Uvod

U dvadeset I prvom veku, u doba tehnologije I napretka, željom ka povezivanju I shvatanju, na obalu je isplivala nova senzacija: društvena mreža. Upravo to je ono što je bilo potrebno. Sada na pomaku mobilnih telefona, tableta, prenosivih I mobilnih mašina, društvene mreže su pridobile na značaju. Više od par milijardi ljudi na svetu ih koristi na neki način. Bilo to održavanje kontakata preko *Facebook-*a, konstantno fotografisanje života I stila na *Instagram*-u, ili samo puko prepucavanje, pametovanje I iznošenje mišljenja I vesti na *Twitter*-u. Svi ljudi su danas povezani I kontakt je jednostavniji.

Društvene ili socijalne mreže su online web servisi koji korisnicima omogućavaju raznovrsne vidove komunikacije i mogućnost lične prezentacije. Ovo su neke od trenutno najpopularnijih društvenih mreža koje su zanimljive za većinu korisnika sa naših prostora: Facebook, Twitter, Instagram, Google+, LinkedIn…

Prisutnost na većini društvenih mreža je besplatan. Na nekim društvenim mrežama postoje opcije koje se plaćaju, ali izbor zavisi od vas jer većina besplatnih mogućnosti u potpunosti mogu zadovoljiti vaše potrebe.

U današnje vreme skoro je nemoguće zamisliti internet poslovanje ili promociju bez prisustva na društvenim mrežama. One su veoma bitne za promociju proizvoda i usluga, brediranje, ali i za poboljšavanje SEO optimizacije vašeg web sajta. Takođe, aktivnim prisustvom na društvenim mrežama možete drastično uvećati broj potencijalnih klijenata i na taj način ih zainteresovati za Vaše proizvode ili usluge.

Društvene mreže mogu biti odličan način kako da posetioce najbolje i najbrže informišete za značajne događaje, aktuelne promocije i novosti vezane za vaše poslovanje ili za proizvode ili usluge koje nudite. Kvalitetnim sadržajem tekstova koje objavljujete možete edukovati vaše klijente ili posetioce ili im pružiti određene savete. Mnogi posetioci će upravo baš preko društvenih mreža od vas zatražiti dodatne informacije ili se raspitati o drugim bitnim aspektima vezanim za vašu delatnost.

Značajan uticaj na uspeh vašeg poslovanja ili na vašu promociju doprineće vaša kreativnost i redovna aktivnost na društvenim mrežama. Veoma je bitno da ne zapostavite ovaj vid promocije i marketinga koji će vam za minimalno ulaganje (vreme i novac) doneti neverovatne rezultate. Naročito imajući u vidu da je u proteklih nekoliko godina primećen značajan pad tradicionalnih vidova oglašavanja kao što su: reklame na radio i TV stanicama, štampa promotivnog materijala…

Tema izabrana za ovaj seminarski je rad je upravo *“Twitter Clone”*. Rešenje na zadatu temu je napraviti klon veb aplikacije kojoj je cilj da izbaci *twit* kao vid informisanja ili kontaktiranja sa korisnicima. Korisnici na originalnoj *Twitter* aplikaciji mogu da označe da im se sviđa *twit* I da odgovore na isti. U ovoj aplikaciji, koja je *bare-bones* aplikacija (aplikacija sa minimalnim zahtevima, takođe poznata I kao MVP) korisnik može da *twittuje* I da zaprati drugog korisnika.

Softver, ili aplikacija, rešenje, za ovu temu je realizovano tako što pratio *MVC design pattern. Front end I Back end* se nalaze u istom projektu I predstavljaju se kao jedna *Sparkjava* aplikacija.

Korisnik želi da zaprati prijatelje kako bi mogao da vidi o čemu oni razmišljaju I pišu, stoga korisnik dobija opciju *follow user* kako bi ostao u kontaktu sa pomenutim. Razlog zašto je programer izostavio *retweet* I *like post* je zbog generalnog polarizovanja mladih I osetljivost koja može da dovede do različitih problema ličnosti I eventualno povrede prijateljstva, rodne ravnopravnosti, rasizma ili drugih kulturoloških I rodnih problema.

Korisnik može da uloguje preko svog naloga I preko svog imena iznese mišljenje. Korisnik nije ograničen svojim nazivom, stoga se preporučuje da svoje korisničko ime osmisli na što kreativniji način, jer je ova platforma napravljena kao odgovor na iskazivanje mišljenja I kreativnosti koja se ne ugleda samo na *tweet*-ove, već I na korisnička imena.

Radi zaštite identiteta I sprečavanja korišćenja informacija, program nema spoljašnje *API*-je, stoga niko ne može da prati akcije korisnika. Takođe, korisnici vole nove I inovativne stvari, stoga je programer na to odgovorio sa bazom podataka koje nije iterativna, već jednovremena, stoga, svaki put kada se sajt ažurira, cela baza će nestati sa istim I ostaće samo *Mock data* koja je tu kako bi populisala bazu kod kreiranja sajta.

# Specifikacije zahteva korisnika

Zahtevi korisnika za ovaj projekat specifično je bila mogućnost da iskžu svoje mišljenje I stavove kroz *post-*ove. Korisnici u ovom slučcaju nemaju ograničeno koliko karaktera mogu da ubace, ovo je *feature* programera kao umetnička sloboda.

Korisnici zahtevaju interfejs u kome mogu na najlakši mogući način iskažu ono što misle. Takođe, žele interfejs koji bi im omogućio da bez problema na telefu pristupe platformi, stoga, platforma je *responsive* I *phone friendly.*

Kako bi ispunio *phone friendly* zahtev, programer je koristio je *Bootstrap 4* CSS procesor I frejmvork kako bi izgled bio lep na oči I sačuvao vreme I resurse.

# Faze razvoja softvera

U narednim koracima će biti objašnjeno kako je aplikacija bila pravljena od samog početka, odabir tehnologija I njihovo kombinovanje.

Backend, frontend I DevOps delovi su bili podeljenji između studenata: Aleksa Cakića I Nikole Bobinca. Za razliku od svojih profesionalnih okruženja, studenti su rešili da zamene uloge I rade suprotno od onoga što su navikli: kolega Cakić je preuzeo bekend, dok je kolega Bobinac preuzeo front end. Razlog ovome je dalje upoznavanje tehnologija I uloga u IT svetu.

U DevOps delu, studenti su koristili git kao sistem za verzionisanje I Majkrosoftovu platformu GitHub kao mesto gde će se nalaziti izvorni kod, takođe I CI/CD kao kombinovanu praksu kontinuirane integracije I developmenta.

Kolega Cakić je, iz modernih Bekend zaduženja preuzeo veći deo DevOps pozicije zbog takozvanog Build Tool-a Apache Maven gde je postavio sve dependency-je I koji je koristio za CI/CD gde će se preko GitHub platforme Maven skupljati u .war fajl I uploadovati na hosting sajt zvan Heroku.

## Postavka bekenda

Odabir tehnologija za bekend je tekao u dogovoru gde su se obe partije složile da bude programski jezik Java sa Spring frejmvorkom I SparkJava mikrofrejmvorkom, sa malom, brzom relacionom bazom podataka HyperSQL i Apache Maven build toolom.

## SparkJava

Spark je Javin mikrofrejmvork za pravljenje web aplikacija u kotlinu I Javi 8. Spark Framevork je jednostavan i izražajan Java / Kotlin DSL veb framevork napravljen za brzi razvoj. Namera Sparka je da pruži alternativu programerima Kotlin / Jave onima koji žele da razviju svoje veb aplikacije što ekspresivnije i sa minimalnim uzorkom. Sa jasnom filozofijom, Spark je dizajniran ne samo da deplojment učini produktivnijim, već i da kod učini boljim pod uticajem Sparkove elegantne, deklarativne i izražajne sintakse.  
Spark je uvom projektu pokrio većinu funkcionalnosti. com.minitwit.config.WebConfig sadrži mapiranje. Konstruktor kao parametar uzima instancu com.minitwit.service.impl.MiniTwitService koja služi za pokretanje biznis logike aplikacije. MiniTwiTService je stateless, što znači da ne sadrži podatke o stanju klijenta, stoga može biti sigurno sačuvana kao instance promenljiva I može se deliti kroz sve client rekvestove.  
Putanje koje sadrži su sledeće:

* / - Za autentikovane korisnike, predstavlja korisnikov timeline koji sadrži svoje I poruke ljudi koje korisnik prati.
* /public - Predstavlja timeline sa porukama svih ljudi koji koriste aplikaciju.
* /login - Kao get request, predstavlja sign in formu. Ako je korisnik autentikovan, ova ruta će preusmeriti korisnika na svoj timeline. Kao post request, radi kao pravi sign in.
* /register - Kao get rekvest, predstavlja sign up formu. Ako je korisnik autentikovan, ova putanja će preusmeriti na korisnikov timeline. Kao post radi pravu registraciju novog korisnika.
* /t/:username - Predstavlja timeline datog korisnika, ako korisnik ne postoji, statusni error 404 će se vratiti.
* /t/:username/follow - Dodaje trenutnog autentikovanog korisnika kao pratioca datog korisnika, ako korisnik ne postoji, statusni error 404 će biti vraćen. Ako korisnik nije autentikovan, ova putanja će usmeriti na log in stranicu.

*get*("/t/:username/unfollow", (*req*, *res*) -> {

*String* username = *req*.params(":username");

*User* profileUser = service.getUserbyUsername(username);

*User* authUser = getAuthenticatedUser(*req*);

service.unfollowUser(authUser, profileUser);

*res*.redirect("/t/" + username);

*return null*;

});

Otpraćuje trenutnog autentikovanog korisnika kao pratioca datog korisnika, ako korisnik ne postoji, statusni error 404 će biti vraćen. Ako korisnik nije autentikovan, ova putanja će usmeriti na log in stranicu.

*/\**

*\* Shows a users timeline or if no user is logged in,*

*\* it will redirect to the public timeline.*

*\* This timeline shows the user's messages as well*

*\* as all the messages of followed users.*

*\*/*

*get*("/", (*req*, *res*) -> {

*User* user = getAuthenticatedUser(*req*);

*Map*<*String*, *Object*> map = *new* HashMap<>();

map.put("pageTitle", "Timeline");

map.put("user", user);

*List*<*Message*> messages = service.getUserFullTimelineMessages(user);

map.put("messages", messages);

*return new* ModelAndView(map, "timeline.ftl");

}, *new* FreeMarkerEngine());

Prvo, uzima autentikovanog korisnika sa metodom koja samo uzima podatke iz trenutnog sessiona.

*private User* getAuthenticatedUser(*Request request*) {

*return request*.session().attribute(USER\_SESSION\_ID);

}

Onda, pravi se Mapa koja će sadržati sve promenljive korištene u Freemarker templejtu.

Primetiti kako je servis pozvan kako bi preuzeo poruke sa timeline-a. Ovaj servis je samo fasada za korisnika I DAO message-

*/\*\**

*\* Displays view of timeline for logged in user.*

*\** ***@param user***

*\** ***@return***

*\*/*

*@Override*

*public List*<*Message*> getUserFullTimelineMessages(*User user*) {

*if* (*user* == *null*)

{

*throw new* IllegalArgumentException("The object 'user' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("id", *user*.getId());

*String* sql = "select message.\*, user.\* from message, user " +

"where message.author\_id = user.user\_id and ( " +

"user.user\_id = *:id* or " +

"user.user\_id in (select followee\_id from follower " +

"where follower\_id = *:id*))" +

"order by message.pub\_date desc";

*List*<*Message*> result = template.query(sql, params, messageMapper);

*return* result;

}

Koristeći springovu NamedParameterJdbcTemplate klasu, kveri se pokreće sa parametrima koji su prosleđeni iz mape. RowMapper instanca kreira objekte Message-a iz vraćenog ResultSet-a.

*create table user* (

user\_id *integer primary key GENERATED BY DEFAULT AS IDENTITY*(*START WITH* 100),

username *varchar*(50) *not null*,

email *varchar*(50) *not null*,

pw *varchar*(255) *not null*

);

Jedna tabela skladišti korinikove informacije, druga tabela skladišti informacije o porukama (sa referencom na tabelu korisnika) I treća tabela sadrži odnose korisnika (praćenje).

Korisnička profilna slika je gravatar. GravatarUtil klasa gradi URL za uzimanje slike sa korisničkog e-maila.

Nazad na korisnički timeline, u poslednjem koraku, mapa sa svim promenljivama je prosleđena Freemaker Template-u.

Većina ruta provera da li je korisnik autentikovan ili da li korisnik postoji radi redirekcije na odgovarajuću stranicu ili kako bi poslala grešku. Kako bi stvari ostale čiste, ovo se dešava pre filtera.

*get*("/t/:username/follow", (*req*, *res*) -> {

*String* username = *req*.params(":username");

*User* profileUser = service.getUserbyUsername(username);

*User* authUser = getAuthenticatedUser(*req*);

service.followUser(authUser, profileUser);

*res*.redirect("/t/" + username);

*return null*;

});

Prvo, kod proverava da li je korisnik autenkikovan. Ako ne postoji ni jedan, korisnik će biti preusmeren na log in stranicu. Onda proverava da li postoji korisnik koji će biti praćen. Ako ne postoji, onda će biti izbačena greška.

Još jedan primer je POST ruta.

*post*("/register", (*req*, *res*) -> {

*Map*<*String*, *Object*> map = *new* HashMap<>();

*User* user = *new* User();

*try* {

*MultiMap*<*String*> params = *new* MultiMap<*String*>();

*UrlEncoded*.*decodeTo*(*req*.body(), params, "UTF-8");

*BeanUtils*.*populate*(user, params);

} *catch* (*Exception e*) {

*halt*(501);

*return null*;

}

*String* error = user.validate();

*if* (StringUtils.*isEmpty*(error)) {

*User* existingUser = service.getUserbyUsername(user.getUsername());

*if* (existingUser == *null*) {

service.registerUser(user);

*res*.redirect("/login?r=1");

*halt*();

} *else* {

error = "The username is already taken";

}

}

Jedini način pristupa POST parametrima u Sparku je preko Request.body() metode. Ova metoda vraća String primera:

username=user050&email=user@mail.com&password=12345

Na sreću, sa sparkom dolazi I Jetty Server koja ima klasu za parsiranje ovih tipova URL-a.

UrlEncoded.decodeTo(String, MultiMap, String, int) koja uzima 4 parametra.

* String sa parametrima koji će se pročitati
* instancu MultiMap-a gde će parametri biti sačuvani
* enkoding stringa (UTF-8)
* maksimalni broj ključeva koji će biti smešteni u mapu.

Kada su svi parametri smešteni u mapu, metoda BeanUtils.populate(Object, Map) se koristi kako bi postavila propertije iz objekta u mapu. Jedina restrikcija su ključevi (parametri) koji moraju biti identični imenu propertija u objektu.

Oko je objekat uspešno ispunjen, sledeći korak je da se validira informacija koju je korisnik poslao. Metoda za validiranje bi trebala da bude u samom modelu objekta.

*public String* validate() {

*String* error = *null*;

*if* (StringUtils.*isEmpty*(username)) {

error = "You have to enter a username";

} *else if* (!EMAIL\_ADDRESS\_REGEX.matcher(email).matches()) {

error = "You have to enter a valid email address";

} *else if* (StringUtils.*isEmpty*(password)) {

error = "You have to enter a password";

} *else if* (!password.equals(password2)) {

error = "The two passwords do not match";

}

*return* error;

}

Jedina validacija koja ne može biti urađena u ovoj metodi je ona u kojoj se proverava da li je korisničko ime već uzeto.

*User* existingUser = service.getUserbyUsername(user.getUsername());

*if*(existingUser == *null*) {

service.registerUser(user);

res.redirect("/login?r=1");

halt();

} *else* {

error = "The username is already taken";

}

Ako je sve u redu, korisnik je registrovan I preusmeren na login stranicu sa porukom o uspešnoj registraciji.

Ako dođe do greške, mapa sa porukom greške I unesenim vrednostima će biti prosleđena Freemarker template-u kako bi predstavio informaciju.

## **Java Spring**

Java spring je frejmvork I *inversion of control container* za java platformu. Osnovni featuri mogu biti korišćeni u bilo kojoj java aplikaciji, ali već postoje ekstenzije za pravljenje aplikacija povrh JAVA EE platforme. Iako frejmvork ne implicira ni jedan model programiranja, postao je popularan kao dodatak na EJB model. Spring je open source.

U ovom projektu, spring služi da pruži podršku za dependancy injection, transaction management I data access pored ostalih stvari kako bi se umanjio “plumbing” koji je potreban kako bi se bildovala Java aplikacija.

Spring menadžuje dependensije kroz koncept poznat kao Application context, nešto kao centralni registar koji se koristi za menadžovanje životnog ciklusa objekata aplikacije.

Postoje dva načina na koji se konfigurišu beanovi u Spring aplikaciji:

1. XML
2. Anotacije

Ova aplikacija uzima drugi pristup. Main klasa je com.minitwit.App:

*@Configuration*

*@ComponentScan*({"com.minitwit"})

*public class App* {

*public static void* main(*String*[] *args*) {

*Spark*.*staticFileLocation*("/public");

*port*(*getHerokuAssignedPort*());

*System*.*setProperty*("com.google.inject.internal.cglib.$experimental\_asm7", "true");

*AnnotationConfigApplicationContext* ctx = *new* AnnotationConfigApplicationContext(*App*.*class*);

*new* WebConfig(ctx.getBean(*MiniTwitService*.*class*));

ctx.registerShutdownHook();

}

*static int* getHerokuAssignedPort() {

*ProcessBuilder* processBuilder = *new* ProcessBuilder();

*if* (processBuilder.environment().get("PORT") != *null*) {

*return Integer*.*parseInt*(processBuilder.environment().get("PORT"));

}

*return* 4567; *//return default port if heroku-port isn't set (i.e. on localhost)*

}

}

I klasa ima dve vrlo važne anotacije

* @Configuration – Ova anotacija je bitna jer govori Spring da je ova klasa sadrži konfiguracione informacije.
* @ComponentScan({“com.minitwit”}) – ova anotacija kao argument prima listu paketa gde će Spring tražiti anotirane klase kako bi ih dodao u konfiguraciju.

Unutar com.miniTwit paketa mogu se naći sledeće anotirane klase:

com-minitwit.config.DatabaseConfig:

*/\*\**  
 *\* Databse Configuration file holding the Spring Bean*  
 *\* that has source for HSQL setup and scripts.*  
 *\**  
 *\** ***@author*** *Aleksa Cakic*  
 *\*/*  
*@Configuration*  
*public class DatabaseConfig* {  
  
 *@Bean*  
 *public DataSource* dataSource() {  
  
 *EmbeddedDatabaseBuilder* builder = *new* EmbeddedDatabaseBuilder();  
 *EmbeddedDatabase* db = builder  
 .setType(*EmbeddedDatabaseType*.HSQL)  
 .addScript("sql/create-db.sql")  
 .addScript("sql/insert-data.sql")  
 .build();  
 *return* db;  
 }

U ovoj klasi se nalazi metoda anotirana sa @Bean. Ova metoda dodaje objekat vraćen metodom u Springov Application Context, čineći ga pristupačnim ostalim beanovima.

Ova klasa konfiguriše datasource aplikacije. Koristi springovu podršku za HSQLDB kao embedovanu bazu ili in-memory bazu.

*com.minitwit.service.impl.MiniTwitService*

*package com.minitwit.service.impl*;

*import com.minitwit.dao.MessageDao*;

*import com.minitwit.dao.UserDao*;

*import com.minitwit.model.LoginResult*;

*import com.minitwit.model.Message*;

*import com.minitwit.model.User*;

*import com.minitwit.util.PasswordUtil*;

*import org.springframework.beans.factory.annotation.Autowired*;

*import org.springframework.stereotype.Service*;

*import java.util.List*;

*/\*\**

*\* MiniTwit service provider.*

*\**

*\** ***@author*** *Aleksa Cakic*

*\*/*

*@Service*

*public class MiniTwitService* {

*@Autowired*

*private UserDao* userDao;

*@Autowired*

*private MessageDao* messageDao;

*public List*<*Message*> getUserFullTimelineMessages(*User user*) {

*return* messageDao.getUserFullTimelineMessages(*user*);

}

*public List*<*Message*> getUserTimelineMessages(*User user*) {

*return* messageDao.getUserTimelineMessages(*user*);

}

*public List*<*Message*> getPublicTimelineMessages() {

*return* messageDao.getPublicTimelineMessages();

}

*public User* getUserbyUsername(*String username*) {

*return* userDao.getUserbyUsername(*username*);

}

*public void* followUser(*User follower*, *User followee*) {

userDao.insertFollower(*follower*, *followee*);

}

*public void* unfollowUser(*User follower*, *User followee*) {

userDao.deleteFollower(*follower*, *followee*);

}

*public boolean* isUserFollower(*User follower*, *User followee*) {

*return* userDao.isUserFollower(*follower*, *followee*);

}

*public void* addMessage(*Message message*) {

messageDao.insertMessage(*message*);

}

*public LoginResult* checkUser(*User user*) {

*LoginResult* result = *new* LoginResult();

*User* userFound = userDao.getUserbyUsername(*user*.getUsername());

*if* (userFound == *null*) {

result.setError("Invalid username");

} *else if* (!*PasswordUtil*.*verifyPassword*(*user*.getPassword(), userFound.getPassword())) {

result.setError("Invalid password");

} *else* {

result.setUser(userFound);

}

*return* result;

}

*public void* registerUser(*User user*) {

*user*.setPassword(*PasswordUtil*.*hashPassword*(*user*.getPassword()));

userDao.registerUser(*user*);

}

*public void* setUserDao(*UserDao userDao*) {

*this*.userDao = *userDao*;

}

*public void* setMessageDao(*MessageDao messageDao*) {

*this*.messageDao = *messageDao*;

}

}

Ovde se nalaze još dve važne anotacije @Service I @Autowired.

Kao što Bean anotacija obeležava metodu, tako I @Service obelažava “servis” koji će biti dodat u Application Context.

@Autowired je anotacija koja govori Springu kuda da injectuje objekte. Spring će sam instancirati objekte I dependasije.

com.minitwit.dao.impl.UserDaoImpl:

*package com.minitwit.dao.impl*;

*import com.minitwit.dao.UserDao*;

*import com.minitwit.model.User*;

*import org.springframework.beans.factory.annotation.Autowired*;

*import org.springframework.jdbc.core.RowMapper*;

*import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate*;

*import org.springframework.stereotype.Repository*;

*import javax.sql.DataSource*;

*import java.util.HashMap*;

*import java.util.List*;

*import java.util.Map*;

*/\*\**

*\* Setting up Spring implementations of DAO (Data Access Object)*

*\* In computer software, a data access object (DAO) is a pattern that provides an abstract interface*

*\* to some type of database or other persistence mechanism.*

*\* By mapping application calls to the persistence layer,*

*\* the DAO provides some specific data operations without exposing details of the database.*

*\* This isolation supports the single responsibility principle.*

*\* It separates what data access the application needs,*

*\* in terms of domain-specific objects and data types (the public interface of the DAO),*

*\* from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).*

*\* <p>*

*\* Although this design pattern is equally applicable to most programming languages,*

*\* most types of software with persistence needs, and most types of databases,*

*\* it is traditionally associated with Java EE applications and with relational databases*

*\* (accessed via the JDBC API because of its origin in Sun Microsystems' best practice guidelines[1]*

*\* "Core J2EE Patterns" for that platform).*

*\**

*\** ***@author*** *Aleksa Cakic*

*\*/*

*@Repository*

*public class UserDaoImpl implements UserDao* {

*private NamedParameterJdbcTemplate* template;

*@Autowired*

*public* UserDaoImpl(*DataSource ds*) {

template = *new* NamedParameterJdbcTemplate(*ds*);

}

*/\*\**

*\* Get method that fetches user by it's Username as query data.*

*\** ***@param username***

*\** ***@return***

*\*/*

*@Override*

*public User* getUserbyUsername(*String username*) {

*if* (*username* == *null* || *username*.equals(""))

{

*throw new* IllegalArgumentException("The object 'username' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("name", *username*);

*String* sql = "SELECT \* FROM user WHERE username=*:name*";

*List*<*User*> list = template.query(

sql,

params,

userMapper);

*User* result = *null*;

*if*(list != *null* && !list.isEmpty()) {

result = list.get(0);

}

*return* result;

}

*/\*\**

*\* Method that allows function of following another user.*

*\** ***@param follower***

*\** ***@param followee***

*\*/*

*@Override*

*public void* insertFollower(*User follower*, *User followee*) {

*if* (*follower* == *null*)

{

*throw new* IllegalArgumentException("The object 'follower' cannot be null");

}

*if* (*followee* == *null*)

{

*throw new* IllegalArgumentException("The object 'followee' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("follower", *follower*.getId());

params.put("followee", *followee*.getId());

*String* sql = "insert into follower (follower\_id, followee\_id) values (*:follower*, *:followee*)";

template.update(sql, params);

}

*/\*\**

*\* Method that allows user to unfollow another user.*

*\** ***@param follower***

*\** ***@param followee***

*\*/*

*@Override*

*public void* deleteFollower(*User follower*, *User followee*) {

*if* (*follower* == *null*)

{

*throw new* IllegalArgumentException("The object 'follower' cannot be null");

}

*if* (*followee* == *null*)

{

*throw new* IllegalArgumentException("The object 'followee' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("follower", *follower*.getId());

params.put("followee", *followee*.getId());

*String* sql = "delete from follower where follower\_id = *:follower* and followee\_id = *:followee*";

template.update(sql, params);

}

*/\*\**

*\* Method that that checks is a current logged in user is a follower of another user.*

*\** ***@param follower***

*\** ***@param followee***

*\** ***@return***

*\*/*

*@Override*

*public boolean* isUserFollower(*User follower*, *User followee*) {

*if* (*follower* == *null*)

{

*throw new* IllegalArgumentException("The object 'follower' cannot be null");

}

*if* (*followee* == *null*)

{

*throw new* IllegalArgumentException("The object 'followee' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("follower", *follower*.getId());

params.put("followee", *followee*.getId());

*String* sql = "select count(1) from follower where " +

"follower.follower\_id = *:follower* and follower.followee\_id = *:followee*";

*Long* l = template.queryForObject(sql, params, *Long*.*class*);

*return* l > 0;

}

*/\*\**

*\* Method that allows registration of the user as the new user.*

*\** ***@param user***

*\*/*

*@Override*

*public void* registerUser(*User user*) {

*if* (*user* == *null*)

{

*throw new* IllegalArgumentException("The object 'user' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("username", *user*.getUsername());

params.put("email", *user*.getEmail());

params.put("pw", *user*.getPassword());

*String* sql = "insert into user (username, email, pw) values (*:username*, *:email*, *:pw*)";

template.update(sql, params);

}

*/\*\**

*\* Auxillary method that helps by mapping out User.*

*\*/*

*private RowMapper*<*User*> userMapper = (*rs*, *rowNum*) -> {

*User* u = *new* User();

u.setId(*rs*.getInt("user\_id"));

u.setEmail(*rs*.getString("email"));

u.setUsername(*rs*.getString("username"));

u.setPassword(*rs*.getString("pw"));

*return* u;

};

}

com.minitwit.dao.impl.MessageDaoImpl:

*package com.minitwit.dao.impl*;

*import com.minitwit.dao.MessageDao*;

*import com.minitwit.model.Message*;

*import com.minitwit.model.User*;

*import com.minitwit.util.GravatarUtil*;

*import org.springframework.beans.factory.annotation.Autowired*;

*import org.springframework.jdbc.core.RowMapper*;

*import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate*;

*import org.springframework.stereotype.Repository*;

*import javax.sql.DataSource*;

*import java.util.HashMap*;

*import java.util.List*;

*import java.util.Map*;

*/\*\**

*\* Setting up Spring implementations of DAO (Data Access Object)*

*\* In computer software, a data access object (DAO) is a pattern that provides an abstract interface*

*\* to some type of database or other persistence mechanism.*

*\* By mapping application calls to the persistence layer,*

*\* the DAO provides some specific data operations without exposing details of the database.*

*\* This isolation supports the single responsibility principle.*

*\* It separates what data access the application needs,*

*\* in terms of domain-specific objects and data types (the public interface of the DAO),*

*\* from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).*

*\* <p>*

*\* Although this design pattern is equally applicable to most programming languages,*

*\* most types of software with persistence needs, and most types of databases,*

*\* it is traditionally associated with Java EE applications and with relational databases*

*\* (accessed via the JDBC API because of its origin in Sun Microsystems' best practice guidelines[1]*

*\* "Core J2EE Patterns" for that platform).*

*\* <p>*

*\* Adding null check for dao ~NB*

*\**

*\** ***@author*** *Aleksa Cakic and Nikola Bobinac*

*\*/*

*@Repository*

*public class MessageDaoImpl implements MessageDao* {

*private static final String* GRAVATAR\_DEFAULT\_IMAGE\_TYPE = "monsterid";

*private static final int* GRAVATAR\_SIZE = 48;

*private NamedParameterJdbcTemplate* template;

*@Autowired*

*public* MessageDaoImpl(*DataSource ds*) {

template = *new* NamedParameterJdbcTemplate(*ds*);

}

*/\*\**

*\* Method that stores and shows Users view of the particular user's timeline.*

*\** ***@param user***

*\** ***@return***

*\*/*

*@Override*

*public List*<*Message*> getUserTimelineMessages(*User user*) {

*if* (*user* == *null*)

{

*throw new* IllegalArgumentException("The object 'user' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("id", *user*.getId());

*String* sql = "select message.\*, user.\* from message, user where " +

"user.user\_id = message.author\_id and user.user\_id = *:id* " +

"order by message.pub\_date desc";

*List*<*Message*> result = template.query(sql, params, messageMapper);

*return* result;

}

*/\*\**

*\* Displays view of timeline for logged in user.*

*\** ***@param user***

*\** ***@return***

*\*/*

*@Override*

*public List*<*Message*> getUserFullTimelineMessages(*User user*) {

*if* (*user* == *null*)

{

*throw new* IllegalArgumentException("The object 'user' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("id", *user*.getId());

*String* sql = "select message.\*, user.\* from message, user " +

"where message.author\_id = user.user\_id and ( " +

"user.user\_id = *:id* or " +

"user.user\_id in (select followee\_id from follower " +

"where follower\_id = *:id*))" +

"order by message.pub\_date desc";

*List*<*Message*> result = template.query(sql, params, messageMapper);

*return* result;

}

*/\*\**

*\* Will display the conted of the public timeline to everyone, logged in or not.*

*\** ***@return***

*\*/*

*@Override*

*public List*<*Message*> getPublicTimelineMessages() {

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

*String* sql = "select message.\*, user.\* from message, user " +

"where message.author\_id = user.user\_id " +

"order by message.pub\_date desc";

*List*<*Message*> result = template.query(sql, params, messageMapper);

*return* result;

}

*/\*\**

*\* Method that allows User to insert message (tweet) into timeline, public timeline*

*\* and personal timeline. Message (Twit is stored in HSQL database).*

*\** ***@param m***

*\*/*

*@Override*

*public void* insertMessage(*Message m*) {

*if* (*m* == *null* || *m*.equals(""))

{

*throw new* IllegalArgumentException("The object 'message' cannot be null");

}

*Map*<*String*, *Object*> params = *new* HashMap<*String*, *Object*>();

params.put("userId", *m*.getUserId());

params.put("text", *m*.getText());

params.put("pubDate", *m*.getPubDate());

*String* sql = "insert into message (author\_id, text, pub\_date) values (*:userId*, *:text*, *:pubDate*)";

template.update(sql, params);

}

*/\*\**

*\* This method maps out the messages by helping in storing and saving code and performance.*

*\*/*

*private RowMapper*<*Message*> messageMapper = (*rs*, *rowNum*) -> {

*Message* m = *new* Message();

m.setId(*rs*.getInt("message\_id"));

m.setUserId(*rs*.getInt("author\_id"));

m.setUsername(*rs*.getString("username"));

m.setText(*rs*.getString("text"));

m.setPubDate(*rs*.getTimestamp("pub\_date"));

m.setGravatar(*GravatarUtil*.*gravatarURL*(*rs*.getString("email"), GRAVATAR\_DEFAULT\_IMAGE\_TYPE, GRAVATAR\_SIZE));

*return* m;

};

}

Ove klase su bazirane na DAO paternu kako bi abstrakovali perzistentnost implementacije detalja u aplikaciji. Ako bi se koristio druga baza podataka ili ORM, samo treba da se uradi druga implementacija interfejsa com.minitwit.dao.UserDao I com.minitwit.dao.MessageDao.

Ove klase su anotirane kao @Repository umesto kao @Service. @Repository se preporučuje za svaku klasu koja implementuje DAO ili Repository pattern. Spring koristi @Component kao generični stereotip za bilo koju menedžovanu komponentu.

Vratiti se na main metodu, metoda pravi springov Application Context tipa AnnotationConfigApplicationContext, prosleđujući argument klasi App kako bi Spring mogao da isčita anotacije na ovoj klasi I skenira pakete com.minitwit za druge klase I Spring anotacije.

*/\*\**  
 *\** ***@author*** *Aleksa Cakic*  
 *\*/*  
*@Configuration*  
*@ComponentScan*({"com.minitwit"})  
*public class App* {  
  
 *public static void* main(*String*[] *args*) {  
  
 *Spark*.*staticFileLocation*("/public");  
  
 *port*(*getHerokuAssignedPort*());  
  
 *System*.*setProperty*("com.google.inject.internal.cglib.$experimental\_asm7", "true");  
  
 *AnnotationConfigApplicationContext* ctx = *new* AnnotationConfigApplicationContext(*App*.*class*);  
  
 *new* WebConfig(ctx.getBean(*MiniTwitService*.*class*));  
  
 ctx.registerShutdownHook();  
 }  
  
 *static int* getHerokuAssignedPort() {  
  
 *ProcessBuilder* processBuilder = *new* ProcessBuilder();  
  
 *if* (processBuilder.environment().get("PORT") != *null*) {  
 *return Integer*.*parseInt*(processBuilder.environment().get("PORT"));  
 }  
  
 *return* 4567; *//return default port if heroku-port isn't set (i.e. on localhost)*  
}

Linija posle toga stvara instancu WebConfig prosleđujući kao argument bean tipa MiniTwitService. Ovaj bean ima dependencije kao DAO-u koji zavise od Data Source-a HSQLDB-a da bi radili.

U nekom trenutku će se ApplicationContext zatvoriti. Može se samo zatvoriti sa metodom ctx.close(), jer će pristup in-memory bazi biti izgubljen.

Srećom, postoji metoda registerShutDownHook() koja registruje shutdown hook sa JVM runtime-om, zatvarajući ovaj context sa JVM-omovim gašenjem. Ovo je savršeno za aplikacije koje će slušati konekciju dokle god se ne zatvori, baš kao ova.

## **Maven (uzeto iz https://studenti.rs/skripte/informacione-tehnologije/maven/)**

Maven je moćan alat za upravljanje projektima koji se temelji na POM (Project Object Model).

Koristi se za izgradnju projekata, dependency i dokumentaciju. Pojednostavljuje postupak sastavljanja poput ANT-a ( Java library and command-line tool).

Napredniji je od ANT-a.

Ukratko, možemo reći da je Maven alat koji se može koristiti za izgradnju i upravljanje bilo kojim projektima temeljenim na Javi. Maven olakšava svakodnevni posao Java programera i općenito pomaže pri razumijevanju bilo kojeg Java projekta.

Maven je alat koji se koristi pri razvoju aplikacija u cilju lakše integracije sa već postojećim bibliotekama klasa.

Dependacies korišćene

* org.webjars:jquery:3.5.1
* com.sparkjava:spark-core:2.9.1
* com.sparkjava:spark-template-freemarker:2.3
* org.springframework.security:spring-security-web:5.2.6.RELEASE
* org.springframework.security:spring-security-config:5.2.6.RELEASE
* org.freemarker:freemarker:2.3.25-incubating
* org.springframework:spring-core:5.2.6.RELEASE
* org.springframework:spring-context:5.2.6.RELEASE
* org.springframework:spring-jdbc:5.2.6.RELEASE
* org.hsqldb:hsqldb:2.3.4
* org.apache.commons:commons-dbcp2:2.1.1
* de.svenkubiak:jBCrypt:0.4.1
* commons-beanutils:commons-beanutils:1.9.3
* org.slf4j:slf4j-simple:1.7.22
* org.apache.maven.plugins:maven-dependency-plugin:3.1.2

Prvo, treba se uraditi konfigurisanje kako bi bili sigurni da maven koristi Javu 8. To se radi kroz maven-compiler-plugin.

Drugi plugin se koristi za lakše pokretanje aplikacije iz komandne linije uz sledeću komandu:

mvn exec:java -Dexec.mainClass="com.example.MainClass" -Dexec.args="arg0 arg1 arg2"

Kao što se da videti, trebalo bi dobro poznavanje paketa I imena klasa kako bi se pokrenulo kao I parametre koje prima. Srećom, to se sve može deklarisati u pom.xml fajlu sa exec-maven-plugin kako bi se program pokrenuo samo sa:

mvn exec:java.

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.3.2</version>

<configuration>

<source>${jdk.version}</source>

<target>${jdk.version}</target>

</configuration>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>versions-maven-plugin</artifactId>

<version>2.1</version>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<mainClass>com.minitwit.App</mainClass>

<arguments>

</arguments>

</configuration>

</plugin>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

<configuration>

<descriptorRefs>

*<!-- This tells Maven to include all dependencies -->*

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass>com.minitwit.App</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<version>2.3</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>copy</goal>

</goals>

<configuration>

<artifactItems>

<artifactItem>

<groupId>com.github.jsimone</groupId>

<artifactId>webapp-runner</artifactId>

<version>8.5.11.3</version>

<destFileName>webapp-runner.jar</destFileName>

</artifactItem>

</artifactItems>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

# Front end

## Freemaker Template

Ovaj templejt je mehanizam korišten za generisanje HTML stranica aplikacije.

Zahvaljujući Spark-Freemarker biblioteci, mogu se samo upakovati parametri templejt I vratiti ModelAndView objekat pokazujući na templajt fajl. Primer koda:

*/\**

*\* Presents the login form or redirect the user to*

*\* her timeline if it's already logged in*

*\*/*

*get*("/login", (*req*, *res*) -> {

*Map*<*String*, *Object*> map = *new* HashMap<>();

*if* (*req*.queryParams("r") != *null*) {

map.put("message", "You were successfully registered and can login now");

}

*return new* ModelAndView(map, "login.ftl");

}, *new* FreeMarkerEngine());

Ova aplikacija koristi samo četiri templejta:

* login.ftl – za login stranicu
* register.ftl – za registracionu stranicu
* timeline.ftl – ja javni I korisnički timeline-a
* masterTemplate.ftl – Sadrži glavni layout aplikacije.

Većina web aplikacija koristi isti layout za sve svoje stranice pošto nije praktično ponavljati kod ovog layouta na svim stranicama. Većina template-a sadrži neke mehanizme kako bi layout bio u jednom fajlu.

U slučaju Freemarker templejta, sadrži include direktivu, ali možda neće odgovarati svim klasama. Za ovaj tip projekta kombinacija macro I import direktiva je bolji.

*<#macro masterTemplate title*="Welcome"*>*

<!DOCTYPE *html*>

<html *lang*="en">

<head>

<meta *http-equiv*="Content-Type" *content*="text/html; charset=UTF-8">

<meta *http-equiv*="X-UA-Compatible" *content*="IE=edge">

<meta *name*="viewport" *content*="width=device-width, initial-scale=1">

<link *rel*="shortcut icon" *type*="image/x-icon"

*href*="https://www.shareicon.net/data/16x16/2016/09/07/827030\_bird\_512x512.png">

<title>*${title}* | MiniTwit

</title>

<link *rel*="stylesheet" *href*="https://use.fontawesome.com/releases/v5.6.3/css/all.css">

<link *rel*="stylesheet" *type*="text/css" *href*="/css/bootstrap.min.css">

<link *rel*="stylesheet" *type*="text/css" *href*="/css/style.css">

<link *rel*="script" *href*="../../../assets/scripts.js">

</head>

<body>

<div *class*="container">

<nav *class*="navbar navbar-dark" *role*="navigation">

<div *class*="navbar-header">

<button *type*="button" *class*="navbar-toggle" *data-toggle*="collapse"

*data-target*="#navbarToggleExternalContent">

<span *class*="icon-bar" *style*="background-color: #0c7cd5"></span>

<span *class*="icon-bar" *style*="background-color: #0c7cd5"></span>

<span *class*="icon-bar" *style*="background-color: #0c7cd5"></span>

</button>

<a *class*="navbar-brand" *href*="/">

<i *class*="fas fa-dove"></i>

MiniTwit

<h1 *style*="display: none;">MiniTwit</h1>

</a>

</div>

<div *class*="collapse navbar-collapse" *id*="navbarToggleExternalContent">

<ul *class*="nav navbar-nav navbar-right">

*<#if user*??*>*

<li><a *href*="/">My Timeline</a></li>

<li><a *href*="/public">Public Timeline</a></li>

<li><a *href*="/logout">Sign Out [*${user*.*username}*]</a></li>

*<#else>*

<li><a *href*="/public">Public Timeline</a></li>

<li><a *href*="/register">Sign Up</a></li>

<li><a *href*="/login">Sign In</a></li>

*</#if>*

</ul>

</div>

</nav>

<div *class*="container">

*<#nested />*

</div>

<footer *class*="footer">

<div *class*="icon-bar jumbotron">

<p *class*="display-4">

<i *class*="fas fa-dove"></i>

MiniTwit &mdash; University Assignment

</p>

<h2 *class*="display-4">Twitter Clone</h2>

<br>

<p>Done by: </p>

<div *class*="people-names js-aleksa-name">

<p>Aleksa Cakic; <span *class*="text-primary text-capitalize text-danger">SI 23/17</span> </p>

</div>

<p> And </p>

<div *class*="people-names js-nikola-name">

<p>Nikola Bobinac; <span *class*="text-primary text-capitalize text-danger">SI 18/17</span> </p>

</div>

<div *class*="js-aleksa-links links-hidden">

<h2 *class*="display-4">

<a *href*="https://www.facebook.com/profile.php?id=100001669704325" *target*="\_blank">

<i *class*="fab fa-facebook-square"></i>

</a>

<a *href*="https://www.linkedin.com/in/aleksa-cakic-b8a426184/" *target*="\_blank">

<i *class*="fab fa-linkedin-in"></i>

</a>

<a *href*="https://gamemess.itch.io/" *target*="\_blank">

<i *class*="fas fa-gamepad"></i>

</a>

<a *href*="https://soundcloud.com/lorelei-frost/tracks" *target*="\_blank">

<i *class*="fab fa-soundcloud"></i>

</a>

<a *href*="https://www.youtube.com/user/AkiFilozoff" *target*="\_blank">

<i *class*="fab fa-youtube"></i>

</a>

<a *href*="https://github.com/Alexayy" *target*="\_blank">

<i *class*="fab fa-github"></i>

</a>

<a *href*="https://gitlab.com/Alexayy" *target*="\_blank">

<i *class*="fab fa-gitlab"></i>

</a>

</h2>

</div>

<div *class*="js-nikola-links links-hidden">

<h2 *class*="display-4">

<a *href*="https://www.facebook.com/nbobinac" *target*="\_blank">

<i *class*="fab fa-facebook-square"></i>

</a>

<a *href*="https://www.linkedin.com/in/nikola-bobinac-5813b1150/" *target*="\_blank">

<i *class*="fab fa-linkedin-in"></i>

</a>

<a *href*="https://www.youtube.com/shadonerd" *target*="\_blank">

<i *class*="fab fa-youtube"></i>

</a>

<a *href*="https://github.com/Shadochi" *target*="\_blank">

<i *class*="fab fa-github"></i>

</a>

<a *href*="https://gitlab.com/nikolabobinac" *target*="\_blank">

<i *class*="fab fa-gitlab"></i>

</a>

<a *href*="https://www.twitch.tv/shadonerd" *target*="\_blank">

<i *class*="fab fa-twitch"></i>

</a>

</h2>

</div>

<br>

<p>Tech Stack: </p>

<p><a *href*="http://sparkjava.com/" *target*="\_blank">Spark Java</a>,

<a *href*="https://spring.io/" *target*="\_blank">Java Spring</a>,

<a *href*="https://freemarker.apache.org/" *target*="\_blank">Java FreeMaker Template Engine</a>,

<a *href*="https://getbootstrap.com/" *target*="\_blank">Bootstrap 4 CSS Framework</a>,

<a *href*="https://maven.apache.org/" *target*="\_blank">Apache Maven</a>,

<a *href*="www.heroku.com" *target*="\_blank">Heroku</a>,

<a *href*="http://hsqldb.org/" *target*="\_blank">HSQLDB</a>

</p>

</div>

</footer>

</div>*<!-- /container -->*

<script *src*="https://code.jquery.com/jquery-3.3.1.slim.min.js"

*integrity*="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"

*crossorigin*="anonymous"></script>

<script *src*="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"

*integrity*="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

*crossorigin*="anonymous"></script>

<script *src*="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"

*integrity*="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

*crossorigin*="anonymous"></script>

<script *src*="scripts.js"></script>

</body>

</html>

*</#macro>*

Macro definiše fragment templejta koji dozvoljava parametrima sa prvobitnim vrednostima (kao $(title)) da budu korišteni kao korisnički definisane direktive. Sadrži nested elemente koji će biti pokrenuti u context-u gde je macro pozvan.

Kako bi se koristio master template, treba se import-ovati direktiva I dostaviti sadržaj koji će biti zamenjen nested elementom. Sledi primer login.ftl

*<#import* "masterTemplate.ftl" *as layout />*

*<@layout*.*masterTemplate title*="Sign In"*>*

*<#if message*??*>*

<div *class*="alert alert-success">

*${message}*

</div>

*</#if>*

*<#if error*??*>*

<div *class*="alert alert-danger">

<strong>Error:</strong> *${error}*

</div>

*</#if>*

<h3>Sign In</h3>

<form *class*="form-horizontal" *action*="/login" *role*="form" *method*="post">

<div *class*="form-group">

<label *for*="username" *class*="col-sm-2 control-label">Username: </label>

<div *class*="col-sm-10">

<input *type*="text" *class*="form-control" *name*="username" *id*="username" *placeholder*="Username"

*value*="*${username*!*}*"/>

</div>

</div>

<div *class*="form-group">

<label *for*="password" *class*="col-sm-2 control-label">Password: </label>

<div *class*="col-sm-10">

<input *type*="password" *class*="form-control" *name*="password" *placeholder*="Password">

</div>

</div>

<div *class*="form-group">

<div *class*="col-sm-offset-2 col-sm-10">

<button *type*="submit" *class*="btn btn-default">Sign In</button>

</div>

</div>

</form>

*</@layout*.*masterTemplate>*

Primetiti kako je master template importovan I referenciran sa identifierom layout, I kako su vrednosti title parametra prosleđene. HTML generisan ovom korisničkom direktivom će biti zamenjen tamo gde master template definiše nested element.

## CSS

CSS podrazumeva stilizaciju osnovnih elementata HTML-a koji se nadovezuju i omogućavaju uštedu vremena s poboljšanjem efekata u HTML dokumentu.

CSS korišten u ovom projektu je male strukture, koristeći više za bojenje, pozadinu I Jquery pristup funckijama.

body {

padding-top: 20px;

padding-bottom: 20px;

}

.*footer* {

padding-top: 19px;

padding-right: 15px;

padding-left: 15px;

color: #555555;

border-top: 1px solid #000000;

}

.*navbar* {

background-color: #C0C0C0;

}

.*container* {

background-image: url("http://api.thumbr.it/whitenoise-361x370.png?");

}

*@media* (min-width: 768px) {

.*container* {

max-width: 730px;

}

}

.*links-hidden* {

display: none; */\* There was "sol" written causing an error, deleted word sol\*/*

}

.*people-names* {

text-decoration: underline;

text-decoration-color: black;

}

.*people-names*:*hover* {

text-decoration-color: red;

}

.*border-links-aleksa* {

border: 1px solid red;

border-radius: 16px;

display: inline-block

}

.*border-links-nikola* {

border: 1px solid green;

border-radius: 16px;

display: inline-block

}

Bootstrap korišten u ovom projektu je pozamašan. Od kreiranja navbara do kontejnera I responsivnosti sajta. Bootstrap je CSS frejmvork koji se fokusira na responzivnost I čistu stranicu. Ime bootstrap funkcije se dodaje u klasu HTML-a odakle će se html elementi dalje formirati iz css-a I scss-a bootstrap fajlova.

FontAwesome je toolkit zasnovan na CSS-u I Less-u. Kreirao ga je Dave Gandy radi sinergije sa BootStrap CSS Frejmvorkom I kasnije je pripojen u BootStrap CDN. Font awesome se isto koristi kao I BootStrap.

**JS I JQuery**

**---**

# DevOps sloj

## Git (https://help.superhosting.rs/sta-je-to-git/)

Git je sistem za kontrolu i istoriju verzije koda koji se koristi uglavnom od strane programera za razvoj veb stranica i aplikacija. I sam Git projekat koristi ovaj sistem, zbog mnogobrojnih izmena u njegovom kodu (C, Shell).

Pomoću Gita moguće je da jedan ili više programera primeni promene u kodu u fajlovima jednog projekta tako da se izbegnu nedoslednosti i sukobi zbog tih promena.

Git čuva istoriju izmena, šta je tačno u kodu promenjeno, od strane koga i kada, i olakšava povratak na prethodne verzije.

Git radi sa repozitorijumom (repository), koji predstasvlja direktorijum koji sadrži fajlove projekta i poseban sistemski direktroijum pod nazivom .git.

Platforma na kojoj je kod hostovan je po izboru oba programera Github.

Prvobitno je rad počeo samo sa master granom, ali se kasnije tokom razvijanja aplikacije razgranao u dosta grana koje su se na kraju prispojile.

## Deplyment I Hosting

Live verzija sajta je gurnuta preko Heroku hosting platforme. Uzimajući u obzir modernost današnjih platformi, Heroku može da povuče kod sa GitHub platforme. Kada povuče kod, prepoznaće bazu I krenuti da automatski konfiguriše najbolju opciju. Imajući u vidu Maven automation build tool, Heroku može na svom instance serveru da preuzme I pokrene dependacy-je I zapakuje ih, u ovom slučaju u War file.

Pošto inicijalna konfiguracija nije uspela, u main metodi je definisana metoda koja će reći instance serveru koji port da zauzme jer Heroku ne zna kako da konfiguriše SparkJava aplikaciju.

*static int* getHerokuAssignedPort() {

*ProcessBuilder* processBuilder = *new* ProcessBuilder();

*if* (processBuilder.environment().get("PORT") != *null*) {

*return Integer*.*parseInt*(processBuilder.environment().get("PORT"));

}

*return* 4567; *//return default port if heroku-port isn't set (i.e. on localhost)*

}

**Faze razvoja softvera**

Font – Times new roman

Line spacing – 1.5

Position: Justified

Default Body text – size 12

Header 3 – size 14

Header 2 – Size 16

Header 1 – Size 18

dodati sadžaj. Izvori su linkovani u naslovu, zaključak, brojeve na stranicama

- JavaScript

- JS

- Jquery