

Razvoj web aplikacije za muzičku društvenu mrežu

Završni rad 1. ciklusa

Mentor: Doc.dr. Samir Omanović, dipl.ing.el.

Student: Dajan Bračković

Sarajevo, septembar 2014.

Postavka rada

Odsjek: Računarstvo i informatika

Tema: Razvoj web aplikacije za muzičku društvenu mrežu

Student: Dajan Bračković

Cilj:

Primarni cilj je upoznavanje neke od novijih tehnologija za razvoj web baziranih aplikacija. Izabran je Play Framework, koji je baziran na MVC arhitekturnom pattern-u i (u ovom slučaju) Java programskom jeziku na serverskoj strani.

Opis:

Kao praktični dio ovog rada razvijena je jednostavna web aplikacija koja predstavlja muzičku društvenu mrežu gdje korisnici mogu dijeliti muziku. Kroz ovu aplikaciju su prikazani principi, načini implementiranja određenih funkcionalnosti te i neki teoretski aspekti Play framework-a. Aplikacija je primjer jednostavne društvene mreže koja će omogućiti korisnicima da razmjenjuju muziku.

Plan rada:

1. Upoznavanje sa tehnologijama potrebnim za razvoj aplikacije
2. Planiranje razvoja aplikacije
3. Razvoj aplikacije

Mentor: Doc.dr. Samir Omanović, dipl.ing.el.

Sadržaj

Postavka rada	2
Sadržaj	3
Sažetak.....	4
Abstract	4
Uvod	6
O Play framework-u.....	7
Instalacija Play-a	8
Play2 plugin za Eclipse Scala IDE	10
Sadržaj Play projekta	11
Izrada aplikacije	12
Opis.....	12
Prijava na sistem.....	13
Mapiranje modela	14
Povezivanje sa bazom podataka.....	16
Dizajniranje i kreiranje pogleda (views)	17
Statički resursi	19
Kontroleri (controllers).....	20
Primjer ponašanja kontrolera.....	22
Security.Authenticator kontroler	27
Testiranje u Play framework-u	27
Zaključak	30
Pojmovi i skraćenice	31
Indeks slika	34
Literatura	36
Knjige i predavanja:	36
Linkovi:.....	36

Sažetak

Web je vjerovatno trenutno najveće softversko tržište i u ovom radu se prikazuje proces izrade jedne web aplikacije u jednoj relativno novijoj tehnologiji čija se inicijalna verzija pojavila 2007 godine, a malo kasnije se počela sve češće pojavljivati kao jedan novi uzbudljiviji izbor za Java web programere. Ovaj rad predstavlja i neke od osobina Play Framework-a[4] koji postaje sve popularniji među Java web developerima. Play framework za razvoj web aplikacija u Java ili Scala programskom jeziku objedinjuje razne tehnologije čijom integracijom se postiže finalni rezultat i vidu web aplikacije. Ove tehnologije su opisane u ovom dokumentu. To su tehnologije poput Javascripta, CSS-a i HTML-a i druge. Play nudi alternative i neka unapređenja u smislu korištenja CoffeeScripta umjesto JavaScripta i recimo LESS-a umjesto CSS-a. Play omogućava integriranje raznih tehnologija i mnoge kombinacije istih, te su neke od tih tehnologija i opisane. Play predstavlja jedan vid MVC arhitekture te su osobine MVC arhitekturnog patterna opisane. Predstavljene su neke karakteristike MVC arhitekturnog pattern-a, ali i neke konkretne osobine MVC arhitekture u Play framework-u. Kao baza podataka je napravljena MySQL baza podataka. Play nudi mnoštvo baza podataka kao izbor. Ovdje je iskorištena MySQL baza podataka, kreirana putem WampServer softvera.

Softveri korišteni pri izradi ove aplikacije podrazumijevaju Eclipse IDE okruženje u koje je ugrađen plugin za Play framework, WampServer koji je iskorišten za kreiranje relacione baze podataka i nadgledanje iste putem phpmyadmin opcije koju nudi WampServer, te web browser, što je u ovom slučaju bio Google Chrome. Za izradu web aplikacije koristeći Play Framework bio je dovoljan i samo web browser jer Play framework omogućava pregled, nadgledanje, testiranje i uređivanje cijelog projekta i koda putem browser-a. Za implementaciju ove aplikacije je upotrebljeno Eclipse IDE razvojno okruženje te play2 plugin za ovo razvojno okruženje koji omogućava integriranje opcija Play framework-a u razvojno okruženje i upotrebu istih.

Abstract

Web is probably the biggest software marketplace. This paper shows the process of implementing a web application using one of the newer frameworks for web programmers whose initial version became available in 2007, but it was couple of years later that it became much more used as an option for web programmers, especially for Java web developers. Play framework is tool for creating web application using Java or Scala programming language and it combines different technologies in order to produce it's final product and that is web application. Technologies like HTML, Javascript, CSS and many more. Play framework provides alternatives and improvements for Javascript and CSS in Coffescript and LESS respectively. As Play framework uses a lot of technologies and tools, the fundamentals of these technologies are a part of this document. Play is MVC based web Framework and concepts of MVC architecture are explained in this paper. The paper presents some features of MVC architecture in Play framework specifically. The database used for this project is MySql database. WampServer was used for creating the MySql database. Play offers many databases for developers, so any kind of relational database could be used for this purpose. The default one in Play applications is H2 database.

The software used for working on this application includes Eclipse IDE which supports Play plugin, WampServer used for creating, controlling and monitoring the database using phpmyadmin option which is provided by WampServer, and finally web browser. Browser in this case is Google Chrome. Web browser is everything a developer really needs for creating application using Play framework, as Play offers GUI in web browser for monitoring, editing, controlling, running and testing the project.

Uvod

Uvod

Za potrebe ovog rada je implementirana web aplikacija pod nazivom „Malina“ u Play framework-u [4] te su i predstavljene osnove ovog framework-a. Ova aplikacija predstavlja društvenu mrežu gdje korisnici iste mogu da međusobno razmjenjuju muzički sadržaj u vidu Youtube link-ova. Osobine Play framework-a korištenog za razvoj ove aplikacije su pored toga što su prikazane iz teoretskog aspekta, predstavljene i objašnjene na konkretnim primjerima iz aplikacije.

Web [1] je veoma popularan i izazovan za programiranje i predstavlja najveće softversko tržište. Aplikacija je razvijena u jednom od novijih framework-a za izradu web aplikacija, te je isti sve popularniji među web developerima. Društvene mreže također predstavljaju nešto što je veoma popularno u internet svijetu, pogotovo među mlađom populacijom. Java uz ASP.NET [1] predstavlja najkorišteniju serversku tehnologiju današnjice. Tako da se može reći da su popularne tehnologije iskorištene za izradu aplikacije na temu koja također predstavlja nešto aktuelno u svijetu web aplikacija.

Aplikacija „Malina“ je kreirana po uzoru na društvene mreže tipa Soundcloud i Last.fm koje predstavljaju društvene mreže za dijeljenje muzičkog sadržaja. Aplikacija „Malina“ također omogućava neke slične funkcionalnosti koje nude i ove dvije društvene mreže, ali je i jednostavnija za upotrebu jer nudi jednostavniji interfejs za korisnika, gdje isti može podijeliti muzički sadržaj na zajednički home page vidljiv svim prijavljenim korisnicima.

Prvo poglavlje ovog rada govori nešto općenito o Play framework-u te opisuje neke tehnologije koje su karakteristične za Play. Nakon toga slijedi poglavlje koje opisuje kako započeti svoj prvi Play projekat, kako instalirati Play framework i kako ugraditi play2 [13] plugin u Eclipse razvojno okruženje [7]. To je poglavlje „Instalacija Play-a“. Naredno poglavlje govori o sadržaju Play projekta, tj. o nekim bitnim fajlovima i direktorijima u projektu. Sljedeća poglavlja se tiču izrade aplikacije i opisivanja konkretnih funkcionalnosti u aplikaciji „Malina“. Poglavlje koje slijedi nakon opisivanja izrade aplikacije se zove „Testiranje u Play framework-u“ i govori o testiranju projekta. Zadnje poglavlje predstavlja zaključak.

O Play framework-u

Play framework je open-source web framework, napisan u Java [1] i Scala [7] programskom jeziku, te nudi Javu i Scalu kao dvije opcije za serversku stranu koda. Programski jezik Scala se može predstaviti kao jedan vid unapređenja Java programskog jezika. Dosta Java programera prelazi na Scalu jer pruža neke olakšice posebno u sintaksnom smislu. Budući da se Scala kod kompajlira u Java bajtkod, u koji se i Java također kompajlira i pokreće se na JVM [7] (Java Virtual Maschine), znači da i Scala ustvari radi na istoj platformi kao i Java, a to je JVM.

Kao što sam već rekao baziran je na MVC [3] arhitekturalnom patternu. MVC je softverski arhitekturalni pattern koji aplikaciju razdvaja u tri glavne cjeline: Model, View i Controller. On također omogućava i interakciju između ove tri cjeline. MVC razdvaja prezentaciju, interakciju i podatke. **Model** predstavlja podatke i poslovnu logiku aplikacije i obično je slika stanja u bazi podataka. Model obavlja svu manipulaciju nad podacima koji se nalaze u bazi podataka pozivajući određene metode. Model također obavještava view o promjenama stanja. **View** sadrži komponente koje definišu i odnose se na prikaz korisničkog interfejsa aplikacije. View predstavlja vizuelnu reprezentaciju modela. **Controlleri** su komponente koje izvršavaju interakciju sa korisnikom, rade sa modelima, i selektuju koji view će se prikazati. Kontroler upravlja interakcijom sa korisnikom te proslijeđuje instrukcije view-u i model-u. U web aplikacijama je kontroler zadužen i za procesiranje HTTP zahtjeva i za validaciju podataka.

Jedna od prednosti MVC arhitekture što omogućava da različite vrste programera rade na odvojenim dijelovima aplikacije istovremeno. Tako da na primjer jedan programer može raditi dizajn grafičkog interfejsa dok drugi radi na poslovnoj ili kontrolnoj logici. MVC dozvoljava da se podaci mijenjaju neovisno od njihove prezentacije i obrnuto.

Jedan od nedostataka MVC arhitekturalnog pattern-a može biti to što povlači sa sobom dodatni kod i dodatnu kompleksnost koda i kada je sistem jednostavan.

Play omogućava i upotrebu ranije spomenutih unapređenja nekih od tehnologija i rad sa istim. Pod istim se podrazumijeva **LESS** [7] i **CoffeeScript** [7]. **LESS** je razvio Alexis Sellier. Prva verzija se pojavila 2009 godine. To je ustvari dinamički stilski jezik i ima neke mogućnosti koje CSS [1] nema, a to su varijable, funkcije, operatori te dozvoljava ugnježavanje elemenata. **CoffeeScript** je programski jezik koji se prekompajlira u JavaScript [1] i razvio ga je Jeremy Ashkenas. Jezik se također pojavio 2009 godine.

Play je djelo programera Guilleme Bort-a. Prva verzija se pojavila 2007 godine. Verzija 1.1 je 2010 godine premještena na GitHub. Play je inspirisan Ruby On Rails [7] web framework-om. Play je u potpunosti **RESTful** [1] framework. Koristi **JBoss Netty** [7] kao web server, te zahvaljujući tome može usluživati velike zahtjeve asinhrono. Play koristi **SBT**(Scala Build Tool) [7] koji predstavlja nešto slično Maven-u, te pomoću njega vodi računa o svim zavisnostima, te je dodavanje istih zaista vrlo jednostavno. SBT također omogućava konstantno kompajliranje i build projekta. U Play je integrisana podrška za unit testiranje. Za mapiranje modela u bazu Play koristi **Ebean**, ORM(Object-realtional Mapping) alat [7]. Ebean sa svojim jednostavnim API-om čini operacije spašavanju bazu veoma jednostavnim. Mnogo je jednostavniji od Hibernate-a [3] koji također predstavlja ORM alat za Java projekte.

Kao jednu od činjenica koju su stvoritelji ovog framework-a istakli a koja Play razlikuje od ostalih web frameworka je to da je Play napravljen od strane web developer-a za web developere. Play opisuju kao veoma jednostavan framework za izradu „reactive applications“ [4], koje ustvari moraju imati osobine kao što su velika dostupnost i što kraće vrijeme odziva da bi odgovorile zahtjevima velikog broja korisnika i što bolje radile pod velikim „opterećenjem“. Play aplikacije se baziraju na event-driven modelu koji je baziran na asinhronoj komunikaciji. Play omogućava da se razviju aplikacije koje su skalabilne i lako mogu odgovoriti na zahtjeve povećavajućeg broja korisnika. Tvorci Play-a ističu da se malo razmišlja o mogućim kvarovima tokom izrade aplikacije te ne sakrivaju svoje neslaganje s tim, te su kroz Play aplikacije omogućili da se kvarovi kako oni kažu „izoliraju“ te da se pomoću toga onemogući njihov utjecaj na rad ostatka aplikacije. Vjerovatno najpoznatija web stranica koja koristi Play framework je LinkedIn. Za potrebe ovog rada je napravljena ipak jedna jednostavnija aplikacija koja nije u potpunosti iscrpila sve opcije koje Play nudi.

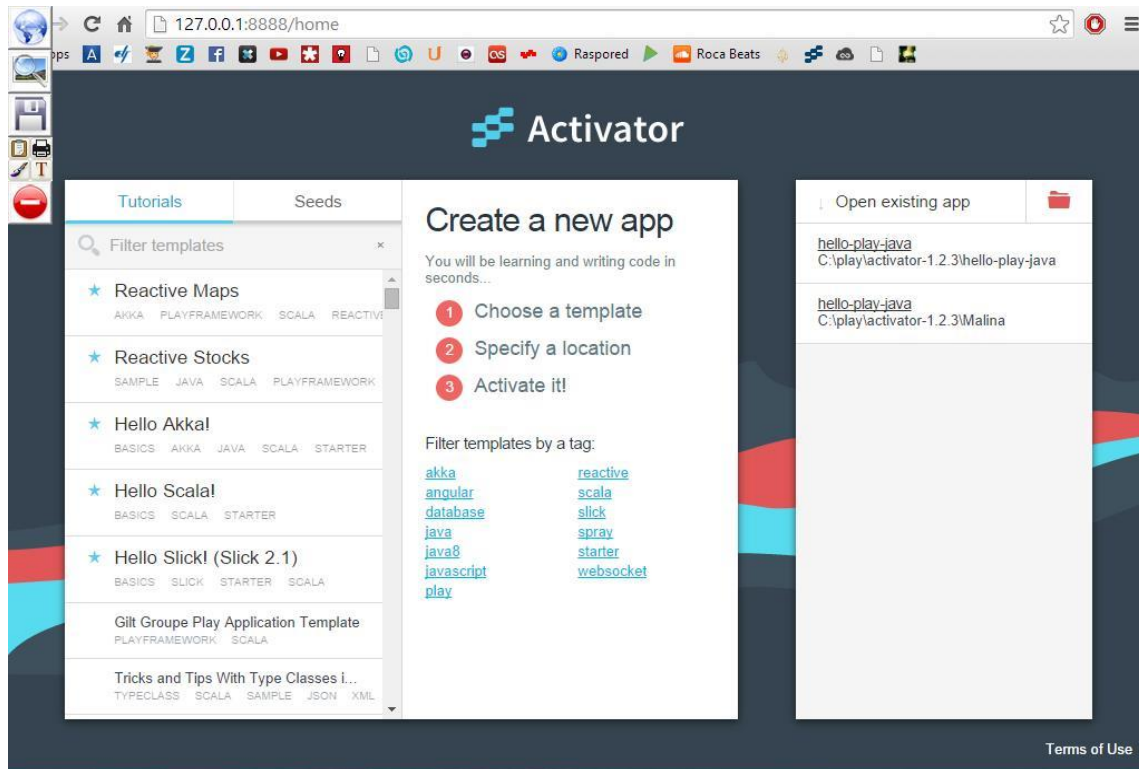
Instalacija Play-a

Play je poprilično jednostavno instalirati te zatim započeti sa izradom svoje aplikacije. Sve što je potrebno je skinuti paket sa njihove oficijelne stranice <https://www.playframework.com>. Može se iskoristiti i ovaj link <https://typesafe.com/platform/getstarted> na kojem se također nudi posljednja verzija Play-a sa još nekoliko ugrađenih komponenti koje omogućavaju izgradnju takozvanih „reactive applications“. Jedna od tih komponenti je Akka [12] koja stoji iza event-driven modela, skalabilnosti i izoliranja kvarova.

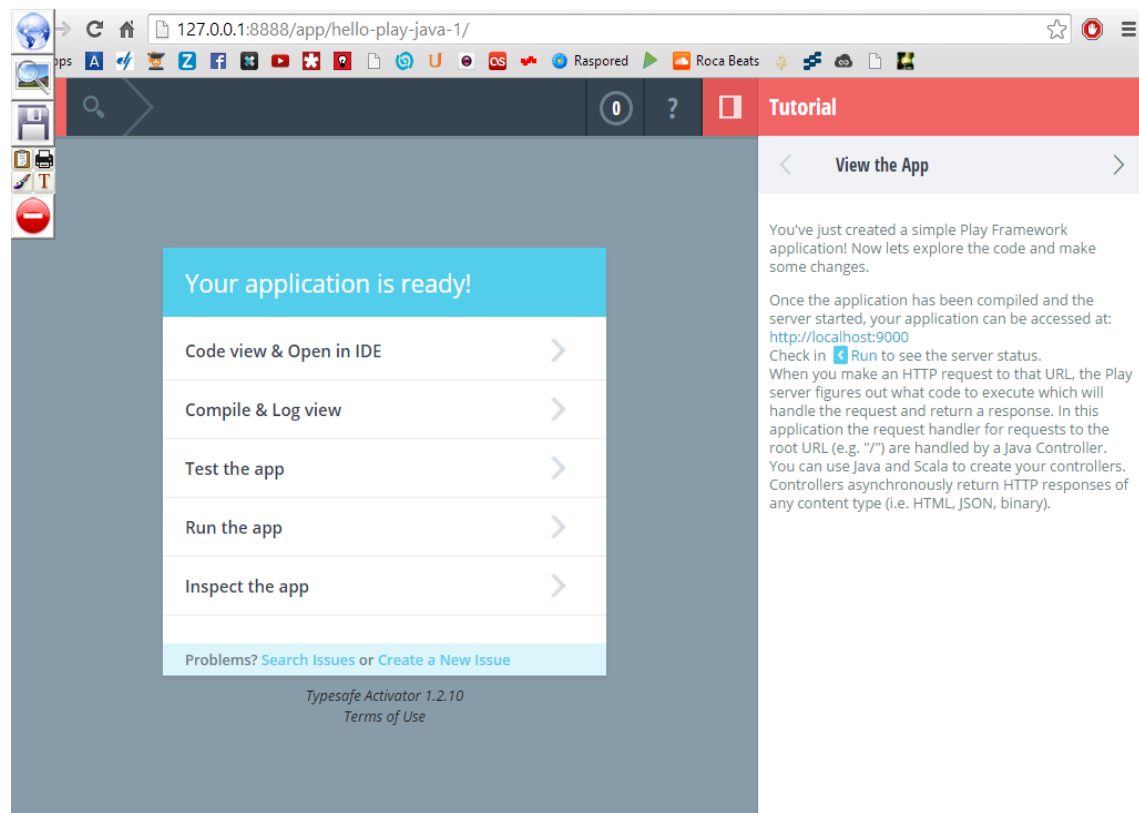
Da bi mogli koristiti Play, potreban je JDK 6 ili noviji. Na računaru korištenom za izradu ove aplikacije je instaliran JDK 8, što trenutno predstavlja posljednju verziju Java SE Development Kit-a.

Nakon downloada paketa potrebno ga je naravno raspakovati. Bilo bi dobro ubaciti putanju raspakovanog direktorija u sistemsku varijablu PATH. Do nje se dolazi na sljedeći način: Control Panel\System and Security\System -> Advanced System Settings -> Environment Variables. Ukoliko se pokrene activator.bat, otvorit će se u browseru korisnički interfejs koji nam omogućava da izaberemo neki od template-a te napravimo svoju aplikaciju na osnovu izabranog template-a. Novi tab se otvara na adresi 127.0.0.1:8888.

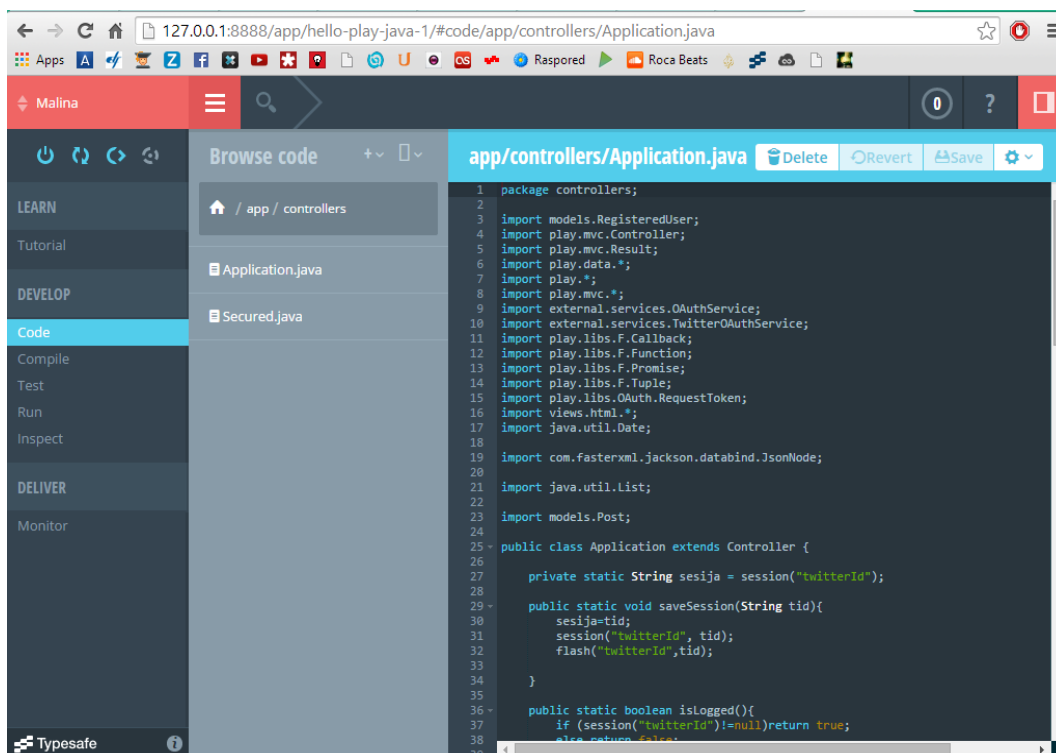
Razvoj web aplikacije za muzičku društvenu mrežu, Dajan Bračković
Elektrotehnički fakultet Sarajevo, akademska 2013-2014. godina.



Slika 1 – GUI u browseru za kreiranje Play projekta



Slika 2 – Activator GUI za upravljanje Play projektom



Slika 3 – Prikaz koda u Activator GUI-u u browseru

Na ovim slikama se vidi da je putem browsera moguće pokrenuti, pregledati, testirati, kompajlirati projekat, kao i vršiti izmjene u kodu. U browseru se također prikazuju greške i errori u kodu. Tako da je ustvari jedina neophodna stvar za razvoj Play aplikacije web browser. Tu je naravno i opcija Open in IDE, tako da je projekat moguće importovati u Eclipse ili IntelliJ razvojno okruženje. Za neke od template-a Play nudi i tutorijale koji objašnjavaju pobliže određene karakteristike za koji je template namijenjen. Projekat je moguće kreirati i putem konzole. Aplikacija „Malina“ je kreirana putem konzole. Nakon pokretanja konzole, potrebno se pozicionirati u raspakovani direktorij te pokrenuti komandu activator new. Nakon pokretanja komande activator new, konzola prikazuje 4 osnovna template-a i nudi nam prikaz ostalih 178 template-a. Zatim unosom template-a i naziva aplikacije kreiramo aplikaciju. Ista se može pokrenuti kroz konzolu ukoliko unutar direktorija projekta pokrenemo komandu activator run ili play run te se zatim aplikacija otvara u browseru na portu 9000. Za kreiranje neophodnih Eclipse fajlova potrebno je pokrenuti komandu eclipse. Nakon toga je moguće projekat importovati u eclipse. Pomoću konzole je također moguće otvoriti GUI u browseru koristeći komandu activator ui, nakon koje se otvara prozor localhost adrese sa portom 8888.

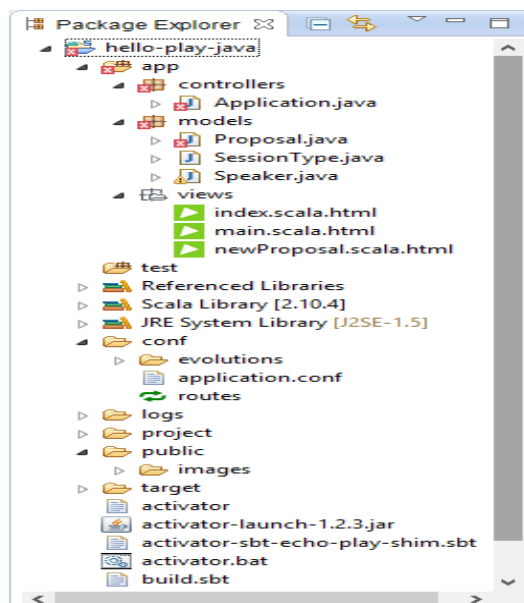
Play2 plugin za Eclipse Scala IDE

Za Eclipse postoji Play2 [13] plugin. Međutim za upotrebu plugina potrebno je koristiti Scala IDE Eclipse dostupan na ovom linku <http://scala-ide.org/download/sdk.html>. Naravno moguće je koristiti bilo koji Eclipse međutim samo će uz ovaj plugin biti dostupne neke privilegije kao naprimjer Code Assist. Play-ovi templati su napisani u Scali tako da ključne riječi u kodu neće biti prepoznate kao takve ukoliko se ne koristi Scala IDE. Ukoliko je Java jezik vaše aplikacije, potrebno je samo da se prebacite u Java perspektivu u Eclipse razvojnom okruženju. Play2 plugin se download-uje putem

Eclipse-ove opcije Help->Install New Software. Plugin je dostupan na linku <http://download.scala-ide.org/sdk/e38/scala210/stable/site> i on se nalazi u grupi Scala IDE plugins.

Sadržaj Play projekta

Direktorij jednog Play projekta izgleda kao na sljedećoj slici:



Slika 4 – Sadržaj Play projekta

App direktorij se uglavnom sastoji od 3 glavna paketa controllers, models i views. Ova 3 paketa su karakteristična za MVC bazirane projekte. Paket controllers sadrži sve controller klase. Paket models sadrži modele koji se koriste u projektu. Paket views sastoji sve view-e. Naravno po potrebi moguće je dodavati i dodatne pakete u app direktorij.

U test direktoriju se nalaze default-ne JUnit [2] testovi. Tu su dvije klase obično po default-u. ApplicationTest klasa sadrži neke jednostavne JUnit testove. Klasa IntegrationTest dolazi sa jednim integracijskim testom za browser.

Još jedan bitan fajl je application.conf [4] fajl koji se nalazi u conf direktoriju. U njemu se nalazi sva konfiguracija projekta.

Routes fajl [4] definiše mapiranje između HTTP ključnih riječi tipa GET[1] i POST [1], putanje i koda koji je potrebno izvršiti da bi se taj zahtjev uslužio.

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

GET    /                               controllers.Application.index()
GET    /register                      controllers.Application.register() ...
```

Slika 5 – Routes fajl

Na primjer ukoliko se uputi zahtjev za stranicu *localhost:9000*, time se uputio GET zahtjev za / (tarabu) te je time pozvana *index()* metoda iz *Application* kontrolera. Ukoliko klijent uputi serveru zahtjev na URI *localhost:9000/register* to će u kontroleru pozvati metodu *register()*.

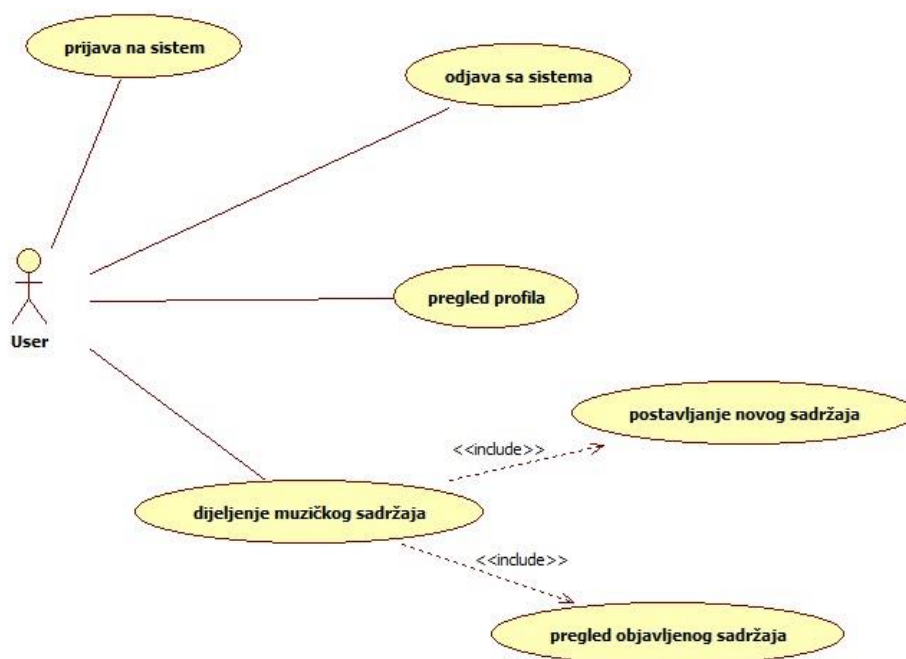
Public direktorij sadrži sve statičke fajlove kao što su na primjer slike. One su u poddirektoriju *images*. U public direktoriju obično su još 2 poddirektorija pored *images* direktorija, a to su direktoriji *javascripts* i *stylesheets*, koji sadrže respektivno JavaScript i CSS fajlove. Pristup ovim fajlovima je definisan u *routes* fajlu.

Izrada aplikacije

Opis

Aplikacija kreirana za potrebe ovog rada predstavlja muzičku društvenu mrežu, koja se zove Malina. Aplikacija „Malina“ i svojim dizajnom i skupom jednostavnih funkcionalnosti podsjeća na malinu, uz koju idu osobine slatka i jednostavna.

Korisnici ove aplikacije imaju jedan vrlo jednostavan user interfejs na raspolaganju koji im omogućava registraciju, prijavu te zatim i dijeljenje željenog muzičkog sadržaja u vidu YouTube linkova. Svi linkovi idu na jedan zajednički dashboard. Oni se na istom prikazuju u redoslijedu od najnovijih do najstarijih. Korisnik naravno ima i opciju odjave sa sistema. To se fino vidi na sljedećem dijagramu slučajeva upotrebe.

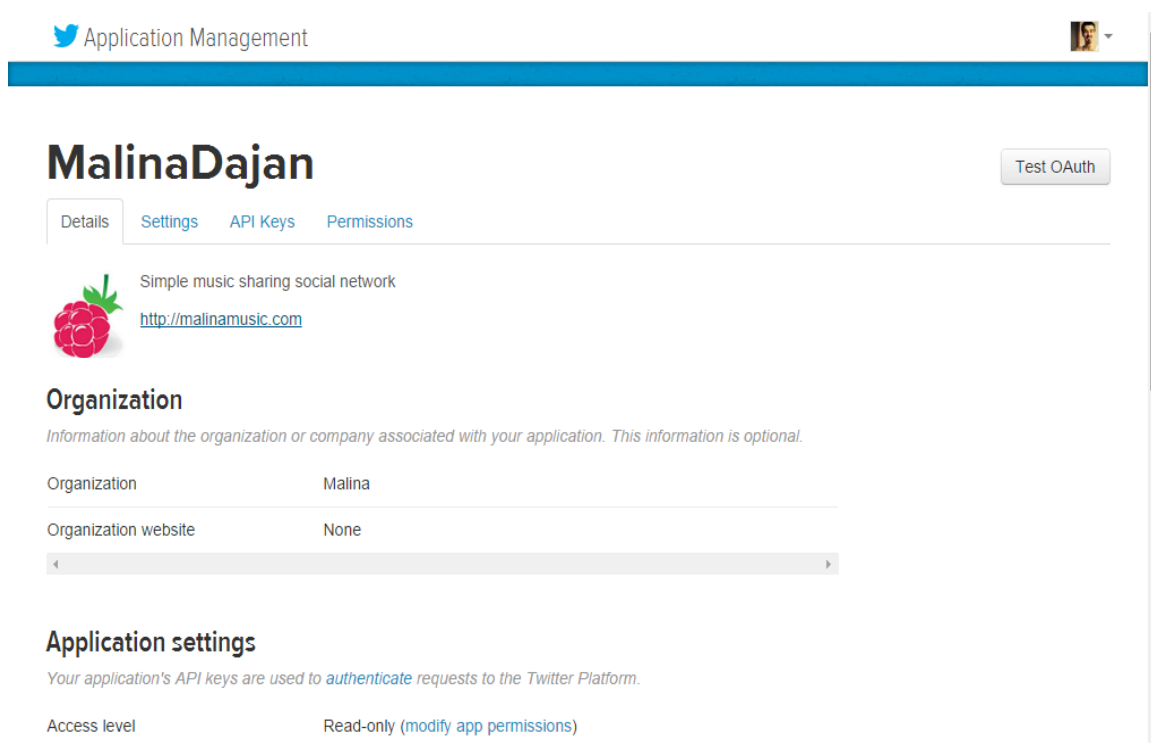


Slika 6 – Dijagram slučajeva upotrebe za korisnika

Prijava na sistem

Korisnik se na sistem prijavljuje samo ukoliko ima Twitter korisnički račun. Tako da je za prijavu na sistem potrebno i dovoljno imati profil na Twitteru. Prvobitna namjera je bila implementirati prijavu na sistem putem Facebook-a i njegovog OAuth servisa međutim ispostavilo se mnogo težim od očekivanog integrisanje Facebook-ovog JavaScripta sa serverskim Java kodom i preuzimanje potrebnih informacija i njihovo konvertovanje u Java objekte. S toga je implementirana prijava na sistem putem Twitter-a i njegovog OAuth servisa jer se ispostavilo da postoji više dokumentacije na raspolaganju konkretno za Twitter OAuth i njegovu upotrebu u Play framework-u.

Prije integrisanja Twitter OAuth servisa [8] u aplikaciju potrebno je prethodno registrovati aplikaciju na stranici oauth provajdera, u ovom slučaju je to Twitter. To se može uraditi na stranici <https://dev.twitter.com>.



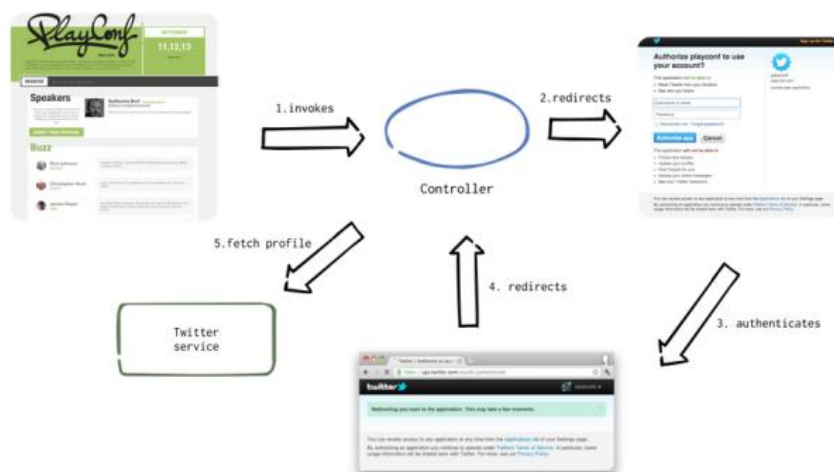
Slika 7 – Aplikacija Malina registrovana na Twitter-u

Pod tabom API keys se nalaze dvije informacije API key i API secret [8] koje su potrebne aplikaciji za povezivanje za Twitter OAuth servisom i pomoću njih se vrši komunikacija aplikacije i Twitter OAuth provajdera. Ove informacije bi trebale ostati tajne i ne bi trebalo biti moguće korisnicima da joj pristupe ili pročitaju jer bi u suprotnom korisničke informacije mogle biti dostupne nekom ko bi to mogao zloupotrijebiti.

OAuth omogućava jednostavan način interakcije sa zaštićenim korisničkim podacima. To je protokol koji omogućava sigurnu autorizaciju. Ovde je iskorišten za pristup korisničkim podacima na Twitteru. Kada korisnik klikne na dugme login, aplikacija će usmjeriti korisnika na Twitter. Nakon uspješne autentifikacije Twitter nas upućuje nazad do aplikacije šaljući pri tome token za potvrdu.

Pomoću ovog tokena možemo pozvati Twitterov REST servis i pristupiti korisničkom profilu i spasiti korisničke podatke u bazu podataka. Sljedeća slika grafički prikazuje tok operacije prijavljivanja.

Registration flow



Slika 8 – Tok operacije prijavljivanja preko Twitter OAuth-a

Nakon uspješne prijave korisniku se prikazuje home page dostupan samo prijavljenim korisnicima. To je ustvari dashboard na kojem se prikazuju svi linkovi koje su korisnici objavili.

Mapiranje modela

Nakon dobavljenih korisničkih podataka, iste je zaista, koristeći Ebean [7], vrlo jednostavno učiniti perzistentnim. Ebean je Object Relational Mapping alat za Javu. Nešto slično u .NET-u predstavlja Entity Framework [7]. Ebean je ugrađen u jezgro Play Framework-a. Da bi se omogućio potrebno je samo otkomentarisati jednu liniju koda u application.conf fajlu, a to je linija `ebean.default="models.*"`. Nakon toga su Ebean opcije dostupne našoj aplikaciji.

U projektu Malina se nalaze 2 klase koje se mapiraju u bazu podataka. To su klase Post i RegisteredUser. Na primjeru klase Post su predstavljene neke od karakteristika Ebean-a.

```
package models;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;
import com.fasterxml.jackson.databind.JsonNode;
import play.data.validation.Constraints.MaxLength;
import play.data.validation.Constraints.Required;
import play.db.ebean.Model;
import controllers.Application;
import java.util.List;

@Entity
public class Post extends Model{

    @Id
    public Long id;

    @Required
    public String url;

















    public static Model.Finder<String,Post> find = new
        Model.Finder<String,Post>(String.class, Post.class);

    @Required
    public Date postDate = new Date(System.currentTimeMillis());

    public static List<Post> listaPostova(){
        return Post.find.orderBy("postDate desc").findList();
    }
}
```

Slika 9 - Model Post

Da bi klasa bila Ebean potrebno je da nasljeđuje Model klasu, te iznad staviti anotaciju @Entity. Nekoliko je neophodnih importa i oni su vidljivi iznad definicije klase na slici. Play automatski za attribute kreira gettere i settere tako da se za to programer ne mora brinuti. Za definisanje id kolone, iznad atributa je potrebno samo dodati anotaciju @Id. Anotacija @Required specificira da kolona ne može imati vrijednost null. Ovi atributi su preslikani u tabelu istog naziva u bazi podataka. Na slici ispod je prikazan izgled tabele post u bazi podataka.

Options			id	url	post_date	
	 Edit	 Copy	 Delete	1	https://www.youtube.com/embed/_nW5AF0m9Zw	2014-09-03 18:11:04
	 Edit	 Copy	 Delete	2	https://www.youtube.com/embed/O8Uhn-dU3Gg	2014-09-04 13:57:47
	 Edit	 Copy	 Delete	3	https://www.youtube.com/embed/bIS8twlzuMw	2014-09-07 19:37:07
	 Edit	 Copy	 Delete	4	https://www.youtube.com/embed/90JmmMbFKfw	2014-09-08 13:00:50

Slika 10 – Tabela Post-ova u bazi podataka

Model.Finder je klasa koja predstavlja pomoć Ebean-u za upite prema bazi. Ona čini upite veoma jednostavnima. Recimo u metodi *listaPostova()* se kod *Post.find.orderBy("postDate desc").findList();* prevodi u upit „*SELECT * FROM 'post' ORDER BY post_date DESC;*“ te rezultat upita pretvara u listu objekata tipa Post.

Klasa RegisteredUser predstavlja registrovane korisnike. Instanca ove klase se kreira pri samom procesu registracije. Sadrži attribute koje Twitter šalje aplikaciji nakon uspješne autentifikacije, a to su name, twitterId, description i pictureUrl. Ova četiri atributa su dostupna preko Twitter OAuth-a i se šalju u vidu JSON [1] formata te se zatim smjeste u odgovarajuće attribute instance RegisteredUser klase. Pored ovih sadrži atribut Id koji kao i u slučaju Post klase predstavlja id

kolonu u bazi podataka. Zatim se vrši spašavanje u bazu putem Ebean alata. To se čini pozivom metode `save()` Ebean modela.

```
@Entity
public class RegisteredUser extends Model{

    @Id
    public Long id;

    @Required
    public String name;

    @MaxLength(value = 200)
    public String description;

    @Required
    public String pictureUrl;

    @Required
    public String twitterId;

    @Required
    public Date registrationDate = new Date(System.currentTimeMillis());

    public static String fixUrl(String url){
        url = url.replace("_normal", "");
        return url;
    }

    public static RegisteredUser fromJson(JsonNode twitterJson) {
        RegisteredUser u = new RegisteredUser();
        u.name = twitterJson.findPath("name").asText();
        u.twitterId = twitterJson.findPath("screen_name").asText();
        u.description = twitterJson.findPath("description").asText();
        u.pictureUrl = twitterJson.findPath("profile_image_url").asText();
    }
}
```

Slika 11 - Model RegisteredUser

Povezivanje sa bazom podataka

Za potrebe ovog projekta je upotrijebljena MySQL [7] baza podataka. Povezivanje sa bazom u Play framework-u je vrlo jednostavno. Tako da je i veza sa MySQL bazom veoma jednostavna za implementiranje. Potrebno je u `application.conf` fajlu dodati sljedeće linije koda:

```
db.default.driver=com.mysql.jdbc.Driver
db.default.url="jdbc:mysql://localhost:3306/malina"
db.default.user=root
# db.default.password=""
```

Ovim je definisano koji driver je potreban sa vezu sa bazom, a to je naravno mysql driver. Definisan je i url baze, gdje malina predstavlja naziv baze. Root je default-ni username u ovom slučaju za phpmyadmin, a default-ni password je prazan string.

Pored ovih linija potrebno je još jednu malu stvar uraditi da bi mysql driver funkcionisao. Potrebno je u `build.sbt` fajlu dodati dependency za mysql driver. U `build.sbt` je potrebno dodati dependency za Ebean. Na sljedećoj slici je prikazan `build.sbt` fajl i kako on treba da izgleda zajedno sa potrebnim zavisnostima.


```
import play.Project._  
  
name := ""Malina""  
  
version := "1.0-SNAPSHOT"  
  
libraryDependencies += Seq(  
  javaEbean,  
  "mysql" % "mysql-connector-java" % "5.1.26",  
  "org.webjars" %% "webjars-play" % "2.2.2",  
  "org.webjars" % "bootstrap" % "2.3.1",  
  "com.restfb" % "restfb" % "1.6.12")  
  
playJavaSettings
```

Slika 12 – Build.sbt fajl

Za kreiranje MySQL baze podataka je korišten WampServer [7] i njegovo grafičko okruženje phpmyadmin.

Dizajniranje i kreiranje pogleda (views)

U Play je ugrađen Scala-baziran način kreiranja template-a, pod nazivom Scalate [14]. On je u potpunosti inspirisan Microsoftovim ASP.NET Razor-om [7]. Kao i Razor, Play-ovi Scala template-i predstavljaju jednostavan način ubacivanja serverskog (Scala) koda u html sintaksu. Play Scala template su jednostavni tekstualni fajlovi koji sadrže male blokove Scala koda. Template sistem je napravljen tako da je jednostavan za korištenje front-end developerima. Oni se kompajliraju kao Scala funkcije, te se moguće greške prikazuju u browseru, kao i sve greške u Play projektima.

Template se kreira u views paketu app direktorija Play aplikacije. Tako na primjer ukoliko se želi kreirati template index, konvencija imenovanja template-a nalaže da će njegov naziv biti index.scala.html.

Template-ima se veoma jednostavnu mogu proslijediti parametri. Oni su zapravo nešto kao funkcije. Na vrhu template-a se moraju deklarirati parametri koje prima. Template-i se pozivaju iz kontrolera te im se pri tom pozivu prosljeđuju parametri. Na jednom primjeru će biti prikazan ovaj način prosljeđivanja parametara i pozivanje istih iz template-a.

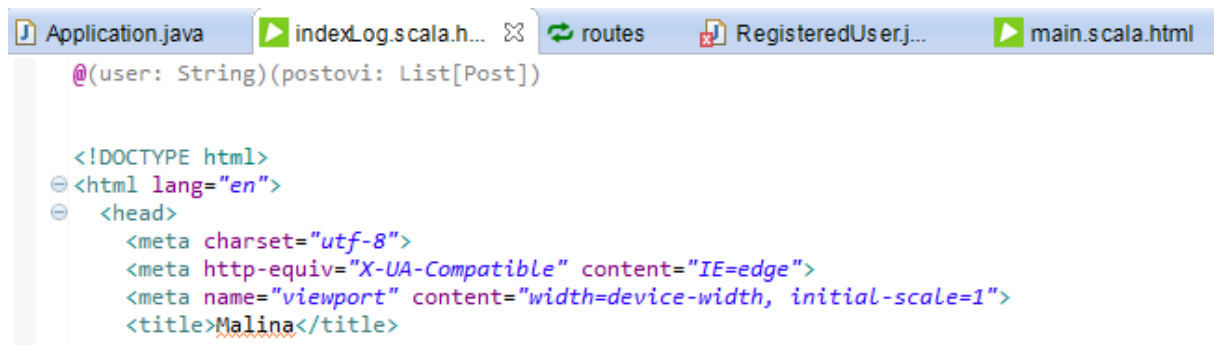
View koji se prikazuje kada se korisnik prijavi na sistem u aplikaciji Malina je indexLog.scala.html template. Kontroler naravno poziva taj template prosljeđujući mu 2 parametra, string i listu koja sadrži elemente tipa Post, koji su dobavljeni iz baze podataka i predstavljaju do tad postavljene linkove na stranicu.

Na slici ispod je metoda *indexLog()* iz kontrolera Application koja se poziva nakon što se korisnik prijavi na sistem. Dio koda iznad deklaracije metode govori da je ovo metoda koja se može pozvat samo ukoliko je korisnik prijavljen na sistem a ukoliko nije vraća korisnika na login view. Vidimo da se indexLog view template-u prosljeđuju 2 parametra. To su sesija koja predstavlja string i sadrži ime prijavljenog korisnika, tačnije njegov twitterId, te Post.listaPostova() koja kupi podatke iz tabele 'post' u bazi podataka i vraća ih kao listu podataka tipa Post.

```
@Security.Authenticated(Secured.class)
public static Result indexLog(){
    return ok(views.html.indexLog.render(sesija, Post.listaPostova()));
}
```

Slika 13 – indexLog() akcija kontrolera Application

Na sljedećoj slici se vidi dio indexLog.scala.html template-a. Na samom vrhu su definisana 2 parametra koja template prima, a to su parametri user tipa String i parametar postovi tipa List<Post>.



```
@(user: String)(postovi: List[Post])

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Malina</title>
```

Slika 14 – indexLog.scala.html template

Na sljedećoj slici vidimo kako se poziva Scala kod iz template-a. Koristimo specijalni karakter @ koji naznačava početak dinamičkog iskaza. Nismo obavezni eksplicitno zatvoriti iskaz. To je automatizirano, te kraj iskaza Scalate sam može zaključiti na osnovu koda. Na slici je primjer obične for petlje u Scala programskom jeziku koja za svaki Post p, ispisuje njegov atribut url koji predstavlja URL YouTube linka i smješta ga u src atribut iframe html taga koji kreira okvir za YouTube link.

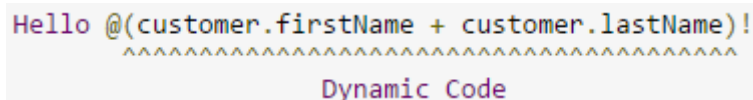
```
</div>
<div class="col-md-6 col-md-offset-3">

@for(p <- postovi){
  <iframe width="420" height="315" src=@p.url frameborder="0" allowfullscreen></iframe>
}
</div>
```

Slika 15 – Dio koda koji ispisuje Post-ove u HTML formatu

Scalate template mašina sama otkriva kraj dinamičkog izraza analizirajući kod. S toga ova sintaksa dozvoljava samo jednostavne iskaze. Ukoliko se želi malo složeniji iskaz ubaciti u html kod, programer je dužan koristiti zagrade () [4].

Npr.



```
Hello @(customer.firstName + customer.lastName)!
*****
Dynamic Code
```

Slika 16 – Primjer složenijeg izraza napisanog u Scali koji se može ubaciti u HTML kod

Na stranici Scalate je dat zanimljiv opis ovog Scala Template motora. „SCALATE – Scala Template Engine: like JSP without the crap but with added Scala coolness“.

Statički resursi

Kao što je ranije spomenuto, CSS i JS fajlovi se nalaze u public direktoriju našeg projekta u stylesheets i javascripts poddirektorijima respektivno. Isto je i sa slikama koje se nalaze u poddirektoriju images. Play uslužuje zahtjeve prema ovim statičkim resursima kao bilo koji drugi HTTP zahtjev. Koristi isti način usmjeravanja koji koriste i drugi resursi i definisan je u routes fajlu. Play ima ugrađen Assets kontroler koji brine o usluživanju zahtjeva za statičke resurse. Dio koda u routes fajlu koji se odnosi na Assets kontroler je na slici ispod.

```
# Map static resources from the /public folder to the /assets URL path
GET    /assets/*file controllers.Assets.at(path="/public", file)
```

Slika 17 – Assets kontroler u routes fajlu

Ukoliko se u browser unese URI `localhost:9000/assets/images/malina.png`, time se kreira HTTP GET zahtjev za ovaj statički resurs te će browser prikazati tu sliku. Naravno potrebno je da ta slika postoji u navedenom direktoriju.

Kao i za svaki kontroler mapiran u routes fajlu, i za Assets kontroler se kreira inverzni kontroler u `controllers.routes.Assets`. Ovo omogućava obrnuto usmjeravanja (reverse routing). Ukoliko se iz Scala template-a želi pozvati neki statički resurs iz public foldera, npr. `bootstrap.min.css` iz poddirektorija `stylesheets`, potreban je sljedeći kod:

```
<script src="@routes.Assets.at("javascripts/bootstrap.min.js")"></script>
```

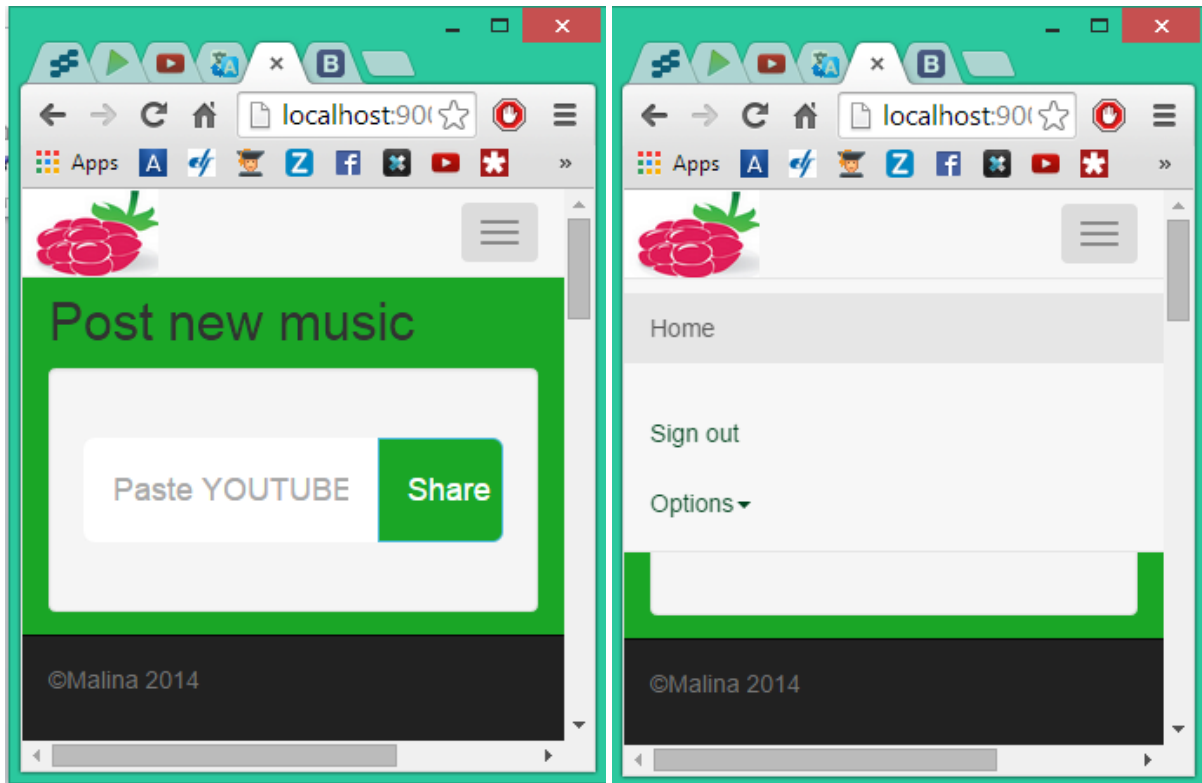
Slika 18 – Pozivanje bootstrap.min.js resursa preko inverznog rutiranja

Ovo se prevodi u:

```
<script src="/assets/javascripts/bootstrap.min.js"></script>
```

Slika 19 Pozivanje bootstrap.min.js resursa bez inverznog rutiranja

Za kreiranje samog dizajna stranice korišten je Bootstrap [15] te njegovog CSS i JavaScript skupa fajlova. Na slici iznad je prikazano importovanje bootstrapovog JavaScript fajla u jedan od template-a. Da nije korišten Bootstrap za front-end dizajn, kontrole na stranici ne bi bilo jednako dobro posložene kao što je to postignuto uz pomoć Bootstrap-a. Upotreba Bootstrap-a je osigurala pristojan izgled stranice i pri umanjenoj veličini prozora. Na slikama ispod su prikazani maksimalno umanjeni prozor sa otvorenom stranicom Malina.



Slika 20 – Izgled aplikacije Malina pri maksimalno umanjenim prozorima

Ovakav izgled umanjenog prozora ne bi bio ovakav kakav jest da nije bilo Bootstrap-ovih CSS i JS fajlova. Vjerovatno ništa ne bi bilo dovoljno vidljivo kad bi se prozor smanjio do mjere koliko je umanjen na slici iznad. Riječ je o takozvanom „responsive“ dizajnu, tako da se aplikacija tj. njen dizajn napravi prilagodljivijim za različite veličine ekrana, što je za web aplikacije neophodno.

Kontroleri (controllers)

Ovdje neće biti riječi o kontrolerima u MVC smislu. Bit će riječi o kontrolerima ali konkretno u Play framework-u. Kontroleri predstavljaju vezu između modela i HTTP protokola. Ponašaju se kao ljepljivo između modela i transportnog sloja.

Kontroler je ništa više nego klasa koja nasljeđuje `play.mvc.Controller` apstraktnu klasu i implementira nekoliko akcija. Akcija je metoda u kontroleru koja obrađuje parametre zahtjeva i proizvodi rezultat koji se šalje klijentu. Većina zahtjeva koje primi Play aplikacija se obrađuju akcijama. Akcija vraća `play.mvc.Result` vrijednost, koja predstavlja HTTP odgovor koji će se poslati klijentu. Na slici ispod je primjer jednog veoma jednostavnog kontrolera koji ima jednu akciju i ona se zove `index()` [4].

```
package controllers;

import play.*;
import play.mvc.*;

public class Application extends Controller {

    public static Result index() {
        return ok("It works!");
    }

}
```

Slika 21 – Primjer jednostavnog kontrolera sa jednom akcijom `index()`

Ova akcija će klijentu vratiti 200 OK HTTP odgovor te će se u browseru ispisati „It works!“. Naravno potrebno je da u routes fajlu postoji index metoda mapirana sa odgovarajućim URI-em, tj. da postoji linija u routes fajlu kao npr „GET / controllers.Application.index()“.

Akcije vraćaju Result vrijednost. Result vrijednost je HTTP odgovor sa statusnim kodom, nizom HTTP zaglavlja i tijelom koje se šalje klijentu. Ove vrijednosti su definisane u `play.mvc.Result` i `play.mvc.Results` klasama. `play.mvc.Results` klasa nudi nekoliko metoda za kreiranje HTTP odgovora kao što je naprimjer gore iskorištena `ok()` metoda koja vraća statusni kod 200, koji označava uspješno obrađen HTTP zahtjev. Na slici ispod je nekoliko metoda klase `play.mvc.Results` koje kreiraju različite vrijednosti tipa `Response` [4].

```
Result ok = ok("Hello world!");
Result notFound = notFound();
Result pageNotFound = notFound("<h1>Page not found</h1>").as("text/html");
Result badRequest = badRequest(views.html.form.render(formWithErrors));
Result oops = internalServerError("Oops");
Result anyStatus = status(488, "Strange response type");
```

Slika 22 – Metode klase `play.mvc.Results`

Ukoliko bi se web preglednik htio preusmjeriti na novi URL u akciji `index()`, potrebno bi bilo izmijeniti liniju koda `return ok("It Works!");` u npr. `return redirect("/user/home");`. Ovo će klijenta preusmjeriti na URI `/user/home`. I u ovom slučaju se podrazumijeva da u routes fajlu postoji akcija mapirana za ovaj URI. Preusmjeravanje je moguće i na drugu akciju. Slika ispod prikazuje akciju `logout()` iz aplikacije Malina kontrolera `Application`. Ovdje se browser preusmjerava na akciju `login()`, te se korisniku prikazuje interfejs za prijavu.

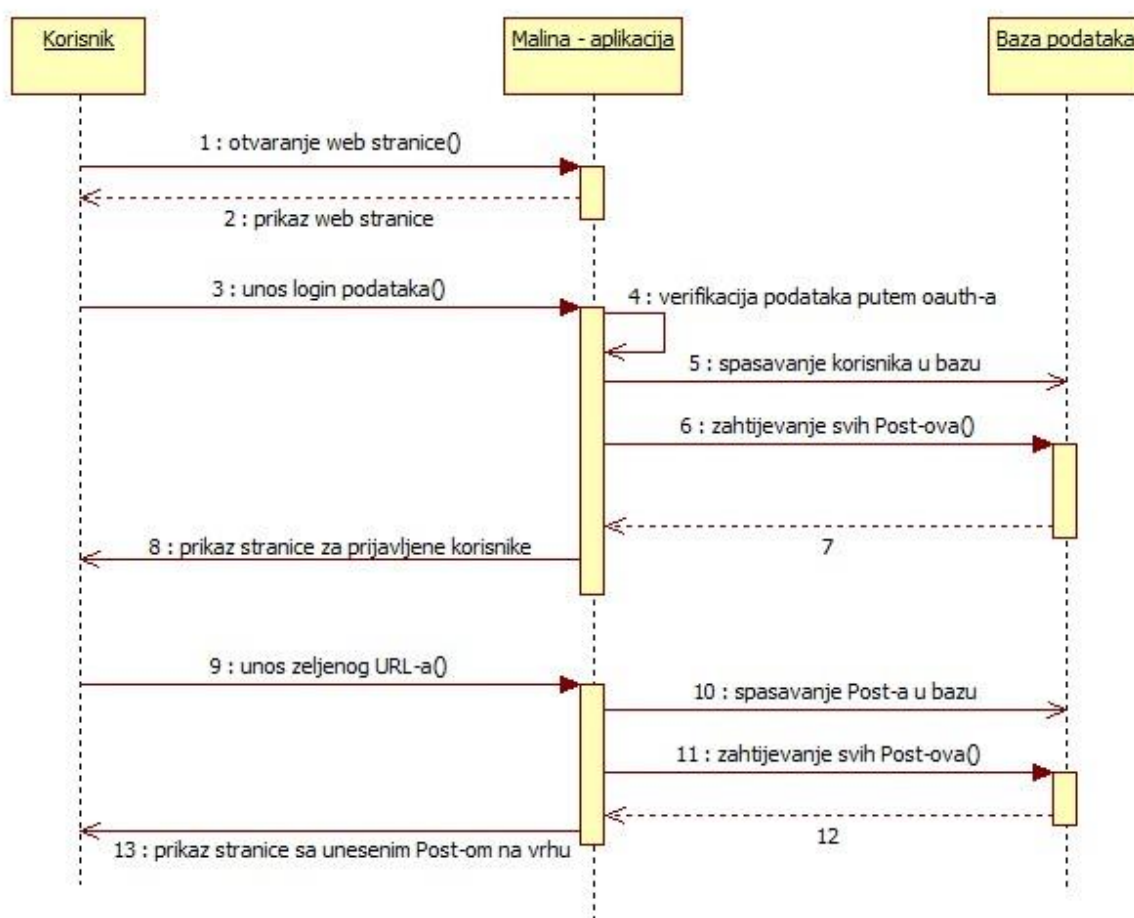
```
public static Result logout() {
    sesija="";
    session().clear();
    flash("success", "You've been logged out");
    flash("twitterId", "");
    return redirect(
        routes.Application.login()
    );
}
```

Slika 23 – `logout()` akcija kontrolera `Application`

U aplikaciji Malina su kreirana 2 kontrolera. To su Application.java i Secured.java kontroleri. Application kontroler je glavni kontroler, ako se smijem tako izraziti. U njemu je implementirana većina akcija koje obrađuju zahtjeve.

Primjer ponašanja kontrolera

Kroz primjer toka radnje koji se dešava kada korisnik želi da podijeli novi sadržaj na zajedničkom dashboard-u će biti prikazane i opisane neke od akcija implementiranih u Application kontroleru. Tok operacija potrebnih da se izvrše za kompletiranje ove akcije će biti prikazan na dijagramu sekvence.



Slika 24 – Dijagram sekvence – postavljanje novog Post-a od strane korisnika

Korisnik otvarajući web stranicu šalje GET zahtjev za / serveru. U routes fajlu je taj URI mapiran sa akcijom kontrolera Application koja se zove `index()`.

```
GET      /                                controllers.Application.index()
```

Slika 25 – Mapiranje akcije `index()` u routes fajlu

To znači da će se akcija `index()` pobrinuti za stvaranje odgovora koji se vraća klijentu, tj. web browseru.

```
public static Result index() {  
    if(isLogged())return redirect(routes.Application.indexLog());  
    return ok(views.html.index.render("Malina"));  
}
```

Slika 26 – Implementacija akcije `index()` u kontroleru `Application`

Vidimo da ova metoda preusmjerava korisnika na `indexLog()` akciju koja vraća home page ali za prijavljene korisnike. U slučaju da korisnik nije prijavljen ova akcija vraća `Result` kreiran pomoću metode `ok()` kojoj se proslijeđuje `index.scala.html` view-u.

Zatim korisnik klikom na Sign in dugme otvara login view. Time se ustvari serveru šalje zahtjev GET za URI `/login`.

```
GET    /login                                controllers.Application.Login()
```

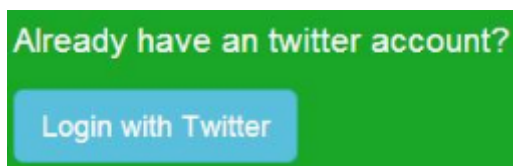
Slika 27 – Mapiranje akcije `login()` u routes fajlu

Akcija `login()` kontrolera `Application` je dužna kreirati odgovor koji će se proslijediti klijentu.

```
public static Result login() {  
    if(isLogged())return redirect(routes.Application.indexLog());  
    return ok(views.html.login.render("Malina"));  
}
```

Slika 28 – Implementacija akcije `login()` u kontroleru `Application`

Korisnik zatim ukoliko već ima Twitter korisnički račun klikom na dugme „Login with Twitter“ otvara stranicu gdje je dužan da unese korisničke podatke i loguje se u aplikaciju pomoću svog Twitter korisničkog računa.



Slika 29 – Dugme `Login with Twitter` koje poziva akciju `register()`

Klik na ovo dugme usmjerava na URI `/register`. Izgled mapiranja ovog URI-ja sa akcijom kontrolera u routes fajlu izgleda kao na slici ispod:

```
GET    /register                                controllers.Application.register()
```

Slika 30 – Mapiranje akcije `register()` u routes fajlu

Dok akcija *register()* kontrolera Application izgleda ovako:

```
public static Result register(){
    if(isLogged())
        return redirect(routes.Application.indexLog());
    else{
        String callbackUrl = routes.Application.registerCallback().absoluteURL(request());
        Tuple<String, RequestToken> t = service.retrieveRequestToken(callbackUrl);
        flash("request_token", t._2.token);
        flash("request_secret", t._2.secret);
        return redirect(t._1);
    }
}
```

Slika 31 – Implementacija akcije *register()* u kontroleru Application

Akcija *register()* prvo u slučaju da je korisnik već prijavljen preusmjerava korisnika na *indexLog()* akciju koja vraća home page za prijavljene korisnike. U suprotnom slučaju će dohvatiti RequestToken te nakon spašavanja informacija iz tokena u flash cookie, proslijediti korisnika na *registerCallback()* akciju.

Akcija *registerCallback()* je zadužena za instanciranje modela RegisteredUser od parametara poslanih od strane Twitter OAuth provajdera. U sklopu iste se i instanca tog RegisteredUser-a spašava u bazu podataka, ukoliko se korisnik po prvi put prijavljuje u aplikaciju Malina. Nakon spašavanja korisnika u bazu, izvršava se preusmjeravanje na akciju *indexLog()* koja vraća homepage za prijavljene korisnike. Nešto složeniji kod je sadržan u akciji *registerCallback()* te se neće ulaziti u sve detalje. Akcija *registerCallback()* je prikazana na slici ispod:

```
public static Result registerCallback(){
    RequestToken token = new RequestToken(flash("request_token"), flash("request_secret"));
    String authVerifier = request().getQueryString("oauth_verifier");
    Promise<JsonNode> userProfile = service.registeredUserProfile(token, authVerifier);
    userProfile.onRedeem(new Callback<JsonNode>(){

        @Override
        public void invoke(JsonNode twitterJson) throws Throwable {
            RegisteredUser user = RegisteredUser.fromJson(twitterJson);
            RegisteredUser reguser = RegisteredUser.searchByTwitterId(user.twitterId);
            sesija= user.twitterId;
            if(!user.twitterId.equals(reguser.twitterId) && !reguser.twitterId.equals(""))
                user.save();
            Application.saveSession(user.twitterId);
        }

    });
    RegisteredUser zadnji = RegisteredUser.lastUser();
    Application.saveSession(zadnji.twitterId);
    return redirect(routes.Application.indexLog());
}
```

Slika 32 – Implementacija akcije *registerCallback()* u kontroleru Application

Akcija *indexLog()* je akcija koja klijentu vraća home page za prijavljene korisnike, kao što je već rečeno. Njena implementacija izgleda ovako:

```
@Security.Authenticated(Secured.class)
public static Result indexLog(){
    return ok(views.html.indexLog.render(sesija, Post.listaPostova()));
}
```

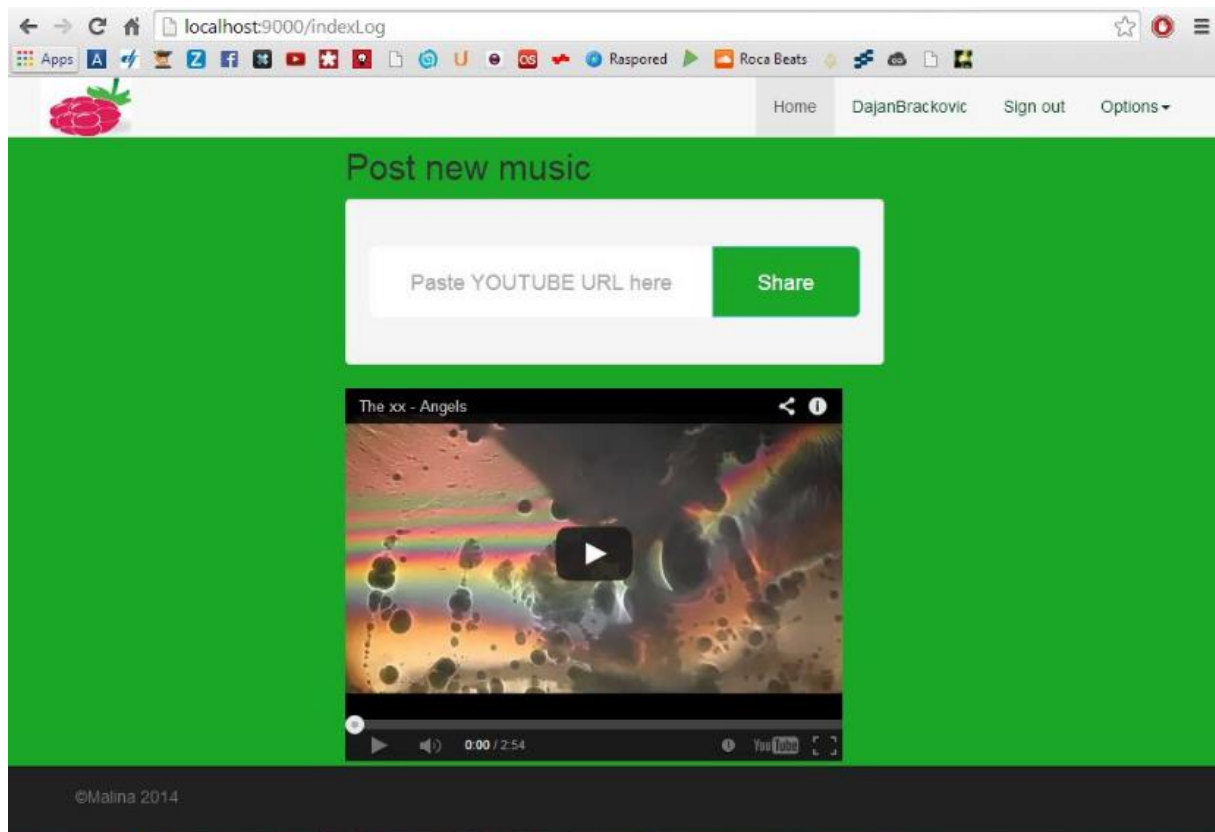
Slika 33 – Implementacija akcije *indexLog()* u kontroleru *Application*

Dok dio routes fajla koji se odnosi na *indexLog* akciju izgleda ovako:

```
GET    /indexLog                               controllers.Application.indexLog()
```

Slika 34 – Mapiranje akcije *indexLog()* u routes fajlu

Akcija *indexLog()* vraća klijentu *indexLog.scala.html* view na kojem su prikazani svi Post-ovi prikupljeni iz baze podataka te oni predstavljaju do tad objavljene Post-ove. To izgleda ovako:



Slika 35 – Home page za prijavljene korisnike (*indexLog.scala.html* view)

Kada korisnik unese željeni URL u polje za unos te zatim klikne na dugme Share, browser upućuje POST zahtjev serveru na URI */newPost*. Mapiranje ovog URI-a u routes fajlu izgleda ovako:

```
POST    /newPost                               controllers.Application.newPost()
```

Slika 36 – Mapiranje akcije *newPost()* u routes fajlu

Vidimo da se akcija `newPost()` brine za usluživanje ovog zahtjeva. Ona je definisana u kontroleru `Application` i izgleda ovako:

```
@Security.Authenticated(Secured.class)
public static Result newPost(){
    DynamicForm requestData = Form.form().bindFromRequest();
    String link = requestData.get("link");
    String code = link.substring(link.lastIndexOf('=')+1);
    Post post = new Post();
    post.url = "https://www.youtube.com/embed/"+code;
    post.postDate = new Date(System.currentTimeMillis());
    post.save();
    return redirect(routes.Application.indexLog());
}
```

Slika 37 – Implementacija akcije `newPost()` u kontroleru `Application`

Prva linija koda u definiciji akcije predstavlja način na koji se nakon submit-anja forme, preuzimaju njeni parametri iz zahtjeva. Za to je zadužena `bindFromRequest()` metoda klase `play.data.Form`. Zatim se prerađeni parametar zajedno sa trenutnim datumom smješta u objekat tipa `Post`, koji se na kraju spašava u bazu. Na kraju se vrši preusmjeravanje na `indexLog()` akciju koja kupi sve `Post`-ove iz baze i reda ih po datumu od najnovijeg do najstarijeg. To se na kraju i prikazuje u browseru. U `indexLog.scala.html` view-u forma za postavljanje `Post`-a izgleda ovako:

```
<form method="POST" action="@routes.Application.newPost()">
  <div class="input-group">
    <input class="btn btn-lg" name="link" id="link" type="text" placeholder="Paste YOUTUBE URL here" required>
    <a href="#"><button class="btn btn-info btn-lg" type="submit" onclick="">Share</button>
  </div>
</form>
```

Slika 38 – Forma za postavljanje novog `Post`-a u `indexLog.scala.html` view-u

Security.Authenticator kontroler

Drugi kontroler implementiran za potrebe ove aplikacije se zove Secured i nasljeđuje klasu Security.Authenticator [10]. Na slikama na kojima su prikazane implementacije akcija iz kontrolera Application vidi se da iznad nekih stoji anotacija @Security.Authenticated(Secured.class). Ova anotacija obilježava akcije koje mogu usluživati samo prijavljene korisnike. Kada se pozove akcija označena sa anotacijom @Security.Authenticated(Secured.class), pozvat će se metoda getUsername(http.Context ctx) iz Security.Authenticator klase koja će pokušati pročitati „twitterId“ atribut iz sesije. Ukoliko je atribut tu, akcija će se izvršiti. Ako atributa nema vratit će se odgovor servera sa 401 errorom. Na slici ispod je implementacija kontrolera Secured.

```
public class Secured extends Security.Authenticator {  
  
    @Override  
    public String getUsername(Context ctx) {  
        return ctx.session().get("twitterId");  
    }  
  
    @Override  
    public Result onUnauthorized(Context ctx) {  
        return redirect(routes.Application.login());  
    }  
}
```

Slika 39 – Secured kontroler

Testiranje u Play framework-u

Testovi u Play aplikacijama se stavljaju u test direktorij. Ovaj direktorij se kompajlira jedino u slučaju da se aplikacija pokrene u test načinu rada. Mogu se napraviti 3 vrste testova u Play framework-u:

Unit testovi [2] se prave pomoću JUnit framework-a. JUnit je unit-test framework za Java programski jezik. Unit testovi predstavljaju testiranje pojedinačnih komponenti softvera i utvrđivanje da li iste rade adekvatno sa svim očekivanim tipovima ulaza.

Funkcionalni testovi [2] se prave poslije integracijskih testova i provjeravaju da li integrisani sistem izvršava funkcije opisane u specifikaciji zahtjeva. Funkcionalni testovi se također pišu koristeći JUnit framework.

Acceptance testovi [2] predstavljaju testove prihvatljivosti. Na play stranici ove testove zovu Selenium testovi jer se pišu koristeći Selenium framework. Testovi prihvatljivosti provjeravaju da li sistem ispunjava zahtjeve kupca i omogućuju kupcu da vidi da li je dobio ono što traži. Razlikuju se od ostalih testova u tome što se testovi prihvatljivosti prave na osnovu korisničkih priča ili user-stories. User-stories pišu korisnici kao primjere opisivanja stvari koje sistem treba da radi po njihovom mišljenju. Na osnovu toga se kasnije pišu testovi prihvatljivosti.

Aplikaciju Malina je izgrađena na template-u koji su bili ugrađeni neki JUnit testovi. To su 3 JUnit testa. Prvi se zove simpleTest(). Testira jednu banalnu stvar. Provjerava da li su 2 i 2 jednaki. Druga 2 testa provjeravaju da li index.scala.html template sadrži određeni String. Prvi od ta 2 testira da li template sadrži String koji mu je proslijeđen kao parametar. Da budem konkretan taj String je „Malina“. Treći test, ili drugi od dva testa koji testiraju prisustvo Stringa u index.scala.html template-u, provjerava content-type, charset i statusni kod odgovora servera.

To je vidljivo na slici ispod:

```
public class ApplicationTest {

    @Test
    public void simpleCheck() {
        int a = 1 + 1;
        assertThat(a).isEqualTo(2);
    }

    @Test
    public void indexTemplateShouldContainTheStringThatIsPassedToIt() {
        running(fakeApplication(), new Runnable() {
            public void run() {
                Content html = views.html.index.render("Malina");
                assertThat(contentType(html)).isEqualTo("text/html");
                assertThat(contentAsString(html)).contains("Malina");
            }
        });
    }

    @Test
    public void indexShouldContainTheCorrectString() {
        running(fakeApplication(), new Runnable() {
            public void run() {
                Result result = callAction(routes.ref.Application.index());
                assertThat(status(result)).isEqualTo(OK);
                assertThat(contentType(result)).isEqualTo("text/html");
                assertThat(charset(result)).isEqualTo("utf-8");
            }
        });
    }
}
```

Slika 40 – JUnit testovi u Play framework-u

Kao ugrađen u jezgro template-a na osnovu kojeg je napravljena aplikacija došao je i jedan integracijski test. Za njega je neophodno da aplikacija bude pokrenuta te koristi metodu *fakeApplication()* kojom se to simulira.

```
@Test
public void test() {
    running(testServer(3333, fakeApplication(inMemoryDatabase())), HTMLUNIT, new Callback<TestBrowser>() {
        public void invoke(TestBrowser browser) {
            browser.goTo("http://localhost:3333");
            assertThat(browser.pageSource()).contains("Malina");
        }
    });
}
```

Slika 41 – Integracijski test

Testove je moguće pokrenuti putem konzole pomoću komande *test* ili *test-only*. Moguće je to naravno uraditi i putem GUI-a u web browseru koji podržava testiranje aplikacije.

Na sljedećoj slici se vide uspješno pokrenuti testovi putem konzole:

```
[Malinal] $ test
[info] Compiling 2 Java sources to C:\play\activator-1.2.3\Malina\target\scala-2
.10\test-classes...
[info] IntegrationTest
[info] + test
[info]
[info]
[info] Total for test IntegrationTest
[info] Finished in 0.198 seconds
[info] 1 tests, 0 failures, 0 errors
[info] ApplicationTest
[info] + simpleCheck
[info] + indexShouldContainTheCorrectString
[info] + indexTemplateShouldContainTheStringThatIsPassedToIt
[info]
[info]
[info] Total for test ApplicationTest
[info] Finished in 0.009 seconds
[info] 3 tests, 0 failures, 0 errors
[info] Passed: Total 4, Failed 0, Errors 0, Passed 4
[success] Total time: 63 s, completed Sep 12, 2014 5:59:44 PM
```

Slika 42 – Prikaz pokretanja testova u konzoli

Zaključak

Rezultat ovog rada je jedna jednostavna muzička društvena mreža pod nazivom Malina. Mogu reći da aplikacija što dizajno što svojom jednostavnošću odgovara nazivu Malina.

Iako se čitanjem dokumenta da zaključiti da je bio poprilično jednostavan posao implementirati aplikaciju Malina. Međutim Play može ponuditi dosta stvari koje mogu zadati probleme, pa čak i od samog početka i instaliranja Play framework-a. Ali kada se programer uhoda te upozna sa Play-om, može se zaključiti da Play nudi jedan vrlo moderan i jednostavan razvoj web aplikacija.

Jedan od problema Play framework-a je to što je izašlo mnogo verzija istog i što one nastavljaju izlaziti. Zbog te činjenice je ponekad teško doći do dokumentacije za određenu stvar koja vam predstavlja problem. Tokom razvoja je moguće da se ne može ili je vrlo teško doći do informacija za rješavanje određenog problema iz razloga što ima veliki broj verzija i po 2 tipa dokumentacije za svaku verziju, dokumentacija za Javu i dokumentacija za Scalu. Play nudi opcije integrisanja raznih tehnologija i alata. Tako da dokumentacija za integraciju svih tih mogućih tehnologija nije još u potpunosti sastavljena. Play je nešto novije u svijetu web aplikacija, te je sigurno stackoverflow podrška za Play sigurno siromašnija u poređenju sa podrškom za neke druge tehnologije. Ali to se svakodnevno poboljšava, jer mislim da Play zaista ima potencijala. Potrebno je zaista malo vremena da se uhodate u sve što se tiče Play-a, da bi ustvari shvatili da je prilično jednostavno.

Dokument pored malo detaljnije opisanih razvijenih funkcionalnosti u aplikaciji Malina, sadrži i neke osnove Play framework-a, karakteristike, prednosti istog, te i neke mane. Također instaliranje Play framework-a je opisano u dokumentu, kao i integracija Play2 plugina za Eclipse Scala IDE.

Dokument opisuje i neke osnovne teoretske aspekte povezane sa Play framework-om kao što su npr. Scala, MVC, Ebean i drugi. Nažalost nisam imao prostora i vremena da praktično ispitam sve opcije koje Play zaista pruža i nudi svojim korisnicima. Pokušao sam ih bar iz teoretskog aspekta približiti onome ko bude čitao ovo.

Mislim da ovaj dokument može poslužiti i kao priručnik nekom ko ranije nije razvijao u Play framework-u. Ja sam upravo bio takva osoba. Pored informacija koje sam našao u Play dokumentaciji i na internetu generalno, mislim da sam navodio i neke stvari koje su konkretno meni predstavljale probleme a nisu toliko bile dostupne na internetu.

Pojmovi i skraćenice

Pojam/skraćenica	Značenje
HTTP	Hypertext Transfer Protocol – protokol za pristup podacima na World Wide Web-u. To je protokol aplikacionog nivoa TCP/IP protokola
WWW	World Wide Web – sistem međusobno povezanih hipertekst dokumenata kojima se pristupa putem Interneta
CSS	Cascading Style Sheets – stilski jezik koji služi za opis i definisanje prezentacije dokumenata napisanih u HTML jeziku
HTML	HyperText Markup Language – prezentacijski jezik za izradu web stranica
JavaScript	Dinamički programski jezik. Koristi se kao dio web browsera i omogućava skripte na klijentskoj strani koje služe za interakciju sa korisnikom.
CoffeeScript	Programski jezik koji se prekompajlira u Javascript. Imaju istu ulogu. CoffeeScript se može shvatiti kao modernija verzija JavaScripta.
LESS	Dinamički stilski jezik. Ima neke mogućnosti koje CSS nema, a to su npr. varijable.
MySQL	Druga na svijetu najčešće korištena open-source relaciona baza podataka.
MVC	Model View Controller – softverski arhitekturni patern koji razdvaja prezentaciju, interakciju i podatke.
Eclipse IDE	Jedan od najpoznatijih i najkorištenijih open-source okruženja za razvoj Java baziranih aplikacija.
IntelliJ	Razvojno okruženje kao i Eclipse. Ima 2 verzije, community i ultimate. Community je free dok ultimate nije.
JBoss Netty	Klijent- server framework za razvoj Java mrežnih aplikacija kao što su protokol serveri i klijenti.
ORM	Object Relational Mapping – mapiranje objekata iz objektno-orijentiranih u tip objekta potrebnih za spašavanje u bazu podataka.
LinkedIn	Biznis-orijentirana društvena mreža. Nastala je 2002. godine.
Akka	Open-source alat za razvoj distribuiranih i visoko konkurentnih aplikacija na JVM.
GUI	Graphical User Interface – grafički korisnički interfejs. Omogućava interakciju korisnika sa elektroničnim uređajima putem grafičkih ikona
Java SE Development Kit.	Skup alata za programiranje u Java programskom Java virtuelnu mašinu.
JUnit	Framework za implementaciju Unit testiranja u Java programskom jeziku

GET	Jedna od metoda HTTP protokola koje označavaju koja će se akcija izvršiti nad određenim resursom. Koristi se za dobavljanje resursa sa servera.
POST	Metoda HTTP protokola kao i GET, međutim koristi se za dostavu podataka na određeni lokaciju.
URI	Uniform Resource Identifier – niz karaktera koji predstavlja ime određenog resursa. Koriste ga mnogi mrežni protokoli.
Twitter	Društvena mreža koja omogućava korisnicima da šalju i čitaju 140 karaktera duge poruke zvane „tweet-ovi“
OAuth	Standard za autorizaciju. Omogućava klijentskim aplikacijama siguran pristup serverskim resursima pod dozvolom vlasnika resursa.
Java	Odnosi se na 2 pojma. Može predstavljati i programski jezik, jedan od najpoznatijih i najkorištenijih. S druge strane može predstavljati skup softverskih proizvoda i specifikacija stvorenih od strane Oracle kompanije.
API key	Predstavlja isto što i javni ključ u kriptografiji asimetričnog šifriranja.
API secret	Predstavlja isto što i privatni ključ u kriptografiji asimetričnog šifriranja.
ASP.NET	Framework za razvoj dinamičkih web aplikacija. Razvijen od strane Microsoft-a za razvoj web stranica, web aplikacija i web servisa.
Razor	Jednostavna programerska sintaksa za ubacivanje serverskog koda u HTML za ASP.NET web aplikacije.
SCALATE	Isto značenje kao i Razor, samo za Java programski jezik i Java web aplikacije.
JSP	Java Server Pages – tehnologija za razvoj dinamičkih web stranica. Sličan je PHP-u. Zasniva se na ubacivanju serverskog koda u HTML.
Framework	Univerzalna ponovno iskoristiva platforma za razvoj softverskih aplikacija, rješenja i proizvoda.
Submit	Predstavlja akciju slanja podataka iz HTML forme serveru
Homepage	Označava glavnu web stranicu neke internet stranice. Često se koristi i izraz index page.
charset	Atribut taga meta u HTML jeziku. Da bi browser prikazao stranicu kako treba, mora znati set karaktera koji stranica koristi. To je definisano u tagu meta atributom charset.
Content-type	Zaglavlje u HTTP poruci, određuje vrstu sadržaja koju server šalje klijentu.
JSON	JavaScript Object Notation – format podataka neovisan od programskog jezika. Jednostavan

	ljudima za čitanje i pisanje, a jednostavan i računarima za obrađivanje.
Developer	Programer. Web developer je izraz koji se koristi za programera koji razvija web aplikacije.
Default	Označava vrijednost ili svojstvo dodijeljeno automatski softverskoj komponenti, programu, varijabli ili uređaju.

Indeks slika

Indeks slike	Opis
Slika 1	GUI u browseru za kreiranje Play projekta
Slika 2	Activator GUI za upravljanje Play projektom
Slika 3	Prikaz koda u Activator GUI-u u browseru
Slika 4	Sadržaj Play projekta
Slika 5	Routes fajl
Slika 6	Dijagram slučajeva upotrebe za korisnika
Slika 7	Aplikacija Malina registrovana na Twitter-u
Slika 8	Tok operacije prijavljivanja preko Twitter OAuth servisa
Slika 9	Model Post
Slika 10	Tabela Post-ova u bazi podataka
Slika 11	Model RegisteredUser
Slika 12	Build.sbt fajl
Slika 13	indexLog() akcija kontrolera Application
Slika 14	indexLog.scala.html template
Slika 15	Dio koda koji ispisuje Post-ove u HTML formatu
Slika 16	Primjer složenijeg izraza napisanog u Scali koji s može ubaciti u HTML kod
Slika 17	Assets kontroler u routes fajlu
Slika 18	Pozivanje bootstrap.min.js resursa preko inverznog rutiranja
Slika 19	Pozivanje bootstrap.min.js resursa bez inverznog rutiranja
Slika 20	Izgled aplikacije Malina pri maksimalno umanjenim prozorima
Slika 21	Primjer jednostavnog kontrolera sa jednom akcijom index()
Slika 22	Metode klase play.mvc.Results
Slika 23	logout() akcija kontrolera Application
Slika 24	Dijagram sekvence – postavljanje novog Post-a od strane korisnika
Slika 25	Mapiranje akcije index() u routes fajlu
Slika 26	Implementacija akcije index() u kontroleru Application
Slika 27	Mapiranje akcije login() u routes fajlu
Slika 28	Implementacija akcije login() u kontroleru Application
Slika 29	Dugme Login with Twitter koje poziva akciju register()
Slika 30	Mapiranje akcije register() u routes fajlu
Slika 31	Implementacija akcije register() u kontroleru Application
Slika 32	Implementacija akcije registerCallback() u kontroleru Application
Slika 33	Implementacija akcije indexLog() u kontroleru

	Application
Slika 34	Mapiranje akcije indexLog() u routes fajlu
Slika 35	Homepage za prijavljene korisnike (indexLog.scala.html view)
Slika 36	Mapiranje akcije newPost() u routes fajlu
Slika 37	Implementacija akcije newPost() u kontroleru Application
Slika 38	Forma za postavljanje novog Post-a u indexLog.scala.html view-u
Slika 39	Secured kontroler
Slika 40	JUnit testovi u Play framework-u
Slika 41	Integracijski test
Slika 42	Prikaz pokretanja testova u konzoli

Literatura

Knjige i predavanja:

1. Samir Ribić, Web tehnologije, Kemigrafika Trade d.o.o., Sarajevo, 2014

Linkovi:

2. Dženana Đonko, 2014, Pouzdanost i kontrola kvalitete softvera - <http://c2.etf.unsa.ba/course/view.php?id=121>
3. Novica Nosović, 2014, Softverski inženjering - <http://c2.etf.unsa.ba/course/view.php?id=118>
4. Dokumentacija za Play framework - <https://www.playframework.com/documentation/2.3.x/Home>
5. Stackoverflow podrška za Play - <https://stackoverflow.com/questions/tagged/playframework>
6. Set vrlo korisnih video tutorijala za razvijanje Java Play aplikacije - <https://typesafe.com/how/online-training/play-java>
7. Wikipedia - <http://en.wikipedia.org/wiki/Wikipedia>
8. Twitter OAuth - <https://dev.twitter.com/oauth>
9. W3Schools - <http://www.w3schools.com>
10. Autentifikacija u Play framework-u - <http://alexgaribay.com/2014/06/16/authentication-in-play-framework-using-java/>
11. WampServer - <http://www.wampserver.com/en/>
12. Akka - <http://akka.io/>
13. Play2 plugin - <http://scala-ide.org/docs/tutorials/play/>
14. Scalate - <http://scalate.fusesource.org/>
15. Bootstrap - <http://getbootstrap.com/>