



Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin



SEMINARSKI RAD
Predmet: Razvoj softvera otvorenog koda
Društvena mreža: Twitter Clone

Predmetni nastavnik:
Doc. Dr. Zoltan Kazi

Autori rada
Aleksa Cakić SI 23/17
Nikola Bobinjac SI 18/17

Zrenjanin 2020.

Sadržaj

1. Uvod.....	3
2. Specifikacije zahteva korisnika	5
3. Faze razvoja softvera	6
4. Prikaz softvera	10
5. Prikaz realizacija i implementacije	17
5.1. Postavka bekenda	17
5.2. Postavke frontenda	17
5.3. SparkJava.....	17
5.4. Java Spring	24
5.5. Maven	43
5.6. Freemaker Template	47
5.7. CSS	56
5.8. Javascript i jQuery	58
5.9. Git	59
5.10. Deplyment I Hosting.....	61
5.11. Baza podataka	61
5.12. Testiranje Aplikacije	62
6. Korišteni alati i softveri	63
7. Literatura.....	64

1. 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, brežiranje, 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 posetioći 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 rodni 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.

2. Specifikacije zahteva korisnika

Zahtevi korisnika za ovaj projekat specifično je bila mogućnost da iskžu svoje mišljenje I stavove kroz *post*-ove. Korisnici u ovom slučaju 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 telefonu pristupe platformi, stoga, platforma je *responsive* I *phone friendly*.

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

3. 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 podeljeni 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 backend, 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 Backend 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.

Prvobitni plan je bio minimalan klon Twitter aplikacije sa najvažnijim funkcionalnostima. Cilj je bio lakoća razumevanja koda, snalaženje korisnika i ispunjenje zadatka uslova za ispit i postignutih poena.

Proces je počeo istraživanjem šta čini Twitter aplikaciju. Izgled, funkcionalnosti i tehnologije. Prateći blogove i testimonijale zaposlenih u Twitteru, studenti su pomno pratili i razvijali u vidu prethodno pomenutih izvora.

Razvoj je počeo pripremanjem simultanim postavljanjem bekenda i Freemarkera gde se inicijalno postavila početna *Hello World* aplikacija. Putem mock podataka je usput testirano prikazivanje podataka sa bekenda, iako je Springom simuliran efekat JakartaEE JSP-a. Sam proces je krenuo sređivanjem main metode kako bi uspešno pokretala aplikaciju. Zatim, napravljeni su WebConfig i Modeli. Kako se razvijao backend, paralelno se u *pom.xml* dodavalo dzavisnosti aplikacije i dodatno konfigurisanje.

Kada je prvi mock bekenda bio završen, sledeo je DAO sloj aplikacije i njihove implemetacije. U slučaju ove aplikacije, koja je bliža *Repository* dizajn paternu, DAO slojevi su dozvolili dependancy injection. Paralelno uz te konfiguracije i taj sloj, servis je uveliko bio u razvoju i pratio funkcionalnosti DAO sloja i *WebConfig*-a.

Projekat koji je počeo sa nekoliko autogenerisanih Spring fajlova uz pomoć IntelliJ IDEA-e, je uz iskusne i mlade ruke studenata krenuo da prima oblik ozbiljne aplikacije. Posle završetka DAO sloja, dalji tok se okretao postavljanju mocka frontenda i bezbednosti na bekendu. Dodate su Login i User provere, *PasswordUtil* kao sigurnosni sloj, hašovanje pasvorda i konstantna provera autentikacije korisnika na glavnoj stranici.

Baza podataka je naknadno dodata i hardkodovana radi populacije kako bi se ispunio glavni tajmlajn i kako bi aplikacija pri otvaranju izgledala ispunjeno. Kada je to bilo završeno, bilo je potrebno završiti sigurnost i bezbednost i dodati su hederi koji dodatno pojačavaju zaštitu, kao i unapređen email REGEX u User klasi.

Gravat je dodat na kraju kako bi svaki korisnik mogao da ima svoju jedinstvenu sliku na profilu. Bilo nasumično ili slika povezana sa emailom, daje novu dimenziju iako je ovo projekat rađen kao seminarski rad.

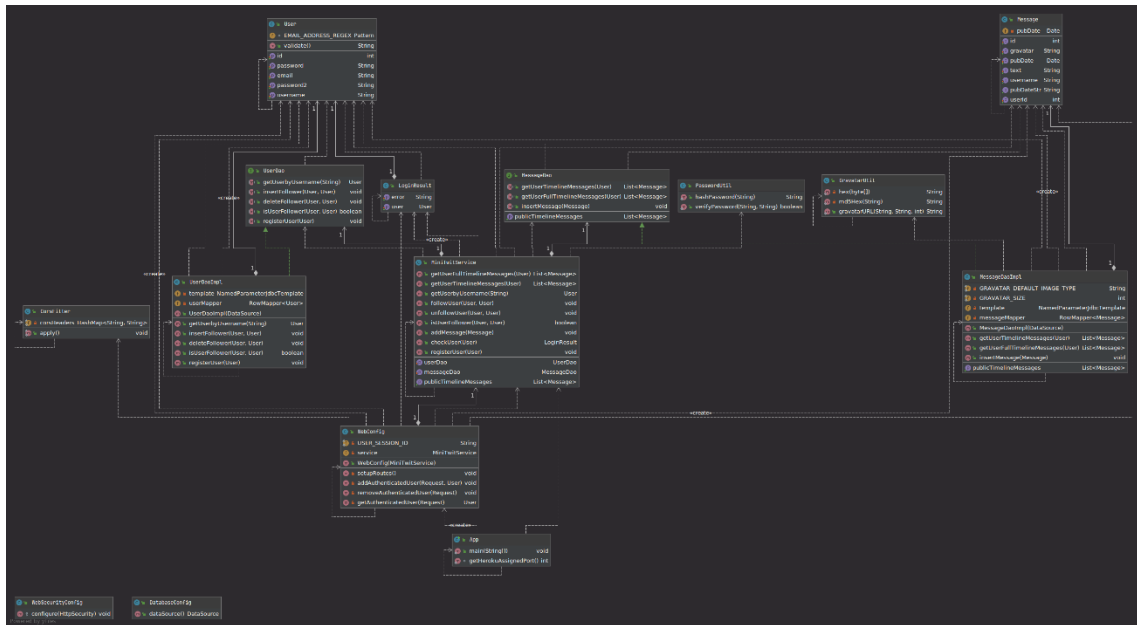


Slika 1. Dijagram baze

Dok se bekind razvijao, bilo je preko potrebno testiranje in-memory baze koja je upisana u sistem aplikacije i postoji dok aplikacija radi. Postman softver je pokrio dalju potrebu testiranja API-ja i njegove konekcije sa bazom i načinom upisivanja podataka, procesovanja i kreiranja baze. Uzimajući u obzir da je baza in-memory tipa, baza će biti jedinstvena pri svakom pokretanju i pri svakom pokretanju će se praviti nova.

Dizajn baze je prost. Postoje tri tabele koje nisu povezane, ali zato logika u kodu reguliše ID intidžere i time simulira *foreign key*-eve. Dodatavati FK je redundantno u ovom slučaju jer je sam projekat simulacija projekta sa otvorenim kodom i sam kod

aplikacije će regulisati simulaciju FK.



Slika 2. Klasni dijagram aplikacije

Klasni dijagram aplikacije je izrazito kompleksan. Na slici iznad je predstavljen klasni dijagram backend logike koji je generisan i prepravljen uz pomoć JetBrains-ovog IDE-a IntelliJ IDEA. Pored standardnih opisa relacionih notacija, IntelliJ IDEA ima svoje tri jedinstvene notacije. Isprekidana zelena strlica sa isprekidanom linijom označava *implements* funkciju, plava relacija označava *extends* funkciju dok se crvena pojavljuje kod unutrašnjih klasa.

Kako se po slici vidi, „najzauzetije“ klase su *DaoImpl*, *MiniTwitService* kao i *WebConfig* koji formiraju relacije sa skoro svakom klasom. Dijagram predstavljen je viđen i dokumentovan u implementacionoj perspektivi.

WebSecurityConfig i *DatabaseConfig* su izdvojene i bez relacija jer nemaju veze sa drugim klasama, osim sa samim sobom i preko Spring anotacija će biti prepoznati od strane Springa, te će se loadovati u run time-u Springa. Tu će se isčitati anotacije i zadati zadaci i pokrenuti ostale funkcije.

Klase *User* i *Message* imaju najviše relacija. Klasa *User* ima 2 Dependency relacije sa *UserDaoImpl*. Dependency relacija je semantička konekcija između zavisnih i nezavisnih model elemenata. Ova relacija postoji između dva elementa ako se definicija jednog elementa promeni, imaće uticaj na drugi element takođe. Uzimajući u obzir da je relacija jednosmerna, ona potiče iz implementacione klase, naznačujući da su obe klase sklone promenama, tačnije, ako se nešto u klasi *UserDaoImpl* promeni, promeniće se i definicija *User* klase. Druga dependency relacija će samo jednom promeniti *User* model i to tada kada se kreira. To označava ključna reč `<<create>>`. U klasi *UserDaoImpl* se može videti zelena strelica ka *UserDao* interfejsu što implicira da *UserDaoImpl* implementira *UserDao* interfejs. Gledajući kardinalitet između klase i interfejsa. Još

jedna relacija koja se vidi jeste agregacija. U ovom slučaju je *User* model kolekcija te jedne klase – jedan korisnik ima jednu instancu *UserDaoImpl* klase.

Slišan slučaj je sa *Message* delom dijagrama. Zajedno i *Message* i *User* zavise od servisa jer ako dođe do promene u nekoj od klasa, sve zavisnosti koje dolaze iz servisa će morati biti obavestene kroz metode, kako od same registracije, logina, post-a ili follow/unfollow-a kako bi sve promene bile uredno implementirane.

Sve to ne bilo moguće da nema *WebConfig* klase koja će postaviti sve rute na kojima će se odvijati neka od funkcija ili funkcije koje će kreirati *DAO* klase implementacije i kroz agregacionu relaciju sa servisom spojiti logički ostale komponente. Ta relacija se zaniva na „ima“ ili „has a“ relaciji gde taj *WebConfig* „ima“ servis. U klasi *WebConfig* se nalazi konstruktor koji za parametar prima *MiniTwitService*.

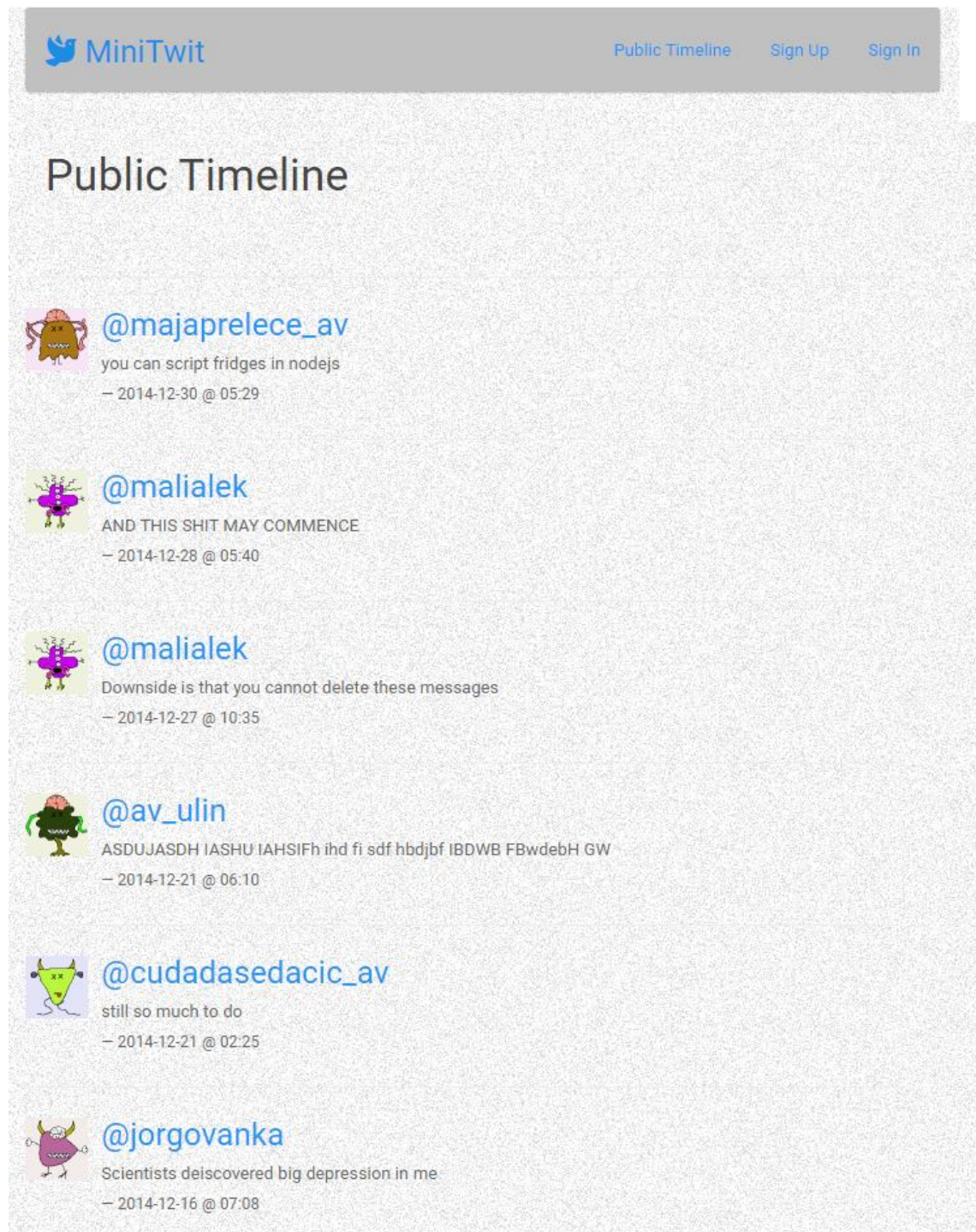
Po hijerarhijskoj logici, *App* je klasa koja optočinje sve. To je *main* klasa koja sadrži *main* metodu programa. Ima samo-relaciju i zavisnu stvaralčku relaciju sa *WebConfig* klasom naglašavajući svoju važnost. Dalje, zavisnom relacijom sa *MiniTwitService* klasom se implicira promenljivost i fleksibilnost same aplikacije. Menjanjem *main* metode ili klase dalje zahteva adaptaciju servisa i ostalih klasa povezanih sa tim servisom što može značiti da postoji veliki prostor za greške u bildovima, ili može značiti samu skalabilnost aplikacije.

Ono što nije pomenuto, ali je od velike važnosti, jesu *LoginResult*, *PasswordUtil*, *GravatarUtil* i *CorsFilter* klase. *LoginResult* klasa ima relacije sa korisničkim modelom i servisom. Tačnije, servis ima zavisni odnos gde kada se pozove servis, *LoginResult* će se stvoriti zbog uslova postavljenih u kodu. Jedino je logično kreirati to kroz servis i dalje ići ka smeru modela korisnika koji je pokrenuo servis radi ili registracije ili autentikacije. Razumevajući relacije, *PasswordUtil* ima samo jednu sa servisom gde služi radi provere i heširanja. *Gravatar* klasa je povezana sa *DAO*-m za poruke jer će se nalaziti u telu poruke jer svaka poruka nosi svog korisnika i svaki korisnik nosi svoj *Gravatar*. *CorsFilter* je povezan sa *WebConfig* klasom jer je tu jedino korištena. Tu će *CORS* filter da reguliše pristup i bezbednost ruta kroz *Cross Origin* sistem.

4. Prikaz softvera

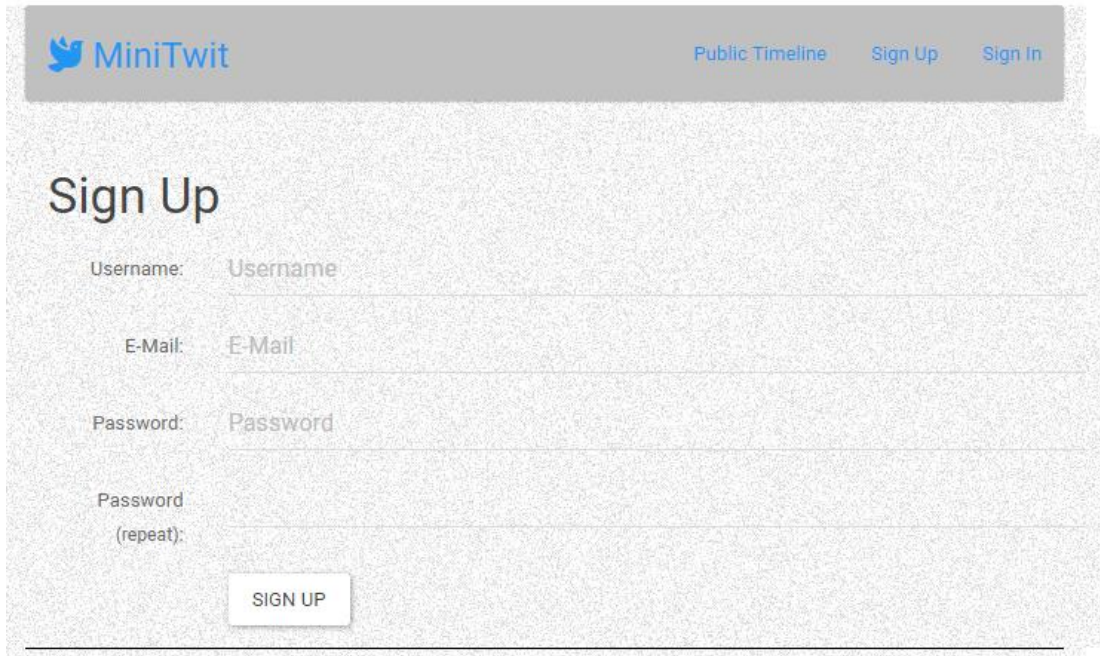
Gotovom softveru je moguće pristupiti na linku: <https://tfzr-minitwit-clone.herokuapp.com/>

Prilikom pristupa softveru korisnik se redirektuje na javni zid, gde može da vidi sve objave sa mreže, takođe u meniju ima opciju i da napravi nalog ili da uđe na sajt sa već postojećim nalogom (*slika 3*).



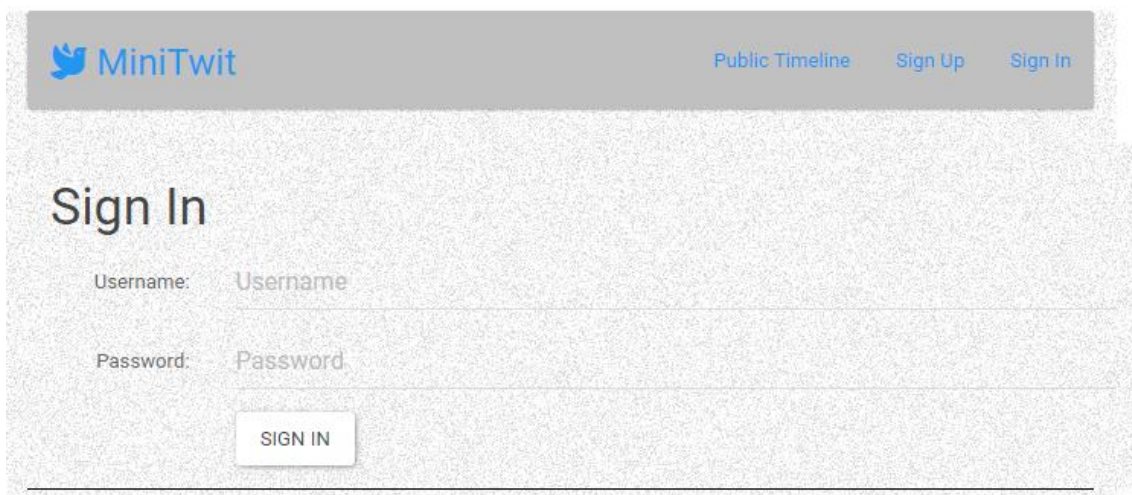
Slika . Početna strana softvera

Prilikom pristupa “Sign up” stranici korisnik je predstavljen sa formom gde treba da unese željeno korisničko ime, svoju email adresu i lozinku za sajt, kao i da potvrdi lozinku (*slika 4*).

The image shows a web browser window displaying the 'MiniTwit' sign-up page. At the top, there is a navigation bar with the 'MiniTwit' logo on the left and three links: 'Public Timeline', 'Sign Up', and 'Sign In' on the right. The main heading of the page is 'Sign Up'. Below the heading, there are four input fields: 'Username:' with a placeholder 'Username', 'E-Mail:' with a placeholder 'E-Mail', 'Password:' with a placeholder 'Password', and 'Password (repeat):' with a placeholder 'Password'. A 'SIGN UP' button is located at the bottom of the form.

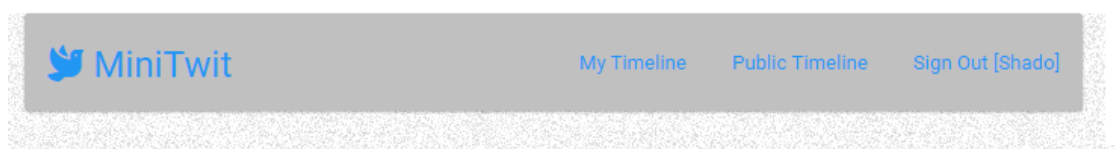
Slika 4. Forma za registraciju

Ukoliko korisnik već poseduje nalog, može da pristupi „Sign In“ stranici gde je potrebno da unese svoje korisničko ime i šifru kako bi pristupio sajtu kao registrovan korisnik (*slika 5*).

The image shows a web browser window displaying the 'MiniTwit' sign-in page. At the top, there is a navigation bar with the 'MiniTwit' logo on the left and three links: 'Public Timeline', 'Sign Up', and 'Sign In' on the right. The main heading of the page is 'Sign In'. Below the heading, there are two input fields: 'Username:' with a placeholder 'Username' and 'Password:' with a placeholder 'Password'. A 'SIGN IN' button is located at the bottom of the form.

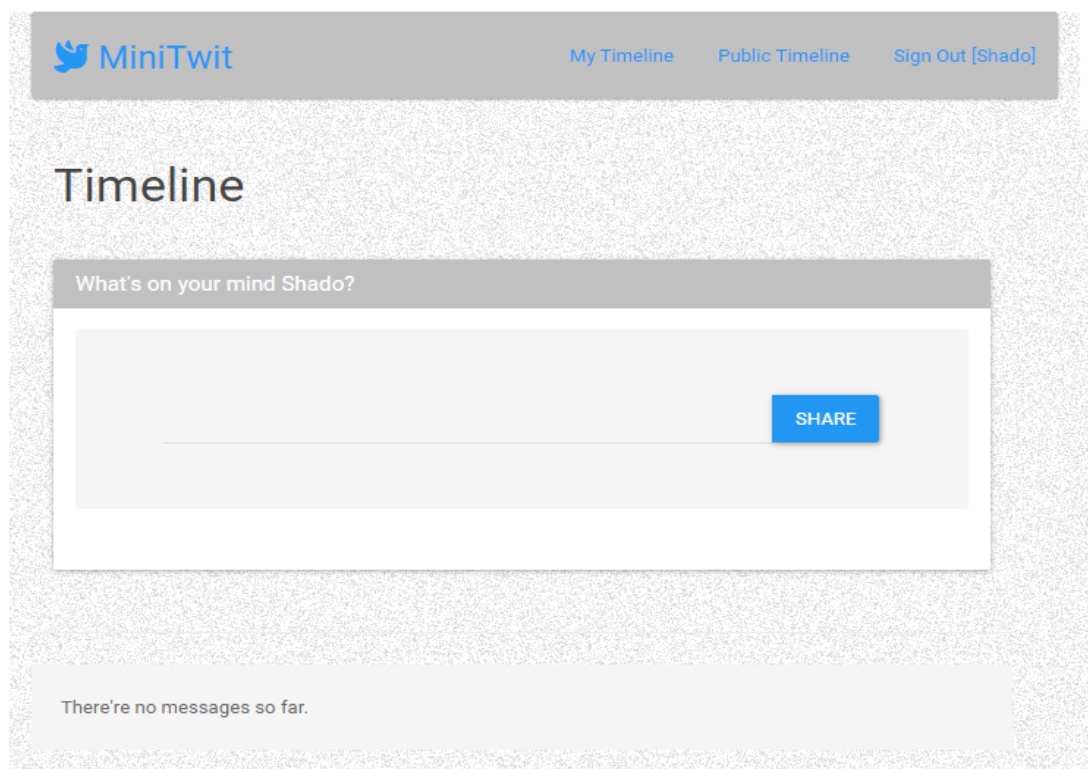
Slika 5. Forma za pristup sa postojećim nalogom

Nakon uspešnog prijavljivanja ili registracije, opcije u glavnom meniju se menjaju, korisnik može da pristupi svom zidu objava gde može da pravi nove objave, javnom zidu objava i ima opciju da se izloguje iz svog naloga (*slika 6*).

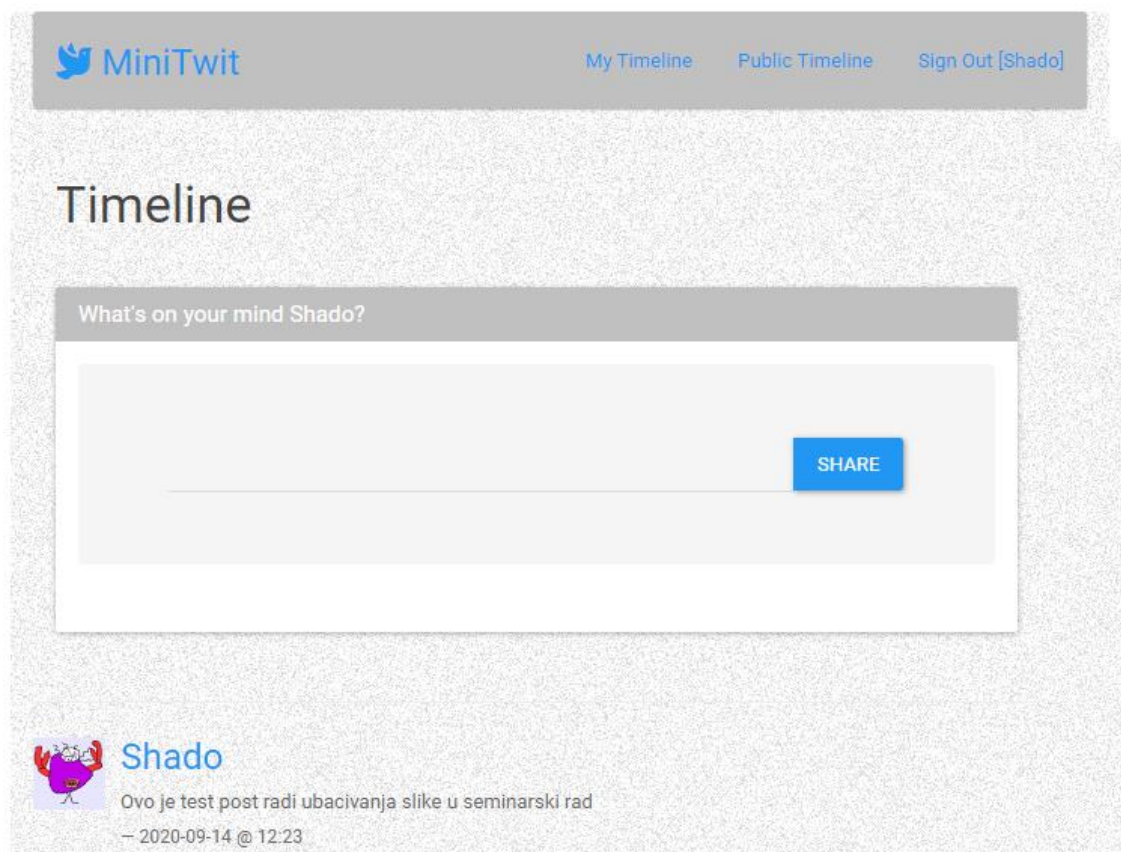


Slika 6. Meni nakon što korisnik pristupi svom nalogu

Kada korisnik pristupi “My timeline” dobija pristup formi za deljenje novih objava (slika 7), i može da vidi istoriju svojih objava (slika 8).

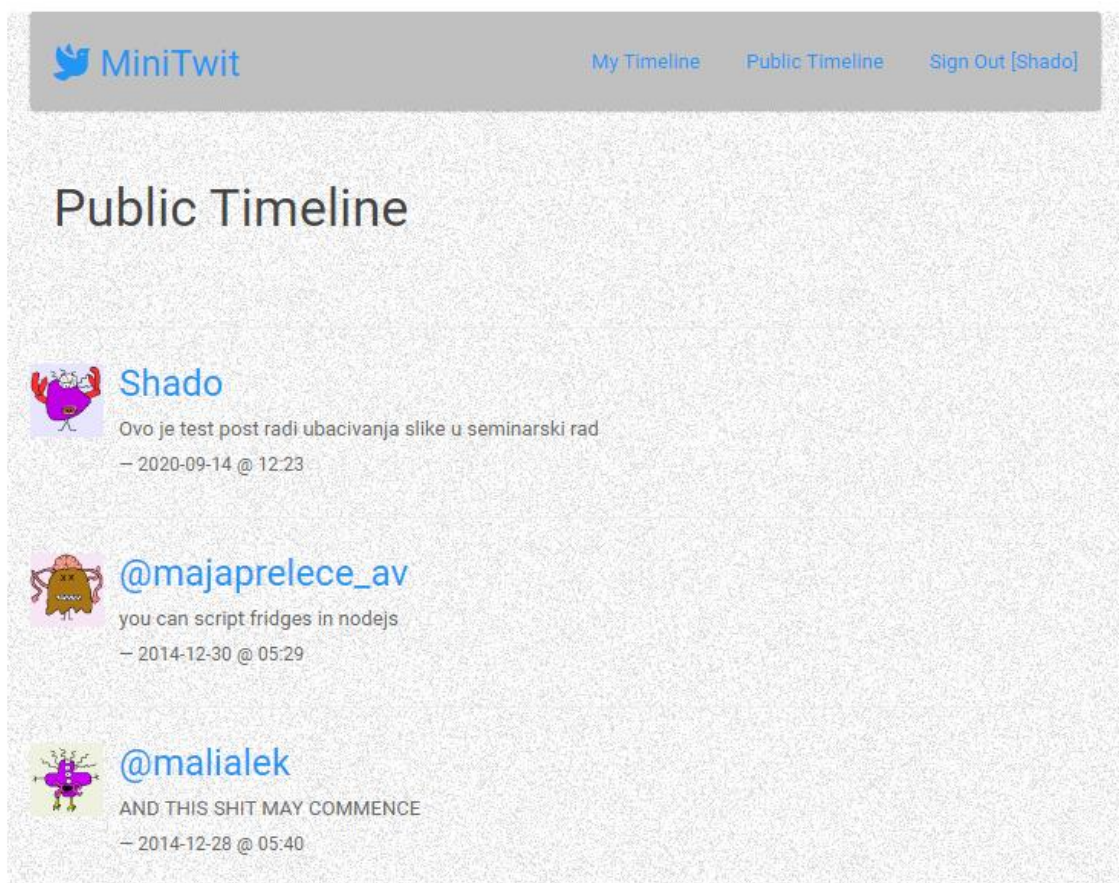


Slika 7. Forma za novu objavu nakon što korisnik svojim nalogom pristupi sajtu



Slika 8. Objava korisnika na njegovom zidu

Prilikom odlaska na "Public Timeline" stranicu, nove objave od korisnika će se pojaviti na javnom zidu, objave svih korisnika su sortirane od najnovije ka najstarijoj (*slika 9*).



Slika 9. Javni zid nakon što korisnik napravi novu objavu

Footer sekcija sadrži informacije o projektu, ko su autori projekta i koje su tehnologije korišćene (*slika 10*).

Klikom na ime bilo kog od autora prikazuje se sekcija footera sa linkovima za društvene mreže tog autora (*slika 11*). Kako bi se dalo do znanja da su imena klikabilna, kada se hoveruje cursorom preko njih, podvučena linija menja boju u crvenu

MiniTwit — University Assignment

Twitter Clone

Done by:

Aleksa Cakic; SI 23/17

And

Nikola Bobinac; SI 18/17

Tech Stack:

Spark Java, Java Spring, Java FreeMaker Template Engine,
Bootstrap 4 CSS Framework, Apache Maven, Heroku, HSQLDB

Slika 10. Footer sekcija bez linkova za društvene mreže

MiniTwit — University Assignment

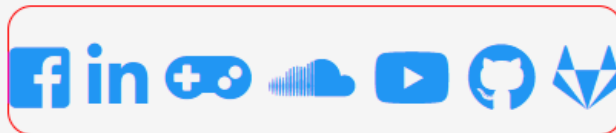
Twitter Clone

Done by:

Aleksa Cakic; SI 23/17

And

Nikola Bobinac; SI 18/17



Tech Stack:

Spark Java, Java Spring, Java FreeMaker Template Engine,
Bootstrap 4 CSS Framework, Apache Maven, Heroku, HSQLDB

Slika 11. Link sekcija autora softvera

5. Prikaz realizacija i implementacije

5.1. Postavka bekenda

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

5.2. Postavke frontenda

Odabir tehnologija za frontend je tekao tako što su se kolege složile da koriste biblioteku Bootstrap 4 CSS Framework, javascript i jQuery za sve frontend poslove. Za slike na frontendu korišćen je servis pod nazivom Gravatar, koji pruža jedinstvene avatare.

5.3. SparkJava

Spark je Javin mikrofrejmwork za pravljenje web aplikacija u kotlinu i Javi 8. Spark Framework je jednostavan i izražajan Java / Kotlin DSL veb framework napravljen za brzi razvoj. Namera Sparka je da pruži alternativu programerima Kotlin / Java onima koji žele da razviju svoje veb aplikacije što ekspresivnije i sa minimalnim uzorkom. Sa jasnom filozofijom, Spark je dizajniran ne samo da deployment učini produktivnijim, već i da kod učini boljim pod uticajem Sparkove elegantne, deklarativne i izražajne sintakse.

Spark je ovom 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 rekveste.

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 autentikovao, ova ruta će preusmeriti korisnika na svoj timeline. Kao post request, radi kao pravi sign in.
- `/register` - Kao get request, predstavlja sign up formu. Ako je korisnik autentikovao, ova putanja će preusmeriti na korisnikov timeline. Kao post radi pravi 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 autentikovao, 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;  
});
```

Tabela 1. Get metoda za unfollow

Otpraćuje trenutnog autentikovanog korisnika kao pratioca datog korisnika, ako korisnik ne postoji, statusni error 404 će biti vraćen. Ako korisnik nije autentikovao, 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());
```

Tabela 2. Metoda za prikazivanje timeline-a korisnika ili redirekciju

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);  
  
}
```

Tabela 3. Proveravanje sesije korisnika

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

Primititi 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;

}
```

Tabela 4. Metoda za timeline ulogovanog korisnika

Koristeći springovu NamedParameterJdbcTemplate klasu, kveri se pokreće sa parametrima koji su prosleđeni iz mape. RowMapper instance 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
);
```

Tabela 5. SQL skripta za kreiranje tabele User

Jedna tabela sačini korisnikove 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 Freemarker Template-u.

Većina ruta proverava da li je korisnik autentikovao 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;

});
```

Tabela 6. Metoda za follow/praćenje korisnika

Prvo, kod proverava da li je korisnik autentikovao. 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";

    }

}

```

Tabela 7. Metoda za registraciju korisnika

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 svojstva iz objekta u mapu. Jedina restrikcija su ključevi (parametri) koji moraju biti identični imenu svojstva u objektu.

Ako 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;  
}
```

Tabela 8. Validacija da li su unete korisničke informacije dobre

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";  
  
}
```

```
}
```

Tabela 9. Provera da li je korisničko ime zauzeto

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.

5.4. Java Spring

Java spring je frejmwork 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 frejmwork 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)

}

}

```

Tabela 10. Glavna metoda projekta

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;
    }
}

```

Tabela 11. Konfiguracija baze podataka

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;
}
}

```

Tabela 12. Servisni sloj

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 and Nikola Bobinac
 */

```

@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 xx;

}

/**
 * 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 checks if 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;

};

}

```

Tabela 13. Implementacija UserDao sloja

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;
};
}

```

Tabela 14. Implementacija MessageDao sloja

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 menadžovanu 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 and Nikola Bobinac
 */
@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)
    }
}
```

Tabela 15. Main metoda primer 2

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.

5.5.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.

Dependencies 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 deklarirati 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>

```

Tabela 16. Maven konfiguracija

5.6. 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());

```

Tabela 17. Template hendler

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: #0c7c95"></span>

<span class="icon-bar" style="background-color: #0c7c95"></span>

<span class="icon-bar" style="background-color: #0c7c95"></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>
```


</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 — University Assignment

</p>

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

<p>Done by: </p>

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

<p>Aleksa Cakic; SI 23/17 </p>

</div>

<p> And </p>

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

<p>Nikola Bobinac; SI 18/17 </p>

</div>

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

<h2 class="display-4">

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

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

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

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

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

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

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

</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>
```


<p>Tech Stack: </p>

<p>Spark Java,</p>

Java Spring,</p>

Java FreeMaker Template Engine,</p>

Bootstrap 4 CSS Framework,</p>

Apache Maven,</p>

Heroku,</p>

HSQLDB</p>

</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-UO2eT0CpHqdSIQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"

crossorigin="anonymous"></script>

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

integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIlly6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"

crossorigin="anonymous"></script>

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

</body>

```
</html>

</#macro>
```

Tabela 18. Glavni Templejt

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>

```

Tabela 19. Login forma template

Primititi kako je master template importovan I referenciran sa identifierom layout, I kako su vrednosti title parametra prosledene. HTML generisan ovom korisničkom

direktivom će biti zamenjen tamo gde master template definiše nested element.

5.7.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 funkcijama.

```
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

}

```

Tabela 20. CSS stilovi

Bootstrap korišten u ovom projektu je pozamašan. Od kreiranja navbara do kontejnera i responsivnosti sajta. Bootstrap je CSS frejmwork 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 Frejmworkom i kasnije je pripojen u Bootstrap CDN. Font awesome se isto koristi kao i Bootstrap.

5.8.Javascript i jQuery

Javascript, odnosno jQuery ispisan za ovaj projekat se odnosi na linkove za društvene mreže autora. Prilikom klika na neko od imena autora pojavljuje se sekcija koja inače ima *hidden* CSS stil i to je kontejner koji sadrži linkove ka društvenim mrežama za tog autora. Ako je jedan kontejner već otvoren i klinke se na drugog autora već otvoreni kontejner će se zatvoriti i otvoriće se kontejner sa linkovima za drugog autora. Ako se klikne na ime autora za kojeg je kontejner već otvoren taj kontejner će se zatvoriti. Preko jQuery i određenih klasa bez stilove binduju se odgovarajuće on click jQuery akcije sa odgovarajućim elementima u HTML-u.

Javascript i jQuery kod je sledeći:

```
$('.js-nikola-name').on('click', function() {
    if (!$('.js-aleksa-links').hasClass("links-hidden"))
    {
        $('.js-aleksa-links').addClass("links-hidden");
        $('.js-aleksa-links').removeClass("border-links-aleksa");
    }
    if ($('.js-nikola-links').hasClass("links-hidden"))
    {
        $('.js-nikola-links').removeClass("links-hidden");
        $('.js-nikola-links').addClass("border-links-nikola");
    }
    else
    {
        $('.js-nikola-links').addClass("links-hidden");
        $('.js-nikola-links').removeClass("border-links-nikola");
    }
});

$('.js-aleksa-name').on('click', function() {
    if (!$('.js-nikola-links').hasClass("links-hidden"))
```

```

        {
            $('js-nikola-links').removeClass("border-links-nikola");
            $('js-nikola-links').addClass("links-hidden");
        }
    if ($('js-aleksa-links').hasClass("links-hidden"))
    {
        $('js-aleksa-links').addClass("border-links-aleksa");
        $('js-aleksa-links').removeClass("links-hidden");
    }
    else
    {
        $('js-aleksa-links').addClass("links-hidden");
        $('js-aleksa-links').removeClass("border-links-aleksa");
    }
});

```

Tabela 21. Javascript i jQuery skripta za footer

5.9.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 predstavlja direktorijum koji sadrži fajlove projekta i poseban sistemski direktorijum 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 (slika 12).

Default branch			
master	Updated 15 hours ago by Alexayy	✓	Default
Your branches			
nikola-fix	Updated last month by Nikola Bobinac	✓	11 0
nikola-fix-footer	Updated last month by Shadochi	✓	13 0
nikola-fix-dao	Updated last month by Nikola Bobinac	✓	42 0
Active branches			
nikola-fix	Updated last month by Nikola Bobinac	✓	11 0
nikola-fix-footer	Updated last month by Shadochi	✓	13 0
nikola-fix-dao	Updated last month by Nikola Bobinac	✓	42 0
Stale branches			
dependabot/maven/spring.version-5.2.6.RELEASE	Updated 4 months ago by dependabot[bot]	✓	59 1
science-paper	Updated 4 months ago by Alexayy		46 1

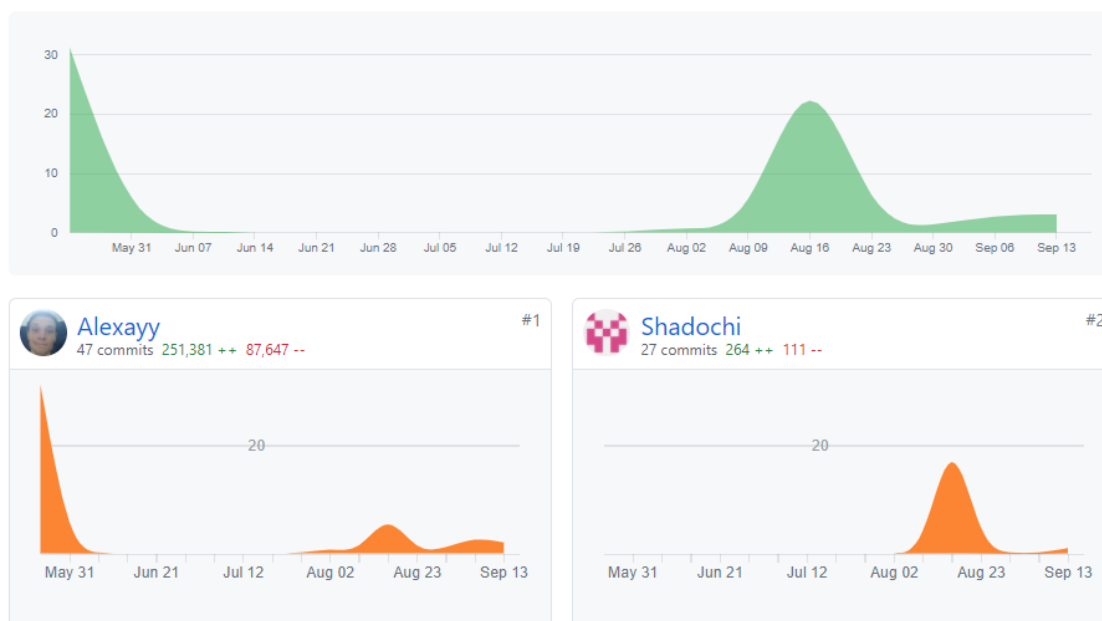
Slika 12. Grananje na gitu iz ugla kolege Nikola Bobinac

Obe kolege su radile na razvoje aplikacije i imaju broj commitova u dvocifrenim brojkama (slika 13.)

May 24, 2020 – Sep 17, 2020

Contributions: Commits ▾

Contributions to master, excluding merge commits



Slika 13. Grafik commitova

5.10. Deployement 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 dependency-je i zapakuje ih, u ovom slučaju u War file.

Pošto inicijalna konfiguracija nije uspeła, 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)  
}
```

Tabela 22. Metoda koja vraća određeni port na Heroku serveru

5.11. Baza podataka

Baza podataka je lokalna in-memory, i zbog toga nosi ime HSQLDB7261F6855D, pošto se kreira tokom deploymenta sajta.

Baza podataka se sastoji iz 3 tabele:

- Korisnik (user)
- Pratioc (follower)
- Poruka (message)

```
drop table user if exists;  
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
);

drop table follower if exists;
create table follower (
    follower_id integer,
    followee_id integer
);

drop table message if exists;
create table message (
    message_id integer primary key GENERATED BY DEFAULT AS
IDENTITY(START WITH 100),
author_id integer not null,
text varchar(160) not null,
pub_date timestamp
);

```

Tabela 23. SQL naredbe za kreiranje tabela

5.12. Testiranje Aplikacije

Za testiranje većine aplikacije korišćena je end-to-end testing metodologija. Ova metodologija testira tok aplikacije od početka do kraja. Uloga ove metodologije jeste da simuliše scenario pravog korisnika i validira sistem i njegove komponente za integraciju i integritet podataka.

Za određene API rute je takođe korišćena alatka Postman, koja pomaže pri testiranju API endpoint-ova RESTful API aplikacija ili bilo kojih drugih endpoint-ova programa. Postman dopušta da se bez prevelike muke pozivaju API rute i lako menjaju parametri po potrebi kako bi se moglo brzo testirati što više slučajeva podataka.

6. Korišteni alati i softveri

1. JetBrains IntelliJ IDEA
2. Visual Studio Code
3. Spark Java
4. Java Spring
5. Java FreeMaker Template Engine
6. Bootstrap 4 CSS Framework
7. Apache Maven
8. jQuery
9. Heroku
10. HSQLDB
11. Postman

7. Literatura

1. <http://sparkjava.com/tutorials/twitter-clone>
2. <https://www.manning.com/books/spring-in-action-fifth-edition>
3. <https://www.geeksforgeeks.org/jquery-tutorials/>
4. <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
5. <https://www.tutorialspoint.com/spring/index.htm>
6. <https://help.superhosting.rs/sta-je-to-git/>
7. <https://studenti.rs/skripte/informacione-tehnologije/maven/>