditi programer ako mu ne treba i dodatak se jednostavno uključuje spreman za korištenje jednom naredbom.

```
grails s2-quickstart moj.paket User Role
```

2.2. AngularJS

AngularJS je radni okvir otvorenog koda namijenjen za izradu klijentske strane web aplikacije, odnosno za prezentaciju aplikacije. Razvija i održava ga *Google* zajedno sa zajednicom. *AngularJS* pomaže u razvoju jednostraničnih web aplikacija koje na klijentskoj strani koriste samo *HTML, CSS* i *JavaScript*. Cilj mu je nadograditi web aplikacije s MVC mogućnostima i tako olakšati razvoj i testiranje aplikacije.

AngularJS je izgrađen oko ideje da se koristi deklarativno programiranje (engl. *Declarative programming*) za izradu korisničkog sučelja i povezivanja komponenti. Deklarativno programiranje je programska paradigma u kojoj se program sastoji od malih elemenata koji nisu direktno povezani i koji čine strukturu programa bez eksplicitnog definiranja programskog toka.

Ciljevi koji su se pokušali ispuniti prilikom razvoja radnog okvira su:

- Razdvajanje manipulacije DOM elementima od aplikacijske logike. Ovaj cilj olakšava testiranje aplikacije.
- Testiranje aplikacije je jednako važno kao i njeno pisanje. Težina testiranja aplikacije jako ovisi o njenoj strukturi.
- Snažno odvajanje klijenta i poslužitelja na način da klijent od poslužitelja dobiva podatke isključivo putem zahjeva. To omogućava paralelan razvoj klijenta i poslužitelj, te se poslužitelj može promatrati kao servis koji nam daje i sprema podatke.

Jedna od glavnih značajki je dvostrano (engl. *two-way*) povezivanje podataka koje povezuje HTML poglede sa JavaScript objektima. Povezivanje se događa bez ikakve dodatne manipulacije DOM elementima ili upravljanja događajima, kao što to radi drugi popularni radni okvir *jQuery* [9]. Svaka promjena u pogledu će se odmah odraziti na JavaScript objekt i obrnuto.

AngularJS proširuje HTML tako da uvodi direktive koje dodaju određeno ponašanje DOM elementima ili čak u potpunosti zamjenjuju element s predloškom iz direktive. Angular dolazi sa skupom već ugrađenih direktivi kao što su: `ngApp`, `ngModel` i `ngView`.

`ngApp` je direktiva koja govori angularu da je DOM element koji ju sadrži korijen naše aplikacije. Obično se stavlja na HTML oznake `<html>` ili `<body>`.

`ngView` je direktiva koja se stavlja u HTML dokument koji predstavlja glavni raspored (engl. *layout*) aplikacije, obično je to `index.html`. Ona označava mjesto na koje će angular, kod svake promijene rute, staviti odgovarajući parcijalni pogled.

`ngModel` je vjerojatno najkorištenija direktiva, koristi se za povezivanje HTML oznaka `<input>`, `<select>`, `<textarea>` sa svojstvom na djelokrug (engl. *scope*) kontrolera.

Djelokrug, u angularu se označava sa `$scope`, je objekt koji predstavlja kontekst kontrolera i predstavlja vezu između kontrolera i pogleda. Svi izrazi u HTML-u se razrješavaju na djelokrug kontrolera pod kojim oni spadaju. Djelokruzi su postavljeni u hijerarhijsku strukturu slično kao i DOM struktura. Vršni djelokrug se naziva `$rootScope` i iz njega se izvode svi ostali.

UI Bootstrap [10] je implementacija *Bootstrap*-ovih komponenti tako da koriste *Angular*-ove direktive, te se tako odlično uklapaju u aplikaciju razvijenu korištenjem *AngularJS* radnog okvira. Više o *Bootstrap*-u u poglavlju 2.3.

2.2.1. Jednostavna AngularJS aplikacija

Tipična AngularJS aplikacija se sastoji od više komponenata. Komponente su JavaScript skripte i parcijalni pogledi. Skripte se sastoje obično od jedne glavne skripte u kojoj se definira angular modul koji predstavlja našu aplikaciju, mapiraju rute na poglede i kontrolere i sluša na neke događaje koji se tiču cijele aplikacije. Na primjer, uspješna prijava ili odjava korisnika.

Kontroleri, direktive, filtri i servisi se definiraju u zasebnim skriptama i obično se za svaku ovu komponentu definira zaseban angular modul.

Parcijalni pogledi se obično stavljaju u zaseban direktorij `partials`. Pogledi su HTML datoteke koje ne sadrže potpuni HTML dokument s oznakama `<html>`, `<head>` i `<body>`. Oni su samo predlošci koji će biti „ubačeni“ u glavnu datoteku na mjesto na kojem je definirana direktiva `ngView`. Svaki parcijalni pogled je

mapiran na kontroler, vezom 1 na 1, i djelokrug angular izraza u pogledu odgovara djelokrugu kontrolera.

Glavni ulaz u aplikaciju je u jednoj HTML datoteci koja učitava sve potrebne stilove, biblioteke, radne okvire i našu aplikaciju u obliku skripti, odnosno CSS datoteka. U ovom primjeru to je `index.html` datoteka, čiji je sadržaj prikazan na slici Slika 2.1 - Sadržaj index.html datoteke.

```
1  <html ng-app="MyApp">
2      <head>
3          <script src="angular.min.js"></script>
4          <script src="app.js"></script>
5      </head>
6      <body ng-controller="AppCtrl">
7          <h2>{{ title }}</h2>
8
9          <label for="title">Unesite naslov</label>
10         <input type="text" ng-model="title" id="title">
11
12         <h2>Boje</h2>
13         <label for="color">Nova boja</label>
14         <input type="text" ng-model="newColor" id="color">
15         <input type="submit" ng-click="addColor(newColor)" value="Dodaj">
16
17         <ul ng-repeat="color in colors">
18             <li>{{ color }}</li>
19         </ul>
20     </body>
21 </html>
```

Slika 2.1 - Sadržaj index.html datoteke

Na liniji *1* je definirana aplikacija direktivom `ngApp` kojoj je predan naziv aplikacije. U zaglavlju HTML dokumenta su referencirane skripte za radni okvir i skripta u kojoj se nalazi aplikacija. Direktivom `ngController` je definirano koji kontroler će biti zadužen za upravljanje izrazima unutar oznaka `<body>`.

Angular izrazi mogu biti definirani unutar dvostrukih vitičastih zagrada kao što je to definirano na liniji *7*. `{{ title }}` označava da se na to mjesto treba ispisati ono što je definirano u JavaScript varijabli `title` u djelokrugu trenutnog kontrolera. U oznaci `<input>` na liniji *7* je korištena direktiva `ngModel` koja direktno povezuje sadržaj polja za unos s varijablom u djelokrugu. Kada bi se u to polje unio tekst, on bi odmah bio prikazan na svim mjestima na kojima se ta varijabla koristi unutar pogleda.

Gumb „Dodaj“ ima definiranu direktivu `ngClick`. Ona će na pritisak gumba pozvati funkciju `addColor(newColor)` koja je definirana u trenutnom djelokrugu i kojoj se predaje varijabla `newColor` koja je također u djelokrugu.

HTML element `<ul>` na liniji *17* ima definiranu direktivu `ngRepeat`. Toj direktivi se predaje izraz u kojem mora biti JavaScript polje definirano na djelokrugu. Direktiva stvara po jedan HTML predložak za svaki element polja. Predloškom se smatraju svi elementi unutar `<ul>` oznaka. U ovom primjeru će se stvoriti četiri `<li>` oznake, po jedna za svaku boju definiranu u polju.

Sadržaj datoteke u kojoj je definirana aplikacija i kontroler prikazan je na slici Slika 2.2 - Sadržaj app.js datoteke. Na liniji *1* je definiran angular modul `MyApp` koji se sastoji od modula `MyApp.controllers`.

```
1   angular.module('MyApp', ['MyApp.controllers']);
2
3   angular.module('MyApp.controllers', [])
4       .controller('AppCtrl', ['$scope', function($scope) {
5           $scope.colors = ['Crvena', 'Plava', 'Zelena', 'Bijela'];
6
7           $scope.addColor = function(color) {
8               $scope.colors.push(color);
9           };
10      }]);
```

Slika 2.2 - Sadržaj app.js datoteke

U djelokrugu kontrolera `AppCtrl` je definirano polje `colors` i funkcija `addColor`. Kada pogled pozove funkciju i onda doda boju iz argumenta u polje u djelokrugu, automatski će se i na pogledu u listu u kojoj se ispisuju boje dodati novi HTML element `<li>` s novom bojom zahvaljujući angularovom dvostranom povezivanju podataka i direktivi `ngRepeat`.

2.3. Bootstrap

Bootstrap [11] je kolekcija alata za razvoj web stranica. Sadrži HTML i CSS predloške za izradu web formi, gumbi, navigacije i ostalih komponenti korisničkog sučelja. Također sadrži opcionalne *JavaScript* dodatke kao što su: tranzicije HTML elemenata, iskočne prozore, padajuće izbornike i poruke koje se pojave kada je kursor iznad ikone, teksta, slike, poveznice i ostalih elemenata korisničkog sučelja.

Bootstrap je razvijen od *Mark Otto*-a i *Jacob Thornton*-a, zaposlenika tvrtke *Twitter* kao radni okvir koji će uvesti konzistenciju u razvoj web stranica. Prije *Bootstrap*-a, korištene su razne biblioteke za izradu korisničkog sučelja, što je dovelo do nekonzistentnosti i visokog troška održavanja takvih aplikacija.

Od verzije 3.0 *Bootstrap* je prihvatio „*mobile first*“ filozofiju koja podržava responzivan dizajn bez potrebe za dodatnim naglašavanjem da je on responzivan.

Postao je jedan od najpopularnijih alata za razvoj web stranica i vrlo važan projekt otvorenog koda. Programski kôd je izdan pod *MIT* licencom, što znači da se slobodno može koristiti bez ikakvih ograničenja. Projekt održava tim koji ga je inicijalno razvio uz potporu zajednice.

Bootstrap je odličan alat kada razvijamo web aplikaciju i ne želimo se previše brinuti o samom izgledu aplikacije. *Bootstrap* je tu da se brine o tome za nas, a mi samo trebamo koristiti njegove stilove i *HTML* predloške. Ako nam se neki dio ne sviđa ili želimo da se neki dio drugačije prikazuje, samo trebamo pregaziti (engl. *Override*) *Bootstrap*-ovu definiciju toga i napisati svoju.

2.3.1. Način korištenja

Da bi se *Bootstrap* mogao koristiti u našoj *HTML* stranici potrebno je preuzeti *Bootstrap CSS* stil i uključiti ga u našu *HTML* datoteku. Ako se želi koristiti *Bootstrap*-ove *JavaScript* dodatke potrebno je dodatno referencirati i *jQuery* biblioteku, te potom *Bootstrap*-ovu *JavaScript* datoteku. U *HTML* odsječku ispod je primjer jednostavne *HTML* stranice s uključenim *Bootstrap*-om.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4     <title>Primjer korištenja Twitter Bootstrap-a</title>
5     <link href="bootstrap.css" rel="stylesheet">
6   </head>
7   <body>
8     <div class="container">
9       <h1>Moj sadržaj</h1>
10      <form>
11        <div class="form-group">
12          <label for="search">Unesite pojam pretrage</label>
13          <input type="text" class="form-control" id="search">
14        </div>
15        <button type="submit" class="btn btn-default">Pretraga</button>
16      </form>
17    </div>
18    <script src="jquery.min.js"></script>
19    <script src="bootstrap.min.js"></script>
20  </body>
21  </html>
```

U odsječku iznad na linije *5* je referenciran *Bootstrap*-ov stil. Na liniji *18* i *19* su referencirane *jQuery* i *Bootstrap JavaScript* biblioteke. U tijelu *HTML* dokumenta je definiran *div* element na liniji *8* ukrašen s *container* razredom koji je dio *Bootstrap*-ovog stila i koji centrira sadržaj koji se nalazi unutar njega.

Između linija *10* i *16*, definirana je jednostavna forma za pretragu koja koristi preddefinirane *Bootstrap*-ove stilove kako bi bila kvalitetno prikazana s odgovarajućim marginama i ispunjavanjima (engl. P*adding*).

3. Opis aplikacije

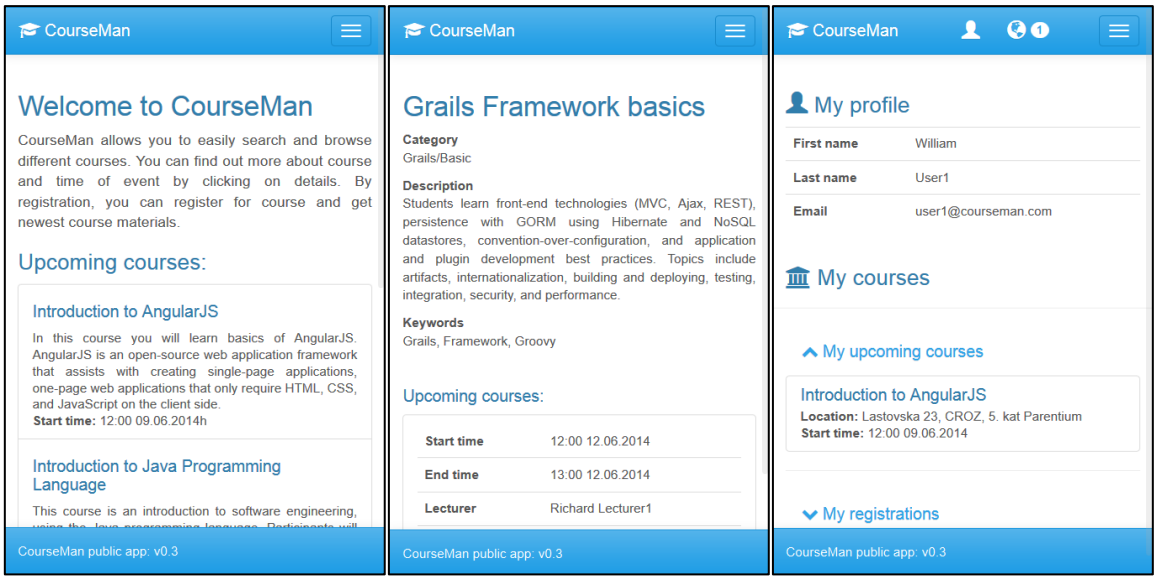
Aplikacija za upravljanje tečajevima sastoji se od korisničke aplikacije koja je prvenstveno namijenjena korisnicima mobilnih uređaja i administratorske aplikacije putem koje administrator upravlja svim segmentima sustava.

3.1. Opis korisničke aplikacije

Korisnička aplikacija je posebna web aplikacija responzivnog dizajna prilagođena mobilnim uređajima. Posebno je dizajnirana kako bi osjećaj korištenja bio što sličniji nativnim mobilnim aplikacijama. Mobilni uređaji nisu jedini koji mogu pristupiti aplikaciji i ona se jednako dobro prikazuje i na osobnim računalima.

Ideja korisničke aplikacije je omogućiti pretraživanje raznih tečajeva na jednostavan način. Korisnik ne treba biti prijavljen kako bi pretraživao tečajeve. Tečaj se pretražuje po nazivu, ključnim riječima i kategoriji. Ostale funkcionalnosti za koje nije potrebna registracija su pristup detaljnijim informacijama o tečaju, te informacija o vremenu održavanja pojedinog tečaja. Jedan tečaj može imati više termina održavanja i korisnici mogu odabrati njima najpovoljniji.

Kada se korisnik odluči za registraciju u sustav, tada dobiva mogućnost prijave na tečaj. Sama prijava ne znači nužno da mjesta na tečaju ima i da će korisnik biti prihvaćen na njega. Da li je prihvaćen ili ne određuje administrator na temelju broja trenutno popunjenih mjesta ili nekom drugom kriteriju i on svakog korisnika ili prihvaća ili odbija. Prilikom prijave je moguće ostaviti komentar kojeg će administrator vidjeti prilikom odluke. Korisnik dobiva obavijest o njegovoj odluci. Jednom kada je korisnik prihvaćen na tečaj, on dobiva sve informacije o statusu i obavijesti o novim materijalima vezanim uz točno taj termin kojeg je prijavio.



Slika 3.1 – Izgled korisničke aplikacije: početni ekran, detalji tečaja i profil

Sa završetkom tečaja svi korisnici koji su sudjelovali na njemu dobivaju obavijest i otvara im se mogućnost da ocijene tečaj. Tada mogu vidjeti prosječnu ocjenu korisnika za svaki termin održavanja tečaja, a ta ocjena se koristi kako bi se izračunala prosječna ocjena svih termina određenog tečaja.

3.2. Opis administratorse aplikacije

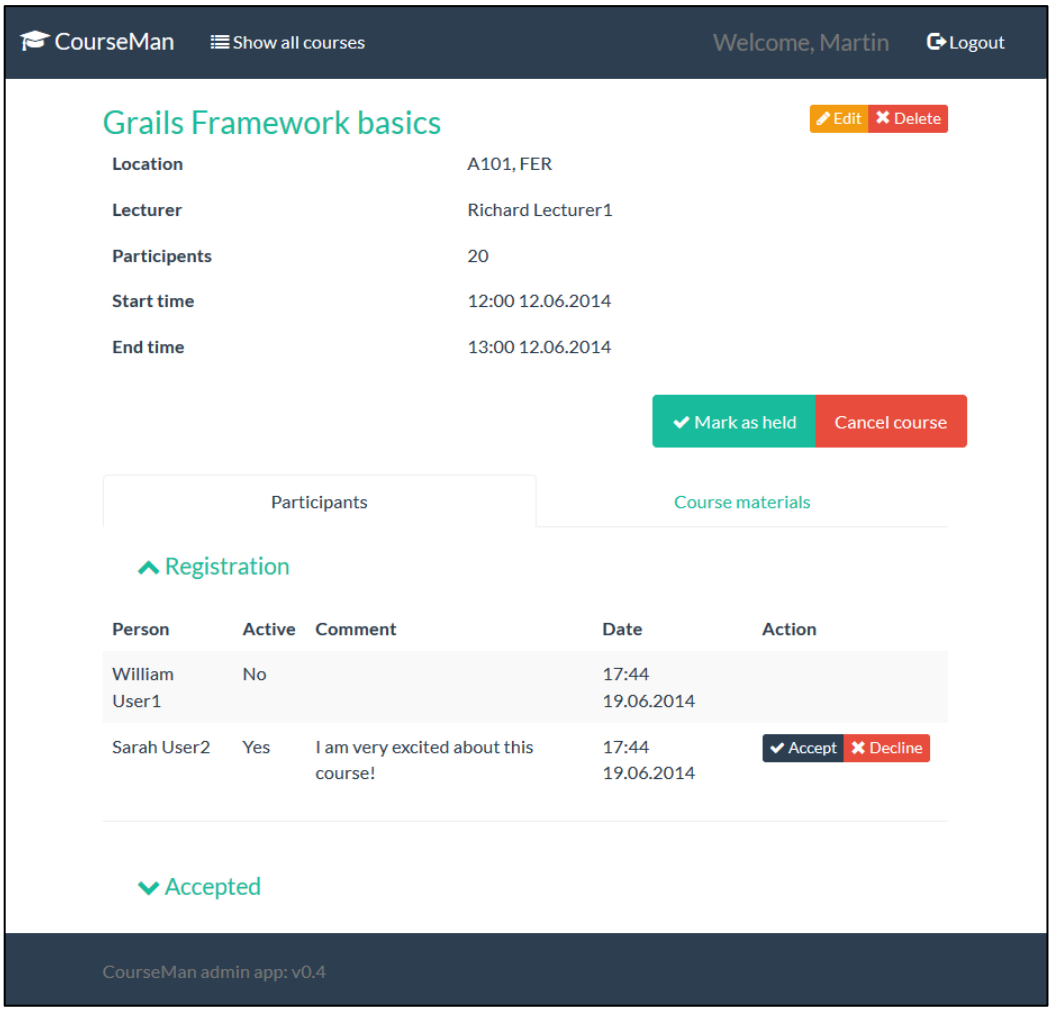
Administratorska aplikacija nije dostupna svima i pristup se ostvaruje prijavom s odgovarajućim korisničkim računom. Nije dovoljna sama registracija, već je potrebno imati i posebno pravo za pristup.

Svrha administratorske aplikacije je omogućiti ovlašenim osobama upravljanje tečajevima. Upravljanje tečajem se ponajprije odnosi na definiranje tečaja. Pri definiranju novog tečaja potrebno je obavezno navesti ime tečaja, kratak opis i kategoriju u koju tečaj spada. Opcionalno, definiraja se detaljan opis tečaja i ključne riječi po kojima se vrši pretraživanje. Jednom kada je tečaj definiran, on se pojavljuje u popisu svih tečajeva i administrator tada može dodavati termine u kojima se pojedini tečaj održava.

3.2.1. Termini tečaja

Definiranje termina sastoji se od unosa lokacije, vremena početka i maksimalnog broja pristupnika. Opcionalno, unosi se vrijeme završetka i predavač, koji mora biti korisnik sustava s odgovarajućom dozvolom.

Kada se završni unos termin dobiva status „nadolazeći“ i korisnici dobivaju mogućnost prijave na tečaj. Kod svake prijave na tečaj administrator dobiva obavijesti o tome. Jednom kada je korisnik prijavljen, administrator donosi odluku hoće li ga prihvatiti ili ne. Na slici Slika 3.2 – Ekran za upravljanje terminom tečaja je prikazano kako to izgleda u aplikaciji. Potom korisnik dolazi na listu prihvaćenih, odnosno odbijenih. Korisniku se šalje obavijest o odluci putem sustava i putem elektroničke pošte na adresu koju je korisnik unio prilikom registracije. Svaka prijava ostaje u sustavu i time se dobiva povijesni pregled.



Slika 3.2 – Ekran za upravljanje terminom tečaja

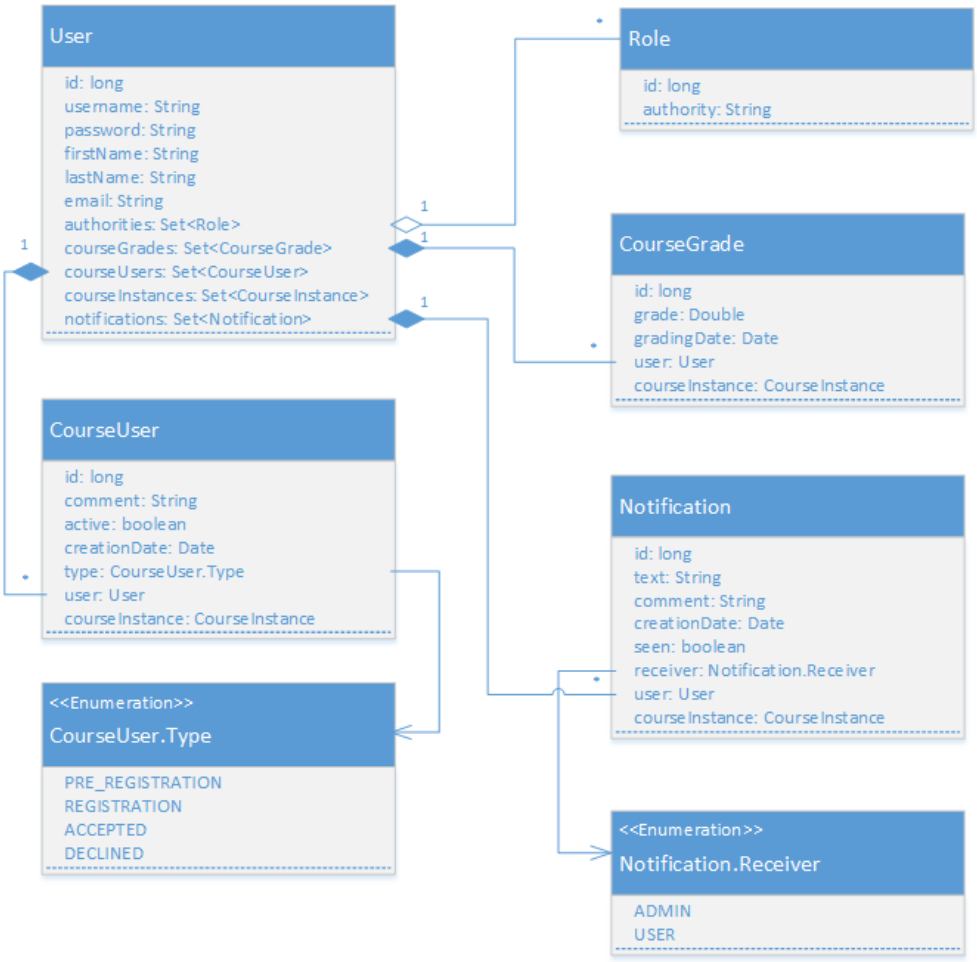
Druge funkcionalnosti vezane uz termin tečaja su unos materijala koji će se koristiti na tečaju. Ti materijali mogu biti prezentacije, neke upute vezane uz tečaj ili bilo što što bi korisnicima koristilo prilikom samog održavanja.

Administrator može u svakom trenutku promijeniti status tečaja. Mogući statusi su „otkazan“ i „održan“. Status „otkazan“ administrator postavlja ukoliko nema dovoljno zainteresiranih korisnika ili zbog nekih drugih razloga. Nakon što se tečaj održi administrator mijenja status u „održan“. Prilikom promijene statusa šalje se odgovarajuća obavijest svim.

4. Opis podatkovnog modela aplikacije

Podatkovni model aplikacije izgrađen je oko dva glavna entiteta. To su entitet *User* koji predstavlja svakog korisnika aplikacije uključujući i administratora i entitet *CourseInstance* koji predstavlja jedan termin održavanja tečaja. Gotovo svi drugi entiteti imaju neki tip veze na jedan od ova dva. Svaki entitet ima primarni ključ naziva *id*. Model je izgrađen na način da su definirani domenski entiteti u programskom jeziku *Groovy*. Iz njega je radni okvir *Grails,* kojeg sam odabrao za izradu aplikacije, stvorio relacijsku bazu podataka. Na slikama Slika 4.1 - Dijagram razreda oko entiteta *User* i Slika 4.2 - Dijagram razreda oko entiteta *CourseInstance* se nalaze UML dijagrami razreda izgrađenog modela podataka.

4.1. Entitet *User* i vezani entiteti



Slika 4.1 - Dijagram razreda oko entiteta *User*

User:

Entitet *User* predstavlja svakog korisnika aplikacije bilo onog koji se želi samo prijaviti na tečaj bilo administratora koji upravlja sustavom. Glavni atributi ovog entiteta su: korisničko ime (atr. *username*), zaporka (atr. *password*) koja je hash-irana, ime (atr. *firstName*), prezime (atr. *lastName*) i adresa elektroničke pošte (atr. *email*). Svaki korisnik ima skup *ovlasti* (atr. *authorities*) predstavljene entitetom *Role* na temelju kojih se određuje što korisnik smije napraviti, a što ne smije. Ostali entiteti na koje *User* ima veze su: *Notification* (atr. *notifications*), *CourseGrade* (atr. *courseGrades*), *CourseUser* (atr. *courseUsers*) i *CourseInstance* (atr. *courseInstances*).

Role:

Entitet *Role* predstavlja jednu ovlast u sustavu. Glavni atribut je sam naziv ovlasti (atr. *authority*). Entitet nema veze na druge entitete. Primjer naziva ovlasti iz aplikacije su „*ROLE_USER*“ za korisnika, „*ROLE_LECTURER*“ za predavača i „*ROLE_ADMIN*“ za administratora. Korisnik pri registraciji automatski dobiva ovlasti za korisnika.

CourseGrade:

CourseGrade predstavlja ocjenu jednog termina tečaja od strane jednog korisnika. Sastoji se od atributa ocjena (atr. *grade*) koja može biti u rasponu od *0* do *5* sa zaokruživanjem na dvije decimale i datuma ocjene (atr. *gradingDate*) koji predstavlja vremenski trag ocjenjivanja. Kako bi se znalo koji korisnik je dao ocjenu i pazilo da jedan korisnik može dati samo jednu ocjenu po terminu tečaja entitet ima vezu na *User* i *CourseInstance*. Taj par veza mora biti jedinstven za svaki *CourseGrade*.

Notification:

Ovaj entitet služi za sve obavijesti za korisnike u sustavu. Atribut tekst (atr. *text*) predstavlja glavni tekst obavijesti koji je obično preddefiniran. Komentar (atr. *comment*) je dio obavijesti koji služi da bi onaj tko šalje obavijest mogao ostaviti dodatnu kratku poruku za primatelja. Ostali atributi su: datum slanja obavijest (atr. *creationDate*), zastavica pomoću koje saznajemo da li je obavijest viđena (atr. *seen*) i enumeracija *Notification.Receiver* (atr. *receiver*) putem koje razlikujemo obavijesti za korisnika i administratora. Kako bi znali za kojeg korisnika i za koji termin tečaja je obavijest vezana, entitet sadrži veze na *User* i *CourseInstance* koja ne mora biti jedinstvena na razini obavijesti.

Notification.Receiver:

Enumeracija s kojom saznajemo za koga je namijenjena obavijest. Sastoji se od dvije vrijednosti: „*ADMIN*“ ako ju treba vidjeti administrator i „*USER*“ ako ju treba vidjeti korisnik.

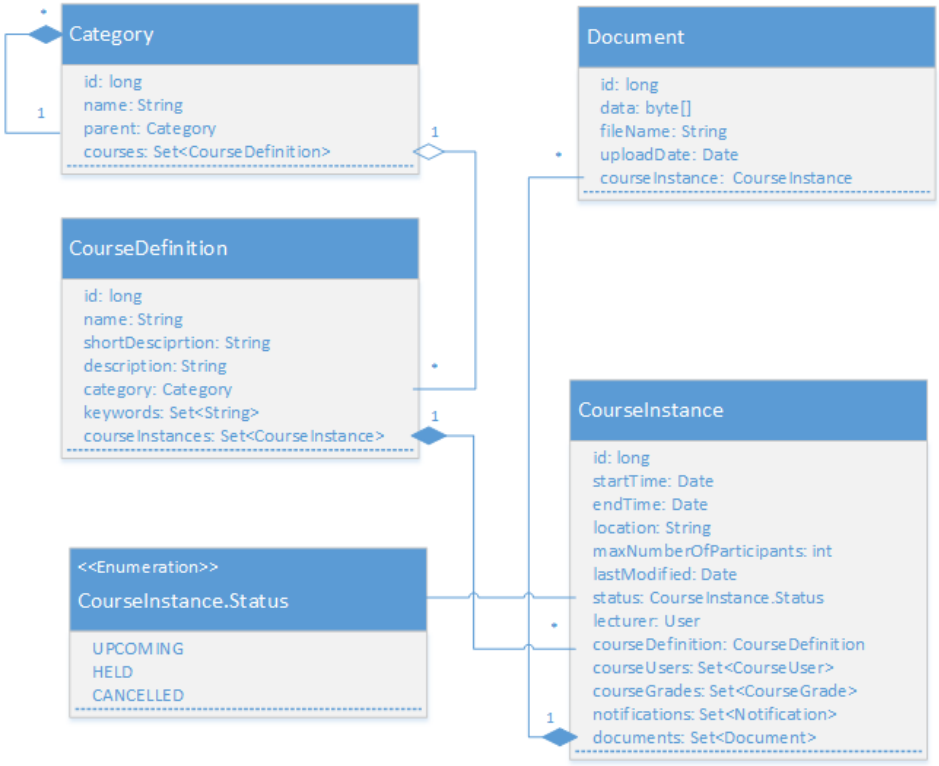
CourseUser:

Entitetom *CourseUser* modeliramo vezu između korisnika i jednog termina tečaja. Sastoji se od komentara (atr. *comment*) kojim korisnik može poslati kratak komentar administratoru prilikom prijave ili iskazivanja interesa, a administrator može napisati razlog odbijanja ili pozdravnu poruku kod prihvaćanja na tečaj. Atributom *active* označavamo da li je veza aktivna. Jedan korisnik može imati više veza na isti termin tečaja, za svaki tip veze po jednu. Neaktivnom vezom dobivamo povijesni pregled korisnikove zainteresiranosti ili prijave. Posljednji atribut koji nije veza označava samo vrijeme stvaranja entiteta (atr. *creationDate*). Ostala dva atributa *user* i *courseInstance* služe kako bi mogli povezati korisnika s određenim terminom tečaja.

CourseUser.Type:

Ova enumeracija određuje kojeg je tipa veza između korisnika i tečaja koje entitet *CourseUser* povezuje. Veza može biti: „*PRE_REGISTRATION*“ što predstavlja iskazivanje zainteresiranosti korisnika za tečaj, „*REGISTRATION*“ prijava na tečaj, „*ACCEPTED*“ označava da je korisnik prihvaćen i „*DECLINED*“ označava da je korisnik odbijen.

4.2. Entitet *CourseInstance* i vezani entiteti



Slika 4.2 - Dijagram razreda oko entiteta *CourseInstance*

CourseInstance:

Termin tečaja je uz entitet *User* najvažniji entitet u aplikaciji. On predstavlja jedan termin održavanja nekog tečaja. Definiraju ga obavezni atributi: vrijeme početka (atr. *startTime*), lokacija (atr. *location*) i maksimalan broj pristupnika (atr. *maxNumberOfParticipants*). Ostali atributi koji su opcionalni su vrijeme završetka (atr. *endTime*) i predavač (atr. *lecturer*). Predavač mora biti jedan od korisnika sustava koji ima odgovarajuću dozvolu. Termin tečaja mora biti vezan na neki određeni tečaj koji je predstavljen entitetom *CourseDefinition*. Za neki termin moguće je dodati materijale koji su dostupni samo za taj termin i oni su predstavljeni atributom *documents*. Također, moguće je dobiti sve obavijesti poslane korisnicima putem atributa *notifications*. Korisnička ocjena određenog termina tečaja je definirana vezom preko atributa *courseGrades*, te sve veze između korisnika i određenog termina su predstavljene atributom *courseUsers*. Na kraju atribut *status* određuje u kojoj je fazi životnog ciklusa određeni termin tečaja.

CourseInstance.Status:

Enumeracija *CourseInstance.Status* služi da bi dobili informaciju u kojem je stanju određeni termin tečaja. Kada se definira novi termin, on je u stanju „*UPCOMING*“ što predstavlja nadolazeći tečaj. Po završetku tečaja administrator mijenja status u „*HELD*“, a ako se iz nekog razloga održavanje termina otkazuje, postavlja se status „*CANCELLED*“.

CourseDefinition:

Ovaj entitet opisuje tečaj i kada korisnik želi pretražiti tečaj, pretraživanje se vrši nad atributima ovog entiteta. Glavni atribut koji se pojavljuje na gotovo svim mjestima gdje se spominje tečaj je naziv (atr. *name*). Drugi važan atribut je *shortDescription*. Koristi se u prikazima tečaja u listama i njegov opis bi trebao privući korisnika da otvori tečaj i pregleda ostale detalje, ne smije biti veći od 255 znakova. Atribut *description* predstavlja glavni opis tečaja i on nema ograničenu duljinu. U njega bi trebale ići najvažnije informacije o tečaju. U svrhu pretrage i naglašavanja ključnih riječi, entitet ima atribut *keywords* koji predstavlja skup ključnih riječi. Svaki tečaj spada u jednu kategoriju (atr. *category*) po kojoj se također vrši pretraga. Definicija tečaja nema mogućnost prijave korisnika, te stoga *CourseDefinition* ima vezu na termine tečaja putem atributa *courseInstances*.

Category:

Radi jednostavnije pretrage, svaki tečaj spada u jednu kategoriju i glavna zadaća ovog entiteta je da to omogućava. Kategorija tečaja je predstavljena atributoma *name* koji predstavlja naziv kategorije. Kategorija je modelirana na način da ima stablastu hierarhiju. Tako jedna kategorija može imati više podkategorija. Ime kategorije mora biti jedinstveno na razini podkategoriju u kojoj se nalazi. Da bi se ostvarila stablasta struktura svaka kategorija ima vezu na nad kategoriju (atr. *parent*). Također, moguće je dobiti sve tečajeve u kategoriji putem atributa *courses* koji je po tipu podataka skup.

Document:

Entitet *Document* predstavlja neki materijal koji se koristi na nekom terminu tečaja. Nema ograničenja na tip dokumenta, pa administrator može podijeliti korisnicima bilo koju datoteku. Binarni zapis datoteke pohranjen je u atributu *data*,

naziv datoteke u atributu *fileName*, te datum upload-a u atributu *uploadDate*. Dokument je vezan na neki termin tečaja pa ima vezu na njega putem atributa *courseInstance*.

5. Izrada aplikacije

Prvi korak u izradi aplikacije je odabir arhitekture i radnih okvira s kojim će se raditi. Za ovaj rad je korišten radni okvir *Grails* za poslužiteljsku stranu zbog jednostavnosti definiranja modela podataka i jednostavnosti kod posluživanja korisničkih zahtjeva odgovarajućim podacima. Za klijentsku stranu je korišten radni okvir *AngularJS* zbog jednostavnosti u manipulaciji podacima i „punjenju" podataka unutar HTML-a. Ovi radni okviri su potpuno odvojeni jedan od drugog i to pruža mogućnost paralelnog razvoja jedne i druge strane.

5.1. Poslužiteljske aplikacije

Poslužitelj je razvijan koristeći radni okvir *Grails*. Na poslužitelju je potrebno definirati model podataka, ulazne točke za korisničke aplikacije i kontrolere koji će obrađivati pristigle zahtjeve.

Poslužitelj se sastoji od četiri glavna direktorija.

- `Domain` – Sadrži domenske entitete
- `Controllers` – Sadrži sve kontrolere
- `Views` – Sadrži ulazne točke za svaku aplikaciju
- `Conf` – Sadrži konfiguraciju aplikacije

Prilikom razvoja aplikacije potrebno je konstantno testiranje i česta izmjena modela. Česta izmjena modela nam onemogućava da podatke jednom unesemo u bazu i nakon toga ju stalno koristimo. Kako se svi podaci ne bi morali ručno unositi kod svakog pokretanja poslužitelja, u konfiguracijskom direktoriju aplikacije se nalazi datoteka `Bootstrap.groovy` koja se izvršava kod svakog pokretanja. U nju se programski podaci spremaju u bazu i nakon što je aplikacija pokrenuta ona već sadrži te podatke. Ako sada mijenjamo model, samo u toj datoteci trebamo napraviti odgovarajuće izmijene i ponovno pokrenuti poslužitelj.

URL mapiranje je važan korak u izradi poslužitelja. To je zapravo način na koji će klijentske aplikacije komunicirati s poslužiteljom i način na koji će korisnici pristupati klijentskoj, odnosno administratorskoj aplikaciji. URL za administratorsku aplikaciju je `/admin`, a za klijentsku `/public`. Korisničke aplikacije komuniciraju s poslužiteljom preko posebnih URL-ova ovisno o razini ovlasti koja je potrebna da bi se zahtjev izvršio. Administratorski zahtjevi se šalju na URL koji počinje s `/api/admin`, javni zahtjev na `/api/public` i zahtjevi od registriranih korisnika na `/api/restricted`. Ovim odvajanjem zahtjeva se onemogućava da korisnik izvršava zahtjeve za koje nema ovlasti.

Budući da klijentske aplikacije razmjenjuju podatke s poslužiteljom isključivo preko HTTP zahtjeva, potrebno je serijalizirati podatke kako bi ih obje strane mogle razumjeti. Odabrani format za serijalizaciju je JSON zbog toga što s njim klijentske JavaScript aplikacije najlakše barataju. Jednom kada možemo serijalizirati naš domenski entitet, imamo problem da se serijaliziraju svi atributi, što nam u većini slučajeva ne treba, a u nekim predstavlja problem. Entitet *User* sadrži atribut koji čuva zaporku i to nikako ne želimo slati preko mreže. Kako bi se serijalizirali jedno oni atributi koje mi to želimo, potrebno je na neki način definirati što se želi serijalizirati. U ovoj aplikaciji to je napravljeno tako da je za svaki entitet definirano koji podaci se serijaliziraju. Jedan entitet se obično koristi na više mjesta i nije nužno da svako mjesto treba iste podatke, pa je i za svako mjesto korištenja definirana serijalizacija. Za ovo je korišten *Grails* dodatak *Grails-Marshallers* [12].

5.2. Klijentske aplikacije

Razvijene su dvije klijentske aplikacije, obje koristeći radni okvir *AngularJS*. Jedna je namijenjena administratoru, a druga svim korisnicima, registriranim i neregistriranim.

Obje aplikacije imaju istu strukturu direktorija. Svaka ima svoj specifičan css stil, direktorij s parcijalnim pogledima i direktorij u kojem se nalaze AngularJS skripte. Struktura `web-app` direktorija prikazana je u nastavku.

- `public` – Direktorij koji sadrži korisničku aplikaciju
  - `css`
  - `partials`
  - `scripts`
- `admin` – Direktorij koji sadrži administratorsku aplikaciju
  - `css`
  - `partials`
  - `scripts`
- `scripts`
  - `CourseMan` – Direktorij sa zajedničkim skriptama koje se koriste u obe aplikacije

Direktorij `scripts` u `admin` i `public` direktoriju sadrži AngularJS aplikaciju. Angular aplikacije se sastoji od direktorija u kojem se nalaze kontroleri i direktorija u kojem se nalaze servisi.

U angular kontrolerima se nalazi logika vezana uz parcijalni pogled za kojeg je on zadužen. Kontroleri ne sadrže kôd koji direktno šalje HTTP zahtjeve. Kada kontroler treba dohvatiti ili spremiti neki podatak, on ubaci (engl. *inject*) servis koji je zadužen za taj podatak i njemu delegira posao. Time se dobiva mogućnost višekratnog korištenja istih servisa u različitim kontrolerima. Također, kôd u kontroleru postaje razumljiviji jer sadrži samo logiku za pogled.

Angular servisi služe za dohvat, spremanje, izmjenu i brisanje podataka. Jedino se u njima stvaraju HTTP zahtjevi i unose adrese na kojima će poslužitelj prihvatiti ili poslati podatke.

Obe aplikacije imaju neke dodirne točke. Dodirne točke su adrese na koje servisi u aplikacijama šalju zahtjeve, način spremanja prijavljenog korisnika, globalni događaji vezani u autentifikaciju i angular filtri za koje ima smisla da se koriste u obe aplikacije. Također vrlo velika dodirna točka je servis koji je zadužen za prijavu, odjavu i registraciju korisnika.

Kako se ne bi duplicirao ovaj kôd u obje aplikacije u direktoriju `scripts/CourseMan` se nalaze skripte koje predstavljaju zaseban angular modul koji sadrži sve dodirne točke. Ovaj modul se potom može jednostavno ubaci u svaku aplikaciju.

Zaključak

Prednost jednostraničnih web aplikacija je mogućnost izrade intuitivnog i interaktivnog korisničkog sučelja i potpuna odvojenost klijenta od poslužitelja što omogućava paralelan razvoj.

AngularJS je odličan radni okvir koji omogućava izradu lako održivih klijentskih aplikacija. Prilagodljiv je raznim arhitekturama, bilo to MVC, MVP ili MVVM arhitektura. Angular aplikacije trebaju poslužitelja s kojeg će dobivati potrebne podatke. Poželjno je da je poslužitelj također jednostavan i da omogućava brz razvoj i jednostavno održavanje.

Grails radni okvir je upravo to. Omogućava vrlo jednostavnu izradu podatkovnog modela, spremanje i dohvaćanja potrebnih podataka putem GORM-a. Posluživanje tih podataka klijentskoj aplikaciji je također vrlo jednostavno korištenjem *Grails* kontrolera. Serijalizacija podataka u *JSON* ili *XML* format je ugrađena u radni okvir. Velika prednost *Grails* radnog okvira je njegova nadogradivost putem dodataka (engl. *plugins*) koji pružaju nove funkcionalnosti.

Aplikacija za upravljanje tečajevima korisnicima pruža mogućnost jednostavne pretrage i prijave na tečaj. Za korisnike na tečaju ona omogućava jednostavno preuzimanje materijala i dobivanje obavijesti o dostupnosti novih materijala. Administratoru se nudi mogućnost jednostavnog definiranje novih tečajeva i definiranja termina za postojeće tečajeve. Za svaki termin tečaja je omogućeno jednostavno upravljanje s prijavljenim korisnicima i diseminacija materijala.

Literatura

[1] HTML, http://www.w3.org/html/, Preuzeto 10. Travnja 2014.

[2] CSS, http://www.w3.org/Style/CSS/, Preuzeto 11. Travnja 2014.

[3] JAVASCIPRT, https://developer.mozilla.org/en-US/docs/Web/JavaScript, Preuzeto 18. Travnja 2014.

[4] GRAILS, https://grails.org/, Preuzeto 20. Travnja 2014.

[5] ANGULARJS, https://angularjs.org/, Preuzeto 01. Svibnja 2014.

[6] GROOVY, http://groovy.codehaus.org/, Preuzeto 01. Svibnja 2014.

[7] SPRING, https://spring.io/, Preuzeto 12. Svibnja 2014.

[8] SPRING SECURITY, http://grails-plugins.github.io/grails-spring-security-core/, Preuzeto 13. Svibnja 2014.

[9] JQUERY, http://jquery.com/, Preuzeto 15. Svibnja 2014.

[10] UI BOOTSTRAP, http://angular-ui.github.io/bootstrap/, Preuzeto 20. Svibnja 2014.

[11] BOOTSTRAP, http://getbootstrap.com/, Preuzeto 20. Svibnja 2014.

[12] GRAILS MARSHALLERS, https://github.com/aruizca/grails-marshallers, Preuzeto 22. Svibnja 2014.

Sažetak

Web aplikacija je bilo koja aplikacija koja koristi internetski preglednik kao klijenta putem kojeg se prikazuje korisniku. Responzivna web aplikacija je web aplikacija koja se jednako dobro prikazuje na velikim i malim rezolucijama ekrana. U zadnje vrijeme je sve popularnija izrada web aplikacija prilagođenih mobilnim uređajima umjesto izrade klasične mobilne aplikacije. Aplikacija za upravljanje tečajevima je jednostranična web aplikacija izgrađena na klijentskoj strani korištenjem AngularJS i Bootstrap radnih okvira, te Grails radnog okvira na poslužiteljskoj strani. Ona omogućava korisnicima jednostavno pretraživanje i prijavu na tečajeve. Administratoru omogućava jednostavno upravljanje tečajevima, prijavama korisnika na tečajeve, distribucija materijala i slanje obavijesti.

Summary

Web application is any application that uses web browser as client with which it is presented to users. Responsive web application is web application which is equally good displayed on large and small screen resolutions. Lately, development of web applications designed for mobile devices is becoming more popular than development native mobile applications. Course management mobile application is single page web application built with AngularJS and Bootstrap frameworks for frontend and Grails framework on backend. Users can easily browse and register for various courses. Administrator can easily manage courses, user registrations and sending out notifications and course materials.

Skraćenice

- HTML HyperText Markup Language
- CSS Cascading Style Sheets
- URL Uniform resource locator
- DOM Document Object Model
- API application programming interface
- JVM Java Virtual Machine
- JSON JavaScript Object Notation

Privitak

Instalacija programske podrške

Za pokretanje aplikacije potrebno je napraviti sljedeće korake:

1. Ukoliko nemate, potrebno je instalirati JRE (Java runtime environment) www.oracle.com
2. Skinuti Apache Tomcat poslužitelj I raspakirati ga negdje na disku http://tomcat.apache.org/
3. U direktorij `/{TOMCAT_ROOT}/webapps` kopirati `CourseMan.war` datoteku
4. Pokrenuti `startup.bat` datoteku unutar `/{TOMCAT_ROOT}/bin` direktorija
5. Sada bi se trebao pokrenuti server i ako sve prođe uredu na adresi http://localhost:8080/CourseMan/public se nalazi korisnička aplikacija. Administratorska aplikacija: http://localhost:8080/CourseMan/admin