

Programsko inženjerstvo

Ak. god. 2023./2024.

Gradska šteta

Dokumentacija, Rev. 2

Grupa: *VelicanstveniTimRaketa*

Koordinator: *Davor Najev*

Datum predaje: *19.1.2024.*

Nastavnik: *Izv. prof. dr. sc. Vlado Sruk*

Sadržaj

1	Dnevnik promjena dokumentacije	3
2	Opis projektnog zadatka	5
3	Specifikacija programske potpore	9
3.1	Funkcionalni zahtjevi	9
3.1.1	Obrasci uporabe	11
3.1.2	Sekvencijski dijagrami	22
3.2	Ostali zahtjevi	27
4	Arhitektura i dizajn sustava	28
4.1	Baza podataka	30
4.1.1	Opis tablica	30
4.1.2	Dijagram baze podataka	34
4.2	Dijagram razreda	35
4.3	Dijagram stanja	37
4.4	Dijagram aktivnosti	38
4.5	Dijagram komponenti	39
5	Implementacija i korisničko sučelje	40
5.1	Korištene tehnologije i alati	40
5.2	Ispitivanje programskog rješenja	42
5.2.1	Ispitivanje komponenti	42
5.2.2	Ispitivanje sustava	47
5.3	Dijagram razmještaja	49
5.4	Upute za puštanje u pogon	50
5.4.1	Konfiguracija frontenda	50
5.4.2	Konfiguracija backenda	50
5.4.3	Konfiguracija baze podataka	51
6	Zaključak i budući rad	53

Popis literature	55
Indeks slika i dijagrama	57
Dodatak: Prikaz aktivnosti grupe	58

1. Dnevnik promjena dokumentacije

Kontinuirano osvježavanje

Rev.	Opis promjene/dodatka	Autori	Datum
0.1	Napravljen predložak.	Ivan Šimunić	26.10.2023.
0.2	Dodan opis projektnog zadatka.	Nino Ćurko	4.11.2023.
0.3	Dodani aktori i njihovi funkcijski zahtjevi	Nino Ćurko	7.11.2023.
0.3	Dodan dio obrazaca uporabe	Nino Ćurko	7.11.2023.
0.3.1	Dodani opisi obrazaca uporabe i ostali zahtjevi	Nino Ćurko	9.11.2023.
0.4	Dodani dijagrami obrazaca uporabe	Nino Ćurko	14.11.2023.
0.4.1	Dodani sekvencijski dijagram	Nino Ćurko, Josip Šare, Davor Najev, Dominik Zoričić	13.11.2023.
0.5	Dodan opis korištene arhitekture te opisa baze podataka i njezinih tablica	Nino Ćurko	16.11.2023.

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

Rev.	Opis promjene/dodatka	Autori	Datum
0.5.1	Dodan prototip verzije dijagrama razreda	Nino Ćurko, Josip Šare	3.12.2023.
0.6	Dodan dijagram aktivnosti i dijagram stanja	18.12.2023.	

2. Opis projektnog zadatka

U ovo moderno doba i vrijeme, kada su tehnološka sredstva uvijek nadomak ruke, nekolicina entuzijasta se dosjetila kako doskočiti u pomoć svom gradu. Naime, štete po cestama i javnim površinama se ne smanjuju, ali to je upravo cilj jednog ovakvog projekta.

Ono što ovaj projekt nastoji ponuditi jest upravo jedno pravo suvremeno i efikasno rješenje pristupačno svima, a usmjereno samo prema jednoj stvari - otklanjanju štete javnih površina u gradu. Cilj projekta je svim ljudima omogućiti jednostavni pristup aplikaciji preko koje će moći prijavljivati uočena oštećenja javnih površina i cesta u područjima gradova čime sveukupno kumulira poboljšanje života u društvu.

Ukratko, tijekom zamišljenih događaja je idući: Korisnik uoči oštećenu gradsku imovinu ili cestu te istu želi prijaviti gradskom uredu nadležnom za taj tip štete. Korisnik učitava našu aplikaciju te šalje prijavu sa određenim popunjenim podacima. Gradski ured dobiva obavijest o prijavi te kad istu riješi korisnik koji ju je prijavio dobiva obavijest da je ona razriješena.

Pri učitavanju početne stranice aplikacije korisniku je vidljiva karta, opcija za registraciju (ili prijavu), te opcija za podnošenje prijave. Registrirani kao i neregistrirani korisnici imaju mogućnost slanja prijave, ali razlika je u tome što neregistrirani korisnik dobiva jedinstveni broj pomoću kojeg prati status prijave, registrirani korisnik uvijek može samo ući u povijest svojih prijava i vidjeti status svake podnesene

Neregistrirani korisnik se u svakom trenutku može registrirati, te pri registraciji trebe navesti iduće podatke:

- Ime
- Prezime
- Korisničko ime
- E-mail

- Lozinku

Registrirani korisnici se uvijek mogu prijaviti u sustav pomoću svog korisničkog imena i lozinke. Ako je korisnik prijavljen u sustav, ima mogućnost odjave ponuđenu u gornjem desnom kutu. Registrirani korisnici su podijeljeni na iduć uloge:

- Klijent
- Administrator
- Gradski ured

Klijent na karti ima označene sve aktivne prijave. Pri pregledu prijava, prijave može filtrirati po tematici (tipu) i lokaciji. Pri predaji prijave, korisnik mora unijeti odgovarajuće podatke. Također korisniku nije zabranjeno mijenjanje vlastitih osobnih podataka ako vidi potrebu za tim.

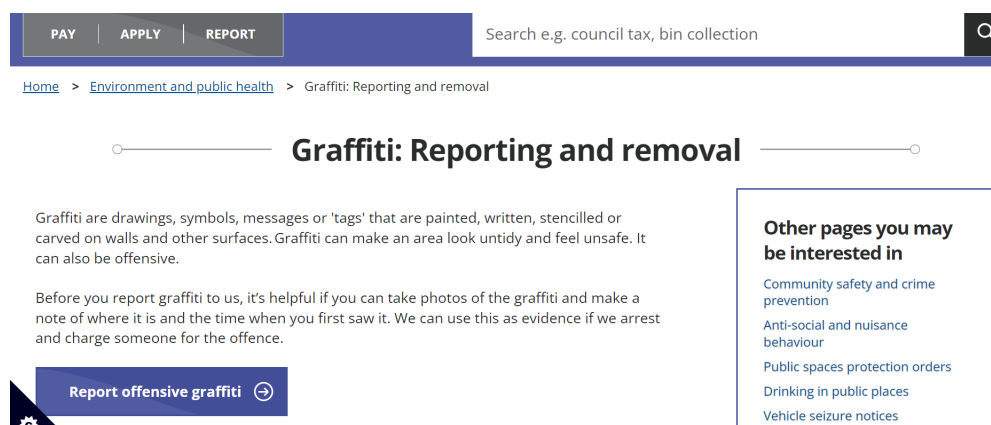
Prijava koja se šalje u sustav se sastoji od atributa:

- Naziv
- Kratki opis
- Geografske koordinate

U slučaju da korisnik šalje prijavu koja je na obližnjoj lokaciji neke već druge prijave poslane u sustav sustav ga o tome obavještava tako da mu izbacuje obližnju prijavu i ponudu mu opciju nadovezivanja na tu istu u slučaju da korisnik uoči kako je to upravo ta prijava koju je on htio poslati. Uz sve to, korisnik ocionalno može poslati i sliku viđenog i potom sustav, u slučaju da nisu unesene koordinate, langitudu i longitudu izvlači sa meta podataka fotografije.

Web stranica koja nudi sličnu uslugu, ali je bazirana na vandalizmu i isključivo grafitima u Londonu je:

<https://www.newham.gov.uk/publichealth-safety/graffiti-reporting-removal>, a izgled je na slici 2.1.

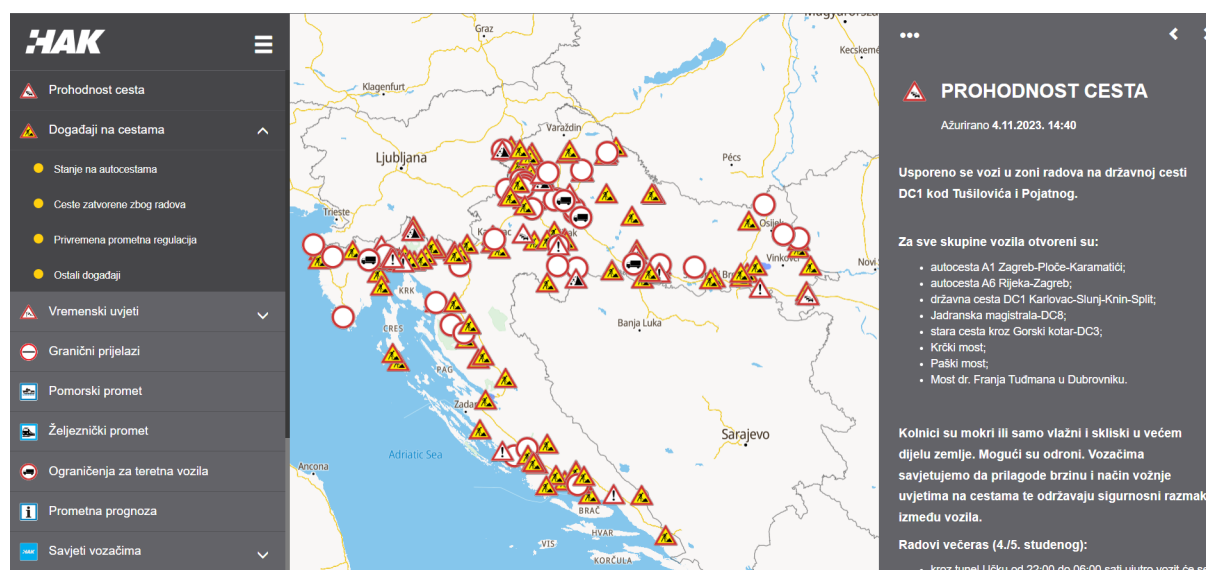


Slika 2.1: Izgled navedene stranice

Administrator je uloga koja ima najveće ovlasti. On je tu da kontrolira sve vezano za stranicu pa tako ima mogućnost uređivanja podataka pojedinih prijava kojih smatra nevaljalima kao i njihovo brisanje iz baze podataka. Pored toga, može brisati profile registriranih korisnika za koje procjeni da krše pravila ponašanja na aplikaciji, te potvrđuje ili opovrgava registraciju novih gradskih ureda za koje procijeni validnost.

Uz navedene korisnike sustava, još postoje i gradski uredi/vijeća koji primaju prijave na temelju tematike problema za koju su zaduženi. Tako će puknuće na cestama primiti isključivo javni zavod za ceste, probleme sa zgradama će obrađivati ured za izgradnju i prostorno uređenje, probleme sa vodovodom ured za vodovod... Dodatno, svaki gradski ured uvijek ima mogućnost promjene statusa prijave koji će prijaviteljima omogućiti da jasno vide ukoliko je prijava razriješena ili nije. Uz to, ured može povezati one prijave koje korisnici nisu povezali ukoliko shvati da se radi o istom problemu iste tematike na konkretno bliskom području. Nadalje, ured prihvaća zahtjeve registriranih korisnika za ulazak u ured, to jest participaciju kako bi korisnik (koji je zaposlenik u tom uredu) dobio ovlasti gradskog ureda. Kreiranje (registracija) novih ureda u sustavu mora biti odobrena od strane administratora.

Sustav će sve prijave obrađivati u stvarnom vremenu pa će korisnici u bilo kojem trenutku i sami uvid da li je npr. neko puknuće na cesti razriješeno te da li se tom cestom promet uopće može odvijati. Sličnu mogućnost nudi stranica HAK-a na idućoj adresi: <https://www.hak.hr/info/stanje-na-cestama/#prohodnost-cesta> na slici refhak.



Slika 2.2: HAK - stanje na cestama

Ovakva aplikacije sama po sebi već ima klijentelu i predispozicije za uspješno korištenje i popularizaciju. Uz štete javnih površina i cesta moglo bi se dodati prijava za zastoje primjerice tramvaja na određenoj lokaciji (naravno u gradovima gdje je tramvajski prijevoz omogućen) kako bi bilo vidljivo svim korisnicima aplikacije u stvarnom vremenu. Uz to moglo bi se područje aplikacije proširiti na prometne nesreće kako bi korisnici uz status da li je cesta zatvorena ili ne, mogli uz sliku procijeniti prohodnost iste. Neke vizualne stvari za korisnike koje se mogu još dodati bi bile gledanje tuđih profila, kao i dopisivanje porukama kako bi se doznale konkretnije informacije o prijavi od osobe koja je tu prijavu napravila. Besmisleno je izostaviti je Hrvatska turistička zemlja, te aplikaciji bi se još mogli dodati ostali osnovni jezici za inozemne korisnike. Kako bi se što više promoviralo prijavljivanje šteta, u aplikaciju bi se dala ugraditi neka online nagrađivanja; primjerice dostupnost povećanja levela i stjecanja points-a.

3. Specifikacija programske potpore

3.1 Funkcionalni zahtjevi

Dionici:

1. Regitrirani korisnik
 - (a) Klijent
 - (b) Administrator
2. Neregistrirani (anonimni) korisnik
3. Razvojni tim
4. Gradski ured

Aktori i njihovi funkcionalni zahtjevi:

1. Klijent (inicijator) može:
 - (a) Prijaviti se u sustav koristeći svoj email i lozinku
 - (b) Uređivati osobne podatke
 - (c) Poslati prijavu u sustav
 - Unijeti naziv za prijavu, opis prijave, geografske koordinate, te opcionalno fotografiju
 - Odabrati koordinate preko karte ili unijeti najbližu adresu
 - Povezati svoju prijavu na postojeću (ako takva postoji)
 - (d) Pregledati prijave i podatke vezane za njih
 - Odabrati aktivnu prijavu na karti
 - Pregledati povijest svojih prijava
 - Filtrirati pregled prijava po lokaciji i temi
 - (e) Registrirati novi gradski ured
 - (f) Zatražiti ulazak za participaciju u određeni gradski ured
 - (g) Mijenjati podatke aktivne prijave

2. Administrator (inicijator) može:

- (a) Pregledati popis svih prijava ikad napravljenih u sustavu
 - Uređivati podatke prijava
 - Brisati prijave
- (b) Pregledati i uklanjati registrirane profile po potrebi
- (c) Odobravati kreaciju novih gradskih ureda

3. Gradski ured (inicijator) može:

- (a) Pregledati popis aktivnih zaprimljenih prijava
 - Prihvatiti (ili odbiti) određenu prijavu
 - Spojiti nepovezane prijave
 - Promijeniti status prihvaćene prijave ovisno o uspješnosti odrađene te iste prijave
- (b) Prihvatiti (ili odbiti) zahtjev korisnika za ulazak u taj ured
- (c) Poslati izvješće o odrađenoj prijavi na E-mail klijent

4. Neregistrirani korisnik (incijator) može:

- (a) Registrirati se u sustav koristeći ime, prezime, mail, username i lozinku
- (b) Poslati prijavu u sustav
 - Unijeti naziv za prijavu, opis prijave, geografske koordinate, te opcionalno fotografiju
 - Odabrati koordinate preko karte ili unijeti najbližu adresu
 - Povezati svoju prijavu na postojeću (ako takva postoji)
 - Zaprimiti jedinstveni ID za podnesenu prijavu

5. Baza podataka (sudionik) može:

- (a) Pohraniti sve podatke o korisnicima i njihovim ovlastima
- (b) Pohraniti svaku prijavu sa koreliranim podacima za istu

3.1.1 Obrasci uporabe

Opis obrazaca uporabe

UC01 - Registracija korisnika u sustav

- **Glavni sudionik:** Neregistrirani korisnik
- **Cilj:** Stvoriti korisnički račun
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Klijent bira opciju "registracija" na sučelju web aplikacije
 2. Klijent unosi tražene podatke
 3. Korisnik je upisan u bazu podataka
- **Opis mogućih odstupanja:**
 - 2.a Klijent unosi neispravni/postojeći username ili email
 1. Sustav obavještava korisnika o problemu i briše mu unesena polja
 2. Korisnik mijenja podatke u ispravne i registracija uspješno se pri-vede kraju

UC02 - Unos nove prijave u sustav

- **Glavni sudionik:** Korisnik
- **Cilj:** Podnijeti novu prijavu
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Korisnik upisuje tražene podatke pri unosu prijave
 2. Sustav javlja ako postoji vremenski bliska prijava na toj lokaciji
 3. Korisnik može povezati svoju prijavu na postojeću (ako takva postoji)
 4. Prijava se predaje i zapisuje u sustav
- **Opis mogućih odstupanja:**
 - 1.a Korisnik nije naveo sve zahtijevane podatke
 1. Sustav obavještava korisnika o problemu i javlja mu da popuni tražena polja

UC03 - Pregled aktivnih prijava u sustavu

- **Glavni sudionik:** Korisnik
- **Cilj:** Pregled postojećih prijava
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Korisnik otvara pregled svih postojećih prijava
 2. Korisniku se nudi opcija filtriranja po temi i lokaciji
 3. Na sučelju se prikazuju filtrirane prijave

UC04 - Prijava u sustav

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Prijaviti se svojim profilom u sustav
- **Sudionici:** Baza podataka
- **Preduvjet:** Registracija
- **Opis osnovnog tijeka:**
 1. Korisnik upisuje korisničko ime i lozinku
 2. Sustav javlja potvrdu ispravnosti unesinih podataka
 3. Korisniku se učitava njemu prilagođeno sučelje
- **Opis mogućih odstupanja:**
 - 1.a Korisnik krivo unio korisničko ime/lozinku
 1. Sustav obavještava korisnika o problemu i javlja mu da ispravi tražena polja

UC05 - Uređivanje podataka prijave

- **Glavni sudionik:** Administrator
- **Cilj:** Korigirati podatke vezane za odabranu prijavu
- **Sudionici:** Baza podataka
- **Preduvjet:** Dodijeljena prava administratora
- **Opis osnovnog tijeka:**
 1. Prikazuju se sve prijave
 2. Administrator može filtrirati prijave
 3. Administrator izmjenjuje podatke prijave
 4. Izmjenjena prijava se sprema u bazu podataka

UC06 - Brisanje prijave

- **Glavni sudionik:** Administrator
- **Cilj:** Obrisati određenu prijavu iz baze podataka
- **Sudionici:** Baza podataka
- **Preduvjet:** Dodijeljena prava administratora
- **Opis osnovnog tijeka:**
 1. Administratoru se prikazuje pregled prijave
 2. Administrator bira prijavu koju želi izbrisati
 3. Prijava se uklanja iz baze podataka i više nije vidljiva u aplikaciji

UC07 - Pregled podataka registriranih profila korisnika

- **Glavni sudionik:** Administrator
- **Cilj:** Uvid u profile registriranih korisnika i njihovih podataka
- **Sudionici:** Baza podataka
- **Preduvjet:** Dodijeljena prava administratora
- **Opis osnovnog tijeka:**
 1. Korisnik bira opciju za pregled svih profila
 2. Otvara mu se lista svih registriranih profila zajedno sa njihovim osobnim podacima

UC08 - Pregled liste osobnih podataka vlastitog profila

- **Glavni sudionik:** Klijent
- **Cilj:** Pregledati osobne podatke svog profila
- **Sudionici:** Baza podataka
- **Preduvjet:** Registracija
- **Opis osnovnog tijeka:**
 1. Korisnik ulazi u opis svog profila
 2. Sustav mu prikaže username, e-mail i lozinku

UC09 - Odjava iz sustava

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Odjaviti se iz sustava
- **Sudionici:** Baza podataka
- **Preduvjet:** Aktivna prijava
- **Opis osnovnog tijeka:**
 1. Korisnik bira opciju odjava
 2. Sustav ga vraća na početnu stranicu web aplikacije

UC10 - Brisanje korisničkog računa

- **Glavni sudionik:** Administrator
- **Cilj:** Obrisati određeni profil
- **Sudionici:** Baza podataka
- **Preduvjet:** Dodijeljena prava administratora
- **Opis osnovnog tijeka:**
 1. Administrator bira profil koji želi ukloniti
 2. Sustav ga za provjeru pita da potvrdi odluku
 3. Administrator potvrđuje i profil se uklanja iz baze podataka

UC11 - Pregledavanje povijesti svojih prijava

- **Glavni sudionik:** Klijent
- **Cilj:** Pregledati svu povijest privedenih prijava
- **Sudionici:** Baza podataka
- **Preduvjet:** Registracija
- **Opis osnovnog tijeka:**
 1. Korisnik bira opciju za prikazivanje povijesti prijava
 2. Sustav mu na sučelju prikazuje sve njegove prijave
 3. Korisnik dodatno može filtrirati iste po temi i lokaciji

UC12 - Pregledavanje profila gradskih ureda

- **Glavni sudionik:** Administrator
- **Cilj:** Pregledati popis svih postojećih gradskih ureda
- **Sudionici:** Baza podataka
- **Preduvjet:** Dodijeljena prava administratora
- **Opis osnovnog tijeka:**
 1. Administrator odabire opciju za pregled svih gradskih ureda zapisanih u sustavu
 2. Sustav mu na sučelje prikazuje gradske urede zapisane u bazi podataka zajedno sa njihovim opisima

UC13 - Povezivanje na postojeću prijavu

- **Glavni sudionik:** Korisnik
- **Cilj:** Vezati se na vremenski blisku prijavu na istom području
- **Sudionici:** Baza podataka
- **Opis osnovnog tijeka:**
 1. Korisnik predaje prijavu
 2. Sustav mu javlja za vremenski blisku prijavu na toj lokaciji
 3. Korisnik bira hoće li se vezati na postojeću prijavu ili kreirati vlastitu

UC14 - Odabir aktivnih prijava sa karte

- **Glavni sudionik:** Klijent
- **Cilj:** Pogledati na karti neriješene prijave i detalje za iste
- **Sudionici:** Baza podataka
- **Preduvjet:** Postojanje aktivnih prijava
- **Opis osnovnog tijeka:**
 1. Klijent na karti odabire opciju za uvid u aktivne prijave
 2. Sustav mu sve lokacije aktivnih prijava prikazuje na karti
- **Opis mogućih odstupanja:**
 - 1.a U sustavu nema aktivnih prijava
 1. Sustav korisnika izbacuje iz pregleda karte i javlja mu da nema aktivnih prijava

UC15 - Mijenjanje sadržaja aktivne prijave

- **Glavni sudionik:** Klijent
- **Cilj:** Promijeniti sadržaj vlastite aktivne prijave
- **Sudionici:** Baza podataka
- **Preduvjet:** Postojanje aktivne prijave
- **Opis osnovnog tijeka:**
 1. Klijent odabire svoju aktivnu prijavu koju želi izmjeniti
 2. Klijent mijenja atribut u prijavi
 3. Prijava s ažuriranim podakom se zapisuje u bazu podataka
- **Opis mogućih odstupanja:**
 - 1.a U trenutku odabira prijave, prijava više nije aktivna
 1. Sustav korisniku javlja da prijava više nije aktivna
 2. Korisnik je preusmjeren na početnu stranicu

UC16 - Pregled zaprimljenih prijava

- **Glavni sudionik:** Gradski ured
- **Cilj:** Pregledati sve zaprimljene prijave
- **Sudionici:** Baza podataka
- **Preduvjet:** Postojanje zaprimljenih prijava
- **Opis osnovnog tijeka:**
 1. Gradski ured bira izabire pregled svih aktivnih prijava
 2. Sustav mu iz baze podataka omogući uvid u sve prijave namijenjene njemu

- **Opis mogućih odstupanja:**

- 2.a Npostojanje prijava namijenjenih njemu

- 1. Sustav mu javlja da trenutno nema aktivnih prijava

UC17 - Prihvaćanje ili odbijanje zaprimljene prijave

- **Glavni sudionik:** Gradski ured

- **Cilj:** Privatiti ili odbiti prijavu iz pregleda prijava

- **Sudionici:** Baza podataka, Gradski uredi

- **Preduvjet:** -

- **Opis osnovnog tijeka:**

- 1. Gradski ured u pregledu svih prijava bira za pojedinu prijavu hoće li je prihvatiti ili odbiti

- 2. U slučaju prihvaćanja prijave, ta se ista uklanja it popisa aktivnih drugim uredima za koje je bila namijenjena

UC18 - Promjena statusa određene prijave

- **Glavni sudionik:** Gradski ured

- **Cilj:** Promijeniti status prihvaćene prijave

- **Sudionici:** Baza podataka

- **Preduvjet:** Prihvat određene prijave

- **Opis osnovnog tijeka:**

- 1. Gradski ured za prihvaćenu prijavu mijenja njen status ovisno je li uspješno odrađena ili ne

- 2. Status se zapisuje u bazu podatka

- 3. U slučaju neuspješno odrađene prijave, ta se ista vraća u popis aktivnih svim uredima za koje je i prije bila namijenjena

UC19 - Povezivanje nepovezanih prijava

- **Glavni sudionik:** Gradski ured

- **Cilj:** Spojiti više pojedinih prijava u jednu cjelinu

- **Sudionici:** Baza podataka, Korisnik (inicijator prijave)

- **Preduvjet:** -

- **Opis osnovnog tijeka:**

- 1. Gradski ured iz pregleda prijava odabire prijave koje želi povezati

- 2. Svaki prijavitelj čija je prijava povezana na druge sada to vidi u sustavu

UC20 - Prihvaćanje zahtjeva za ulazak registriranog korisnika u gradski ured

- **Glavni sudionik:** Gradski ured
- **Cilj:** Prihvatiti ili odbiti zahtjev korisnika za ulazak u gradski ured
- **Sudionici:** Baza podataka, Korisnik (inicijator zahtjeva)
- **Preduvjet:** Poslan zahtjev za ulazak u gradski ured
- **Opis osnovnog tijeka:**
 1. Gradski ured gleda poslane zahtjeve vezane za ućlanjenje korisnika u taj određeni ured
 2. U slučaju da zahtjevi postoje gradski ured bira hoće li prihvatiti novog člana ili ne
- **Opis mogućih odstupanja:**
 - 1.a Nepostojanje aktivnih zahtjeva za ulazak u ured
 1. Sustav javlja da trenutno nema poslanih zahtjeva

UC21 - Slanje zahtjeva za ulazak u gradski ured

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Poslati određenom gradskom uredu zahtjev za ulaz u isti
- **Sudionici:** Baza podataka, Gradski ured
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Korisnik odabire u tražilici određeni gradski ured
 2. Pri pronalasku traženog ureda korisnik šalje zahtjev za ulaz u isti

UC22 - Registracija novog gradskog ureda

- **Glavni sudionik:** Registrirani korisnik
- **Cilj:** Kreirati novi gradski ured
- **Sudionici:** Baza podataka, Administrator
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Korisnik odabire opciju kreiraj gradski ured
 2. Pri kreiranju novog gradskog ureda unosi podatke za isti
 3. Na temelju podataka administrator i baza podataka prihvaćaju ili odbijaju zahtjev za kreiranjem ureda

UC23 - Uređivanje osobnih podataka

- **Glavni sudionik:** Klijent
- **Cilj:** Promijeniti osobne podatke u bazi podataka
- **Sudionici:** Baza podataka
- **Preduvjet:** -
- **Opis osnovnog tijeka:**
 1. Korisnik bira opciju uređivanje osobnih podataka
 2. Mijenja podatak, te se novi zapisuje u bazu podataka
- **Opis mogućih odstupanja:**
 - 2.a Unos neispravnog podatka
 1. Baza podataka mu dojavljuje o grešci i traži ga da promijeni unos
 2. Korisnik odustaje od unosa ili prepravlja unos podataka

UC24- Praćanje prijave jedinstvenim ID-om

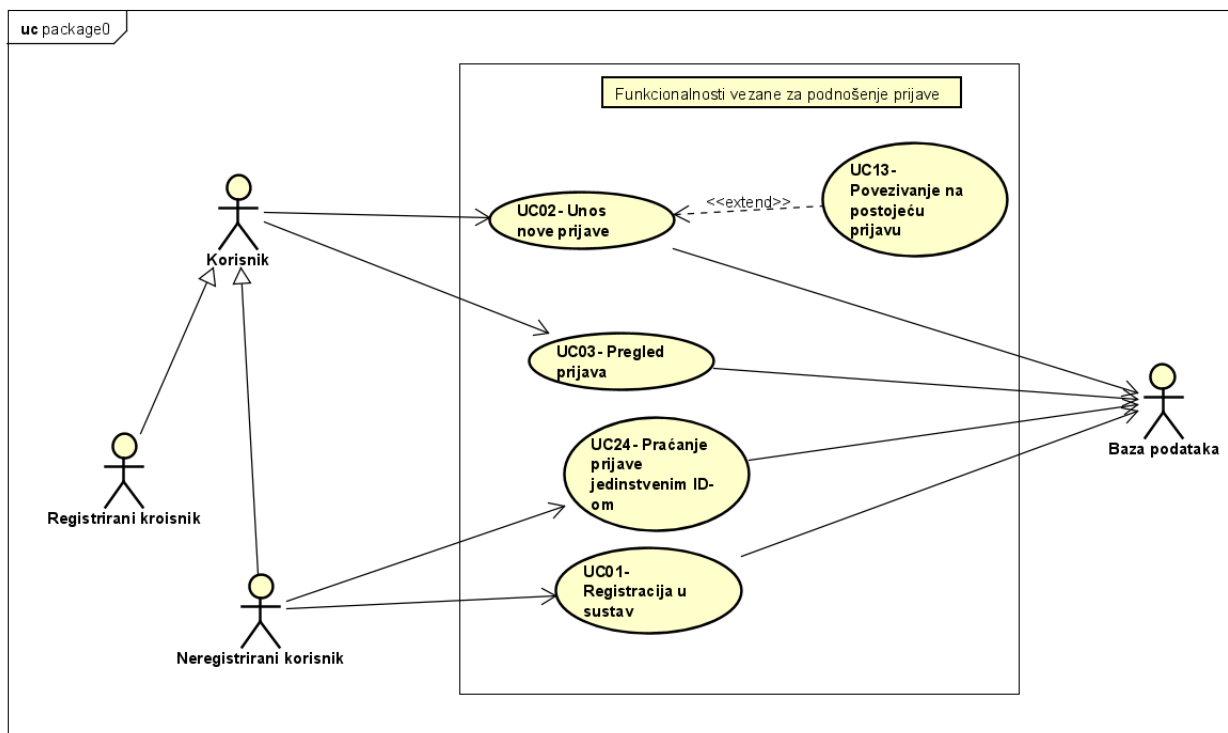
- **Glavni sudionik:** Neregistrirani korisnik
- **Cilj:** Pregledati svou prijavu unosom jedinstvenog broja
- **Sudionici:** Baza podataka
- **Preduvjet:** Postojanje jedinstvenog broja
- **Opis osnovnog tijeka:**
 1. Neregistrirani korisnik u tražilicu unosi ID koji je dobio pri unosu prijave
 2. Sustav mu pokaže njegovu prijavu i status ako je unesen broj važeći
- **Opis mogućih odstupanja:**
 - 1.a Unos neispravnog podatka
 1. Baza podataka mu dojavljuje o grešci i traži ga unese ispravan ID ili da odustane od prijave
 2. Korisnik bira opciju odustani ili opet unosi ID

UC25- Slanje obavijesti vezane za odrađenu prijavu na E-mail klijent

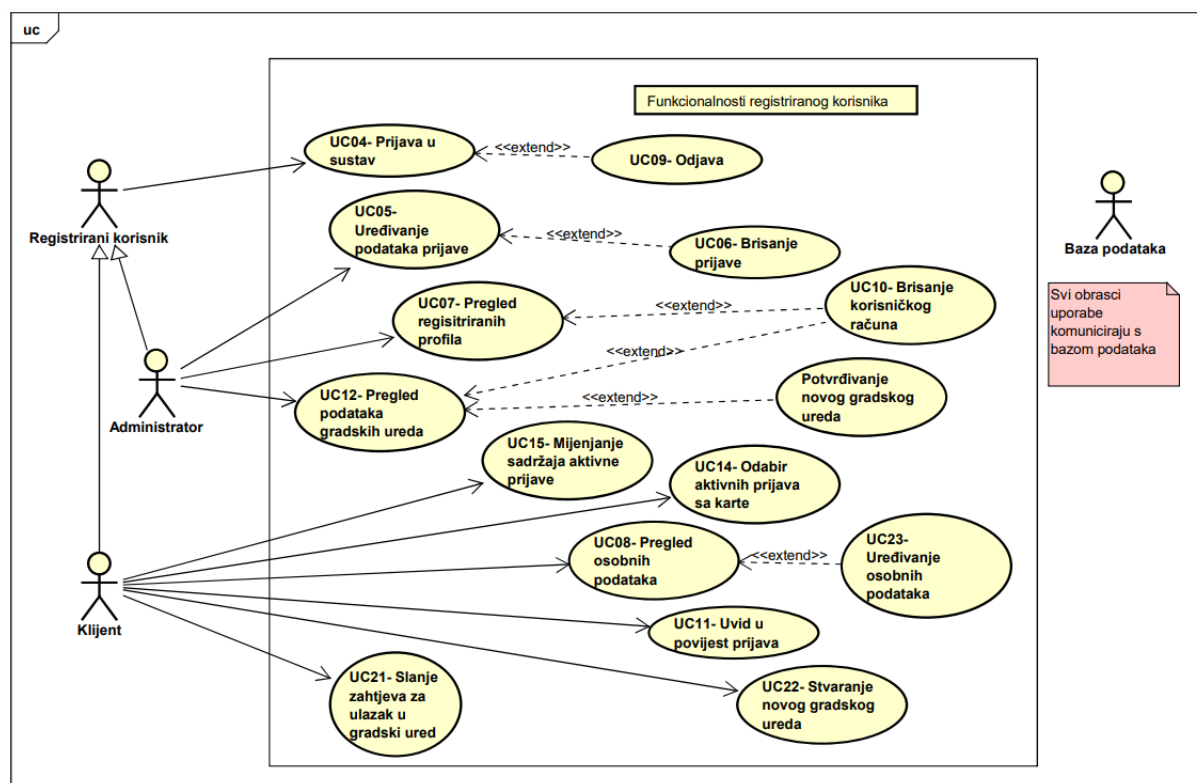
- **Glavni sudionik:** Gradski ured
- **Cilj:** Isporučiti objavu vezanu za odrađenu prihvaćenu prijavu na E-mail klijenta
- **Sudionici:** E-mail klijent
- **Preduvjet:** -
- **Opis osnovnog tijeka:**

1. Gradski ured šalje prouku o prijavi na E-mail klijent
2. E-mail klijent prikazuje podnosiocu prijave obavijest

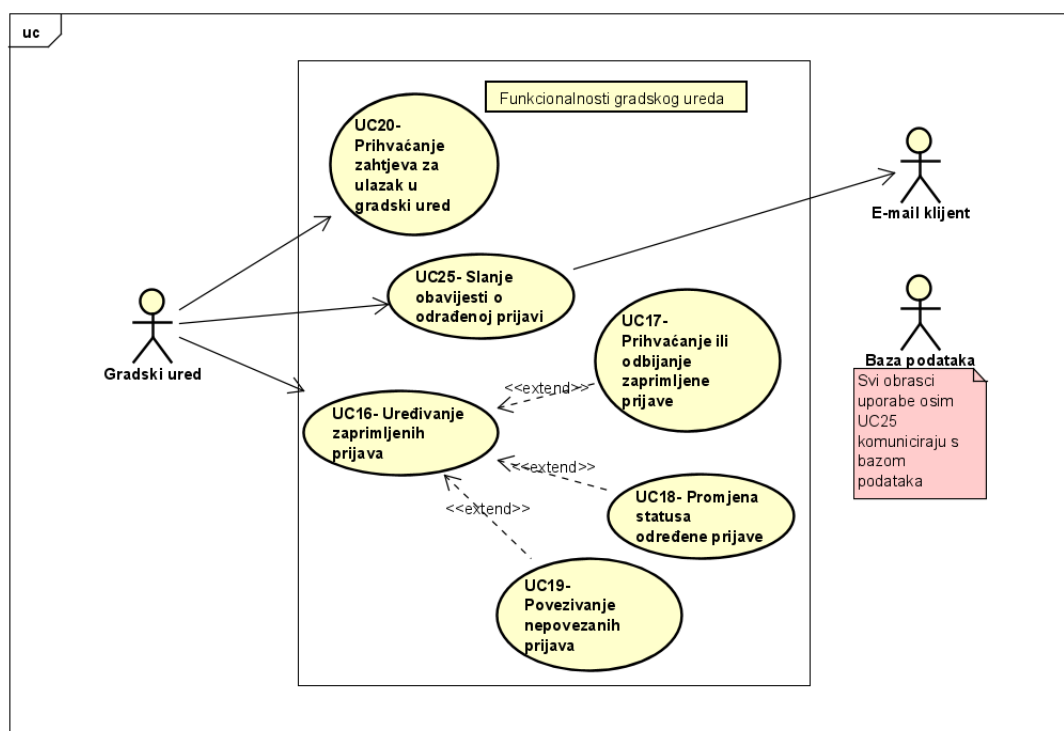
Dijagrami obrazaca uporabe



Slika 3.1: Dijagram obrasca uporabe - Funkcionalnosti vezane za podnošenje prijave



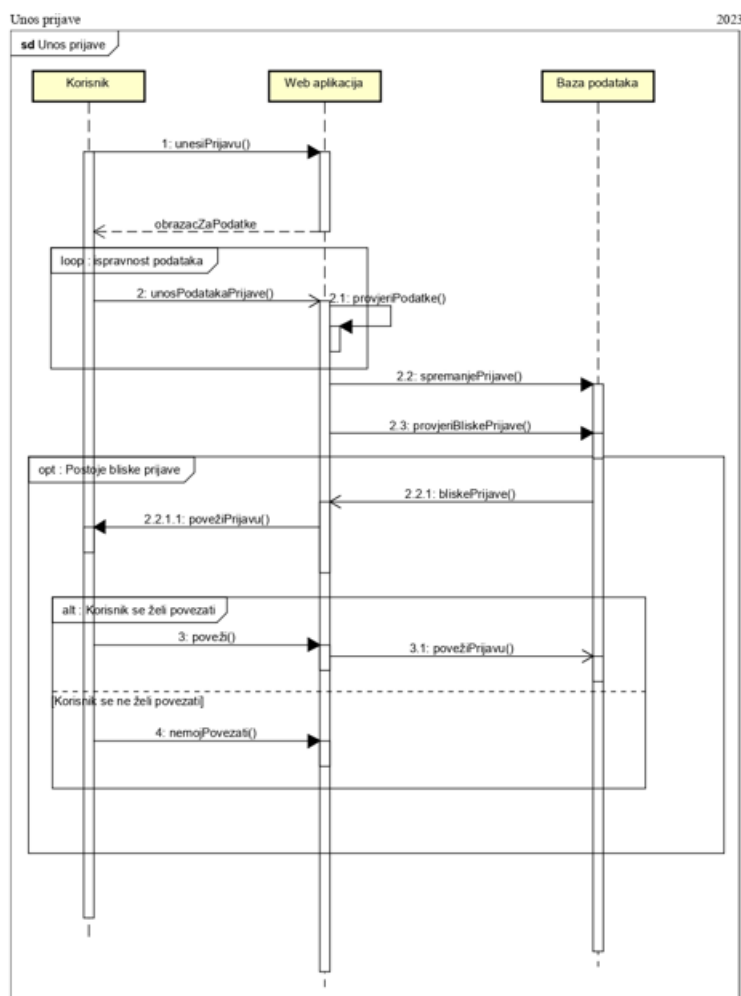
Slika 3.2: Dijagram obrasca uporabe - Funkcionalnosti registriranog korisnika (administrator i klijent)



Slika 3.3: Dijagram obrasca uporabe - Funkcionalnosti gradskog ureda

3.1.2 Sekvencijski dijagrami

Obrasci uporabe UC02, UC03, UC13, UC14 - unos nove prijave, pregled postojećih prijavi, povezivanje na postojeću prijavu, uvid u aktivne prijave sa karte Korisnik (prijavljeni ili anonimni) može podnijeti novu prijavu sustavu. Korisnik odabire opciju unosa nove prijave nakon čega poslužitelj vraća formu za upis podataka prijave. Korisnik šalje prijavu te sustav provjerava unos korisnika i provjerava u postoje li vremenski i prostorno bliske aktivne prijave te ako postoje, vraća formu korisniku gdje se može spojiti na neku od bliskih prijavi. Korisnik šalje poslužitelju hoće li se i na koju povezati te poslužitelj sprema novu prijavu u bazu podataka. Korisnik također može poslati zahtjev za pregledom svih postojećih prijavi uz opcionalne filtere po temi i lokaciji na što poslužitelj reagira dohvaćanjem tih podataka iz baze te slanjem istih korisniku.

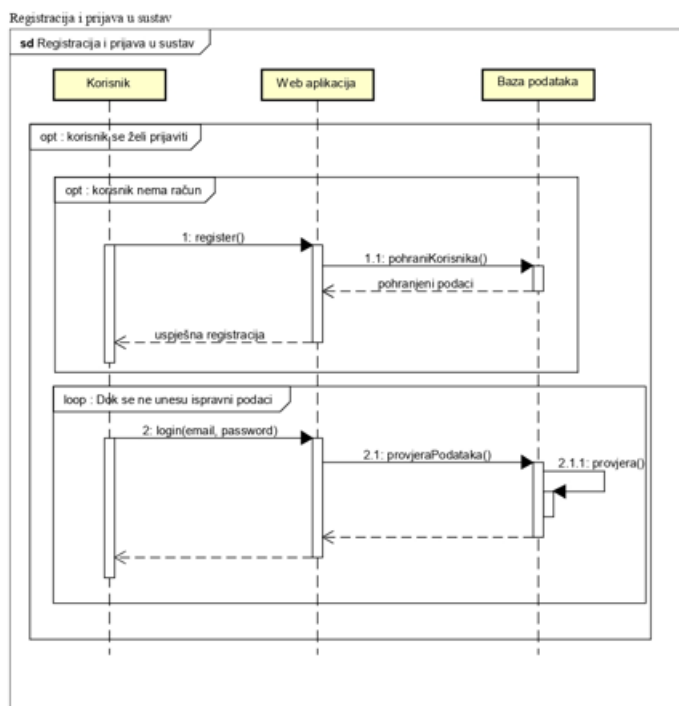


Slika 3.4: Sekvencijski dijagram - UC02, UC03, UC13, UC14

Obrasci uporabe UC01, UC04, UC08, UC11, UC15 - registracija i prijava korisnika u sustav

Neprijavljeni korisnik može poslati zahtjev za registracijom na što poslužitelj odgovara formom za registraciju. Kad korisnik pošalje podatke, poslužitelj ih validira te sprema u bazu podataka. Prijava korisnika u sustav, pregled osobnih podataka, pregled povijesti vlastitih prijava, mijenjanje sadržaja aktivne prijave.

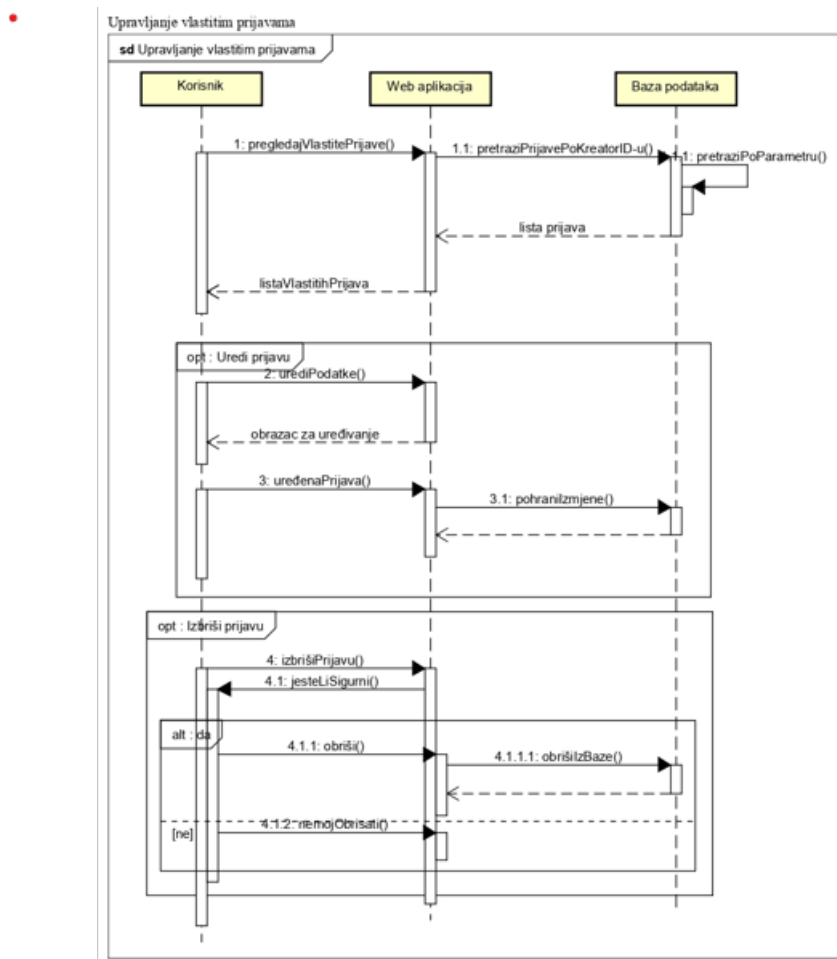
Registrirani korisnik može poslati zahtjev za prijavom u sustav na što poslužitelj odgovara formom za prijavu. Kad korisnik prijavu pošalje, poslužitelj ju validira uz pomoć baze podataka te obavještava korisnika o uspiješnosti prijave. Prijavljeni korisnik može poslati poslužitelju zahtjev za pregledom povijesti vlastitih prijava na što ih poslužitelj dohvaća iz baze podataka te ih vraća korisniku. Na isti način prijavljeni korisnik može pregledati osobne podatke.



Slika 3.5: Sekvencijski dijagram - UC01, UC04, UC08, UC11, UC15

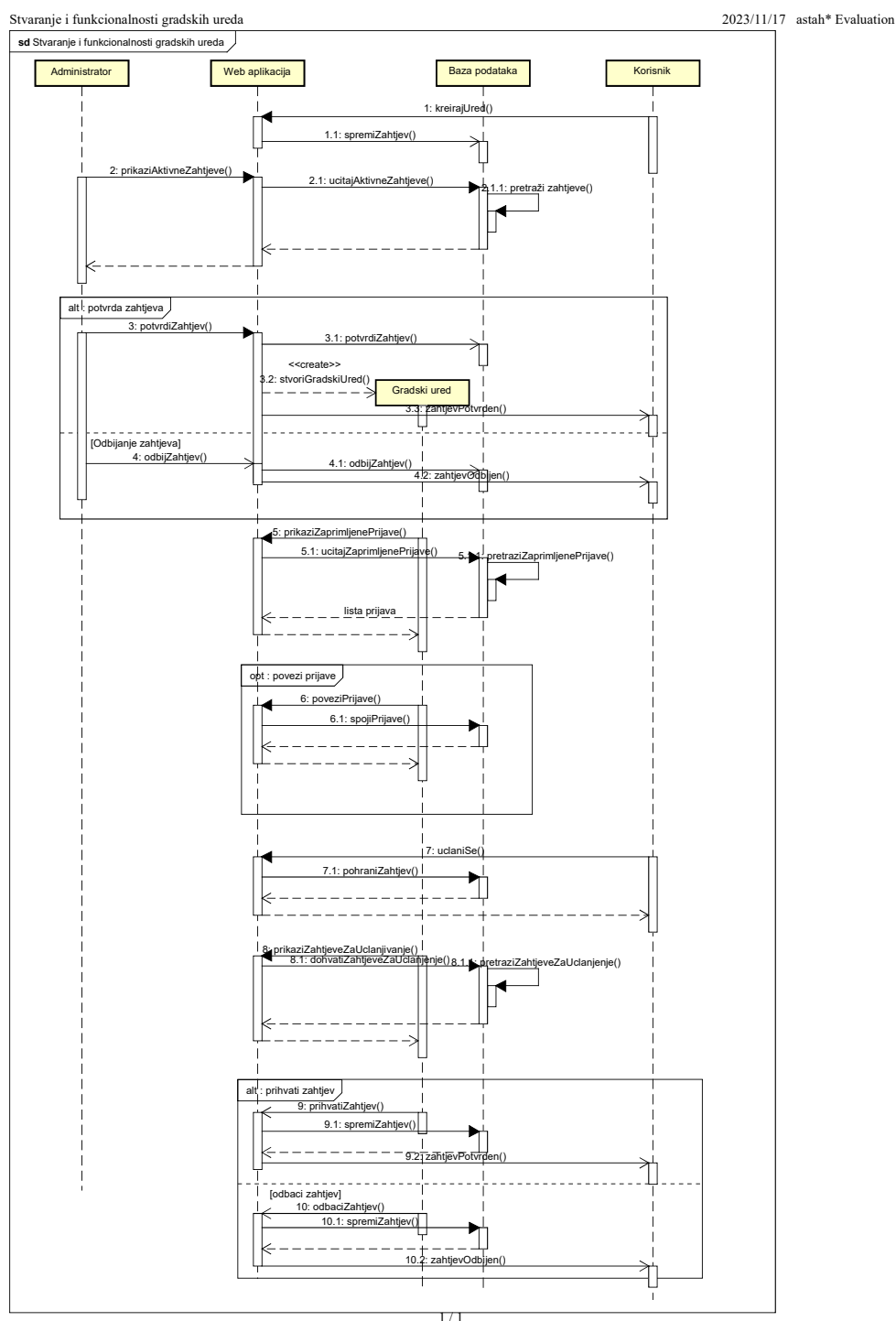
Obrazac uporabe UC05, UC06, UC11 - uređivanje podataka prijave, brisanje prijave, pregled korisnika

Korisnik u svakom trenutku može pregledati povijest svojih prijava. Prijave koje su još aktivne, tj. nisu odrađene može izbrisati u potpunosti iz sustava ili im mijenjati podatke.



Slika 3.6: Sekvencijski dijagram - UC05, UC06, UC11, UC15

Obrasci uporabe UC22, UC20, UC16, UC17, UC18, UC19 Prijavljeni korisnik može odlučiti kreirati novi gradski ured. Administrator to prvo mora odobriti da bi se transakcija uspješno dovila. Gradski ured može prihvatiti prijave koje je zaprimio, te listu istih uređivati po volji - mijenjati status ovisno o uspješnosti odrađene prijave, povezivati nepovezanih prijava. Također prijavljeni korisnici imaju mogućnost slanja zahtjeva za ulazak u gradski ured, što gradski ured može prihvatiti ili odbiti.



Slika 3.7: Sekvencijski dijagram - UC22, UC20, UC16, UC17, UC18, UC19

3.2 Ostali zahtjevi

- Sustav treba biti implementiran u obliku web aplikacije
- Aplikacija treba biti uvijek dostupna
- Aplikacije treba pružati usluge u stvarnom vremenu
- Učitavanje aplikacije ne smije trajati duže od 2 sekunde
- Pristup bazi podataka ne smije trajati duže od 2 sekunde
- Sustav treba biti organiziran u obliku MVC
- Sustav na poslužiteljskoj strani je napisan u programskom jeziku Java te radnom okviru Spring Boot
- Sustav na klijentskoj strani je implementiran programskim jezikom JavaScript te radnom okviru React.js
- Podaci se spremaju u bazu podataka koristeći JPA
- Arhitektura aplikacije mora biti u obliku klijent-poslužitelj
- Pri pristupu aplikaciji se koristi protokol HTTPS
- Aplikacije treba biti prilagođena i desktop uređajima kao i mobilnim uređajima
- Veza s bazom podataka mora biti kvalitetno zaštićena, brza i otporna na vanjske greške
- Pristup sustavu mora biti omogućen iz javne mreže pomoću HTTPS

4. Arhitektura i dizajn sustava

Arhitektura ovog projekta se može razlučiti na tri podsustava:

- **Web aplikacija**
- **Web poslužitelj**
- **Baza podataka**

Web poslužitelj je podsustav kojemu je zadaća spremanje, obrađivanje i dostavljanje klijentima sadržaj web stranica. Preko web aplikacije poslužitelj komunicira sa klijentom koji je u ovom slučaju web preglednik. Komunikacija se odvija putem protokola aplikacijskog sloja interneta - HTTP (Hypertext Transfer Protocol). On je tu da "reagira" na akcije koje mu web preglednik proslijedi te o ovisno o potrebi proljeđuje zahtjev na web aplikaciju.

Web preglednik je program koji fizičkom korisniku omogućuje učitavanje i pregled web aplikacije. On je tu kao prevoditelj što znači da interpretira prikaz koda pisanog i uređivanog u HTML-u u izgled razumljiv korisnicima. Putem web preglednika korisnik šalje HTTP zahtjeve web poslužitelju, no isto tako web poslužitelj je tu da prilagodi prikaz HTTP odgovora i prikaže ih korisniku.

Baza podataka je podsustav u podatkovnom sloju kojem je primaran uloga sigurno spremanje podataka, a detaljnije o njoj se govori u poglavlju 4.1.

Web aplikacija je dio sustava kojeg korisnik koristi za obrađivanje željenih zahtjeva koji uzrokuju pristupanje podacima u bazi podataka. Web aplikacija za interpretiranje dostavlja web pregledniku određeni HTML kod, a podijeljena je na back-end i na front-end o kojima je riječ u nastavku.

- **Front-end** ili klijentska strana je dio aplikacije koji služi za sve ono vidljivo na web pregledniku. On je prezentacijski sloj koji korisniku omogućuje jednostavnu komunikaciju sa sustavom. Tehnologija koja je korištena za realizaciju ovog dijela web aplikacije je programski jezik JavaScript, preciznije radni okvir React u kombinaciji sa TypeScriptom.

- **Back-end** ili poslužiteljska stran je dio aplikacije koji je zadužen za obrađivanje dobivenih zahtjeva i vršenja funkcionalnih akcija. Tehnologija u kojoj je pisan je programski jezik Java, preciznije u radnom okviru SpringBoot.

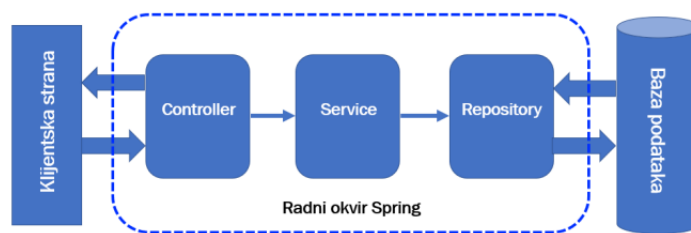
Okruženje u kojem je pisan front-end je Visual Studio Code, a back-end u IntelliJ-u. To nisu fiksna okruženja i može se koristiti bilo koji drugi IDE ili text editor. Arhitektura sustava se temelji na višeslojnom arhitekturnom stilu koji je podržan od strane SpringBoot tehnologije.

Višeslojni stil arhitekture

Višeslojna arhitektura sastoji se od idućih slojeva:

- klijentske strane koja omogućuje prikaz korisničkog sučelja
- nadglednika koji korisničku i poslužiteljsku stranu
- usluge koja obavlja poslovnu logiku i ostvaruje temeljnu funkcionalnost i zadaću web aplikacije
- repozitorija koji definira način pristupanja podacima
- baze podataka koja u našem slučaju podatke sprema u relacijsku bazu PostgreSQL

Osnovna značajka ovakvog stila arhitekture su da svaki sloj pruža uslugu drugom sloju, a skriva svoj skup usluga. Jedna od prednosti ovog stila je olakšavanje ostvarivanja podjele brige u web aplikaciji jer se svaki sloj brine isključivo o svojoj funkcionalnosti. Klijentska strana razgovara sa web sučeljem, nadglednik putem REST API-ja pruža vanjsko sučelje i prihvata HTTP zahtjeve te poziva odgovarajuće usluge, a potom te iste vraća kao odgovor klijentu. Sloj usluge definira pristup uslugama web aplikacije (tko i kako pristupa). Sloj repozitorija nam osigurava pristup podacima, te preslikavanje domenskih objekata u bazu podataka koristeći formu 1:1. Skica ovakve arhitekture je dana na slici 4.1



Slika 6.5. Povezanost slojeva u radnom okviru Spring.

Slika 4.1: Primjer višeslojne arhitekture korištenjem radnog okvira Spring

4.1 Baza podataka

dio 1. revizije

Za potrebe sustava koristit će se relacijska baza podataka koja nam omogućuje jednostavnije modeliranje stvarnog svijeta. Baza je implementirana u PostgreSQL-u zbog njegove jednostavnosti i jer je tim najbolje upoznat s tim sustavom; njegovim limitacijama i pravilima. Entiteti stvarnog svijeta su prevedeni kao tablice (relacije) koje imaju ime i svoj skup atributa. Baza podataka osigurava nam jednostavnu pohranu, umetanje, izmjenu i dohvat podataka, te garantira njihovu sigurnost. Baza podataka koristi sljedeće attribute:

- prijave
- lokacije
- slike
- sorisnici
- tipovi_ostecenja
- gradski_uredi

4.1.1 Opis tablica

prijave je entitet koji sadrži sve važne informacije o podnesenim prijavama. Sastoji se od atributa: `id`, `ostecenje_id`, `lokacija_id`, `kreator_id`, `parent_prijava_id`, `prvo_vrijeme_prijave`, `vrijeme_otklona`, `naziv`, `opis`. Ovaj entitet u vezi je One-to-One s tablicom lokacije preko atributa `lokacija_id`, sa entitetom slike je u vezi One-to-Many preko atributa `id`. Sa entitetom tipovi_ostecenja je u vezi One-to-Many preko atributa `ostecenje_id`, u vezi Many-to-One sa entitetom korisnici preko atributa `kreator_id`, te je sama sa

sobom u refleksivnoj vezi One-to-Many preko atributa id.

prijave		
id	INT	jedinstveni identifikator prijave
ostecenje_id	INT	jedinstveni identifikator tipa ostecenja
kreator_id	INT	jedinstvni identifikator korisnika koji je poslao prijavu
parent_prijava_id	INT	jedinstveni identifikator prijave na koju se prijava nadovezala
prvo_vrijeme_prijave	INT	vrijeme slanja prijave
vrijeme_otklona	INT	vrijeme otklona prijave
opis	VARCHAR	opis poslane prijave
naziv	INT	konkretno ime poslane prijave

lokacije je entitet koji sadrži osnovne podatke o geografskoj lokaciji pojedine prijave. Sastoji se od atributa: lokacija_id, latitude, longitude. Preko atributa id je povezana vezom One-to-One sa relacijom Prijave.

Lokacije		
lokacija_id	INT	jedinstveni identifikator prijave
latitude	INT	geografska latituda lokacije
longitude	INT	geografska longituda lokacije

korisnici je entitet koji sadrži osobne podatke o registriranim korisnicima kao i njihovu pripadnost vijecu i ulogu koju obnašaju u sustavu. Sastoji se od atributa: id, ime, prezime, username, email, password, ured_id, role, active. Povezani su sa relacijom prijave u vezi One-to-Many preko atributa id, te sa relacijom gradski_uredi u vezi Many-to-One preko atributa ured_id.

korisnici		
id	INT	jedinstveni identifikator korisnika
ime	VARCHAR	ime registriranog korisnika
prezime	VARCHAR	prezime registriranog korisnika
username	VARCHAR	korisničko ime registriranog korisnika
email	VARCHAR	e-mail adresa registriranog korisnika
password	VARCHAR	lozinka za prijavu registriranog korisnika
ured_id	INT	jedinstveni identifikator vijeca/ureda kojem pripada
active	INT	jedinstveni token sesije prijave korisnika
role	VARCHAR	uloga koju korisnik obnaša u sustavu

slike je entitet koji sadrži podatke vezane za sliku podnešene prijave. Sastoji se od atributa: id, podatak, prijava_id. Preko entiteta prijava_id je povezan tablicom prijave u vezi Many-to-One.

slike		
id	INT	jedinstveni identifikator slike pojedine prijave
podatak	VARCHAR	string bitova koji sadrže sliku
prijava_id	INT	jedinstveni identifikator prijave za koju je poslana slika

tipovi_osećenja je entitet koji sadrži podatke vezane za opis oštećenja kao i id ureda koji se bavi tim tipom oštećenja. Sastoji se od atributa: id, naziv. Preko atributa id je povezan sa relacijom prijave u vezi Many-to-One, te je sa relacijom gradski_uredi u vezi One-to-One povezan preko istog atributa id.

tipovi_osećenja		
id	INT	jedinstveni identifikator vrste oštećenja

Nastavljeno na idućoj stranici

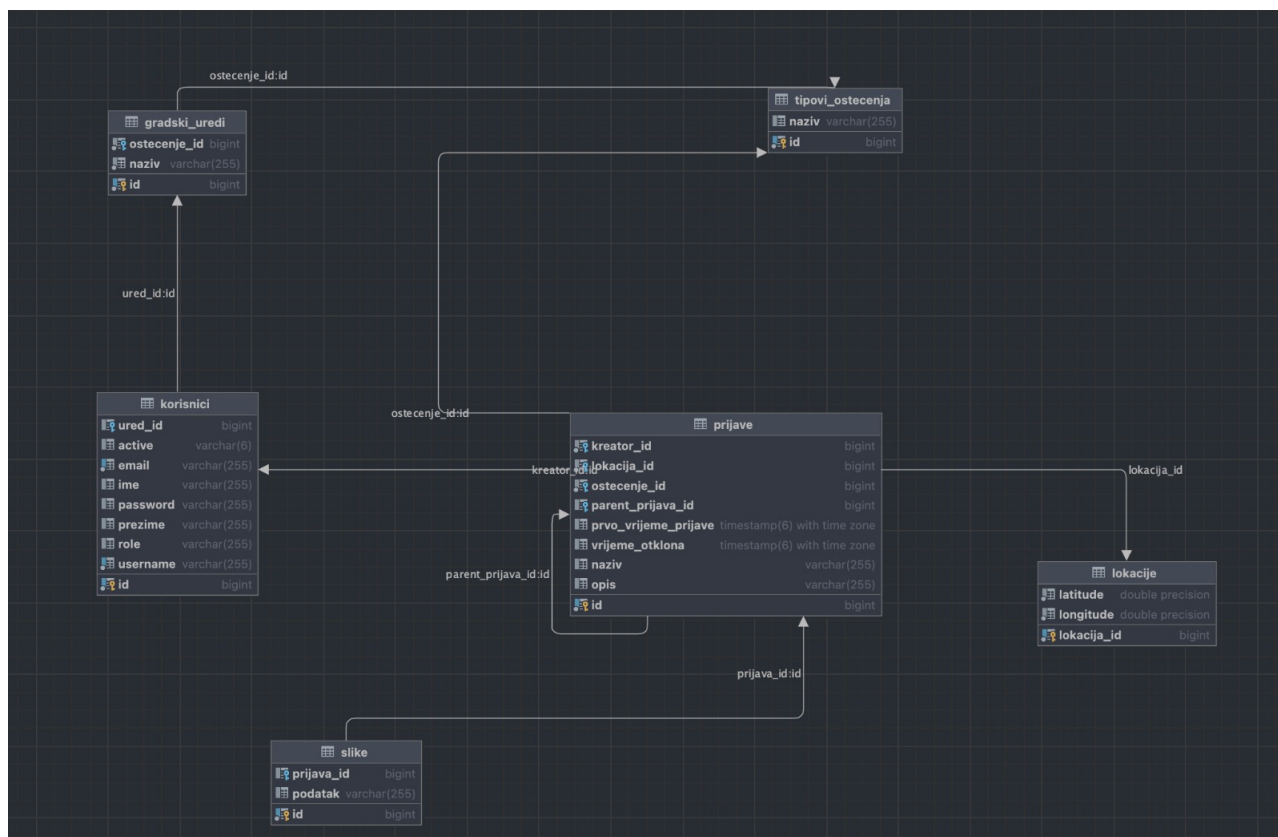
Nastavljeno od prethodne stranice

tipovi_ostecenja		
naziv	VARCHAR	naziv konkretnog oštećenja

gradski_uredi je entitet koji sadrži podatke vezane za opis pojedinog vijeća/ureda koji je zadužen za otklanjanje određenog tipa oštećenja. Sastoji se od atributa: id, naziv i ostecenje_id. Preko atributa ostecenje_id je povezan sa relacijom tipovi_ostecenja u vezi Many-to-One, te je povezan sa relacijom korisnici preko atributa id u vezi One-to-Many.

gradski_uredi		
id	INT	jedinstveni identifikator pojedinog registriranog vijeća/ureda
naziv	VARCHAR	puni naziv vijeća u aplikaciji
ostecenje_id	INT	identifikator vrste oštećenja koje za koje je taj ured zadužen

4.1.2 Dijagram baze podataka

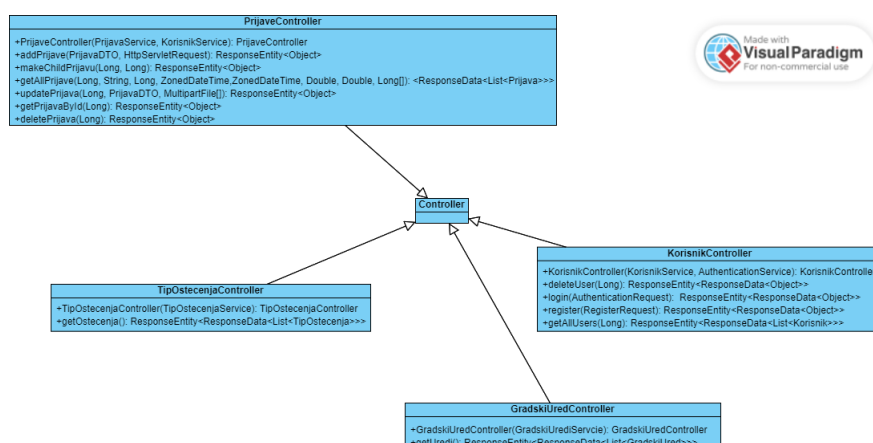


Slika 4.2: Relacijski dijagram baze podataka

4.2 Dijagram razreda

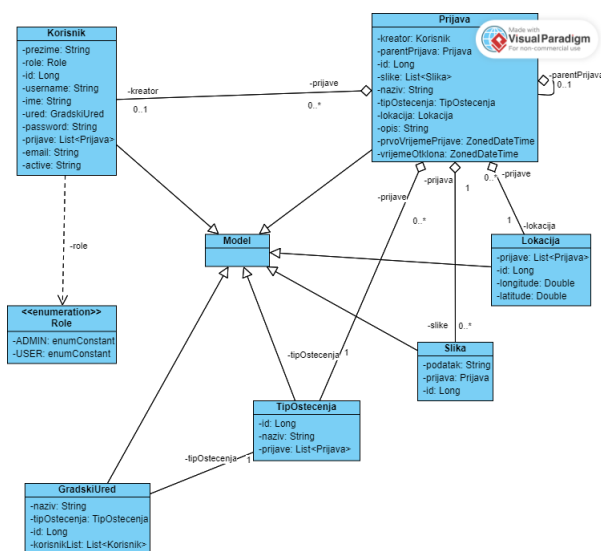
Na slikama 4.3 i 4.4 prikazani su razredi i njihove relacije na serverskoj strani aplikacije. Konkretnije rečeno prikazani su modeli i kontroleri.

Svi kontroleri nasljeđuju razred Controller. Metode koje su u njima implementirane ovise o DTO objektima, koji se realiziraju pomoću metoda implementiranih u Domain dijelu razreda.



Slika 4.3: Dijagram razreda - Controllers dio

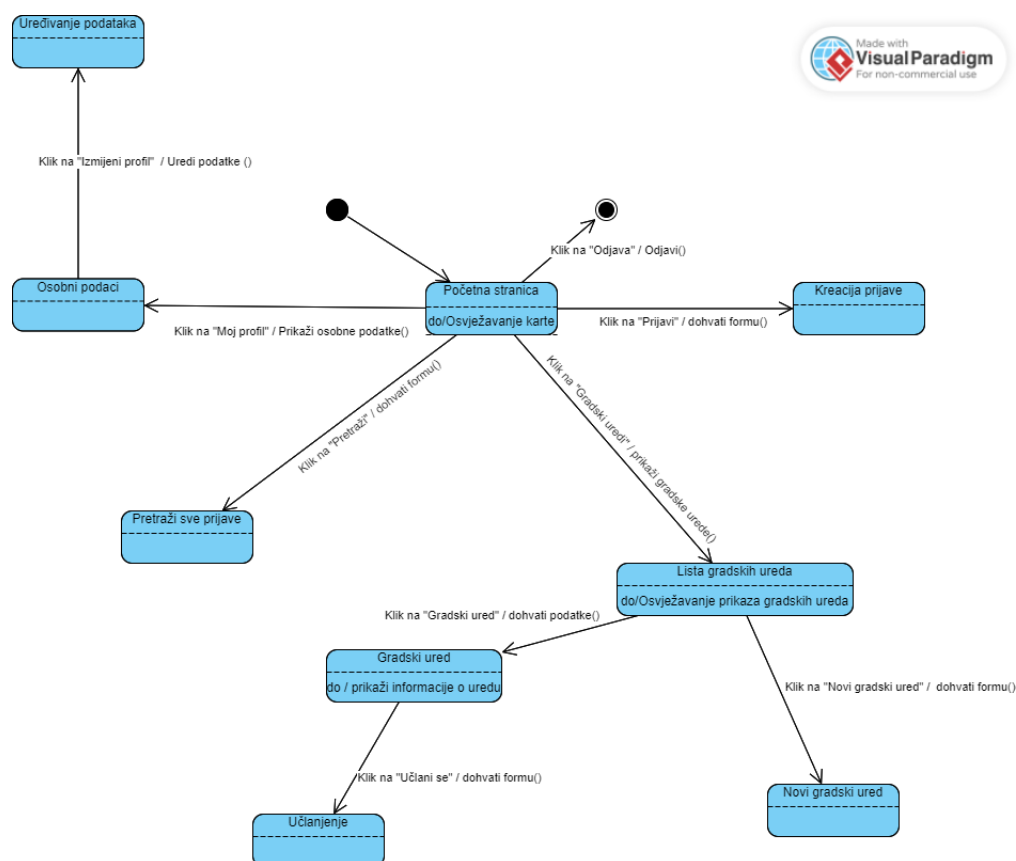
Domain dio razreda preslikava strukturu baze podataka, te svi razredi nasljeđuju razred Model. Ovako ilustrirani razredi su zapravo odraz onih relacija koji su zapisani u bazi podataka, te direktno komuniciraju sa bazom podataka. Razred Korisnik je ilustracija bilo kojeg registriranog usera koji ima svoju ulogu (povezan sa enumom Role), te može podnijeti prijavu u sustav (povezanost sa razredom Prijava preko private atributa prijave). Prijave kada se konstruiraju zadaje im se lokacija (pa su preko atributa lokacija povezane sa razredom Lokacija), slika (agregatna povezanost sa razredom Slika preko varijable slike) i tip oštećenja (agregatna povezanost sa razredom tipOstecenja preko atributa tipOstecenja). Razred GradskiUred je reprezentacija gradskog ureda koje je povezan sa tipom oštećenja preko atributa tipOstecenja.



Slika 4.4: Dijagram razreda - Controllers dio

4.3 Dijagram stanja

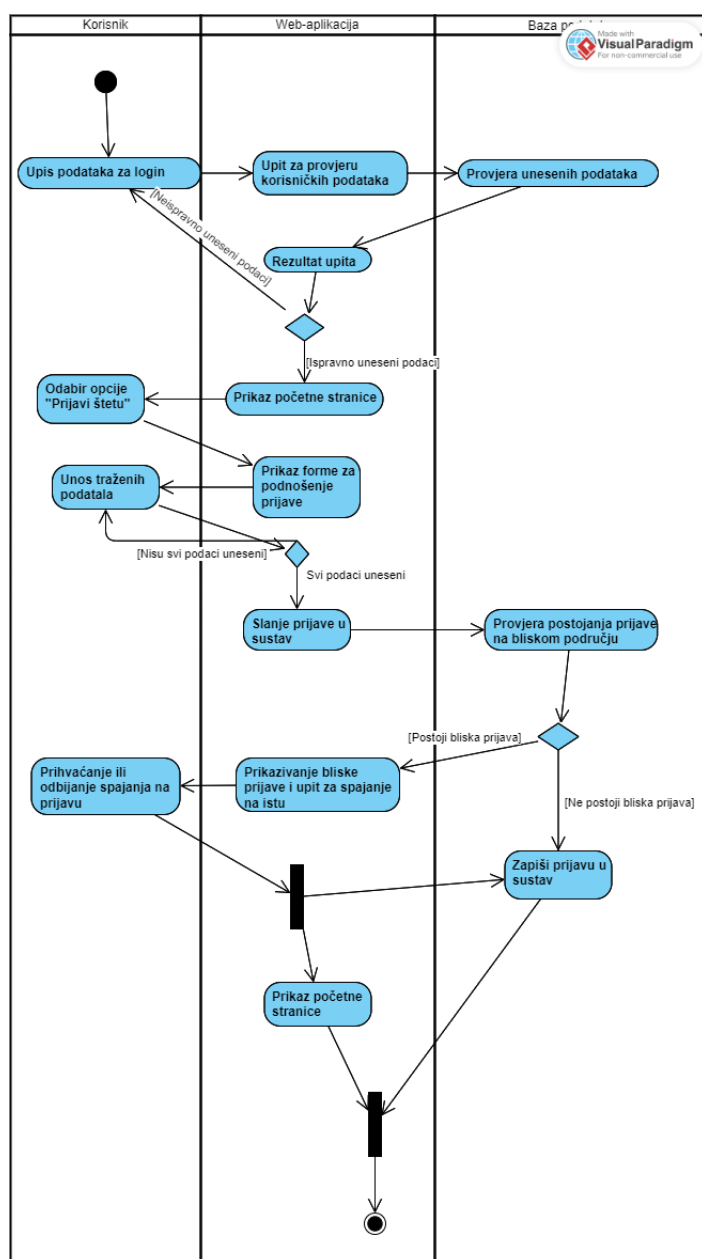
Dijagrami stanja su tu da predoče stanja objekta, te prijelaze iz jednog stanja u drugo na poticaj nekakvog događaja (okidača). Na slici je prikazan dijagram stanja za prijavljenog korisnika. Korisniku je prikazana početna stranica sa Google Maps kartom na kojoj su ozačene prijave koje su aktivne, ali i opcija za podnošenje nove prijave kao i pretraga prijava po određenim filterima. Pri odabiru opcije za podnošenje nove prijave otvara mu se forma koja zahtjeva unos određenih podataka karakterističnih za vrstu štete. Također, ako korisnik ne želi podnijeti novu prijavu u sustav, već po pretragama postojećih, može samo tu istu odabrati i klikom na "Pošalji" se ta ista prijava šalje opet u sustav. Nadalje, korisnik klikom na "Gradski uredi" može vidjeti listu postojećih gradskih ureda, ali i kreirati novi gdje mu se otvara forma koja zahtjeva unos podataka za novi ured. Korisnik u svakom trenutku može kliknuti na ikonicu profila i odabirom "Izmijeni profil" uređivati svoje podatke, ali i pregledati vlastite prijave klikom na "Pregled mojih prijave". Korisnik se iz sustava odjavljuje odabirom opcije "Odjava."



Slika 4.5: Dijagram stanja - Prijavljeni korisnik

4.4 Dijagram aktivnosti

Dijagram aktivnosti se koristi za opis modela toka upravljanja ili toka podataka. U modeliranju jednog takvog dijagrama aktivnosti, podrazumijeva se da se novi korak inicira tek kada prethodni završi. Na dijagramu prikazanom na idućoj slici prikazano je podnošenje prijave u sustav nakon logina korisnika. Korisnik su se nakon opcije prjavi otvara forma sa traženim podacima. Kada unese tražene podatke, korisnik podnosi prijavu u sustav.



Slika 4.6: Dijagram aktivnosti - Dodavanje prijave

5. Implementacija i korisničko sučelje

5.1 Korištene tehnologije i alati

Dio zadužen za frontend i komunikaciju s korisnikom je ostvaren pomoću knjižnice React, te je pisan u programskom jeziku TypeScript. React, kao biblioteka je široko raspostranjen u području stvaranja korisničkog sučelja. Jedna od glavnih prednosti jest objedinjavanje visokih performansi brzine aplikacije, korištenjem virtualnog DOM-a, te omogućuje podršku za ponovnu uporabivost komponenti i nadogradnju sustava (zbog svoje enkapsuliranosti). Razvila ga je grupa Meta (Facebook) i naglask se stavlja na njegovu pripadnost knjižnicama, a ne radnim okvirima.

Razvojno okruženje u kojem je pisan cijeli frotend je Visual Studio Code(tzv. VSC). VSC je integrirano razvojno okuženje u vlasništvu tvrtke Microsoft. Danas je to jedna od najkorištenijih tehnologija za uređivanje i pisanje koda, a to je s dobrim razlogom jer on nudi mnoštva ekstenzija za razne programske jezike (pa tako i za TypeScript u kojem je naša aplikacija pisana), ali i jednostavne funkcionalnosti koje ispadaju uvelike korisne te olakšavaju posao pisanja koda, a neke od tih su: debugger, IntelliSense koji služi kao autofill i ugrađena podrška za Git.

Backend sustava pisan je u programskom jeziku Java korištenjem radnog okvira Spring Boot. Spring Boot je implementiran kao specijalizacija generaliziranijeg radnog okvira Spring, a omogućuje izradu stand-alone aplikacija. Dobrim dijelom olakšava posao izgradnje sustava jer sam konfigurira funkcionalnosti Springa, ali i sam ima već podešene one dijelove web aplikacije koji se učestalo koriste (npr. servleti).

Cijeli backend je pisan u razvojnom okruženju InntelliJ IDEA. To je integrirano razvojno sučelje u vlasništvu kompanije JetBrains, a prevladavajuće je u području pisanja porgrama u Javi i Kotlinu. Velike funkcionalnosti koje koristi su coding assistance, remote cooperation kao i jednostavnost uporabe.

Za dokumentaciju se koristio programski jezik *LaTeX*. *LaTeX* je jezik za pisanje strukturiranih tesktova, piše se kao običan tekst sa dodanom semantičkom strukturom (ne prikazuje se kao konačan proizvod kao npr. Microsoft Word) što mu omogućuje stabilniji rad, a sve što zahtjeva je instalriana distribucija TeX-a; što

je u našem slučaju MiKTeX. Dokumentacija je pisana u uređivaču teksta zvanom Texmaker koji se ističe po jednostavnosti pisanju *LaTeX* dokumenata kao i u svojoj dostupnosti.

Realizacija baze podataka izvedena je U PostgreSQL-u. PostgreSQL je open-source sustav za upravljanje relacijama bazama podataka. Uvelike naglasak stavlja na ispunjavanju ACID svojstava pri izvođenju transakcija i pristupačan je za korištenje. Unutar PostgreSQL-a korišten je pgAdmin kao grafički alat za upravljanje bazama podataka i njihovh schema. U izradi UML dijagrama korištene su dvije tehnologije: Astah i Visual Paradigm. Oba se alata ističu po svojoj rasprostranjenosti tako što omogućavaju izbor kreiranja svakojakih dijagrama, ali i po svojoj jednostavnosti. Razlika je u tome što je Astah program koje se pokreće lokalno na računalu, a Visual Paradigm se pokreće online i sprema trenutne izmjene po izboru lokalno ili na Cloud.

Kako bi se lakše upravljalo verzijama projekta, korišten je Git. Git je besplatan, open source sustav koji se koristi za upravljanje kako manjih, tako i većih projekata. Prednosti su mu brzina, jednostavnost i lakoća upravljanja projektima u timskom radu. Vanjski repozitorij projekta se nalazi na besplatnoj web platformi GitHub koja omogućuje lako upravljanje projektom svim sudionicima repozitorija. Za produkciju aplikacije korišten je cloud sustav Render. Render olakšava puštanje aplikacija u pogon kako bi bile javno dostupne za pronalazak an internetu.

Kako bi se u timu maksimalno olakšala komunikacija članova kroišen je Discord. Discord je društvena platforma u kojoj je naglasak na jednostavnosti postizanja glasovne, video i tekstne komunikacije u zajednicama koje se nazivaju serveri kojima se pristupa pomoću poveznice, a omogućuju kavlitetan integritet tima (zajednice).

5.2 Ispitivanje programskog rješenja

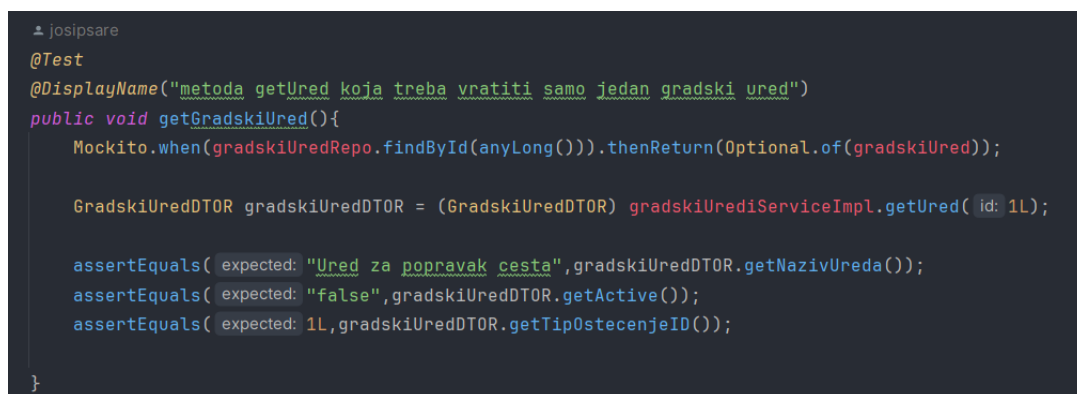
5.2.1 Ispitivanje komponenti

Potrebno je provesti ispitivanje jedinica (engl. *unit testing*) nad razredima koji implementiraju temeljne funkcionalnosti. Razraditi **minimalno 6 ispitnih slučajeva** u kojima će se ispitati redovni slučajevi, rubni uvjeti te izazivanje pogreške (engl. *exception throwing*). Poželjno je stvoriti i ispitni slučaj koji koristi funkcionalnosti koje nisu implementirane. Potrebno je priložiti izvorni kôd svih ispitnih slučajeva te prikaz rezultata izvođenja ispita u razvojnom okruženju (prolaz/pad ispita).

U ovom potpoglavlju razrađeno je testiranje backend dijela aplikacije sa konkretno implementiranim testovima. Testovi su JUnit testovi, a na slici 5.1 prikazan je vremenski interval svakog od 7 testova, kao i verifikacija za njihov prolaz.

Test 1: Metoda koja vraća registrirani gradski ured u bazi podataka

U ovom je testu cilj bio iz baze podataka "izvući" jedan gradski ured koji odgovara određenim atributima. Dobivena relacije je perotvorena u DTO oblik za lakše rukovanje u sustavu.



```
josipsare
@Test
@DisplayName("metoda getUred koja treba vratiti samo jedan gradski ured")
public void getGradskiUred(){
    Mockito.when(gradskiUredRepo.findById(anyLong())).thenReturn(Optional.of(gradskiUred));

    GradskiUredDTOR gradskiUredDTOR = (GradskiUredDTOR) gradskiUredServiceImpl.getUred( id: 1L);

    assertEquals( expected: "Ured za popravak cesta", gradskiUredDTOR.getNazivUreda());
    assertEquals( expected: "false", gradskiUredDTOR.getActive());
    assertEquals( expected: 1L, gradskiUredDTOR.getTipOstecenjeID());
}
```

Slika 5.1: Pregled gradskog ureda određenih atributa metodom getGradskiUred()

Test 2: Metoda koja provjerava postojanost novostvorenog ureda

U ovom je testu cilj bio pregledati sadržaj baze podataka nakon stvaranja novog gradskog ureda, konkretnije ispravnost zapisa njegovih atributa.

```

└─ josipsare
@Test
@DisplayName("metoda koja potvrđuje stvaranje gradskog ureda")
public void potvrdiGradskiUred(){

    when(gradskiUredRepo.findById(anyLong())).thenReturn(Optional.of(gradskiUred));
    when(gradskiUredRepo.save(any(GradskiUred.class))).thenReturn(gradskiUredSaved);
    when(korisniciRepo.findByPendingZahtjevOdredeniUred(anyLong())).thenReturn(List.of(korisnik));

    // Act
    GradskiUredDTOR result = (GradskiUredDTOR) gradskiUrediServiceImpl.potvrdiUred(id: 1L);

    assertEquals( expected: "true", result.getActive());
    assertEquals( expected: "active", korisnik.getUred_status());

    verify(gradskiUredRepo, times( wantedNumberOfInvocations: 1)).findById(1L);
    verify(gradskiUredRepo, times( wantedNumberOfInvocations: 1)).save(gradskiUred);
    verify(korisniciRepo, times( wantedNumberOfInvocations: 1)).findByPendingZahtjevOdredeniUred( ured_id: 1L);
    verify(korisniciRepo, times( wantedNumberOfInvocations: 1)).save(korisnik);
}

```

Slika 5.2: Potvrđivanje novonastalog gradskog ureda metodom potvrdiGradskiUred()

Test 3: Pretraživanje nepostojećeg korisnika u bazi podataka

U ovom je testu cilj bio pronaći nepostojećeg korisnika u bazi podataka i uočiti baca li se adekvatna iznimka.

```

@ExtendWith(MockitoExtension.class)
public class KorisikServiceTest {
    3 usages
    @Mock
    private KorisniciRepository korisniciRepo;
    3 usages
    @InjectMocks
    private KorisnikServiceImpl korisnikService;
    └─ josipsare
    @Test
    @DisplayName("metoda getKorisnik koja treba vratiti iznimku o ne postojećem korisniku")
    public void GetKorisnikFalse(){
        // Arrange
        Long userId = 1L;
        when(korisniciRepo.findById(userId)).thenReturn( t Optional.empty());

        // Act & Assert
        assertThrows(RecordNotFoundException.class, () → korisnikService.getAllUsers(userId));
    }
}

```

Slika 5.3: Pregled postojanosti nepostojećeg korisnika

Test 4: Pregled svih postojećih korisnika u sustavu

U testu prikazanom na idućoj slici kreirano više korisnika sa vlastitim atributima. Novonastali korisnici su dodani u listu te je na testa kraju vraćena odgovarajuća lista u koju su dodani spomenuti korisnici kako bi se provejrio zapis objekata u bazi podataka.

```
@Test
@DisplayName("metoda getKorisnik koja vraća sve korisnike")
public void getKorisnikAll() {
    // Arrange
    List<Korisnik> userList = new ArrayList<>();
    Korisnik korisnik1 = new Korisnik();
    korisnik1.setId(1L);
    korisnik1.setActive("true");
    korisnik1.setUsername("username");
    korisnik1.setIme("name");
    korisnik1.setPrezime("lastname");
    korisnik1.setPassword("pass");
    korisnik1.setEmail("email@email.com");

    Korisnik korisnik2 = new Korisnik();
    korisnik2.setId(2L);
    korisnik2.setActive("true");
    korisnik2.setUsername("username2");
    korisnik2.setIme("name2");
    korisnik2.setPrezime("lastname2");
    korisnik2.setPassword("pass2");
    korisnik2.setEmail("email2@email.com");

    userList.add(korisnik1);
    userList.add(korisnik2);

    when(korisniciRepo.findAll()).thenReturn(userList);
    List<Korisnik> result = korisnikService.getAllUsers(id: null);
    assertEquals(userList, result);
}
```

Slika 5.4: Pregled liste svih postojećih korisnika sa svojim atributima u sustavu metodom getKorisnikAll()

Test 5: Registracija novog korisnika u sustav

U testu broj pet kreiran je jedan user sa svojim pripadajućim atributima i spremljen u bazu podataka. Cilj je dobiti istog tog korisnika pretragom po određenom atributu i učiti ispravnost zapisa novonastalog objekta

```
@Test
@DisplayName("metoda getKorisnik koja vraća sve korisnike")
public void getKorisnikAll() {
    // Arrange
    List<Korisnik> userList = new ArrayList<>();
    Korisnik korisnik1 = new Korisnik();
    korisnik1.setId(1L);
    korisnik1.setActive("true");
    korisnik1.setUsername("username");
    korisnik1.setIme("name");
    korisnik1.setPrezime("Lastname");
    korisnik1.setPassword("pass");
    korisnik1.setEmail("email@email.com");

    Korisnik korisnik2 = new Korisnik();
    korisnik2.setId(2L);
    korisnik2.setActive("true");
    korisnik2.setUsername("username2");
    korisnik2.setIme("name2");
    korisnik2.setPrezime("Lastname2");
    korisnik2.setPassword("pass2");
    korisnik2.setEmail("email2@email.com");

    userList.add(korisnik1);
    userList.add(korisnik2);

    when(korisniciRepo.findAll()).thenReturn(userList);
    List<Korisnik> result = korisnikService.getAllUsers(id: null);
    assertEquals(userList, result);
}
```

Slika 5.5: Registracija novog korisnika

Test 6: Pregled svih prijava u sustavu

U predzadnjem provedenom testu kreirane su nove prijave. Cilj je bio iz baze podataka izvući sve porstojeće prijave u sustavu kako bi se provjerila točnost njihovog zapisa u bazi podataka.

```
 Dominik Zoradic
@Test
public void testGetAllPrijave() {
    TipOstecenja ostecenje = new TipOstecenja();
    ostecenje.setId(1L);
    ostecenje.setNaziv("ostecenje1");

    GradskiUred ured = new GradskiUred();
    ured.setId(1L);
    ured.setTipOstecenja(ostecenje);

    Prijava prijava = new Prijava();
    prijava.setId(1L);
    prijava.setNaziv("prijava1");
    prijava.setGradskiUred(ured);

    Prijava prijava2 = new Prijava();
    prijava2.setId(1L);
    prijava2.setNaziv("prijava2");

    when(prijaveRepo.findAll()).thenReturn(Arrays.asList(prijava, prijava2));
    when(prijaveRepo.findAllByTipOstecenja(ostecenje.getId())).thenReturn(Arrays.asList(prijava));

    List<Prijava> res = prijavaService.getAllPrijave(kreatorid: null, active: null, parentid: null, dateFrom: null, dateTo: null, lat: null, lng: null, uredid: null, ostecenje.getId());

    assertEquals("expected: 'prijava1', res.get(0).getNaziv()");
}
```

Slika 5.6: Pretraga svih prijava regsitriranih u bazi podataka

Test 7: Dodavanje nove prijave u sustav

U zadnjem provedenom testu kreirane su nove prijave. Cilj je bio potvrditi potpunost zapisa nove prijave u bazi podataka.

```
1 Dominik Zoricic +
@Test
public void testAddPrijava() {
    PrijavaDTO prijavaDTO = new PrijavaDTO( naziv: "Test Prijava", opis: "Opis", uredid: 1L, latitude: 45.0, longitude: 15.0, new MultipartFile[]{});
    GradskiUred gradskiUred = new GradskiUred( naziv: "Gradski ured", tipOstecenja: null, korisnikList: null, active: "true");
    Korisnik korisnik = new Korisnik( username: "username", ime: "name", prezime: "password", password: "surname", email: "email@gmail.com");
    Lokacija lokacija = new Lokacija( latitude: 45.0, longitude: 15.0);
    Prijava prijava = new Prijava(lokacija, naziv: "test prijava", gradskiUred, opis: "Opis", korisnik, slike: null, parentPrijava: null, ZonedDateTime.now(), vrijemeOtklona: null);
    prijava.setId(1L);

    when(gradskiUrediRepo.findById(1L)).thenReturn(Optional.of(gradskiUred));
    when(korisnikRepo.findById(1L)).thenReturn(Optional.of(korisnik));
    when(lokacijaRepo.save(any(Lokacija.class))).thenReturn(lokacija);
    when(prijavaRepo.save(any(Prijava.class))).thenReturn(prijava);
    Object result = prijavaService.addPrijava(prijavaDTO, req: null, korisnik);
    assertNotNull(result);
    assertTrue(result instanceof Map);
    Map<String, Object> resultMap = (Map<String, Object>) result;
    assertTrue(resultMap.containsKey("newReport"));
    assertTrue(resultMap.containsKey("nearbyReports"));
    Prijava savedPrijava = (Prijava) resultMap.get("newReport");
    assertNotNull(savedPrijava);
    assertEquals( expected: "test prijava", savedPrijava.getNaziv());
    assertEquals( expected: "Opis", savedPrijava.getOpis());
    verify(lokacijaRepo).save(any(Lokacija.class));
    verify(gradskiUrediRepo).findById(1L);
    // 2 poziva zbog prisutnih slika
    verify(prijavaRepo, times( wantedNumberOfInvocations: 2)).save(any(Prijava.class));
}
```

Slika 5.7: Kreacija novih prijava u sustavu

5.2.2 Ispitivanje sustava

Potrebno je provesti i opisati ispitivanje sustava koristeći radni okvir Selenium¹. Razraditi **minimalno 4 ispitna slučaja** u kojima će se ispitati redovni slučajevi, rubni uvjeti te poziv funkcionalnosti koja nije implementirana/izaziva pogrešku kako bi se vidjelo na koji način sustav reagira kada nešto nije u potpunosti ostvareno. Ispitni slučaj se treba sastojati od ulaza (npr. korisničko ime i lozinka), očekivanog izlaza ili rezultata, koraka ispitivanja i dobivenog izlaza ili rezultata.

Izradu ispitnih slučajeva pomoću radnog okvira Selenium moguće je provesti pomoću jednog od sljedeća dva alata:

- dodatak za preglednik **Selenium IDE** - snimanje korisnikovih akcija radi automatskog ponavljanja ispita

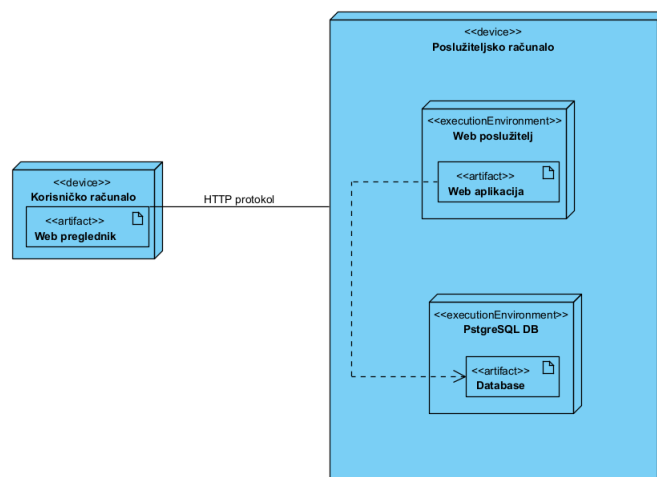
¹<https://www.seleniumhq.org/>

- **Selenium WebDriver** - podrška za pisanje ispita u jezicima Java, C#, PHP koristeći posebno programsko sučelje.

Detalji o korištenju alata Selenium bit će prikazani na posebnom predavanju tijekom semestra.

5.3 Dijagram razmještaja

Dijagram razmještaja opisuje topologiju virtualnih i fizičkih čvorova i komunikacijske puteve između istih. U našem slučaju sustav je temeljen na klijent-poslužitelj odnosu u kojem klijentsko računalo http protokolom razmjenjuje podatke sa serverskim uređajem. Serverska je strana implemenirana pomoću web servisa Render.



Slika 5.8: Dijagram razmještaja

5.4 Upute za puštanje u pogon

5.4.1 Konfiguracija frontenda

Pri konfiguraciji frontenda na poslužitelju Render potrebno je GitHub račun spojiti s Renderom. Idući je korak kreirati novi servis i odabrati projekt Gradska šteta. Regija je Frankfurt (EU Central), za root directory se odabire frontend. Environment se treba postaviti na Node, dok je build command potrebno postaviti na `npm run build`, a start command na `npm run server`. Zadnji korak je postaviti environment varijable ka na idućoj slici i kliknuti Create Web Server.



The screenshot shows the 'Environment Variables' configuration page in the Render dashboard. At the top, there is a title 'Environment Variables' and a subtitle 'Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more.](#)'. Below this, there is a table with two columns: 'Key' and 'Value'. The table contains six rows of environment variables: CLIENT_URL, DB_HIBERNATE, DB_PASS, DB_URL, DB_USERNAME, and PORT. Each row has a text input field for the key, a text input field for the value (all masked with dots), and a trash icon to the right. At the bottom left, there is a link 'Create Environment Group'. At the bottom right, there are two buttons: '+ Add Environment Variable' and 'Save Changes'.

Key	Value
CLIENT_URL
DB_HIBERNATE
DB_PASS
DB_URL
DB_USERNAME
PORT

[Create Environment Group](#) [+ Add Environment Variable](#) [Save Changes](#)

Slika 5.9: Environment varijable za frontend dio

5.4.2 Konfiguracija backenda

Pri konfiguraciji backenda na poslužitelju Render potrebno je GitHub račun spojiti s Renderom. Idući je korak kreirati novi servis i odabrati projekt Gradska šteta. Kao regiju treba odabrati Frankfurt (EU Central), za root directory se odabire backend. Dockerfile se nalazi u backend direktoriju (`/backend`), a Docker Build Context directory je `/backend`. Na idućoj slici su prikazane environment varijable koje treba postaviti i kliknuti na Create Web Server.

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more.](#)

Key	Value	
CLIENT_URL	
DB_HIBERNATE	
DB_PASS	
DB_URL	
DB_USERNAME	
PORT	

[↩ Create Environment Group](#) [+ Add Environment Variable](#) [Save Changes](#)

Slika 5.10: Environment varijable za backend dio

5.4.3 Konfiguracija baze podataka

Pri konfiguraciji poslužitelja baze podataka na Renderu potrebno je postaviti ime baze zajedno sa imenom korisnika baze. Kao regiju postavizi Frankfurt(EU Central) i dovršiti kreiranje sa klikom na Create Database. Environment varijable prikazane su na slici 5.11.

Connections

Hostname

○

dpg-clb2c63mot1c7386u1vg-a

Port

5432

Database

pojip

Username

pojip_user

Password

👁

.....

Internal Database URL

👁

.....

External Database URL

👁

.....

PSQL Command

👁

.....

Slika 5.11: Environment varijable za bazu podataka

6. Zaključak i budući rad

Zadatak našeg tima VelicanstveniTimRaketa bio je proizvesti web aplikaciju kojom suvremeni građanin može uočenu javnu štetu skenirati i prijaviti. Izizevši prva dva tjedna semestra, rad na aplikaciji i dokumentaciji redovno se vršio u nezanimljivoj satnici svakog tjedna. Projekt je bio podjeljen na dvije faze:

1) **Prva faza** je bila problematičnija jer se problem činio apstraktan, a oskudnog znanja za preporučene tehnologije se ne može reći da nije bilo. Svakako poznavanje članova već od prije je uvelike pridonijelo samoj podjeli rada na projektu i olakšanom dogovaranju oko izrade zadataka, ali i preporučivanja tehnologija za samu izvedbu. Tim se podijelio tako što su trojica preuzela ulogu developera na backend-u, dvojica na frontend-u te jedan na dokumentaciji projekta. Kako je prva faza stavila velik naglasak na dokumentaciju projekta pa se prema tome rad i odvijao. Pisanje dokumentacije je pridonijelo i samom razumijavanju problema jer je pomoću dijagrama obrazaca i sekvencijskih dijagrama služila kao smjernica kuda projekt treba ići. Prva je faza trajala do 17.11.2023. kada je i predana aplikacija s generičkim funkcionalnostima, no ono što je važnije jest da se na kraju ove faze cijeli tim približio sa tehnologijama i osjećao komfort u njihovom okruženju.

2) **Druga faza** krenula je odmah početkom prosinca, a naglasak je stavila nešto više ovaj put na funkcionalnost aplikacije uz ponešto manje količine dokumentacije. Ovdje su članovi bili usmjereni da i najmanje sitnice dovedu do prihvatljivog oblika kako bi se aplikacija mogla pustiti u pogon. Ovdje je kemija backend i frontend tima bila ključ napretka pošto je realizacije dijela jednog tima ovisila o implementaciji istog dijela kod drugog tima. Nakon realizacije funkcionalnosti bilo je potrebno dovršiti dokumentaciju traženim dijagramima i napraviti testove za važnije stvari da se ta funkcionalnost pokaže. Ova faza trajala je do 19.1.2024. kada je i finalna verzija uspješno objavljena.

Na početku projekta se ovo sve činilo kao nemoguća misija zbog nepoznavanja tehnologija pa su sukladno tome, naizgled jednostavni zadaci prerasli u komplek-

sne. Na svu sreću u timu je bilo volje i ambicije za napredak pa je iz tjedna u tjedan anksioznost oko projekta nestajala i na kraju se odradio sav željeni dio sustava koji se planirao pa se može reći da je zadovoljstvo tima bilo prisutno. O stečenom iskustvu se može pričati samo u pozitivnom kontekstu. Naime, uistinu smo iz prve ruke okusili kako to sve izgleda, barem donekle, u stvarnom svijetu i radu u timskom okruženju. Naravno, nije sve bilo perfektno pa tako jedna veća mana bila nedostatak poznavanja kako načina pristupa problemu, tako i znanja u potrebnim tehnologijama. Ipak sve se to naposljetku prebrodilo i možemo reći da se sa stečenim znanjem i otvorenog uma može ići samo naprijed.

Popis literature

Kontinuirano osvježavanje

Popisati sve reference i literaturu koja je pomogla pri ostvarivanju projekta.

1. Programsko inženjerstvo, FER ZEMRIS, <http://www.fer.hr/predmet/proinz>
2. I. Sommerville, "Software engineering", 8th ed, Addison Wesley, 2007.
3. T.C.Lethbridge, R.Langaniere, "Object-Oriented Software Engineering", 2nd ed. McGraw-Hill, 2005.
4. I. Marsic, Software engineering book", Department of Electrical and Computer Engineering, Rutgers University, <http://www.ece.rutgers.edu/~marsic/books/SE>
5. The Unified Modeling Language, <https://www.uml-diagrams.org/>
6. Astah Community, <http://astah.net/editions/uml-new>

Indeks slika i dijagrama

2.1	Izgled navedene stranice	7
2.2	HAK - stanje na cestama	8
3.1	Dijagram obrasca uporabe - Funkcionalnosti vezane za podnošenje prijave	20
3.2	Dijagram obrasca uporabe - Funkcionalnosti registriranog korisnika (administrator i klijent)	21
3.3	Dijagram obrasca uporabe - Funkcionalnosti gradskog ureda	22
3.4	Sekvencijski dijagram - UC02, UC03, UC13, UC14	23
3.5	Sekvencijski dijagram - UC01, UC04, UC08, UC11, UC15	24
3.6	Sekvencijski dijagram - UC05, UC06, UC11, UC15	25
3.7	Sekvencijski dijagram - UC22, UC20, UC16, UC17, UC18, UC19	26
4.1	Primjer višeslojne arhitekture korištenjem radnog okvira Spring	30
4.2	Relacijski dijagram baze podataka	34
4.3	Dijagram razreda - Controllers dio	35
4.4	Dijagram razreda - Controllers dio	36
4.5	Dijagram stanja - Prijavljeni korisnik	37
4.6	Dijagram aktivnosti - Dodavanje prijave	38
4.7	Dijagram komponenti	39
5.1	Pregled gradskog ureda određenih atributa metodom getGradskiUred()	42
5.2	Potvrđivanje novonastalog gradskog ureda metodom potvrdiGradskiUred()	43
5.3	Pregled postojanosti nepostojećeg korisnika	44
5.4	Pregled liste svih postojećih korisnika sa svojim atributima u sustavu metodom getKorisnikAll()	45
5.5	Registracija novog korisnika	46
5.6	Pretraga svih prijava registriranih u bazi podataka	46
5.7	Kreacija novih prijava u sustavu	47

5.8	Dijagram razmještaja	49
5.9	Environment varijable za frontend dio	50
5.10	Environment varijable za backend dio	51
5.11	Environment varijable za bazu podataka	52

Dodatak: Prikaz aktivnosti grupe

Dnevnik sastajanja

Kontinuirano osvježavanje

U ovom dijelu potrebno je redovito osvježavati dnevnik sastajanja prema predlošku.

1. sastanak

- Datum: 23. listopada 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Ivan Šimunić, Dominik Zoričić
- Teme sastanka:
 - raspodjela uloga po članovima
 - odabir izvedbenih tehnologija

2. sastanak

- Datum: 26. listopada 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Dominik Zoričić
- Teme sastanka:
 - definiranje osnovnih zahtjeva

3. sastanak

- Datum: 30. listopada 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Dominik Zoričić, Ivan Šimunić
- Teme sastanka:
 - Kreacija baze podataka

4. sastanak

- Datum: 2. studenoga 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Dominik Zoričić, Ivan Šimunić
- Teme sastanka:

- Demo verzija front-enda
- Ostali zahtjevi sustava

5. sastanak

- Datum: 11. studenoga 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Dominik Zoričić, Ivan Šimunić
- Teme sastanka:
 - Komentiranje dijagrama obrazaca uporabe
 - Uspoređivanje obrazaca uporabe sa funkcijama back-end servisa

6. sastanak

- Datum: 16. studenoga 2023.
- Prisustvovali: Nikola Botić, Nino Ćurko, Ivan Elez, Davor Najev, Josip Šare, Dominik Zoričić, Ivan Šimunić
- Teme sastanka:
 - Autentifikacija korisnika
 - Plan za deployment

Tablica aktivnosti

Kontinuirano osvježavanje

Napomena: Doprinosi u aktivnostima treba navesti u satima po članovima grupe po aktivnosti.

	Nino Ćurko	Davor Najev	Nikola Botić	Dominik Zoričić	Josip Šare	Ivan Elez	
Upravljanje projektom	10	11	12	9	10	8	
Opis projektnog zadatka	2						
Funkcionalni zahtjevi	2.5						
Opis pojedinih obrazaca	4						
Dijagram obrazaca	2						
Sekvencijski dijagrami	2				2		
Opis ostalih zahtjeva	1						
Arhitektura i dizajn sustava	2						
Baza podataka	1.5						
Dijagram razreda	2				2		
Dijagram stanja	1						
Dijagram aktivnosti	1						
Dijagram komponenti							
Korištene tehnologije i alati	1						
Ispitivanje programskog rješenja							
Dijagram razmještaja							
Upute za puštanje u pogon							

Nastavljeno na idućoj stranici

Nastavljeno od prethodne stranice

	Nino Ćurko	Davor Najev	Nikola Botić	Dominik Zoričić	Josip Šare	Ivan Elez	
Dnevnik sastajanja	1						
Zaključak i budući rad							
Popis literature							
<i>Postavljanje klijentske strane (front-end)</i>		20	15				
<i>Postavljanje serverske strane (front-end)</i>				15	13	11	
<i>izrada baze podataka</i>	1	1	1	2	1	1	
<i>spajanje s bazom podataka</i>		4		4	4		
<i>Autorizacija i autentikacija</i>		3		5	5		
<i>Punjenje baze podataka</i>				1	1	2	
<i>Implementacija Google maps API-ja</i>		4			1		
<i>Deployment</i>		6		6	6		

Dijagrami pregleda promjena

dio 2. revizije

Prenijeti dijagram pregleda promjena nad datotekama projekta. Potrebno je na kraju projekta generirane grafove s gitlaba prenijeti u ovo poglavlje dokumentacije. Dijagrami za vlastiti projekt se mogu preuzeti s gitlab.com stranice, u izborniku Repository, pritiskom na stavku Contributors.