Web aplikacija za organizaciju timskog rada

Staković, Jakov

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni prijediplomski studij Računarstvo

WEB APLIKACIJA ZA ORGANIZACIJU TIMSKOG RADA

Završni rad

Jakov Staković

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Jakov Staković
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4712, 28.07.2021.
JMBAG:	0165091868
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Web aplikacija za organizaciju timkog rada
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Kratko objasniti načine organizacije timskog rada. Izraditi web aplikaciju koja olakšava suranju članova tima pri izradi zadataka, praćenju napretka i komunikaciju između članova tima. Navesti nekoliko primjera postojećih sličnih gotovih rješenja te napraviti usporedbu izrađene aplikacije s postojećim rješenjima te naglasiti prednosti i nedostatke izrađene aplikacije. Navesti zahtjeve i funkcionalnosti koje treba imati web aplikacija. Opisati postupak izrade web aplikacije kao i njene funkcionalnosti. Projektirati i izraditi bazu podataka koju će web aplikacija koristiti za pohranu svih relevantnih podataka. Predvidjeti sljedeće korisničke profile s

Datum prijedloga ocjene završnog rada od strane mentora:	06.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	11.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	17.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 17.09.2024.

Ime i prezime Pristupnika:	Jakov Staković
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4712, 28.07.2021.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za organizaciju timkog rada**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U suvremenom poslovnom okruženju, učinkovita organizacija timskog rada postala je presudna za postizanje uspjeha. Sa sve većim brojem projekata koji zahtijevaju suradnju među različitim timovima i članovima, potreba za pouzdanim i funkcionalnim alatima za upravljanje timskim radom nikada nije bila veća. Tradicionalni alati poput e-pošte i sastanaka uživo često nisu dovoljni za upravljanje složenim zadacima, koordinaciju članova tima i praćenje napretka projekta. Web aplikacije za organizaciju timskog rada nude suvremeno rješenje koje omogućuje bolju komunikaciju, koordinaciju i upravljanje projektima. Ove aplikacije objedinjuju različite funkcionalnosti kao što su zadaci, kalendari, dokumentacija i komunikacijski kanali u jednu platformu koja je dostupna s bilo kojeg mjesta. Korištenjem takvih alata, timovi mogu povećati svoju produktivnost, smanjiti mogućnost pogrešaka i bolje iskoristiti svoje resurse. Cilj ovog završnog rada je razvoj web aplikacije koja će pojednostaviti organizaciju timskog rada. Naglasak će biti na kreiranju intuitivnog korisničkog sučelja, integraciji ključnih funkcionalnosti za upravljanje zadacima, vremenskim rasporedima kao i osiguravanju sigurnosti podataka. Analizom postojećih rješenja i primjenom suvremenih tehnologija, ovaj rad pruža sveobuhvatan pregled procesa razvoja web aplikacije, od planiranja i dizajna do implementacije i testiranja. Struktura rada podijeljena je u pet poglavlja.

U drugom poglavlju analiziraju se slične aplikacije koje su prisutne u današnje vrijeme te aplikacija koja je poslužila kao inspiracija za ovaj završni rad. Treće poglavlje prikazuje funkcionalne zahtjeve kako bi se jasnije objasnilo kako web aplikacija treba raditi. Nadalje, četvrto poglavlje detaljno opisuje strukturu korisničkog sučelja, uz sve potrebne informacije o kodu odgovornom za funkcionalnosti aplikacije. Takvim pristupom, čitatelji dobivaju bolji uvid u implementaciju aplikacije i tehničke aspekte koji su ključni za njezino funkcioniranje.

1.1. Zadatak završnog rada

Kratko objasniti načine organizacija timskog rada. Izraditi web aplikaciju koja olakšava suradnju članova tima pri izradi zadataka, praćenju napretka i komunikaciju između članova tima. Navesti nekoliko primjera postojećih sličnih gotovih rješenja te napraviti usporedbu izrađene aplikacije s postojećim rješenjima te naglasiti prednosti i nedostatke izrađene aplikacije. Navesti zahtjeve i funkcionalnosti koje treba imati web aplikacija. Opisati postupak izrade web aplikacije kao i njene funkcionalnosti. Projektirati i izraditi bazu podataka koju će web aplikacija koristiti za pohranu

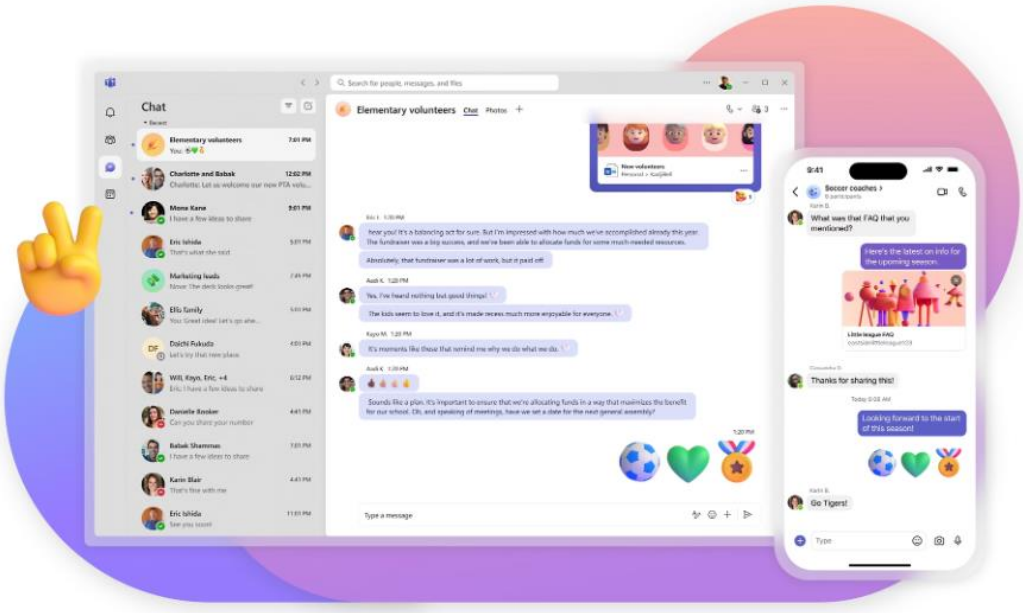
svih relevantnih podataka. Predvidjeti sljedeće korisničke profile s odgovarajućim pravima pristupa: administrator web aplikacije, voditelj organizacije, članovi tima.

2. PREGLED SLIČNIH RJEŠENJA

Trenutno postoji puno aplikacija koje se koriste unutar organizacija ili kompanija kako bi se uspostavila dobra komunikacija, koordinacija, praćenje napredaka na projektu i timski rad. Jedna od vodećih aplikacija koja je bila inspiracija za kreiranje aplikacije je Microsoft Teams. Osim Microsoft Teamsa, aplikacije poput Zooma, Slacka, Jire, i Mondaya također su vrlo popularne u korporacijama, za osobne potrebe i u privatnim sektorima.

2.1. Microsoft Teams

Microsoft Teams [1] nudi osnovne funkcionalnosti registracije i prijave korisnika unutar šireg Microsoftovog ekosustava, što može uključivati organizacijske račune. Administrator ima opsežna ovlaštenja za upravljanje korisnicima, timovima i postavkama organizacije. Također koristi specifične uloge kao što je Team Owner, koji može kreirati i upravljati timovima i kanalima te moderirati sadržaj unutar tima. Team Member, koji se može usporediti s običnim korisnikom u aplikaciji, može sudjelovati u razgovorima, upravljati sadržajem unutar svojih timova i koristiti integrirane alate i resurse. S druge strane, aplikacija prikazana u istraživanju omogućuje korisnicima registraciju i prijavu s različitim pravima prema ulozi korisnika. Slika 2.1. prikazuje Teams aplikaciju i GUI za komunikaciju.



Sl. 2.1. *Teams aplikacija za mobitel i računalo*

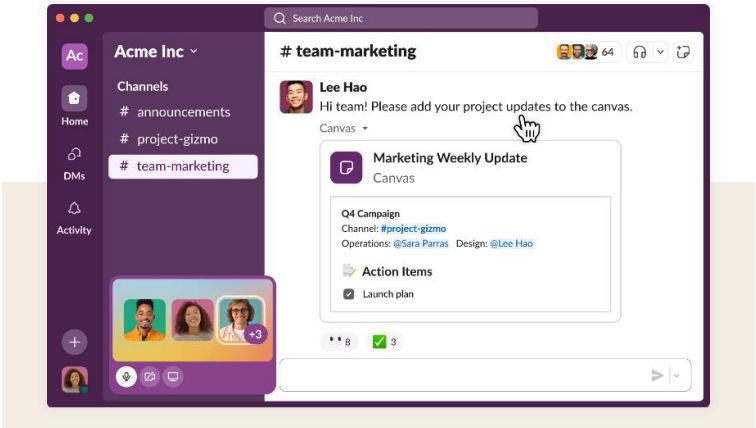
2.2. Zoom

Zoom [2] se fokusira na video konferencije i virtualne sastanke. Administrator u Zoomu ima široke ovlasti za upravljanje korisnicima, sastancima i webinarima. Organizator (engl. *host*) sastanka upravlja postavkama i sudionicima sastanka, dok sudionik (engl. *participant*) sudjeluje u sastancima i koristi funkcionalnosti vezane uz video i audio komunikaciju. Slika 2.2. prikazuje Zoom funkcionalnosti.

Sl. 2.2. *Zoom funkcionalnosti*

2.3. Slack

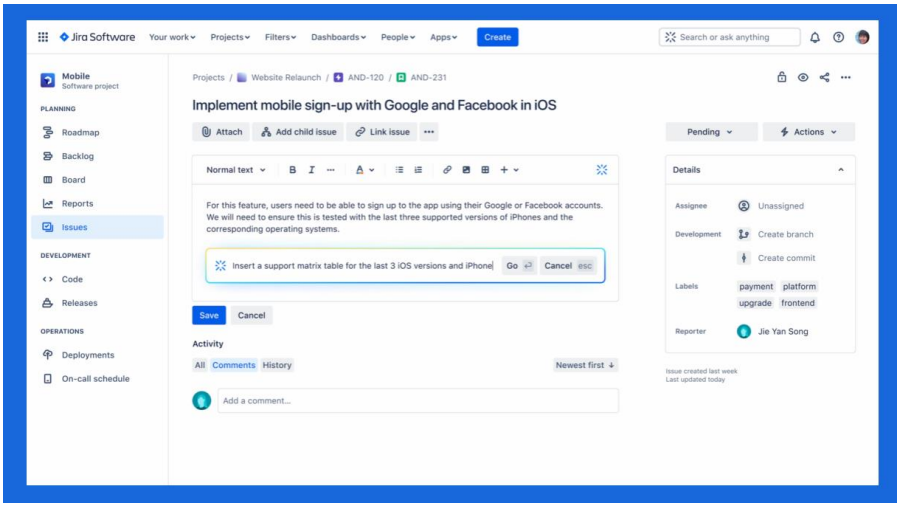
Slack [3] se usredotočuje na timsku komunikaciju putem chat kanala. Administrator upravlja korisnicima, timovima i postavkama organizacije. Upravljač kanala (engl. *channel owner*) upravlja specifičnim kanalima, dok sudionik (engl. *Member*) može sudjelovati u razgovorima i koristiti funkcionalnosti kanala. Slika 2.3. prikazuje Slack funkcionalnosti aplikacije za računalo.



Sl. 2.3. *Slack aplikacija za računalo*

2.4. Jira

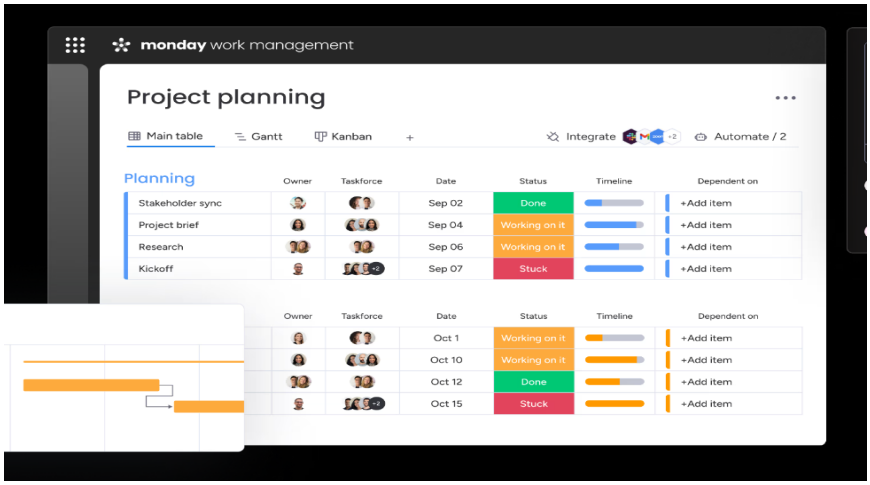
Jira [4] pruža opsežne mogućnosti za upravljanje projektima i zadacima, uključujući fleksibilne radne tokove i napredne mogućnosti praćenja napretka. Aplikacija prikazana u istraživanju nudi specifična prava za administraciju i upravljanje timovima i organizacijama. Jira se ističe kao alat za detaljno upravljanje projektima i zadacima, s naglaskom na agilne metodologije i praćenje rada. Slika 2.4. prikazuje Jira sučelje.



Sl. 2.4. *Jira aplikacija*

2.5. Monday.com

Monday.com [5] pruža vizualne alate za upravljanje projektima i radnim procesima. Administrator ima ovlasti za kreiranje i konfiguriranje radnih prostora, upravljanje korisnicima i postavkama sustava. Menadžer projekta (engl. *project manager*) kontrolira specifične projekte, dok član tima (engl. *team member*) može raditi na zadacima, ažurirati status i surađivati s kolegama. Slika 2.5. prikazue Monday.com aplikaciju.



Sl. 2.5. *Monday.com aplikacija*

2.6. Usporedba s navedenim platformama

U usporedbi s navedenim platformama, izrađena aplikacija omogućuje korisnicima registraciju i prijavu s različitim pravima prema ulozi. Administrator ima opsežna ovlaštenja uključujući brisanje korisnika, upravljanje timovima i organizacijama, te kreiranje i povezivanje timova. Voditelj (engl. *leader*) ima specifična prava vezana uz projekte, uključujući njihovo kreiranje, komentiranje, uređivanje i brisanje, dok obični korisnici mogu pregledavati projekte za timove u kojima se nalaze te upravljati svojim komentarima.

Iako Microsoft Teams, Zoom, Slack, Monday.com i Jira nude specifične alate i funkcionalnosti prilagođene različitim potrebama, aplikacija prikazana u istraživanju fokusira se na detaljno upravljanje korisnicima, timovima i projektima unutar organizacija. Svaka platforma ima svoje jedinstvene značajke koje odgovaraju specifičnim radnim procesima i potrebama korisnika, dok aplikacija iz istraživanja pruža specifična prava i funkcionalnosti za organizacijsko upravljanje i suradnju na projektima.

3. ZAHTJEVI, STRUKTURA I ARHITEKTURA WEB APLIKACIJE

Da bi se funkcionalnost web aplikacije lakše razumjela, objašnjeni su njezini funkcionalni zahtjevi. Ovi zahtjevi omogućuju različitim dijelovima ili modelima unutar aplikacije da međusobno komuniciraju i budu povezani. Što su struktura i model bolje osmišljeni, to je lakše implementirati ključne funkcionalnosti i usmjeriti se na njih, dok se istovremeno izbjegava nepotrebna implementacija sekundarnih elemenata.

3.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi predstavljaju skup karakteristika ili funkcija koje proizvod treba imati kako bi korisnici mogli učinkovito izvršavati svoje zadatke. Pomoću njih detaljno se opisuje kako se sustav treba ponašati u određenim uvjetima. Timovi imaju lakši uvid u razumljivost i jasnoću same aplikacije i njezine izrade.

Zadovoljeni funkcionalni zahtjevi unutar web aplikacije:

- Prijava i registracija korisnika
- Kreiranje korisnika s različitim ulogama (engl. *roles)*
- Svim korisnicima je omogućen prikaz njihovih podataka
- Različito korisničko sučelje ovisno o dodijeljenim ulogama
- Odjava korisnika
- Kreiranje administratorskog (engl. *admin*) korisnika s posebnim mogućnostima
- Korisnik s ulogom admin kreira, uređuje i briše tim sa/bez organizacijom/e (engl. *corporations*)
- Korisnik s ulogom admin kreira, uređuje i briše organizaciju s/bez timovima/a (engl. *teams*)
- Korisnik s ulogom admin briše druge korisnike
- Korisnik s ulogom admin omogućuje dodavanje korisnika u tim
- Korisnik s ulogom admin omogućuje dodavanje tima u organizaciju
- Korisnik s ulogom admin dohvaća sve korisnike, timove, organizacije
- Korisnik s ulogom voditelj kreira, uređuje i briše projekt unutar organizacije za pojedine timove
- Korisnik s ulogom voditelj dohvaća sve projekte vezane za organizaciju
- Korisnik s ulogom voditelj dohvaća sve timove unutar organizacije
- Korisnik s ulogom voditelj briše komentare vezane za određeni projekt

- Korisnik s ulogom korisnik (engl. *user)* omogućuje kreiranje i uređivanje komentara na projektu

Ova web aplikacija prvenstveno je dizajnirana kako bi olakšala rad unutar organizacija i timova. Korisnici se mogu registrirati i prijaviti u sustav. Pri registraciji, podatci se pohranjuju u bazu, a prilikom prijave korisnik dobiva pristup sučelju koje se otključava tek nakon uspješne prijave u sustav. Ključna značajka sustava je mogućnost stvaranja korisnika s različitim ulogama, kao što su administrator, voditelj organizacije i običan korisnik. Svaka uloga ima različit prikaz sučelja s funkcionalnostima prilagođenim njihovim potrebama. Administrator može upravljati organizacijama, timovima i korisnicima, uključujući brisanje i dodavanje članova unutar timova i organizacija, kao i kreiranje, brisanje i uređivanje organizacija i timova. Uloga voditelja organizacije omogućuje kreiranje, brisanje i uređivanje projekata unutar organizacije, te upravljanje timovima. Komunikacija unutar sustava odvija se putem komentara ispod projekata, a svaki korisnik ima mogućnost kreiranja, uređivanja i brisanja komentara.

3.2. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi definiraju atribute kvalitete sustava. Oni predstavljaju skup standarda korištenih za određivanje specifičnih operacija. Pomoću njih se mogu odrediti ograničenja na dizajn sustava u različitim situacijama. Neispunjavanje nefunkcionalnih zahtjeva može dovesti do nezadovoljstva korisnika ili kupaca sustava Potrebni nefunkcionalni zahtjevi su:

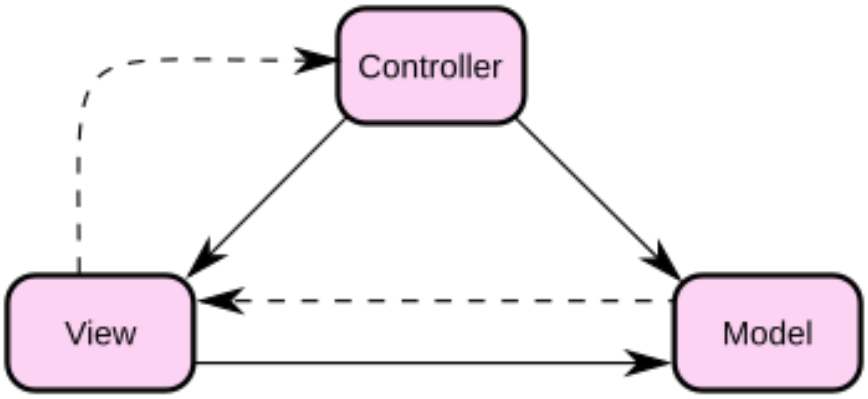
- Sigurnost
- Pouzdanost
- Upotrebljivost
- Performanse
- Skalabilnost
- Dostupnost

Sigurnost podrazumijeva zaštitu osjetljivih korisničkih informacija i sprječavanje neovlaštenog pristupa sustavu. U ovom radu, primijenjene su odgovarajuće sigurnosne mjere kako bi se osiguralo da samo ovlašteni korisnici mogu pristupiti podatcima i funkcijama. Pouzdanost se odnosi na kontinuitet i stabilnost sustava. Upotrebljivost predstavlja grafičko sučelje prilagođeno korisniku radi lakšeg snalaženja prilikom korištenja aplikacije. Rad se fokusirao na intuitivno grafičko sučelje koje korisnicima omogućuje lakše korištenje aplikacije i jednostavniju navigaciju Performanse su od presudne važnosti za brzinu i učinkovitost sustava u ispunjavanju korisničkih

zahtjeva. Skalabilnost je ključna pri proširivanju aplikacije i ispunjavanju novih zahtjeva korisnika. Brzina sustava ne bi smjela opadati s dodavanjem novih funkcionalnosti u aplikaciju u odnosu na prethodne verzije. Dostupnost je važna kako bi korisnici mogli pristupiti sustavu u bilo kojem trenutku. Svi ovi nefunkcionalni zahtjevi detaljno su analizirani i uzeti u obzir tijekom dizajniranja i implementacije sustava.

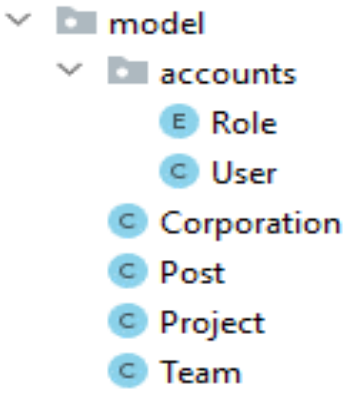
3.3. MVC Arhitektura (engl. *Model View Controller)*

Model-View-Controller (MVC) [9] je softverski obrazac za izradu koji se često koristi u razvoju korisničkih sučelja, posebno za web aplikacije. Aplikacija je podijeljena prema navedenom obrascu na tri komponente: model, pogled (engl. *View)* i kontroler (engl. *Controller),* što omogućuje učinkovitu organizaciju koda, lakše održavanje i učinkovitiji razvoj. Model predstavlja podatke aplikacije i poslovnu logiku. On izravno upravlja podatcima, logikom i pravilima aplikacije. Komunicira s bazom podataka te je odgovoran za dohvaćanje, pohranu i obradu podataka. Model je neovisan o korisničkom sučelju, što znači da promjene u sučelju ne utječu na rukovanje podatcima. Pogled je komponenta koja prikazuje podatke korisniku. On predstavlja korisničko sučelje aplikacije. Pogledi su ono s čim korisnik izravno komunicira, a zaduženi su za prikaz podataka koje model dohvaća na način prilagođen korisniku. Pogled se obično ažurira kada dođe do promjene u modelu, kako bi korisnik uvijek imao uvid u najnovije podatke. Dizajniran je tako da bude dinamičan i prilagodljiv korisničkom unosu. Kontroler djeluje kao posrednik između modela i pogleda. On upravlja korisničkim unosom i ažurira model prema potrebi. Kada korisnik komunicira s pogledom (na primjer klikne gumb ili pošalje obrazac), kontroler obrađuje taj unos i obavlja posao ovisno o funkcionalnostima, često uključujući ažuriranje modela i pogleda. Na slici 3.1. [6]prikazani su odnos, opis i tijek funkcionalnosti u MVC arhitekturi.

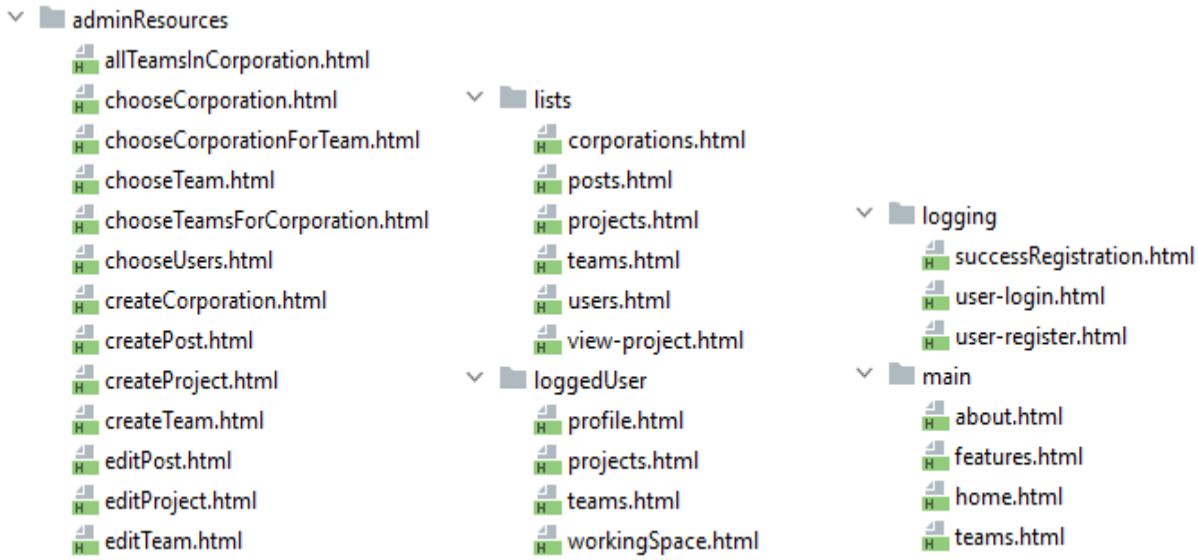


Sl. 3.1. *Vizualni primjer Model View Controller*

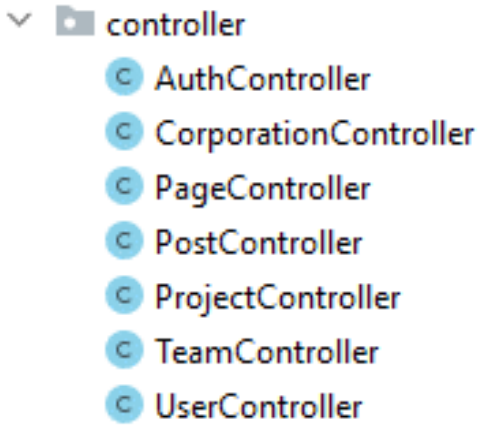
U ranim fazama interneta, MVC arhitektura se primarno koristila na poslužiteljskoj strani, gdje bi klijenti putem obrazaca ili veza slali zahtjeve za ažuriranja i dobivali osvježene prikaze koji su se prikazivali u pregledniku. No, danas se sve više logike prebacuje na klijentsku stranu zahvaljujući razvoju klijentskog skladišta podataka i XMLHttpRequest tehnologiji koja omogućava djelomično osvježavanje stranica po potrebi. Na sljedećim slikama je prikazano rješenje MVC arhitekture unutar aplikacije. Slika 3.2 prikazuje organizaciju svih modela unutar aplikacije koji se koriste kako bi se postigla povezanost u bazi podataka. Slika 3.3. prikazuje sve Viewove kojima korisnik ima pristup te slika 3.4. prikazuje kontrolere koji omogućuju pristup Viewovima.



Sl. 3.2. *Model u aplikaciji*



Sl. 3.3. *Pogled u aplikaciji*



Sl. 3.4. *Kontroler u aplikaciji*

3.4. Arhitektura sustava na klijentskoj strani

Arhitektura sustava [7] na klijentskoj strani obuhvaća sve komponente i procese koji se odvijaju u pregledniku korisnika, uključujući korisničko sučelje i dio aplikacijske logike. Thymeleaf u kombinaciji s HTMLom (engl. *HyperText Markup Language*)i CSSom (engl. *Cascading Style Sheets)* omogućuje kreiranje korisničkog sučelja i dinamičko umetanje podataka iz modela. Bootstrap [8] olakšava primjenu već gotovih klasa koje utječu na dizajn i responzivnost, omogućujući prilagodbu stranica različitim uređajima.

3.5. Arhitektura sustava na strani poslužitelja

Arhitektura sustava na strani poslužitelja predstavlja ključnu komponentu u dizajnu i funkcioniranju modernih aplikacija. Ova arhitektura [9] uključuje sve procese i komponente koji se odvijaju na poslužitelju, s ciljem obrade poslovne logike i upravljanja zahtjevima klijenata. U srži ove arhitekture nalazi se model, koji je odgovoran za implementaciju poslovne logike, obradu podataka i pristup bazi podataka. Model obuhvaća sve operacije vezane uz manipulaciju i pohranu podataka, kao i pravila koja upravljaju kako se podatci obrađuju i koriste unutar sustava. Java se često koristi za implementaciju ovih komponenti, dok sloj za bazu podataka omogućuje komunikaciju s bazom radi dohvaćanja, ažuriranja i pohrane podataka. Na slikama 3.5. do 3.9. prikazani su pojedini modeli u aplikaciji i kako su međusobno povezani. Slika 3.10. prikazuje dijagram toka aplikacije.

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private String email;
    private String address;

    private Long leader;


    @Enumerated(EnumType.STRING)
    public Set<Role> roles;

    @ManyToOne
    @JoinColumn(name = "corporationId")
    private Corporation corporation;

    @ManyToMany(mappedBy = "employees")
    private List<Team> teams;
}
```

Sl. 3.5. *Prikaz modela korisnika*

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "corporations")
public class Corporation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(
            mappedBy = "corporation",
            fetch = FetchType.EAGER,
            cascade = CascadeType.REMOVE)
    private Set<Team> teams = new HashSet<>();

    @OneToOne
    @JoinColumn(name = "representative_id")
    private User representative;
}
```

Sl. 3.6. *Prikaz modela organizacije*

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "teams")
public class Team {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String description;

    @ManyToOne
    @JoinColumn(name = "corporation_id")
    private Corporation corporation;

    @ManyToMany
    @JoinTable(
            name = "team_employee",
            joinColumns = @JoinColumn(name = "teamId"),
            inverseJoinColumns = @JoinColumn(name = "userId")
    )
    private List<User> employees;

    @OneToMany(mappedBy = "team")
    private List<Project> projects;
}
```

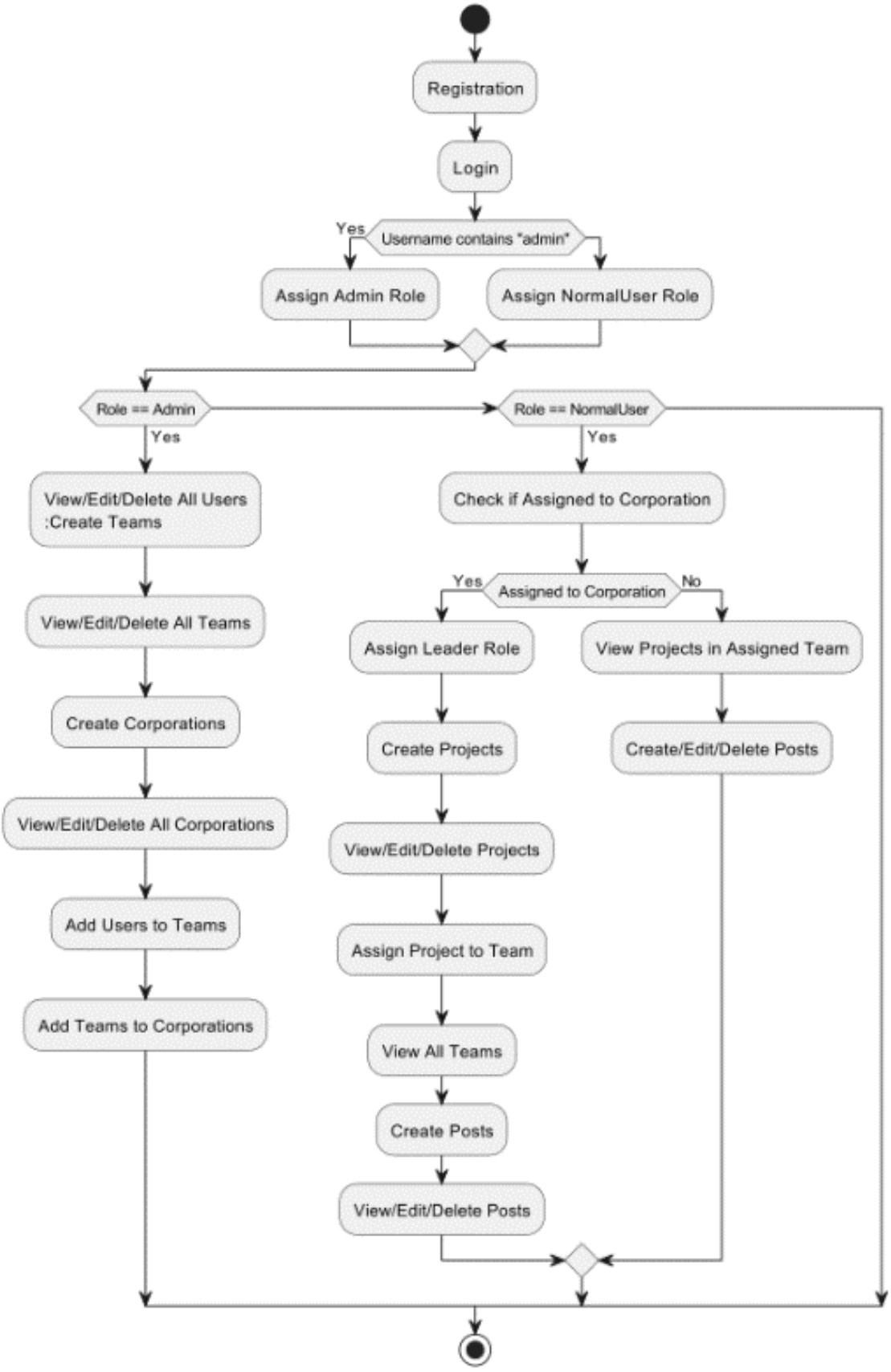
Sl. 3.7. *Prikaz modela tima*

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "projects")
public class Project {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String description;
    private boolean active;
    private LocalDate startDate;
    private LocalDate endDate;
    @ManyToOne
    @JoinColumn(name = "teamId")
    private Team team;

    @OneToMany(mappedBy = "project", cascade = CascadeType.ALL)
    private List<Post> posts;
}
```

Sl. 3.8. *Prikaz modela projekta*

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "posts")
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String content;

    private String title;

    private Timestamp timestamp;

    @ManyToOne
    @JoinColumn(name = "userId")
    private User user;

    @ManyToOne
    @JoinColumn(name = "projectId")
    private Project project;

}
```

Sl. 3.9. *Prikaz modela objave*



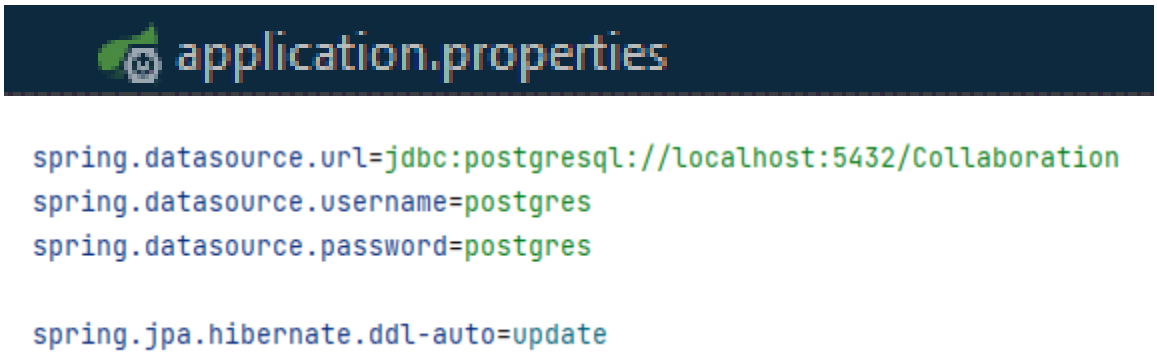
Sl. 3.10. *Prikaz dijagrama toka aplikacije*

4. POSTUPAK IZRADE WEB APLIKACIJE

Ovo poglavlje prikazuje programski kod s kojim se postigla željena funkcionalnost [7] same aplikacije. Korisnici detaljno mogu vidjeti postupke gdje i na koji način se izvršava pojedini dio koda. Prije nego što započnemo s objašnjavanjem koda, važno je napomenuti da su glavne tehnologije korištene za razvoj same aplikacije Spring Boot i Java.

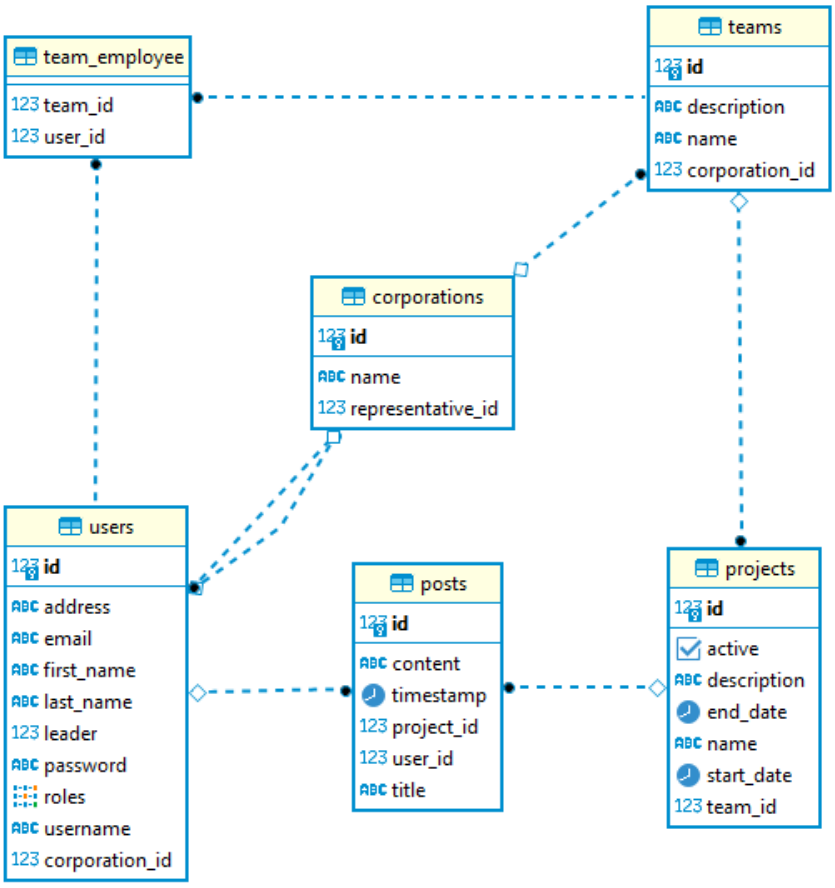
4.1. Spajanje na bazu podataka (PostgreSQL)

Svaka aplikacija treba imati bazu unutar koje će spremati određene resurse. U ovoj aplikaciji korištena je PostgreSQL baza. Kako bi se okvir (engl. *framework*) u kojem je rađena sama aplikacija (Spring Boot) mogao spojiti na bazu podataka trebaju se definirati vrijednosti unutar application.properties datoteke u Spring [10] [10] Boot frameworku.



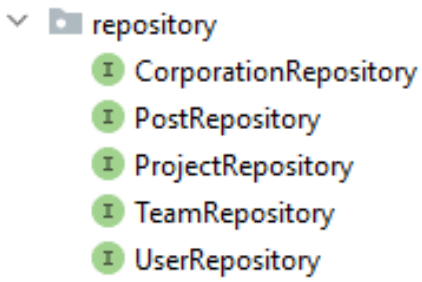
Sl. 4.1. *Datoteka sa svojstvima za spajanje na bazu podataka*

Prema slici 4.1. unutar same application.properties datoteke sa *spring.datasource.url* linijom postavlja se URL putanja za povezivanje s bazom podataka koja se nalazi lokalno na računalu. Ovaj URL sadrži sve potrebne informacije koje Spring Boot aplikaciji trebaju da bi se povezala s lokalnom bazom podataka. *Spring.datasource.username* i *spring.datasource.password* omogućuju pristupanje bazi pomoću korisničkog imena i zaporke definiranih u samoj bazi podataka. Kako bi se neprestano mogli ažurirati podatci u bazi koristi se *spring.jpa.hibernate.ddl-auto=update*. Na slici 4.2. prikazan je odnos modela u bazi pomoću ER dijagrama.



Sl. 4.2. *ER dijagram baze podataka*

Spring framework ima komponentu *JpaRepository* (Java Persistence Application Programming Interface) koja se koristi za rad s bazama podataka putem Java Persistence Application Programming Interfacea (JPA). *JpaRepository* je proširenje *PagingAndSortingRepository*, koje pak proširuje *CrudRepository*. Ova klasa omogućava rad s entitetima (objektima koji su mapirani na tablice u bazi podataka) i pruža razne metode za rad s njima, kao što su spremanje, ažuriranje, brisanje i dohvaćanje podataka. Kao što su modeli definirani i prikazani slikama u prethodnom poglavlju, na slikama od 4.3. do 4.8. prikazana su sučelja za svaki od njih koja omogućuju spremanje, ažuriranje, brisanje i dohvaćanje podataka.



Sl. 4.3. *JpaRepository za komunikaciju s bazom*

Potrebno je kreirati novo sučelje koje će naslijediti sve funkcionalnosti od *JpaRepositorya*. Unutar zagrada < T , ID > predan je model (T) po kojemu se pretražuje po bazi te primarni ključ (ID) koji je definiran unutar modela. Pored već postojećih metoda koje sam *JpaRepository* pruža, mogu se definirati pretraživanja po bazi koje će *JpaRepository* prepoznati. U Spring framework dokumentaciji mogu se pronaći postupci kreiranja funkcija za pretraživanje određenog modela po bazi.

```
public interface UserRepository extends JpaRepository<User,Long> {
    3 usages  JakovStakovic
    User findByEmail(String email);

    2 usages  JakovStakovic
    User findByUsername(String username);

    1 usage  JakovStakovic
    void deleteById(Long id);

    1 usage  JakovStakovic
    List<User> findByLeaderIsNull();

    1 usage  JakovStakovic
    List<User> findAllByIdIn(List<Long> ids);

    1 usage  JakovStakovic
    @Query("SELECT u FROM User u WHERE u.id NOT IN " +
            "(SELECT e.id FROM Team t JOIN t.employees e WHERE t.id = :teamId)")
    List<User> findUsersNotInTeam(@Param("teamId") Long teamId);

}
```

Sl. 4.4. *JpaRepository za komunikaciju s bazom za model korisnika*

```
public interface TeamRepository extends JpaRepository<Team, Long> {
    1 usage  JakovStakovic
    List<Team> findByCorporationId(Long corporationId);

    2 usages  JakovStakovic
    Team findByName(String name);

    1 usage  JakovStakovic
    List<Team> findByIdIn(List<Long> ids);

    1 usage  JakovStakovic
    @Query("SELECT t FROM Team t JOIN t.employees u WHERE u.id = :userId")
    List<Team> findByUserId(@Param("userId") Long userId);
}
```

Sl. 4.5. *JpaRepository za komunikaciju s bazom za model tima*

```
public interface CorporationRepository extends JpaRepository<Corporation, Long> {
    2 usages  JakovStakovic
    Corporation findByName(String name);

    1 usage  JakovStakovic
    void deleteById(Long id);

    14 usages  JakovStakovic
    Optional<Corporation> findById(Long id);

    1 usage  JakovStakovic
    Corporation findByRepresentative(User representative);
}
```

Sl. 4.6. *JpaRepository za komunikaciju s bazom za model organizacije*

```
public interface ProjectRepository extends JpaRepository<Project, Long> {
    1 usage  JakovStakovic
    List<Project> findByTeamCorporationId(Long corporationId);
}
```

Sl. 4.7. *JpaRepository za komunikaciju s bazom za model projekt*

```
public interface PostRepository extends JpaRepository<Post, Long> {
    1 usage  JakovStakovic
    List<Post> findByProjectId(Long projectId);
    1 usage  JakovStakovic
    List<Post> findByUser(User user);
}
```

Sl. 4.8. *JpaRepository za komunikaciju s bazom za model objave*

4.2. Registracija i prijava u sustav

Moderne aplikacije koje imaju korisnike zahtijevaju mogućnosti registracije i prijave u sustav. Prilikom registracije podatci postaju izloženi virtualnom svijetu te je važna njihova zaštita. *Spring Security* je ovisnost (engl. *dependency*) koja služi za autentifikaciju i kontrolu pristupa korisnika tijekom njihove prijave. Koristi se kao standard prilikom dizajniranja Spring aplikacija zbog velike fleksibilnosti i prilagodljivosti. Slika 4.9. prikazuje klase kojima se omogućila prijava u sustav.

Sl. 4.9. *Klase za izradu prilagođene sigurnosti*

Kako je potrebna registracija i prijava korisnika u sustav, postavljena su ograničenja unutar aplikacije koja se odnose na korisnike koji nisu prijavljeni u sustav. Time je ograničen pristup funkcionalnostima same aplikacije. Unutar *SecurityConfig* klase postavljaju se ograničenja na dohvaćanje dijelova aplikacije kojima se može pristupiti kada korisnik nije prijavljen. To je ostvareno korištenjem metode *httpSecurity.authorizeHttpRequests()*, koja provjerava autorizaciju korisnika i, ovisno o tome je li korisnik autoriziran, omogućuje pristup novim stranicama za rad. Prema slici 4.10. mogu se vidjeti stranice kojima korisnik ima pravo pristupa ako nije prijavljen u sustav.

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    JakovStakovic +1
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeHttpRequests(auth -> auth
                        .requestMatchers("/**").permitAll()
                        .requestMatchers("/*").permitAll()
                        .requestMatchers("/css/**").permitAll()
                        .requestMatchers("/images/**").permitAll()
                        .anyRequest().authenticated())
                .formLogin(form -> form
                        .loginPage("/login-user")
                        .loginProcessingUrl("/login-user")
                        .defaultSuccessUrl("/home?messageGreen=true")
                        .failureUrl( authenticationFailureUrl: "/login-user?error=true"))
                .logout(form -> form
                        .logoutUrl("/logout")
                        .logoutSuccessUrl("/home?logout=true"))
                .csrf(AbstractHttpConfigurer::disable);
        return httpSecurity.build();
    }
```

Sl. 4.10. *Implementacija ograničenog pristupa*

Unutar implementacije pronalazimo putanju *./login-user* koja vodi korisnika na prijavu u sustav. Kako bi se korisnik mogao prijaviti u sustav treba se prvo registrirati. Registracija se obavlja pomoću *AuthControllera* koji očitava stranicu za registriranje metodom *register()*. Kada se pošalju podatci za registraciju, aktivira se *createUser()* metoda kojom se u bazu sprema korisnik s dodijeljenim podatcima. Zaporka je osjetljivi podatak za svakoga korisnika te zbog toga *Spring Security* koristi *PasswordEncoder* klasu koja pretvara zaporku u *hash* vrijednost i sprema u bazu podataka. Kada bi se korisnik pokušao prijaviti, otvorila bi se prilagođena stranica (engl. *user-friendly*) koja se dohvaća unutar *AuthControllera*. Kada se pošalju podatci za prijavu pokreće se mehanizam za provjeru autentifikacije pomoću *UserDetails* klase koja se nalazi unutar *Spring Securitya*. Unutar nje su se spremili osnovni podatci korisnika poput elektroničke pošte i zaporke koji se kasnije koriste kako bi se provjerila točnost podataka. To je postignuto pomoću *MyUserInfoDetails* koja implementira *UserDetails* i time preuzima sve funkcionalnosti koje ona ima.

Autentifikacija se odvija unutar *authenticationProvider()* gdje se pronalazi već spremljeni korisnik iz baze i kreira pomoću *MyUserDetailService*. Kada je korisnik kreiran uzima se dodijeljena *hash* zaporka prilikom registracije i provjerava se s zaporkom za prijavu. Ako se podudaraju dolazi do prijave korisnika u sustav te se pojavljuje gumb za odjavljivanje. U suprotnom korisnik je vraćen na stranicu za ponovnu prijavu. Na slikama 4.11. do 4.15. prikazana je implementacija već navedenih klasa potrebnih za uspješnu registraciju i prijavu.

```
@RestController
@AllArgsConstructor
public class AuthController {

    UserService userService;

    CorporationService corporationService;
    JakovStakovic
    @GetMapping("register")
    public ModelAndView register(ModelAndView modelAndView){
        UserDto userDto = new UserDto();
        modelAndView.addObject( attributeName: "userForm", userDto);
        modelAndView.setViewName("logging/user-register");
        return modelAndView;
    }

    JakovStakovic
    @PostMapping("save")
    public ModelAndView createUser(@ModelAttribute("userForm") UserDto userDto,
                                   ModelAndView modelAndView){
        UserDto savedDto = userService.createUser(userDto);
        modelAndView.addObject( attributeName: "registeredUser", savedDto);
        modelAndView.setViewName("logging/successRegistration");
        return modelAndView;
    }

    JakovStakovic
    @GetMapping("login-user")
    public ModelAndView login(ModelAndView modelAndView){
        modelAndView.setViewName("logging/user-login");
        return modelAndView;
    }



    JakovStakovic
    @GetMapping("logout")
    public ModelAndView logout(ModelAndView modelAndView){
        modelAndView.setViewName("main/home");
        return modelAndView;
    }
```

Sl. 4.11. „*AuthController*“ s metodama za registraciju i prijavu

```java
public class MyUserInfoDetails implements UserDetails {
    2 usages
    private String email;
    2 usages
    private String password;
    2 usages
    private Collection<? extends GrantedAuthority> authorities;

    1 usage  JakovStakovic
    public MyUserInfoDetails(User user) {...}

    JakovStakovic
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() { return authorities; }

    1 usage  JakovStakovic
    private Collection<? extends GrantedAuthority> convertRolesToAuthorities(Set<Role> roles) {...}

    JakovStakovic
    @Override
    public String getPassword() { return this.password; }

    JakovStakovic
    @Override
    public String getUsername() { return this.email; }

    9 usages  JakovStakovic
    @Override
    public boolean isAccountNonExpired() { return true; }

    9 usages  JakovStakovic
    @Override
    public boolean isAccountNonLocked() { return true; }

    9 usages  JakovStakovic
    @Override
    public boolean isCredentialsNonExpired() { return true; }

    JakovStakovic
    @Override
    public boolean isEnabled() { return true; }
}
```

Sl. 4.12. *„MyUserInfoDetails“ klasa*

```java
@Configuration
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

    5 usages  JakovStakovic *
    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email);
        if(user == null){
            throw new UsernameNotFoundException("User does not exist! Register if you don't have an account.");
        }
        else{
            return new MyUserInfoDetails(user);
        }
    }
}
```

Sl. 4.13. *„MyUserInfoDetailsService“ klasa*

```java
    JakovStakovic
    @Bean
    public UserDetailsService userDetailsService() {
        return new MyUserDetailsService();
    }

    JakovStakovic
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
        daoAuthenticationProvider.setUserDetailsService(userDetailsService());
        daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());
        return daoAuthenticationProvider;
    }

    JakovStakovic
    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
}
```

Sl. 4.14. *„authenticationProvider“ metoda za provjeru prijave korisnika*

```java
@Transactional
@Override
public UserDto createUser(UserDto userDto) {
    User userUsername = userRepository.findByUsername(userDto.getUsername());
    User userEmail = userRepository.findByEmail(userDto.getEmail());
    if (userUsername != null || userEmail != null) {
        if (userUsername != null) {
            throw new ResourceAlreadyExistsException("User with " + userDto.getUsername() + " already exists!")
        }
        if (userEmail != null) {
            throw new ResourceAlreadyExistsException("User with " + userDto.getEmail() + " already exists!");
        }
    }
    Set<Role> roles = new HashSet<>();
    if(userDto.getUsername().contains("Admin") || userDto.getUsername().contains("admin")) {
        roles.add(Role.ADMIN);
        User user = modelMapper.map(userDto, User.class);
        user.setRoles(roles);
        user.setPassword(passwordEncoder.encode(userDto.getPassword()));
        User savedUser = userRepository.save(user);
        return modelMapper.map(savedUser, UserDto.class);
    } else {
        roles.add(Role.USER);
        User user = modelMapper.map(userDto, User.class);
        user.setRoles(roles);
        user.setPassword(passwordEncoder.encode(userDto.getPassword()));
        User savedUser = userRepository.save(user);
        return modelMapper.map(savedUser, UserDto.class);
    }

}
```

Sl. 4.15. *„createUser“ metoda za kreiranje korisnika u bazi*

4.3. Kontroler (engl. *Controller*)

Kako bi se pristupalo različitim stranicama, koristimo kontrolere koji su povezani s različitim modelima. Svaki kontroler ima ime ovisno o njegovom modelu s kojim radi. Unutar njega se pronalaze anotacije *@RestController* i *@AllArgsContructor*. *@RestController* je kombinacija *@Controller* i *@ResponseBody* anotacije te s *@ResponseBody* anotacijom označava kako se rezultati modela moraju serializitari u HTTP odgovore (JSON ili XML najčešće) umjesto da se vraća „View“ prikaz. *@AllArgsConstructor* je anotacija iz biblioteke Lombok. Biblioteka služi za lakše pisanje standardnog koda (getMethod(), setMethod()…) dok sama anotacija omogućuje automatsko generiranje konstruktora koji prima sve članove klase kao argumente. Slika 4.16. i 4.17. prikazuju *TeamController* i njegovu implementaciju kojom je postignuta funkcionalnost za pristup timovima.

```java
@RestController
@AllArgsConstructor
public class TeamController {

    private UserService userService;

    private CorporationService corporationService;

    private TeamService teamService;

    JakovStakovic
    @GetMapping("all-teams")
    @PreAuthorize("hasAuthority('ADMIN') or hasAuthority('LEADER')")
    public ModelAndView allTeams(ModelAndView modelAndView) {
        List<TeamDto> teamsDto = teamService.findAll();
        modelAndView.addObject( attributeName: "teams", teamsDto);
        modelAndView.setViewName("lists/teams");
        return modelAndView;
    }

    JakovStakovic
    @GetMapping("allTeamsInCorporation")
    public ModelAndView allTeamsInCorporation(ModelAndView modelAndView){
        List<TeamDto> teamsDto = teamService.getTeamsBasedOnUserRole();
        modelAndView.addObject( attributeName: "teamsDto", teamsDto);
        modelAndView.setViewName("adminResources/allTeamsInCorporation");
        return modelAndView;
    }

    JakovStakovic
    @GetMapping("createTeam")
    @PreAuthorize("hasAuthority('ADMIN')")
    public ModelAndView createTeam(ModelAndView modelAndView) {
        TeamDto teamDto = new TeamDto();
        List<Corporation> corporations = corporationService.findAll();
        List<UserDto> userDtos = userService.findAll();
        modelAndView.setViewName("adminResources/createTeam");
        modelAndView.addObject( attributeName: "teamForm", teamDto);
        modelAndView.addObject( attributeName: "corporations", corporations);
        modelAndView.addObject( attributeName: "possibleEmployees", userDtos);
        return modelAndView;
    }
```

Sl. 4.16. *Team kontroler 1.dio*

```
@PostMapping("addUsersToTeam")
@PreAuthorize("hasAuthority('ADMIN') or hasAuthority('LEADER')")
public ModelAndView addUsersToTeam(@RequestParam(value = "employeeIds", required = false) List<Long> ids,
                                   @RequestParam(value = "teamId") Long teamId,
                                   ModelAndView modelAndView){
    teamService.addUsersToTeam(teamId, ids);
    modelAndView.setViewName("redirect:createForm");
    modelAndView.addObject( attributeName: "messageGreen",  attributeValue: true);
    return modelAndView;
}
 JakovStakovic *
@GetMapping("edit/{id}")
@PreAuthorize("hasAuthority('ADMIN') or hasAuthority('LEADER')")
public ModelAndView editTeam(@PathVariable Long id,
                             ModelAndView modelAndView){
    TeamDto teamDto = teamService.getTeamById(id);
    modelAndView.addObject( attributeName: "teamDto", teamDto);
    modelAndView.addObject( attributeName: "corporations", corporationService.findAll());
    modelAndView.addObject( attributeName: "possibleEmployees",
            userService.getUsersForTeam(teamDto.getId()));
    modelAndView.setViewName("adminResources/editTeam");
    return modelAndView;
}

 JakovStakovic
@PostMapping("updateTeam/{id}")
@PreAuthorize("hasAuthority('ADMIN') or hasAuthority('LEADER')")
public ModelAndView updateTeam(@ModelAttribute TeamDto teamDto,
                               @PathVariable Long id,
                               ModelAndView modelAndView){
    teamService.updateTeam(teamDto,id);
    modelAndView.addObject( attributeName: "messageTeamGreen",  attributeValue: true);
    modelAndView.setViewName("redirect:../workingSpace");
    return modelAndView;
}

 JakovStakovic
@PostMapping("delete/{id}")
@PreAuthorize("hasAuthority('ADMIN') or hasAuthority('LEADER')")
public ModelAndView deleteTeam(@PathVariable Long id,
                               ModelAndView modelAndView){
    teamService.deleteTeam(id);
    modelAndView.addObject( attributeName: "teamDelete",  attributeValue: true);
    modelAndView.setViewName("redirect:../all-teams");
    return modelAndView;
}
```

Sl. 4.17. *Team kontroler 2.dio*

Kako su svi kontroleri slični prikazanje su njihove tablice (4.1. do 4.7.) u kojima će biti opisane sve funkcionalnosti pojedinih kontrolera.

Tab. 4.1. *AuthController*

Metoda	Putanja	Funkcionalnost
register	/register	registracija
createUser	/save	spremanje
login	/login-user	provjera prijave
logout	/logout	odjavljivanje
profile	/profile	prikaz

Tab. 4.2. *CorporationController*

Metoda	Putanja	Funkcionalnost
createCorporation	/createCorporation	kreiranje
saveCorporation	/saveCorporation	spremanje
allCorporations	/all-corporations	prikaz
deleteCorporation	/deleteCorporation	brisanje
chooseCorporationForTeam	/chooseCorporationForTeam	uređivanje
chooseTeamsForCorporation	/chooseTeamsForCorporation	uređivanje
saveTeamsForCorporation	/saveTeamsForCorporation	ažuriranje

Tab. 4.3. *PageController*

Metoda	Putanja	Funkcionalnost
home	/home	prikaz
features	/features	prikaz
about	/about	prikaz
teams	/teams	prikaz
workingSpace	/workingSpace	prikaz
teamGroup	/teamGroup	prikaz

Tab. 4.4. *PostController*

Metoda	Putanja	Funkcionalnost
createPost	/createPost/{id}	prikaz
savePost	/savePost/{id}	spremanje
deletePost	/deletePost/{id}	brisanje
editPost	/editPost/{id}	prikaz
updatePost	/updatePost/{id}	ažuriranje

Tab. 4.5. *ProjectController*

Metoda	Putanja	Funkcionalnost
createProject	/create-project	prikaz
saveProject	/saveProject	spremanje
allProjects	/all-projects	prikaz
deleteProject	/deleteProject/{id}	brisanje
viewProject	/viewProject/{id}	prikaz
editProject	/editProject/{id}	prikaz
updateProject	/updateProject/{id}	ažuriranje
Projects	/projects	prikaz

Tab. 4.6. *UserController*

Metoda	Putanja	Funkcionalnost
getAllUsers	/all-users	Prikaz
deleteByUserId	/{userId}	brisanje

Tab. 4.7. *TeamController*

Metoda	Putanja	Funkcionalnost
allTeams	/all-teams	prikaz
allTeamsInCorporation	/allTeamsInCorporation	Prikaz
createTeam	/createTeam	Prikaz
saveTeam	/saveTeam	Spremanje
createForm	/createForm	Prikaz
updateForm	/updateForm	prikaz
Update2ndForm	/update2ndForm	prikaz
addUsersToTeam	/addUsersToTeam	Ažuriranje
editTeam	/editTeam/{id}	prikaz
deleteTeam	/delete/{id}	brisanje

4.4. Data Transfer Object (DTO)

Zaštita podataka velika je odgovornost unutar aplikacija. Kako se ne bi izložili svi podatci korisnika, timova, organizacija i drugih, koristi se *Data Transfer Object* (DTO). DTO je obrazac u softverskom dizajnu koji se koristi za ograničeni prijenos podataka između različitih dijelova sustava. Njime se smanjuje broj poziva potrebnih za dohvaćanje ili slanje podataka te osigurava kontrolu nad podatcima koji se izlažu vanjskom svijetu, sprječavajući potencijalno osjetljive informacije da budu nehotice otkrivene. Za primjer će biti prikazan jedan DTO. Na taj isti način su napravljeni i svi ostali. Slika 4.18. prikazuje *PostDto* klasu i njezinu implementaciju.

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class PostDto {
    private Long id;
    private String content;
    private String title;
    private Timestamp timestamp;
    private UserDto user;
    private ProjectDto project;
}
```

Sl. 4.18. *„PostDto“ klasa*

4.5. Servisi i implementacija servisa

Servisi predstavljaju ključni dio arhitekture aplikacije. Oni služe kao most između kontrolera i poslovne logike te upravljaju interakcijom između različitih slojeva aplikacije. Ovdje su smještene sve operacije koje obavljaju određene poslovne procese kao što su transformacija podataka, obrada zahtjeva i sl.. Važno je da poslovna logika bude izolirana od ostalih slojeva kako bi bila lakša za održavanje i promjenu. Unutar aplikacije svaki model ima svoj servis. Sve funkcionalnosti koje se događaju na jednom modelu nalaze se u posebnom servisu. Servisi se sastoje od sučelja i konkretne implementacije njihovih metoda. Svi servisi su napravljeni na sličan način te će za primjer biti objašnjen samo jedan. Slika 4.19. prikazuje *UserService* sučelje i deklarirane metode.

```
1 implementation  JakovStakovic *
public interface UserService {
    1 usage  1 implementation  JakovStakovic
    UserDto createUser(UserDto userDto);

    2 usages  1 implementation  JakovStakovic
    UserDto findByUserId(Long id);
    8 usages  1 implementation  JakovStakovic
    UserDto findByEmail(String email);

    1 implementation  JakovStakovic
    List<UserDto> findAll();

    8 usages  1 implementation  JakovStakovic
    UserDetails getLoggedInUserDetails();

    1 usage  1 implementation  JakovStakovic
    void updateUser(Long corporationId, Long userId);

    1 usage  1 implementation  JakovStakovic
    void updateUser(User user);

    1 usage  1 implementation  JakovStakovic
    List<UserDto> findUsersWithoutLeaderId();

    2 usages  1 implementation  JakovStakovic
    List<UserDto> getUsersForTeam(Long id);

    1 usage  1 implementation  JakovStakovic
    List<User> getUsersByIds(List<Long> employeeIds);

    1 usage  1 implementation  JakovStakovic
    void removeEmployeeFromTeamAndDelete(Long id);

    3 usages  1 implementation  JakovStakovic
    UserDto toUserDto(User user);
```

Sl. 4.19. *„UserService“ servis*

Unutar *UserService* sučelja deklarirane su metode koje rade s *UserDto* modelom. Razlog za kreiranjem sučelja pa tek onda konkretne implementacije omogućuje moderan razvoj softvera poput: apstrakcije, slabe povezanosti (engl. *loose-coupling*), obrazaca i jednostavnosti promjene. Različiti dijelovi koda ne trebaju poznavati detalje implementacije te i dalje mogu komunicirati preko sučelja. To olakšava zamjenu implementacije bez potrebe za promjenama u ostatku koda. Za primjer implementacije metoda uzeti ćemo par njih na kojima ćemo objasniti razmišljanje kreiranja samih metoda unutar cijele aplikacije. Slika 4.20. prikazuje tri definirane metode unutar *UserServiceImpl* klase u kojoj se nalazi konkretna implementacija deklariranih metoda *UserServicea.*

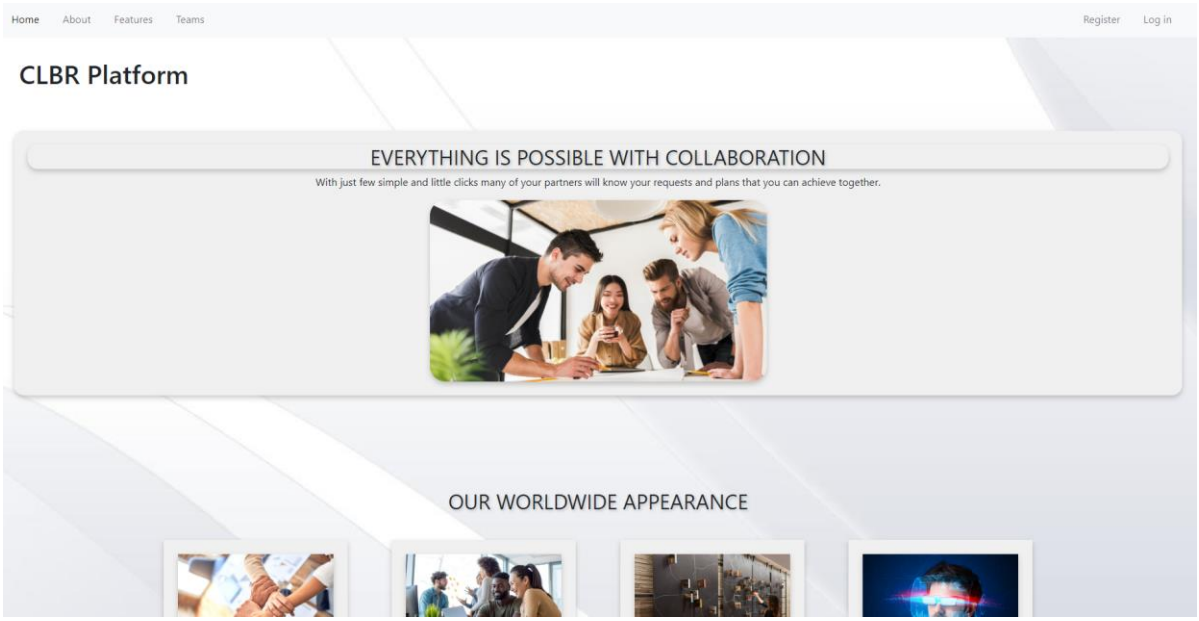
```
@Override
public UserDto findByEmail(String email) {
    User foundUser = userRepository.findByEmail(email);
    if (foundUser == null) {
        throw new ResourceNotFoundException("User doesn't exist!");
    }
    return modelMapper.map(foundUser, UserDto.class);
}

1 usage  JakovStakovic
@Override
public List<UserDto> findUsersWithoutLeaderId() {
    List<User> users = userRepository.findByLeaderIsNull();
    List<UserDto> userDtos = users.stream() Stream<User>
            .map(this::toUserDto) Stream<UserDto>
            .collect(Collectors.toList());
    return userDtos;
}


2 usages  JakovStakovic
@Override
public List<UserDto> getUsersForTeam(Long id) {

    List<User> users = userRepository.findUsersNotInTeam(id);

    List<UserDto> userDtos = users.stream() Stream<User>
            .map(this::toUserDto) Stream<UserDto>
            .collect(Collectors.toList());
    return userDtos;
}
```

Sl. 4.20. *Metode „UserServiceImpl“ klase*

Krenuti ćemo s prvom metodom *findByEmail(String email).* Unutar aplikacije u jednom dijelu želimo pronaći korisnika temeljenog na njegovoj elektroničkoj adresi. Kada pronađemo korisnika ne dajemo sve podatke koje su postavljene unutar *User* modela već koristimo *modelMapper* klasu koja postavlja vrijednosti iz *User* modela unutar *UserDto* koji će nam biti izložen vanjskom sučelju. Time ograničavamo izlaganje zaporke i drugih osjetljivih informacija. Druga metoda, *findUsersWithoutLeaderId(),* dohvaća sve korisnike koji nemaju voditelja. Koristi se *userRepository* za pronalaženje korisnika kojima je *leaderId* jednak *null* vrijednosti, a zatim se rezultirajuća lista korisnika transformira u listu *UserDto* objekata koristeći Java Stream Application Programming Interface. Treća metoda, *getUsersForTeam(Long id),* dohvaća korisnike koji nisu članovi određenog tima, također koristeći Stream Application Programming Interface za mapiranje rezultata u DTO objekte. Slike i opis ovih metoda su primjer kako se kroz cijelu aplikaciju kreiraju metode koje komuniciraju s bazom podataka. Svaka metoda izolira interakciju s bazom podataka koristeći repozitorije i koristi DTO objekte za prijenos podataka između slojeva aplikacije. To omogućuje čistoću koda, olakšava održavanje, te omogućuje jasnu separaciju poslovne logike od logike pristupa podatcima.

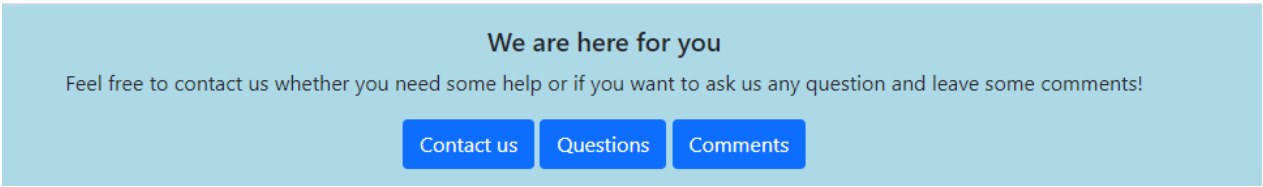
5. PRIKAZ RADA WEB APLIKACIJE

Prikaz rada web aplikacije (GUI) ključan je za razumijevanje njenog funkcioniranja i korisničkog iskustva. Kroz ovaj dio, istražit ćemo glavne aspekte korisničkog sučelja web aplikacije, demonstrirajući kako intuitivni dizajn i funkcionalnost mogu unaprijediti interakciju korisnika s aplikacijom te olakšati postizanje njihovih ciljeva.



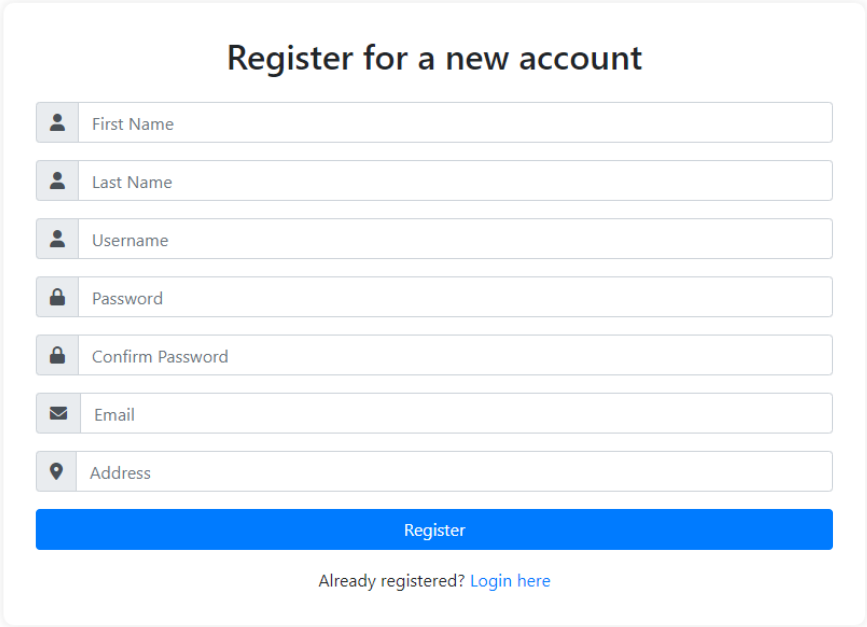
Sl. 5.1. *Naslovna stranica*

Prema slici 5.1. prva stranica koja se otvori prilikom pristupa Collaboration Platform aplikaciji je naslovna stranica. Na vrhu stranice nalazi se navigacijska lista koja ima ostale stranice koje su hard kodirane kojima svi imaju pristup. S desne strane može se vidjeti registracija i prijava s kojima se pristupa drugim dijelovima aplikacije. Kada se korisnik prijavi, to postaje gumb za odjavu i provjeru svojega profila. Slika 5.2. prikazuje podnožje stranice gdje se nalaze tipke kojima korisnici imaju pristup kontaktiranja voditelja aplikacije.



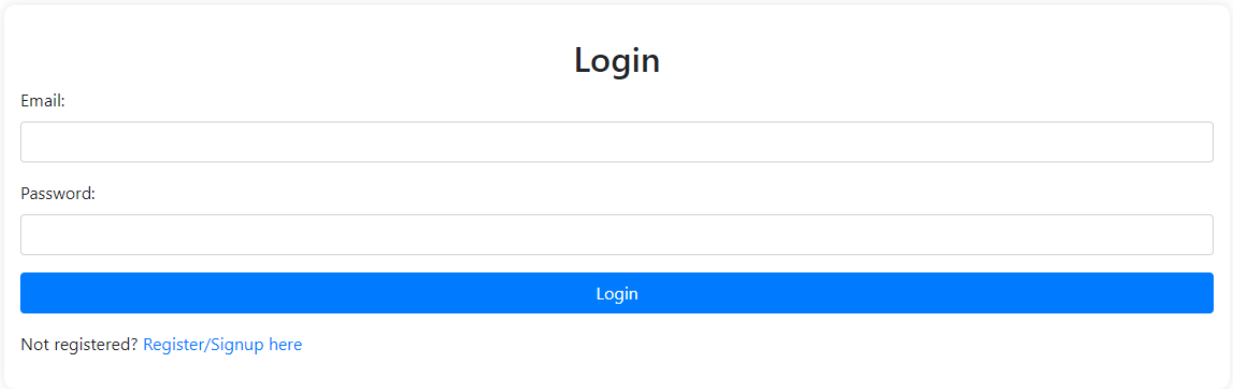
Sl. 5.2. *Podnožje stranice*

Prilikom pritiska na registraciju, otvara se korisnički prilagođeno sučelje prikazano slikom 5.3. koje služi za prijavu korisnika u sustav. Potrebni su osobni podatci poput imena, prezimena, korisničkog imena, zaporke, elektroničke pošte i adrese stanovanja.



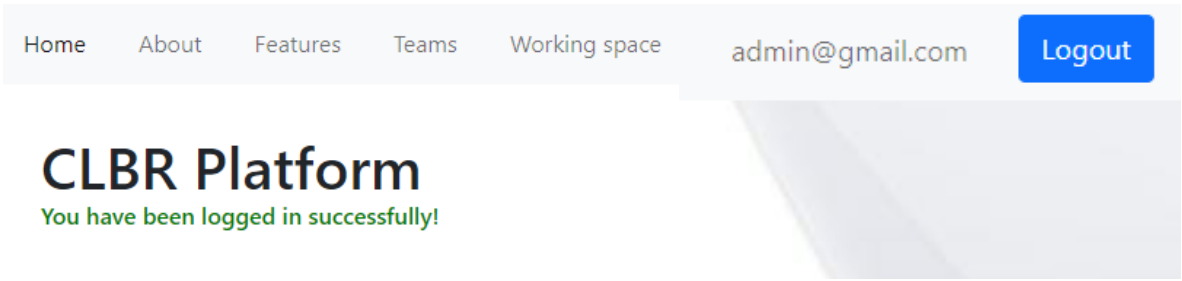
Sl. 5.3. *Forma za registraciju*

Klikom na prijavu, otvara se korisnički prilagođeno sučelje prikazano slikom . 5.4. u kojem se korisnik prijavljuje s elektroničkom poštom i zaporkom.



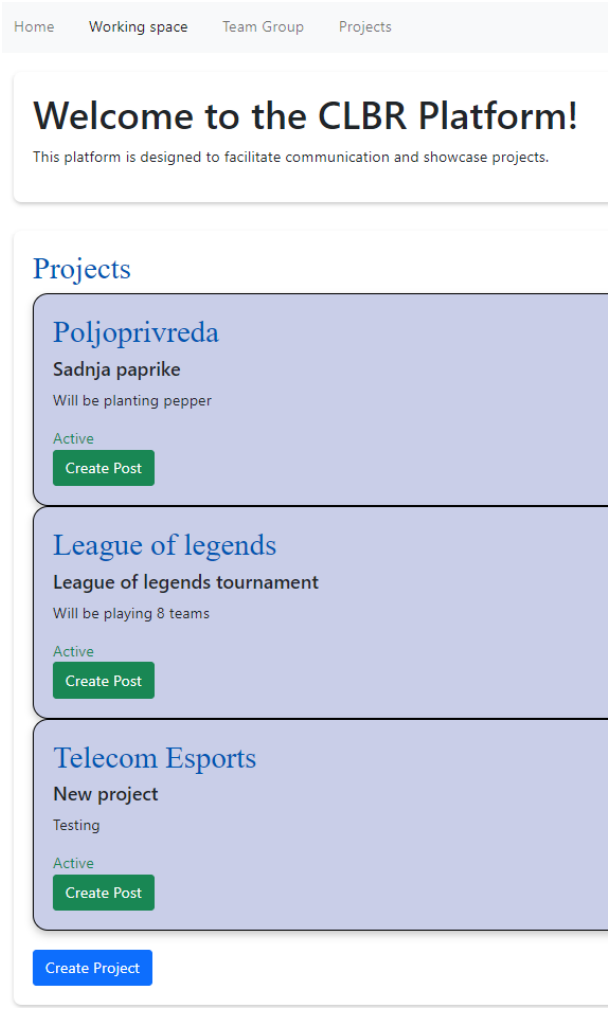
Sl. 5.4. *Forma za prijavu*

Prijavom u sustav korisniku se otvara nova tipka na navigacijskoj traci *Working Space* koja omogućuje pristup radnome dijelu te potvrda s porukom o uspješnoj prijavi u aplikaciju. Slika 5.5. prikazuje novu navigacijsku traku s dodatnim tipkama.

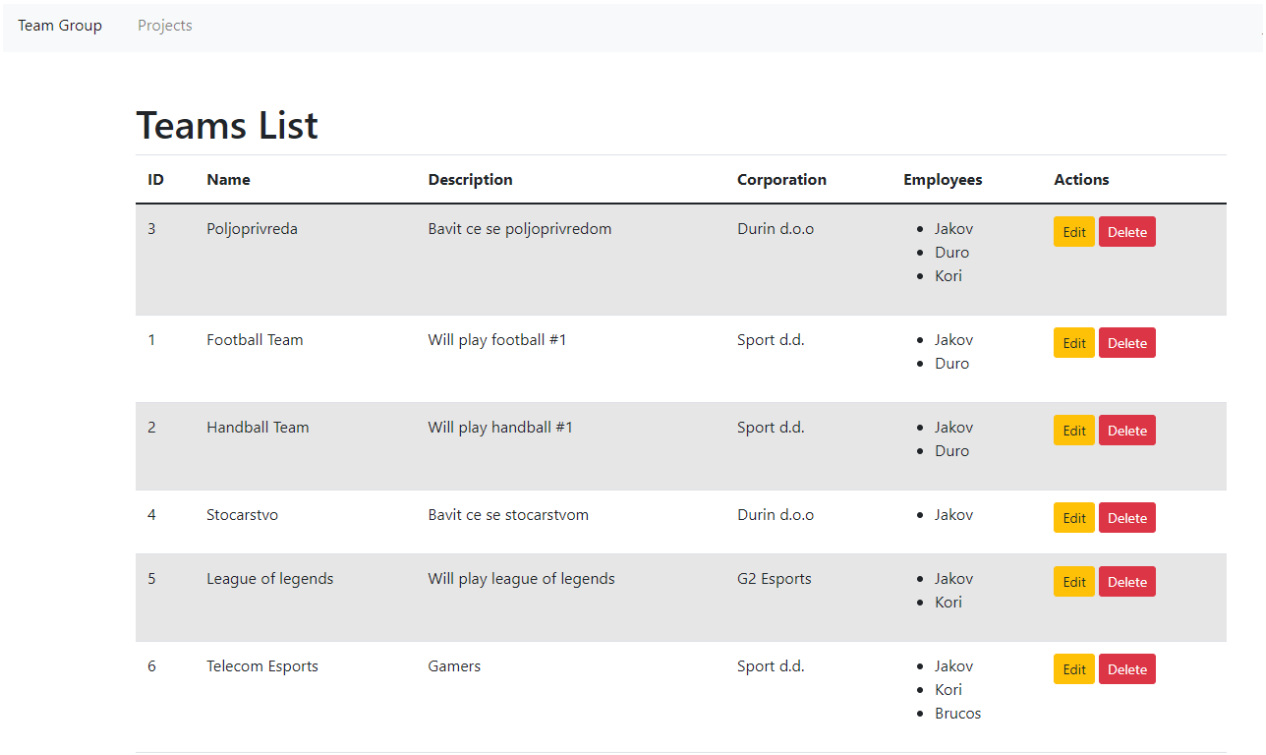


Sl. 5.5. *Working Space, profil i odjavljivanje*

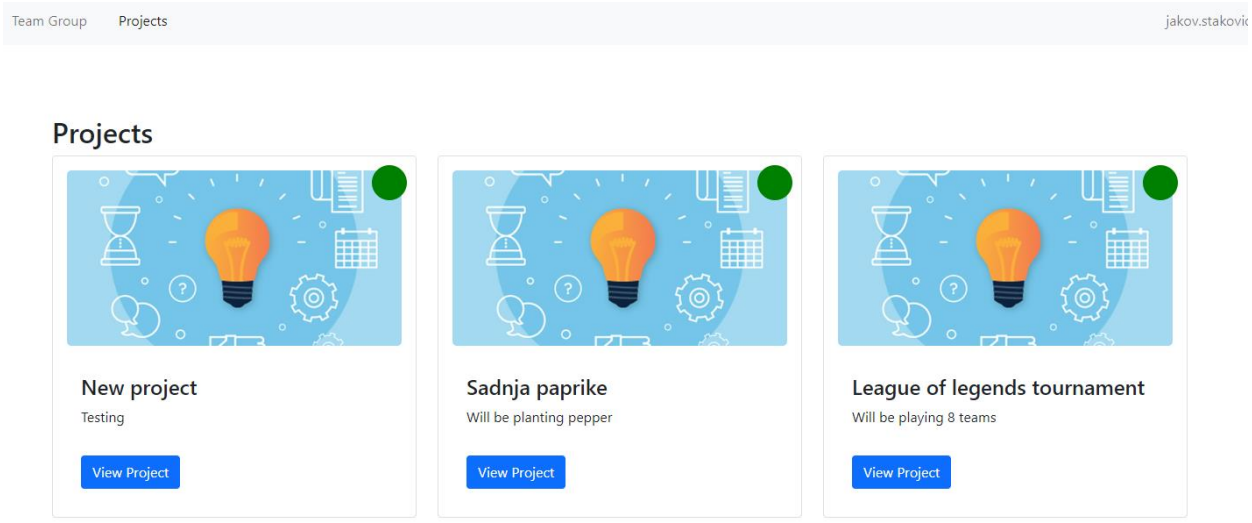
Na slici 5.6. prikazan je *Working Space*. Prostor unutar kojeg se nalaze projekti za određeni tim u kojem se nalazi pojedini korisnici. Svaki projekt ima ime tima, ime projekta, opis što se radi na tom projektu i je li aktivan. Omogućeno je kreiranje komentara za svakog prijavljenog korisnika. Na navigacijskoj traci se otvaraju nove tipke koje vode na timove koji prikazuju sve timove unutar kojih se korisnik nalazi (slika 5.7.) i tipku projekti koja prikazuje sve projekte unutar kojih se korisnik nalazi (slika 5.8.).



Sl. 5.6. *Working Space stranica*

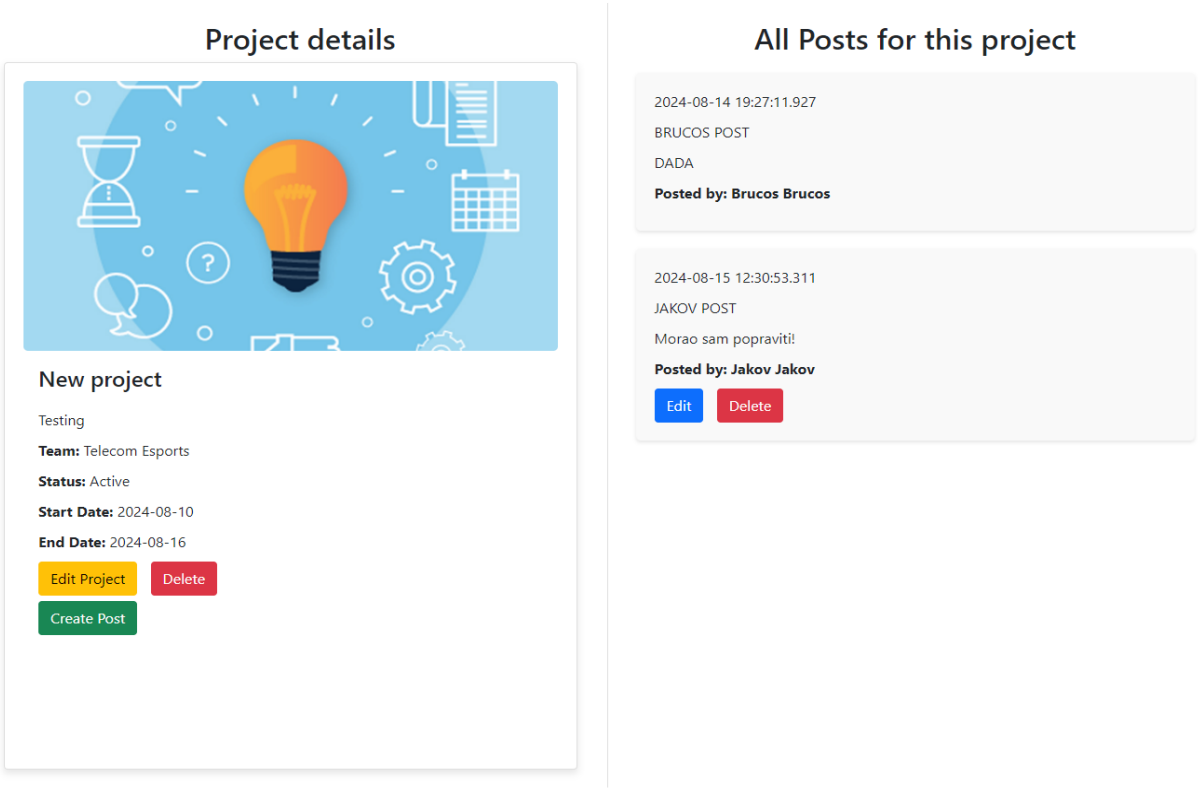


Sl. 5.7. *Team Group stranica*

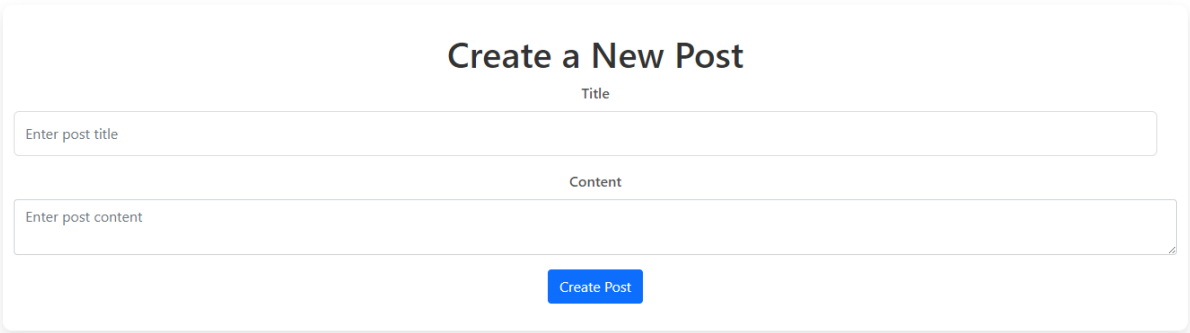


Sl. 5.8. *Projekti stranica*

Kada korisnik pritisne tipku *View Project*, otvara se postojeći projekt s mogućnostima uređivanja (slika 5.9.). Ako je korisnik postavljen kao leader organizacije, ima dodatna prava unutar aplikacije. Kao leader, korisnik može upravljati svim projektima unutar organizacije, što uključuje pregledavanje svih projekata, uređivanje postojećih, dodavanje novih zadataka ili promjena, te brisanje projekata, što omogućuje potpuno uklanjanje projekata iz sustava zajedno sa svim povezanim podatcima. Također, korisnik može dodavati komentare na bilo koji projekt ili zadatak unutar projekta, čime se omogućuje bolja komunikacija i suradnja unutar tima (slika 5.10.).

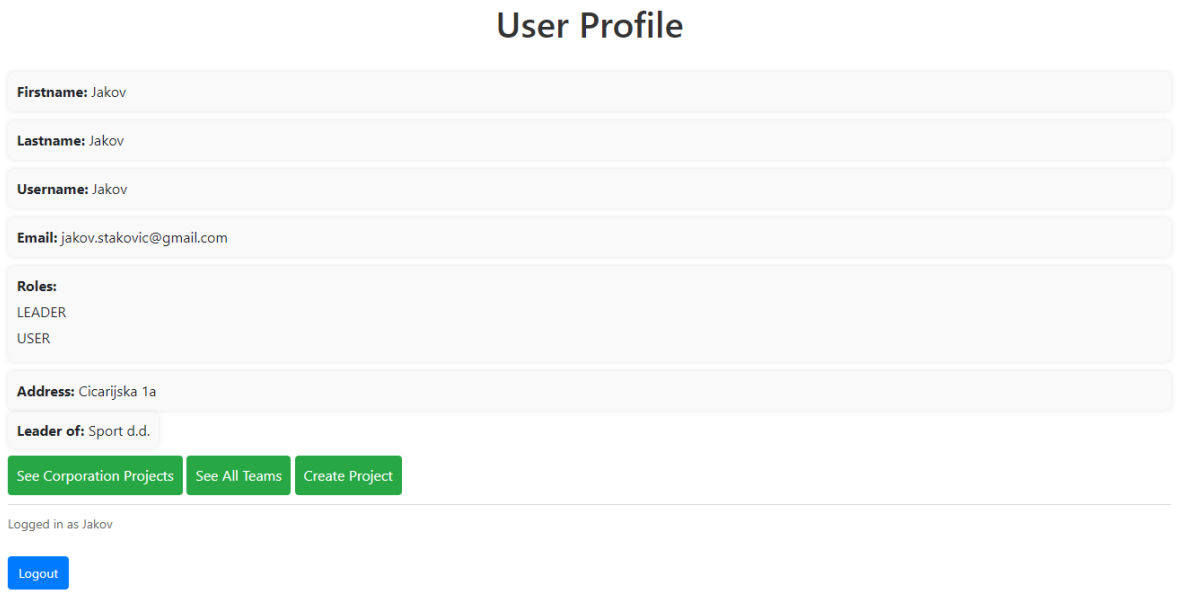


Sl. 5.9. *„View project“*



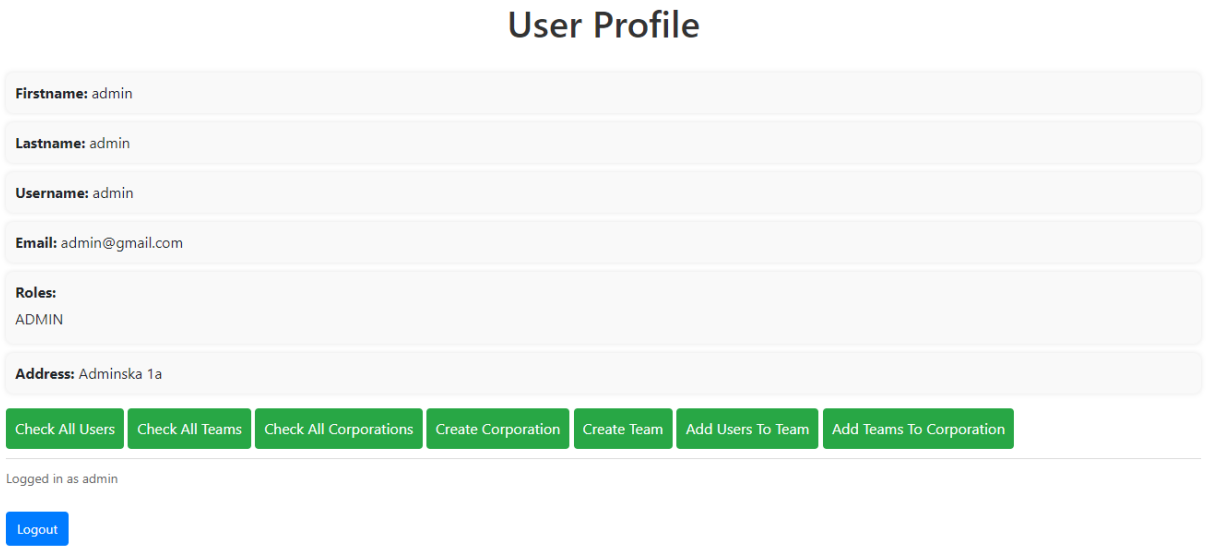
Sl. 5.10. *Kreiranje komentara*

Kako bi korisnik mogao vidjeti svoje podatke koristimo tipku koja se nalazi pored tipke za odjavljivanje na kojoj je prikazana elektronička pošta prijavljenog korisnika. Slika 5.11. i 5.12. prikazuju profil korisnika s ključnim informacijama o korisniku i omogućuje pristup raznim funkcijama unutar aplikacije (ovisno o pravima korisnika). Profil sadrži gumbe koji omogućuju



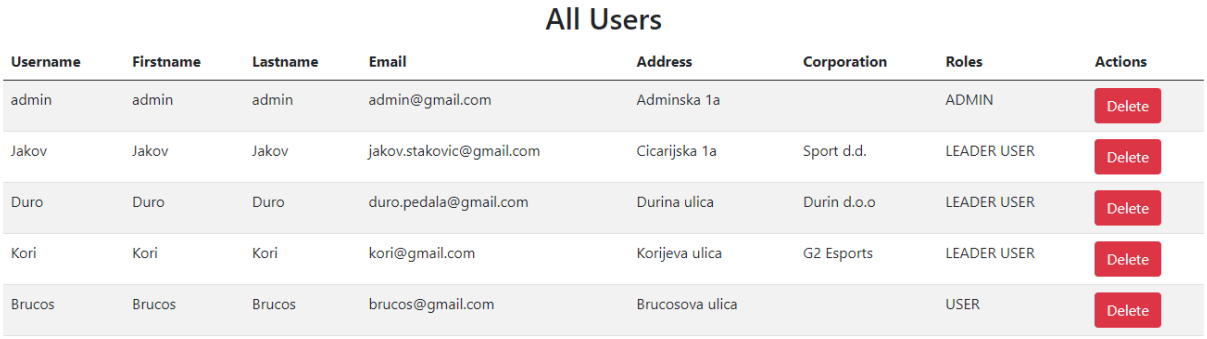
Sl. 5.11. *Prikaz podataka korisnika (leader prava)*

brzo kretanje do različitih dijelova aplikacije. Klikom na gumb "See All Teams" korisnik može pregledati sve timove unutar kojih se nalazi, dok gumb "See Corporation Projects" omogućuje pregled svih projekata unutar organizacije. Gumb "Create Project" omogućuje korisniku da pokrene proces stvaranja novog projekta unutar organizacije.



Sl. 5.12. *Prikaz podataka korisnika (admin prava)*

Profil korisnika s admin pravima pruža sveobuhvatan pregled i kontrolu nad raznim elementima unutar aplikacije. Na profilu se mogu vidjeti podatci o korisniku, kao i gumbi koji omogućuju administriranje različitih aspekata sustava. Admin može pregledavati sve korisnike unutar aplikacije (slika 5.13.), s opcijom brisanja korisnika ako je potrebno. Također, može provjeravati i upravljati svim timovima, što uključuje mogućnost uređivanja i brisanja timova. Admin ima pristup i svim organizacijama te može brisati organizacije po potrebi (slika 5.14.). Uz to, ima mogućnost kreiranja novih organizacija i timova (slika 5.15. i 5.16.), dodavanja korisnika u timove (slika 5.17.), kao i dodavanja timova u organizacije (slika 5.18.). Ove funkcionalnosti omogućuju adminu potpunu kontrolu nad upravljanjem korisnicima, timovima, i organizacijama unutar aplikacije.



All Users

Username	Firstname	Lastname	Email	Address	Corporation	Roles	Actions
admin	admin	admin	admin@gmail.com	Adminska 1a		ADMIN	Delete
Jakov	Jakov	Jakov	jakov.stakovic@gmail.com	Cicarijska 1a	Sport d.d.	LEADER USER	Delete
Duro	Duro	Duro	duro.pedala@gmail.com	Durina ulica	Durin d.o.o	LEADER USER	Delete
Kori	Kori	Kori	kori@gmail.com	Korijeva ulica	G2 Esports	LEADER USER	Delete
Brucos	Brucos	Brucos	brucos@gmail.com	Brucosova ulica		USER	Delete

Sl. 5.13. *Prikaz svih korisnika*

Corporations and Their Teams

Sport d.d.

ID	Leader of Corporation	Corporation	Description	Team Name	Employees
6	Jakov	Sport d.d.	Gamers	Telecom Esports	Jakov Kori Brucos
2	Jakov	Sport d.d.	Will play handball #1	Handball Team	Jakov Duro
1	Jakov	Sport d.d.	Will play football #1	Football Team	Jakov Duro

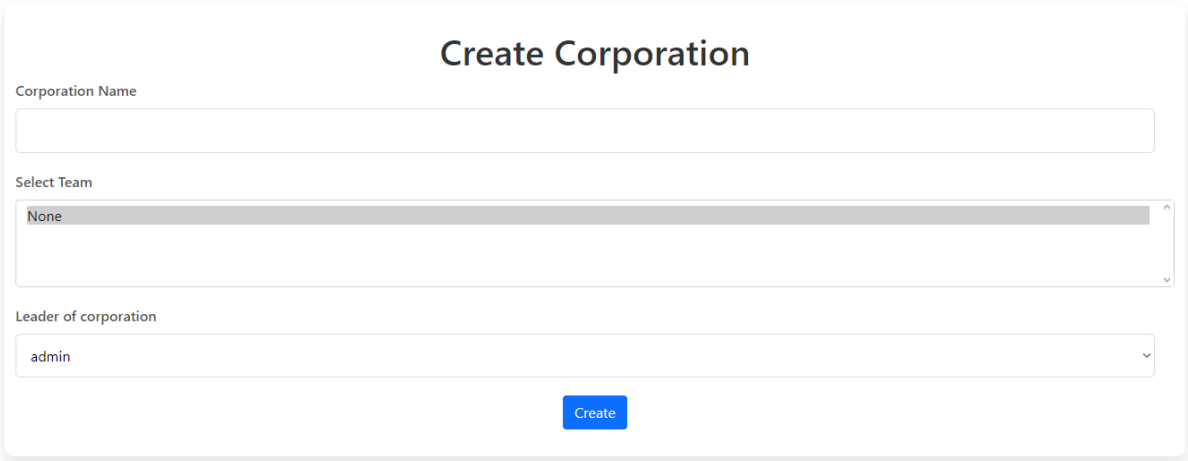
Delete

Durin d.o.o

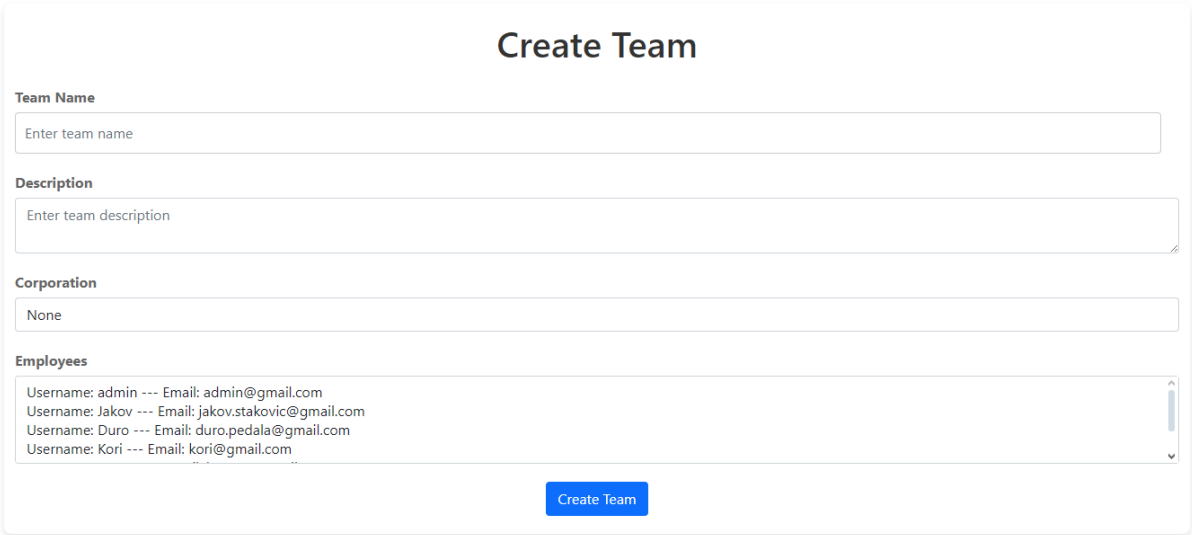
ID	Leader of Corporation	Corporation	Description	Team Name	Employees
3	Duro	Durin d.o.o	Bavit ce se poljoprivredom	Poljoprivreda	Jakov Duro Kori
4	Duro	Durin d.o.o	Bavit ce se stocarstvom	Stocarstvo	Jakov

Delete

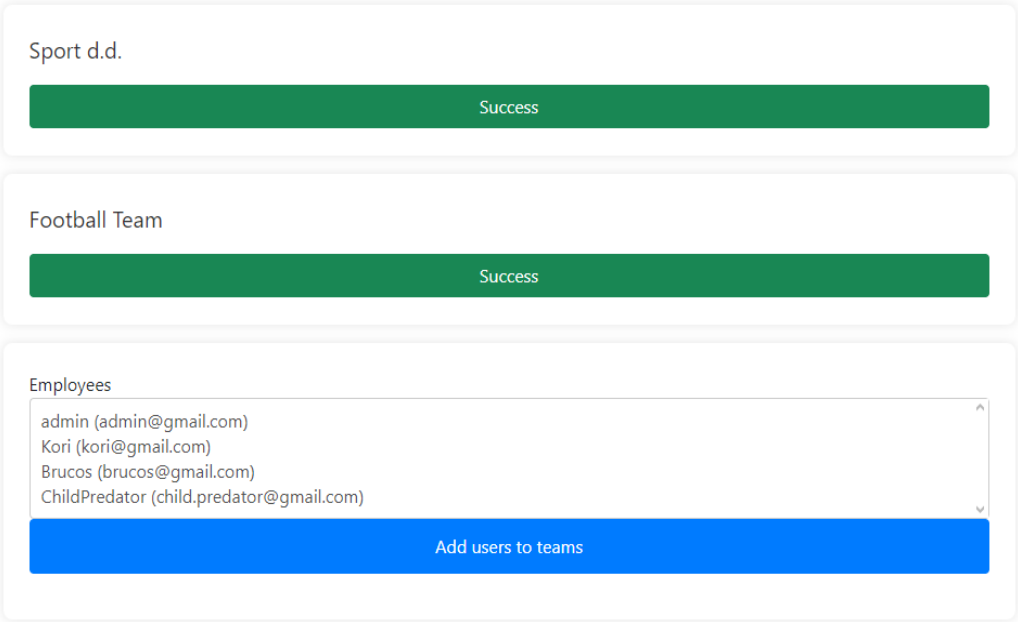
Sl. 5.14. *Prikaz svih organizacija*



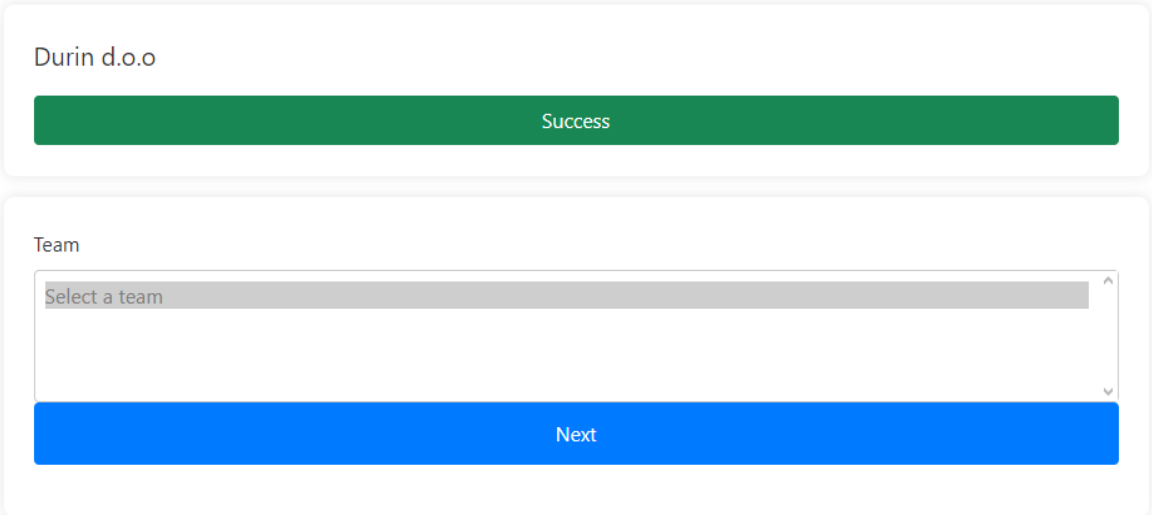
Sl. 5.15. *Forma za kreiranje organizacije*



Sl. 5.16. *Forma za kreiranje tima*



Sl. 5.17. *Forma za dodavanje korisnika u organizaciju*



Sl. 5.18. *Forma za dodavanje timova u organizaciju*

6. ZAKLJUČAK

Kroz ovaj završni rad prikazani su svi ključni aspekti razvoja web aplikacije, od tehničkih elemenata do korisničkog sučelja i upravljanja podatcima. Razvoj aplikacije naglasio je važnost intuitivnog dizajna koji poboljšava korisničko iskustvo, dok istovremeno osigurava sigurno i učinkovito upravljanje podatcima. Korisničko sučelje aplikacije je dizajnirano tako da je lako razumljivo i jednostavno za korištenje. Implementacija naprednih administrativnih funkcionalnosti, kao što su upravljanje korisnicima, timovima i organizacijama, omogućila je centraliziranu kontrolu nad poslovnim procesima unutar aplikacije, čime je poboljšana fleksibilnost i suradnja unutar organizacija. Tijekom razvoja pojavile su se određene prepreke, posebno u vezi s dodavanjem novih korisnika unutar organizacija i omogućavanjem višestrukih komentara na projekte. Prva prepreka bila je povezivanje korisnika s organizacijama u bazi podataka, što je zahtijevalo precizno upravljanje relacijama i osiguravanje referencijalnog integriteta kako bi se spriječile pogreške. Druga prepreka odnosila se na omogućavanje korisnicima postavljanje više komentara na projekte, što je zahtijevalo pažljiv dizajn baze podataka kako bi se podržalo višestruko povezivanje korisnika i projekata, uz očuvanje konteksta i povijesti svakog komentara. Rješenje ovog rada pokazuju da dobro osmišljena web aplikacija može značajno unaprijediti produktivnost i suradnju unutar organizacija, pružajući korisnicima alate potrebne za učinkovito obavljanje njihovih zadataka.

LITERATURA

[1] „Microsoft Teams“, dostupno na: https://www.microsoft.com/hr-hr/microsoft-teams/group-chat-software/, pristup 30.6.2024.

[2] „Zoom“, dostupno na: https://www.zoom.com/en/products/virtual-meetings/, pristup 30.6.2024.

[3] „Slack“, dostupno na: https://slack.com, pristup 30.6.2024.

[4] „Jira“, dostupno na: https://www.atlassian.com/software/jira, pristup 30.6.2024.

[5] „Monday.com“, dostupno na: https://monday.com, pristup 30.6.2024.

[6] „Model View Controller“ dostupno na: https://upload.wikimedia.org/wikipedia/commons/thumb/b/b5/ModelViewControllerDiagram2.svg/525px-ModelViewControllerDiagram2.svg.png, pristup 3.7.2024.

[7] „WEB APPLICATION DEVELOPMENT“, dostupno na: https://baou.edu.in/assets/pdf/PGDCA-202_slm.pdf, pristup 1.7.2024.

[8] „Get started with Bootstrap“, dostupno na: https://getbootstrap.com/docs/5.2/getting-started/introduction/, pristup 15.7.2024.

[9] „What is an application architecture?“, dostupno na: https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture, pristup 28.6.2024.

[10] „Spring 6 & Spring Boot 3 for Beginners“, dostupno na: https://www.udemy.com/course/learn-spring-boot/learn/lecture/33872902?start=555#overview, pristup 5.6.2024.

[11] „10 common Software Arhitectural Patterns“, dostupno na: https://flatrocktech.com/blog/architectural-patterns, pristup 28.6.2024.

[12] „Stack Overflow“, dostupno na: https://stackoverflow.com, pristup 16.7.2024.

[13] „MVC arhitecture“, dostupno na: https://developer.mozilla.org/en-US/docs/Glossary/MVC, pristup 3.7.2024.

SAŽETAK

Ovaj završni rad prikazuje web aplikaciju koja omogućuje upravljanje korisnicima, timovima i projektima unutar organizacije. Rad se usmjerava na primjenu Jave i Spring frameworka kako bi se postiglo robustno rješenje. Java je korištena za razvoj logike aplikacije, dok je Spring framework omogućio implementaciju ključnih funkcionalnosti kao što su autentifikacija korisnika i upravljanje sesijama. Korištenje Spring Boota omogućilo je brzu konfiguraciju i pokretanje aplikacije, a Spring Data pružilo je podršku za rad s podatcima putem ORM (engl. *Object-Relational Mapping*) tehnologija. Ove tehnologije su osigurale efikasno upravljanje korisnicima, timovima i projektima, kao i dodavanje komentara na projekte. Postignuto rješenje uključuju uspješnu implementaciju aplikacije koja omogućuje administrativnim korisnicima da upravljaju svim aspektima organizacije, kao i da dodaju komentare na projekte. Upotreba suvremenih tehnologija omogućen je razvoj skalabilnog i učinkovitog rješenja koje ispunjava sve zahtjeve administrativnih funkcionalnosti.

Ključne riječi: administracija, Java, organizacije, Spring, timovi

ABSTRACT

Web Application for Teamwork Organization

This thesis presents a web application that enables the management of users, teams, and projects within an organization. The focus is on the application of Java and the Spring framework to achieve a robust solution. Java is used for developing the application's logic, while the Spring framework provides the implementation of key functionalities such as user authentication and session management. The use of Spring Boot facilitates rapid configuration and deployment of the application, and Spring Data supports data handling through ORM (Object-Relational Mapping) technologies. These technologies ensure efficient management of users, teams, and projects, as well as the ability to add comments to projects. The achieved solution includes a successful implementation of an application that allows administrative users to manage all aspects of the organization and add comments to projects. The use of modern technologies enables the development of a scalable and efficient solution that meets all administrative functionality requirements.

Keywords: administration, Java, organizations, Spring, teams

ŽIVOTOPIS

Jakov Staković rođen je 7. rujna 2002. godine u gradu Rotterdamu u Nizozemskoj. Osnovno školovanje započeo je u osnovnoj školi Sesvetska Sopnica, a zatim ga nastavlja u srednjoj školi Treća gimnazija u Zagrebu. Završava srednju školu 2021. te upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer preddiplomski sveučilišni studij Računarstvo, Programsko inženjerstvo.